

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ
ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

МЕТОДИЧЕСКИЕ УКАЗАНИЯ
по выполнению лабораторных работ
по дисциплине «Базы данных»
для студентов направления подготовки
09.03.04 «Программная инженерия»

Ставрополь
2026

Лабораторная работа №1 Проектирование ER и реляционной модели данных

Введение

В настоящее время реляционные базы уверенно являются неотъемлемой частью большинства информационных систем. В этой области прикладного программирования стремительно развиваются новые технологии, платформы реализации и среды разработки приложений. Тем не менее, остаются классическая часть и общепризнанный подход к процессу проектирования, разработки и сопровождения реляционной базы данных, разбитый на последовательные этапы, которые обычно приходится реализовывать. Данная работа имеет целью усвоение и закрепление теоретических знаний в области проектирования реляционных баз данных и изучения особенностей конкретных систем управления базами данных. Проектирование их включает следующие этапы:

1) концептуальное проектирование, содержащее системный анализ рассматриваемой предметной области поставленной задачи и составление ее словесного описания. Этот этап в данной работе отсутствует, то есть каждый студент получает словесное описание объектов системной области и связей, существующих между ними, в качестве исходного задания. На данном этапе необходимо представить концептуальную модель в виде ER-диаграммы;

2) логическое проектирование базы данных. На этом этапе осуществляется выбор СУБД (системы управления базами данных), с помощью которой база данных будет реализована. В лабораторной работе предполагается использование СУБД MS SQL Server. Поскольку эта СУБД реализует реляционную модель данных, то необходимо осуществить переход от ER-модели к реляционной модели и затем нормализовать полученные отношения;

3) создание таблиц, входящих в реляционную модель с учетом существующих между ними связей. В данной лабораторной работе требуется составить для каждой создаваемой таблицы оператор CREATE TABLE языка SQL.

Построение ER-модели

ER-модель (модель «сущность – связь») – широко используемый инструмент разработки баз данных, представляющий собой формализованное графическое представление предметной области в виде прямоугольников, линий и специальных символов. Преимущество ER-модели

заключается в том, что она понятна и разработчикам баз данных, и заказчикам, то есть будущим ее пользователям, и потому все возможные изменения в структуре данных согласуются между ними именно на уровне этой модели. ER-модель никак не связана ни с какой конкретной СУБД, этот выбор осуществляется впоследствии. В основе ER-модели лежат понятия «сущность» («entity») и «связь» («relation»). Не существует удовлетворительного формального описания понятия «сущность». Можно сказать, что это – совокупность значимых объектов, существующих в рамках предметной области. Например, при построении системы учета успеваемости студентов придется оперировать сущностями «студенты», «преподаватели», «дисциплины», а в случае создания банковской системы – сущностями «клиенты» и «счета». Каждая сущность включает набор атрибутов, представляющих собой именованные свойства сущности. Например, сущность «студенты» может иметь следующие атрибуты:

- 1) номер зачетной книжки;
- 2) ФИО;
- 3) код группы;

4) дату рождения.

Сущность – совокупность объектов, а каждый объект этой совокупности – экземпляр сущности. Экземпляры должны быть различимы, то есть не должно быть двух экземпляров с одинаковыми наборами атрибутов. Каждая сущность имеет набор атрибутов, уникальный для каждого экземпляра сущности, называемый ключом. Например, для сущности «студенты» ключ состоит из одного атрибута «Номер зачетной книжки», так как точно не существует двух и более студентов с одинаковым значением этого атрибута.

Между сущностями могут быть установлены связи, определяющие, как сущности соотносятся или взаимодействуют между собой. Связь может существовать между двумя сущностями (бинарная связь) или между сущностью и ею же самой (рекурсивная связь). Наличие связи означает, что сущность имеет атрибут или набор атрибутов которые составляют ключ. Например, сущность «студенты» и сущность «учебные группы» связаны между собой атрибутом «код группы», который является ключом сущности «учебные группы» и присутствует в наборе атрибутов сущности «студенты».

Связи бывают трех (четырёх) типов:

1) связь **«один к одному»**. При этой связи один элемент первой сущности связан не более чем с одним экземпляром второй сущности. Этот тип связи встречается сравнительно редко;

2) связь **«один ко многим» («многие к одному»)**. В этом случае один экземпляр первой сущности связан с несколькими экземплярами второй сущности, но один экземпляр второй сущности связан не более чем с одним экземпляром первой сущности. Первая сущность считается в данной связи основной, вторая – подчиненной. Примером может служить рассмотренная выше связь между сущностями «учебные группы» и «студенты». Сущность «учебные группы» является здесь основной, сущность «студенты» – подчиненной, так как одна группа состоит из нескольких студентов, следовательно, одному экземпляру сущности «группы» соответствует несколько экземпляров сущности «студенты», принадлежащих этой группе;

3) связь **«многие ко многим»**. В этом случае каждый экземпляр первой сущности может быть связан с несколькими экземплярами второй сущности, а каждый экземпляр второй сущности – с несколькими экземплярами первой. В качестве примера можно привести связь между сущностями «преподаватели» и «дисциплины». Эта связь имеет тип «многие ко многим», так как один преподаватель может вести несколько учебных дисциплин, а одна дисциплина – вестись несколькими преподавателями.

Важной характеристикой связи является ее **обязательность**, или **модальность**. Обязательность связи означает, что любому экземпляру сущности обязательно соответствует хотя бы один экземпляр другой сущности, состоящей с первой в обязательной связи. В противном случае связь является необязательной. Связь может иметь неодинаковую модальность с разных концов. Например, каждая учебная группа имеет куратора из числа преподавателей, однако не каждый преподаватель является куратором.

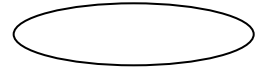
Для построения ER-модели необходимо сначала выявить сущности, характерные для заданной предметной области, и определить для каждой сущности набор атрибутов, имеющих значение с учетом поставленной задачи. Затем необходимо определить все имеющиеся связи между сущностями и характеристики этих связей. После этого строится ER-диаграмма, т. е. собственно графическое изображение модели с использованием одной из существующих нотаций. Для простых случаев ER-диаграмму можно рисовать вручную, для более сложных можно воспользоваться специализированным программным обеспечением, известным как CASE-средства.

Базовые понятия

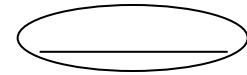
Сущность (объект)



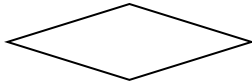
Атрибут сущности (свойство, характеризующее объект)



Ключевой атрибут (атрибут, входящий в первичный ключ)



Связь



Кардинальность показывает, какое максимальное количество экземпляров данной сущности может участвовать в конкретной связи (точнее экземпляре связи).

Кардинальность бинарных связей («один к одному») степени 1 будем обозначать следующим образом:



$(0, 1)$



$(1, 1)$

Кардинальность бинарных связей степени n «один ко многим» («многие к одному») будем обозначать так:



$(0, N)$



$(1, N)$

Сущности в ER-диаграмме изображаются в виде прямоугольников. Атрибуты в виде овалов, ключевые атрибуты подчеркиваются. Связи между сущностями изображаются в виде линий, соединяющих прямоугольники соответствующих сущностей. Для сущности, находящейся со стороны «многие», линия связи должна заканчиваться значком из трех расходящихся линий. Необязательность связи обозначается значком, имеющим на конце небольшую окружность.

В качестве примера создадим несколько упрощенную модель системы учета успеваемости студентов. Оценки, получаемые студентами по изучаемым дисциплинам, заносятся в специальную таблицу, где указываются номер зачетной книжки студента, код дисциплины, табельный номер преподавателя, дата получения оценки и собственно оценка. Студент не может в один и тот же день два раза сдавать одну и ту же дисциплину. Студенты разбиты по группам. Каждая группа имеет свой код и куратора из числа преподавателей. Каждый преподаватель может вести несколько дисциплин, каждая учебная дисциплина может вестись несколькими преподавателями.

Таким образом, можно выделить следующие сущности:

- 1) студенты;
- 2) преподаватели;
- 3) учебные дисциплины;
- 4) группы;
- 5) оценки.

Атрибутами сущности «студенты» будут:

- 1) номер зачетной книжки;
- 2) ФИО;
- 3) номер группы.

Ключом здесь будет номер зачетной книжки.

Сущность «**преподаватели**» будет иметь атрибуты:

- 1) табельный номер;
- 2) ФИО.

Ключом сущности, естественно, является табельный номер преподавателя.

У сущности «**учебные дисциплины**» будет два атрибута:

- 1) код дисциплины; - ключевой
- 2) наименование дисциплины.

Сущность «**группы**» включает следующие атрибуты:

- 1) код группы;
 - 2) название группы
 - 3) табельный номер куратора группы.
- Здесь код группы – ключевой атрибут.

Сущность «**оценки**» имеет следующие атрибуты:

- 1) номер зачетной книжки студента;
- 2) код дисциплины;
- 3) таб. номер преподавателя;
- 4) дата получения оценки;
- 5) оценка.

Ключ – уникальный идентификатор сущности. В данном случае ключом сущности «оценки» будет набор из 4 атрибутов: номер зачетной книжки, код дисциплины, таб.номер преподавателя и дата получения оценки, так как предполагается, что студент в один день по некоторой дисциплине может получить только одну оценку.

Теперь рассмотрим связи, имеющиеся между сущностями.

1. Так как в одной группе числятся несколько студентов, между сущностями «группы» и «студенты» существует связь «один ко многим». Эта связь обязательна с обеих сторон.
2. Так как у каждой группы есть куратор из числа преподавателей, между сущностями «группы» и «преподаватели» существует связь «один к одному», необязательная со стороны «преподаватели», так как не каждый преподаватель является куратором какой-либо группы.
3. Поскольку каждый преподаватель может вести несколько дисциплин, а одна дисциплина – вестись несколькими преподавателями, то между сущностями «преподаватели» и «дисциплины» имеется связь «многие ко многим». Поскольку любая дисциплина преподается хотя бы одним преподавателем, а любой преподаватель преподает хотя бы одну дисциплину, то связь обязательна с обеих сторон.
4. Между сущностью «студенты» и сущностью «оценки» существует связь «один ко многим». Каждый студент неоднократно получает оценки по изучаемым дисциплинам. Возможно, что студент не успел получить еще ни одной оценки, следовательно, связь является необязательной со стороны «студенты».
5. Между сущностью «преподаватели» и сущностью «оценки» существует связь «один ко многим». Каждый преподаватель выставляет оценки по своим дисциплинам. Существует вероятность, что преподаватель не поставил ни одной оценки, потому что только что был принят на работу, следовательно, связь является необязательной со стороны «преподаватели».
6. Между сущностью «дисциплины» и сущностью «оценки» существует связь «один ко многим», так как по каждой дисциплине может быть выставлено много оценок. Существует вероятность, что по некоторой дисциплине не было еще выставлено ни одной оценки, следовательно, связь является необязательной со стороны «дисциплины».

На рис. 1 показана ER-диаграмма, построенная для данной задачи с учетом существующих связей.

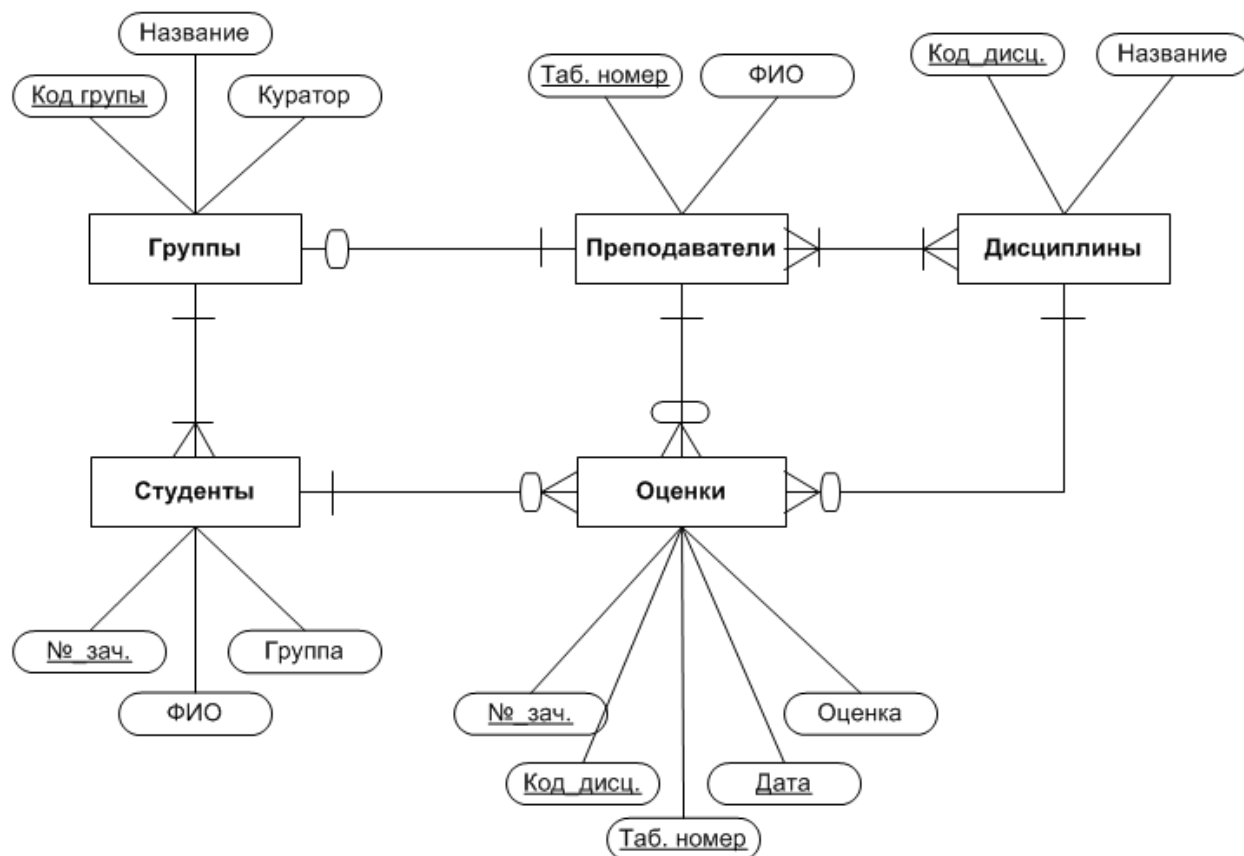


Рис. 1

Переход к реляционной модели

На следующем этапе проектирования необходимо преобразовать ER-модель в реляционную. Основной структурой реляционной модели является отношение (relation), графической интерпретацией которого служит таблица. Каждое отношение состоит из некоторого ограниченного числа кортежей, а содержание каждого кортежа определяется набором атрибутов отношения. Каждый атрибут имеет определенный тип, значение которого берется из определенного домена. Кортежам отношения соответствуют строки таблицы, количество столбцов таблицы равно количеству атрибутов отношения, а тип величины, находящейся в соответствующем столбце, определяется типом соответствующего атрибута.

Отношения могут быть связаны между собой посредством набора атрибутов, одинаково содержащихся в обоих отношениях. Связи между отношениями в реляционной модели (в отличие от ER-модели) могут иметь только тип «один ко многим» («многие к одному»), т. е. одно отношение всегда будет основным, а второе – подчиненным, и одному кортежу основного отношения могут соответствовать несколько кортежей подчиненного отношения. Данное соответствие означает, что у обоих кортежей значения набора атрибутов, по которому связаны отношения, совпадают. Обычно у основного отношения данный набор атрибутов является первичным ключом и, следовательно, уникальным для каждого кортежа. У второго отношения данный набор атрибутов называется внешним ключом для данной связи. Кортежей с одинаковым значением внешнего ключа может быть сколько угодно, но не может существовать кортежей со значением внешнего ключа, которому не соответствовал бы какой-либо первичный ключ основного отношения. Связь «один к одному» рассматривается как частный случай связи «один ко многим». Связи

«многие ко многим» в реляционной модели быть не может. Они заменяются на отношения «один ко многим» или «многие к одному».

Чтобы преобразовать ER-модель в реляционную модель, необходимо выполнить следующие действия:

1. Каждой сущности ER-модели ставится в соответствие отношение реляционной модели, при этом каждому атрибуту сущности ставится в соответствие атрибут отношения реляционной модели. Ключ сущности становится первичным ключом соответствующего отношения

(PRIMARY KEY). Имена сущностей и отношений, равно как и атрибутов, могут не совпадать. Желательно при указании имен отношений и атрибутов реляционной модели использовать латиницу, поскольку эти имена чаще всего являются идентификаторами в некотором языке программирования.

2. В каждое отношение, соответствующее подчиненной сущности, добавляется набор атрибутов, соответствующий ключу основной сущности, если, конечно, он там не присутствовал. В любом случае этот набор атрибутов становится внешним ключом в подчиненном отношении

(FOREIGN KEY).

3. При обязательном характере связи у атрибутов, соответствующих внешнему ключу, устанавливается свойство отсутствия неопределенных значений (NOT NULL).

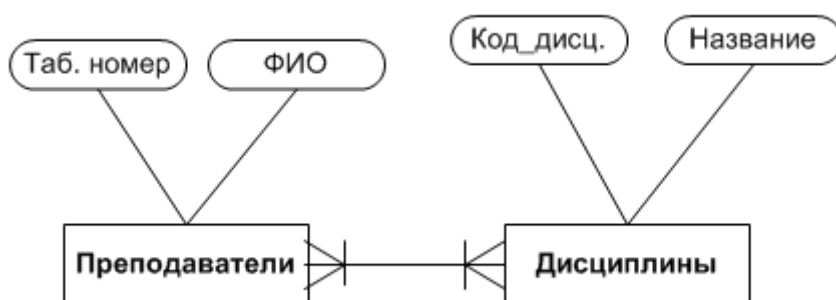
4. Если в ER-модели имеются связи «многие ко многим», то их надо преобразовать в связи «один ко многим», поскольку связи «многие ко многим» в реляционной модели не допускаются. Для этого в реляционную модель добавляется связующее отношение, атрибуты которого соответствуют атрибутам первичных ключей обоих отношений, участвующих в связи «многие ко многим». Связующее отношение будет находиться в связи «один ко многим» с каждым из этих отношений. В рассматриваемом примере связь «один ко многим» имеют сущности «преподаватели» и «дисциплины». В реляционной модели вводится связующее отношение, атрибутами которого будут «id_subject» и «tab_num». Первый атрибут соответствует первичному ключу сущности «дисциплины», а второй – первичному ключу сущности «преподаватели». Это отношение будет иметь связь «один ко многим» с отношениями, соответствующими сущностям «преподаватели» и «студенты» (рис. 2).

В принципе, уже на уровне ER-модели можно провести ее нормализацию. В данной работе требуется, чтобы модель удовлетворяла условиям третьей нормальной формы.

Напомним, что отношение находится в первой нормальной форме, если значения всех его атрибутов атомарны, т. е. представляют собой простые, а не составные типы данных. В нашем примере это условие выполняется для всех сущностей. Значениями атрибутов всех сущностей являются только числа, строки или даты. Массивов, структур или подтаблиц среди них нет.

Сущность удовлетворяет условиям второй нормальной формы, если каждый неключевой атрибут зависит от всего первичного ключа, но не зависит от какой-либо части первичного ключа. Из определения следует, что условие второй нормальной формы может быть нарушено только у составного первичного ключа.

ER-модель



Реляционная модель

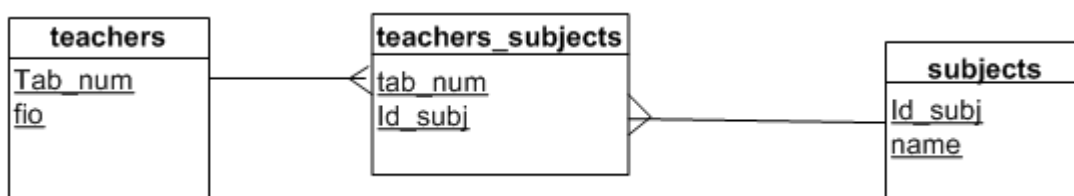


Рис. 2

В приведенном примере составной первичный ключ имеется у сущности «оценки». Неключевым атрибутом здесь является только один атрибут «оценка». Его значение определяется всей совокупностью ключевых атрибутов, поэтому сущность «оценки» условиям второй нормальной формы удовлетворяет.

Сущность находится в третьей нормальной форме, если она удовлетворяет условиям второй нормальной формы и каждый неключевой атрибут зависит только от первичного ключа, но не зависит от какого-либо неключевого атрибута. Все сущности нашего примера удовлетворяют условиям третьей нормальной формы.

Полученную реляционную модель изобразим в графической форме. Каждому отношению будет соответствовать прямоугольник, в который будут вписаны имена атрибутов отношения и их типы. Набор возможных типов определяется официальным стандартом языка SQL, однако у конкретных СУБД обычно имеются расхождения со стандартом. В нашей модели атрибуты могут принимать текстовые, числовые значения или представляют собой какую-либо дату. Рассмотрим эти типы подробнее:

1. Тип даты в стандарте SQL и в каждой СУБД определяется по-своему.
2. Для представления строчных атрибутов используется стандартный тип Character(n) (сокращенно CHAR(n)), поддерживаемый любой СУБД. Здесь n – максимальная длина атрибута в символах. Длину каждого строчного атрибута необходимо согласовывать с заказчиком, но в данной работе ее можно выбирать произвольно, по усмотрению разработчика.

3. Для числовых атрибутов можно использовать тип integer, если атрибут может принимать только целочисленные значения, или тип float, если атрибут может иметь дробную часть.

На рис. 3 приведена реляционная модель для нашего примера. В ней 6 отношений, причем 5 из них соответствуют 5 сущностям ER-модели, а 6 отношение teachers_subjects появилось в результате преобразования связи «многие ко многим» между сущностями «преподаватели» и «дисциплины» к двум связям «один ко многим» между отношениями subjects и teachers_subjects, а также teachers и teachers_subjects.

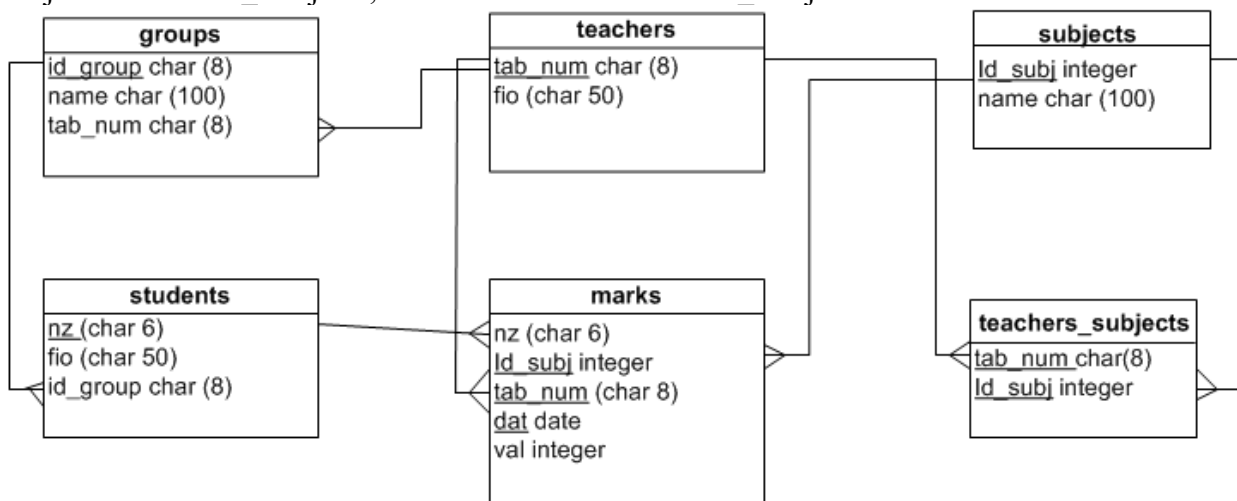


Рис.3

На рис. 3 линии, соответствующие связям «один ко многим», четко соединяют именно те атрибуты двух отношений, которые участвуют в этой связи. Отношение, участвующее в связи со стороны «многие», является подчиненным, а это означает, что атрибут подчиненного отношения может брать значения только из списка значений соответствующего ему по связи атрибута главного отношения.

1. Связь между отношениями teachers и groups является связью один к одному, поскольку только один из преподавателей может быть куратором группы. Однако эта связь рассматривается как частный случай связи «один ко многим», причем отношение groups является подчиненным, так

как значение атрибута curator (табельный номер преподавателя) берется из табельных номеров отношения teachers. Таким образом, атрибут curator подчиненного отношения groups является внешним ключом для первичного ключа tab_num главного отношения teachers.

Создание таблиц базы данных

Та часть языка SQL, которая служит для создания объектов базы данных, называется DDL (Data Definition Language). Основными объектами любой базы данных являются таблицы, соответствующие отношениям реляционной модели. Для создания таблицы употребляется оператор CREATE TABLE. Синтаксис команды имеет следующий вид:
CREATE TABLE <имя таблицы> (<имя столбца> <тип данных столбца>[<ограничения столбца>] [,(<определение столбца> <тип данных>[<ограничения столбца>])][<определения ограничений таблицы>])

Имена столбцов и их типы для каждой таблицы приведены на рис. 3.

На столбцы в данной работе могут быть наложены следующие ограничения:

1) ограничение NOT NULL, означающее, что значение столбца должно быть обязательно задано при вводе новой записи. По умолчанию обычно действует ограничение NULL, по которому задавать значение для данного столбца не обязательно;

- 2) ограничение UNIQUE, означающее, что в данной таблице не должно быть двух и более записей с одинаковым значением этого столбца;
- 3) ограничение первичного ключа PRIMARY KEY. Это ограничение для столбца возможно в том случае, если первичный ключ таблицы состоит из одного данного столбца. В противном случае столбцы, входящие в первичный ключ, указываются в конце всего оператора CREATE TABLE в ограничениях для всей таблицы. То, что столбец является первичным ключом, означает также и то, что его значение должно быть обязательно задано (ограничение NOT NULL), и это значение должно быть уникальным для данной таблицы (ограничение UNIQUE);
- 4) ограничение внешнего ключа REFERENCES <ИМЯ ГЛАВНОЙ ТАБЛИЦЫ>. Оно накладывается на столбец в том случае, если он является внешним ключом в связи «один ко многим» с другой, главной в данной связи, таблицей. В этом случае любое значение данного столбца должно соответствовать некоторому значению первичного ключа главной таблицы.

Последние два ограничения называются ограничениями ссылочной целостности базы данных и служат для обеспечения корректности данных, содержащихся в таблицах.

Ограничения всей таблицы в конце оператора CREATE TABLE предназначены для определения ограничений, накладываемых на несколько столбцов сразу. Например, это происходит при наличии составных первичного или внешнего ключа.

Ниже приведены операторы CREATE TABLE для создания каждой таблицы рассматриваемого примера:

- 1) таблицы subjects:

```
Create table subjects(id_subj integer primary key, name char(100));
```

- 2) таблицы teachers:

```
Create table teachers(tab_num char(8) primary key, fio char(50) not null);
```

- 3) таблицы teachers_subjects, связывающей таблицы teachers и subjects:

```
Create table teachers_subjects (id_subj integer references subjects, tab_num char(6) references teachers, constraint pk primary key(id_subj, tab_num)).
```

Здесь столбцы id_subj и tab_num заданы как внешние ключи для главных таблиц subjects и teachers соответственно, а совокупность столбцов id_subj и tab_num объявлена как первичный ключ таблицы, причем ограничение первичного ключа имеет имя pk;

- 4) таблицы groups:

```
Create table groups(id_group char(8) primary key, name char(100), tab_num char(8) references teachers).
```

Здесь первый столбец таблицы является первичным ключом, второй – название группы, третий столбец является внешним ключом, ссылающимся на таблицу teachers;

- 5) таблицы students:

```
Create table students(nz char(6) primary key, fio char(50) not null, id_group char(8) references groups).
```

При создании таблицы для столбца fio (фамилия) указано ограничение not null. Это означает, что при вводе новой записи значение для этого столбца должно быть указано обязательно.

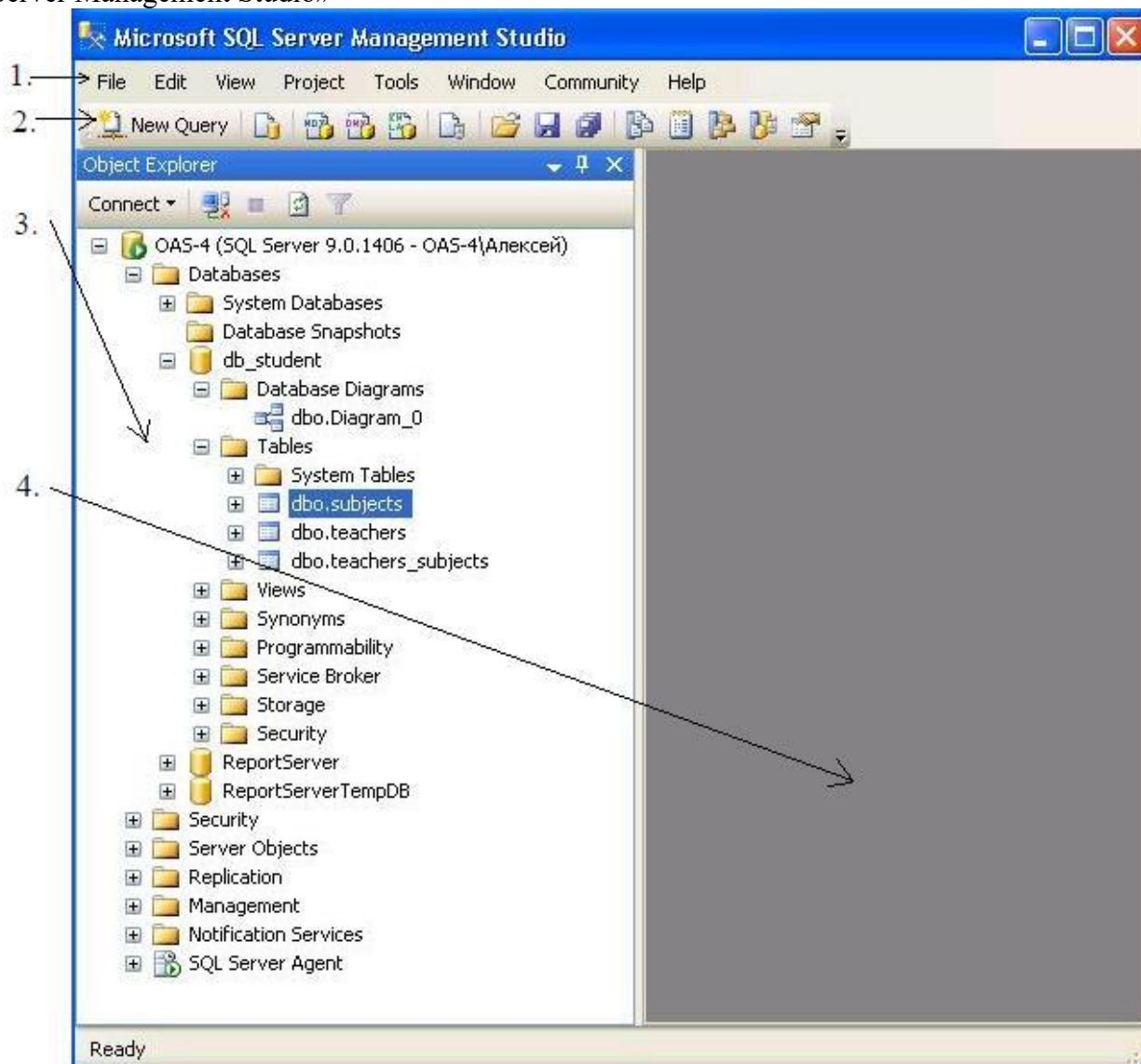
- 6) таблицы marks:

```
Create table marks(nz char(6) references students, id_subj integer references subjects, tab_num char(8) references teachers, dat date, val integer not null, constraint pmk primary key (nz, id_subj, tab_num, dat)).
```

Здесь первые три столбца являются внешними ключами, ссылающимися на первичные ключи родительских таблиц students, subjects и teachers, а первые четыре столбца в совокупности образуют составной первичный ключ, имеющий имя pmk.

SQL Server Management Studio является одним из основных программных средств администрирования СУБД SQL Server.

Запустим **SQL Server Management Studio**, проверим включение сервера. Для запуска MS SQL Server 2005, 2008 выберем утилиту **SQL Server Management Studio** и запустим ее. После нажатия кнопки «Connect» появится окно среду разработки «SQL Server Management Studio»



Данное окно имеет следующую структуру:

1. Оконное меню – содержит полный набор команд для управления сервером и выполнения различных операций.
2. Панель инструментов – содержит кнопки для выполнения наиболее часто производимых операций. Внешний вид данной панели зависит от выполняемой операции.
3. Панель «Object Explorer» - обозреватель объектов. Обозреватель объектов – это панель с древовидной структурой, отображающая все объекты сервера, а также позволяющая производить различные операции, как с самим сервером, так и с БД. Обозреватель объектов является основным инструментом для разработки БД.
4. Рабочая область. В рабочей области производятся все действия с БД, а также отображается её содержимое.

В IDE (интегрированная среда разработки) используются два типа окон: компонентов и документов.

Окна компонентов представляют такие элементы как Object Explorer, Solution Explorer, Properties и другие. Эти окна перечислены в меню View и применяются для отображения компонентов IDE, не являющиеся документами. У большинства окон можно менять месторасположение.

Окна документов позволяют просматривать и редактировать сценарии, XML-документы и любые другие текстовые документы. Они всегда размещены в основной области интерфейса, в верхней части окна документов видны вкладки, при помощи которых можно получить доступ к необходимому документу.

Базу данных в **SQL Server Management Studio** можно создать разными способами. Например, с помощью команды:

```
CREATE DATABASE STUDENT;
```

Для написания кода в **SQL Server Management Studio** нужно нажать кнопку «Создать запрос» («New query») на панели инструментов «Стандартная» («Standart»). Для выполнения команды нажать F5 или кнопку с красным восклицательным знаком EXECUTE.

