МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное образовательное учреждение
высшего образования
«СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Пятигорский институт (филиал) СКФУ

МЕТОДИЧЕСКИЕ УКАЗАНИЯ
к лабораторным работам по дисциплине «Управление данными» для студентов направления 09.03.02 «Информационные системы и технологии»

Пятигорск, 2025

Методические указания к лабораторным работам по дисциплине «Управление данными» для студентов направления 09.03.02 «Информационные системы и технологии» подготовлены в соответствии рабочей программой с учетом квалификационных требований ФГОС ВО.

СОДЕРЖАНИЕ

Лабораторная работа №1. Составление технического задания на разработку сайта

Цель работы: научиться разрабатывать техническое задание на разработку сайта.

Теоретическое обоснование

Теоретическое обоснование

На техническое задание существует стандарт ГОСТ 19.201-78 «Техническое задание. Требования к содержанию и оформлению». В соответствии с этим стандартом техническое задание должно содержать следующие разделы:

- введение;
- основания для разработки;
- назначение разработки;
- требования к программе или программному изделию;
- требования к программной документации;
- технико-экономические показатели;
- стадии и этапы разработки;
- порядок контроля и приемки.

При необходимости допускается в техническое задание включать приложения.

В зависимости от особенностей программы или программного изделия допускается уточнять содержание разделов, вводить новые разделы или объединять отдельные из них.

2. СОДЕРЖАНИЕ РАЗДЕЛОВ

2.1. В разделе «Введение» указывают наименование, краткую характеристику области применения программы или программного изделия и объекта, в котором используют программу или программное изделие.

2.2. В разделе «Основания для разработки» должны быть указаны: документ (документы), на основании которых ведется разработка; организация, утвердившая этот документ, и дата его утверждения; наименование и (или) условное обозначение темы разработки.

2.3. В разделе «Назначение разработки» должно быть указано функциональное и эксплуатационное назначение программы или программного изделия.

2.4. Раздел «Требования к программе или программному изделию» должен содержать следующие подразделы:

требования к функциональным характеристикам; требования к надежности; условия эксплуатации; требования к составу и параметрам технических средств; требования к информационной и программной совместимости; требования к маркировке и упаковке; требования к транспортированию и хранению; специальные требования.

2.4.1. В подразделе «Требования к функциональным характеристикам» должны быть указаны требования к составу выполняемых функций, организации входных и выходных данных, временным характеристикам и т. п.
2.4.2. В подразделе «Требования к надежности» должны быть указаны требования к обеспечению надежного функционирования (обеспечения устойчивого функционирования, контроль входной и выходной информации, время восстановления после отказа и т.п.).
2.4.3. В подразделе «Условия эксплуатации» должны быть указаны условия эксплуатации (температура окружающего воздуха, относительная влажность и т.п. для выбранных типов носителей данных), при которых должны обеспечиваться заданные характеристики, а также вид обслуживания, необходимое количество и квалификация персонала.
2.4.4. В подразделе «Требования к составу и параметрам технических средств» указывают необходимый состав технических средств с указанием их основных технических характеристик.
2.4.5. В подразделе «Требования к информационной и программной совместимости» должны быть указаны требования к информационным структурам на входе и выходе и методам решения, исходным кодам, языкам программирования и программным средствам, используемым программой. При необходимости должна обеспечиваться защита информации и программ.
2.4.6. В подразделе «Требования к маркировке и упаковке» в общем случае указывают требования к маркировке программного изделия, варианты и способы упаковки.

2.4.7. В подразделе «Требования к транспортированию и хранению» должны быть указаны для программного изделия условия транспортирования, места хранения, условия хранения, условия складирования, сроки хранения в различных условиях.

2.5а. В разделе «Требования к программной документации» должен быть указан предварительный состав программной документации и, при необходимости, специальные требования к ней.

2.5. В разделе «Технико-экономические показатели» должны быть указаны: ориентировочная экономическая эффективность, предполагаемая годовая потребность, экономические преимущества разработки по сравнению с лучшими отечественными и зарубежными образцами или аналогами.
2.6. В разделе «Стадии и этапы разработки» устанавливают необходимые стадии разработки, этапы и содержание работ (перечень программных документов, которые должны быть разработаны, согласованы и утверждены), а также, как правило, сроки разработки и определяют исполнителей.
2.7. В разделе «Порядок контроля и приемки» должны быть указаны виды испытаний и общие требования к приемке работы.
2.8. В приложениях к техническому заданию, при необходимости, приводят: перечень научно-исследовательских и других работ, обосновывающих разработку; схемы алгоритмов, таблицы, описания, обоснования, расчеты и

другие документы, которые могут быть использованы при разработке; другие источники разработки.

Аппаратура и материалы. Для выполнения лабораторной работы необходим персональный компьютер со следующими характеристиками: процессор Intel с тактовой частотой 2000 МГц и выше; оперативная память – не менее 128 Мбайт; свободное дисковое пространство – не менее 800 Мбайт; устройство для чтения компакт-дисков; монитор типа SuperVGA (число цветов – 256) с диагональю не менее 17 . **Программное обеспечение** – операционная система WINDOWS 98 / NT / ME / 2000 / XP/ Vista/ 7, MicrosoftWord.

Указания по технике безопасности. Правила техники безопасности при выполнении лабораторной работы совпадают с общепринятыми для пользователей персональных компьютеров. Студентам запрещается самостоятельно производить ремонт персонального компьютера, установку и удаление программного обеспечения. В случае неисправности персонального компьютера необходимо сообщить об этом обслуживающему персоналу лаборатории (оператору, администратору). Необходимо соблюдать правила техники безопасности при работе с электрооборудованием, не касаться электрических розеток металлическими предметами; рабочее место пользователя персонального компьютера должно содержаться в чистоте; не разрешается возле персонального компьютера принимать пищу, напитки.

Методика и порядок выполнения работы

Составить ТЗ на разработку сайта в соответствии с ГОСТ 19.104-78.

Для оформления отчета можно использовать любой текстовой редактор. Тему согласовать с преподавателем.

Задание. Составить техническое задание на разработку сайта для информационной системы по варианту, согласованному с преподавателем.

Оформить титульный лист по образцу (на странице 9)

Содержание отчета и его форма

Отчет по лабораторной работе должен состоять из: 1) титульного листа (образец приведен в лекции 9);

2) технического задания.

Вопросы для защиты работы

1. Назовите, какой раздел технического задания можно считать основным и почему.
2. Какую информацию должны содержать остальные разделы?

3.В чем основная сложность разработки технического задания?

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное образовательное учреждение высшего образования
«СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

УТВЕРЖДАЮ
Зав кафедрой СУиИТ,
_____.
«___» _____2025г.

Техническое задание на разработку базы данных

Руководитель _____
Исполнитель _____
ФИО, группа

2025

Лабораторная работа №2. Архитектура распределенных Internet приложений. Установка пакета Denwer.

Цель работы: научиться устанавливать пакет для web-разработчика Denwer.

Теоретическое обоснование

Установка базового пакета виртуального сервера Denver. Базовый пакет содержит большинство необходимых программ и утилит:

Инсталлятор (поддерживается также инсталляция на flash-накопитель). Apache, SSL, SSI, mod_rewrite, mod_php.

PHP5 с поддержкой GD, MySQL, sqLite.

MySQL5 с поддержкой транзакций.

Система управления виртуальными хостами, основанная на шаблонах. Чтобы создать новый хост, нужно лишь добавить директорию в каталог

/home, править конфигурационные файлы не требуется. По умолчанию уже поддерживаются схемы именования директорий многих популярных хостеров; новые можно без труда добавить.

Система управления запуском и завершением всех компонентов Денвера.

phpMyAdmin — система управления MySQL через Web-интерфейс.

Эмулятор sendmail и SMTP-сервера (отладочная «заглушка» на localhost:25, складывающая приходящие письма в /tmp в формате.eml); поддерживается работа совместно с PHP, Perl, Parser и т.д.

Подготовка к работе с сетью

Многие ассоциируют слово «сеть» с Интернетом, локальной сетью или хотя бы модемом. И совершенно напрасно. Фраза «настроим сеть» может иметь смысл даже в том случае, когда ни одного из перечисленных устройств у компьютера нет! Здесь имеется ввиду лишь установка драйверов и сетевых протоколов, которые позволят Apache запуститься и работать на локальной машине.