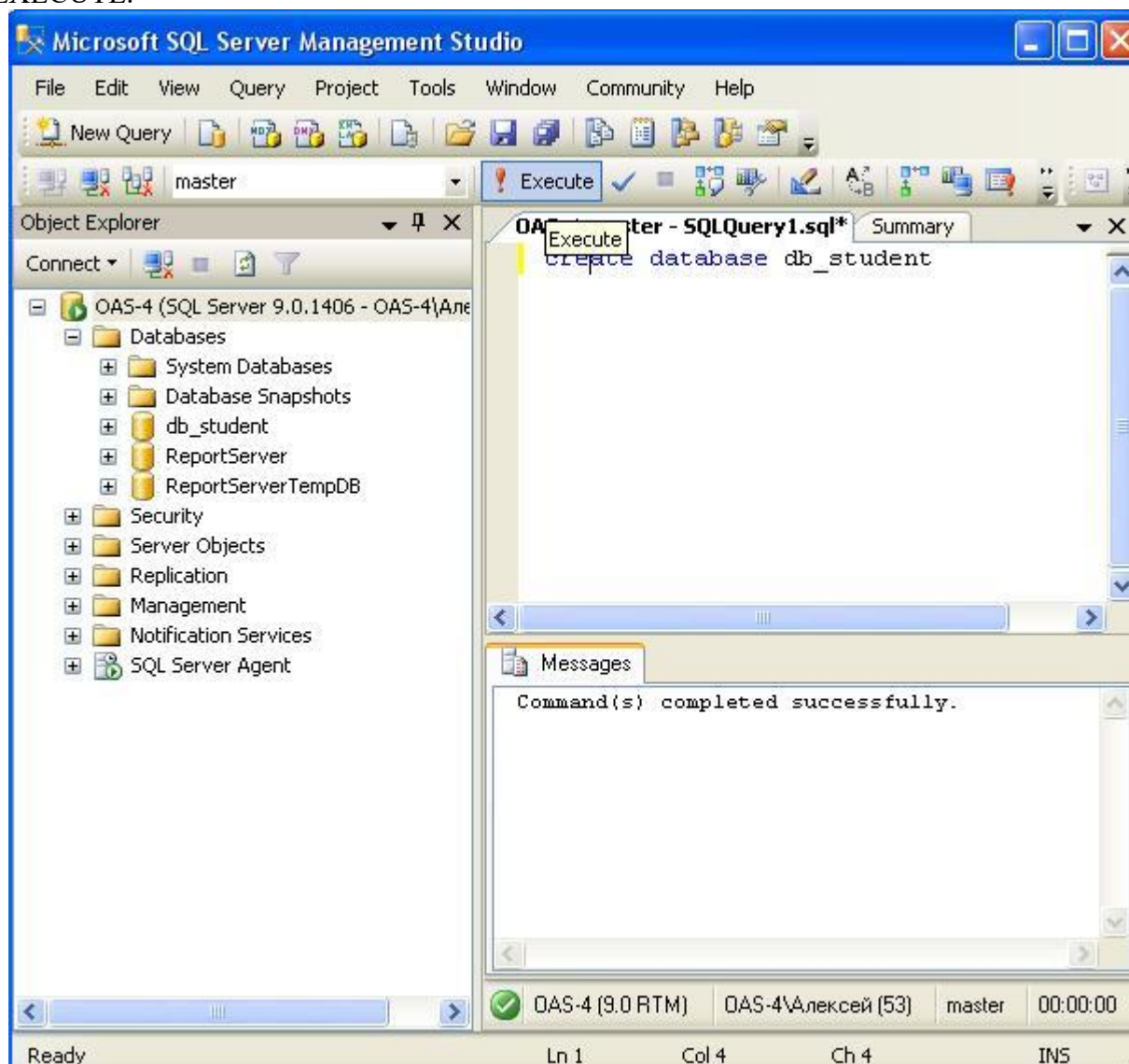
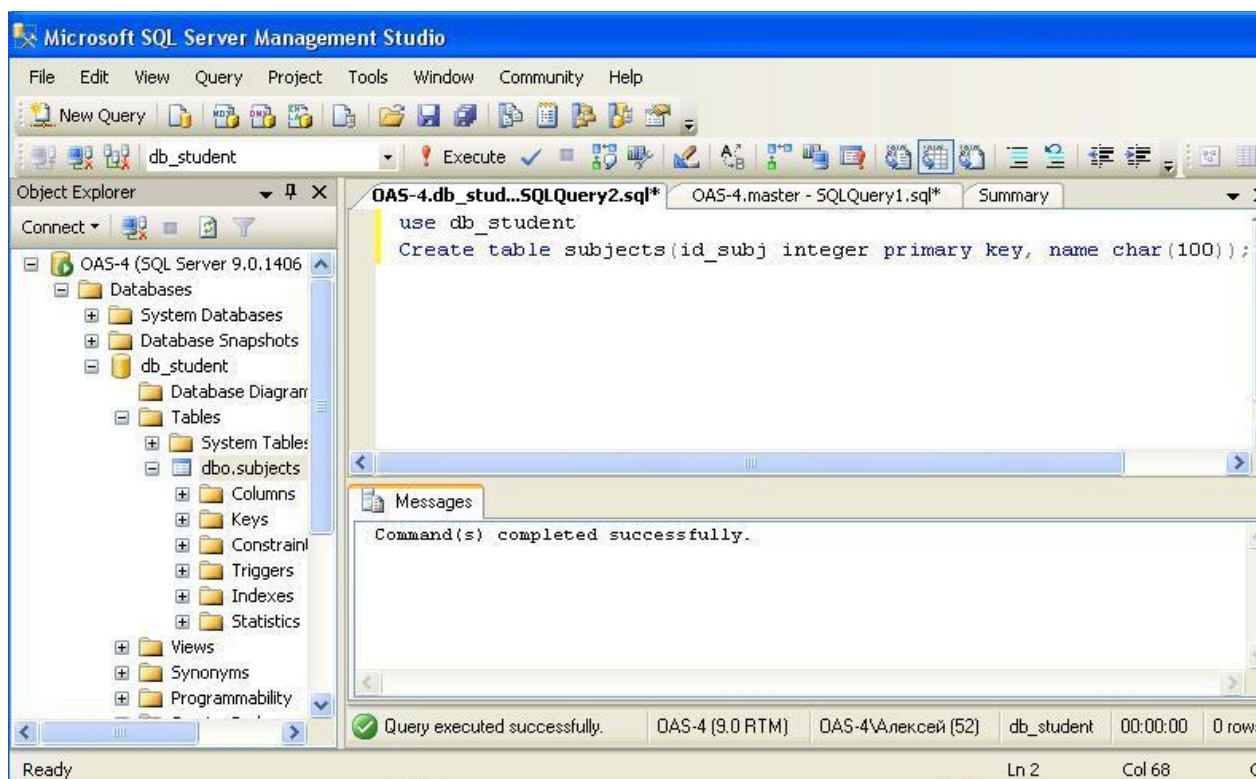


Рис.1

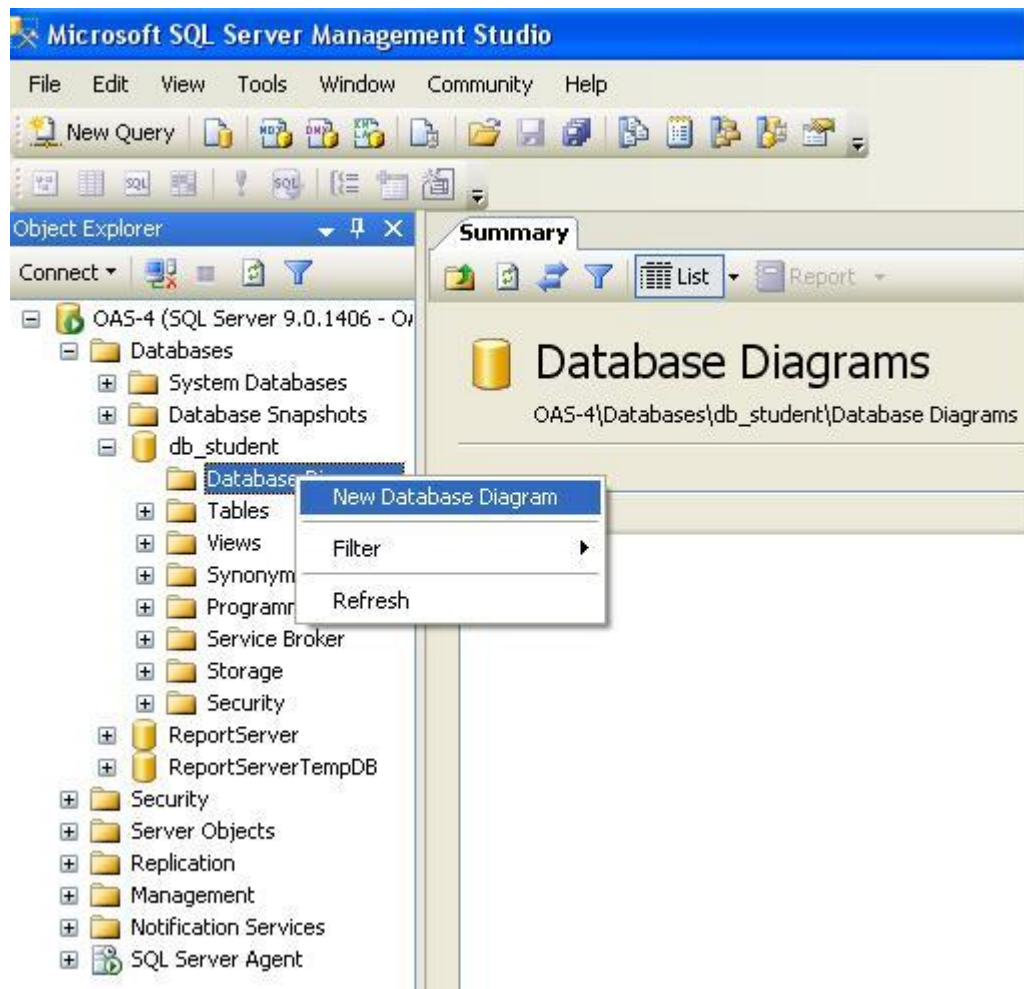
Также с помощью T-SQL создать аналогичным образом таблицы в базе данных. Создадим новый запрос и напишем:

```
use db_student  
Create table subjects(id_subj integer primary key, name char(100));
```

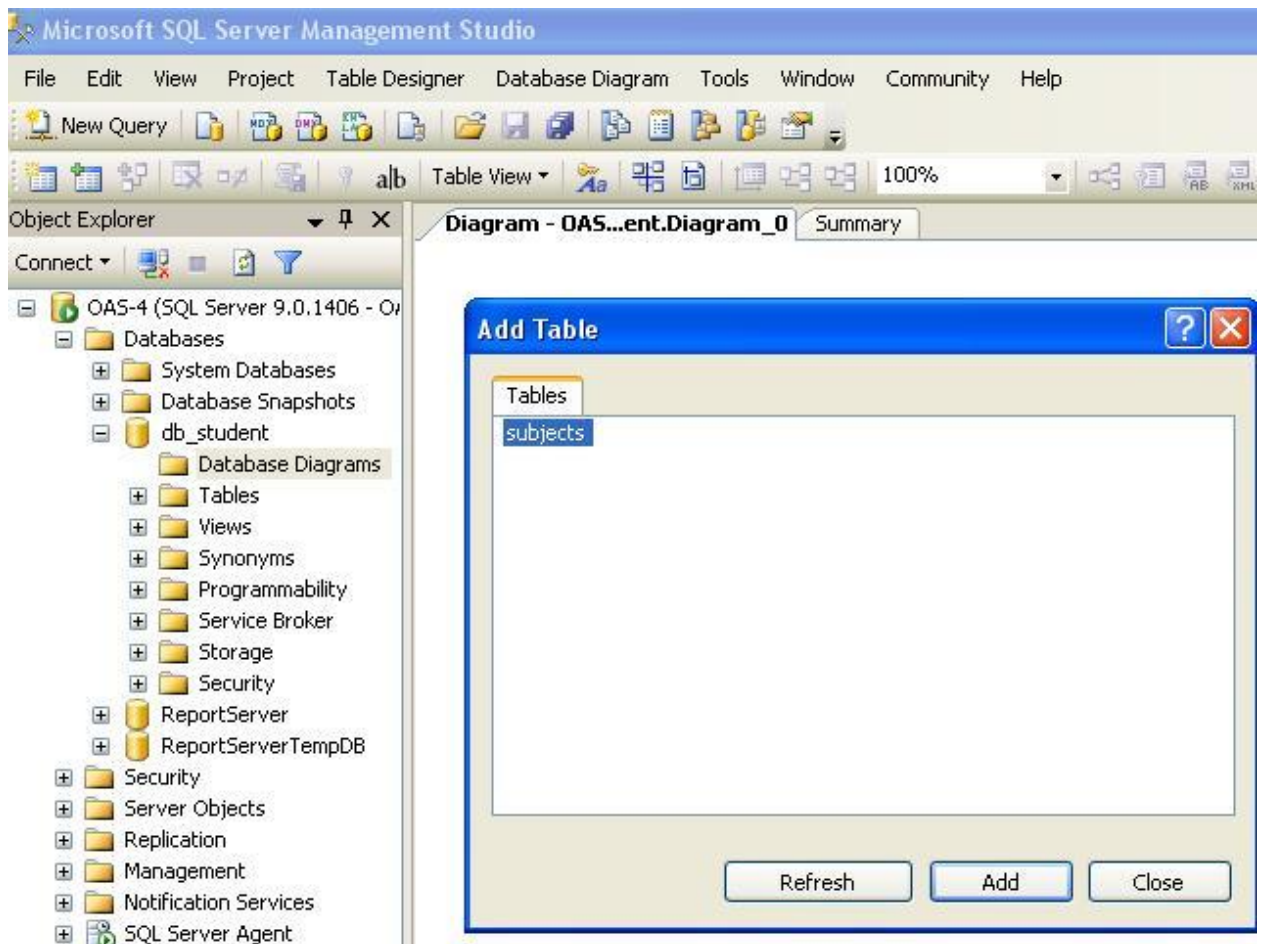


Создается новая таблица subjects, которую можно найти в Databases->db_student->Tables. Аналогичным образом создаются другие таблицы. По необходимости делаем Refresh на папке Tables, для отображения вновь созданных таблиц.

Схема данных строиться в Databases->db_student->Database Diagramme. Для этого создаем новую диаграмму.



И добавляем в нее таблицу subjects.



Создадим ещё две таблицы:

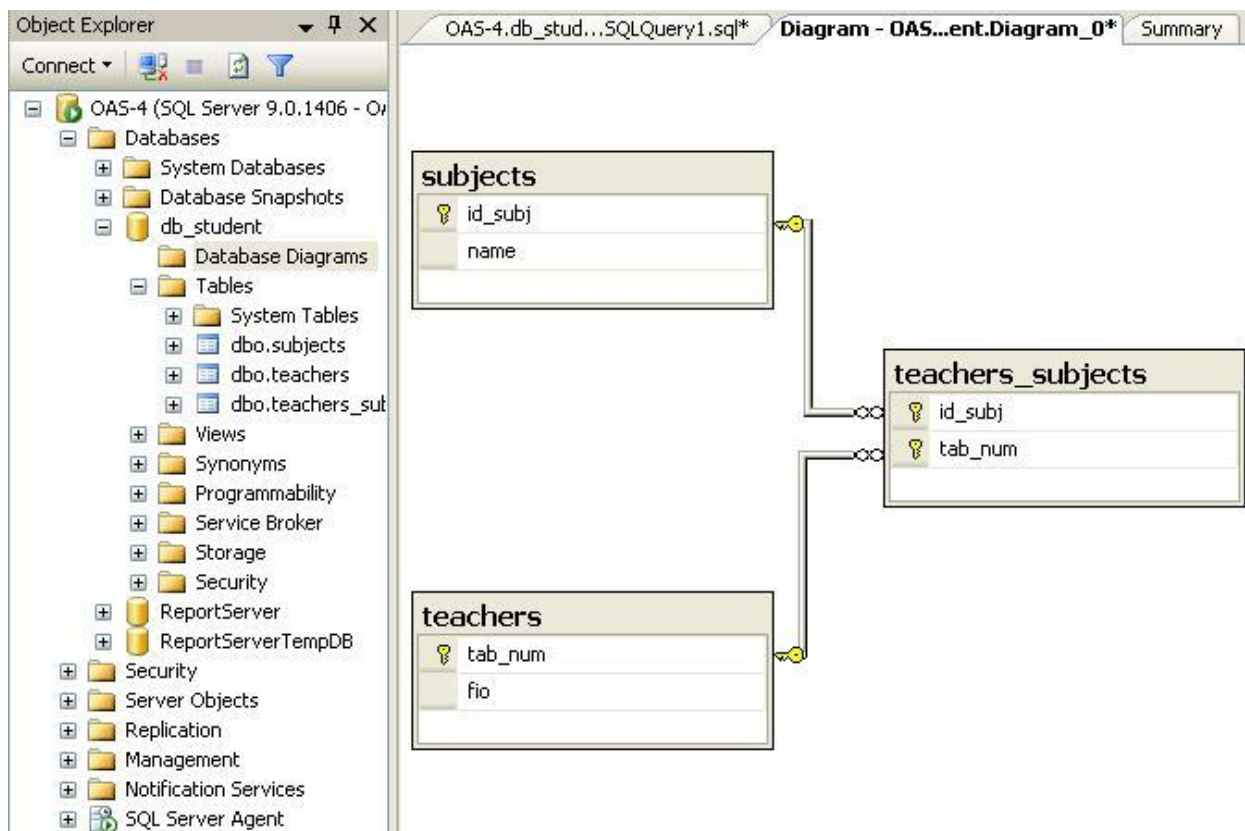
```
use db_student
```

```
Create table teachers(tab_num char(8) primary key, fio char(50) not null);
```

```
use db_student
```

```
Create table teachers_ subjects (id_subj integer references subjects, tab_num char(8) references teachers, constraint pk primary key(id_subj, tab_num)).
```

И добавим их в диаграмму. В результате получится:



При этом связи создадутся автоматически.
Аналогичным образом создаются другие таблицы и связи. В вашей лабораторной работе необходимо создать все таблицы и связи.

Примеры описания предметной области

1)

Агентство занимается продажей авиабилетов на различные рейсы, ведет учет проданных билетов и учет пассажиров, купивших билеты, поэтому возникает потребность в хранении и обработке данных, сгруппированных следующим образом:

- 1) информация о расписании рейсов (номер рейса, тип самолета, пункт отправления, пункт назначения, дата вылета, время вылета, время полета, цена билета);
- 2) информация о свободных местах на рейс (номер рейса, дата вылета, общее количество мест, количество свободных мест);
- 3) информация о пассажирах, купивших билеты на рейсы (номер паспорта, фамилия, имя, отчество, номер рейса, дата вылета);
- 4) архив, в который помещается информация о выполненном рейсе (номер рейса, дата вылета, общее количество мест, количество проданных мест).

2)

Издательство – предприятие, занимающееся выпуском разнообразной печатной продукции. Издательство заключает договор с заказчиком (клиентом) на выполнение заказа. Заказчиком может выступать частное лицо или организация. Частное лицо может быть автором издания (или одним из авторов, если их несколько) или представителем автора.

Организация для контактов с издательством также имеет своего представителя – контактную персону. Заказ может быть книгой, брошюрой, рекламным проспектом, буклетом, бюллетенем для голосования или каким-либо другим видом издательской продукции. Подготовленные издательством материалы заказчика печатаются в типографиях, где издательство размещает свои заказы.

Информацию о работе издательства можно сгруппировать следующим образом:

- сведения о заказчиках (частное лицо или организация, личные данные контактной персоны, адрес, телефон, факс);
- сведения о заказах (номер заказа, заказчик, вид печатной продукции, издание, типография, дата приема заказа, отметка о выполнении, дата выполнения заказа);
- сведения об изданиях (код издания, автор и название, объем в печатных листах, тираж, номер заказа);
- сведения об авторах (личные данные: Ф.И.О., домашний адрес, телефон)
- сведения о типографиях (название, адрес, телефон).

В отчете лабораторной работы должны быть отражены следующие этапы:

1. Описание **предметной области** (см. «Примеры описания Предметной области»);
2. Представление ER-модели (можно от руки нарисовать)
3. Представление реляционной модели (можно от руки нарисовать)
4. Описание запросов по созданию таблиц на языке SQL
5. Представление схемы

Контрольные вопросы:

1. Основные элементы ER модели.
2. Виды связей.
3. Отличия между ER и реляционной моделью.
4. Обязательность связи.
5. Определение трёх нормальных форм.
6. Операторы CREATE TABLE

Лабораторная работа № 2. Проектирование баз данных с использованием пакета ERwin

Целью работы является освоение процесса информационного моделирования для заданной предметной области с помощью инструментальной среды ERwin.

Постановка задачи

Пользуясь пакетом ERwin, по варианту задания сформировать модель данных для реализации БД с указанными требованиями. Структура связей между таблицами определяется правилами: один-ко-многим или многие-ко-многим.

При выполнении работы обеспечить соблюдение следующих требований:

- корректность модели данных;
 - полнота информации, представляемой в модели данных.
- При необходимости студент самостоятельно дополняет модель требуемыми атрибутами и сущностями для адекватного представления рассматриваемой задачи;
- без избыточность информации, представляемой в модели данных. После завершения моделирования разработанная структура данных транслируется средствами пакета ERwin в базу данных MS Access.

Порядок выполнения.

1. Формулировка задания

Задание соответствует заданию лабораторной работы №3. Предметная область: Библиотека (учет движения книг).

Основные предметно-значимые сущности: Студенты, Книги. Основные предметно-значимые атрибуты сущностей:

- студент – номер читательского билета, фамилия, код факультета, телефон факультета;
- книги–библиотечный шифр книги, название книги, фамилия первого автора, дата выхода, город издательства, название издательства, количество страниц, количество экземпляров хранения, наличие;

- информация о факультете – код, телефон факультета; Основные требования к функциям системы:

- должен вестись учет выдачи/возврата книг (дата выдачи книги, дата возврата книги);

- должен вестись учет бронирования книг (дата заказа).Основные запросы:

- выбрать книги, которые находятся у читателей и неопределенного читателя;

- выбрать читателей, которые брали ту или иную книгу суказанием даты выдачи книги и даты сдачи книги читателем;

- выбрать книги, пользующиеся наибольшим спросом.

2. Используемые результаты

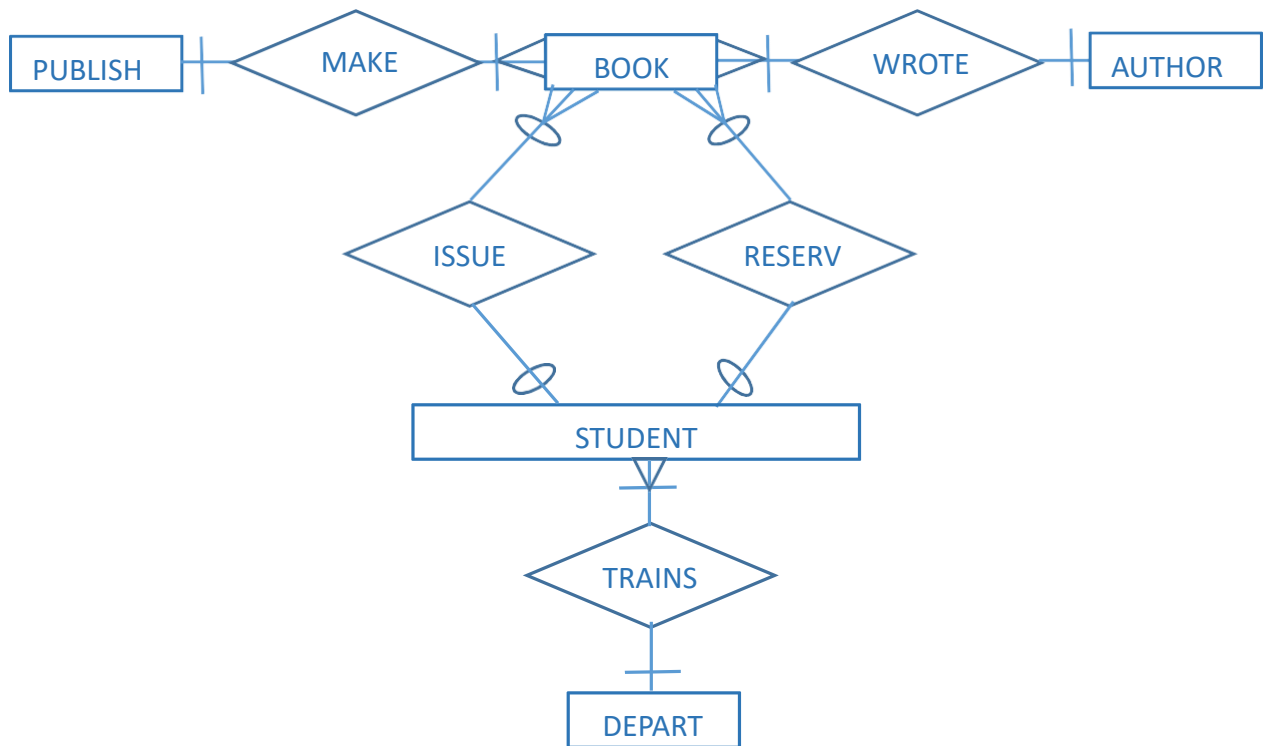
Отношения (сущности):

STUDENT(SNum, SName, FCode)

BOOK(BCode, title, date, volume, copies, available)¹ISSUE(ICode, SNum, BCode, issue_date, return_date)RESERV(RCode, SNum, BCode, order_date)AUTHOR(A_ID, AName)

PUBLISH(P_ID, town, PName)DEPART (FCode, FPhone) .

ER-диаграмма:



3. Подготовительная работа

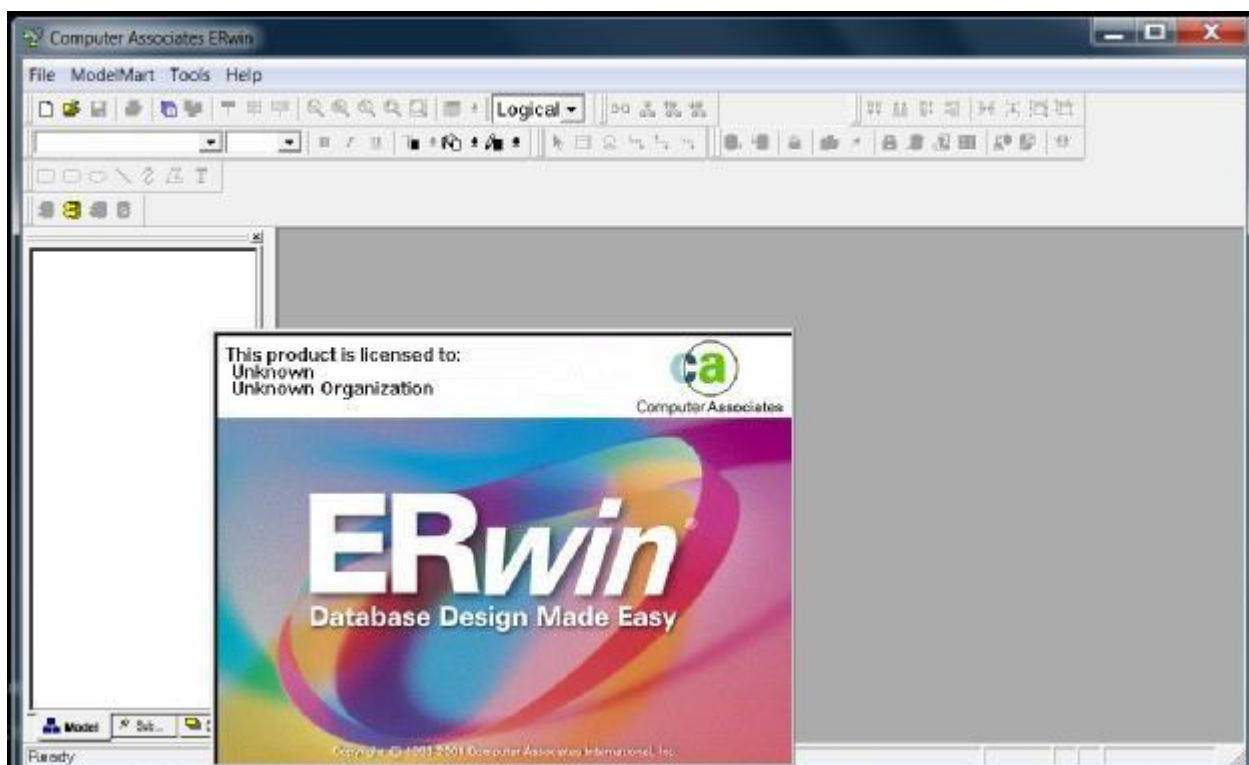
- скачайте со страницы УММ в папку с именем «Чулюков» приложение ERWin (файл ERwin 4.0.rar);
- разархивируйте приложение в эту же папку; при этом создается папка «ERwin 4.0»;

-
- создайте в папке «Чулюков» папку со своей фамилией, например, «Иванов».

4. Запуск ERwin

Пример выполнения:

- в папке «ERwin 4.0» откройте файл @start.bat; На экране появится :



- после того как закроется окно рекламы, выполните команду
File

-> New;

В окне <Computer Associates ERwin> появится окно <Create Model – Select Template> что по-русски «Создать модель – Выбрать шаблон» (рис.4.1).

Сформируем в среде пакета ERWin модель в соответствии с полученными ранее результатами. Для этого запускаем ERWin, и создаем новую модель. В качестве типа модели указываем «Logical/Physical», сервер – «Access2000», нажимаем <OK>.

На экране останется основное окно (рис. 4.2).

Основное окно программы содержит следующие части:

- 1) Область модели данных или рабочая область
- 2) Панели инструментов
- 3) Браузер элементов модели

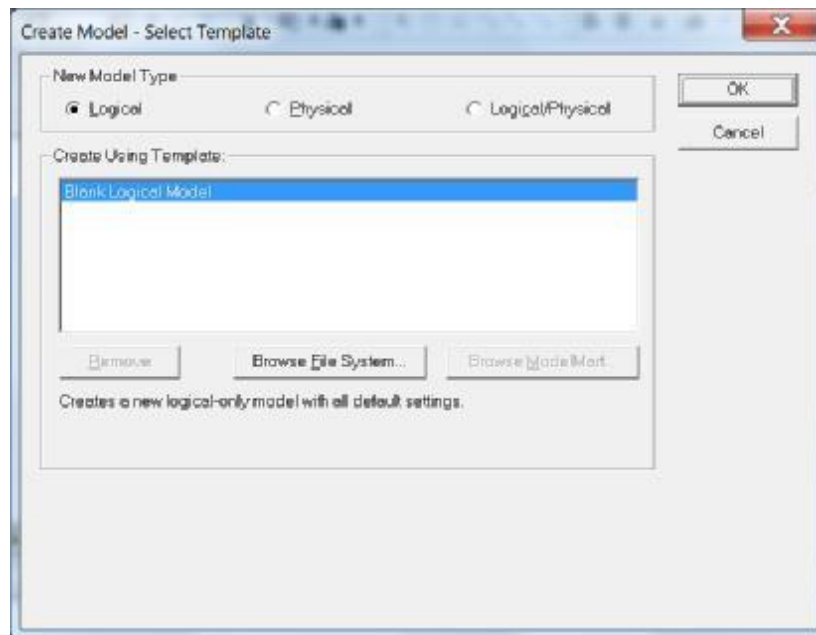


Рис. 4.1 – Диалоговое окно *Create Model – Select Template*

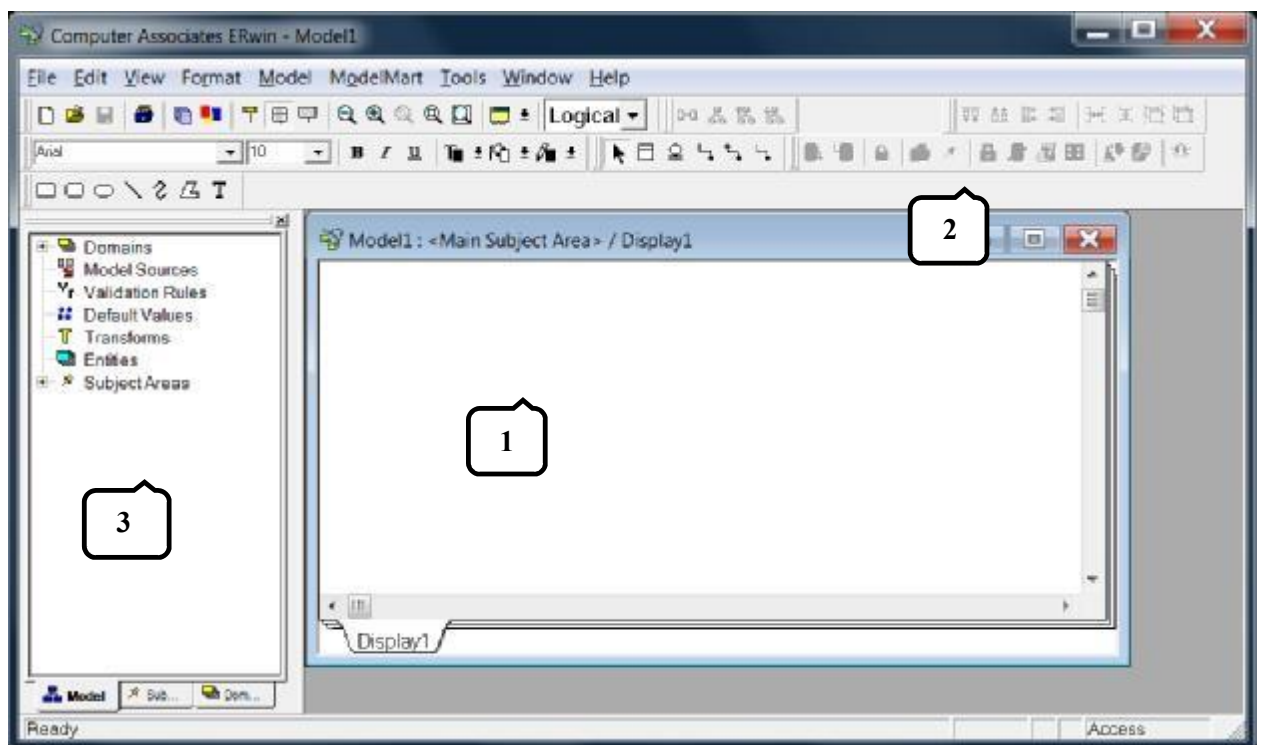


Рис. 4.2 – Основное окно *ERwin*

При создании новой модели пользуются панелью инструментов <Toolbox> (рис. 4.3).



а



б

Рис. 4.3 – Панели *Toolbox* для логического (а) и физического (б) уровней модели в нотации *IDEFIX*

<Toolbox> – инструментальные кнопки создания элементов модели, сущностей и связей. Набор кнопок незначительно различается для логической и физической модели. Содержит кнопки: стрелка – выбор объекта, создание независимой сущности, создание связки- подкатегории, создание сущности-представления, создание идентифицирующей и неидентифицирующей связи «один-к-одному», создание связи «многие-ко-многим», создание связи для сущности-представления.

5. Работа с сущностями

Чтобы добавить в модель новую сущность, необходимо на панели <Toolbox> выбрать инструмент <Create entity> , после чего щелкнуть мышью на свободном месте рабочей области. Появится рамка новой сущности, и программа перейдет в режим ввода имени сущности. Завершается ввод имени нажатием на клавишу Enter.

После ввода имени сущности в зависимости от режима отображения возможны три варианта:

- 1) в режиме отображения имен сущностей программа просто отобразит вновь добавленную сущность;
- 2) в режиме отображения атрибутов предложит вводить имена для ключевых атрибутов (этот режим обычно включен по умолчанию);
- 3) в режиме ввода определений перейдет к вводу определения для вновь добавленной сущности.

В режиме ввода атрибутов после ввода каждого имени атрибута нажатие клавиши Enter приводит к добавлению нового атрибута. Сначала вводятся имена ключевых атрибутов. После ввода всех ключевых атрибутов можно нажать клавишу Tab и перейти к вводу неключевых атрибутов.

Добавим сущность BOOK. Для этого:

- выберем инструмент <Create entity> и щелкнем мышью на свободном месте рабочей области;

Появится рамка новой сущности с именем по умолчанию E/1 (рис. 4.4)

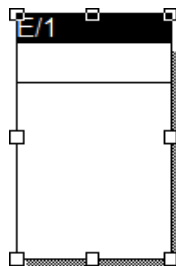


Рис. 4.4 – Созданная рамка сущности

- введите имя сущности BOOK;
- нажмите Enter;

Будет предложено начать вводить ключевые атрибуты (рис. 4.5):

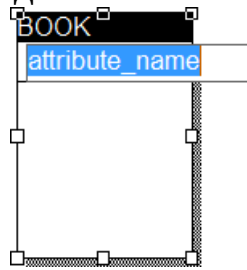


Рис. 4.5 – Поле для ввода первого ключевого атрибута

- введите имя ключа BCode;
- так как ключевой атрибут один, то нажмите Tab для перехода к вводу неключевых атрибутов;
- введите атрибут title и нажмите Enter;
- также введите остальные атрибуты; Получим следующий результат (рис. 4.6):

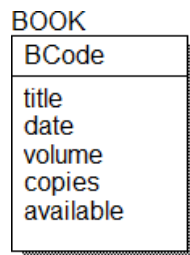


Рис. 4.6 – Сущность BOOK с ключом BCode и неключевыми атрибутами

- аналогично сформируйте все остальные сущности модели (рис. 4.7);

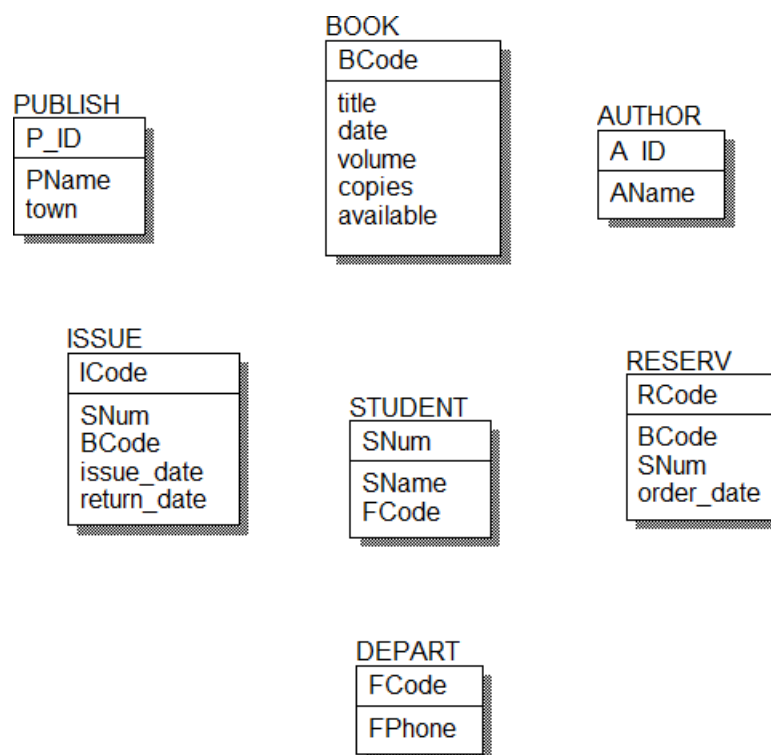


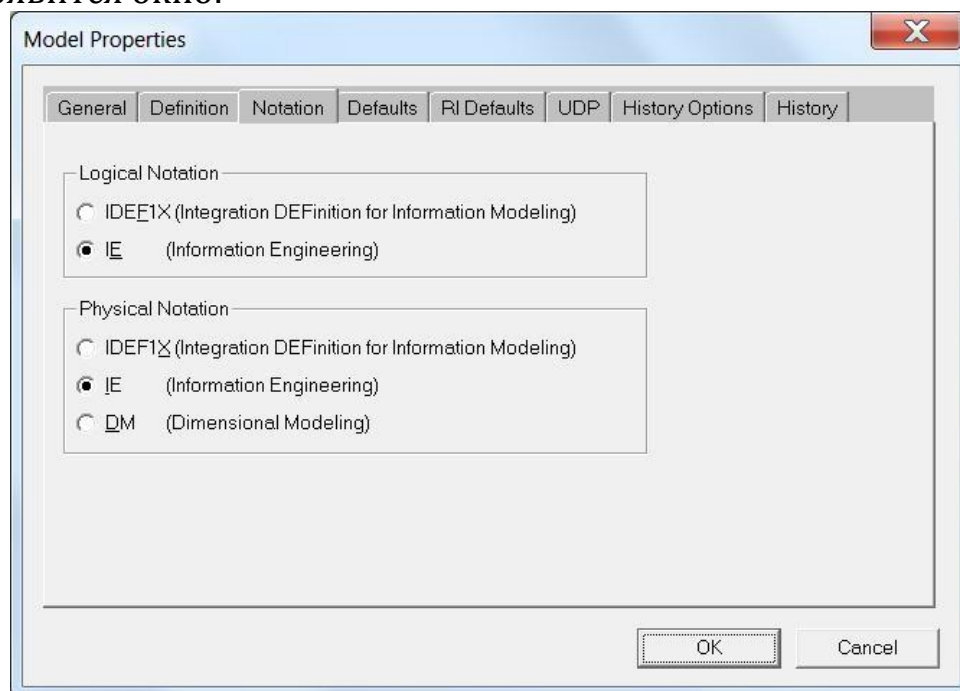
Рис. 4.7 – Все сущности модели

6. Работа со связями

Для того, чтобы получить модель со связями вида как на ER-диаграмме надо установить нотации IE (Information Engineering). Для этого:

- выполнить команду Model - > Model Properties...

Появится окно:



Во вкладке Notation установить нотацию IE для логической и

физической моделей.

После этого панели Toolbox примут следующий вид (рис. 4.8).



Рис. 4.8. Панели Toolbox для логического (а) и физического (б) уровней модели в нотации IE

Для установления связей между сущностями будем пользоваться панелью <ToolBox> (рис. 4.8). На логическом уровне (рис. 4.8.а) правой кнопкой с пунктирной линией.

[1]. Установим связь между сущностями PUBLISH и BOOK. Так как считается, что каждая конкретная книга издана одним издательством, а каждое издательство издает множество книг, то эта связь будет один-ко-многим.

Для создания новой связи следует:

- установить курсор на нужной кнопке в палитре инструментов (правая кнопка на рис.4.8.а) и нажать левую кнопку мыши;
- щелкнуть сначала по односвязной, а затем по n-связной сущности.

В результате получим связь:

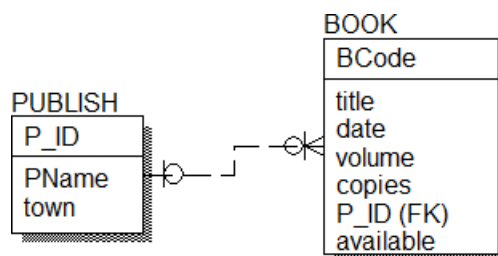


Рис. 4.9 – Связь установлена

Заметьте, что ключ P_ID сущности PUBLISH добавился в качестве внешнего ключа (FK, Foreign Key) в сущность BOOK.

Теперь необходимо установить класс принадлежности сущностей в соответствии с разработанной ранее схемой. Он обязательный для обеих сущностей. Для редактирования свойств связи следует:

- “кликнуть” правой кнопкой мыши по связи и выбрать в выпавшем меню пункт <Relationship Properties> (рис. 4.10);
- в закладке <General> в области <Cardinality> выбрать <One or More

(P) (один или много), что соответствует случаю обязательного класса принадлежности сущностей;

- щелкнуть <OK>.

Relationships

Relationship: PUBLISH R/28 BOOK

New... Delete

General Definition Rolename RI Actions UDP

Verb Phrase

Parent-to-Child: R/28

Child-to-Parent:

Relationship Cardinality

Summary: Zero-or-One-to-One-or-More (P)

Cardinality

☐ Zero, One or More

☒ One or More (P)

☐ Zero or One (Z)

☐ Exactly:

Relationship Type

☐ Identifying

☒ Non-Identifying

Nulls

☒ Nulls Allowed

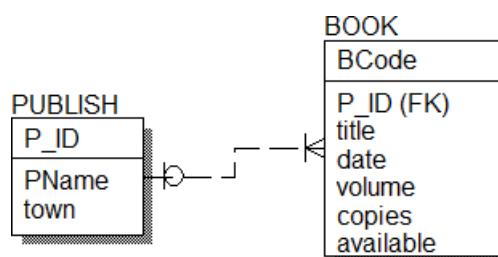
☐ No Nulls

☐ Logical Only

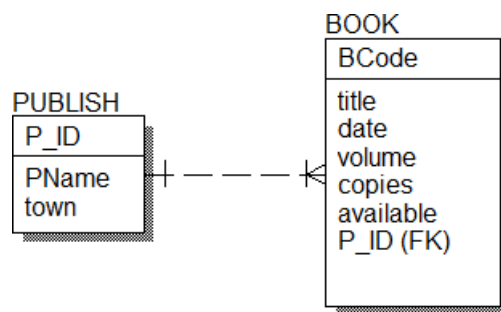
OK Cancel

Рис. 4.10 – Окно редактирования связей. Установка обязательного классапринадлежности

В результате получим следующую связь:



Стилизованный ноль на связи у сущности PUBLISH говорит о том, что P_ID может принимать пустые значения (NULL values). В нашем случае пустые значения не допускаются, при всех операциях с базой данных в этом поле обязательно должно быть какое-то значение. Значение «NULL» не является нулем, пустой строкой и т.д. Оно говорит о том, что в данном поле вообще отсутствует какое-либо значение. Установить свойство связи NoNulls можно также в окне редактирования связей. В итоге для этой связи получим:



[2]. Аналогичным образом установим связи между всеми остальными сущностями в соответствии с проектом.

В результате получим логическую модель проекта (рис. 4.11).

[3]. Сравните эту схему с ER-диаграммой, полученной при «ручном» проектировании.

7. Редактирование физической модели

После того, как общий вид связей определен, следует переходить к редактированию физической модели. Для этого в панели инструментов выбирается из выпадающего списка вид модели <Physical>.

Модель приобретает вид, как на рисунке 4.12.

На физическом уровне модель выглядит так, как она должна быть реализована средствами СУБД. В п.3 в качестве сервера базы данных была выбрана СУБД Access.

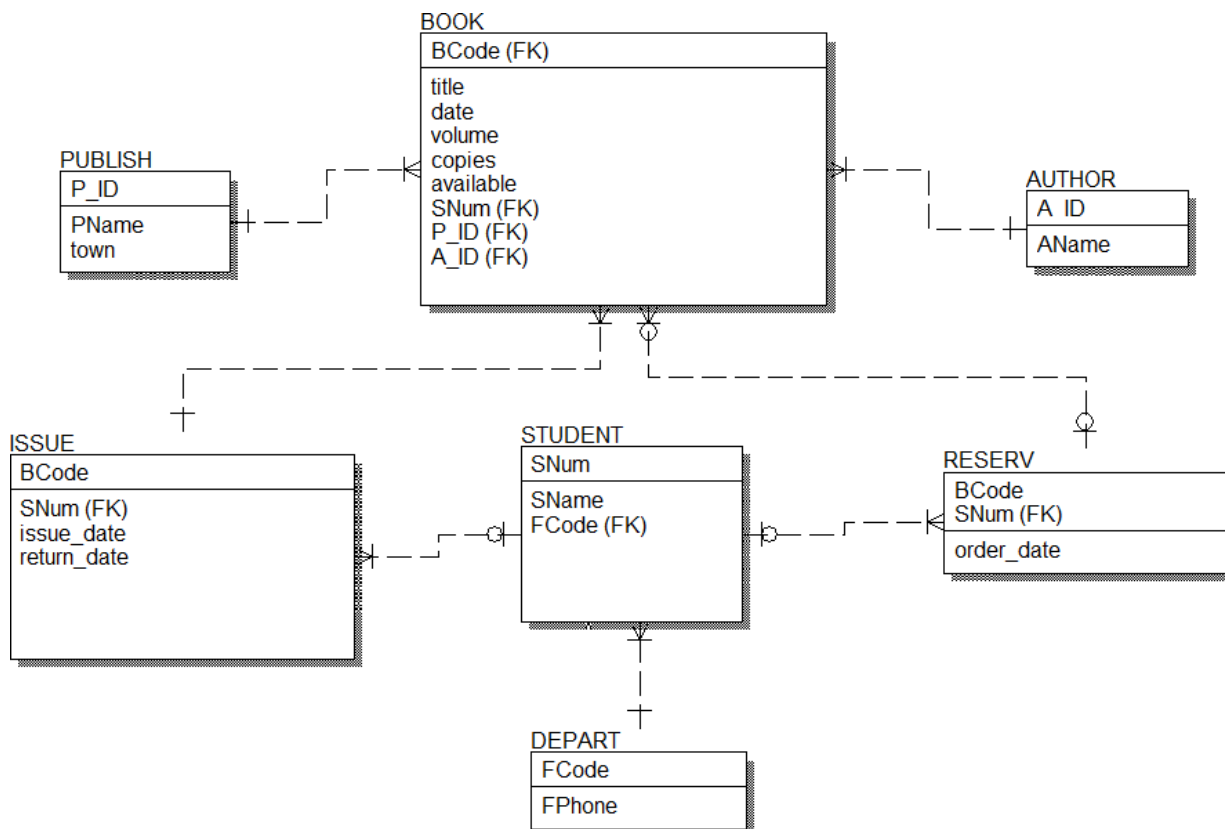


Рис. 4.11 – Логическая модель проекта

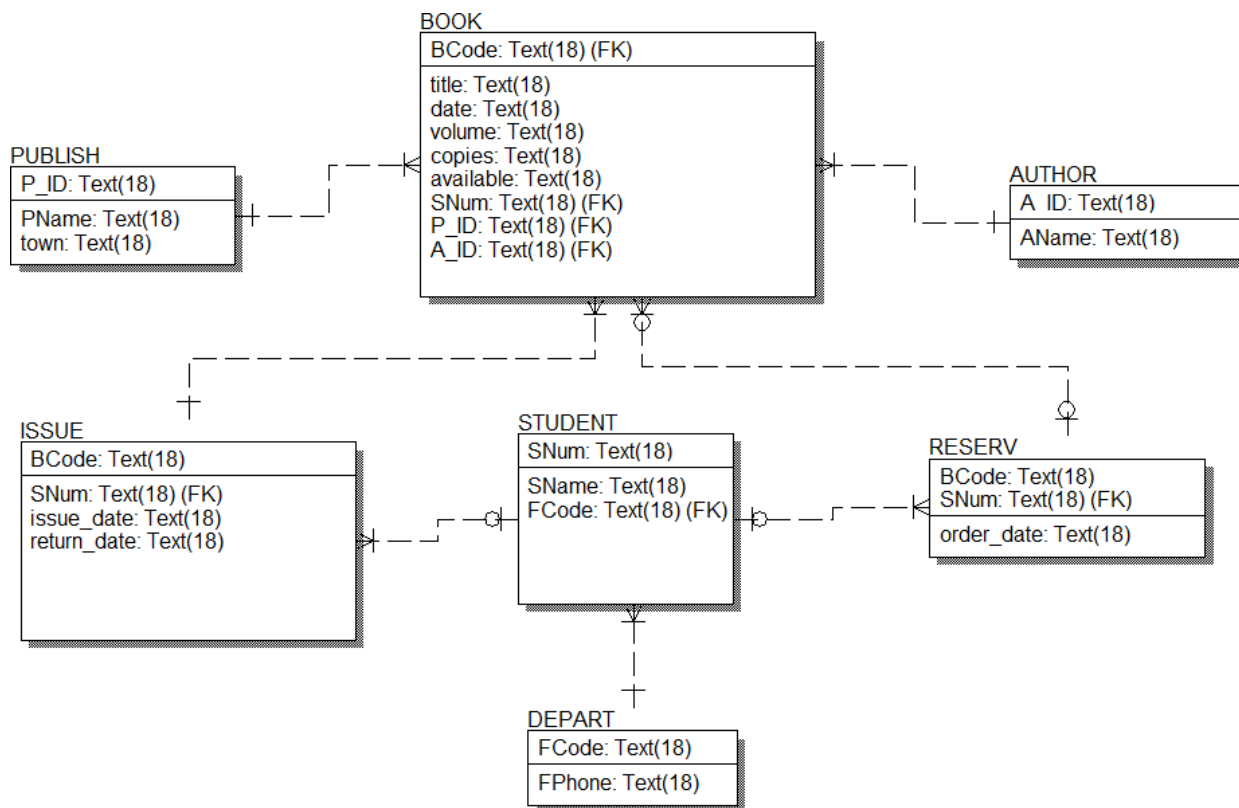


Рис. 4.12 – Физическая модель проекта по умолчанию

Для переименования атрибутов (при необходимости) надо

сделать двойной клик на сущности, и выбирая атрибут из списка, кнопкой

<Rename> вызывать диалог переименования атрибута.

Для работы с типами данных атрибутов используется вкладка <Access> в правой части окна. Следует обратить внимание, что предлагаемые типы данных зависят от типа данных, установленного для атрибута на вкладке <General>. Если при создании модели или в меню

<Database-Choose Database> был указан другой тип сервера, то вкладка с типами данных атрибутов (обычно она идет сразу после вкладки

<General>) будет иметь имя выбранного типа базы данных. Например, при выборе типа базы SQL Server, эта вкладка будет называться <SQLServer>.

Итак, следующий шаг – проверка типов атрибутов. Следует внимательно проверить все типы данных, используемых для атрибутов, а также допустимость пустых значений. Для облегчения восприятия рекомендуется включить отображение дополнительной информации об атрибутах. Для этого надо щелкнуть правой кнопкой мыши на свободном месте модели (где нет атрибутов и связей), в появившемся меню выбрать пункт <Table Display>, и далее проставить галочки напротив пунктов <Column Datatype>, <Null Option>, <Primary Key Designator>, <ForeignKeyDesignator> (рис. 4.13).

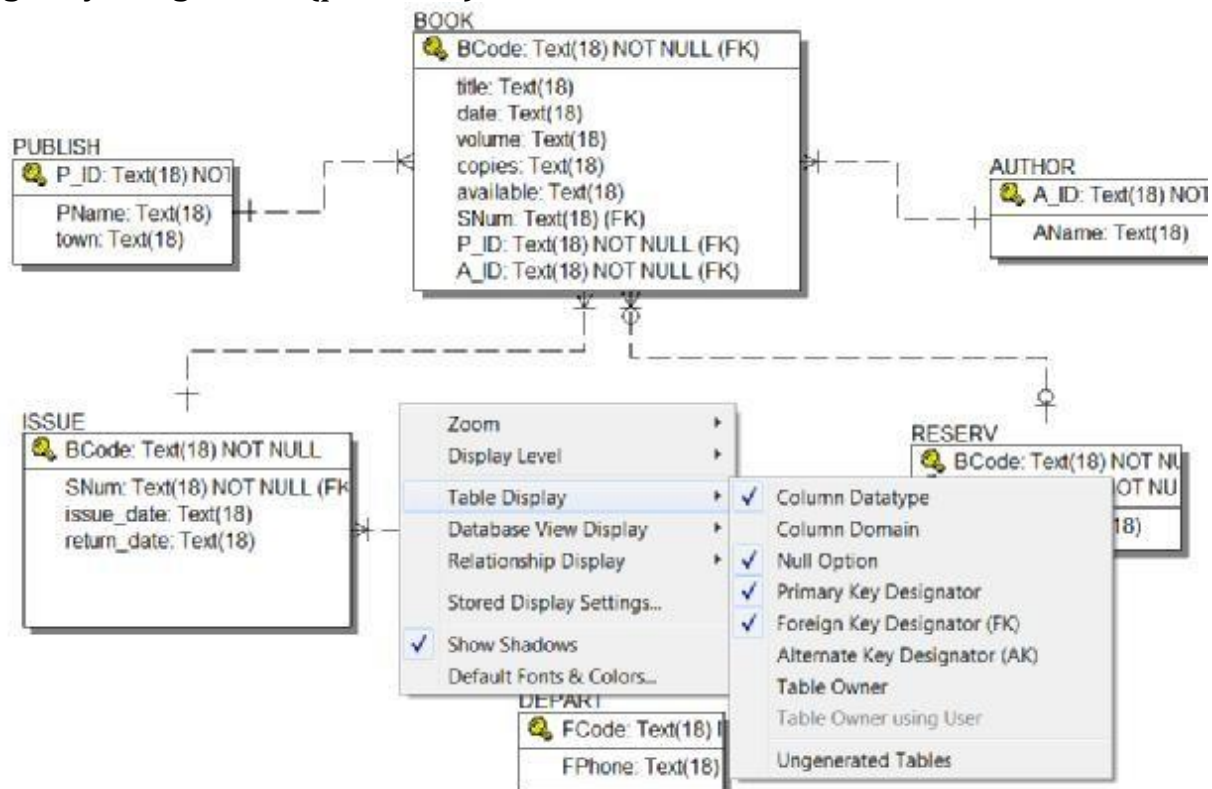


Рис. 4.13 – Установка вида отображения информации об атрибутах

[1]. Установите типы данных для всех атрибутов в соответствии с таблицами 4.1 – 4.7. Типы внешних ключей (FK) в дочерних сущностях установятся автоматически после их установки в родительских сущностях.

Таблица 4.1. Структура таблицы PUBLISH

Column	Access DataType	Average Width	Not Null
P_ID	AutoNumber		√
PName	Text	18	
town	Text	18	

Таблица 4.2. Структура таблицы BOOK

Column	Access DataType	Average Width	Not Null
BCode	AutoNumber		√
title	Text	18	
date	Date/Time		
volume	Integer		
copies	Byte		
avaiable	Yes/No		

Таблица 4.3. Структура таблицы AUTHOR

Column	Access DataType	Average Width	Not Null
A_ID	AutoNumber		√
AName	Text	18	

Таблица 4.4. Структура таблицы STUDENT

Column	Access DataType	Average Width	Not Null
SNum	AutoNumber		√
SName	Text	18	

Таблица 4.5. Структура таблицы DEPART

Column	Access DataType	Average Width	Not Null
FCode	AutoNumber		√
FPhone	Byte		

Таблица 4.6. Структура таблицы ISSUE

Column	Access DataType	Average Width	Not Null
BCode	AutoNumber		√
issue_date	Date/Time		
return_date	Date/Time		

Таблица 4.7. Структура таблицы RESERV

Column	Access DataType	Average Width	Not Null
BCode	AutoNumber		√
SNum	AutoNumber		√
order_date	Date/Time		

Для примера рассмотрим определение типов атрибутов отношения PUBLISH. Для этого:

- сделайте двойной клик по сущности: в результате откроется окно <Columns>, и перейдите на вкладку <Access> (рис. 4.14).
- выберите ключевой атрибут P_ID – на вкладке Access отобразятся свойства атрибута по умолчанию:
Access Datatype (тип данных Access) –
Text(18) Average Width (размер поля) –
18
Опция NotNull – установлена;
- поменяем для этого атрибута тип: в списке типов Access выберем Auto Number (счетчик);
- тип атрибутов PName и town оставим по умолчанию text(18);
- нажмите <OK>.

Аналогично установите тип и если надо размеры атрибутов всех остальных отношений. В результате получим физическую модель проекта как на рис. 4.15.

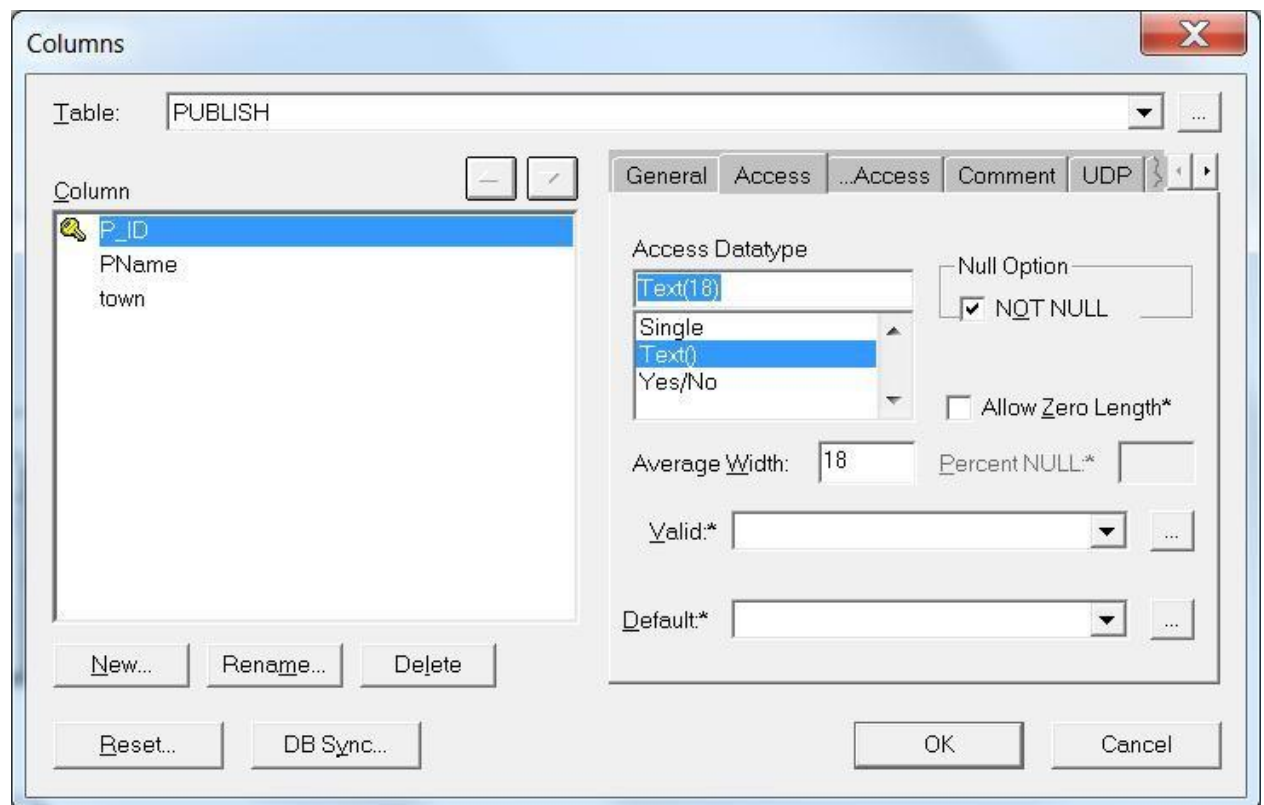


Рис. 4.14 – Окно свойств атрибутов, вкладка Access

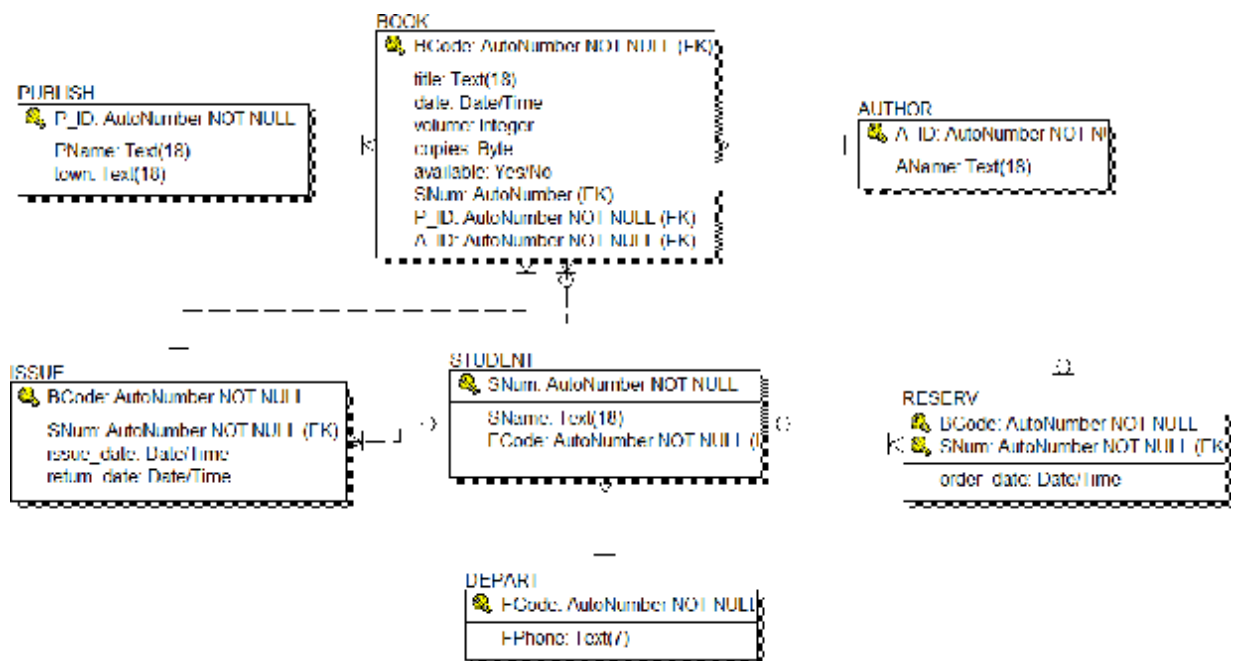


Рис. 4.15 – Физическая модель проекта

Лабораторная работа № 3. Настройка MySQL и работа в mysql

По сути, система баз данных представляет собой способ манипулирования информационными списками. При этом информация может поступать из различных источников. Например, это могут быть данные, полученные в ходе исследований, деловые записи, заказы потребителей, спортивная статистка, отчеты о продажах, отчеты об ошибках или оценки учащихся. Несмотря на то, что СУБД могут содержать широкий диапазон информации, вы не будете использовать СУБД только из-за нее самой.

СУБД проявляет себя как мощный инструмент тогда, когда информация, которую необходимо организовать и которой уже необходимо манипулировать, становится объемной и сложной, вследствие чего записи становятся непонятными и трудно обрабатываемыми вручную. Конечно, базы данных могут использоваться большими корпорациями, обрабатывающими миллионы транзакций в день. Но СУБД может также потребоваться для маломасштабных операций, которые обрабатывает один человек в личных целях.

СУБД часто применяется для таких задач, для решения которых обычно используются картотеки. Действительно, базу данных можно представить в некотором роде большой картотекой. Можно назвать несколько очень серьезных преимуществ ведения данных в электронном виде перед хранением информации вручную. Например, в офисе, где ведутся учетные записи клиентов, использование СУБД MySQL значительно облегчит вам жизнь.

Каких преимуществ можно ожидать от использования той СУБД, как СУБД MySQL? Это зависит от ваших конкретных задач и требований и может сильно варьироваться.

Итак, как же работает СУБД MySQL? Лучший способ узнать это - самостоятельно с ней поработать.

Вы научитесь:

- устанавливать, настраивать, проверять работу сервера;
- соединяться с сервером MySQL.

На занятии вы должны :

1. *изучить учебный материал*

История MySQL

В настоящее время реляционные системы управления базами данных (СУБД) являются важным инструментом во многих областях, начиная с таких традиционных областей применения, как бизнес, научные исследования, образование, и заканчивая разработкой поисковых серверов в Internet. Однако, несмотря на важность наличия хорошей базы данных для введения. И доступа к информационным ресурсам, многие организации не применяют их в своей работе. Исторически сложилось так, что СУБД стоили очень дорого, а продавцы устанавливали очень высокие цены, как на программное

обеспечение, так и на услуги по технической поддержке. Кроме того, механизмы СУБД требовали удовлетворения существенных требований по производительности от аппаратных платформ, что еще больше повышало стоимость таких решений.

В последние годы ситуация резко изменилась как с точки зрения программного обеспечения, так и с точки зрения аппаратуры. Компьютеры одновременно дешевеют и становятся мощнее. При этом обозначилась тенденция создания высокопроизводительных операционных систем, которые МОЖНО купить по цене дешевых лазерных дисков или даже получить бесплатно через Internet, таких как операционные системы, созданные на базе ОС BSD UNIX (FreeBSD, NetBSD, OpenBSD), а также различные версии ОС Linux (RedHat, Caldera, LinuxPPC, ASPLinux, SuSe).

Создание операционных систем, позволяющих максимально использовать возросшие возможности компьютеров, произошло прежде всего благодаря тому, что были разработаны и свободно распространялись такие средства разработки, как компилятор GNU C gcc. Эти попытки создания программного обеспечения, которое было бы доступно для каждого, кто хочет его получить, дало толчок движению, которое сейчас известно как Open Source movement, и дало жизнь многим важным и нужным программам.

Программное обеспечение баз данных тоже стало более доступным. Достаточно вспомнить такие СУБД, как Postgres и mSQL, которые можно получить за невысокую плату или совсем бесплатно. Совсем недавно такие мощные коммерческие производители, как Informix и Oracle начали предлагать свое программное обеспечение совсем бесплатно для таких ОС, как Linux. (Однако эти продукты поставляются обычно в двоичном виде и без поддержки, что снижает их пользу.)

Одним из новейших явлений на "арене" недорогих или бесплатных баз данных является MySQL, реляционная СУБД, типа клиент/сервер, созданная в Скандинавии. СУБД MySQL включает в себя SQL-сервер и программы-клиенты, осуществляющие доступ к серверу, средства администрирования и программный интерфейс для программирования своих собственных программ.

СУБД MySQL "уходит своими корнями" в 1979 год и происходит от СУБД UNIREG, разработанной Михаэлем Видениусом ("Монти") по заказу шведской компании ТсХ. В 1994 году компания ТсХ начинает искать SQL-сервер для создания Web-приложений. Было опробовано несколько коммерческих серверов, но те оказались слишком медленными для больших таблиц данных, которые использовались в компании ТсХ. Они также обратили внимание на СУБД mSQL, но та не совсем удовлетворяла задачам компании ТсХ. Поэтому Монти начал работать над созданием нового сервера. Программный интерфейс был разработан как аналог mSQL, так как тогда в наличии было несколько бесплатных средств mSQL. Пользуясь аналогичным интерфейсом, эти же средства можно использовать для СУБД MySQL с минимальными затратами на перенос.

В 1995 году Дэвид Оксмарк, работающий в компании Detron НВ, начал "давление" на ТсХ с тем, чтобы она начала распространять СУБД MySQL через Internet. Кроме того, Дэвид принял участие в работе над документацией. Версия 3.11.1 СУБД MySQL была выпущена в свет в 1996 году в виде двоичной дистрибуции для работы под управлением ОС Linux и Solaris. Сегодня MySQL работает на многих платформах и распространяется как в двоичных кодах, так и в исходных текстах.

СУБД MySQL нельзя причислить в полной мере к проектам Open Source, так как при определенных условиях покупка лицензии все же требуется. Тем не менее MySQL пользуется широкой популярностью среди сторонников движения Open Source, так как условия лицензирования здесь не очень строгие. (По сути, MySQL распространяется бесплатно за исключением тех случаев, когда вы намереваетесь ее продавать или продавать услуги, создаваемые с ее помощью.)

Но популярность СУБД MySQL не ограничивается только сообществом Open Source. Да, она работает на персональных компьютерах (при этом многие разработки, производящиеся на MySQL, создаются на недорогих Linux-системах). Но MySQL обладает отличной переносимостью и может с тем же успехом использоваться на дорогих коммерческих операционных системах (таких как Solaris, Irix или Windows) и на любой аппаратуре вплоть до мощных серверов. Более того, так же как и ее более "дорогие соперники", она позволяет обрабатывать большие базы данных, содержащие миллионы записей.

Теперь СУБД MySQL предстала перед нами во всей красе: бесплатные операционные системы, работающие на мощных, но не очень дорогих персональных компьютерах, предоставляющих в распоряжение пользователей значительную вычислительную мощность и более широкий выбор операционных систем, чем прежде. Снижение экономических барьеров позволяет получить доступ к базам данных большему количеству людей и организаций, чем когда бы то ни было.

Прошли те времена, когда многие организации могли только мечтать об использовании огромных возможностей мощных реляционных СУБД. Теперь это уже не проблема. Цены на СУБД снизились, поэтому использовать СУБД можно уже и индивидуальным пользователям. Люди, даже не мечтавшие об использовании баз данных, теперь могут широко их применять, например, для реализации задачи хранения и просмотра данных о генеалогических исследованиях, ведения собственных коллекций (бабочки, марки, бейсбольные карточки и т.д.), помощи в своем бизнесе на начальной стадии или обеспечения поисковых возможностей персональных Web-страниц.

Преимущество MySQL перед другими базами данных

Если вы действительно ищете бесплатную или недорогую СУБД, можно остановить свой выбор на одной из трех кандидатур: MySQL, mSQL и Postgres или на бесплатных, но неподдерживаемых СУБД от крупных производителей. Сравнивая СУБД MySQL с другими базами данных, всегда помните о том,

какой фактор является для вас самым важным. Производительность, поддержка, возможности (соответствие стандартам языка SQL, возможность расширения и т.д.); условия лицензирования и ограничения и, наконец, цена. Принимая во внимание эти соображения, СУБД MySQL обладает многими преимуществами.

Быстродействие. MySQL - достаточно быстродействующая СУБД. Разработчики склоняются к мнению, что СУБД MySQL является одной из самых быстрых баз данных из имеющихся на современном рынке. В этом можно удостовериться, посетив Web-узел <http://www.mysql.com/benchmark.html>. Эта страница позволяет делать сравнительную проверку производительности на Web-узле MySQL.

Простота использования. СУБД MySQL является высокопроизводительной и относительно простой в использовании СУБД, которую значительно проще установить и администрировать, чем многие большие системы.

Цена. СУБД MySQL распространяется бесплатно для домашнего использования.

Поддержка языка запросов. MySQL "понимает" команды языка SQL (Structured Query Language - структурированный язык запросов). Этот язык применяется во всех современных СУБД. MySQL также поддерживает интерфейс ODBC (Open Database Connectivity), протокол интерфейса с базами данных, разработанный компанией Microsoft.

Возможности. Сервер позволяет одновременно подключаться неограниченному количеству пользователей. Доступ к серверу СУБД MySQL можно осуществить в интерактивном режиме с помощью различных интерфейсов, позволяющих вводить запросы и просматривать полученные результаты: это программы-клиенты, работающие с командной строкой, Web - браузеры или программы-клиенты, работающие в системе X Window. Кроме того, в наличии имеются программные интерфейсы для таких языков, как C, Perl, Java, PHP и Python. Таким образом, можно использовать как готовое клиентское программное обеспечение, так и создавать свое собственное.

Взаимодействие и безопасность. MySQL предназначена для работы в сети и может быть доступна через Internet, таким образом, с данными можно работать в любой точке земного шара. Но при этом СУБД MySQL снабжена развитой системой защиты от несанкционированного доступа.

Переносимость. СУБД MySQL отлично работает как под управлением самых различных версий UNIX, так и под управлением систем, не использующих UNIX, таких как Windows и OS/2. СУБД MySQL работает как

на домашних ПК, так и на мощных серверах.

Открытое распространение. Дистрибуция СУБД MySQL легкодоступна. Для этого достаточно воспользоваться Web браузером. Если вы не понимаете как что-либо работает, просмотрите исходный код. Если вам что-то в работе не нравится, можно внести коррективы.

Кроме того, СУБД MySQL имеет хорошую поддержку.

- СУБД MySQL снабжена расширенным справочным руководством (450 страниц и постоянное дополнение).
- Можно заключить контракты на техническую поддержку с самими разработчиками MySQL.
- Есть список рассылки, на который может подписаться любой желающий. В нем принимает участие очень много грамотных пользователей, в том числе включая и самих разработчиков MySQL. Этого будет достаточно для большинства пользователей СУБД MySQL.

Ответы на вопросы, размещенные в списке рассылки, можно получить за считанные минуты. Обнаружив ошибку, разработчики исправят ее за считанные дни (иногда часы!), и исправления можно получить немедленно по Internet. Это резко контрастирует с нередко разочаровывающим опытом работы с бюрократическими отделами поддержки больших компаний, продающих СУБД.

Если вы находитесь на этапе выбора, то СУБД MySQL является идеальным кандидатом для проведения оценки. Эту СУБД можно проверить в работе безо всякого риска и финансовых расходов. Если у вас возникают какие-либо проблемы, можно обратиться к списку рассылки. Оценка всегда требует определенного времени, НО это не относится к рассматриваемой СУБД, ее установка и установка требует гораздо меньше времени, чем многие другие современные СУБД.

2. Выполнить задание:

2.1. СУБД MySQL. Использование клиентского *mysql*

Создание БД, таблиц, осуществление основных запросов

1. Выполнить соединение с сервером. Для этого потребуется зайти в **Far-с:/** и, если вы видите файл [mysql.exe](#) (Не запускайте его!!!) набрать в командной строке следующую команду и нажать **Enter!!!** :

```
mysql -u stud -p -h 192.168.xx.xx  
пароль ****
```

(IP-адрес сервера и имя пользователя и пароль узнать у преподавателя)

где с помощью опции -h задается хост для вашего компьютера 192.168.XX.XX, и опции -u задается имя пользователя для регистрации, -p означает использование пароля.

После этой команды у Вас должно появиться:
Welcome to the MySQL monitor. Commands end with ; or \g.
Your MySQL connection id is 6 to server version:5.5.29

Type 'help;' or '\h' for help. Type '\c' to clear the buffer.
mysql>

После выполнения каждой команды - сохранять скриншоты!!!

Любой пользователь может захотеть создать персональную базу данных для собственных экспериментов, но делать это разрешено только пользователю **root**. Он же может создавать учетные записи новых пользователей и предоставлять им необходимые привилегии. Рассмотрим пример (**набрать и после выполнения сделать его скриншот, причем XXistXX это Ваше сетевое имя**):

CREATE DATABASE XXistXX;

Первая инструкция создает базу данных XXistXX.

Пропишите созданную базу данных с помощью команды **use xxistgg;**

Создайте в этой БД **XXistXX** таблицу **stud** с полями **uid** (тип данных integer, размер поля 7), **Lastname** (тип данных char, размер поля 30), **Name** (тип данных char, размер поля 30), **kurs** (тип данных integer, размер поля 1). Для этого наберите команду и сохраните ее скриншот:

create table stud(uid integer(7), lastname char(30), name char(30), kurs integer(1));

Создайте, самостоятельно определив типы данных и размеры полей, таблицу **Prep** с полями: **uid, lastname, name, staj, disciplina** (**Наберите команду, выполните ее и сохраните в виде скриншота**).

Создайте самостоятельно базу данных **Kolledjxxistgg**, где **xxistgg** - **ваше сетевое имя**. (**Наберите команду, выполните ее и сохраните в виде скриншота**).

Выполните запросы: (набрать и после выполнения сделать его скриншот)

select now();

select user();

select version();

О результатах, которые выдает каждый из этих запросов Вы сможете прочитать в пункте "Установка и завершение связи с сервером"

Выведите справку по работе в mysql с помощью команды (набрать и после выполнения сделать скриншот).

\h;

По этой справке выпишите себе основные команды необходимые для работы в программе mysql.

Наберите команду для отображения всех баз данных на данном сервере:

show databases;

Для завершения соединения с сервером 192.168.xx.xx наберите команду **exit;** или **\q;**

Изучите материал

Оператор **GRANT** предназначен для создания пользователей MySQL и позволяет настроить их привилегии.

Синтаксис оператора **GRANT**:

```
GRANT privileges (columns)  
ON what  
TO user IDENTIFIED BY "password"  
WITH GRANT OPTION
```

Для успешного его выполнения обязательно нужно правильно определить следующую информацию.

- **privileges** (привилегии). Привилегии присваиваются определенному пользователю. Используемые в операторе GRANT спецификаторы привилегий описаны в представленной ниже таблице.