Итак, самый простой тест: откройте Пуск — Выполнить и введите там команду (рисунок 2.1):

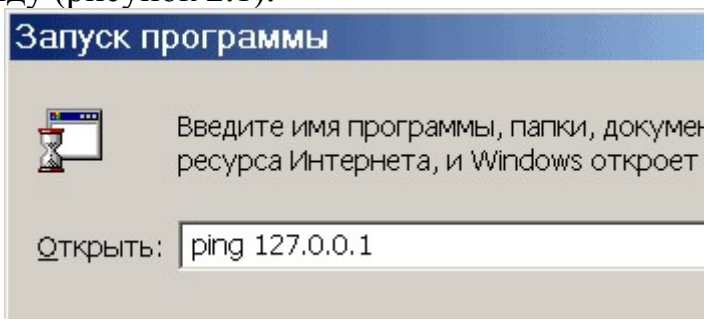


Рисунок 2.1 – Пинг локального сервера

После нажатия Enter вы должны увидеть примерно следующую картину (рисунок.2.2):

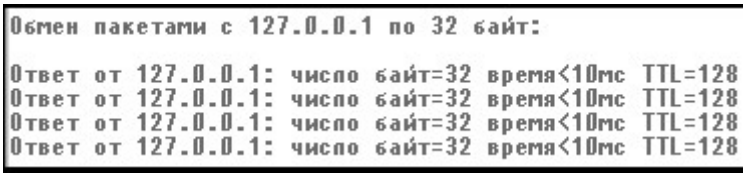


Рисунок 2.2 – Результаты команды «пинг» локального сервера

Процесс продолжается несколько секунд. Если вы это видите, то все в порядке, и вы можете приступать к инсталляции дистрибутива. Если же, например, окно лишь «мигнет» (откроется и тут же закроется), либо же будут выведены какие-нибудь сообщения об ошибках, значит, сетевые протоколы не установлены.

Установка дистрибутива

Запустите инсталлятор Денвера. Вы увидите перед собой следующее (рисунок 2.3):

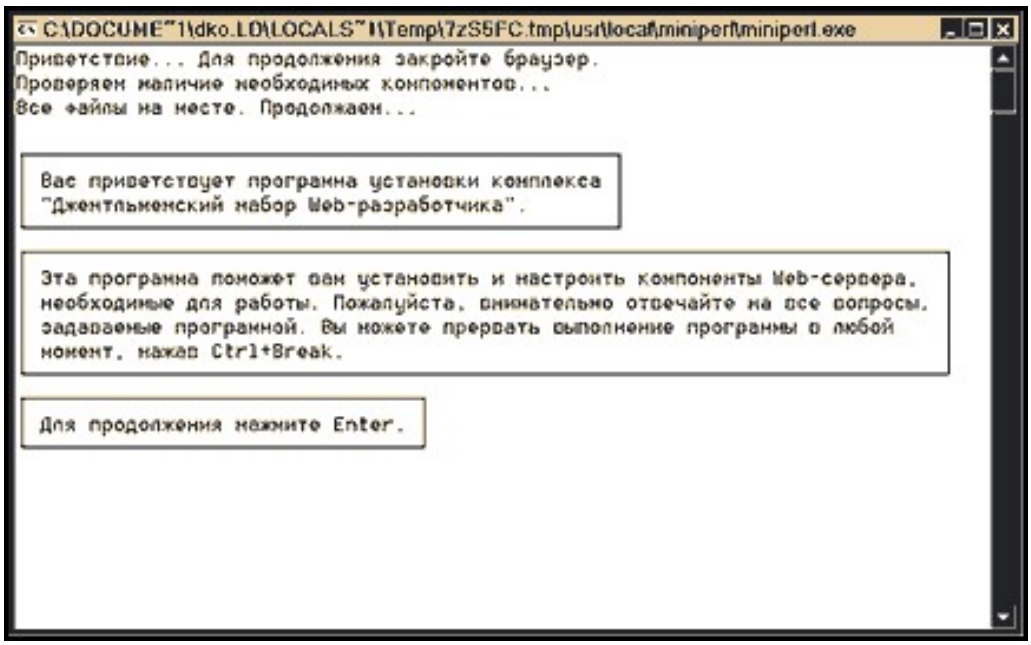


Рисунок 2.3 – Установка Денвера

Вас спросят о том, в какой каталог вы хотели бы установить комплекс (по умолчанию используется C:\WebServers, вам нужно лишь нажать Enter, чтобы согласиться с этим выбором). В указанном каталоге будут расположены абсолютно все компоненты системы, и вне его никакие файлы в дальнейшем не создаются (исключая ярлыки на Рабочем столе).

Настоятельно рекомендуется устанавливать комплекс в каталог первого уровня — то есть, C:\WebServers, а не, например, C:\My\WebServers.

Дело в том, что инсталляторы пакетов расширений ищут базовый комплект именно на первом уровне по всем дискам. И, если не находят, заставляют ввести имя директории вручную.

Далее вам предложат ввести имя виртуального диска, который будет связан с только что указанной директорией. Рекомендуется согласиться со значением по умолчанию (Z:). Важно, что диска с этим именем еще не должно содержаться в системе — чаще всего так и происходит с диском Z:.

После этого начнется копирование файлов дистрибутива, а под конец вам будет задан вопрос, как именно вы собираетесь запускать и останавливать комплекс. У вас есть две альтернативы:

Создавать виртуальный диск при загрузке машины (естественно, инсталлятор позаботится, чтобы это происходило автоматически), а при остановке серверов его (диск) не отключать.

Создавать виртуальный диск только по явной команде старта комплекса (при щелчке по ярлыку запуска на Рабочем столе). И, соответственно, отключать диск от системы — при остановке серверов. Вы должны воспользоваться именно этим способом.

Первый запуск Денвера

После установки щелкайте по созданному инсталлятором ярлыку Start Denwer на Рабочем столе, а затем, дождавшись, когда все консольные окна

исчезнут, открывайте браузер и набирайте в нем адрес: http://localhost/denwer/. Выходить из Интернета при этом не обязательно

(рисунок 2.4).

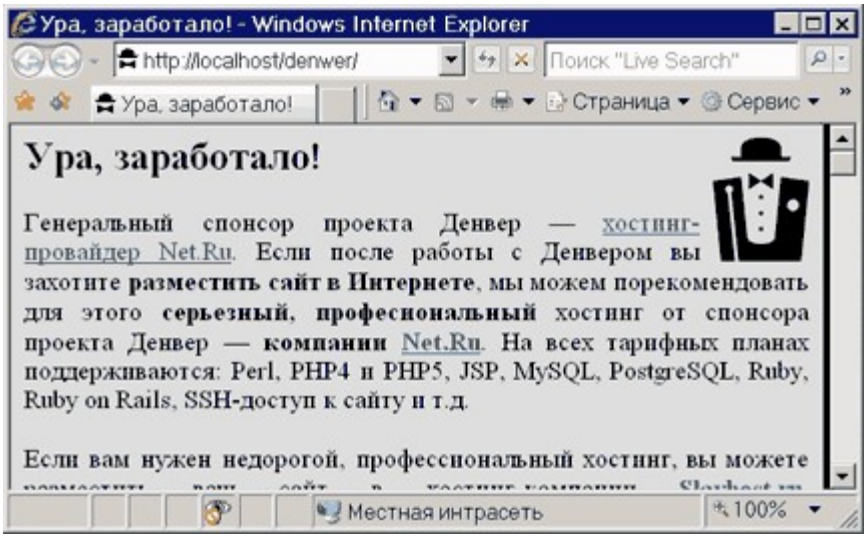


Рисунок 2.4 – Стартовая страница Денвера

Если тестовая страница все же не загрузится, проверьте:

Отключен ли у вас прокси-сервер в настройках браузера?

Запущен ли Денвер? Если да, нет ли ошибок при щелчке на пиктограмме пера (справа внизу)?

Не запущен ли у вас какой-то другой Web-сервер, который мешает Денверу (часто бывает в Windows XP)? Например, Microsoft IIS? Если да, отключите его.

Денвер прошел тестирование в следующих ОС:

Windows 95/98/ME; Windows NT/2000/XP/2003; Windows Vista.

Windows 7.

Работа с виртуальными хостами

Вниманию пользователей Windows NT, 2000 или XP (и старше). Прежде, чем продолжить, убедитесь, что у вас запущена служба «DNSклиент». Это

можно сделать, открыв Панель управления — Администрирование — Службы. В противном случае, виртуальные хосты работать не будут.

Добавить новый виртуальный хост в Денвере чрезвычайно просто.

Пусть это будет test1.ru. Вам нужно проделать следующее (рисунок 2.5):

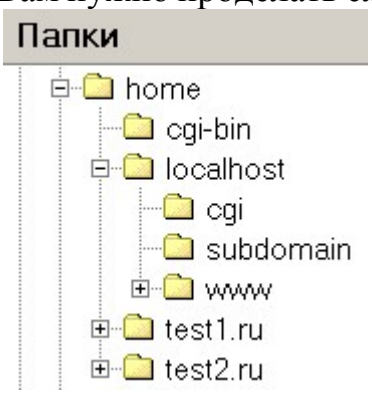


Рисунок 2.5 – Структура папок Девера

- Создать в папке /home директорию с именем, совпадающим с именем виртуального хоста (в нашем случае test1.ru). Имя директории содержит точку. Эта директория будет хранить директории документов доменов третьего уровня для test1.ru. Например, имя abc.test1.ru связывается сервером с директорией /home/test1.ru/abc/, а имя abc.def.test1.ru — с /home/test1.ru/abc.def/. Ну и, конечно, поддиректория www соответствует адресам www.test1.ru и просто test1.ru. На рисунке показано, как может выглядеть директория /home. Не забудьте создать папку www в директории виртуального хоста, ведь именно в ней будут храниться его страницы и скрипты.
- Перезапустить сервер, воспользовавшись, например, ярлыком Restart Denwer на Рабочем столе.