Спецификатор привилегий	Разрешенная операция
ALTER	Изменение таблиц и индексов
CREATE	Создание баз данных и таблиц
DELETE	Удаление существующих записей из таблиц
DROP	Удаление баз данных и таблиц
INDEX	Создание и удаление индексов
INSERT	Вставка новых записей в таблицы
REFERENCES	Не используется
SELECT	Извлечение существующих записей из таблиц
UPDATE	Изменение существующих записей таблиц
FILE	Чтение и запись файлов сервера
PROCESS	Просмотр информации о внутренних потоках сервера и их удаление
RELOAD	Перезагрузка таблиц разрешений или обновление журналов, кэша компьютера или кэша таблицы
SHUTDOWN	Завершение работы сервера

ALL	Все операции. Аналог - ALL PRIVILEGES
USAGE	Полное отсутствие привилегий

Спецификаторы привилегий, входящие в первую группу этой таблицы, применяются к базам данных, таблицам и столбцам. Спецификаторы второй группы определяют административные привилегии. Как правило, они используются довольно редко, поскольку позволяют пользователю влиять на работу сервера. (Не каждому пользователю, например, необходима привилегия **SHUTDOWN**.) В третью группу входят два отдельных спецификатора: спецификатор **ALL** предоставляет "все привилегии", а **USAGE** означает "полное отсутствие привилегий". В последнем случае создается новый пользователь, не обладающий никакими правами.

- **columns** (столбцы). Столбцы, к которым применяются определенные привилегии. Этот параметр необязателен и используется только при установке привилегий для столбцов. Имена нескольких столбцов отделяются друг от друга запятыми.
- **what** (что). Уровень применения привилегий. Привилегии могут быть глобальными (применяемыми ко всем базам данных и их таблицам), уровня баз данных (применяемыми ко всем таблицам определенной базы данных) или уровня таблицы. Используя оператор **columns**, можно определить также привилегии уровня столбца.
- **user** (пользователь). Пользователь, которому присваиваются привилегии. В некоторых версиях MySQL необходимо указывать как имя пользователя, так и компьютер, с которого он сможет подключаться. Такой способ задания легко позволяет определить двух пользователей с одинаковым именем, но подключающихся с разных компьютеров. Возможности MySQL позволяют их различать и наделять различными правами.

Имя пользователя в некоторых версиях MySQL представляет собой имя, используемое при подключении к серверу. Оно вовсе необязательно должно быть связано с именем регистрации в ОС UNIX или Windows. Если имя пользователя MySQL не определено явным образом, клиентские программы по умолчанию используют его в качестве регистрационного имени, однако это необязательно. Не существует также каких-либо особых требований, чтобы суперпользователь MySQL, обладающий максимальными правами, имел имя **root**. По желанию его можно изменить в таблицах разрешений на **nobody**, если для выполнения определенных операций требуются его полномочия.

- **password** (пароль). Присвоенный пользователю пароль, который не является обязательным. Если для нового пользователя опустить выражение **IDENTIFIED BY**, пароль ему присвоен не будет (что не совсем разумно с точки зрения безопасности). Если же этот оператор задается для уже существующего пользователя, введенный пароль заменит используемый до настоящего момента. Старый пароль

останется неизменным, если новый не будет определен. Строка пароля, задаваемая с помощью выражения `IDENTIFIED BY`, должна представлять собой буквенную строку, которую при записи зашифрует оператор `GRANT`. Поэтому не следует использовать функцию `PASSWORD()`, применяемую с оператором `SET PASSWORD`.

Оператор **WITH GRANT OPTION** является необязательным. С его помощью можно предоставить пользователю все привилегии, определенные оператором **GRANT** для других пользователей. Этот оператор можно применять для делегирования возможностей определенных категорий другим пользователям. В именах пользователей, баз данных, таблиц и паролях, записываемых в таблицу разрешений, строчные буквы отличаются от заглавных. Регистр в именах компьютеров и столбцов таблиц не учитывается.

Пользователю базы данных необязательно знать, как установить MySQL. В крупных организациях есть системные администраторы, которые этим занимаются. Что касается разработчиков, то им нужно понимать особенности данного процесса. Именно здесь у них появляется доступ к различным конфигурационным установкам, с помощью которых можно настроить производительность программы. Естественно, необходимо обладать правами администратора на том компьютере, где MySQL устанавливается в виде сервиса, запускаемого автоматически. Программу можно запускать также из персональных учетных записей.

Загрузка дистрибутива

MySQL можно установить двумя способами: скомпилировав исходные коды программы или воспользовавшись предварительно скомпилированными двоичными файлами. Первый вариант допускает больше возможностей в плане конфигурации, но более продолжителен. Второй вариант удобнее, так как есть готовые дистрибутивы для многих операционных систем. Сейчас существуют версии MySQL для FreeBSD, HP-UX, IBM AIX, Linux, MacOS X, SCO, SGI Irix, Solaris и многих вариантов Microsoft Windows.

Информацию обо всех дистрибутивах можно получить на Web-узле www.mysql.com. Там же публикуются последние новости о программе.

Инструментарий, поставляемый с MySQL

Дистрибуция MySQL включает следующие программные продукты:

SQL-сервер. Эта программа обеспечивает доступ к базам данных.

Клиентские программы для доступа к серверу. Это интерактивная программа, позволяющая делать запросы и просматривать полученные результаты, и несколько административных и сервисных программ, помогающих работать с вашим Web-узлом. Одна сервисная программа позволяет управлять сервером. Другие позволяют импортировать или экспортировать данные, проверять права доступа и т. д.

Клиентская программа, позволяющая создавать свои собственные программы. Клиентские программы можно писать на C, так как сама библиотека написана на C, но библиотека также позволяет создавать шлюзы для написания программного обеспечения на других языках программирования.

Проверка исходных требований

Если устанавливаются двоичные файлы, следует убедиться в том, что система соответствует исходным требованиям программы. Важным моментом является поддержка потоков. При установке в старых версиях некоторых операционных систем MySQL требует наличия отдельной библиотеки потоковых функций POSIX. POSIX - это международный стандарт, определяющий работу системных сервисов, а потоки - это механизм, позволяющий программам выполнять несколько задач одновременно.

Современные операционные системы поддерживают стандарт POSIX. Если вы все же не уверены, проверьте в интерактивной документации, будет ли программа MySQL работать в данной версии операционной системы.

Выбор версии

Команда разработчиков MySQL публикует тестовые и стабильные версии дистрибутивов отдельно. Информацию о статусе той или иной версии программы можно найти на Web-узле. Эти же сведения закодированы в названии дистрибутива.

В целом лучше выбирать самую новую версию. Исключение составляют лишь альфа - версии, в которых впервые вводятся новые функциональные возможности. Программа MySQL отвечает самым высоким критериям качества и надежности, поэтому ее бета-версии зачастую вполне сопоставимы с финальными версиями других программ.

Установка и завершение связи с сервером

Здесь будут рассмотрены различные способы взаимодействия с сервером MySQL. Допускаются клиентские подключения по протоколу IP, а также через сокеты UNIX и именованные каналы. Имеется ряд готовых клиентов. Простейшие из них - это утилиты командной строки, являющиеся частью проекта MySQL. Кроме них есть графические клиенты для различных операционных систем. Те, кто работают в Windows, могут пользоваться специальным драйвером ODBC, а любой, у кого есть Web-браузер, сможет получить в свое распоряжение Web-клиенты.

Клиент-серверное взаимодействие средствами TCP/IP

Программа MySQL работает по протоколам TCP/IP, как и другие Internet-сервисы. Соединения различаются по имени узла и номеру порта. По умолчанию используется порт 3306, но это конфигурируемый параметр.

В MySQL применяется архитектура "клиент/сервер". Выделенный сервер способен принимать запросы от множества клиентов. На сервер ложится достаточно большая нагрузка, тогда как клиенты представляют собой всего лишь программные оболочки.

Имя клиентского компьютера, с которого устанавливается соединение, преобразуется в IP-адрес: четыре числа, разделенных точками. Например, специальному имени localhost соответствует адрес 127.0.0.1. Иногда такое преобразование выполняется на основании записей локальных таблиц ядра, но чаще всего - с помощью сервера DNS (Domain Name System - система доменных имен).

По умолчанию запрос на подключение поступает через порт 3306. Этот порт постоянно прослушивается сервером MySQL. При ответе на запрос сервер создает сеанс связи с клиентом. За сеансом закрепляются два порта: один будет использоваться для отправки данных, а другой - для их приема.

В ходе сеанса клиент посылает серверу команды, имеющие вид инструкций SQL. В ответ на некоторые инструкции сервер возвращает данные, а клиент форматирует их для отображения на экране.

В проект MySQL входит много специальных клиентов, а также один клиент общего назначения, называемый **mysql**.

Различие между MySQL и mysql

Во избежание разночтений, надо подчеркнуть, что **MySQL** относится ко всей СУБД MySQL, а "**mysql**" - название конкретной клиентской программы. В разговорной речи это звучит одинаково, но в написании одно наименование пишется заглавными буквами, а другое - прописными.

С одной стороны, MySQL, произносится как "май-эс-кью-элл". С другой стороны, SQL произносится как "сиквел". Это зависит от того, кого вы спрашиваете. Произносите это слово как вам угодно, но будьте готовы к тому, что кто-нибудь, когда-нибудь вас поправит.

Если программа MySQL установлена правильно, то путь к утилите **mysql** будет указан в переменной среды **PATH**. По умолчанию в UNIX утилита будет записана в каталог **/usr/local/bin**, а в Windows - в каталог **c:\mysql\bin**.

Если не заданы другие установки, утилита **mysql** пытается подключиться к серверу MySQL на узле localhost (порт 3306), используя для доступа к базе данных регистрационное имя текущего пользователя. Стандартные привилегии доступа программы MySQL разрешают любому пользователю локального узла обращаться к серверу.

Для подключения к удаленному серверу необходимо указать его адрес с помощью опции **-h host_name** (альтернативная форма **--host=host_name**). Альтернативный порт задается опцией **--port**, имя пользователя - опцией **-u user_name** (альтернативная форма **--user=user_name**), пароль - опцией **-p** (альтернативная форма **--password**). Значение опции указывается после знака равенства.

Использование опций подключения:

```
C:\mysql\bin>mysql --user=root --password --host=localhost --port=3306
```

```
Enter password: *****
```

```
Welcome to the MySQL monitor. Commands end with ; or \g.
```

```
Your MySQL connection id is 6 to server version: 3.23.56-nt
```

```
Type 'help;' or '\h' for help. Type '\c' to clear the buffer.
```

```
mysql>
```

Последним параметром утилиты **mysql** является имя базы данных.

Установленное соединение с сервером можно прервать в любой момент командой **QUIT**:

```
mysql> QUIT
```

Вуе

Кроме того, завершить сеанс можно комбинацией клавиш **<Ctrl+C>**.

Ввод запросов

Запросы к базе данных можно вводить только после подключения к серверу.

Для ввода запроса в mysql достаточно его напечатать, ввести в конце точку с запятой и нажать клавишу **<Enter>**. Точка с запятой индицирует окончание запроса (завершать запрос можно и двумя символами **"\g"**, если вам не лень вводить два символа вместо одного).

После ввода запроса mysql передает его серверу на выполнение. После обработки запроса сервер передает результат клиенту mysql. Клиент отображает полученный результат.

Вот пример простого запроса и его результат:

```
mysql> SELECT NOW();
```

```
+-----+
| NOW()          |
+-----+
| 2013-10-15 11:31:25 |
+-----+
1 row in set (0.00 sec)
```

Он заключается в выводе текущей даты и времени. (Сама по себе функция NOW() бесполезна, но ею можно воспользоваться в самых различных выражениях, - например, для вычисления разницы между датами.)

Программа mysql также выводит количество строк в результате.

Программа mysql воспринимает точку с запятой как окончание запроса, поэтому, при необходимости, запросы могут занимать несколько строк.

```
mysql> SELECT NOW(),
```

```
-> USER(),
```

```
-> VERSION();
```

```
+-----+-----+-----+
| NOW()          | USER()      | VERSION()    |
+-----+-----+-----+
| 2004-02-15 11:22:18 | root@localhost | 3.23.56-nt |
+-----+-----+-----+
1 row in set (0.00 sec)
```

Обратите внимание, что подсказка меняется с **mysql>** на **->** после ввода первой строки запроса. Это означает, что mysql ожидает продолжения ввода запроса. Это очень важный элемент обратной связи, показывающий пользователю, что даже если вы забыли ввести точку с запятой, mysql ожидает ее ввода и показывает, что ввод запроса еще не завершен. Другими словами, пока пользователь сидит и удивляется, почему так долго нет ответа от mysql, клиент mysql, в свою очередь, ожидает завершения ввода запроса!

В большинстве случаев регистр, на котором вводится команда, не имеет значения. Запросы, приведенные ниже, эквивалентны.

```
SELECT USER()
```

```
SELECT user()
```

SELECT UsEr()

При вызове функции в запросе недопустим пробел между именем функции и последующими скобками.

```
mysql> SELECT NOW();
```

```
+-----+
| NOW()      |
+-----+
| 2004-02-15 11:31:25 |
+-----+
1 row in set (0.00 sec)
```

```
mysql> SELECT NOW ();
```

ERROR 1064: You have an error in your SQL syntax near '()' at line 1

Эти два запроса выглядят аналогично, но второй сообщает об ошибке из-за того, что скобки не следуют прямо за именем функции.

Начав вводить запрос, а затем передумав, введите "\c" для его отмены.

```
mysql> SELECT NOW(),
```

```
->USER(),
```

```
->\c
```

```
mysql>
```

Обратите внимание, что подсказка опять изменилась на mysql>. Это свидетельствует о том, что mysql ожидает ввода нового запроса.

Запросы можно сохранять в файле и затем запускать их на выполнение прямо из файла, не создавая вновь. Для этого можно воспользоваться функциями переназначения ввода/вывода. Например, если запрос сохранен в файле my_file.sql, его можно выполнить с помощью команды:

```
mysql < my_file.sql
```

Таким образом можно вызвать любой файл. Для того чтобы показать, что файл содержит операторы SQL, был указан суффикс .sql.

USER()

Эта функция возвращает имя пользователя, начавшего текущий сеанс работы с сервером. Сообщает также имя компьютера, за которым работает пользователь.

```
mysql> SELECT USER();
```

```
+-----+
| USER()      |
+-----+
| root@localhost |
+-----+
1 row in set (0.00 sec)
```

VERSION()

Эта функция сообщает номер версии сервера MySQL.

```
mysql> SELECT VERSION();
```

```
+-----+
| VERSION() |
```

+-----+

| 3.23.56-nt |

+-----+

1 row in set (0.00 sec)

Отчет :

- Показать результат преподавателю.
- Удалить программу MySQL.

Лабораторная работа № 4. Создание базы данных и таблиц

Работа с базой данных предполагает несколько этапов.

- Создание (инициализация) базы данных.
- Создание таблиц в базе данных.
- Взаимодействие с таблицами посредством операций вставки, выборки, модификации или удаления данных.

Выборка существующих данных фактически является самой частой операцией при работе с базой данных. В числе других наиболее частых операций над таблицами базы данных являются вставка новых данных, их модификация и удаление.

Начнем с самого начала. Поэтому нашей первой операцией будет наименее часто выполняемая операция - создание базы данных. Затем создание таблиц, загрузка начальных данных, а уже потом наиболее популярная в базах данных операция - выборка.

Вы научитесь:

- настраивать MySQL-client программу
- создавать базу данных;
- открывать базы данных на сервере;
- создавать таблицы в базе данных.

Начало занятия:

1. *изучить учебный материал:*

1.1. Правила присвоения имен в СУБД MySQL

Почти каждый оператор SQL неким образом обращается к базе данных или элементам, ее составляющим. Этот раздел освещает синтаксические правила ссылки на базы данных, таблицы, столбцы, индексы и псевдонимы. Имена рассматриваются с точки зрения чувствительности к регистру.

Ссылка на элементы баз данных

При присвоении имен элементам базы данных мы ограничены набором допустимых символов и длиной имени. Вид имени также зависит от контекста, в котором оно используется.

Допустимые символы. Имя может состоять из любого алфавитно-цифрового символа из набора символов, которые использует сервер, плюс символы '_' (символ подчеркивания) и '\$' (знак доллара). Имена могут начинаться с любого допустимого символа, включая цифры. Однако имя столбца не может состоять из одних цифр: это сделает его неотличимым от чисел. В СУБД MySQL предусмотрена уникальная возможность начинать имя с цифры. При этом необходимо быть особенно внимательным при использовании имен, содержащих символ "Е" или "е". Такие имена приводят к неоднозначности. Выражение $23e + 14$ может означать сложение значения столбца 23e плюс 14. А что может значить $23e+14$? Это имя или это число в научной нотации?

Длина имени. Имена баз данных, таблиц, столбцов и индексов имеют длину до 64 символов. Имена псевдонимов могут иметь до 256 символов.

Квалификаторы имен. Для обращения к базе данных достаточно просто указать ее имя.

USE db_name

SHOW TABLES FROM db_name

где *db_name* - имя вашей базы данных (соответствует сетевому имени).

Для ссылки на таблицу есть два варианта: дать полностью квалифицированное имя таблицы, состоящее из двух частей, разделенных точкой, - имени базы данных и имени таблицы:

SHOW TABLES FROM db_name.tbl_name

SELECT * FROM db_name.tbl_name

где *db_name.tbl_name* - **имя вашей базы данных.имя вашей таблицы**

При указании только имени таблицы обращение происходит к текущей базе данных. Два следующие оператора идентичны:

SHOW TABLES FROM member

SELECT * FROM samp_db.member

Для ссылки на столбец есть три варианта: указание полностью квалифицированного имени столбца, указание частично квалифицированного имени столбца и указание неквалифицированного имени столбца. Полностью квалифицированное имя столбца (в нотации **db_name.tbl_name.col_name**) дает ссылку на определенный столбец из определенной таблицы, входящей в состав определенной базы данных. Частично квалифицированное имя столбца (записанное как **tbl_name.col_name**) ссылается на определенный столбец в определенной таблице. Вот два запроса, ссылающиеся на одноименные столбцы. Но из контекста предложений FROM видно, что они относятся к разным таблицам:

SELECT last_name, first_name FROM president

SELECT last_name, first_name FROM member

SHOW TABLES FROM member

SELECT * FROM samp_db.member

Обычно давать полностью квалифицированные имена не нужно, хотя это и не будет ошибкой. Использование оператора USE делает базу данных текущей. В операторе SELECT таблица становится текущей таблицей для всех столбцов в данном операторе, если она одна указана в предложении FROM. Имена таблиц нужно квалифицировать только при наличии в операторе нескольких таблиц (когда таблицу или базу данных нельзя определить по контексту). Вот несколько ситуаций, при которых возникает такая неоднозначность.

- Запросы, ссылающиеся на таблицы из нескольких баз данных. Любая таблица, не являющаяся в данный момент текущей, должна указываться нотацией *db_name.tbl_name*. Это помогает определить, из какой базы данных использовать таблицу.
- Запросы, выбирающие столбцы из нескольких таблиц. Это запросы, в которых столбец с одним и тем же именем присутствует в нескольких таблицах.

Чувствительность к регистру в операторах SQL

Правила, которым подчиняется синтаксис операторов SQL с точки зрения чувствительности к регистру, зависят от части оператора и от операционной системы, под управлением которой работает сервер.

- Ключевые слова и имена функций. Ключевые слова и имена функций не зависят от регистра. Они могут вводиться в любом регистре. Вот абсолютно идентичные операторы:
- **SELECT NOW()**
- **select now()**

SeLeCt nOw()

- Имена баз данных и таблиц. Имена баз данных и таблиц подчиняются правилам именования каталогов и файлов в операционной системе, работающей на сервере. Вследствие этого, чувствительность к регистру имен баз данных и таблиц подчиняется правилу, предусмотренному для имен файлов и принятому в операционной системе. Для сервера, работающего под управлением ОС UNIX, имена баз данных и таблиц зависят от регистра, а в ОС Windows имена файлов не зависят от регистра, поэтому имена баз данных и таблиц поведут себя аналогичным образом.

Об этом нужно помнить в случае, когда ставится задача переноса базы данных из ОС UNIX в ОС Windows. Если в процессе работы с базой данных были созданы две таблицы, например, abc и ABC, то после переноса базы данных в среду ОС Windows они будут неразличимы. Как этого избежать? Достаточно выработать для себя правило, регламентирующее создание всех баз и таблиц только в одном регистре. Только тогда при переносе данных из одной операционной среды в другую можно не беспокоиться о неоднозначности данных.

- Имена столбцов и индексов. В СУБД MySQL имена столбцов и индексов не чувствительны к регистру. Следующие операторы будут эквивалентны:
- **SELECT name FROM student**
- **SELECT NAME FROM student**

SELECT nAmE FROM student

- Имена псевдонимов. В СУБД MySQL псевдонимы чувствительны к регистру. Псевдоним можно объявлять в любом регистре (верхнем или нижнем, в обоих сразу), но при дальнейших обращениях к нему необходимо четко соблюдать первоначальный регистр.

Опыт показывает, что независимо от того, является база данных, таблица или псевдоним чувствительным к регистру, их необходимо указывать в одном и том же регистре на протяжении одного и того же запроса. Это правило не относится к ключевым словам языка SQL, именам функций, столбцов и индексов, которые можно вводить в разных регистрах.

Естественно, необходимо стремиться делать запросы как можно более читабельными и стремиться вырабатывать определенный стиль ввода элементов запроса.

1.2. Типы данных и столбцов в MySQL

Типы данных СУБД MySQL

В СУБД MySQL существует несколько типов данных. Это общие категории, по которым можно классифицировать данные

Цифровые данные

Числами будем считать значения, наподобие 48 или 193.62. СУБД MySQL различает целые числа (числа без дробной части) или числа с плавающей точкой (числа с дробной частью). Целые представляются в десятичном или шестнадцатеричном формате.

Целое состоит из последовательности цифр. Целое в шестнадцатеричном представлении состоит из '0x', с последующими шестнадцатеричными цифрами (от 0 до 9 и от a до f). Например, 0x0a равно десятичному 10, а 0xffff равно 65535. Нецифровые шестнадцатеричные цифры могут быть представлены как в нижнем, так и в верхнем регистре. Но ведущая последовательность '0x' должна быть представлена только в нижнем регистре.

Числа с плавающей точкой состоят из последовательности цифр, разделенных точкой. Одна из этих последовательностей (но не все сразу) может быть пустой.

СУБД MySQL может работать и с научной нотацией чисел. Этот формат выглядит следующим образом: число в целом или плавающем формате, затем символ экспоненты "e" или "E", знак "+" или "-" и целочисленная экспоненциальная часть. 1.34E+12 или 43.27e-1 - это числа в правильной научной нотации. Однако 1.34E12 является неправильной научной нотацией, так как перед экспонентой отсутствует символ знака. Шестнадцатеричные числа не могут быть представлены в научной нотации: символ "e", предваряющий экспоненту, здесь является полноправным числом, и таким образом выражение становится неоднозначным.

Для обозначения отрицательных значений, числа предваряются знаком минус "-".

Строковые (символьные) данные

Строками являются такие значения, как "Madison, Wisconsin" или "patient shows improvement". Для обозначения строки она берется в одинарные или двойные кавычки.

Для обозначения специальных символов применяются управляющие последовательности. Они представлены в таблице. 1. Последовательность начинается с символа обратной косой черты ("\"). Для временного отключения от обычных правил интерпретации символов. Обратите внимание, что байт NUL это не то же самое, что значение NULL. NUL - это байт с нулевым значением, а NULL символизирует отсутствие какого-либо значения вообще.

Таблица 1. Управляющие последовательности

Последовательность	Значение
\0	NUL (ASCII 0)

\'	Одинарная кавычка
\"	Двойная кавычка
\b	Backspace
\n	Новая строка
\r	Возврат каретки
\t	Табуляция
\\	Обратная косая черта

В строковых переменных тоже используется шестнадцатеричный формат. Синтаксис шестнадцатеричного формата был показан ранее на примере числовых значений. Но в этом случае шестнадцатеричные цифры интерпретируются как ASCII-коды и преобразуются в символы. Результат будет строкой. Например, при интерпретации 0x616263 получим "abc".

Календарные данные

Дата и время - это значения типа "1999-06-17" или "12:30:43". СУБД MySQL воспринимает также комбинированные значения типа дата/время, такие как "1999-06-17 12:30:43". Обратите внимание, что СУБД MySQL отображает даты в формате год-месяц-день. Этот факт удивляет новичков, начинающих работать с СУБД MySQL, несмотря на то, что этот формат является стандартом ANSI SQL. Отображать дату можно в любом формате. Для этих целей существует функция DATE FORMAT(), но по умолчанию принят формат с предшествующим годом! Поэтому при вводе даты сначала вводится год.

Пустое значение (Null)

Null - это значение, находящееся вне всяких типов. Обычно оно используется для определения значений "нет значения", "неизвестное значение", "вне диапазона", "ничто из вышеперечисленного" и т.д. Пустые значения можно добавлять в таблицы, осуществлять поиск пустых значений по таблицам, проверять, является или не является значение пустым. Арифметические операции с пустыми значениями невозможны. (Такое значение возвращает Null.)

Типы столбцов

В MySQL поддерживаются три основных типа столбцов: числа, строки и значения даты/времени. К первому типу относятся целые числа и числа с плавающей запятой. Строки содержат цепочки текстовых или двоичных символов. Значения даты/времени могут быть представлены в самых разных форматах.

Тип столбца определяет диапазон значений, которые можно заносить в данный столбец. Здесь не допускается та гибкость, которая свойственна переменным. Тип столбца жестко задается при определении таблицы. При записи в ячейку значения другого формата осуществляется автоматическое приведение типа.

Спецификация большинства типов допускает указание размерности в скобках. Это значение определяет максимальное количество символов в представлении значения, не считая знака "минус" и цифр после запятой для чисел. В случае чисел с плавающей запятой можно указать вторую размерность, определяющую количество цифр после запятой. Как правило, размерность не играет никакой роли для числовых столбцов. Более крупные значения принимаются и отображаются правильно. Разве что числа с плавающей запятой могут округляться до нужной разрядности. Размерность важна для строковых столбцов. Слишком длинные строки усекаются в момент вставки в таблицу.

Числа

Числа могут храниться в целом и десятичном форматах, а также в формате с плавающей запятой. Особенностью десятичных чисел является то, что они хранятся в целочисленном виде с указанием точного количества цифр после запятой. Числа с плавающей запятой вычисляются приблизительно, с той или иной точностью.

Целые числа

Далее в таблице перечислены целочисленные типы данных и соответствующие им диапазоны значений. Числа, выходящие за пределы диапазона, преобразуются в минимальное или максимальное значение.

Диапазоны целочисленных типов

Тип	Знаковый диапазон	Беззнаковый диапазон
TINYINT	-127-127	0-255
SMALLINT	-32768-32767	0-65535
MEDIUMINT	-8388608-8388607	0-16777215
INT, INTEGER	-2147483648-2147483647	0-4294967295
BIGINT	-9223372036854775808- 9223372036854775807	0- 18446744073709551615

Синтаксис спецификации целочисленного типа таков:

<тип>[(<размерность>)] [UNSIGNED] [ZEROFILL]

Как минимум, нужно указать имя типа. Следом за ним в скобках приводится требуемая размерность. Флаг UNSIGNED обозначает беззнаковое число. Флаг ZEROFILL говорит о том, что в случае необходимости число должно быть дополнено ведущими нулями до нужной размерности. Применение флага ZEROFILL продемонстрировано далее.

Создание целочисленного столбца с флагом zerofill

```
mysql> CREATE TABLE testint (
-> i INT(11) UNSIGNED ZEROFILL
```

```
-> );
Query OK, 0 rows affected (0.03 sec)
```

```
mysql> INSERT INTO testint VALUES (123);
Query OK, 1 row affected (0.02 sec)
```

```
mysql> SELECT * FROM testint;
+-----+
| i      |
+-----+
| 00000000123 |
+-----+
1 row in set (0.02 sec)
```

Числа с плавающей запятой

Числа плавающей запятой представляют собой приблизительные значения. Они подходят для столбцов, где хранятся дробные величины или числа, выходящие за пределы самого крупного целочисленного диапазона (BIGINT). Чтобы не происходило непредсказуемых погрешностей вычислений, MySQL округляет вставляемые числа до требуемой точности, которая указана в определении столбца, хотя на практике такие погрешности никогда не возникают.

Имеются два типа чисел с плавающей запятой: числа одинарной точности и числа двойной точности. Их диапазоны приведены далее.

Диапазоны целочисленных типов

Тип	Диапазон
FLOAT	$\pm 1,175494351\text{E}-38$ - $\pm 3,402823466\text{E}+38$
DOUBLE, DOUBLE PRECISION, REAL	$\pm 2,2250738585072014\text{E}-308$ - $\pm 1,7976931348623157\text{E}+308$

Синтаксис спецификации типа с плавающей запятой таков:

```
<тип>[ {<размерность>, <точность>}] [ZEROFILL]
```

Точность - это количество цифр после запятой. Флаг ZEROFILL указывает на необходимость дополнения числа ведущими нулями до нужной размерности. Беззнаковые числа с плавающей запятой в MySQL не поддерживаются.

При описании столбца данного типа достаточно указывать одну лишь размерность. Это соответствует спецификации ODBC. Согласно ей, столбец, размерность которого меньше чем 24, будет иметь тип FLOAT, иначе - тип.

Создание столбцов с плавающей запятой

```
mysql> CREATE TABLE testfloat(
-> f FLOAT(20),
-> d FLOAT(40)
-> );
```

Query OK, 0 rows affected (0.00 sec)

```
mysql> DESCRIBE testfloat;
```

Field	Type	Null	Key	Default	Extra
f	float	YES		NULL	
d	double	YES		NULL	

2 rows in set (0.02 sec)

Десятичные числа

Десятичные числа имеют фиксированное количество цифр после запятой. Эти цифры **ВЫЧИСЛЯЮТСЯ** точно, в отличие от чисел с плавающей запятой. Столбцы данного типа удобно использовать для хранения денежных величин, где погрешности округлений недопустимы.

Создание столбца типа DECIMAL демонстрируется в далее. Размерность определяет общее количество цифр числа, включая те, что стоят после запятой. В случае размерности 6 и точности 2 допустимый диапазон значений будет таким: -9999,99 - 9999,99. Беззнаковые десятичные числа не поддерживаются.

Создание десятичного столбца

```
mysql> CREATE TABLE testdec (  
-> m DECIMAL(6,2) ZEROFILL  
-> );
```

Query OK, 0 rows affected (0.00 sec)

```
mysql> INSERT INTO testdec VALUES (123.45);
```

Query OK, 1 row affected (0.00 sec)

```
mysql> SELECT * FROM testdec;
```

m
0123.45

1 row in set (0.00 sec)

Строки

Строки представляют собой цепочки символов и бывают трех типов: ASCII-строки, большие двоичные объекты и перечисления. Длина ASCII-строк ограничена 255-ю символами. Большой двоичный объект (Binary Large Object, BLOB) может содержать до 16 Мбайт текста, а в MySQL версии 4.1 этот предел будет расширен до 4 Гбайт. Перечисления - это строки с предопределенным набором возможных значений.

ASCII-строки

ASCII-строки имеют тип CHAR либо VARCHAR. Описание этих типов вместе с их псевдонимами приведено далее. Столбец типа CHAR имеет фиксированную размерность. Когда значения такого столбца записываются на диск, они дополняются до нужной размерности хвостовыми пробелами, которые автоматически удаляются при извлечении этих значений. По сути, если в такой столбец заносится строка с хвостовыми пробелами, они удаляются.

Столбец типа VARCHAR хранится в базе данных в виде строки переменной длины, а в остальном ведет себя подобно столбцу типа CHAR. Это подразумевает удаление хвостовых пробелов при записи строки в ячейку. В MySQL версии 4.1 в столбцах типа

VARCHAR хвостовые пробелы удаляться не будут. В настоящее время аналогичные функции можно реализовать с помощью типа TINYTEXT.

Типы ASCII-строк

Тип	Диапазон
CHAR, CHARACTER	Строка фиксированной длины
VARCHAR, CHARACTER VARYING	Строка переменной длины

Синтаксис спецификации строкового типа таков:

[NATIONAL] <тип>[{<размерность>}] [BINARY]

Значение размерности определяет максимальную длину строки. Более длинные строки будут усекаться. Если присутствует флаг BINARY, то в операциях сортировки и сравнения значений данного столбца будет учитываться регистр символов. В противном случае регистр не играет роли. Флаг NATIONAL делает доменом столбца набор символов, установленный в системе по умолчанию. Это стандартный режим работы программы MySQL, поэтому данный флаг можно не указывать. Он присутствует для совместимости со стандартом SQL92.

Большие двоичные объекты

Большие двоичные объекты ведут себя подобно строкам переменной длины, но способны хранить огромные информационные массивы. Кроме того, хвостовые пробелы не отсекаются. Столбцы данного вида могут быть либо двоичными, либо текстовыми с дополнительной дифференциацией по размерности. Столбцы семейства BLOB являются двоичными, поэтому в операциях сравнения будет учитываться регистр. Столбцы семейства TEXT являются текстовыми и не чувствительны к регистру. Как и в случае обычных строк, значения, превышающие максимальную длину, усекаются. У столбцов данного типа отсутствуют значения по умолчанию. Подобная возможность появится в MySQL версии 4.1.

Типы больших двоичных объектов

Тип	Максимальная длина
-----	--------------------

TINYBLOB, TINYTEXT	255 байтов
BLOB, TEXT	64 Кбайт (65535 байтов)
MEDIUMBLOB, MEDIUMTEXT	16 Мбайт (16777215 байтов)
LONGBLOB, LONGTEXT	4 Гбайт (4294967295 байтов)

Перечисления и множества

К строковым типам относятся также перечисления и множества. Ячейка типа ENUM (перечисление) в определенный момент времени содержит лишь одно значение из списка возможных. Ячейка типа SET (множество) может содержать несколько уникальных значений из числа тех, что входят в множество.

Перечисления и множества

Тип	Максимальное число значений
ENUM	65535
SET	64

Значение, заносимое в ячейку типа ENUM, указывается в строковом виде. Значение ячейки типа SET записывается в виде строки, содержащей разделенные запятыми элементы множества. Это подразумевает, что сами элементы не содержат запятых.

Далее демонстрируется создание таблицы со столбцами типа ENUM и SET, а также вставка значений в эти столбцы. Строки, которые не указаны в определении перечисления или множества, игнорируются, за исключением пустых строк.

Работа с перечислениями и множествами

```
mysql> CREATE TABLE testenum (
  -> e ENUM ('a', 'b', 'c'),
  -> s SET ('a', 'b', 'c')
  -> );
```

Query OK, 0 rows affected (0.00 sec)

```
mysql> INSERT INTO testenum VALUES ('a', ('b,c,x'));
Query OK, 1 row affected (0.01 sec)
```

```
mysql> INSERT INTO testenum VALUES (2, 5);
Query OK, 1 row affected (0.00 sec)
```

```
mysql> SELECT * FROM testenum;
+-----+-----+
| e   | s   |
+-----+-----+
| a   | b,c |
```

```
| b | a,c |
+-----+-----+
```

2 rows in set (0.00 sec)

Если в качестве значения перечисления указано число, оно считается индексом элемента в определении перечисления. Нумерация элементов начинается с единицы. Ячейки типа SET интерпретируются как битовые поля. Первому элементу множества соответствует младший значащий бит.

Значения даты/времени

В MySQL поддерживаются пять типов столбцов, хранящих информацию о дате/времени. Чаще всего применяются типы DATETIME и TIMESTAMP. Для всех типов значение 0 является допустимым.

Типы больших двоичных объектов

Тип	Диапазон значений
DATE	'1000-01-01'-'9999-12-31'
DATETIME	'1000-01-01 00:00:00' - '9999-12-31 23:59:59'
TIME	'-838:59:59'-'838:59:59'
TIMESTAMP	'1970-01-01 00:00:00' - '2037-12-31 23:59:59'
YEAR	1970-2069 или 1901-2155

Значения даты/времени вполне можно хранить в целочисленных или строковых столбцах, но лишь благодаря специальным типам данных появляется возможность эффективно манипулировать этими значениями. Например, поддерживается операция сложения дат, что продемонстрировано Далее. Для управления такими столбцами на уровне строк или чисел потребовалось бы писать специальные программы. Значения даты/времени могут присутствовать в предложении ORDER BY и их можно сравнивать друг с другом.

Работа со значениями даты/времени

```
mysql> CREATE TABLE testtime (
-> ts TIMESTAMP,
-> dt DATETIME,
-> d DATE,
-> t TIME,
-> y YEAR(2)
-> );
```

Query OK, 0 rows affected (0.00 sec)

```
mysql> INSERT INTO testtime (dt, d, t, y)
-> VALUES ('1989-10-17 17:04:13', 19970322,
-> '10:15:00', 95);
```

Query OK, 1 row affected (0.01 sec)

```
mysql> SELECT * FROM testtime \G
***** 1. row *****
ts: 20040220231813
dt: 1989-10-17 17:04:13
d: 1997-03-22
t: 10:15:00
y: 95
1 row in set (0.00 sec)
```

```
mysql> UPDATE testtime
-> SET y = NOW(),
-> dt = dt + INTERVAL 15 MONTH;
Query OK, 1 row affected (0.01 sec)
Rows matched: 1 Changed: 1 Warnings: 1
```

```
mysql> SELECT * FROM testtime \G
***** 1. row *****
ts: 20040220232023
dt: 1991-01-17 17:04:13
d: 1997-03-22
t: 10:15:00
y: 04
1 row in set (0.00 sec)
```

Значения данного типа являются составными, представляющими собой набор целочисленных значений. У каждого компонента свой диапазон. Например, номер месяца должен находиться в интервале от 1 до 12, номер дня - от 1 до 31, номер часа - от 0 до 23, а число минут и секунд может меняться от 0 до 59.

MySQL не проверяет осмысленность таких дат, как, например, 31 февраля. Разработчики MySQL справедливо полагают, что подобные проверки должны осуществляться не в СУБД, а в прикладных программах.

Допускаются неполные даты наподобие 1970-00-00. Это означает, что дата относится к 1970 году, но месяц и день неизвестны.

Вводимые данные могут быть представлены по-другому, в строковом или числовом виде. Предполагается, что дата записывается как набор целых чисел в следующем порядке: год, месяц, день, час, минуты, секунды. Номер года может состоять из двух или четырех цифр. Поля года, месяца и дня являются обязательными. Отсутствующие поля времени заполняются нулями.

Значение даты, записанное в числовом виде, представляет собой цепочку цифр. Дробная часть числа отбрасывается. В строковом представлении компоненты разделяются знаками пунктуации. Это может быть любой печатный символ кроме буквы и числа. Между номером дня и часа может стоять пробел, по между всеми остальными компонентами пробелы недопустимы. Неправильные даты обнуляются. К двухсимвольным номерам

года в диапазоне 00-69 добавляется значение 2000, а к номерам в диапазоне 70-99 добавляется 1900.

Спецификация типов DATETIME, DATE и TIME не поддерживает понятия размерности. Значения этих типов всегда содержат определенное количество информации. Столбцы типа YEAR могут иметь размерность 2 или 4.

Столбцы типа TIMESTAMP по умолчанию имеют размерность 14.

Допускаются также размерности 12, 8 и 6. Нечетное число преобразуется в ближайшее большее четное. Размерность определяет, какая часть спецификации даты/времени должна отображаться в результатах запроса.

Реальные значения столбца всегда хранят полную спецификацию.

Необычный диапазон значений столбца TIMESTAMP объясняется тем, что в UNIX время измеряется в виде количества секунд прошедших с начала так называемой эпохи (1 января 1970 г.). Это значение хранится в виде четырехбайтового целого числа.

В самую пенимо ячейку типа TIMESTAMP автоматически записывается текущее Время при изменении других ячеек строки. Данная особенность была продемонстрирована в предыдущем примере. Чтобы этого избежать, нужно явно занести в ячейку какое-то значение.

1.3. Создание и удаление таблиц

Таблицы являются основными строительными блоками базы данных. В них содержатся строки и столбцы ваших данных. С помощью команд определения данных SQL можно создавать, удалять и манипулировать таблицами (добавлять, удалять, переставлять столбцы и менять их параметры).

В большинстве реализации SQL таблицей владеет создавший ее пользователь, выдавая разрешения на ее использование другим пользователям. Чтобы создать таблицу, по меньшей мере, надо сделать следующее.

- Задать имя таблицы.
- Задать имена составляющих ее столбцов.
- Определить тип данных для каждого столбца.
- Определить (или использовать заданный по умолчанию) нулевой статус для каждого столбца - допускается или запрещается использование в столбце нулевых значений.
- Значения по умолчанию (автоматически вводимые значения , например текущая дата в соответствующем столбце заказа).
- Ограничения на возможные значения в столбцах
- Ограничения на определение первичных ключей.
- Проверка соответствия между первичными и внешними ключами.

СУБД MySQL позволяет создавать и удалять таблицы, модифицировать их структуру с помощью традиционных SQL-операторов **CREATE TABLE**, **DROP TABLE**. Каждый из этих операторов имеет дополнения, сделанные для диалекта языка SQL, принятого в СУБД MySQL. Это было сделано для повышения их эффективности. Операторы **CREATE TABLE** и **DROP**

TABLE предназначены для создания и удаления индексов уже существующих таблиц.

Оператор **CREATE TABLE**

Создание таблиц осуществляется оператором **CREATE TABLE**. Это, одна из наиболее сложных SQL-инструкций. Общий формат ее таков:

```
CREATE [TEMPORARY] TABLE [IF NOT EXISTS] имя_таблицы  
[(спецификация, ...)]  
[опция...]  
[[IGNORE | REPLACE] запрос]
```

Флаг *TEMPORARY* задает создание временной таблицы, существующей в течение текущего сеанса. По завершении сеанса таблица удаляется.

Временным таблицам можно присваивать имена других таблиц, делая последние временно недоступными.

Спецификатор **IF NOT EXISTS** подавляет вывод сообщений об ошибках в случае, если таблица с указанным именем уже существует. Имени таблицы может предшествовать имя базы данных, отделенное точкой. Если это не сделано, таблица будет создана в базе данных, которая установлена по умолчанию.

Чтобы задать имя таблицы с пробелами, необходимо заключить его в обратные кавычки, например **'player list'**. То же самое нужно будет делать во всех ссылках на таблицу, поскольку пробелы используются для разделения идентификаторов.

Разрешается создавать таблицы без столбцов, однако в большинстве случаев спецификация хотя бы одного столбца все же присутствует. Спецификации столбцов и индексов приводятся в круглых скобках и разделяются запятыми. Формат спецификации следующий:

```
имя тип  
[NOT NULL | NULL]  
[DEFAULT значение]  
[AUTO_INCREMENT]  
[PRIMARY KEY]  
[ссылка]
```

Типы столбцов рассматривались на предыдущей странице. Спецификация типа включает название типа и его размерность.

По умолчанию столбцы принимают значения **NULL**. Спецификатор **NOT NULL** запрещает подобное поведение.

У любого столбца есть значение по умолчанию. Если оно не указано, программа MySQL выберет его самостоятельно. Для столбцов, принимающих значения **NULL**, значением по умолчанию будет **NULL**, для строковых столбцов - пустая строка, для целочисленных столбцов - нуль. Изменить эту установку позволяет предложение **DEFAULT**.

Поля-счетчики, создаваемые с помощью флага *AUTOINCREMENT*, игнорируют значения по умолчанию, так как в них записываются порядковые номера. Тип счетчика должен быть беззнаковым целым. В

таблице может присутствовать лишь одно поле-счетчик. Им не обязательно является первичный ключ.

Пример создания таблицы:

Создание таблицы

```
CREATE TABLE IF NOT EXISTS player (  
    /*Столбцы*/  
    ID INT(11) UNSIGNED NOT NULL AUTO_INCREMENT,  
    Nickname CHAR(8) NOT NULL,  
    Password CHAR(8) NOT NULL,  
    Rank FLOAT(4, 2) NULL DEFAULT 50.0,  
    Region ENUM('North', 'South', 'East', 'West') NOT NULL,  
    Team TINYINT(3) UNSIGNED NOT NULL,  
  
    /*Индексы*/  
    PRIMARY KEY (ID),  
    INDEX Region, Rank),  
    FOREIGN KEY (Team) REFERENCES team (ID)  
)
```

Таблицы типа **MyISAM** и **InnoDB** следят за тем, какие значения генерируются счетчиком, поэтому в случае удаления какой-либо записи ее порядковый номер не будет использован повторно. В таблицах других типов порядковый номер вычисляется путем прибавления единицы к максимальному значению столбца. Если из таблицы удалить все записи, нумерация начнется с единицы.

Чтобы активизировать работу счетчика, необходимо вставить в таблицу строку, в которой соответствующее поле равно 0 или NULL.

Спецификация **PRIMARY KEY** позволяет назначить столбец первичным ключом. При этом для столбца будет создан индекс.

В конце определения столбца может стоять предложение **REFERENCES**, синтаксис которого такой же, как и в описанном ниже предложении **FOREIGN KEY**, но подобный тип ограничений не поддерживается в MySQL.

В круглых скобках задаются спецификации не только столбцов, но также ограничений и индексов.

[CONSTRAINT имя] FOREIGN KEY имя (столбец, ...) [ссылка]

Эта спецификация тоже не играет никакой роли в MySQL, существуя лишь в целях внешней совместимости. Имеет смысл включать ее в инструкцию **CREATE TABLE** в качестве "документации" к схеме базы данных. В принципе, предложение **FOREIGN KEY** помечает группу столбцов как зависящую от набора столбцов другой таблицы, формируя связь между таблицами, но в MySQL правильность этой связи не контролируется.

Спецификация ссылки имеет следующий вид:

REFERENCES таблица [(столбец, ...)] [MATCH FULL | MATCH PARTIAL]

[ON DELETE правило] [ON UPDATE правило]

Синтаксис правил удаления и обновления таков:

**{RESTRICT | CASCADE | SET NULL | NO ACTION | SET DEFAULT}
FULLTEXT [INDEX] [имя] (столбец [(длина)], ...)**

Эта спецификация задает полнотекстовый индекс для указанных столбцов.

{KEY | INDEX} [имя] (столбец [(длина)], ...)

Эта спецификация задает индекс для указанных столбцов.

PRIMARY KEY (столбец [(длина)],...)

Эта спецификация делает группу из одного или нескольких столбцов первичным ключом. Первичный ключ не может содержать дублирующиеся значения и значения NULL. У каждой таблицы есть не более чем один первичный ключ.

UNIQUE [INDEX] [имя] (столбец [(длина)], ...)

Эта спецификация накладывает на группу столбцов ограничение уникальности.

Табличные опции

После спецификаций столбцов и индексов может стоять произвольное число опций. Они разделяются пробелами, а не запятыми. Далее демонстрируется задание четырех опций.

Задание табличных опций

```
CREATE TABLE team (  
    /* Столбцы */  
    ID TINYINT(3) UNSIGNED NOT NULL AUTO_INCREMENT,  
    Name CHAR(16) NOT NULL,  
  
    /* Индексы */  
    PRIMARY KEY (ID)  
)
```

```
TYPE=MYISAM  
AUTO_INCREMENT=100  
AVG_ROW_LENGTH=9 MAX_ROWS=20
```

TYPE = тип

Опция *TYPE* задает формат хранения таблицы. По умолчанию таблицы имеют тип MyISAM. Для таких таблиц поддерживается большинство функциональных возможностей, за исключением транзакций. Типы HEAP, ISAM, MERGE и MyISAM доступны всегда. Поддержка остальных типов включается на этапе компиляции.

Типы таблиц

Тип	Описание
<i>BDB</i>	Таблицы этого типа поддерживают транзакции благодаря библиотеке функций Berkeley DB
<i>Berkeley_db</i>	Синоним типа BDB
<i>HEAP</i>	Таблицы этого типа хранятся в памяти

<i>InnoDB</i>	Таблицы этого типа поддерживают транзакции благодаря библиотеке функций Innobase
<i>ISAM</i>	Этот формат использовался старыми версиями MySQL
<i>MERGE</i>	Это коллекция таблиц MyISAM, интерпретируемых как одно целое
<i>MYISAM</i>	Это стандартный тип таблиц

***AUTO_INCREMENT* = начальное_значение**

Эта опция задает начальное значение поля-счетчика. Она доступна лишь для таблиц типа My ISAM.

***AVG_ROW_LENGTH* = длина**

Эта опция помогает программе MySQL создавать указатели записей. Комбинация опций *AVG_ROW_LENGTH* и *MAXROWS* определяет объем дискового пространства, занимаемый таблицей, и, следовательно, длину указателей.

***CHECKSUM* = {0 | 1}**

Если эта опция включена, программа MySQL будет хранить контрольную сумму каждой записи. Это помогает осуществлять контроль ошибок с помощью инструкции *CHECK TABLE*. По умолчанию опция отключена. Кроме того, она доступна только для таблиц типа MyISAM.

***COMMENT* = комментарий**

С помощью этой опции таблицу можно снабдить комментарием. Длина комментария не должна превышать 60 символов, но в MySQL версии 4.0 предел возрастет до 255 символов.

***MAX_ROWS* = число_строк**

Эта опция сообщает программе MySQL о максимальном числе записей, которое планируется хранить в таблице. Это не жесткий предел, а лишь подсказка, на основании которой программа выделяет таблице дисковое пространство. Чем меньше данное значение, тем короче будут указатели записей, что ускорит их поиск.

***MIN_ROWS* = число_строк**

Эта опция задает предполагаемое минимальное число записей, которое планируется хранить в таблице.

***PACK_KEYS* = {0 | 1}**

Если эта опция включена, программа MySQL будет осуществлять сжатие всех индексов. Обычно сжимаются только индексы столбцов CHAR и VARCHAR, когда их размерность начинает превышать 8 байтов. Это позволяет экономить дисковое пространство при наличии в столбцах большого количества дублирующихся значений.

***PASSWORD* = пароль**

Пароль используется для шифрования определения таблицы, но в открытых версиях MySQL эта возможность не поддерживается. Она доступна лишь в коммерческих версиях программы.

DELAY_KEY_WRITE = {0|1}

Эта опция заставляет программу MySQL откладывать обновление индексов до того момента, когда таблица будет закрыта. По умолчанию ключи обновляются при каждом изменении. Данная опция доступна только для таблиц типа MyISAM.

ROW_FORMAT = DEFAULT | DYNAMIC | STATIC | COMPRESSED

Эта опция задает формат хранения.

RAID_TYPE = {1 | STRIPED | RAID0}

RAID_CHUNKS = число_блоков

RAID_CHUNKSIZE = размер

Опции семейства *RAID* задаются все вместе. Поддержка функций RAID должна быть включена на этапе компиляции. В версии 3.23.36 поддерживается лишь тип *STRIPED*, а остальные два типа определены как его синонимы.

Таблица, для которой установлены эти опции, будет распределена на несколько файлов. Опция *RAIDCHUNKS* указывает число файлов. Опция *RAID_CHUNKSIZE* задает размер каждого файла в килобайтах. Когда первый файл заполняется, создается следующий файл. Такая технология позволяла обходить существовавшие когда-то ограничения на размеры файлов, что не актуально в современных операционных системах.

UNION = (таблица, ...)

Опция UNION задает слияние перечисленных таблиц.

Оператор DROP TABLE

Инструкция **DROP TABLE** удаляет все файлы, относящиеся к таблице. Она имеет следующий синтаксис:

DROP TABLE [IF EXISTS] таблица[,таблица...] [RESTRICT | CASCADE]

Спецификация **IF EXISTS** подавляет вывод сообщения об ошибке, выдаваемого в случае, если заданная таблица не существует. Можно указывать несколько имен таблиц, разделяя их запятыми.

Флаги *RESTRICT* и *CASCADE* игнорируются в MySQL версии 3.23. Они предназначены для выполнения сценариев, созданных в других СУБД. Далее демонстрируется удаление двух таблиц: message и log.

Удаление таблиц

DROP TABLE IF EXISTS message, log

Пример создания таблицы

CREATE TABLE Блюда

```
(    БЛ      SMALLINT NOT NULL,
    Блюда   CHAR (70) NOT NULL,
    В       CHAR (1),
    Основа  CHAR (10),
    Выход   FLOAT,
    Труд    SMALLINT );
```

Давайте изучим синтаксис этого оператора :

- Первое предложение **-CREATE TABLE-** не вызывает особых затруднений, если, конечно выбранное имя таблицы соответствует

правилам записи идентификаторов в вашей системе.

- Вторая строка начинается с открывающейся круглой скобки, ввод которой обязателен.
- Затем задайте имя первого столбца и через пробел укажите его тип данных.(В некоторых базах данных вы должны определить длину, размер или точность данных- обычно в виде целого числа в скобках,следующих сразу за описанием типа данных.)
- Ключевые слова NULL и NOT NULL , заключенные в квадратные скобки необязательны. Вертикальная черта между ними означает , что можно использовать только одну из этих опций. а не сразу две. Если вы ничего не выберете, система сделает это за вас. По умолчанию SQL-92 разрешает нулевые значения, но по этому пути следует не все коммерческие версии. Возьмите за привычку всегда явно указывать одну из этих опций, даже если вас устраивают значения, выбираемые по умолчанию. В будущем это может существенно упростить вашу работу.
- Третья строка , заключенная в квадратные скобки и с многоточием в конце , повторяет определение столбца. Квадратные скобки указывают ,что определять второй столбец необязательно, а многоточие означает , что таким образом можно определить любое необходимое число столбцов(в рамках ограничений системы - обратитесь к справочному руководству по своей СУБД).
- Обратите внимание, что каждое определение столбца отделяется от предыдущего определения запятой, а также не забудьте о закрывающейся круглой скобке после списка столбцов.

В результате создается пустая базовая таблица Блюда, а в системный каталог помещается строка, описывающая эту таблицу. Отметим, что в профессиональных СУБД имя таблицы дополняется именем пользователя, который издал предложение CREATE TABLE. Если этот пользователь зарегистрирован в системе под именем Kirillov, то в каталоге будет зарегистрирована таблица Kirillov.Блюда и указанный пользователь может обращаться к ней по имени Kirillov.Блюда или по сокращенному имени Блюда..

Чтобы создать таблицу надо:

После того, как вы подсоединились с MySQL-сервером и увидели список активных баз данных вы должны сделать следующее:

1. Щелкните на той базе данных имя которой совпадает с <compXX>, так чтобы слева от нее открылась папка.
2. Далее нажмите правой кнопкой мыши на ее надписи и в выпадающем меню выберите 'SQL Query'.

Появится окно с тремя закладками - Query, Results, Status. Вам понадобятся только первые две закладки. Первая (Query) служит для ввода SQL-запросов, а вторая (Results) для просмотра результатов исполнения SQL-запросов. ВНИМАНИЕ! Везде, где написано istXX, вам надо подставить вместо XX номер своей группы.

- Введите в окно ввода: (на закладке Query)

CREATE TABLE istXX(Name TEXT, Surname TEXT, Telephone TEXT, Age INT)

и нажмите кнопку выполнения запроса (она выглядит в виде зеленой стрелки вправо).

Далее компьютер должен автоматически переключить вас на вкладку Results, где вы сможете посмотреть результат выполнения вашего запроса.

Для создания базы данных и таблиц в ней необходимо подключиться к серверу. Для этого воспользуемся клиентской программой **mysql.exe**.

2. выполнить:

Для подключения к удаленному серверу необходимо указать его адрес с помощью опции **-h host_name**, где **host_name** = IP - адрес сервера (всем подключаться к - 192.168.XX.XX), имя пользователя - опцией **-u user_name**, где **user_name** - ваш уникальный идентификатор (Например – Xxist091), пароль - опцией **-p** (пароль после опции вводить не надо, он будет запрошен у вас позже).

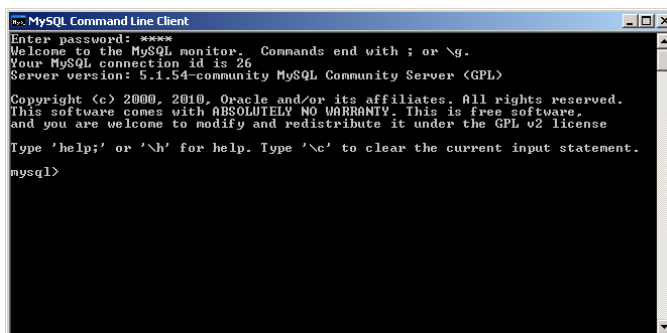
Пример соединения с сервером:

Перед вводом кода необходимо войти в директорию с установленным MySQL, (обычно C:\MySQL\Bin\).

```
mysql -u stud -p -h 192.168.14.146
```

```
Enter password: 1111
```

В результате вы присоединитесь к серверу и перед вами появится следующее:



После этого необходимо открыть свою базу данных:

```
use XXist091
```

Выборка базы данных

Там уже будет создана ваша база данных (*XXistGG*, где *XX* - ваш номер, а *GG* - номер вашей группы), которую вы создали на предыдущем практическом занятии.

Создать таблицы: **UserProfile, BookShop, MusicShop, products**.

UserProfile - содержит информацию о пользователе.

BookShop - содержит описание имеющихся в продаже книг.

MusicShop - содержит описания имеющихся в продаже музыкальных произведений.

products - содержит информацию о товарах

Описание таблиц:

Таблица UserProfile

Название поля	Описание
Fname	Имя пользователя
Lname	Фамилия пользователя
Userid	Уникальный ID пользователя
Password	Зашифрованный пароль пользователя
Address	Адрес
City	Город
ZipCode	Почтовый код
Gender	Пол
Age	Возраст
Emailed	Адрес электронной почты
PhoneNumber	Номер телефона

Таблица BookShop

Название поля	Описание
ItemNo	Уникальный ID книги
ItemType	Тип книги
Title	Название книги
Author	Автор книги
Price	Цена книги

Таблица MusicShop

Название поля	Описание
ItemNo	Уникальный идентификатор музыкального альбома
ItemType	CD/кассета
Title	Название альбома
Artist	Исполнитель
Price	Цена альбома

Типы столбцов вы должны указать сами.

Пример создания таблицы **BookShop**:

```
CREATE TABLE BookShop (
  itemNo Varchar(20) NOT NULL,
  itemType Varchar(20) NOT NULL,
  title Varchar(60) NOT NULL,
  author Varchar(60) NOT NULL,
  price FLOAT NOT NULL );
```

Для выполнения практического задания следующей лекции необходимо создать таблицу products на сервере 192.168.16.90. Ее структура представлена на скриншоте ниже

```
mysql> describe products;
```

Field	Type	Null	Key	Default	Extra
id_product	int(11)	YES		NULL	
name	varchar(60)	YES		NULL	
price	decimal(7,2)	YES		NULL	
count	int(11)	YES		NULL	
mark	float(4,1)	YES		NULL	
description	text	YES		NULL	
id_catalog	int(11)	YES		NULL	

```
7 rows in set (0.01 sec)
```

Чтобы просмотреть созданные таблицы наберите команду **show tables;** .

Чтобы просмотреть структуру созданной таблицы введите **describe имя_таблицы;**

Совет

При работе с программным кодом вам понадобится следующая команда \c, которая позволяет в случае необходимости очистить окно команд. Например, если при вводе программного кода вы допустили ошибку, а уже начали набирать другую строку, вам необходимо набрать эту команду. После этого, чтобы не вводить программный код заново при помощи стрелочек вверх/вниз выбирать нужные команды и нажимать Enter

Вы создали свою базу данных, добавили в нее три таблицы. Созданная БД будет использована при дальнейшем изучении MySQL.

Отчет :

- Выполненные задания сдайте преподавателю.

Лабораторная работа № 5. Модификация структуры таблицы

Работа с базой данных предполагает несколько этапов.

- Создание (инициализация) базы данных.
- Создание таблиц в базе данных.
- Взаимодействие с таблицами посредством операций вставки, выборки, модификации или удаления данных.

Выборка существующих данных фактически является самой частой операцией при работе с базой данных. В числе других наиболее частых операций над таблицами базы данных являются вставка новых данных, их модификация и удаление.

Начнем с самого начала. Поэтому нашей первой операцией будет наименее часто выполняемая операция - создание базы данных. Затем создание таблиц, загрузка начальных данных, а уже потом наиболее популярная в базах данных операция - выборка.

Вы научитесь:

- открывать базы данных на сервере;
- создавать таблицы в базе данных.;
- модифицировать структуру таблиц;
- просматривать свойства столбцов таблицы;
- создавать первичные ключи.

Начало занятия:

1. Изучить учебный материал:

1.1. Модификация структуры таблицы

Оператор **ALTER TABLE** в СУБД MySQL обладает достаточно универсальными возможностями.

Оператор **ALTER TABLE** может пригодиться в том случае, когда становится очевидно, что старая структура таблицы уже не соответствует текущим задачам. Появилась какая-то дополнительная информация, или таблица содержит данные, ставшие уже избыточными. Существующие столбцы оказались слишком малы, или опыт работы с таблицей показал, что они непредусмотрительно были объявлены слишком большими и для экономии пространства и оптимизации запросов их требуется уменьшить. Возможно, просто была допущена ошибка в имени столбца при создании таблицы с помощью оператора **CREATE TABLE**. Вот несколько таких ситуаций.

- Вы ведете на Web-сервере анкету и сохраняете результаты каждого опроса как запись таблицы. Затем вы решили модифицировать анкету и добавить в нее несколько вопросов. Для этого необходимо добавить столбцы в таблицу для того, чтобы сохранять новые вопросы.
- Предположим, что вы участвуете в научном проекте. Каждому эксперименту вы присваиваете номер и сохраняете его в столбце с параметром **AUTOINCREMENT**. По всем оценкам, исследования должны длиться достаточно долго. Количество регистрирующих записей должно составлять около 50000. Вы совершенно справедливо приняли решение для хранения номеров опытов выбрать тип

UNSIGNED SMALLINT, который может хранить до 65535 значений. Однако по полученным результатам проект был признан успешным и его финансирование возобновилось. По предварительным оценкам может быть сделано еще 50000 записей. Необходимо изменить имеющийся тип столбца на более вместительный.

- Размеры столбцов могут меняться и в другую сторону. Предположим, был создан столбец типа CHAR (255), но в процессе работы с таблицей оказалось, что длина значений в этом столбце не превышает 100 символов. Длину столбца можно уменьшить.

Синтаксис оператора **ALTER TABLE** имеет следующий вид:

Инструкция **ALTER TABLE** позволяет менять определение таблицы. Для этого необходимо иметь привилегии **ALTER**, **CREATE** и **INSERT**. Общий ее формат таков:

ALTER [IGNORE] TABLE имя спецификация[, спецификация ...]

Поскольку программа MySQL способна вносить незаметные изменения в определения таблиц, существует вероятность того, что инструкция **ALTER** не возымеет никакого эффекта.

В качестве параметров инструкция *ALTER TABLE* принимает имя таблицы и как минимум одну спецификацию изменений. Спецификации отделяются друг от друга запятыми, и подобная форма записи является расширением стандарта языка SQL.

Флаг **IGNORE** заставляет программу MySQL игнорировать дубликаты, если данные, уже находящиеся в таблице, конфликтуют с ее новым определением. Например, когда столбец, содержащий дублирующиеся данные, объявляется первичным ключом, то по умолчанию программа отказывается вносить изменения. При наличии флага *IGNORE* изменения будут учтены, а все дублирующиеся записи, кроме одной, - выброшены.

В случае добавления нового столбца соответствующие ячейки существующих записей будут заданы равными *NULL*, если столбец это допускает, или же в них будут записаны значения по умолчанию.

Ниже описаны все возможные варианты спецификаций.

ADD [COLUMN] определение [FIRST | AFTER столбец]

С помощью этой спецификации к таблице добавляется новый столбец.

Формат определения столбца должен быть таким же, как и в инструкции *CREATE TABLE*. По умолчанию столбец добавляется в конец списка, но с помощью ключевого слова **FIRST** его можно объявить первым, а с помощью предложения *AFTER* - стоящим после заданного столбца.

Далее демонстрируется добавление двух столбцов, располагаемых в определенном порядке. Возможно, эти столбцы не были учтены при создании таблицы.

Добавление столбцов в заданном порядке

```
mysql> ALTER TABLE user
```

```
-> ADD middleName VARCHAR(32) AFTER username,
```

```
-> ADD prefix VARCHAR(32) FIRST;
```

Другая причина добавления столбцов - введение первичного ключа. Так, далее в таблицу вставляется столбец-счетчик. В результате каждая строка получит уникальный идентификатор в поле *ID*.

Добавление первичного ключа со счетчиком

```
mysql> ALTER TABLE user  
-> ADD ID INT NOT NULL AUTO_INCREMENT PRIMARY KEY FIRST;
```

ADD [COLUMN] (определение, определение, ...)

С помощью этой спецификации в конец списка столбцов добавляется группа новых столбцов. Определения столбцов разделяются запятыми.

Модификация первичного ключа

```
mysql> ALTER TABLE address  
  
-> MODIFY ID INT NOT NULL AUTO_INCREMENT PRIMARY KEY  
FIRST;
```

Добавление группы столбцов

```
mysql> ALTER TABLE user ADD (  
-> middleName VARCHAR(32),  
-> prefix VARCHAR(32)  
-> );
```

ADD [CONSTRAINT имя] FOREIGN KEY имя (столбец, ...) ссылка

Эта спецификация существует для совместимости с другими СУБД. Факт существования столбцов не проверяется, а информация об ограничении не сохраняется в таблице. Разработчики MySQL планируют добавить функции хранения внешних ключей в версию 4.0.

ADD FULLTEXT [имя] (столбец, ...)

Эта спецификация предназначена для добавления к указанным столбцам полнотекстового индекса.

ADD {KEY | INDEX} [имя] (столбец, ...)

Эта спецификация позволяет добавить индекс к одному или нескольким столбцам. Ключевые слова *KEY* и *INDEX* являются синонимами.

Аналогичную функцию выполняет также инструкция *CREATE INDEX*.

Добавление индекса

```
mysql> ALTER TABLE user  
-> ADD INDEX (username);
```

ADD PRIMARY KEY (столбец, ...)

С помощью этой спецификации к таблице добавляется первичный ключ. У таблицы может быть только один такой ключ, поэтому существующий ключ необходимо предварительно удалить.

Добавление первичного ключа

```
mysql> ALTER TABLE address  
-> ADD PRIMARY KEY(ID);
```

ADD UNIQUE [имя] (столбец, ...)

Эта спецификация налагает на заданные столбцы ограничение уникальности. Если в столбцах содержатся дублирующиеся значения, инструкция *ALTER*

завершится неудачей. Воспользуйтесь флагом *IGNORE*, чтобы вызвать принудительное изменение.

ALTER [COLUMN] столбец DROP DEFAULT

С помощью этой спецификации из определения столбца удаляется описание значения по умолчанию. Для столбца будет выбрано новое стандартное значение на основании его типа и допустимости значений *NULL*. Если значения *NULL* разрешены, выбор будет сделан в их пользу. В противном случае будет выбрано значение 0 или пустая строка.

Отмена значений по умолчанию

```
mysql> ALTER TABLE address  
-> ALTER prefix DROP DEFAULT;
```

ALTER [COLUMN] столбец SET DEFAULT литерал

Эта спецификация назначает столбцу значение по умолчанию. Подобное изменение не затрагивает существующие записи.

Задание значения по умолчанию

CHANGE [COLUMN] столбец определение

С помощью этой спецификации меняется определение столбца: его имя, размерность и тип. Подобная возможность не предусмотрена в стандарте языка SQL. Другие СУБД тоже допускают модификацию определений столбцов, но программа MySQL является наиболее гибкой в этом плане. MySQL пытается привести существующие данные к новому типу. Если столбец проиндексирован, его размерность не может стать меньше, чем размерность индекса. Например, наличие индекса первых 16 символов столбца типа *VARCHAR* означает, что размерность столбца тоже должна составлять не менее 16 символов.

Далее показано, как тип столбца меняется с *VARCHAR* (32) на *ENUM*.

Обратите внимание на то, что имя столбца повторяется дважды. Это не ошибка. Первый раз идентифицируется существующий столбец, а второй раз дается его новое определение.

Изменение определения столбца

```
mysql> ALTER TABLE address  
-> CHANGE prefix  
-> prefix ENUM('Mr.', 'Mrs.', 'Miss', 'Ms');
```

DROP [COLUMN] столбец

Эта спецификация предназначена для удаления столбца из таблицы и не является частью стандарта языка SQL. Индексы, охватывающие удаляемый столбец, автоматически перестраиваются. Если в индекс входил только один этот столбец, индекс удаляется.

Удаление столбца

```
mysql> ALTER TABLE address  
-> DROP middleName;
```

DROP PRIMARY KEY

Эта спецификация аналогична предыдущей, но удаляется не произвольный столбец, а лишь первичный ключ. Если для таблицы не задан первичный ключ, удаляется первый уникальный индекс.

DROP INDEX индекс

С помощью этой спецификации удаляется указанный индекс. Подобная возможность не предусмотрена в стандарте.

Удаление индекса

```
mysql> ALTER TABLE address  
-> DROP INDEX lastName;
```

MODIFY [COLUMN] определение

Эта спецификация служит для изменения определения столбца, кроме его имени. Данная возможность появилась в MySQL под влиянием СУБД Oracle.

Модификация определения столбца

```
mysql> ALTER TABLE address  
-> MODIFY prefix ENUM('Mr.','Mrs.','Miss1','Ms');
```

ORDER BY столбец

Данная спецификация предназначена для изменения физического порядка записей по значениям заданного столбца. В некоторых случаях это позволяет ускорить выполнение запросов к таблице. Обратите внимание на то, что переупорядочение записей происходит лишь один раз. Последующие операции вставки и удаления приведут к изменению порядка записей.

RENAME [TO] имя

Эта спецификация позволяет менять имя таблицы. Аналогичную функцию выполняет инструкция RENAME TABLE.

Изменение имени таблицы

```
mysql> ALTER TABLE address  
-> RENAME addr;
```

1.2. Получение информации о базах данных и таблицах

Познакомимся с несколькими операторами, позволяющими получить информацию о базах данных и таблицах. Они полезны для отслеживания структуры базы данных, особенно для предварительного просмотра свойств столбцов таблицы перед внесением изменений в их структуру с помощью оператора *ALTER TABLE*.

Для получения самой разноплановой информации о базе данных и таблицах можно воспользоваться оператором SHOW.

SHOW DATABASES	Перечень баз данных на сервере
SHOW TABLES	Перечень таблиц в текущей базе данных
SHOW TABLES FROM имя_базы	Перечень таблиц в указанной базе данных
SHOW COLUMNS FROM имя_таблицы	Отображение информации о столбцах в указанной таблице
SHOW INDEX FROM имя_таблицы	Отображение информации об индексах в указанной таблице
SHOW TABLE STATUS	Отображение описательной информации о

	таблицах в текущей базе данных
SHOW TABLE STATUS FROM имя_базы	Отображение описательной информации о таблицах в указанной базе данных

Операторы **DESCRIBE имя_таблицы** и **EXPLAIN имя_таблицы** являются синонимами оператора **SHOW COLUMNS FROM имя_таблицы**.

Синтаксис приведенных выше операторов:

DESCRIBE

Инструкция **DESCRIBE** возвращает таблицу, содержащую описание одного или нескольких столбцов заданной таблицы. Общий формат инструкции таков:

{DESCRIBE | DESC} таблица [столбец | условие_отбора)

Для ленивых: вместо слова **DESCRIBE** можно указывать его сокращенную форму **DESC**. В простейшем случае задается только таблица. В результате будет выдано описание каждого ее столбца. Имени таблицы может предшествовать имя базы данных. Допускается указывать конкретный столбец или шаблон имени. Аналогичные результаты выдает инструкция **SHOW COLUMNS**. Далее приведены результаты инструкции **DESCRIBE**. В первых двух колонках указаны имя и тип каждого столбца. В колонке "Null" будет стоять **YES**, если столбец допускает значения **NULL**. Для столбцов, являющихся частью первичного ключа, в четвертой колонке будет стоять **PRI**. Если же столбец входит в состав другого индекса, то в этой колонке будет указано **NUL**. В пятой колонке отображается значение по умолчанию. В последней колонке приводится дополнительная информация о столбце. В частности, здесь указывается, является ли столбец полем-счетчиком.

Инструкция DESCRIBE

mysql> **DESCRIBE mysql.user;**

Field	Type	Null	Key	Default	Extra
Host	char(60) binary		PRI		
User	char(16) binary		PRI		
password	char(16)				
Select_priv	enum('N','Y')			N	
Insert_priv	enum('N','Y')			N	
Update_priv	enum('N','Y')			N	
Delete_priv	enum('N','Y')			N	
Create_priv	enum('N','Y')			N	
Drop_priv	enum('N','Y')			N	
Reload_priv	enum('N','Y')			N	
Shutdown_priv	enum('N','Y')			N	
Process_priv	enum('N','Y')			N	
File_priv	enum('N','Y')			N	
Grant_priv	enum('N','Y')			N	
References_priv	enum('N','Y')			N	

Index_priv	enum('N','Y')			N		
Alter_priv	enum('N','Y')			N		

17 rows in set (0.05 sec)

EXPLAIN

Инструкция *EXPLAIN* описывает способ выполнения указанного запроса или же эмулирует инструкцию *DESCRIBE* для заданной таблицы:

EXPLAIN {запрос | таблица}

В качестве запроса разрешается вводить любую допустимую инструкцию *SELECT*. В результате будет выдана таблица с описанием индексов, используемых для выполнения операции объединения. Благодаря этому можно узнать, влияют ли индексы на повышение производительности объединений. Далее показаны типичные результаты, выдаваемые инструкцией *EXPLAIN*.

Инструкция EXPLAIN

```
mysql> EXPLAIN
```

```
-> SELECT *
-> FROM mysql.user, mysql.db
-> WHERE mysql.user.host = mysql.db.host;
```

table	type	possible_keys	key	key_len	ref	rows	Extra
user	ALL	NULL	NULL	NULL	NULL	5	
db	ALL	NULL	NULL	NULL	NULL	2	where used

2 rows in set (0.06 sec)

Таблица, возвращаемая инструкцией *EXPLAIN*, содержит одну строку для каждой таблицы в порядке их упоминания. В первой колонке приводится имя таблицы, во второй - тип объединения. Возможны следующие типы: *ALL*, *const*, *eq_ref*, *index*, *range*, *ref*, *system*.

Тип *ALL* означает полное сканирование таблицы. Обычно для такой таблицы требуется индекс.

Тип *const* означает, что в таблице есть одна совпадающая запись, а тип *system* указывает на таблицу, состоящую из одной записи. В обоих случаях программа MySQL может хранить запись в памяти в течение всего времени выполнения запроса, что приведет к повышению производительности.

Тип *eq_ref* означает ссылку по равенству, т.е. для каждой комбинации записей предыдущих таблиц из данной таблицы извлекается одна запись. В большинстве случаев это наилучший тип с точки зрения производительности.

Тип *index* означает, что сканированию подлежит весь индекс. Это почти так же плохо, как и тип *ALL*, но чуть быстрее, поскольку количество элементов индекса обычно меньше, чем число записей в таблице.

Тип *range* указывает на то, что условию отбора может удовлетворять подмножество записей и каждую из них придется сканировать на каждом проходе.

Тип *ref* означает, что условию отбора будет удовлетворять подмножество записей, но для ссылки на них используется индекс. Это имеет место в том случае, когда адресуется индексируемый столбец, допускающий дублирующиеся значения.

В колонке *possible_keys* перечислены индексы, которые могут быть использованы для повышения эффективности запроса. Здесь может стоять *NULL*.

В колонке *key* показаны задействованные индексы. Здесь тоже может стоять *NULL*. В колонке *key_len* сообщается о том, сколько частей составного ключа используется.