Как только вы начнете создавать виртуальные хосты, Контроллер удаленного доступа на некоторых системах может предлагать вам альтернативу (рисунок 2.6 или 2.7):

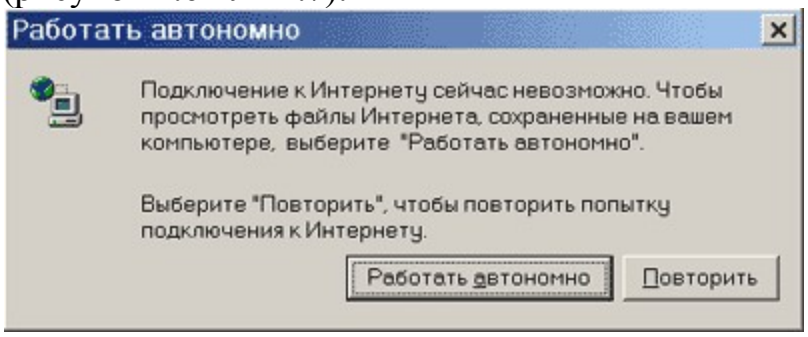


Рисунок 2.6 – Пинг локального сервера

или так:

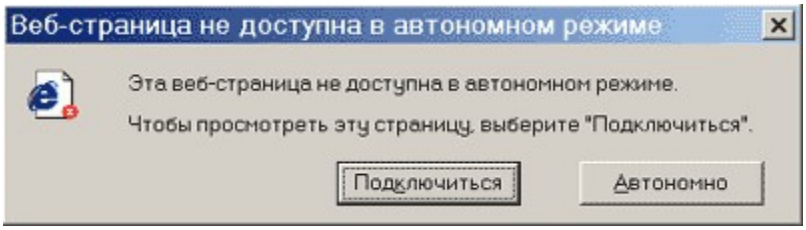


Рисунок 2.7 – Пинг локального сервера

Всегда выбирайтеПодключиться или Повторить.

Ни в коем случае не давайте ему ответ Автономно.

Но если ваш Контроллер удаленного доступа не реагирует и на ответПодключиться начинает набирать номер на модеме, откройте Сервис — Свойства обозревателя — Подключение и в разделе Настройка удаленного доступа поставьте флажок Не использовать (или Never Dial a connection).

Это рекомендации для пользователей Windows 2000. На всех остальных системах пункты меню и кнопки могут называться немного подругому, но смысл остается тот же.

Многие версии Windows поставляются со включенным по умолчанию прокси-сервером. Это может вызвать кое-какие проблемы при работе с Денвером (впрочем, легко разрешимые).

- Если после запуска Денвера страница http://localhost не работает, вероятнее всего, вам нужно отключить прокси-сервер в настройках браузера. Для "простых" хостов (вроде localhost, test, dklab и т.д.) обычно достаточно флажка «Не использовать прокси-сервер для локальных адресов» на вкладке Свойства обозревателя — Подключение — Настройка сети — Дополнительно.
- Если localhost работает, а test1.ru (и вообще хосты, имя которых состоит из нескольких частей) — нет, то, вероятно, ваш браузер не может распознать последний хост как локальный. Вам необходимо либо полностью отключить прокси-сервер, либо же перечислить хосты в списке Подключение — Настройка сети — Дополнительно — Исключения.

Методика и порядок выполнения работы

Изучить процесс установки, описанный в разделе «Теоретическое обоснование». Установить пакет Денвер.

Содержание отчета и его форма

Отчет по лабораторной работе должен состоять из:

1) названия лабораторной работы;
2) описание установки пакета Денвер.

Вопросы для защиты работы:

1. Какие утилиты входят в базовый пакет Денвер?

Лабораторная работа №3.Основная информация о MySQL. Создание базы данных

Цель работы: научиться создавать и администрировать базу данных MySQL.

Теоретическое обоснование

MySQL - это популярная система управления базами данных (СУБД), очень часто применяемая в сочетании с PHP.

База данных представляет собой структурированную совокупность данных. Эти данные могут быть любыми - от простого списка предстоящих покупок до перечня экспонатов картинной галереи или огромного количества информации в корпоративной сети. Для записи, выборки и обработки данных, хранящихся в компьютерной базе данных, необходима система управления базой данных, каковой и является ПО MySQL. Поскольку компьютеры замечательно справляются с обработкой больших объемов данных, управление базами данных играет центральную роль в вычислениях. Реализовано такое управление может быть по-разному - как в виде отдельных утилит, так и в виде кода, входящего в состав других приложений.

MySQL - это система управления реляционными базами данных. В реляционной базе данных данные хранятся не все скопом, а в отдельных таблицах, благодаря чему достигается выигрыш в скорости и гибкости. Таблицы связываются между собой при помощи отношений, благодаря чему обеспечивается возможность объединять при выполнении запроса данные из нескольких таблиц. SQL как часть системы MySQL можно охарактеризовать как язык структурированных запросов плюс наиболее распространенный стандартный язык, используемый для доступа к базам данных.

MySQL - это ПО с открытым кодом. Применять его и модифицировать может любой желающий. Такое ПО можно получать по Internet и использовать бесплатно. При этом каждый пользователь может изучить исходный код и изменить его в соответствии со своими потребностями.

Использование программного обеспечения MySQL регламентируется лицензией GPL (GNU General Public License), http://www.gnu.org/licenses/, в которой указано, что можно и чего нельзя делать с этим программным обеспечением в различных ситуациях.

Почему веб-программисты отдают предпочтение СУБД MySQL? MySQL является очень быстрым, надежным и легким в использовании. Если вам требуются именно эти качества, попробуйте поработать с данным сервером. MySQL обладает также рядом удобных возможностей,

разработанных в тесном контакте с пользователями. Первоначально сервер MySQL разрабатывался для управления большими базами данных с целью обеспечить более высокую скорость работы по сравнению с существующими на тот момент аналогами. И вот уже в течение нескольких лет данный сервер успешно используется в условиях промышленной

эксплуатации с высокими требованиями. Несмотря на то что MySQL постоянно совершенствуется, он уже сегодня обеспечивает широкий спектр полезных функций. Благодаря своей доступности, скорости и безопасности MySQL очень хорошо подходит для доступа к базам данных по Internet.

Технические возможности СУБД MySQL MySQL является системой клиент-сервер, которая содержит многопоточный SQL-сервер,

обеспечивающий поддержку различных вычислительных машин баз данных, а также несколько различных клиентских программ и библиотек, средства администрирования и широкий спектр программных интерфейсов (API). Мы также поставляем сервер MySQL в виде многопоточной библиотеки, которую можно подключить к пользовательскому приложению и получить компактный, более быстрый и легкий в управлении продукт. Доступно также большое количество программного обеспечения для MySQL, в большей части - бесплатного.

MySQL состоит из двух частей: серверной и клиентской. Сервер MySQL постоянно работает на компьютере. Клиентские программы (например, скрипты PHP) посылают серверу MySQL SQL-запросы через механизм сокетов (то есть при помощи сетевых средств), сервер их обрабатывает и запоминает результат. То есть скрипт (клиент) указывает, какую информацио он хочет получить от сервера баз данных. Затем сервер баз данных посылает ответ (результат) клиенту (скрипту).

Почему всегда передается не весь результат? Очень просто: дело в том, что размер результирующего набора данных может быть слишком большим, и на его передачу по сети уйдет чересчур много времени. Да и редко когда бывает нужно получать сразу весь вывод запроса (то есть все записи, удовлетворяющие выражению запроса). Например, нам может потребоваться лишь подсчитать, сколько записей удовлетворяет тому или иному условию, или же выбрать из данных только первые 10 записей. Механизм использования сокетов подразумевает технологию клиент-сервер, а это означает, что в системе должна быть запущена специальная программа — MySQL-сервер, которая принимает и обрабатывает запросы от программ. Так как вся работа происходит в действительности на одной машине, накладные расходы по работе с сетевыми средствами незначительны (установка и поддержание соединения с MySQL-сервером обходится довольно дешево).

Структура MySQL трехуровневая: базы данных — таблицы — записи. Базы данных и таблицы MySQL физически представляются файлами с расширениями frm, MYD, MYI. Логически - таблица представляет собой совокупность записей. А записи - это совокупность полей разного типа. Имя базы данных MySQL уникально в пределах системы, а таблицы - в пределах базы данных, поля - в пределах таблицы. Один сервер MySQL может поддерживать сразу несколько баз данных, доступ к которым может разграничиваться логином и паролем. Зная эти логин и пароль, можно работать с конкретной базой данных. Например, можно создать или удалить в ней таблицу, добавить записи и т. д. Обычно имя-идентификатор и пароль

назначаются хостинг провайдерами, которые и обеспечивают поддержку MySQL для своих пользователей.