В колонке *ref* приводится список столбцов или констант, с помощью которых осуществлялся отбор записей. В колонке *rows* отображается число записей, участвующих в запросе.

В колонке *Extra* приводится дополнительная информация об объединении. Ключевое слово *Distinct* говорит о том, что *MySQL* перестает сканировать записи при обнаружении первого совпадения. Сообщение **Not exists** означает, что программа оптимизировала левое внешнее объединение, пропустив последующие операции сканирования для совпавшей комбинации. Сообщение **range checked for each record** выдается, когда ни один из индексов не устроил программу полностью, т.е. на каждом этапе сканирования используются разные индексы. Сообщение **Using filesort** указывает на необходимость дополнительного прохода для сортировки таблицы результатов. Сообщение **Using index** означает, что информация о столбцах извлекалась только из индексов, т.е. не понадобилось искать сами записи. Сообщение **Using temporary** говорит о необходимости создания временной таблицы. Если же в колонке *Extra* содержится сообщение **Where used**, значит, для отбора записей было использовано предложение *WHERE*.

SHOW DATABASES

Инструкция *SHOW DATABASES* возвращает список баз данных, существующих на сервере:

SHOW DATABASES [LIKE шаблон]

Пример инструкции показан далее. В предложении *LIKE* задается шаблон имен баз данных.

Вывод списка баз данных

```
mysql> SHOW DATABASES LIKE '%mysql';
+-----+
| Database (%mysql) |
+-----+
| mysql             |
+-----+
1 row in set (0.00 sec)
```

SHOW TABLES

Инструкция *SHOW TABLES* возвращает список таблиц, существующих в указанной базе данных:

SHOW [OPEN] TABLES [FROM база_данных] [LIKE шаблон]

По умолчанию отображаются все таблицы текущей базы данных. Флаг *OPEN* ограничивает список только открытыми таблицами. В этом случае во втором столбце указывается, сколько раз таблица кэшировалась и использовалась.

Получение списка открытых таблиц

```
mysql> SHOW OPEN TABLES FROM mysql;
```

```
+-----+-----+
| Open_tables_in_mysql | Comment          |
+-----+-----+
| user                 | cached=1, in_use=0 |
| db                   | cached=1, in_use=0 |
+-----+-----+
```

2 rows in set (0.00 sec)

SHOW TABLE STATUS

Инструкция *SHOW TABLE STATUS* выводит статусную информацию о таблицах. Ее синтаксис таков:

SHOW TABLE STATUS [FROM база_данных] [LIKE шаблон]

В предложении *LIKE* задается шаблон имен таблиц. Список столбцов возвращаемой таблицы приведен в таблице. Описание каждой таблицы базы данных занимает одну строку.

Статусная информация о таблицах

Столбец	Описание
Auto_increment	Следующее значение поля-счетчика
Avg_row_length	Средняя длина записи
Checktime	Время последнего выполнения инструкции CHECK TABLE
Comment	Комментарии, указанные при создании таблицы
Createoptions	Параметры создания таблицы
Createtime	Время создания таблицы
Data_free	Число выделенных, но неиспользуемых байтов
Data_length	Длина файла данных
Index_length	Длина индексного файла
Max_data_length	Максимальная длина файла данных
Name	Имя таблицы
Rows	Число записей в таблице

Row_format	Формат хранения записей: сжатый, динамический или фиксированный
Type	Тип таблицы
Updatetime	Время последнего обновления таблицы

SHOW COLUMNS

Инструкция *SHOW COLUMNS* возвращает описание столбцов таблицы. Она имеет следующий синтаксис:

SHOW [FULL] {COLUMNS | FIELDS}

FROM таблица [**FROM** база_данных] [**LIKE** шаблон]

Флаг **FULL** задает выдачу для каждого столбца информации о привилегиях. Предложение *LIKE* позволяет отобрать из таблицы только те столбцы, имена которых соответствуют указанному шаблону. Аналогичные результаты выдает инструкция *DESCRIBE*.

Инструкция SHOW COLUMNS

mysql> **SHOW COLUMNS FROM user;**

```
+-----+-----+-----+-----+-----+
| Field      | Type          | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+
| Host       | char(60) binary |      | PRI |          |       |
| User       | char(16) binary |      | PRI |          |       |
| password   | char(16)       |      |     |          |       |
| Select_priv | enum('N','Y')  |      |     | N        |       |
| Insert_priv | enum('N','Y')  |      |     | N        |       |
| Update_priv | enum('N','Y')  |      |     | N        |       |
| Delete_priv | enum('N','Y')  |      |     | N        |       |
| Create_priv | enum('N','Y')  |      |     | N        |       |
| Drop_priv  | enum('N','Y')  |      |     | N        |       |
| Reload_priv | enum('N','Y')  |      |     | N        |       |
| Shutdown_priv | enum('N','Y') |      |     | N        |       |
| Process_priv | enum('N','Y') |      |     | N        |       |
| File_priv  | enum('N','Y')  |      |     | N        |       |
| Grant_priv  | enum('N','Y')  |      |     | N        |       |
| References_priv | enum('N','Y') |      |     | N        |       |
| Index_priv  | enum('N','Y')  |      |     | N        |       |
| Alter_priv  | enum('N','Y')  |      |     | N        |       |
+-----+-----+-----+-----+-----+
17 rows in set (0.00 sec)
```

2. Выполнить задание:

Задание 1.

Для подключения к удаленному серверу необходимо указать его адрес с помощью опции **-h host_name**, где host_name = IP - адрес сервера (всем подключаться к - 192.168.XX.XX), имя пользователя - опцией **-u user_name**, где user_name - ваш уникальный идентификатор, пароль - опцией **-p** (пароль после опции вводить не надо, он будет запрошен у вас позже).

Все эти параметры узнайте у преподавателя.

Пример соединения с сервером:

Перед вводом кода необходимо войти в директорию, куда вы скопировали файл **mysql.exe**

```
mysql -u stud -p -h 192.168.14.146
```

```
Enter password: 1111
```

После этого необходимо открыть свою базу данных: **\u XXistGG**

Выбор базы данных

Там уже будет создана ваша база данных (**XXistGG**, где XX - ваш номер, а GG - номер вашей группы), которую вы создали на предыдущем практическом занятии.

Создать таблиц: **Transaction** - содержит записи об операциях пользователя.

Описание заданий:

1.1. Выбрать свою базу данных.

1.2. В ней создать таблицу **Transaction**

Transaction - содержит записи об операциях пользователя.

Таблица Transaction

Название поля	Описание
OrderNo	Уникальный идентификатор операции пользователя
UserId	ID пользователя
ItemNo	Уникальный идентификатор товара. Этому элементу должна соответствовать строка в таблице <i>MusicShop</i> или <i>BookShop</i>
Quantity	Количество товаров ItemNo, заказанных пользователем
Date	Дата осуществления пользователем операции
Status	Статус товара - поставлен/ожидается (shipped/pending)

Произведите "*Изменение определения столбца*" **Статус** (пример смотрите в лекции)

Типы столбцов вы должны указать сами.

*При работе с программным кодом вам понадобится следующая команда \с, которая позволяет в случае необходимости очистить окно команд. Например, если при вводе программного кода вы допустили ошибку, а уже начали набирать другую строку, вам необходимо набрать эту команду. После этого, чтобы не вводить программный код заново при помощи стрелочек вверх/вниз выбирать нужные команды и нажимать **Enter**.*

1.3. Изменить таблицу **UserProfile**, добавив следующие поля:

Название поля	Описание
CardNo	Номер кредитной карточки
ExpiryDate	Дата истечения срока действия кредитной карточки
CardType	Тип кредитной карточки - Master/Visa
AccountBalance	Сальдо счета пользователя

- 1.4. Создать для таблиц первичные ключи, предварительно удалив ранее созданный в таблице **BookShop**:

Название таблицы	Название поля
UserProfile	userId
BookShop	itemNo
MusicShop	itemNo
Transaction	orderNo

- 1.5. Для таблицы **Transaction** первичный ключ должен быть со счетчиком.
1.6. Задать значение по умолчанию для поля **quantity** таблицы **Transaction** равным **0**.
1.7. Переименовать таблицы **BookShop** в **Book** и **MusicShop** в **Music**.

Задание 2.

- 2.1. Выбрать свою базу данных.
2.2. Изменить типы столбцов, используя оператор **ALTER**, в таблицах **UserProfile**, **Book**, **Music** и **Transaction** на следующие:

Описание таблиц:

Таблица UserProfile

Название поля	Тип столбца
Fname	VARCHAR(32)
Lname	VARCHAR(32)
Userid	VARCHAR(16)
Password	VARCHAR(16)
Address	VARCHAR(128)
City	VARCHAR(64)
ZipCode	VARCHAR(8)
Gender	VARCHAR(8)
Age	INTEGER
Emailed	VARCHAR(64)

PhoneNumber	VARCHAR(16)
cardNo	VARCHAR(16)
expiryDate	DATE
cardType	ENUM("Master", "Visa")
accountBalance	FLOAT

Таблица Book

Название поля	Тип столбца
ItemNo	VARCHAR(20)
ItemType	VARCHAR(20)
Title	VARCHAR(60)
Author	VARCHAR(60)
Price	FLOAT

Таблица Music

Название поля	Тип столбца
ItemNo	VARCHAR(20)
ItemType	VARCHAR(20)
Title	VARCHAR(60)
Artist	VARCHAR(60)
Price	FLOAT

Таблица Transaction

Название поля	Тип столбца
OrderNo	INT
Userid	VARCHAR(20)
ItemNo	VARCHAR(20)
Quantity	INT
Date	DATE
Status	ENUM("shipped",

	"pending")
--	------------

Все столбцы должны иметь тип **NOT NULL**

Пример изменения типов столбцов для таблицы Book:

```
mysql> ALTER TABLE Book
```

```
-> CHANGE ItemNo
```

```
-> ItemNo VARCHAR(20) NOT NULL;
```

```
mysql> ALTER TABLE Book
```

```
-> CHANGE ItemType
```

```
-> ItemType VARCHAR(20) NOT NULL;
```

```
mysql> ALTER TABLE Book
```

```
-> CHANGE Title
```

```
-> Title VARCHAR(60) NOT NULL;
```

```
mysql> ALTER TABLE Book
```

```
-> CHANGE Author
```

```
-> Author VARCHAR(60) NOT NULL;
```

```
mysql> ALTER TABLE Book
```

```
-> CHANGE Price
```

```
-> Price FLOAT NOT NULL;
```

2.3. Создание индексов

Создайте индексы по полям **ItemNo**, **Title** и **Author/Artist** в таблицах **Book** и **Music**.

Вы создали свою базу данных, включающую четыре таблицы. Созданная БД будет использована при дальнейшем изучении MySQL.

Отчет :

- Выполненные задания сдайте преподавателю.

Лабораторная работа № 6. Наполнение БД

Можно назвать несколько методов добавления информации в базу данных. Можно добавлять записи в таблицы вручную. Для этой цели служит оператор **INSERT**. Можно добавлять записи прямо из файла или как исходные данные с помощью команды **LOAD DATA** или в виде предварительно созданных и сохраненных в файле операторов **INSERT**. В этой практической работе вы должны применить все эти методы.

Вы научитесь:

- добавлять записи в таблицу;
- просматривать свойства столбцов таблицы;
- создавать первичные ключи.

Начало занятия:

1. *изучить учебный материал:*
 - [Вставка записей](#)
 - [Обновление записей](#)
 - [Удаление записей](#)

Для создания таблицы и модификации структуры уже созданных таблиц необходимо подключиться к серверу. Для этого воспользуемся клиентской программой **MySQL-Front**. (скачать с [FTP://DATA.SCC/sql/MySQL-Front](ftp://data.scc/sql/MySQL-Front))

Для подключения к удаленному серверу необходимо указать:

Сервер - **192.168.16.50**

Пользователь -stud, Пароль - 111, База данных- *ваш уникальный идентификатор* (Например – **01pov19**)

Выполнить следующие задания:

1. Выбрать свою базу данных.
2. Добавить в таблицу **UserProfile** следующие данные:

Fname	Lname	Userid	Password	Address	City	ZipCode	Gender
Ирина	Малий	1		Ул. Ленина 330	Ставрополь	355000	Ж
Сергей	Лубяной	2		Ул. Лермонтова 456 кв. 17	Курск	365000	М
Марина	Бекетова	3		Ул. Первомайская 41 кв. 51	Краснодар	357000	Ж

Евгений	Сердюков	4		Ул. Мира 123	Саратов	345000	М
Евгений	Савченко	5		Ул. Сенная 26 кв. 15	Челябинск	385000	М

Age	Email	PhoneNumber	CardNumber	ExpiryDate	CardType	AccountBalance
23		23-56-78	987456213	01.03.06	Master	31
21		45-68-97	564321978	01.05.05	Visa	2580
35		36-73-68	978456352	01.01.03	Master	1500
26		45-66-54	458621473	01.12.07	Visa	250
45		74-12-41	459365121	01.06.09	Master	600

☐ Добавить в таблицу **Book** следующие данные:

ItemNumber	ItemType	Title	Author	Price
5-98547-501-3	Программирование	PHP: настольная книга программиста	А. Мазуркевич	234
5-93286-049-9	Web программирование	Профессиональное PHP программирование	Л. Алгерих	500
5-94157-059-7	Web дизайн	JavaScript в Web - дизайне	Владимир Дронов	204
5-7989-0229-3	Web программирование	Как программировать на C++	Х. М. Дейтл	450
5-9518-	Базы данных	Visual FoxPro 7	С. А. Каратыгин	180

0006-4				
5-8459-0158-2	Программировани е	MySQL	Поль Дюбуа	200
5-279-02204-7	Базы данных	Язык C++	Подбельски й В. В.	120
5-8459-0291-6	Базы данных	Библиотека профессионала MySQL	Леон Аткинсон	230

☐ Добавить в таблицу **Music** следующие данные:

ItemNo	ItemType	Title	Artist	Price
100	CD	Вечерняя коллекция	Кабриолет	80
101	кассета	Первый контакт	Hi-Fi	30
102	CD	Овалы	Вячеслав Бутусов	40
103	кассета	200 по встречной	ТаТУ	30
104	CD	Группа крови на рукаве	Кино	120
105	кассета	Fallen	Evanescence	30
106	CD	Мама	Витас	80
107	кассета	Seance At The Che Chaebol	VACUUM	250
108	CD	The Rapsody vol.2	Nostradame feat.	80
109	касета	Sleep	ENIGMA	30
110	CD	The Very Best	Rammstein	130
111	кассета	Dj Dancys Collection	Christina Aguilera	100
112	CD	Escape	Enrique Iglesias	140
113	кассета	Навека	Корни	70
114	CD	EuroIVever	E-Type	80

Добавьте в каждую из таблиц по 3 похожих записи, используя инструкцию *INSERT INTO*.

Для этого создайте текстовый файл с записями каждой таблицы.

*1. Измените в таблице **Book** следующие записи:*

*5.1. В записи, в которой поле *ItemNo* = 5-98547-501-3, изменить поле *ItemType* на "Web программирование";*

*5.2. В записи, в которой поле *ItemNo* = 5-7989-0229-3, изменить поле *ItemType* на "Программирование";*

*5.3. В записи, в которой поле *ItemNo* = 5-8459-0158-2, изменить поле *ItemType* на " Базы данных ";*

*5.4. В записи, в которой поле *ItemNo* = 5-279-02204-7, изменить поле *ItemType* на " Программирование ";*

- Удалить в таблице **Music** записи, цена которых равна 30*
- Сохраните в текстовой файле примеры запросов: на удаление, добавление, модификацию данных для Вашей базы данных.*

Отчет :

- Выполненные задания сдайте преподавателю.*

Лабораторная работа № 7-8. Создание запросов. Создание сложных запросов

Зачем же вводить данные в базу, если не для их выборки и последующей обработки? Для этого и предназначен оператор **SELECT**. Возможно, это самый популярный оператор языка SQL, но верно и другое. Это самый сложный оператор языка SQL.

Перед выполнением практической работы необходимо изучить следующий материал:

1.1. Запросы

Инструкция **SELECT** извлекает строки из одной или нескольких таблиц. Формат этой инструкции сложнее, чем у других инструкций:

```
SELECT имя_столбца, ...  
FROM имя_таблицы, ...  
WHERE условие_отбора  
GROUP BY имя_столбца, ...  
ORDER BY имя_столбца, ...  
LIMIT лимит
```

Правда, многие предложения инструкции являются необязательными.

Выборка всех строк

```
SELECT * FROM item
```

Вверху представлен простейший вариант инструкции **SELECT**. В данном случае отображаются все столбцы всех строк таблицы *item*. Символ звездочки (*) заменяет собой имена всех столбцов. Запрос, показанный далее, возвращает тот же результат, но имена столбцов на этот раз перечислены явно. Порядок имен важен, так как именно в этом порядке столбцы войдут в результирующую таблицу

Указание имен столбцов в инструкции SELECT

```
SELECT ID, Name, Price, Description  
FROM item
```

Если указаны не все столбцы, операция выборки превратится в операцию проекции. Дополнительный фильтр записей задается с помощью предложения **WHERE**.

Фильтрация строк и столбцов

```
SELECT ID, Name  
FROM item  
WHERE ID > 2 AND ID < 10
```

Показанное вверху предложение **WHERE** состоит из двух операций сравнения. В условиях отбора могут присутствовать многие операторы языков третьего поколения, а также целый ряд встроенных функций. Можно также вычислять выражения, не ссылаясь на таблицы. Например, далее показано, как узнать текущие дату и время, "извлекая" значение из встроенной функции **NOW ()**.

"Выборка" значения функции

```
mysql> SELECT NOW();
```

```
+-----+
| now() |
+-----+
| 2004-03-29 20:06:22 |
+-----+
1 row in set (0.00 sec)
```

Объединения

Объединение создается путем указания нескольких таблиц в предложении **FROM**. Существуют два варианта записи объединений. В первом из них в предложении **FROM** указывается выражение объединения. Этим способом чаще всего создают внешние объединения. Второй вариант - применение условия отбора, заданного в предложении **WHERE**, к произведению таблиц, перечисленных в предложении **FROM**. Так получаются внутренние объединения.

Для примера создадим вторую таблицу *item_option*, которая будет участвовать в операциях объединения с таблицей *item*. Далее показаны инструкции, создающие обе таблицы и заносящие в них данные. В таблице *item_option* хранятся описания возможных вариантов изготовления товаров, перечисленных в таблице *item*.

Создание таблиц *item* и *item_option*

```
CREATE TABLE item (
    ID INT(6) NOT NULL AUTO_INCREMENT,
    Name CHAR(32) NOT NULL,
    Price DECIMAL(4,2) NOT NULL,
    Description CHAR(255) DEFAULT 'No Description',
    PRIMARY KEY(ID),
    KEY (Name)
);
```

```
INSERT INTO item VALUES (1, 'Toothbrush', 1.25, NULL);
INSERT INTO item VALUES (2, 'Comb', 2.50, NULL);
INSERT INTO item VALUES (3, 'Brush', 3.00, NULL);
INSERT INTO item VALUES (4, 'Toothpaste', 0.75, NULL);
```

```
CREATE TABLE item_option (
    ID INT(6) NOT NULL AUTO_INCREMENT,
    Item INT(6) NOT NULL,
    Name CHAR(32) NOT NULL,
    PRIMARY KEY(ID),
    FOREIGN KEY(Item) REFERENCES Item (ID)
);
```

```
INSERT INTO item_option VALUES (1, 2, 'Red Plastic');
INSERT INTO item_option VALUES (2, 2, 'Blue Plastic');
```

Чтобы создать внутреннее объединение таблиц, достаточно указать их имена в предложении **FROM** инструкции **SELECT**. При отсутствии предложения **WHERE** получится самый общий вариант такого объединения - декартово произведение, в котором каждая строка первой таблицы объединена с каждой строкой второй таблицы. Если же задано условие отбора, то в результаты запроса попадут лишь те строки таблицы произведения, в которых значения связанных столбцов удовлетворяют требуемому критерию (связанным называется столбец, присутствующий в условии отбора). Далее выполняется объединение двух таблиц по равенству столбцов *item.ID* и *item_option.Item*, т.е. для каждой строки первой таблицы находится соответствующий элемент второй таблицы. Из объединенной таблицы отбираются три столбца: с названием и ценой товара, а также с названием варианта его изготовления.

Внутреннее объединение таблиц с использованием предложения **WHERE**

```
mysql> SELECT item.Name, item.Price, item_option.Name AS 'Option Name'
-> FROM item, item_option
-> WHERE item.ID = item_option.Item;
```

```
+-----+-----+-----+
| Name | Price | Option Name |
+-----+-----+-----+
| Comb | 2.50 | Red Plastic |
| Comb | 2.50 | Blue Plastic |
+-----+-----+-----+
```

2 rows in set (0.00 sec)

Вверху содержатся примеры псевдонимов таблиц и столбцов. Третий столбец результатов запроса называется так же, как и первый, поэтому во избежание путаницы он переименуется в *Option Name*. На псевдонимы столбцов можно ссылаться в предложениях **GROUP BY** и **ORDER BY**, рассматриваемых ниже.

Псевдонимы таблиц необходимы для того, чтобы не возникала путаница с одноименными столбцами. Например, в обеих таблицах есть столбец *Name*, поэтому MySQL выдаст сообщение об ошибке, если не уточнить имя таблицы, к которой относится столбец. Псевдоним таблицы обычно выбирается максимально коротким, чтобы полные имена столбцов было удобно набирать. Длина псевдонима не может превышать 255 символов. Далее показано то же самое внутреннее объединение, что и вверху, но на этот раз в предложении **FROM** присутствует операция **INNER JOIN**. В подобном случае условие отбора переносится из предложения **WHERE** в предложение **ON**.

Внутреннее объединение таблиц в предложении **FROM**

```
mysql> SELECT item.Name, item.Price, item_option.Name AS 'Option Name'
-> FROM item INNER JOIN item_option
-> ON item.ID = item_option.Item;
```

```
+-----+-----+-----+
```

Name	Price	Option Name
Comb	2.50	Red Plastic
Comb	2.50	Blue Plastic

2 rows in set (0.00 sec)

Иногда внутреннее объединение не дает нужных результатов. Оно формирует пары только тех строк, между которыми найдено соответствие. Все остальные строки исключаются. Чтобы включить в полученные выше результаты названия все товаров, нам потребуется выполнить внешнее объединение, в данном случае левое

Левое внешнее объединение

```
mysql> SELECT item.Name, item.Price, item_option.Name AS 'Option Name'
-> FROM item LEFT JOIN item_option
-> ON item.ID = item_option.Item;
```

Name	Price	Option Name
Toothbrush	1.25	NULL
Comb	2.50	Red Plastic
Comb	2.50	Blue Plastic
Brush	3.00	NULL
Toothpaste	0.75	NULL

5 rows in set (0.00 sec)

Как видите, отображаются пять строк. В объединенную таблицу вошли все строки таблицы *item*, даже те, которым не найдено соответствие. Строка, не имеющая "пары", дополняется значениями **NULL**.

Упорядочение результатов запроса

Предложение *ORDER BY* содержит имена столбцов, по которым осуществляется сортировка результатов запроса. На столбец можно сослаться по имени (краткому или полному), псевдониму или порядковому номеру в предложении *SELECT* (нумерация столбцов начинается с единицы).

Сортировка товаров по названию

```
mysql> SELECT item.Name, item.Price, item_option.Name AS 'Option Name'
-> FROM item LEFT JOIN item_option
-> ON item.ID = item_option.Item
-> ORDER BY item.Name;
```

Name	Price	Option Name
Brush	3.00	NULL
Comb	2.50	Red Plastic
Comb	2.50	Blue Plastic
Toothbrush	1.25	NULL

Toothpaste	0.75	NULL
------------	------	------

5 rows in set (0.00 sec)

Предложение **ORDER BY** может содержать не одно имя, а целый список. Столбцы перечисляются в порядке убывания приоритетов. В первую очередь записи сортируются по столбцу, указанному первым. Если возникает "конфликт" (две или более строк имеют одинаковое значение в данном поле), проверяется второй столбец, затем третий и т.д.

По умолчанию сортировка всех столбцов ведется по возрастанию. Если нужно задать обратный порядок, укажите рядом с именем столбца ключевое слово **DESC**. Далее показан пример сортировки товаров по цене, начиная от самых дорогих.

Сортировка товаров по цене

```
mysql> SELECT item.Name, item.Price, item_option.Name AS 'Option Name'
-> FROM item LEFT JOIN item_option
-> ON item.ID = item_option.Item
-> ORDER BY item.Price DESC;
```

Name	Price	Option Name
Brush	3.00	NULL
Comb	2.50	Red Plastic
Comb	2.50	Blue Plastic
Toothbrush	1.25	NULL
Toothpaste	0.75	NULL

5 rows in set (0.00 sec)

Группировка результатов запроса

Предложение **GROUP BY** позволяет группировать записи, вошедшие в результаты запроса, с тем чтобы над ними можно было выполнять статистические функции. Все элементы списка возвращаемых столбцов должны иметь одно значение для каждой группы строк. Столбцы, указываемые в предложении **GROUP BY**, называются столбцами группировки. Они по определению имеют одинаковое значение во всех строках группы. В предложении **SELECT** запроса с группировкой разрешается указывать только столбцы группировки и статистические функции, а также выражения с ними.

Далее представлен запрос, в котором записи объединенной таблицы группируются по названию товара и его цене. В принципе, достаточно выполнять группировку только по полю *Name*, но в этом случае столбец *Price* нельзя было бы включить в результаты запроса.

Группировка записей

```
mysql> SELECT item.Name, item.Price, COUNT(item_option.ID) AS 'Options'
-> FROM item LEFT JOIN item_option
-> ON item.ID = item_option.Item
```

-> *GROUP BY item.Name, item.Price*
 -> *ORDER BY Options;*

Name	Price	Options
Toothbrush	1.25	0
Brush	3.00	0
Toothpaste	0.75	0
Comb	2.50	2

4 rows in set (0.00 sec)

В третьем столбце содержатся результаты выполнения встроенной статистической функции **COUNT ()**. Для каждой группы она подсчитывает число значений, не равных *NULL*.

Ограничение числа возвращаемых записей

С помощью предложения **LIMIT** можно ограничивать число записей, возвращаемых в упорядоченной таблице результатов запроса. Если задано одно число, будет выдана таблица, содержащая указанное количество строк, начиная с самой первой.

Если указаны два числа, разделенных запятой, то первое из них определяет начальную строку выборки, а второе - размер выборки.

Далее показан запрос, возвращающий описание товара, у которого наибольшее число вариантов изготовления. Если таких записей окажется две или более, будет выбрана та из них, которая стоит первой в списке.

Предложение **LIMIT** инструкции **SELECT** является специфичной особенностью MySQL.

Ограничение числа возвращаемых записей

```
mysql> SELECT item.Name, item.Price, COUNT(item_option.ID) AS 'Options'
-> FROM item LEFT JOIN item_option
-> ON item.ID = item_option.Item
-> GROUP BY item.Name, item.Price
-> ORDER BY Options DESC, Name
-> LIMIT 1;
```

Name	Price	Options
Comb	2.50	2

1 row in set (0.00 sec)

1.2. Синтаксис оператора **SELECT**

Инструкция **SELECT** предназначена для извлечения информации из таблиц. Она имеет следующий синтаксис:

```
SELECT
  [DISTINCT | DISTINCTROW | ALL]
  [HIGH_PRIORITY]
```

```

[SQL_BIG_RESULT I SQL_SMALL_RESULT]
[SQL_BUFFER_RESULT]
[STRAIGHT_JOIN]
выражение[, выражение ...]
[INTO (OUTFILE I DUMPFILE} файл опции]
[FROM таблица[, таблица]
    [WHERE условие]
    [GROUP BY столбец[, столбец]]
    [HAVING условие]
    [ORDER BY столбец [ASC | DESC][, столбец [ASC I DESC]]]
    [LIMIT [смещение,] число_строк]
    [PROCEDURE процедура] ]

```

Инструкция вычисляет выражения, указанные в списке возвращаемых столбцов. В выражения могут входить литералы, переменные, функции и столбцы. Последние относятся к таблицам, перечисленным в предложении **FROM**. Результатом запроса будет таблица извлеченных записей, содержащая в качестве заголовка названия столбцов.

На столбцы ссылаются по имени. Если возникает неоднозначность, необходимо также указывать имя таблицы и базы данных. Полное имя столбца состоит из имени базы данных, имени таблицы и короткого имени столбца, разделенных точками. Таблицам могут быть назначены псевдонимы в предложении *FROM*, чтобы полные имена столбцов получались короче. Каждому выражению в списке возвращаемых столбцов тоже может быть присвоен псевдоним с помощью предложения *AS*. Этот псевдоним появится в заголовке возвращаемой таблицы. Благодаря псевдониму на выражение можно ссылаться в других предложениях инструкции *SELECT*, например *GROUP BY*. Если псевдоним представляет собой несколько слов с пробелами, требуются кавычки.

Использование псевдонимов столбцов

```
SELECT NOW() AS 'Current Time'
```

Псевдоним отдельного столбца может упоминаться в любой части инструкции *SELECT*. Псевдонимы выражений запрещены в предложении *WHERE*.

Флаг ***DISTINCT*** указывает на необходимость удаления дублирующихся записей из таблицы результатов. В данном случае дубликатами считаются такие записи, которые имеют одинаковые значения во всех столбцах таблицы результатов. Речь не идет о проверке первичного ключа или ограничения уникальности. Например, если запросить значения поля "Возраст" таблицы сотрудников, то наверняка некоторые значения будут одинаковыми, хотя в исходной таблице все записи различаются по идентификатору.

DISTINCTROW - это синоним флага *DISTINCT*. Флаг ***ALL*** разрешает появление дубликатов и установлен по умолчанию.

Флаг ***HIGH_PRIORITY*** означает принудительное выполнение запроса, даже если имеется отложенная инструкция **UPDATE**. Обычно операции обновления выполняются в первую очередь.

Флаг ***SQL BIG RESULT*** заставляет программу MySQL создавать в случае необходимости временные таблицы на диске. Это требуется для запросов к большому числу таблиц или запросов, возвращающих большую таблицу результатов, особенно если ее нужно сортировать при наличии флага ***DISTINCT*** или предложения ***ORDER BY***.

Флаг ***SQL_SMALL_RESULT*** говорит о том, что временные таблицы должны создаваться в памяти. В новых версиях MySQL этот флаг не нужен.

Флаг ***SQL_BUFFER_RESULT*** заставляет предварительно помещать результаты во временную таблицу, а не сразу же посылать их клиенту. Это позволяет быстрее освобождать таблицы, участвующие в запросе.

Флаг ***STRAIGHT_JOIN*** говорит о том, что таблицы должны включаться в объединение в том порядке, в котором они перечислены в предложении ***FROM***.

В предложении ***INTO*** задается файл, куда должна быть записана таблица результатов. Полный формат этого предложения таков:

```
INTO {OUTFILE | DUMPFILE} файл
      [FIELDS [TERMINATED BY разделитель]
        [[OPTIONALLY] ENCLOSED BY ограждающий_символ]
        [ESCAPED BY управляющий_символ]]
      [LINES TERMINATED BY признак_конца_строки]
```

Поскольку запись в файл осуществляется на сервере, необходимо иметь привилегию ***FILE***. Более того, запрещается перезаписывать существующий файл. Файл, созданный в режиме ***OUTFILE***, можно будет загрузить с помощью инструкции ***LOAD DATA INFILE*** (см. выше). Режим ***DUMPFILE*** предназначен для записи столбцов типа ***BLOB*** или ***TEXT***. В этом режиме запрос должен возвращать одну запись с одним-единственным полем.

Управляющие символы в данном случае не используются.

В предложении ***FROM*** указываются таблицы, из которых извлекаются данные. Здесь может стоять либо список таблиц, разделенных запятыми, либо одна из показанных ниже спецификаций объединения:

```
таблица [CROSS] JOIN таблица
таблица STRAIGHT_JOIN таблица
таблица INNER JOIN таблица
      [{ON условие | USING (столбец, ...)}]
таблица (LEFT | RIGHT) [OUTER] JOIN таблица
      [{ON условие | USING (столбец, ...)}]
таблица NATURAL [{LEFT | RIGHT} [OUTER]] JOIN таблица
```

В операции объединения могут участвовать таблицы стандартной базы данных, таблицы заданной базы данных, а также виртуальная таблица, созданная в левой части объединения. В последнем случае создается цепочка объединений. В принципе, она существует лишь до тех пор, пока выполняется инструкция ***SELECT***, но в данном контексте на нее можно ссылаться как на настоящую таблицу, т.е. справа от нее может стоять оператор объединения.

Синтаксис табличной ссылки таков:

[база_данных.]таблица AS псевдоним

[USE INDEX {индекс, ...}] [IGNORE INDEX {индекс, ...}]

Перед именем таблицы может стоять имя базы данных. Предложение *AS* задает псевдоним таблицы. Этот псевдоним можно использовать во всех остальных предложениях инструкции, в частности в операциях объединения, в условиях отбора, в списке возвращаемых столбцов и т.д. Предложение *USE INDEX* заставляет программу MySQL использовать указанные индексы, если это возможно, а предложение *IGNORE INDEX*, наоборот, запрещает некоторые индексы. Эти подсказки позволяют повысить производительность запроса.

Операция **CROSS JOIN** создает произведение таблиц. В этом случае каждая строка левой таблицы по очереди объединяется с каждой строкой правой таблицы и включается в таблицу произведения. Операция **STRAIGHT_JOIN** приводит к тем же результатам.

Далее выполняется самообъединение таблицы, т.е. создается произведение двух копий одной и той же таблицы. Если в исходной таблице 4 строки, то в таблице произведения их будет 16.

Самообъединение таблицы

```
SELECT t1.Name, t2.Name FROM team t1 JOIN team t2
```

Операция **INNER JOIN** включает в таблицу результатов только те строки таблицы произведения, которые удовлетворяют определенному критерию. Это называется внутренним объединением. Предложение **USING** задает столбцы, включаемые в объединение. Эти столбцы должны называться одинаково в обеих таблицах. Далее таблица игроков (player) объединяется с таблицей команд (team) по столбцу идентификатора команды (TeamID).

Внутреннее объединение таблиц с использованием предложения USING

```
SELECT player.Name, team.Name  
FROM player INNER JOIN team USING (TeamID)
```

Предложение **ON** заменяет предложение *WHERE* и может содержать любые критерии сравнения строк двух объединяемых таблиц. Чаще всего таблицы объединяются по равенству двух столбцов, как показано далее. Обратите внимание на различные имена столбцов.

Внутреннее объединение таблиц с использованием предложения ON

```
SELECT player.Name, team.Name  
FROM player INNER JOIN team  
ON (player.Team = team.ID)
```

В левое внешнее объединение (*LEFT JOIN*) включаются все записи внутреннего объединения плюс строки первой таблицы, которые не имеют связи ни с одной строкой второй таблицы. Правое внешнее объединение (*RIGHT JOIN*) строится так же, но относительно правой, а не левой таблицы. Строки, которым не найдено соответствие, дополняются значениями *NULL*. Ключевое слово **OUTER** является необязательным. Оно предназначено для сохранения совместимости с другими СУБД.

Естественное объединение (*NATURAL JOIN*) характерно тем, что сравниваются только одноименные столбцы двух таблиц, причем выполняется их проверка на равенство. Обычно это действительно самый "естественный" тип объединения, так как не нужно указывать ни имена столбцов, ни условие отбора. По умолчанию создается внутреннее естественное объединение, но с помощью ключевых слов *LEFT* и *RIGHT* его можно сделать левым или правым внешним объединением.

Некоторые предложения инструкции *SELECT* доступны лишь тогда, когда присутствует предложение *FROM*. Они выполняются в строго определенном порядке: сначала - *WHERE*, затем - *GROUP BY*, потом - *HAVING* и, наконец - *ORDER BY*.

Предложение *WHERE* задает условия, в соответствии с которыми из таблицы произведения отбираются строки. Если в предложении *FROM* была указана одна таблица, то условие отбора является критерием фильтрации ее строк. Если же выполняется операция объединения, то в предложении *WHERE* указываются правила сопоставления столбцов объединяемых таблиц. Далее создается то же самое объединение, что и выше, но на этот раз с использованием предложения *WHERE*.

Объединение таблиц с использованием предложения WHERE

```
SELECT player.Name, team.Name  
FROM player, team  
WHERE player.Team = team.ID
```

Аргументом предложения *WHERE* является булево выражение. В него может входить произвольное число выражений сравнения, объединяемых с помощью логических операторов. Далее показана более сложная форма предложения *WHERE*.

Составное предложение WHERE

```
SELECT p.Name, t.Name  
FROM player p, team t  
WHERE p.Team = t.ID  
AND (p.Position = 'Catcher' OR p.Position = 'Pitcher')
```

Предложение *GROUP BY* содержит правило группировки строк по одному или нескольким столбцам. Это позволяет применять к столбцам статистические функции. Статистические функции разрешены в списке возвращаемых столбцов и в предложении *HAVING*, но не в предложении *WHERE*, поскольку оно вычисляется раньше, чем *GROUP BY*, т.е. до того, как формируются группы записей.

Запрос, показанный далее, группирует записи по идентификатору команды и возвращает число питчеров в каждой команде при условии, что таковых более девяти.

Запрос с группировкой

```
SELECT t.Name AS 'Team', COUNT(*) AS 'Pitchers'  
FROM player p, team t  
WHERE p.Team = t.ID  
AND p.Position = 'Pitcher'
```

GROUP BY t.ID HAVING Pitchers > 9
ORDER BY Pitchers DESC

В предложении *GROUP BY* можно ссылаться на столбцы по имени или по номеру. Столбцы нумеруются слева направо, а нумерация начинается с единицы. Разрешается также группировать записи по столбцам, не указанным в предложении *SELECT*. Такие столбцы, естественно, не имеют номера.

В предложении **HAVING** указывается такое же булево выражение, как и в предложении *WHERE*. Оно задает условия отбора групп. Поскольку предложение *HAVING* проверяется после предложений *WHERE* и *GROUP BY*, в нем можно использовать псевдонимы столбцов и статистические функции. Разрешается ссылаться только на те столбцы, которые указаны в списке возвращаемых столбцов.