Методика и порядок выполнения работы

1. Запустить Denver
2. В браузере перейти по ссылке http://localhost/Tools/phpMyAdmin/

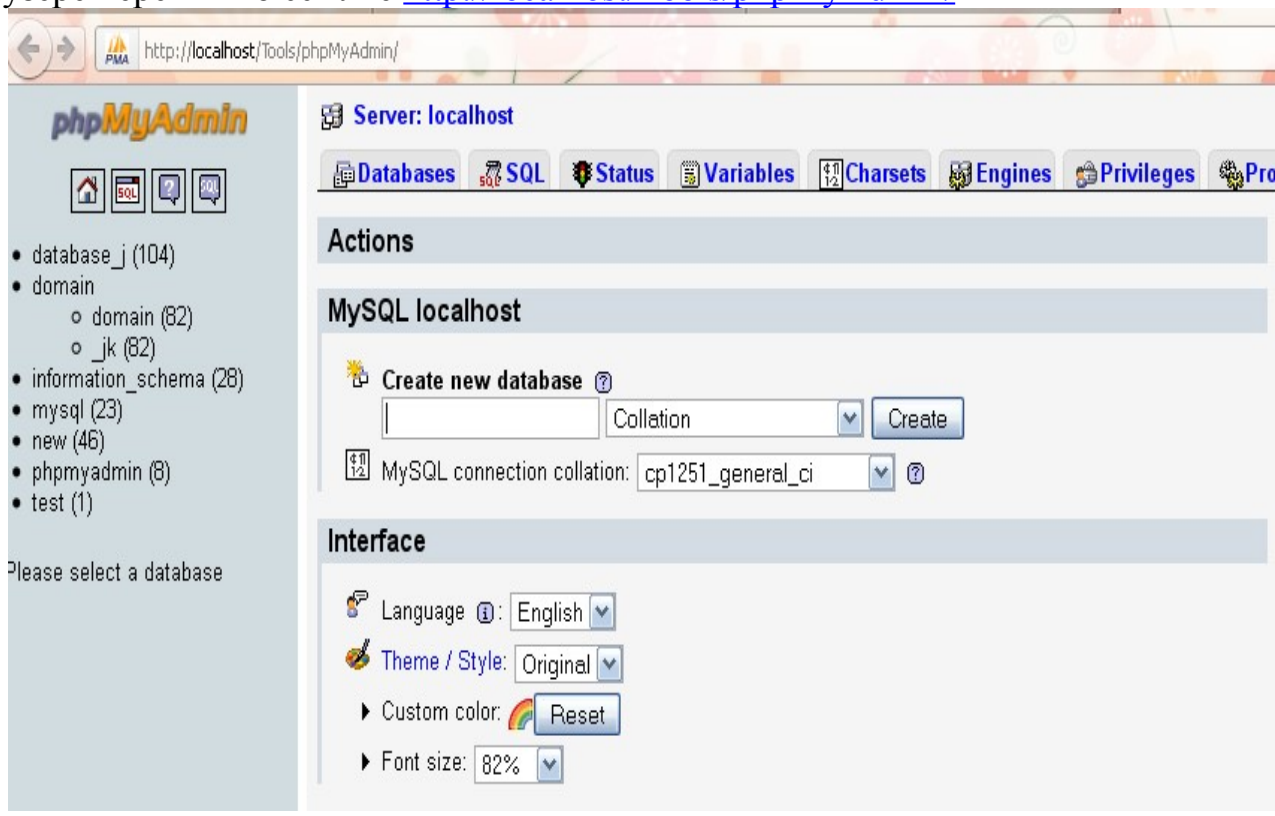


Рисунок 3.1- Окно браузера

В поле Createnewdatabase (рисунок 3.1) ввести имя создаваемой базы данных, в списке MySqlconnection collation выбираем utf8_general_ci, нажимает кнопку create (рисунок 3.2).

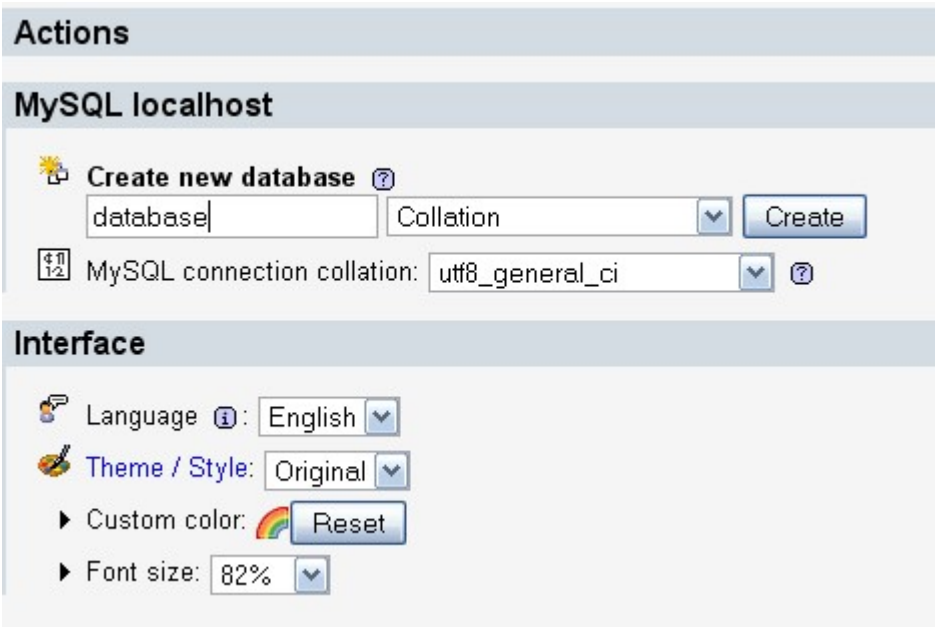


Рисунок 3.2 – Создание БД

Задание

1. Создать нового пользователя базы данных, обладающего всеми привилегиями и установить для него пароль. Воспользуйтесь вкладкой privileges.
2. Создайте пользователя, обладающего ограниченными правами, и задайте ему пароль.

Содержание отчета и его форма

Отчет по лабораторной работе должен состоять из: 1) названия лабораторной работы; 2) описание создания БД.

Вопросы для защиты работы:

1. Что такое привилегии?
2. Какие привилегии Вы выбрали при создании БД?

Лабораторная работа №4. Поля и их типы в MySQL

Цель работы: научиться использовать различные типы данных базы данных MySQL.

Теоретическое обоснование

База данных с точки зрения MySQL (и некоторых других СУБД) - это обыкновенный каталог, содержащий двоичные файлы определенного формата - таблицы. Таблицы состоят из записей, а записи, в свою очередь, состоят из полей. Поле имеет два атрибута - имя и тип.

Тип поля может быть:

- Целым;
- Вещественным;
- Строковым;

- Бинарным;
- Дата и время;
- Перечисления и множества.

Возможные типы данных, диапазоны и описания представлены в таблицах 4.1- 4.5.

Таблица 4.1- Целочисленные типы данных

Тип	Диапазон
TINYINT	-128...+127
SMALLINT	-32768...+32767
MEDIUMINT	-8 388 608...+8 388 607
INT	-2 147 483 648...+2 147 483 647
BIGINT	-9 223 372 036 854 775 808...+9 223 372 036 854 775 807

Вещественные типы записываются в виде:

ТИП (ДЛИНА, ЗНАКИ) [UNSIGNED]

Длина - это колличество знакомест, в которых будет размещено все число при его передаче, а ЗНАКИ - это колличество знаков после десятичной точки, которые будут учитываться. Если указан модификатор UNSIGNED, знак числа учитываться не будет.

Таблица 4.2- Вещественные числа

Тип	Описание
FLOAT	Небольшая точность
DOUBLE	Двойная точность
REAL	То же, что и DOUBLE
DECIMAL	Дробное число, хранящееся в виде строки
NUMERIC	То же, что и DECIMAL

Любая строка - это массив символов. При поиске с помощью оператора SELECT (мы рассмотрим его далее) не учитывается регистр символов: строки "HELLO" и "Hello" считаются одинаковыми.

Можно настроить MySQL на автоматическое перекодирование символов - в этом случае в базе данных строки будут храниться в одной кодировке, а выводиться - в другой.

В большинстве случаев применяется тип VARCHAR или просто CHAR, позволяющий хранить строки, содержащие до 255 символов. В скобках после типа указывается длина строки:

VARCHAR(48); CHAR(73);

Если 255 символов для вашей задачи недостаточно, можно использовать другие типы, например, TEXT.

Таблица 4.3 - Строки

Тип	Описание

TINYTEXT	Максимальная длина 255 символов
TEXT	Максимальная длина 65535 символов (64 Кб)
MEDIUMTEXT	Максимальная длина 16 777 215 символов
LONGTEXT	Максимальная длина 4 294 967 295 символов

Бинарные типы данных также можно использовать для хранения текста, но при поиске будет учитываться регистр символов. К тому же, любой текстовый тип можно преобразовать в бинарный, указав модификатор BINARY:

VARCHAR(30) BINARY;

Таблица 4.4- Бинарные типы данных

Тип	Описание
TINYBLOB	Максимум 255 символов
BLOB	Максимум 65535 символов
MEDIUMBLOB	Максимум 16 777 215 символов
LONGBLOB	Максимум 4 294 967 295

Примечание: Бинарные данные не перекодируются "на лету", если установлена перекодировка символов. Таблица 4.5- Дата и время

Тип	Описание
DATE	Дата в формате ГГГ-ММ-ДД
TIME	Время в формате ЧЧ:ММ:СС
TIMESTAMP	Дата и время в формате timestamp, выводится в виде ГГГГММДДЧЧММСС
DATETIME	Дата и время в формате ГГГГ-ММ-ДД ЧЧ:ММ:СС

Другие типы данных MySQL рассматривать бессмысленно, поскольку применение их в PHP нецелесообразно.