Предложение *ORDER BY* задает правило сортировки строк возвращаемой таблицы. Разрешается сортировать результаты запроса по любому элементу списка возвращаемых столбцов. Как и в случае предложения *GROUP BY*, на столбцы можно ссылаться по номерам.

По умолчанию столбцы сортируются по возрастанию, т.е. флаг *ASC* указывать не обязательно. Для сортировки по убыванию предназначен флаг *DESC*.

Предложение *LIMIT* позволяет отобрать из возвращаемой таблицы требуемое подмножество строк. Если в качестве аргумента указано одно число, возвращаются строки с первой по заданную. Если присутствуют два числа, то первое определяет начальный номер строки, а второе - число возвращаемых строк.

Предложение **PROCEDURE** предназначено для вызова процедуры, которая выполняется над таблицей результатов запроса перед тем, как эта таблица посылается клиенту. В MySQL есть лишь одна такая процедура: *analyse ()*. Для совместимости с *Oracle* разрешается ссылаться на виртуальный столбец **rowid** (псевдоним первичного ключа). Еще один виртуальный столбец - *auto_increment_column* - позволяет находить последнюю вставленную строку таблицы, содержащей столбец-счетчик. Соответствующая проверка выглядит так:

WHERE auto_increment_column IS NULL

Для создания таблицы и модификации структуры уже созданных таблиц необходимо подключиться к серверу. Для этого воспользуемся клиентской программой **Mysql-Front** .(скачать с **FTP://DATA.SCC**)

Выполнить следующие задания:

1. Произведите просмотр данных своей базы, размещенной на сервере 192.168.16.50. Наполните данными базу, если Вы не выполняли предыдущую практическую работу (см Пр.№ 7).

Создайте и сохраните следующие SQL- запросы:

- "Вывести все записи из таблицы **Music** уникальный идентификатор которых находится в диапазоне от **100** до **107**."

- "Вывести первые 3 записи из таблицы **Book** цена которых больше 200. В результат включить столбцы **ItemType**, **Title**, **Author** используя следующие псевдонимы для столбцов соответственно '**Тип книги**', '**Название**', '**Автор**'. Результат выборки упорядочите по убыванию".
- "Вывести **имена** (Fname), **фамилии** (Lname) и **номера кредитных карт** (CardNo) всех пользователей из таблицы **UserProfile** у которых срок действия кредитной карты (ExpiryDate) уже истек". Для указания текущей даты используйте функцию *CURRENT_DATE* (текущая дата).
- "Подсчитайте количество записей таблицы **music**, где уникальный идентификатор (**ItemType**) равен **CD**".
- "Произведите подсчет количества товара определенной категории (**ItemType**) из таблицы **music**".

Добавьте столбцы в таблицу **Transaction**, т.к. столбец **OrderNo** является счетчиком, то при вставке его указывать не надо, для указания текущей даты воспользуйтесь функцией **CURDATE()**.

Название поля	Описание
OrderNo	Уникальный идентификатор операции пользователя
Userid	ID пользователя
ItemNo	Уникальный идентификатор товара. Этому элементу должна соответствовать строка в таблице MusicShop или BookShop
Quantity	Количество товаров ItemNo, заказанных пользователем
Date	Дата осуществления пользователем операции
Status	Статус товара - поставлен/ожидает (shipped/pending)

Наполните данными данную таблицу (несколько записей).

Пример вставки записи по выборке для одного столбца:

```
INSERT IGNORE
INTO Transaction(userID, ItemNo, quantity, date, status)
SELECT userprofile.userID, book.ItemNo, '2', CURDATE(), 'pending'
FROM userprofile, Book
WHERE userprofile.Lname= 'Бекетова' AND Book.title = 'PHP:
настольная книга программиста';
```

Отчет :

- Выполненные задания сдайте преподавателю.

Окончание занятия:

Завершите сеанс.

Создание сложных запросов.

Запрос - команда которую вы даете вашей программе базы данных, и которая сообщает ей чтобы она вывела определенную информацию из таблиц в память. Эта информация обычно посылается непосредственно на экран компьютера или терминала которым вы пользуетесь, хотя, в большинстве случаев, ее можно также послать принтеру, сохранить в файле (как объект в памяти компьютера), или представить как вводную информацию для другой команды или процесса

Цель занятия:

научиться производить выборку данных устраняя двойные значения;
определять критерии отбора данных с помощью предложения where;
использовать стандартные операторы Буля, распознаваемыми в SQL (AND, OR, NOT).

Структура учебной базы данных Typdb.

Таблицы **Продавцы** (*Salespeople*), **Заказчики** (*Customers*), **Порядки** (*Orders*) составляют реляционную базу данных

Эти три представленные таблицы в тексте теоретического материала имеют русские имена- **Продавцов**, **Заказчиков** и **Порядков** с указанием в скобках названия таблиц в составе базы данных.

Таблицы приведены как пример ситуации, когда вы будете использовать СУБД для того, чтобы следить за продавцами, их заказчиками и порядками заказчиков.

Столбцы этих таблиц следующие:

Таблица Продавцы (*Salespeople*)

Поле таблицы	Описание
snum	уникальный номер назначенный каждому продавцу (номер служащего)
sname	имя продавца
city	местонахождение продавца (город)
comm	комиссионные продавцов

Таблица Заказчики (*Customers*)

Поле таблицы	Описание
cnum	уникальный идентификатор назначенный каждому заказчику
name	имя заказчика
city	местонахождение заказчика (город)
rating	код указывающий уровень предпочтения

	данного заказчика перед другими. Более высокий номер указывает на большее предпочтение (рейтинг)
snum	номер продавца назначенного этому заказчику (из таблицы Продавцов)

Таблица **Порядки (Orders)**

Поле таблицы	Описание
onum	уникальный но данный каждому приобретению
amt	значение суммы приобретений
odate	дата приобретения
cnum	номер заказчика делающего приобретение (из таблицы Заказчиков)
snum	номер продавца продающего приобретение (из таблицы Продавцов)

Первый столбец каждой таблицы содержит номера чьи значения различны для каждой строки. Это первичные ключи таблиц. Некоторые из этих номеров также показаны в столбцах других таблиц В этом нет ничего неверного. Они показывают связь между строками, которые используют значение принимаемое из первичного ключа, и строками где это значение используется в самом первичном ключе.

Например, поле *snum* в таблице *Заказчиков* указывает, какому продавцу назначен данный заказчик. Номер поля *snum* связан с таблицей *Продавцов*, которая дает информацию об этих продавцах. Очевидно, что продавец, которому назначены заказчики должен уже существовать, т.е. значение *snum* из таблицы *Заказчиков* должно быть представлено в таблице *Продавцов*. Если это так, то можно говорить, что система находится в состоянии справочной целостности.

Выполните задание

Напишите следующие запросы:

1. Запрос, позволяющий вывести номер порядка, сумму и дату для всех строк из таблицы *Порядков*.
2. Запрос, который вывел бы все строки из таблицы *Заказчиков* для которых номер продавца равен 1001.
3. Запрос, который вывел бы таблицу со столбцами в следующем порядке: *city, sname, snum, comm*.
4. Напишите команду **SELECT**, которая вывела бы оценку (*rating*), сопровождаемую именем каждого заказчика в Донецке.
5. Запрос, который вывел бы значения *snum* всех продавцов в текущем порядке из таблицы *Порядков* без каких бы то ни было

повторений.

6. Запрос, который может дать вам все порядки со значением суммы выше чем 1000 рублей.

7. Запрос, который может выдать вам поля sname и city для всех продавцов в Лондоне с комиссионными выше 10.

8. Запрос к таблице *Заказчиков* чей вывод может включить в заказчиков с оценкой <100, если они не находятся в Риме.

9. Что может быть выведено в результате выполнения запроса:

```
SELECT *
FROM Orders
WHERE (amt < 1000 OR
NOT (odate = 10/03/1990 AND cnum > 2003));
```

10. Что может быть выведено в результате выполнения запроса:

```
SELECT *
FROM Orders
WHERE NOT ((odate = 10/03/1990 OR snum > 1006)
AND amt >= 1500);
```

11. Как можно проще переписать запрос:

```
SELECT snum, sname, city, comm
FROM Salespeople
WHERE (comm > 12 OR comm < 15);
```

12. Два запроса, которые могли бы вывести все порядки на 3 или 4 октября 1990 года.

13. Запрос, который выберет всех заказчиков обслуживаемых продавцами Peel или Notika.

14. Запрос, который может вывести всех заказчиков, чьи имена начинаются с буквы попадающей в диапазон от A до L (от K до L).

15. Запрос, который выберет всех пользователей чьи имена начинаются буквы K.

16. Запрос, который выберет все порядки имеющие нулевые значения или NULL в поле amt(сумма).

Отчет :

- Выполненные задания сдайте преподавателю.

Окончание занятия:

Завершите сеанс.

Лабораторная работа № 9-10. Поиск в MySQL. Поиск регулярных выражений

Цель работы: научиться осуществлять поиск данных, используя операторы LIKE, NOT LIKE, RLIKE, NOT RLIKE

Вы научитесь:

- производить соединение с базой данных
- создавать запросы для поиска информации в БД, расположенной на удаленном сервере

Начало занятия:

Для выполнения практической работы вам потребуется: программа СУБД MySQL и клиентское приложение HeidiSQL

Выполнить следующие задания:

Используя параметры соединения (узнать у преподавателя), соединиться с БД Shop. Выполнить запросы по номеру варианта.

Вариант 1. Номер компьютера 1,5,9,13,17,21,25,29

Напишите запрос, который бы вывел все записи из таблицы **products**, где в названии товара будет встречаться слово Pentium

Напишите запрос, который должен выводить материнские платы дороже 2000

Напишите запрос, который выберет всех покупателей чьи фамилии начинаются с буквы I

Напишите запрос, который бы вывел все записи таблицы **products**, поля **name** которых содержат в своем составе подстроки GIGABYTE и Pentium

Напишите запрос, выводящий всех покупателей, чьи номера от 1 до 5 (используйте диапазон в классах символов)

Напишите запрос, который бы выводил цену товара и его количество в магазине, произведя полнотекстовый поиск слова Pentium по полям **name** и **description**

Напишите запрос, который возвращает все записи, в состав которых входит слово Материнская плата, но из выборки исключаются записи, содержащие слово ASUSTEK (используйте логический режим)

Вариант 2. Номер компьютера 2,6,10,14,18,22,26,30

Напишите запрос, который бы вывел все записи из таблицы **products**, где в названии товара будет встречаться слово Asustek

Напишите запрос, который должен выводить винчестеры дороже 3000

Напишите запрос, который выберет всех покупателей чьи фамилии начинаются с буквы K

Напишите запрос, который бы вывел все записи таблицы **products**, поля **name** которых содержат в своем составе подстроки SAPPHIRE и Seagate

Напишите запрос, выводящий всех покупателей, чьи номера от 1 до 5 (используйте диапазон в классах символов)

Напишите запрос, который бы выводил цену товара и его количество в

магазине, произведя полнотекстовый поиск слова Socket-478 по полям **name** и **description**

Напишите запрос, который возвращает все записи, в состав которых входит слово Видеокарта, но из выборки исключаются записи, содержащие слово GIGABYTE (используйте логический режим)

Вариант 3. Номер компьютера 3,7,11,15,23,27,31

Напишите запрос, который бы вывел все записи из таблицы **products**, где в названии товара будет встречаться слово Samsung

Напишите запрос, который должен выводить процессоры дороже 2500

Напишите запрос, который выберет всех покупателей чьи фамилии начинаются с буквы N

Напишите запрос, который бы вывел все записи таблицы **products**, поля **name** которых содержат в своем составе подстроки GIGABYTE и Seagate

Напишите запрос, выводящий всех покупателей, чьи номера от 1 до 5 (используйте диапазон в классах символов)

Напишите запрос, который бы выводил цену товара и его количество в магазине, произведя полнотекстовый поиск слова 256MB по полям **name** и **description**

Напишите запрос, который возвращает все записи, в состав которых входит слово Винчестер, но из выборки исключаются записи, содержащие слово Maxtor (используйте логический режим)

Вариант 4. Номер компьютера 4,8,12,16,20,24,28,32

Напишите запрос, который бы вывел все записи из таблицы **products**, где в названии товара будет встречаться слово Maxtor

Напишите запрос, который должен выводить оперативную память дороже 1500

Напишите запрос, который выберет всех покупателей чьи фамилии начинаются с буквы L

Напишите запрос, который бы вывел все записи таблицы **products**, поля **name** которых содержат в своем составе подстроки SAPPHIRE и Pentium

Напишите запрос, выводящий всех покупателей, чьи номера от 1 до 5 (используйте диапазон в классах символов)

Напишите запрос, который бы выводил цену товара и его количество в магазине, произведя полнотекстовый поиск слова DDR-400 по полям **name** и **description**

Напишите запрос, который возвращает все записи, в состав которых входит слово Материнская плата, но из выборки исключаются записи, содержащие слово Gigabyte (используйте логический режим)

Отчет :

- Выполненные задания сдайте преподавателю.

Лабораторная работа № 11-12. Полнотекстовый поиск. Логический режим поиска

Цель работы: научиться осуществлять поиск данных, используя операторы MATCH() AGAINST(), IN BOOLEAN MODE

Вы научитесь:

- производить соединение с базой данных
- создавать полнотекстовые индексы
- создавать запросы для полнотекстового поиска информации в БД, расположенной на удаленном сервере

Начало занятия:

Для выполнения практической работы вам потребуется: программа СУБД MySQL и клиентское приложение HeidiSQL.

Выполните соединение с сервером **stud.scc** под пользователем **xxpovgg**, пароль **xxpovgg**, база данных **xxpovgg**, где **xxpovgg** ваше сетевое имя.

Выполнить следующие задания:

1. Создайте в БД **xxpovgg** четыре таблицы, используя следующий SQL-скрипт:

```
CREATE TABLE catalogs (  
id_catalog int(11) NOT NULL auto_increment,  
name varchar(255) default NULL,  
PRIMARY KEY (id_catalog));  
CREATE TABLE orders(  
id_order int(11) NOT NULL auto_increment,  
id_user int(11) NOT NULL default '0',  
ordertime datetime NOT NULL default '0000-00-00 00:00:00',  
number int(11) NOT NULL default '0',  
id_product int(11) NOT NULL default '0',  
PRIMARY KEY (id_order),  
KEY id_product (id_product),  
KEY id_user (id_user));  
CREATE TABLE products (  
id_product int(11) NOT NULL auto_increment,  
name varchar(255) NOT NULL default '',  
price decimal(7,2) default '0.00',  
count int(11) default '0',  
mark float(4,1) NOT NULL default '0.0',  
description text,  
id_catalog int(11) NOT NULL default '0',  
PRIMARY KEY (id_product),  
KEY id_catalog (id_catalog));  
CREATE TABLE users (  
id_user int(11) NOT NULL auto_increment,  
surname varchar(255) default NULL,  
patronymic varchar(255) default NULL,
```

```

name varchar(255) default NULL,
phone varchar(12) default NULL,
email varchar(255) default NULL,
url varchar(255) default NULL,
status enum('active','passive','lock','gold') default 'active',
PRIMARY KEY (id_user));
INSERT INTO catalogs (id_catalog, name) VALUES
(1,'Processory'),
(2,'Materinskie platy'),
(3,'Videoadaptery'),
(4,'Jestkie diski'),
(5,'Operativnaya pamyat');
INSERT INTO orders (id_order, id_user, ordertime, number, id_product)
VALUES
(1,3,'2005-01-04 10:39:38',1,8),
(2,6,'2005-02-10 09:40:29',2,10),
(3,1,'2005-02-18 13:41:05',4,20),
(4,3,'2005-03-10 18:20:00',1,20),
(5,3,'2005-03-17 19:15:36',1,20);
INSERT INTO products (id_product, name, price, count, mark, description,
id_catalog) VALUES
(1,'Celeron 1.8',1595,10,3.6,'Processor CeleronR 1.8GHz, 128kb, 478-PGA,
400Mhz, OEM 0.18',1),
(2,'Celeron 2.0GHz',1969,2,3.7,'Processor CeleronR 2.0GHz, 128KB, 478-PGA,
400MHz, OEM ',1),
(3,'Celeron 2.4GHz',2109,4,3.9,'Processor CeleronR 2.4GHz, 128kb, 478-PGA,
400Mhz, OEM ',1),
(4,'Celeron D 320 2.4GHz',1962,1,4.1,'Processor CeleronR D 320 2.4GHz, 256kb,
478-PGA, 533Mhz, OEM',1),
(5,'Celeron D 325 2.53GHz',2747,6,4.1,'Processor CeleronR D 325 2.53GHz,
256kb, 478-PGA, 533Mhz, OEM ',1),
(6,'Celeron D 315 2.26GHz',1880,6,4.1,'Processor CeleronR D 315 2.26GHz,
256kb, 478-PGA, 533Mhz, OEM ',1),
(7,'Intel Pentium 4 3.2GHz',7259,5,4.5,'Processor IntelR PentiumR4 3.2GHz, 1Mb,
478-PGA, 800Mhz, Hyper-Threading, OEM ',1),
(8,'Intel Pentium 4 3.0GHz',6147,1,4.6,'Processor IntelR PentiumR4 3.0GHz,
512Kb, 478-PGA, 800Mhz, Hyper-Threading, OEM ',1),
(9,'Intel Pentium 4 3.0GHz',5673,12,4.5,'Processor IntelR PentiumR4 3.0GHz,
1Mb, 478-PGA, 800Mhz, Hyper-Treading, OEM ',1),
(10,'Gigabyte GA-8I848P-RS',1896,4,3.9,'Materinskaya plata SOCKET-478
Gigabyte GA-8I848P-RS i848, (800Mhz), DDR, AGP 8x, ATA100, SATA ,
Sound 6ch, USB2.0, ATX',2),
(11,'Gigabyte GA-8IG1000',2420,2,3.8,'Materinskaya plata SOCKET-478
Gigabyte GA-8IG1000
i865g,FSB800/533/400,2chDDR400/333/266(4ñëïòà),Video,AGP,5PCI,ATA-

```

100,S-ATA',2),
 (12,'Gigabyte GA-8IPE1000G',2289,6,3.7,'Materinskaya plata Socket-478 Gigabyte GA-8IPE1000G i865PE(800/533/400Mhz),2ch400/333/266DDR,PCI/AGP,U-100,AC'97,Lan(1Gb),S-ATA,USB 2.0, ATX',2),
 (13,'Asustek P4C800-E Delux',5395,4,4.1,'Materinskaya plata Socket-478 Asustek P4C800-E Delux i875P,FSB800/533Mhz,2chDDR400/333,AGP,6PCI,iEEE1394, Raid, U-133,S-ATA, AC'97, Lan(1000), ATX',2),
 (14,'Asustek P4P800-VM\\L i865G',2518,6,4,'Materinskaya plata Socket-478 Asustek P4P800-VM\\L i865G FSB800/533/400, 2chDDR400/333/266(4ñëîòà),AGP,video,3PCI,ATA-100,S-ATA,lan ,M-ATX',2),
 (15,'EpoX EP-4PDA3I',2289,5,4,'Materinskaya plata Socket-478 EpoX EP-4PDA3I i865PE(800Mhz), 2chDDR, PCI/AGP, SATA, Lan, U-100, RAID, AC'97, LAN, ATX',2),
 (16,'ASUSTEK A9600XT/TD',5156,2,4.7,'Videoadapter ASUSTEK A9600XT/TD 128Mb DDR SDRAM, 2x400MHz DAC, AGP8x, ATI Radeon 9600XT, DVI, TV- out, BOX ',3),
 (17,'ASUSTEK V9520X',1602,6,4,'Videoadapter ASUSTEK V9520X 128Mb DDR SDRAM, 400MHz DAC, AGP 8x, GeForce FX 5200, TV- out, BOX ',3),
 (18,'SAPPHIRE 256MB RADEON 9550',2730,3,3.8,'Videokarta SAPPHIRE 256MB RADEON 9550, TV-out, DVI, OEM ',3),
 (19,'GIGABYTE AGP GV-N59X128D',5886,6,3.6,'Videoadapter GIGABYTE AGP GV-N59X128D FX5900XT OEM ',3),
 (20,'Maxtor 6Y120P0',2456,6,4.5,'Vinchester 120 GB Maxtor 6Y120P0, UDMA-133, 7200rpm, 8MB ',4),
 (21,'Maxtor 6B200P0',3589,4,4,'Vinchester 200 GB Maxtor 6B200P0, UDMA-133, 7200rpm, 8Mb ',4),
 (22,'Samsung SP0812C',2093,5,4,'Vinchester 80 GB Samsung SP0812C, SATA, 7200rpm SpinPoint P80 SerialATA Áóôãðíäý êýø-ïàìýòü 8 MB 7200íá/ìèi Èíòàððôáéñ Serial ATA 1.0',4),
 (23,'Seagate Barracuda ST3160023A',3139,3,4.1,'Vinchester 160 GB Seagate Barracuda ST3160023A, UDMA-100, 7200rpm, 8Mb ',4),
 (24,'Seagate ST3120026A',2468,8,4.2,'Vinchester 120 GB Seagate ST3120026A, UDMA-100, 7200rpm, 8MB ',4),
 (25,'DDR-400 256MB Kingston',1085,20,4.8,'Operativnaya pamyat DDR-400 256MB Kingston ',5),
 (26,'DDR-400 256MB Hynix Original',1179,15,4.6,'Operativnaya pamyat DDR-400 256MB Hynix Original ',5),
 (27,'DDR-400 256MB PQI',899,10,4.2,'Operativnaya pamyat DDR-400 256MB PQI ',5),
 (28,'DDR-400 512MB Kingston',1932,20,4.8,'Operativnaya pamyat DDR-400 512MB Kingston ',5),
 (29,'DDR-400 512MB PQI',1690,12,4.2,'Operativnaya pamyat DDR-400 512MB PQI ',5),

(30,'DDR-400 512MB Hynix',1717,8,4.5,'Operativnaya pamyat DDR-400 512MB Hynix ',5);

INSERT INTO **users** (id_user, surname, patronymic, name, phone, email, url, status) VALUES

(1,'Ivanov','Valeryevich','Alexandr','58-98-78','ivanov@email.ru',NULL,'active'),

(2,'Losev','Ivanovich','Sergey','9057777777','losev@email.ru',NULL,'passive'),

(3,'Simdyanov','Vyacheslavovich','Igor','9056666100','simdyanov@softtime.ru','http://www.softtime.ru/','active'),

(4,'Kuznecov','Valeryevich','Maxim',NULL,'kuznetsov@softtime.ru','http://www.softtime.ru/','active'),

(5,'Nechoroshev','yurevich','Anatolii',NULL,NULL,NULL,'lock'),

(6,'Korneev','Alexandrovich','Alexandr','89-78-36','korneev@domen.ru',NULL,'gold');

1. Создайте в таблице **products** индексы **FULLTEXT** для столбцов **name** и **description**,

в таблице **users** индексы **FULLTEXT** для столбцов **surname**, **name** и **patronymic**, используя оператор ALTER TABLE.

2. Выполните задания по вариантам:

Вариант 1. Номер компьютера 1,5,9,13,17,21,25,29

Напишите запрос, который бы производил полнотекстовый поиск слов Materinskaya plata Gigabyte в таблице **products** по столбцам **name** и **description**.

Напишите запрос, который бы производил полнотекстовый поиск слова Ivan в таблице **users** по столбцам **surname**, **name** и **patronymic**

Напишите запрос, который бы производил полнотекстовый поиск слов ramyat или Videoadapter и обязательно в логическом режиме присутствие слова 256MB в таблице **products** с использованием логических операторов AND, OR, NOT по столбцам **name** и **description**.

Напишите запрос, который бы вывел все записи таблицы **products**, поля **name** которых содержат в своем составе подстроки GIGABYTE и Pentium

Напишите запрос, который бы производил полнотекстовый поиск слова kb в режиме расширения запроса по столбцам **name** и **description** и выводил бы все записи из таблицы **products**

Напишите запрос, выводящий всех покупателей, чьи фамилии от А до І (используйте диапазон в классах символов)

Напишите запрос, который возвращает все записи, в состав которых входит слово Materinskaya plata, но из выборки исключаются записи, содержащие слово ASUSTEK (используйте логический режим)

Вариант 2. Номер компьютера 2,6,10,14,18,22,26,30

Напишите запрос, который бы производил полнотекстовый поиск слов

Videokarta 256MB в таблице **products** по столбцам **name** и **description**.

Напишите запрос, который бы производил полнотекстовый поиск слова Max в таблице **users** по столбцам **surname**, **name** и **patronymic**

Напишите запрос, который бы производил полнотекстовый поиск слов

Vinchester или Videoadapter и обязательно в логическом режиме присутствие слова 256MB в таблице **products** с использованием логических операторов AND, OR, NOT по столбцам **name** и **description**.

Напишите запрос, который бы вывел все записи таблицы **products**, поля **name** которых содержат в своем составе подстроки SAPPHIRE и Seagate

Напишите запрос, который бы производил полнотекстовый поиск слова MHz в режиме расширения запроса по столбцам **name** и **description** и выводил бы все записи из таблицы **products**

Напишите запрос, выводящий всех покупателей, чьи фамилии от I до K (используйте диапазон в классах символов)

Напишите запрос, который возвращает все записи, в состав которых входит слово Videokarta, но из выборки исключаются записи, содержащие слово GIGABYTE (используйте логический режим)

Вариант 3. Номер компьютера 3,7,11,15,23,27,31

Напишите запрос, который бы производил полнотекстовый поиск слов Vinchester Seagate в таблице **products** по столбцам **name** и **description**.

Напишите запрос, который бы производил полнотекстовый поиск слова Valer в таблице **users** по столбцам **surname**, **name** и **patronymic**

Напишите запрос, который бы производил полнотекстовый поиск слов ramuat или Videoadapter и обязательно в логическом режиме присутствие слова 256MB в таблице **products** с использованием логических операторов AND, OR, NOT по столбцам **name** и **description**.

Напишите запрос, который бы вывел все записи таблицы **products**, поля **name** которых содержат в своем составе подстроки GIGABYTE и Seagate

Напишите запрос, выводящий всех покупателей, чьи фамилии от A до L (используйте диапазон в классах символов)

Напишите запрос, который бы производил полнотекстовый поиск слова GHz в режиме расширения запроса по столбцам **name** и **description** и выводил бы все записи из таблицы **products**

Напишите запрос, который возвращает все записи, в состав которых входит слово Vinchester, но из выборки исключаются записи, содержащие слово Maxtor (используйте логический режим)

Вариант 4. Номер компьютера 4,8,12,16,20,24,28,32

Напишите запрос, который бы производил полнотекстовый поиск слов Operativnaya ramuat 512MB в таблице **products** по столбцам **name** и **description**.

Напишите запрос, который бы производил полнотекстовый поиск слова Alex

в таблице **users** по столбцам **surname**, **name** и **patronymic**

Напишите запрос, который бы производил полнотекстовый поиск слов ramyat или Videoadapter и обязательно в логическом режиме присутствие слова 256MB в таблице **products** с использованием логических операторов AND, OR, NOT по столбцам **name** и **description**.

Напишите запрос, который бы вывел все записи таблицы **products**, поля **name** которых содержат в своем составе подстроки SAPPHIRE и исключает Pentium

Напишите запрос, выводящий всех покупателей, чьи фамилии от К до L (используйте диапазон в классах символов)

Напишите запрос, который бы производил полнотекстовый поиск слова **MB** в режиме расширения запроса по столбцам **name** и **description** и выводил бы все записи из таблицы **products**

Напишите запрос, который возвращает все записи, в состав которых входит слово Operativnaya ramyat, но из выборки исключаются записи, содержащие слово Kingston (используйте логический режим)

Отчет :

- Выполненные задания сдайте преподавателю.

Лабораторная работа № 13-14. Создание форума при помощи PHP и MySQL

Цель работы: создать форум, который позволит вам осуществлять добавление и поиск записей, при помощи SQL и PHP.

Выполнить следующее задание:

- Скопируйте по ссылке скрипт, распакуйте его и разместите на сервере в рабочем каталоге.
- Создайте на сервере в своей базе данных следующую таблицу.

Поле	Тип	Ноль	По умолчанию	Дополнительно
m_id	int(255)	NOT NULL	NULL	primary key ; auto_increment
r_id	int(255)		NULL	key
name	varchar(50)	NOT NULL		
subject	varchar(50)	NOT NULL		
msg	text	NOT NULL		
email	varchar(30)		NULL	
m_date	datetime	NOT NULL	0000-00-00 00:00:00	

- Настройте файл config.php из архива.
- Создайте файл message.inc следуя инструкциям.

message.inc

- Подключите файл config.php, т.е. замените необходимые параметры (узнать у преподавателя)
- Создайте функцию connect_db.

```
function connect_db()
{
    global $dbhost,$database_user_name, $database_password;
    $db= //самостоятельно добавьте функцию соединения с базой с
```

параметрами: хост, имя пользователя, пароль

```
    // название переменных описанны в файле config.php
```

```
    return $db;
```

```
}
```

Создайте функцию db_name

```
function db_name()
```

```
{
```

```
//Здесь объявите глобальную переменную database_name
```

```
$db_name = $database_name;
```

```
return $db_name;
}
```

Создайте функцию get_now

```
function get_now()
```

```
{
    $db = connect_db();
    $db_name = db_name();
```

```
// Выберите базу данных при помощи параметров db_name и db
```

```
$sql = "select now() as now";
```

```
$result = // вставьте функцию выполнения запроса с текстом запроса и
параметром db
```

```
$myrow = mysql_fetch_array($result);
```

```
$now = $myrow["now"];
```

```
return $now;
```

```
}
```

- Сохраните этот файл в папку с остальными.
- **Будьте внимательны!!! Сервер чувствителен к регистру, имена файлов задавайте лучше всего в нижнем регистре!!!**
- Исправьте во всех скриптах имя таблицы с board000 на имя своей таблицы.
- Исправьте дизайн по своему усмотрению.
- Добавить несколько тем и сообщений.

Отчет :

1. Создайте отчет по практической в виде html-страницы, на которой разместите 3 скриншота:

- главная страница.
- страница добавления сообщения
- данные базы данных.

2. Сдайте преподавателю выполненные задания.

Лабораторная работа № 15-17. Создание регистрационной формы средствами языка PHP и СУБД MySQL

Вы научитесь:

- создавать Web - страницу.
- получите практические навыки в создании PHP скриптов для обращения к серверу MySQL.

Изучите следующий учебный материал:

1. Почему PHP?

Путешествуя по просторам *Глобальной сети*, не раз встречаешь всевозможные форумы, опросники, разнообразные графики и счетчики систем сбора и обработки статистической информации, которые изменяют свой вид всякий раз, когда пользователь посещает такой сайт снова и снова. Наблюдательному человеку, наверняка приходилось сталкиваться в сети с файлами, которые имеют расширение **.phtml* или **.php** вместо привычных **.htm* или **.html*.

Файлы с расширением **.phpX* (где X- номер версии php) - это HTML-файлы со встроенным в них скрипт-языком PHP (читается как пи-эйч-пи), который интерпретируется и выполняется на сервере. На сегодняшний день PHP, несмотря на свое скромное название (Personal Home Page - персональная домашняя страница), это мощный кросс платформенный набор средств, который располагается на сервере и предназначается для обработки кода, встраиваемого в html-документы. Благодаря этому, появляется возможность создавать динамические Web-страницы. То есть их создают как обычно, используя язык HTML и вставляя в документ, по мере необходимости, PHP - команды.

Файлы, созданные таким образом, хранятся на сервере. Когда посетитель обращается к вашей страничке, PHP обрабатывает встроенные в нее команды и выдает результат браузеру пользователя - точно так же, как это делает CGI-программа, написанная на C или Perl.

Только в отличие от последней, PHP имеет ряд преимуществ:

- создание и отладка скриптов PHP значительно проще, чем отладка и создание скриптов на других языках.
- так как PHP-команды просто вставляются в текст html-документа, отпадает необходимость в различных IDE (интегрированная среда разработки);
- для решения разных специфических задач не нужно писать и отлаживать многочисленные маленькие CGI-программы, что сводит к минимуму время доступа к web-страницам, а также продолжительность разработки страниц и сайта в целом.

Вместе с тем, PHP обладает огромным набором функций и большой гибкостью, которые могут быть значительно расширены с помощью дополнительных внешних библиотек. Может управлять доступом к страницам, генерировать изображения или PDF-документы и тому подобное.

Кроме этого в PHP включена поддержка многих баз данных (databases), что делает написание Web-приложений с использованием БД очень простым. Вот неполный перечень поддерживаемых баз данных: Adabas D, dBase, Empress, FilePro, Informix, InterBase, mSQL, MySQL, Oracle, PostgreSQL, Solid, Sybase, Velocis, Unix dbm.

Вдобавок ко всему PHP понимает протоколы IMAP, SNMP, NNTP, POP3 и даже HTTP, а также имеет возможность работать с сокетами (sockets) и общаться по другим протоколам.

Почему нужно выбирать PHP:

Разработчикам Web-приложений нет необходимости говорить, что web-страницы - это не только текст и картинки. Достойный внимания сайт должен поддерживать некоторый уровень интерактивности с пользователем: поиск информации, продажа продуктов, конференции и т.п. Традиционно все это реализовалось CGI-скриптами, написанными на Perl. Но CGI-скрипты очень плохо масштабируемы. Каждый новый вызов CGI, требует от ядра порождения нового процесса, а это занимает процессорное время и тратит оперативную память. PHP предлагает другой вариант, он работает как часть Web-сервера, и этим самым похож на ASP от Microsoft.

Программы, написанные на PHP, достаточно легко читаемы. Написанный PHP код легко зрительно прочитать и понять, в отличие от Perl-программ. PHP - это великолепный инструмент для создания динамических элементов на Web-сайте. Благодаря тому, что этот язык имеет простой синтаксис и выдвигает достаточно мало требований при использовании, начать работать с ним очень легко, и разработчику не приходится опасаться того, что он увязнет в деталях.

Итак, мы видим, что возможности такого мощного и в то же время доступного скрипт-языка как PHP велики. PHP позволяет легко сохранять и извлекать информацию из различных баз данных: MySQL, Oracle, Microsoft SQL Server и других. Может на лету создавать и модифицировать графические образы, изменять текст с помощью регулярных выражений, улучшать программы с помощью IMAP, LDAP, PDF, SNMP, WDDX, XML и многое другое.

Почему PHP?

Perl - очень хороший, мощный язык, но слишком высоки системные издержки во время вызова программы на каждый запрос страницы, особенно в Windows. Существуют ли альтернативы? В последние годы быстро набирает популярность язык программирования веб-приложений PHP (Personal Home Page, www.php.net). В 1994 году Расмус Лердорф (Rasmus Lerdorf) начал разработку встроенного в HTML языка программирования, исполняющегося на стороне сервера. Со временем к проекту присоединились и другие разработчики, и сейчас PHP - это бурно развивающееся средство программирования, работающее на очень многих серверах в Интернете. Как средство разработки www-приложений PHP сейчас лидирует - вместе с ASP, FrontPage и mod_perl.

Достаточно полно описать язык программирования и сопутствующие инструментальные библиотеки в одной статье невозможно, однако можно постараться: не описывая синтаксис или библиотечные функции, полезно остановиться на особенностях создания скриптов на PHP. А с подробной документацией по PHP можно ознакомиться на сайте www.php.net/manual. **PHP** можно установить в двух вариантах: как отдельный интерпретатор, работающий через интерфейс CGI, или как модуль веб-сервера, встроенный в сам сервер. В последнем случае становятся актуальными все преимущества PHP.

Как и все процедурные языки, PHP можно разделить на собственно язык и библиотеку функций. Существует большое количество инструментальных средств для PHP, интерфейсы ко всем популярным СУБД, почтовым протоколам, к разделяемой памяти, графическим файлам, архивам и множество других инструментов.

2. Основы языка PHP

Скрипт на PHP представляет собой файл, как правило, с суффиксом .phtml, .php3 или .php, который внутри выглядит как обычная страница на HTML. Отличие проявляется только в виде хитрого тега `<?php ?>`, делающего из HTML настоящий язык программирования. Часто применяют сокращенную форму `<? ?>` или, для совместимости с визуальными средствами формирования страниц, `<% %>`. Будем разумно придерживаться традиции и пользоваться тегом `<? ?>`. Внутри этого тега располагается код на языке PHP. Вот тривиальный пример страницы:

```
<? include('header.html' ?>  
<h1>Это текст страницы</h1>  
<? include('footer.html') ?>
```

Посетителю такой страницы будет показан документ с картинками, меню и текстом, красиво упакованным в хитро вложенную таблицу. Отдельные файлы header.html и footer.html будут просто включены в тот документ, что отдается сервером посетителю страницы. Таким образом можно отделить оформление страницы от ее наполнения, сильно облегчая работу по внесению информации на сайт даже неквалифицированным работникам. Текст скрипта выполняется сверху вниз и справа налево, последовательно. Эта последовательность может быть изменена операторами цикла и условными операторами. Например:

```
Тебя зовут  
<? if ($name=='Вася') : ?>  
Вася  
<? else : ?>  
капитан Немо  
<? endif ?>
```

Здесь мы получим текст "Тебя зовут Вася" или "Тебя зовут капитан Немо" в зависимости от значения переменной \$name.

В примере видно, что мы прервали текст тегами PHP. В языке есть несколько функций вывода - echo, printf и др., но приведенный в примере код работает намного быстрее, чем, например, if (\$name=='Вася') echo 'Вася' Перевод строки, идущий после закрытия тега ?>, интерпретатор пропускает. Сделано это для удобства форматирования исходных текстов скриптов.