Задание

Изучите поля и типы данных MySQL.

Содержание отчета и его форма

Отчет по лабораторной работе должен состоять из:

1) названия лабораторной работы;
2) описание типов данных MySQL.

Вопросы для защиты работы:

1. Какие типы данных поддерживает MySQL?
2. Каким образом можно представить дату?

Лабораторная работа №5. Операторы и команды MySQL

Цель работы: научиться использовать операторы и команды базы данных MySQL.

Теоретическое обоснование

Структурированный язык запросов SQL позволяет производить различные операции с базами данных: создавать таблицы, помещать, обновлять и удалять из них данные, производить запросы из таблиц и.т.д.

Далее мы последовательно рассмотрим все эти операторы.

Несмотря на то, что последний стандарт SQL принят в 1992 году, на сегодняшний день нет ни одной СУБД, где бы он полностью выполнялся. Более того, в различных базах данных часть операций осуществляется поразному. Мы будем придерживаться диалекта SQL характерного для СУБД MySQL поэтому не все запросы могут выполняться для других баз данных.

Примечание: Команды SQL не чувствительны к регистру, но традиционно они набираются прописными буквами.

Создание таблиц. Оператор CREATE

Создать таблицу через SQL-запрос позволяет оператор CREATE. Его синтаксис:

```
CREATE TABLE Имя_таблицы
(
Имя_поля1 Тип Модификатор
...
Имя_поляN Тип Модификатор
[первичный ключ]
[внешний ключ]
)
```

Вообще, с помощью оператора CREATE можно создавать и другие объекты, но мы их рассматривать не будем, поскольку их применение весьма ограничено.

В качестве модификаторов можно использовать следующие значения:

- NOT NULL - поле не может содержать неопределенного значения (NULL), то есть поле должно быть явно инициализировано;
- PRIMARY KEY - поле будет первичным ключом (идентификатором записи), по которому можно однозначно идентифицировать запись;
- AUTO_INCREMENT - при вставке новой записи значение этого поля будет автоматически увеличено на единицу, поэтому в таблице не будет двух записей с одинаковым значением этого поля;
- DEFAULT - задает значение, которое будет использовано по

умолчанию, если при вставке записи поле не будет инициализировано явно.

Значение по умолчанию задается следующим образом (таблица 5.1):

Таблица 5.1- Задание значений по умолчанию

Имя_поля	Тип	DEFAULT	Значение, например
NO	INT	DEFAULT	0
NAME	INT	DEFAULT	'Петров'

Теперь создадим таблицы - "Товар", "Клиенты", "Заказы":

```
CREATE TABLE CLIENTS
(
C_NO    int      NOT NULL,
FIO     char(40)  NOT NULL,
ADDR    char(30)  NOT NULL,
CITY    char(15)  NOT NULL,
PHONE   char(11)  NOT NULL
);
```

Таблица CLIENTS содержит поля C_NO (номер клиента), FIO (Фамилия, Имя, Отчество), ADDR (Адрес), CITY (Город) и PHONE

(Телефон). Все эти поля не могут содержать пустого значения (NOT NULL).

```
CREATE TABLE TOOLS
(
T_NO   int         NOT NULL,
DSEC   char(40)    NOT NULL,
PRICE  double(9,2)  NOT NULL,
QTY    double(9,2)  NOTNULL
);
```

Данная таблица будет содержать данные о товарах. Тип double(9,2) означает, что 9 знаков относим под целую чать, и два - под дробную. QTY - этоколличествотоваранаскладе.

```
CREATE TABLE ORDERS
(
O_NO    int        NOT NULL,
DATE    date       NOT NULL,
C_NO    int        NOT NULL,
T_NO    int        NOT NULL,
QUANTITY double(9,2) NOT NULL,
AMOUNT   double(9,2) NOT NULL
);
```

Эта таблица содержит сведения о заказах - номер заказа (O_NO), дату заказа (DATE), номер клиента (C_NO), номер товара (T_NO), колличество

(QUANTITY) и сумму всего заказа AMOUNT (то есть AMOUNT = T_NO * TOOL_PRICE).

Добавление данных в таблицу. Оператор INSERT

Для добавления записей используется оператор INSERT:

INSERT INTO Имя_таблицы [(Список полей)]

VALUES (Список констант);

После выполнения оператора INSERT будет создана новая запись, в качестве значений полей будут использованы соответствующие константы, указанные в списке VALUES.

Теперь добавим данные в наши таблице. Добавить данные можно с помощью оператора INSERT. Рассмотрим пример использование оператора INSERT:

INSERT INTO CLIENTS

VALUES (1, 'Иванов И.И.', 'Вокзальная 3', 'Москва', '09599911100');

Добавляемые значения должны соответствовать тому порядку, в котором поля перечислены в операторе CREATE. Если вы хотите добавлять информацию в другом порядке, то вы должны указать этот порядок в операторе INSERT, например:

INSERT INTO CLIENTS (FIO, ADDR, C_NO, PHONE, CITY)

VALUES ('Петров', 'Мира 29', 2, '-', 'Екатеринбург');

С помощью INSERT мы можем добавлять данные и в определенные поля, например, C_NO и FIO:

INSERT INTO CLIENTS (C_NO, FIO)

VALUES (1, 'Иванов');

Однако, в нашем случае сервер MySQL не выполнит такой запрос, поскольку все остальные поля равны NULL (пустое значение), а наша таблица не принимает пустые значения. Аналогично можно добавить данные в другие таблицы.

В качестве примера, добавим данные в таблицу TOOLS: INSERT INTO TOOLS

VALUES (1, 'Клавиатура ABC', 340.98, 5);

Обратите внимание, что мы пока не указали первичные ключи таблицы, поэтому нам никто не мешает добавить в таблицу одинаковые записи. Добавить дату в поле DATE можно с помощью функции STR_TO_DATE:

INSERT INTO ORDERS VALUES (1, STR_TO_DATE('01/03/10','%m/%d/%y'),1,1,1,340.98)

Данная запись означает, что третьего января 2010 года Иванов И.И. (C_NO=1) заказал одну (QUANTITY=1) клавиатуру ABC (T_NO=1).

Обновление записей. Оператор UPDATE

Синтаксис оператора UPDATE, который используется для обновления записей, выглядит так:

UPDATE Имя_таблицы

SET Поле1 = Значение1,..., ПолеN = ЗначениеN

[WHERE Условие];

Если не задано условие WHERE, будет модифицирована вся таблица, а это может повлечь за собой непредсказуемые последствия, поскольку для всех записей будут установлены одинаковые значения полей, поэтому всегда указывайте условие WHERE.

Предположим, нам необходимо обновить запись, если, например, клиент Иванов переехал в другой город и нам нужно отметить это событие в базе данных. Сделаемследующее:

UPDATECLIENTS

SETCITY = 'Псков' WHEREC_NO = 1;

Данный запрос нужно понимать так: найти запись, поле C_NO которой = 1 (это код клиента Иванова), и установить значение CITY равным "Псков".

Удаление записей. Оператор DELETE

Если нам необходимо удалить всех клиентов, номера которых превышают 5, то мы поступим следующим образом:

DELETE FROM CLIENTS

WHERE C_NO > 5;

С помощью оператора DELETE можно удалить все записи таблицы, указав условие, которое подойдет для всех записей, например:

DELETE FROM CLIENTS;

Если вторая часть оператора DELETE-WHERE не указана, значит, действие оператора распространяется на все записи сразу.

Выбор записей. Оператор SELECT

Добавление, изменение и удаление записей - это, конечно, очень важные команды, но вы часто будете использовать оператор SELECT, который выбирает данные из таблицы. Синтаксис этого оператора более сложен:

SELECT [DISTINCT|ALL] {*| [поле1 AS псевдоним] [,..., полеN AS псевдоним]}

FROM Имя_таблицы1 [,..., Имя_таблицыN]

[WHERE условие]

[GROUP BY список полей] [HAVING условие]

[ORDER BY список полей]

Мы полностью не будем рассматривать оператор SELECT, лучше это делать на конкретном примере. Сейчас мы рассмотрим оператор SELECT в общих чертах. Например, для вывода всех записей из таблицы CLIENTS сделайте следующее:

SELECT * FROM CLIENTS;

В результате вы получите ответ сервера, показанный в таблице 5.2.

Таблица 5.2 – ответ сервера

C_NO	FIO	ADDR	CITY	PHONE
1	Иванов И.И.	Вокзальная 3	Москва	09599911100
1	Иванов И.И.	Вокзальная 3	Москва	09599911100
2	Петров П.П.	Мира 29	Екатеринбург	3438920437

Обратите внимание на первые две записи - они одинаковые. Теоретически, добавление одинаковых записей возможно - ведь мы не указали первичный ключ таблицы. Если вы хотите исключить одинаковые записи из ответа сервера (но не из таблицы), используйте запрос:

```
SELECT DISTINCT *
FROM CLIENTS;
```

Предположим, вы хотите вывести только фамилию и номер телефона клиента, тогда используйте следующий запрос:

```
SELECT DISTINCT FIO, PHONE
FROM CLIENTS;
```

Если вам нужно вывести все товары, цена на которые превышает 800, то воспользуйтесь таким запросом:

```
SELECT *
FROM TOOLS
WHERE PRICE > 800;
```

Вы можете использовать следующие операторы отношений: <, >, =, <>, <=, >=.

Если в вашей таблице присутствуют несколько однофамильцев, то для вывода информации обо всех из них, используйте модификатор LIKE, например:

```
SELECT *
FROM CLIENTS
WHERE FIO LIKE '%Иван%';
```

Приведенный запрос можно причитать так: вывести информацию о клиентах, фамилия которых похожа на 'Иван'.

Если вам необходимо выбрать данные из разных таблиц, то перед именем поля нужно указывать имя таблицы. Вот запрос, который позволяет вывести имена всех клиентов, которые хотя бы один раз покупали товар:

```
SELECT DISTINCT CLIENTS.FIO
FROM CLIENTS, ORDERS
WHERE CLIENTS.C_NO = ORDERS.C_NO;
```

Оператор SELECT позволяет использовать вложенные запросы, однако MySQL их не поддерживает.

Внутренние функции MIN, MAX, AVG, SUM

При работе с оператором SELECT вам доступны несколько очень полезных внутренних функций MySQL, вычисляющих количество элементов (COUNT), сумму элементов (SUM), максимальное и минимальное значения (MAX и MIN), а также среднее значение (AVG).

Следующие операторы выведут, соответственно, колличество записей в таблице CLIENTS, самый дорогой товар и сумму цен всех товаров:

```
SELECT COUNT(*)
FROM CLIENTS;
SELECT MAX(PRICE)
FROM TOOLS;
```

SELECTSUM(PRICE)
FROMTOOLS;

Группировка записей

Оператор SELECT позволяет группировать возвращаемые значения. Например, клиент Иванов (C_NO=1) несколько раз заказывал какой-то товар. Значит, его номер встречается в таблице ORDERS несколько раз. Другой клиент также мог сделать несколько заказов. Мы можем сгруппировать все записи по полю C_NO (номер клиента), а затем вывести сумму заказа каждого клиента.

SELECT CLIENTS.FIO, SUM(ORDERS.AMOUNT) AS TOTALSUM
FROM CLIENTS, ORDERS
WHERE CLIENTS.C_NO = ORDERS.C_NO
GROUP BY ORDERS.C_NO;

Группировку выполняет оператор GROUP BY, который является частью оператора SELECT. Оператор GROUP BY можно ограничить с помощью HAVING. Этот оператор используется для отбора строк, возвращаемых GROUP BY. HAVING можно считать аналогом WHERE, но только для GROUP BY:

HAVING <условие>

Например, нас интересуют только клиенты, которые заказали товаров на общую сумму, превышающую 600:

SELECT CLIENTS.FIO, SUM(ORDERS.AMOUNT) AS TOTALSUM
FROM CLIENTS, ORDERS
WHERE CLIENTS.C_NO = ORDERS.C_NO
GROUP BY ORDERS.C_NO
HAVING TOTALSUM > 600;

ВэтомзапросемыиспользовалипсевдонимстолбцаTOTALSUM.

Сортировка записей

Пока мы не установили первичный ключ, сортировка таблицы не выполняется. Данные будут отображены в порядке их занесения в таблицу. Для сортировки по полю C_NO результата вывода таблицы CLIENTS используется следующий оператор (сама таблица при этом не сортируется):

SELECT *
FROM CLIENTS
ORDER BY C_NO;

Ключи

Предположим, что кто-то добавил в таблицу CLIENTS запись:

1 Сидоров Свободы 7 Калининград 0113452103

В то же время, до этого номер 1 был закреплен за Ивановым. У нас получилось, что один и тот же номер сопоставлен разным клиентам. Чтобы избежать такой путаницы, необходимо использовать первичные ключи:

ALTERTABLECLIENTS
ADDPRIMARYKEY (C_NO)

После этого запроса поле C_NO может содержать только уникальные значения. В качестве первичного ключа нельзя использовать поле, допускающее значение NULL. Создать первичный ключ можно и проще - при создании таблицы следующим образом:

```
CREATE TABLE CLIENTS
(
C_NO    int      NOT NULL,
FIO     char(50) NOT NULL,
ADDR    char(55) NOT NULL,
CITY    char(20) NOT NULL,
PHONE   char(8)  NOT NULL,
PRIMARYKEY (C_NO);
);
```

Таблица ORDERS содержит сведения о заказах. По полю C_NO этой таблице идентифицируется заказчик. Предположим, что в таблицу ORDERS кто-то ввел значение, которого нет в таблице CLIENTS. Кто заказал товар? Нам нужно не допустить подобной ситуации, поэтому следует использовать подобный запрос:

```
ALTER TABLE ORDERS
ADD FOREIGN KEY(C_NO) REFERENCES CLIENTS;
```

Введенные в таблицу ORDERS номера клиентов C_NO должны существовать в таблице CLIENTS. Аналогично нужно добавить внешний ключ по полю T_NO. Эта возможность называется декларативной целостностью.

Команда ALTER используется не только для добавления ключей. Она предназначена для реорганизации таблицы в целом. Вы хотите добавить еще одно поле? Или установить список допустимых значений для каждого из полей. Все это можно сделать с помощью команды ALTER:

```
ALTER TABLE CLIENTS ADD ZIP char(7) NULL;
```

Этот оператор добавляет в таблицу CLIENTS новое поле ZIP типа char. Обратите внимание, что вы не можете добавить новое поле со значением NOT NULL в таблицу, в которой уже есть данные. Например, если компания работает только с клиентами Москвы и Екатеринбурга, то целесообразно ввести список допустимых значений для таблицы CLIENTS:

```
ALTER TABLE CLIENTS
ADD CONSTRAINT INVALID_STATE CHECK (CITY IN ('Москва',
'Екатеринбург'));
```

Использование внешних ключей

Теперь углубимся в изучение SQL. Вы уже знаете, как добавлять первичный ключ, теперь добавим внешний ключ при создании таблицы. Внешние ключи используются для связи одной таблицы с другой. Например, в таблице CLIENTS у нас есть два клиента - Иванов (C_NO=1) и Петров (C_NO=2). Оператор в магазине при оформлении заказа ошибся и указал несуществующий номер, например, C_NO=3. Как мы потом сможем

идентифицировать клиента? Для решения такой проблемы и существуют внешние ключи:

```
CREATETABLET
(
/* Описаниеполейтаблицы */
FOREING KEY KEY_NAME (LIST)
REFERENCES ANOTHER_TABLE [(LIST2)]
[ON DELETE OPTION]
[ON UPDATE OPTION]
);
```

Здесь:

- KEY_NAME - Имя ключа. Имя не является обязательным, но рекомендуется всегда указывать имя ключа - если вы не укажите имя ключа, вы потом не сможете его удалить;
- LIST - это список полей, входящих во внешний ключ. Список разделяется запятыми;
- ANOTHER_TABLE - это другая таблица, по которой

устанавливается не внешний ключ, а необязательный элемент;

- LIST2 - это список полей этой таблицы. Типы полей в списке LIST должны совпадать с типами полей в списке LIST2.

Предположим, что в первой таблице у нас есть поля - NO и NAME - целого и символьного типов соответственно. Во второй таблице у нас есть поля с одинаковыми именами и тапами. Определениевнешнегоключа:

```
FOREIGN KEY KEY_NAME (NO, NAME)
REFERENCES ANOTHER_TABLE (NAME, NO)
```

Это определение некорректно, потому что типы полей NO и NAME не совпадают. Нужно использовать такое определение:

```
FOREIGN KEY KEY_NAME (NO, NAME)
REFERENCES ANOTHER_TABLE (NO, NAME)
[ON DELETE <OPTION>]
[ON UPDATE <OPTION>]
```

Если поля имеют одинаковые имена, как в нашем случае, список LIST2 лучше вообще не указывать.

Необязательные параметры ON DELETE <OPTION> и ON UPDATE <OPTION> определяют действие по обновлению информации в базе данных, при удалении информации из таблицы и при ее обновлении. А действия могут быть следующими:

- CASCADE - удаление или обновление значений везде, где оно встречается. Например, у нас есть таблица клиентов и заказов. Иы хотим удалить запись клиента с номером C_NO=1. Из таблицы заказов будут удалены сведения обо всех заказах, сделанных клиентом;
- NOACTION - вы не сможете удалить информацию из таблицы клиентов до тех пор, пока вы не удалите все заказы, сделанные этим клиентом.

То есть действие NOACTION запрещает удалять запись из основной таблицы, если она используется в дочерней таблице;

- SETNULL - все значения в дочерней таблице будут заменены на NULL (если значения NULL допускаются);
- С помощью параметра SET_DEFAULT вы можете указать значение по умолчанию. Например, если вы укажите SET_DEFAULT 1, то при удалении клиента с любым номером его заказы бубудт приписываться клиенту с номером 1, который есть в таблице CLIENTS.

Удаление полей и таблиц. Оператор DROP

Стандартом SQL не предусмотрено удаление столбцов, однако в MySQL мы это можем сделать:

ALTER TABLE CLIENTS DROP ZIP;

А удалить таблицу еще проще: DROP ORDERS;

Отключение от СУБД

Используя запрос DISCONNECT можно отключиться от используемой базы данных, а затем, используя запрос CONNECT, подключиться к другой базе данных. В некоторых серверах SQL запрос DISCONNECT не работает, а вместо CONNECT применяется запрос USE.