Переменные в PHP отличаются наличием символа \$ (доллар) перед именем самой переменной. Объявлять их никак не надо, так как они создаются автоматически при присвоении им значения и удаляются при выходе из области действия. Переменные бывают типа строка, число, массив и объект. Некоторые трудности может вызвать нестрогое отличие строк от чисел. Это можно пояснить на примере:

```
<?
$a=5; // Число
$b='6b'; // Строка
$c=$b.$a; // Строка: '6b5'
$d="$a$b"; // Строка: '56b'
$e=$a+$b; // Число: 11,br> $f=$b+$a; // Число: 11
?>
```

Интерпретатор очень вольно приводит типы строка и число друг к другу. Хотя чаще всего это даже удобно. Просто надо об этом помнить и не пытаться сравнивать строку с числом:

```
<?
$a=5;
$b='ой!';
if ($a>$b) echo "1";
if ($b>$a) echo "2";
?>
```

В первом случае сравниваются числа 5 и 0 и результат, естественно, положительный. А во втором случае что сравнивается? Документация на этот счет молчит, а эксперимент показывает, что сравниваться будут тоже числа. Это не строго и не очень красиво, и в условных операторах рекомендуется использовать операторы приведения типа, как в языке Си:

```
<? if ((int)$a < (int)$b) : ?>
```

или

```
<? if ((string)$a < (string)$b) : ?>
```

Синтаксис языка PHP вообще очень похож на синтаксис Си, если бы не символы \$ в начале имени переменных и некоторые вольности с типами и массивами. Как в "Поднятой целине" председатель колхоза английский учил: "... Английские слова такие же, как русские, только в конце шипение какое-то, революшн, например..."

Для изменения типа переменной, в принципе, особенного ничего делать не надо, достаточно просто присвоить ей значение нового типа:

```
>?
$a=5; // Целое число
$a='5'; // Строка
```

```
$a=5.5; // Вещественное число  
$a[0]=5; // Массив  
$a['пять']=5; // Хэш-массив  
?>
```

Последняя строка, например, показывает нам очень популярный и удобный тип данных - хэш-массив (hash array). В принципе, это обычный массив, только индексом у него выступает строка, а не целое число, и никакого приведения типов, о котором я говорил ранее, здесь нет. PHP позволяет создавать многомерные сооружения вида массив хэш-массивов, как, например, `$name[5]['ой'][6] = 'не может быть';`

В принципе, массив - это тоже хэш, только в нем индексами являются строки, соответствующие числам. Почему так хитро? Это еще ничего, а вот, например, фраза `$name[5]=0;` создает массив `$name` из 6 элементов (индексы считаются от 0, как в Си), но только 1 элемент реально существует, остальные просто не определены. На практике это не сильно мешает, помните только, что это не Си и не Паскаль.

Конечно, не очень хорошо, но частенько удобно пользоваться конструкциями `<? if ($test!="") : ?>`

вместо правильной

```
<? if (isset($test) && $test!="") : ?>
```

просто потому, что первая запись короче. В первом случае, если переменная `$test` будет не определена, то ее значение будет как бы пустой строкой.

Логично считать, что "никакая" строка - это тоже пустая. Подобная фраза может вызвать предупреждение о синтаксической ошибке, если администратор включил параноидальный контроль синтаксиса, но это бывает редко. Если вам не лень, то пользуйтесь правильными конструкциями и проверяйте наличие переменной перед обращением к ней. А если лень, то считайте неопределенную переменную пустой строкой, нулем, логической ложью и т. п.

Как-то специально объявлять массив или хэш тоже не надо. Но в некоторых случаях это бывает полезно, например, если ваша функция возвращает массив, то она должна вернуть именно массив, а не неопределенное значение. Это, кстати, относится и к функциям:

```
<?  
function initArray($num) {  
if ($num>0) return $a[$num]=$num;  
return $a;  
}  
?>
```

Эта надуманная функция возвращает массив, если `$num` больше нуля, и неопределенное значение - в противном случае, что затрудняет работу с результатом ее выполнения:

```
<?  
$foo=initArray(0);  
...
```

```
$bar=$foo[0]; // Ошибка, $foo не массив.
```

```
...  
?>
```

Здесь вам поможет функция `array()`, возвращающая массив, в том числе и пустой. Либо стоит сразу инициализировать переменную `$a=array()` или вернуть пустой массив как `return array()`. Первый вариант предпочтительнее. В PHP есть даже некоторая объектность, то есть вы можете создавать классы, методы и т. д. Как правило, классы хороши в больших приложениях или как средство ограничения области видимости переменных. Я бы не рекомендовал очень уж активно ими пользоваться: помните, что объектный подход нынче очень моден, но не стоит из него делать фетиш.

Примеры программирования на PHP:

Как и для скриптов на языке Perl, в Интернете есть множество сайтов, содержащих десятки готовых PHP-программ. Такие ресурсы делятся на два вида: наборы функций или классы, - что отражает идеологический подход авторов к программированию.

Программа на языке PHP, как правило, не живет сама по себе, а применяется для обработки запросов через интерфейс CGI. Даже если интерпретатор PHP встроен в сервер как модуль, с точки зрения самой пользовательской программы, она, эта программа, работает через CGI. А для чего творцам чудесных веб-сайтов нужен CGI? Конечно же, для обработки форм. Именно на скрипт указывает параметр `action`-тега

```
<form action="doit.phtml">.
```

В PHP автору не требуется каким-то особым образом извлекать данные формы. В момент начала выполнения скрипта для вас уже существуют и определены переменные, соответствующие одноименным полям. Если форма имеет вид

```
<form action="doit.phtml">  
<input name=f1>  
<input name=f2>  
<input type=submit name=do value="Поехали">  
</form>
```

то при старте скрипта `doit.phtml` в нем уже определены переменные `$f1`, `$f2` и `$do`. Можно указать имя поля в форме как

```
<select name="sel[]" multiple>
```

```
...
```

```
</select>
```

тогда на момент выполнения скрипта, обрабатывающего эту форму, будет определен массив `$sel`, содержащий выбранные пункты списка `select`.

Технология выполнения:

1. Создадим статичную HTML страницу `add.php`.

Листинг add.php

```
<html>
```

```

<head>
<title>Гостевая книга</title>
</head>
<body>
<form method=post action="add.php3"><br>
<font face=Arial size=2>Ваше имя:</font><br> <input type=text
name="name"><br><br>
<font face=Arial size=2>Ваш E-mail:</font><br> <input type=text
name="email"><br><br>
<font face=Arial size=2>Ваше пожелание:</font> <br>
<textarea rows=15 cols=40 name="text"></textarea><br><br>
<input type=submit name=cmdPost value=Отправить>
<input type=reset name=cmdClear value=Очистить>
</form>
<hr>
</body>
</html>

```

И вот, что получается.

The screenshot shows a Microsoft Internet Explorer window titled 'Гостевая книга - Microsoft Internet Explorer'. The address bar shows 'http://www.mylabs.ru/labs_on_php/gb/s'. The form contains three input fields: 'Ваше имя:' (Your name), 'Ваш E-mail:' (Your E-mail), and 'Ваше пожелание:' (Your wish) which is a large text area. Below these fields are two buttons: 'Отправить' (Send) and 'Очистить' (Clear). At the bottom of the page, it says 'Записей нет' (No records).

Листинг add.php3

2. Теперь после открывающего тэга <body> добавим текст скрипта формирующего файл с пожеланиями:

```

<?
If ($cmdPost!=""){
$fp=fopen("new.txt","a+");
fputs($fp, ">font face=Arial size=2>От: <a
href=\"mailto:$email\">$name</a><br>$text<p><br><hr></font>");

```

```
fclose($fp);
echo "<font face=Arial size=2><b>Спасибо. Ваше сообщение добавлено в
Гостевую книгу.</b></font>";
}
?>
```

Где:

```
If ($cmdPost!="")
{
}
```

является условием проверки "Какая кнопка была нажата". В данном случае проверяется содержимое переменной \$cmdPost (кнопка Отправить). Если содержимое переменной равно "". Тогда выполняется код находящийся внутри условия. Если нет, тогда скрипт завершит свою работу и передаст управление тэгам HTML.

\$fp=fopen("new.txt","a+") - открываем файл new.txt с параметром a+ в каталоге где файл add.php3. Параметр a+ говорит, что мы открываем файл для записи и чтения, причем при открытии файла мы помещаем указатель в конец файла. Существуют еще несколько параметров:

'r' - Открыть только для чтения, помещает указатель на начало файла.

'r+' - Открыть для чтения и для записи, помещает указатель на начало файла.

'w' - Открыть только для записи, помещает указатель на начало файла и очищает все содержимое файла. Если файл не существует, создается новый файл

'w+' - Открыть для чтения и для записи, помещает указатель на начало файла и очищает все содержимое файла. Если файл не существует, создается новый файл.

'a' - Открыть только для записи, помещает указатель на конец файла. Если файл не существует, создается новый файл.

'a+' - Открыть для чтения и для записи, помещает указатель на конец файла. Если файл не существует, создается новый файл.

fputs(\$fp, "код HTML или простой текст") - этот код записывает в файл \$fp (т.е. это new.txt) некий текст или HTML-код.

fclose(\$fp). Данный код закрывает файл new.txt (т.е. \$fp).

```
echo "<font face=Arial size=2><b>Спасибо. Ваше сообщение добавлено в
Гостевую книгу.</b></font>";
```

Команда echo выведет в левом верхнем углу HTML страницы текст находящийся в кавычках.

3. Там, где будут отображаться пожелания мы добавим такой код PHP:

```
<?
if (file_exists("new.txt")){
include("new.txt");
}
?>
```

Данный скрипт выполняет вставку файла new.txt только в том случае, если этот файл существует. В противном случае никакой вставки не будет. Кстати, именно в new.txt будут записываться пожелания посетителей.

Основные сведения:

Эти функции обеспечивают доступ к серверам баз данных MySQL по средствам PHP скриптов `mysql_num_rows`

`mysql_num_rows` -- Получает количество столбцов в результате

Описание:

`int mysql_num_rows(int result);`

`mysql_num_rows()` получает количество столбцов в установленном результате

См. также: `mysql_db_query()`, `mysql_query()` и `mysql_fetch_row()`.
Для обратной совместимости `mysql_numrows()` также может быть использована.

`mysql_close`

`mysql_close` -- закрывает MySQL соединение

Описание

`int mysql_close(int [link_identifier] );`

Возвращает: true при успешном завершении, false при ошибке

`mysql_close()` закрывает связь с базой данных MySQL, ассоциированной с определенным идентификатором связи(`link identifier`).

Если идентификатор связи не определен, используется последняя открытая связь.

Имейте в виду, что это не всегда необходимо, так как непостоянные открытые связи автоматически закрываются в конце выполнения сценария.

`mysql_close()` не закроет устойчивые связи сгенерированные функцией `mysql_pconnect()`.

См. также: `mysql_connect()`, и `mysql_pconnect()`.

`mysql_connect`

`mysql_connect` -- открывает соединение с MySQL сервером

Описание

`int mysql_connect(string [hostname] [:port] , string [username] , string [password] );`

Возвращает: Правильный идентификатор связи MySQL при успешном выполнении, или false при ошибке.

`mysql_connect()` устанавливает соединение с MySQL сервером. Все аргументы дополнительные(опциональны), и если они пропущены, то устанавливаются по умолчанию -('localhost', имя пользователя, который владеет процессом, пустой пароль). Строка "имя сервера"("hostname") также может содержать номер порта(подобно "hostname:port").

В случае, если будет сделан второй вызов `mysql_connect()` с теми же аргументами, никакая новая связь не установится - вместо этого, будет возвращен идентификатор связи уже открытой связи.

Как только выполнение сценария закончится, связь с сервером закроется, если оно не было явно закрыто более ранний вызовом `mysql_close()`.

См. также `mysql_pconnect()`, и `mysql_close()`.

`mysql_create_db`

`mysql_create_db` -- Создает базу данных MySQL

Описание

`int mysql_create_db(string database name, int [link_identifier] );`

`mysql_create_db()` пытается создавать новую базу данных на сервере связанном с определенным идентификатором связи.

См. также: `mysql_drop_db()`. Для совместимости с более ранними версиями может также быть использована функция `mysql_createdb()` .

`mysql_db_query`

`mysql_db_query` -- Посылает MySQL запрос

Описание

`int mysql_db_query(string database, string query, int [link_identifier] );`

Возвращает: Правильный идентификатор результата MySQL в результате запроса, или false при ошибке .

`mysql_db_query()` выбирает базу данных и выполняет запрос на нее. Если дополнительный идентификатор связи не определен, функция попытается найти открытую связь с MySQL сервером и если такая связь не обнаружена, то она попытается создать ее, как если бы функция `mysql_connect()` была вызвана без аргументов.

См. также `mysql_connect()`. Для обратной совместимости может быть также использована функция `mysql()`.

`mysql_fetch_array`

`mysql_fetch_array` -- Выбрать результат как ассоциативный массив

Описание

`array mysql_fetch_array(int result);`

Возвращает результат строки запроса как массив, или false при отсутствии результата.

`mysql_fetch_array()` является расширенной версией функции

`mysql_fetch_row()`. Кроме того, что она сохраняет данные в пронумерованных элементах, результатного массива, она также сохраняет ассоциативную связь, используя имена полей как ключи.

Если два или более столбцов имеют одинаковые имена, то связку ключ/значение получает последний из них. Чтобы получить доступ к другому столбцу(-цам) с таким же именем, вы должны использовать индексы или задать псевдонимы(alias) для этих столбцов.

`select t1.f1 as foo t2.f1 as bar from t1, t2`

Важно заметить, что использование `mysql_fetch_array()` НЕСУЩЕСТВЕННО медленнее, чем использование `mysql_fetch_row()`, пока это обеспечивает необходимую задачу.

Более подробно см. `mysql_fetch_row()`

Пример 1.

`mysql fetch array`

`<?php mysql_connect($host,$user,$password);$result =`

`mysql_db_query("database","select * from table");while($row =`

```
mysql_fetch_array($result)) { echo $row["user_id"]; echo  
$row["fullname"];}mysql_free_result($result);?>
```

mysql_select_db

mysql_select_db -- Выбирает Базу Данных MySQL

Описание

```
int mysql_select_db(string database_name, int [link_identifier] );
```

Возвращает: true при успешном выполнении , false при ошибке

mysql_select_db() устанавливает текущую активную базу данных в сервере, которая связывается с определенным идентификатором связи. Если не определен идентификатор связи, принимается последняя открытая связь. Если связь не открыта, функция попытается установить связь, как если бы была вызвана функция mysql_connect().

Каждый последующий вызов mysql_query() будет произведен в активной базе данных.

См. также: mysql_connect(), mysql_pconnect(), и mysql_query()

Для обратной совместимости mysql_selectdb() также может быть использована.

Лабораторная работа № 18 . Создание регистрационной формы с использованием языка PHP и СУБД MySQL

Задание:

1. Создать таблицу **users**, в которой будут храниться данные о пользователях.

Запрос на создание таблицы users:

```
CREATE TABLE users(  
id INT(10) primary key not null auto_increment,  
login VARCHAR(250) NOT NULL,  
pass VARCHAR(250) NOT NULL,  
name VARCHAR(250),  
surname VARCHAR(250),  
age INT(4)  
)
```

2. Создать следующие файлы (Вы можете воспользоваться **Блокнот**):
файл в котором хранятся все переменные, используемые в скрипте (**_vars.php**)

*Создаем файл **_vars.php**, в котором будут храниться все переменные, используемые в скрипте (значения переменных узнать у преподавателя)*

Листинг **vars.php**:

```
<?php  
$HTTP_SERVER_VARS=$GLOBALS[HTTP_SERVER_VARS];  
$PHP_SELF=$HTTP_SERVER_VARS[PHP_SELF];  
if (basename($PHP_SELF)=="_vars.php") { echo "Этот файл нельзя просматривать!"; exit; }  
  
$host="192.168.16.50";  
$user="07pov12";  
$passw="07pov12";  
  
$db="07pov12";  
  
$table="users";  
?>
```

- файл с формой для входа и ссылками (**index.php**)

*Создаем страницу **index.php**, на которой будет размещаться форма для входа пользователей.*

Листинг **index.php**:

```

<html>
<head>
  <title>Форма для входа</title>
</head>
<body>
<form action="enter.php" method="POST">
<h1><center>Введите логин и пароль</center></h1>
<table border=0 cellspacing=0 cellpadding=2 align=center>
  <tr>
    <td><b><font face="verdana" color="#F1F406">Логин:</font></b></td>
    <td><input type="text" name="login" value=""></td>
  </tr>
  <tr>
    <td><b><font face="verdana" color="#F1F406">Пароль:</font></b></td>
    <td><input type="password" name="pass" value=""></td>
  </tr>
  <tr>
    <td colspan=2 align=right><input type="submit" name="enter" value=" Войти "></td>
  </tr>
</table>
</form>
<center><a href="reg.php"><b><font face="verdana" color="#00F0FF">Регистрация</font></b></a> |
<a href="index.php?view=on"><b><font face="verdana" color="#00F0FF">Просмотр пользователей</font></b></a></font></b></center>
<br><br><br>
</body>
</html>

<?php
include "_vars.php"; // Подключение файла с переменными
if($ _GET[view]=="on")
// Ссылка 'Просмотр пользователей' передает на страницу index.php одну переменную методом GET.
//Запись href="index.php?view=on" означает
// что переменная view равна значению 'on' и это условие проверяет это.
{
mysql_connect($host,$user,$passw) or die(mysql_erro()."<br>".mysql_error());
// Соединение с mysql сервером. Функция mysql_erro() - выводит код ошибки mysql, а mysql_error() - текст самой ошибки,
// таким образом вам
// не придется ломать голову над причиной неудачной попытки соединения, mysql за вас напишет почему произошла ошибка.
//В большинстве случаев
// рекомендуется применять именно эти две функции, а не обычное текстовое сообщение, такие как 'не могу соединиться
//с mysql сервером'
mysql_select_db($db) or die(mysql_erro()."<br>".mysql_error()); // Выбор базы данных
$vir=mysql_query("SELECT login, name, surname, age FROM ".$table." ");
// Выполнение запроса на выбор всех пользователей, зарегистрированных в базе данных
echo "<br><table border=0 align=center cellpadding=2>";
// Поскольку информация о пользователях корректнее всего отображать в таблице, рисуем начало таблицы
for($i=0;$i<mysql_num_rows($vir);$i++) // Цикл, который выводит всех пользователей
{
  $rez=mysql_fetch_array($vir);
  // Занесение в переменную $rez результата обработки запроса. Функция mysql_fetch_array() создает ассоциативный массив
  //результата
  // запроса, где ключами являются названия полей, а элементами массива сами данные
  echo "<tr><td bgcolor=";
  if(($i % 2)==0) echo "\"#B9D0F7\""; // Для удобства просмотра, каждый нечетный пользователь будет отображаться синим
  //цветом...
  if(($i % 2)!=0) echo "\"#B8F4B9\""; // ... а каждый четный зеленым
  echo "><b>".$rez[login]."</b> (".$rez[surname]." ".$rez[name]." ".$rez[age].") - ".(date("Y")-$rez[age])." лет </td></tr>";
  // Вывод самих сведений о пользователе
}
echo "</table>";
}
?>

```

- файл с формой регистрации и скриптом регистрирующим новых пользователей в базе данных (**reg.php**)

*Создаем файл **reg.php**, который будет регистрировать новых пользователей в базе данных*

Листинг reg.php:

```

<html>
<head>
<title>Регистрация нового пользователя</title>
</head>
<body>
<h1><center>Регистрация нового пользователя</center></h1>
<form action="reg.php" method="POST">
<table border=0 cellpadding=2 align=center>
<tr>
<td><sup>*</sup> Логин</td>
<td><input type="text" name="login" value="user"></td>
</tr>
<tr>
<td><sup>*</sup> Пароль</td>
<td><input type="password" name="pass1" value=""></td>
</tr>
<tr>
<td><sup>*</sup> Подтвердите пароль</td>
<td><input type="password" name="pass2" value=""></td>
</tr>
<tr>
<td><sup>*</sup> Имя</td>
<td><input type="text" name="name" value=""></td>
</tr>
<tr>
<td><sup>*</sup> Фамилия</td>
<td><input type="text" name="surname" value=""></td>
</tr>
<tr>
<td><sup>*</sup> Год рождения</td>
<td><input type="text" maxlength=4 size=3 name="age" value=""></td>
</tr>
<tr>
<td colspan=2 align=right><input type="submit" name="add" value="Добавить"></td>
</tr>
</table>
<br>
<center>Поля отмеченные знаком <sup>*</sup> обязательны для заполнения</center>
</form>
</body>
</html>
<?php
include "_vars.php"; // Подключаем файл с переменными
if(isset($_POST[add])) // Если была нажата кнопка 'Добавить'
{
// Создаем переменные и помещаем в них переданные с формы данные
$login=$_POST[login];
$pass1=$_POST[pass1];
$pass2=$_POST[pass2];
$name=$_POST[name];
$surname=$_POST[surname];
$age=$_POST[age];
echo "<center>Регистрация нового пользователя ";
if($pass1!=$pass2) { echo "<b>не прошла успешно. <br>Ошибка!</b> Пароли не совпадают. Попробуйте еще раз"; exit; }
// Если пароли не совпадают то выводится сообщение об ошибке и скрипт завершает свою работу ключевым словом exit

if(trim($login)!=" && trim($pass1)!=" && trim($pass2)!=" && trim($name)!=" && trim($surname)!=" && trim($age)!=")
// Если все поля были заполнены. Функция trim() отсекает пробелы в начале и в конце строки, для того чтобы нельзя
// было вписать в поле строку
// состоящую из пробелов, так как php считает что если в строке есть хотя бы один пробел, то она уже не пуста
{
mysql_connect($host,$user,$passwd) or die("Не могу подключиться к серверу mysql");
// Соединение с сервером mysql. В качестве аргументов функции выступают переменные хранящиеся в файле _vars.php
// Конструкция or die указывает на то, какое сообщение об ошибке должно быть выведено в случае неудачного
// подключения к серверу
mysql_select_db($db) or die("Не могу выбрать базу $db"); // Выбор текущей базы данных
$queryText="INSERT INTO $table(login,pass,name,surname,age) VALUES('$login','$pass1','$name','$surname','$age')";
// Помещаем в переменную $queryText текст запроса на добавление нового пользователя
@mysql_query($queryText) or die("Ошибка при выполнении запроса");
// Выполняем sql-запрос, функцией mysql_query(), где строкой запроса выступает переменная $queryText. Символ @ перед функцией
// mysql_query() означает что сообщение об ошибке не будет выведено, а будет выведено именно то которое указано в конструкции
// or die
echo "прошла успешно.<br><br>Пользователь <b>".$surname." ".$name." ".$age."</b> года рождения, ";
echo "с логином <b>".$login."</b> был зарегистрирован";
echo "<br><a href='index.php'>Нажмите сюда чтобы войти</a>";
// Сообщение об успешной регистрации и вывод данных о пользователе. Для того чтобы напечатать символ " в php используется
// слэш \"
}
else { echo "<b>не прошла успешно. <br>Ошибка!</b> Одно или несколько полей не были заполнены"; exit; }
// В случае если одно или несколько полей не были заполнены, то выводится сообщение об ошибке и скрипт завершает свою работу
}
?>

```

- скрипт авторизации пользователя и входа под своим логином и паролем (**enter.php**)

*Создаем файл **enter.php**, по средствам которого пользователи будут входить в свои аккаунты*

*Листинг **enter.php**:*

```

<?php
include "_vars.php";
if (isset($_POST[enter]))
{
    if($_POST[login]!=" " && $_POST[pass]!=" ")
    {
        $login=$_POST[login];
        $pass=$_POST[pass];

        mysql_connect($host,$user,$passw) or die("Не могу подключиться к серверу");
        mysql_select_db($db) or die("Не могу выбрать базу");
        $vir=mysql_query("SELECT * FROM ".$table." WHERE login=\"".$login "\" AND pass=\"".$pass "\"");
        $rez=mysql_fetch_array($vir);

        if($rez['login']==$login && $rez['pass']==$pass)
        {
            $date=date("Y");
            echo "<h1>Доступ разрешен. Добро пожаловать ".$rez[name]. "</h1>";
            echo "<center>Информация:</center><br>";
            echo "Логин: <b>" . $rez[login] . "</b><br>";
            echo "Имя: <b>" . $rez[name] . "</b><br>";
            echo "Фамилия: <b>" . $rez[surname] . "</b><br>";
            echo "Год рождения: <b>" . $rez[age] . "</b><br>";
            echo "Возраст: <b>" . ($date-$rez[age]) . "</b><br>";
        }
        else { echo "Неправильно введены логин или пароль"; exit;}
    }
    else {echo "<center>Логин или пароль не были указаны</center>"; exit;}
}
else {echo "Доступ запрещен"; exit;}
??

```

В результате выполнения практических работ пользователь получает три независимых страницы, при помощи которых происходит работа с сервером баз данных *MySQL*.

Скопируйте вышесозданные файлы каталог на сервер (параметры узнать у преподавателя)

Для этого: откройте *FAR*, перейдите на *FTP*, создайте новое подключение (*Shift+F4*) и поставьте галочку "Спрашивать пароль при подключении". Если у Вас это не получается - позвоните преподавателя!

Для того чтобы файлы можно было запускать, необходимо установить их атрибуты, для этого:

- Установите курсор на файле
- Нажмите комбинацию клавиш **CTRL+A**, в появившемся окне поставьте крестики под колонками **X**:



- Нажмите **продолжить**, проведите эту операцию со всеми файлами ***.php**.

Для проверки результата в строке адреса InternetExplorer наберите необходимую ссылку.

Отчет :

- Выполненные задания сдайте преподавателю.

Окончание занятия:

Завершите сеанс.

**МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное
образовательное учреждение высшего образования
«СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»**

МЕТОДИЧЕСКИЕ УКАЗАНИЯ

к самостоятельной работе по дисциплине
«Базы данных»
для студентов направления
09.03.04 «Программная инженерия»

Ставрополь 2024

ВВЕДЕНИЕ

Целью изучения дисциплины «Базы данных» является формирование у студентов системы теоретических и практических представлений о современных подходах, принципах и методах проектирования баз данных в распределенных, клиент-серверных информационных системах.

Задачи изучения дисциплины:

- изучить вводные вопросы, касающиеся основных положений и принципов организации баз данных в распределенных информационных системах,
- изучить основные клиент-серверные технологии,
- ознакомиться с современными клиент-серверными СУБД MySQL, ORACLE, научиться использовать основные объекты этих СУБД и реализовывать запросы.

Пособие содержит дидактически и методически обработанный и систематизированный материал, раскрывающий основное содержание учебной дисциплины «Базы данных».

1. Организационно-методические рекомендации по освоению дисциплины

Самостоятельная работа студентов является важнейшим условием формирования научного способа познания. Она проводится накануне каждого семинарского (практического, лабораторного) занятия и включает подготовку реферата или сообщения, а также выполнения лабораторной работы.

Самостоятельные занятия (СЗ) являются одной из активных форм обучения.

Самостоятельные занятия по дисциплине «Управление данными в сетях» имеют целью:

- закрепить и углубить знания, полученные студентами на лекциях и в процессе лабораторных занятий;
- привить практические навыки при проектировании распределенных баз данных.

Самостоятельные занятия проводятся в специализированных аудиториях, оборудованных СВТ и возможностью пользования Интернет-ресурсами, в библиотеке.

Предлагаемые методические рекомендации содержат информацию для студентов, необходимую при подготовке и проведении лабораторных занятий по дисциплине «Управление данными в сетях».

3. Методические указания по выполнению самостоятельной работы студентов

Методические указания должны включать следующие разделы:

- цель работы;
- задание, которое должно быть выполнено студентом в результате проведения самостоятельной работы;
- варианты индивидуальных заданий;
- основные теоретические положения, необходимые для выполнения задания, они должны быть краткими и содержать ссылки на литературу, в которой эти положения изложены в объеме, достаточном для выполнения самостоятельной работы;
- этапы выполнения задания с указанием конкретных сроков выполнения каждого из этапов и всего задания в целом;
- требования к оформлению графической и текстовой части самостоятельной работы;
- пример выполнения одного из вариантов задания и оформления отчета;
- библиографический список использованных источников

4. Содержание самостоятельной работы

<u>Название раздела (темы)</u>	<u>Цель определяе тся, исходя из компетент ностного подхода</u>	<u>Фор ма контр оля СРС</u>	<u>Задания для СРС</u>	<u>Требовани я представи ю и оформлени результатов СРС</u>	<u>Рекомендуе мая литература</u>
Введение в теорию баз данных в распределен ных системах. Архитектур а распределен	изучить теорию баз данных в распреде ленных системах, архитектур у распреде	ин диви дуаль ное собес едова ние	сформулир овать и письменно ответить на вопросы для контроля владения компетенция ми данного	письменн о ответить на вопросы для контроля владения компетенция ми данного практическо го занятия	1. Агальцов В.П. Базы данных. В 2-х т. Т. 2. Распределенны е и удаленные базы данных: Учебник / В.П. Агальцов. – М.:

ных баз данных	нных баз данных		практического занятия		Высшее образование. – 2013. – 272с. ISBN 978-5-8199-0394-0;
Серверные СУБД	изучить серверные СУБД	индивидуально е собеседование	сформулировать и письменно ответить на вопросы для контроля владения компетенция ми данного практического занятия	письменно о ответить на вопросы для контроля владения компетенция ми данного практического занятия	2. Цехановский В.В., Чертовской В.Д. – Управление данными: Учебник [Электронный ресурс] – СПб.: Издательство «Лань», 2015. – 432с. URL: http://e.lanbook.com/view/book/65152/page11/
Введение в СУБД MySQL. Создание БД и таблиц в СУБД MySQL	Рассмотреть и проанализировать СУБД MySQL. Команды языка SQL для создания БД и таблиц в СУБД MySQL	индивидуально е собеседование	сформулировать и письменно ответить на вопросы для контроля владения компетенция ми данного практического занятия	письменно о ответить на вопросы для контроля владения компетенция ми данного практического занятия	3. Карвин Билл Программирование баз данных SQL. Типичные ошибки и их устранение. – М.: РИД ГРУПП. – 2012. – 336с. ISBN: 978-5-4252-0510-0
Основные команды пользователя программ MySQL. Выборка информации. Создание сложных запросов	Рассмотреть и проанализировать основные команды пользователей программ MySQL	индивидуально е собеседование	сформулировать и письменно ответить на вопросы для контроля владения компетенция ми данного практического занятия	письменно о ответить на вопросы для контроля владения компетенция ми данного практического занятия	

Репликация в MySQL	Рассмотреть и проанализировать репликацию в MySQL	индивидуально собеседован	сформулировать и письменно ответить на вопросы для контроля владения компетенция ми данного практического занятия	письменно ответить на вопросы для контроля владения компетенция ми данного практического занятия	
Хранимые процедуры. Триггеры. Представления	Рассмотреть и проанализировать хранимые процедуры, триггеры, представления	индивидуально собеседован	сформулировать и письменно ответить на вопросы для контроля владения компетенция ми данного практического занятия	письменно ответить на вопросы для контроля владения компетенция ми данного практического занятия	

5. План-график выполнения СРС

№ №	Название контрольной точки	Срок ее сдачи
1	Самостоятельное изучение темы «Введение в теорию баз данных в распределенных системах. Архитектура распределенных баз данных»	4
2	Самостоятельное изучение темы «Серверные СУБД»	5
3	Самостоятельное изучение темы «Введение в СУБД MySQL. Создание БД и таблиц в СУБД MySQL»	6
4	Самостоятельное изучение темы «Основные команды пользовательских программ MySQL. Выборка информации. Создание сложных запросов»	7
5	Самостоятельное изучение темы «Репликации в MySQL»	8

6	Самостоятельное изучение темы «Хранимые процедуры. Триггеры. Представления»	9
7	Подготовка к лабораторным работам	10

6. Организация контроля знаний студентов

6.1. Формы контроля знаний студентов

Контроль и оценка знаний, умений и навыков студентов осуществляется на лабораторных работах, консультациях, при сдаче экзамена. В ходе контроля знаний преподаватель оценивает понимание студентом содержания дисциплины «Управление данными в сетях», его способность проектировать эффективно функционирующую базу данных для решения конкретных задач.

Контроль знаний студентов может осуществляться в следующих формах:

- текущий контроль знаний;
- итоговый контроль знаний.

Текущий контроль знаний студентов имеет целью:

- дать оценку работы каждого студента по усвоению им учебного материала, выявить недостатки в его подготовке и оказать практическую помощь в их устранении;

Основными формами текущего контроля знаний студентов являются:

- устный контрольный опрос;
- письменный контрольный блиц-опрос;
- защита лабораторной работы;
- промежуточное тестирование;
- проверка конспектов лекций.

Устный контрольный опрос студентов проводится на лекциях (и лабораторных занятиях). По его результатам преподаватель оценивает качество подготовки студента к занятию.

Письменный контрольный блиц-опрос студентов проводится в течение пяти минут на лабораторных занятиях путем письменного ответа их на пять вопросов, заданных преподавателем. Результаты его проведения отмечаются в журнале. На лабораторных занятиях знания и практические навыки студентов оцениваются по 4-балльной системе. Полученные оценки выставляются в журнале.

При проверке конспектов лекций дается анализ качества их ведения. Отмечаются допущенные ошибки, в рецензии преподавателя оценивается качество конспектирования учебного материала, даются рекомендации по улучшению качества конспектирования лекционного материала.

6.2. Рекомендации по подготовке к экзамену

Подготовка к зачету начинается с начала изучения дисциплины.

Необходимо посещать все лекции и лабораторные занятия.

Формой промежуточной аттестации по дисциплине «Управление данными в сетях» является зачет (накопительная форма отчетности), который выставляется по итогам работы студента в 8 семестре.

7. Рекомендации по работе с литературой и источниками

7.1. Рекомендации студентам по организации работы с литературой и источниками

Изучение литературы и источников необходимо начинать с прочтения соответствующих глав учебных изданий, учебных пособий или литературы, рекомендованной в качестве основной или дополнительной по дисциплине «Управление данными в сетях», которые прямо или косвенно относятся к изучаемой теме.

Изучая литературу и источники, студенту рекомендуется вести краткий конспект. Однако не следует переписывать все содержание изучаемой темы, нужно выписывать лишь основные идеи и главные на ваш взгляд мысли. В отдельных случаях, когда встречаются важные определения, понятия, необходимый фактический материал и примеры, статистическая информация, имеющие отношение к изучаемой теме, студенту следует выписать их в виде цитат с полным указанием библиографических источников.

Конспектирование рекомендуемой литературы и источников необходимо вести с распределением собранных материалов по отдельным главам и параграфам согласно учебно-тематическому плану. Необходимо выписывать все выходные данные по используемой литературе и источникам.

Важным этапом при работе с рекомендуемой литературой и источниками является изучение законодательных и нормативных актов федерального, регионального, местного и ведомственного уровней. При изучении Указов Президента РФ, Законов и Кодексов РФ, постановлений, положений, рекомендаций и т.д., студент должен выяснить все изменения и дополнения, которые могли быть внесены после их выхода в свет.

Основой технологии интенсификации обучения на платформе цифровых образовательных технологий являются учебно-иллюстрационные материалы (опорный конспект) по дисциплине «Управление данными в

сетях».

Работа с учебно-иллюстрационными материалами имеет следующие этапы.

1. Изучение теоретических основ учебного материала в аудитории: изложение преподавателем изучаемого материала студентам с объяснением по опорному конспекту;

2. Самостоятельная работа: индивидуальная работа студентов по опорному конспекту; фронтальное закрепление по блокам опорного конспекта.

3. Первое повторение - воспроизведение содержания заданной темы опорного конспекта по памяти.

4. Устное проговаривание материала опорного конспекта - необходимый этап внешнеречевой деятельности при усвоении учебного материала.

5. Второе повторение - взаимопрос и взаимопомощь студентов друг другу.

Применение учебно-иллюстрационных материалов позволяет обобщить сложный по содержанию материал, активизировать мыслительную деятельность студентов.

Необходимо помнить, что главное для студента в самостоятельной работе с рекомендуемой литературой и источниками - это формирование своего индивидуального стиля, который может стать основой в будущей профессиональной деятельности.

7.2. Перечень рекомендуемой литературы

Основная литература

1. Агальцов В.П. Базы данных. В 2-х т. Т. 2. Распределенные и удаленные базы данных: Учебник / В.П. Агальцов. – М.: Высшее образование. – 2013. – 272с. ISBN 978-5-8199-0394-0;

2. Карвин Билл Программирование баз данных SQL. Типичные ошибки и их устранение. – М.: РИД ГРУПП. – 2012. – 336с. ISBN: 978-5-4252-0510-0;

Дополнительная литература:

1. Аносова Н. П. и др. Распределенные базы и хранилища данных. – М.: Интуит. – 2009. – 320с.;

2. Гудов А. М. , Завозкин С. Ю. , Рейн Т. С. Базы данных и системы управления базами данных. Программирование на языке PL/SQL: Учебное пособие. – Кемерово: КГУ. – 2010. – 134с. ISBN: 978-5-8353-1005-0;

3. Сибилев В.Д. Базы данных. Учебно-методическое пособие / ТУСУР, каф. АСУ. – Томск: ТУСУР. – 2006. – 27с.;

7.3. Другие информационные источники

- 1 <http://www.exponenta.ru> – образовательный математический сайт;
- 2 <http://www.intuit.ru> – Национальный открытый университет «ИНТУИТ»;
- 3 <http://www.window.edu.ru> – Единое окно доступа к образовательным ресурсам;
- 4 <http://biblioclub.ru/>;
- 5 <http://e.lanbook.com.>;
- 6 <http://www.netacad.com>.

Материально-техническое обеспечение дисциплины

- индивидуальное взаимодействие со студентами по электронной почте для предварительного ознакомления с их разработками при подготовке к аудиторным занятиям;
- использование на лекциях и лабораторных занятиях мультимедийного оборудования для демонстрации листингов программного кода, скриншотов выполненных запросов, работы программ и пр.;
- включение в лабораторные работы индивидуального поиска, систематизации и анализа информации через Интернет;
- авторские презентации к лекциям;
- лекционная аудитория должна быть оборудована мультимедиа проектором;
- СУБД MySQL;
- клиентское приложение HeidiSQL
- компьютерный класс с предустановленным программным обеспечением.