При использовании PHP нет необходимости использовать данные запросы, поскольку для отключения от сервера MySQL используется функция mysql_close(), а для подключения к серверу MySQL используется функция mysql_connect().

Методика и порядок выполнения работы

Создайте базу данных, состоящую из трех таблиц (CLIENTS, ORDERS, TOOLS), заполните ее данными, используя язык запросов SQL(рисунки 5.1 и 5.2). Установите для таблиц ключевые поля, запрет на ввод нулевых (неопределенных) значений. Используя запросы, проверьте целостность базы данных.

Ввести запрос на вкладке SQL и нажать кнопку Go

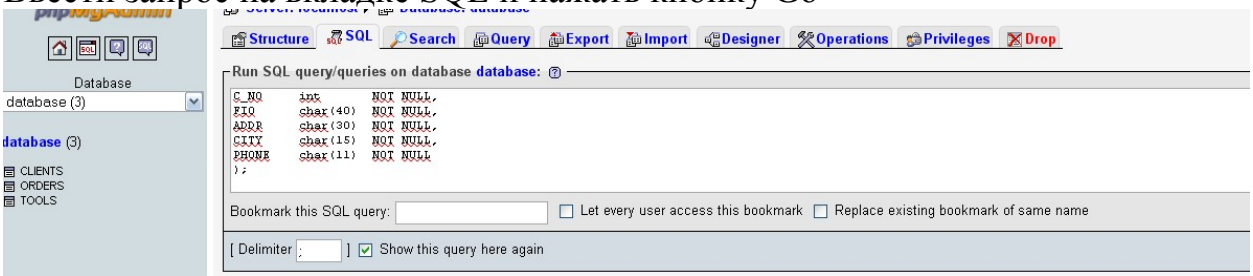


Рисунок 5.1 – Создание БД

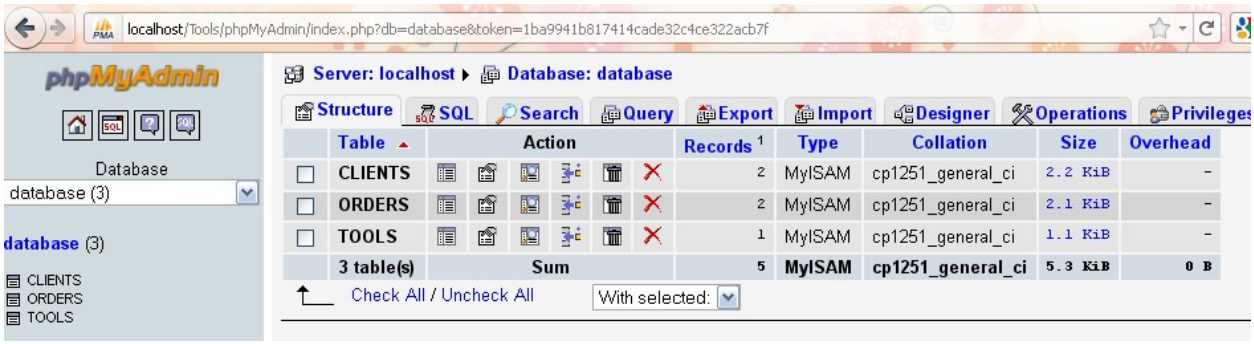


Рисунок 5.2 – Работа с БД

Задание

Создайте БД, состоящую из 3 таблиц.

Содержание отчета и его форма

Отчет по лабораторной работе должен состоять из:

1) названия лабораторной работы;
2) созданных таблиц с указанием ключевых полей и указанием запрета неопределенных значений.

Вопросы для защиты работы:

1. Какие атрибуты могут быть первичными ключами?
2. Каким образом можно запретить ввод неопределенных значений?

Лабораторная работа №6. Функции PHP для работы с MySQL

Цель работы: научиться использовать php функции для работы с базой данных MySQL.

Теоретическое обоснование

Рассмотрим основные функции PHP, применяемые для работы с MySQL сервером.

Функции соединения с сервером MySQL

Основной функцией для соединения с сервером MySQL является mysql_connect(), которая подключает скрипт к серверу баз данных MySQL и выполяет авторизацию пользователя базой данных.

Синтаксисуданнойфункциитакой:

mysql_connect ([string $hostname] [, string $user] [, sting $password]);

Как вы наверно заметили, все параметры данной функции являются необязательными, поскольку значения по умолчанию можно прописать в конфигурационном файле php.ini. Если вы хотите указать другие имя MySQL-хоста, пользователя и пароль, вы всегда можете это сделать.

Параметр $hostname может быть указан в виде: хост:порт.

Функция возвращает идентификатор (типа int) соединения, вся дальнейшая работа осуществляется только через этот идентификатор. При следующем вызове функции mysql_connect() с теми же параметрами новое

соединение не будет открыто, а функция возвратит идентификатор существующего соединения.

Для закрытия соединения предназначена функция mysql_close(int $connection_id).

Вообще, соединение можно и не закрывать - оно будет закрыто автоматически при завершении работы PHP скрипта. Если вы используете более одного соединения, при вызове mysql_close() нужно указать идентификатор соединения, которое вы хотите закрыть. Вообще не закрывать соединения - плохой стиль, лучше закрывать соединения с MySQL самостоятельно, а не надеясь на автоматизм PHP, хотя это ваше право.

Если вы будете использовать только одно соединение с базой данных MySQL за все время работы сценария, можно не сохранять его идентификатор и не указывать идентификатор при вызове остальных функций.

Функция mysql_connect() устанавливает обыкновенное соединение с MySQL. Однако, PHP поддерживает постоянные соединения - для этого используйте функцию mysql_pconnect(). Аргументы этой функции такие же, как и у mysql_connect().

В чем разница между постоянным соединением и обыкновенным соединением с MySQL? Постоянное соединение не закрывается после завершения работы скрипта, даже если скрипт вызвал функцию mysql_close(). Соединение привязывается к PID потомка веб сервера Apache (от имени которого он и работает) и закрывается лишь тогда, когда удаляется процесс-владелец (например, при завершении работы или перезагрузке вебсервера Apache).

PHP работает с постоянными соединениями примерно так: при вызове функции mysql_pconnect() PHP проверяет, было ли ранее установлено соединение. Если да, то возвращается его идентификатор, а если нет, то открывается новое соединение и возвращается идентификатор.

Постоянные соединения позволяют значительно снизить нагрузку на сервер, а также повысить скорость работы PHP скриптов, использующих базы данных.

При работе с постоянными соединениями нужно следить, чтобы максимальное число клиентов Apache не превышало максимального числа клиентов MySQL, то есть параметр MaxClient (в конфигурационном файле Apache - httpd.conf) должен быть меньше или равен параметру max_user_connection (параметр MySQL).

Функция выбора базы данных

Функция mysql_select_db (string $db [, int $id]) выбирает базу данных, с которой будет работать PHP скрипт. Если открыто не более одного соединения, можно не указывать параметр $id.

// Попытка установить соединение с MySQL: if (!mysql_connect($server, $user, $ password)) { echo "Ошибка подключения к серверу MySQL"; exit; }

// Соединились, теперь выбираем базу данных: mysql_select_db($db);

Функции обработки ошибок

Если произойдет ошибка соединения с MySQL, то вы получите соответствующее сообщение и скрипт завершит свою работу. Это не всегда бывает удобно, прежде всего, при отладке скриптов. Поэтому, в PHP есть следующие две функции:

- mysql_errno(int $id);
- mysql_error(int $id);

Первая функция возвращает номер ошибки, а вторая - сообщение об ошибке. В результате мы можем использовать следующее:

echo "ERROR ".mysql_errno()." ".mysql_error()."\n";

Теперь вы будете знать, из-за чего произошла ошибка - вы увидите соответствующим образом оформленное сообщение.

Функции выполнения запросов к серверу баз данных

Все запросы к текущей базе данных отправляются функцией mysql_query(). Этой функции нужно передать один параметр - текст запроса. Текст запроса модет содержать пробельные символы и символы новой строки (\n). Текст должен быть составлен по правилам синтаксиса SQL.

Пример запроса:

$q = mysql_query("SELECT * FROM mytable");

Приведенный запрос должен вернуть содержимое таблицы mytable. Результат запроса присваивается переменной $q. Результат - это набор данных, который после выполнения запроса нужно обработать определенным образом.

Функции обработки результатов запроса

Если запрос, выполненный с помощью функции mysql_query() успешно выполнился, то в результате клиент получит набор записей, который может быть обработан следующими функциями PHP:

- mysql_result() - получить необходимый элемент из набора записей;
- mysql_fetch_array() - занести запись в массив;
- mysql_fetch_row() - занести запись в массив;
- mysql_fetch_assoc() - занести запись в ассоциативный массив;
- mysql_fetch_object() - занести запись в объект.

Также можно определить количество содержащихся записей и полей в результате запроса. Функция mysql_num_rows() позволяет узнать, сколько записей содержит результат запроса:

$q = mysql_query("SELECT * FROM mytable"); echo "Втаблице mytable ".mysql_num_rows($q)." записей";

Запись состоит из полей (колонок). С помощью функции mysql_num_fields() можно узнать, сколько полей содержит каждая запись результата:

$q = mysql_query("SELECT * FROM mytable"); echo "Втаблице mytable ".mysql_num_fields($q)." полей ";

У нас также есть возможность узнать значение каждого поля. Это можно сделать с помощью следующей функции:

mysql_result (int $result, int $row, mixed $field);

Параметр функции $row задает номер записи, а параметр $field - имя или порядковый номер поля.

Предположим, SQL-запрос вернул следующий набор данных:

Email Name Last_Name ---------- ---------- ----------
ivanov@mail.ru Ivan Ivanov petrov@mail.ru Petr Petrov

Вывести это в браузер можно следующим образом:

```
$rows = mysql_num_rows($q);
$fields = mysql_num_fields($q);

echo "<pre>"; for ($c=0; $c<$rows; $c++) { for ($cc=0; $cc<$fields; $cc++) { echo mysql_result($q, $c, $cc)."\t"; echo "\n";
    }
  }
echo "</pre>";
```

Следует отметить, что функция mysql_result() универсальна: зная количество записей и количество полей, можно "обойти" весь результат, но в тоже время, скорость работы данной функции достаточно низка. Поэтому, для обработки больших наборов записей рекомендуется использовать функции mysql_fecth_row(), mysql_fecth_array(), и.т.д.

Функция mysql_fecth_row(int $res) получает сразу всю строку, соответствующую текущей записи результата $res. Каждый следующий вызов функции перемещает указатель запроса на следующую позицию (как при работе с файлами) и получает следующую запись. Если более нет записей, то функция возвращает FALSE. Пример использования данной функции:

```
$q = mysql_query("SELECT * FROM mytable WHERE month=\"$db_m\"
AND day=\"$db_d\"); for ($c=0; $c<mysql_num_rows($q); $c++)
{
$f = mysql_fetch_row($q); echo $f;
}
```

Использовать функцию mysql_fecth_row() не всегда удобно, так как значения всех полей одной записи находятся все в одной строке. Удобнее использовать функцию mysql_fetch_array(), которая возвращает ассоциативный массив, ключами которого будут имена полей.

Функция mysql_fecth_array(int $res [, int $result_type]) возвращает не ассоциативный массив, а массив, заданный необязательным параметром $result_type, который может принимать следующие значения:

- MYSQL_ASSOC - возвращает ассоциативный массив;
- MYSQL_NUM -возвращает массив с числовыми индексами, как в функции mysql_fecth_row();
- MYSQL_BOTH - возвращает массив с двойными индексами, то есть вы можете работать с ним, как с ассоциативным массивом и как со списком (MYSQL_BOTH - это значение по умолчанию для параметра

$result_type.

В PHP есть функция, возвращающая ассоциативный массив с одним индексом:

mysql_fetch_assoc(int $res);

Фактически, данная функция является синонимом для mysql_fetch_array($res, MYSQL_ASSOC);

Пример использования функции mysql_fecth_array():

$q = mysql_query("SELECT * FROM mytable WHERE month=\"$db_m\" AND day=\"$db_d\"); for ($c=0; $c<mysql_num_rows($q); $c++)

{

$f = mysql_fetch_array($q); echo "$f[email] $f[name] $f[month] $f[day]
"; }

Как видно, использовать функцию mysql_fetch_array() намного удобнее, чем mysql_fetch_row().

Функции получения информации о результатах SQL-запросов

PHP предоставляет еще несколько полезных функций, которые позволяют узнать информацию о результатах SQL-запросов.

- Функция mysql_field_name(int $result, int $offset) возвращает имя поля, находящегося в результате $result с номером $offset (нумерация начинается с 0). Другими словами, функция возвращает имя поля с номером $offset.
- Функция mysql_field_type(int $result, int $offset) возвращает тип поля с номером $offset в результате $result (номер задается относительно результата, а не таблицы);
- Функция mysql_field_flags(int $result, int $offset) возвращает пречисленные через пробел флаги (модификаторы), которые имеются у поля с номером $offset. Перечислим все поддерживаемые MySQL флаги (Таблица 6.1).

Таблица 6.1 – Флаги MySQL

Флаг	Описание
not_Null	Поле не может содержать неопределенного значения (NULL), то есть поле должно быть явно инициализировано
Primary_Key	Поле будет первичным ключом - идентификатором записи, по которому можно однозначно идентифицировать запись;
auto_increment	При вставке новой записи значение этого поля будет автоматически увеличено на единицу, потому в таблице никогда не будет двух записей с одинаковым значением этого поля;
Unique_Key	Поле должно содержать уникальное значение;
Multiple_Key	Индекс
Blob	Поле может содержать бинарный блок данных
Unsigned	Поле содержит беззнаковые числа
Zerofill	Вместо пробелов используются символы с кодом

	\0
Binary	Поле содержит двоичные данные
enum	Поле может содержать один элемент из нескольких возможных (элемент перечисления)
timestamp	В поле автоматически заносится текущая дата и время при его модификации

Функция mysql_field_flags() возвращает флаги в виде строки, в которой флаги разделяются пробелами.

Практический пример использования функций PHP-MySQL

Скрипт вывода содержимого таблицы MySQL в виде HTML:

```
<?php
$host = "localhost";
$user = "user";
$password = "secret_password";

// Производим попытку подключения к серверу MySQL:
if (!mysql_connect($host, $user, $password))
{
echo "<h2>MySQL Error!</h2>"; exit;
}

// Выбираем базу данных:
mysql_select_db($db);

// Выводим заголовок таблицы:
echo "<table border=\"1\" width=\"100%\" bgcolor=\"#FFFFE1\">"; echo
"<tr><td>Email</td><td>Имя</td><td>Месяц</td>"; echo
"<td>Число</td><td>Пол</td></tr>";

// SQL-запрос:
$q = mysql_query ("SELECT * FROM mytable");

// Выводим таблицу:
for ($c=0; $c<mysql_num_rows($q); $c++)
{ echo "<tr>";

$f = mysql_fetch_array($q); echo
"<td>$f[email]</td><td>$f[name]</td><td>$f[month]</td>"; echo
"<td>$f[day]</td><td>$[s]</td>";

echo "</tr>";
} echo "</table>";
?>
```

Методика и порядок выполнения работы

Создадим в папке denver-home папку для нашего скрипта. Назовем ее new. В папке new создадим папку www. В папке www создадим файл index.php, в который впишем следующий код, выводящий содержимое таблицы CLIENTS:

```
<?php
$host = "localhost";
$user = "root";//пользователь, созданный по умолчанию в базе данных без пароля
$password = "";

// Производим попытку подключения к серверу MySQL:
if (!mysql_connect($host, $user, $password))
{
echo "<h2>MySQL Error!</h2>"; exit; }
$db="database";//указываем нашу базу данных
// Выбираем базу данных: mysql_select_db($db);

// Выводимзаголовоктаблицы: echo "<table border=\"1\" width=\"100%\" bgcolor=\"#FFFFE1\">"; echo "<tr><td>Номер</td><td>ФИО</td><td>Адрес</td>"; echo "<td>Город</td><td>Номер</td></tr>";

// SQL-запрос:
$q = mysql_query ("SELECT * FROM ClIENTS");

// Выводимтаблицу: for ($c=0; $c<mysql_num_rows($q); $c++)
{ echo "<tr>";

$f = mysql_fetch_array($q); echo "<td>$f[C_NO]</td><td>$f[FIO]</td><td>$f[ADDR]</td>"; echo "<td>$f[CITY]</td><td>$f[PHONE]</td>";

echo "</tr>";
} echo "</table>";
?>
```

В результате получаем таблицу 6.2.

Таблица 6.2 – Таблица CLIENTS

Номер	ФИО	Адрес	Город	Номер
1	Иванов И.И.	Вокзальная 3	Псков	0959991110
2	Петров	Мира 29	Екатеринбург	-
3	Иванов		Псков	

Выведем записи полей всех таблиц

```
$f = mysql_fetch_array($q); echo
"<td>$f[C_NO]</td><td>$f[FIO]</td><td>$f[ADDR]</td>"; echo
"<td>$f[CITY]</td><td>$f[PHONE]</td>";

echo "</tr>";
} echo "</table><br><br><br><br>";

// ВыводимзаголовоктаблицыЗаказы:
echo "<table border=\"1\" width=\"100%\" bgcolor=\"#FFFFE1\">"; echo
"<tr><td>Номер</td><td>Дата</td>"; echo
"<td>Клиент</td><td>Номер</td><td>Точность</td><td>цена</td></tr>
";

// SQL-запрос:
$q = mysql_query ("SELECT * FROM ORDERS");

// Выводимтаблицу: for ($c=0; $c<mysql_num_rows($q); $c++)
{ echo "<tr>"; $f = mysql_fetch_array($q); echo
"<td>$f[O_NO]</td><td>$f[DATE]</td><td>$f[C_NO]</td>"; echo
"<td>$f[T_NO]</td><td>$f[QUANTITY]</td><td>$f[AMOUNT]</td>";

echo "</tr>";
} echo "</table><br><br><br><br>";

// ВыводимзаголовоктаблицыИнструменты: echo "<table border=\"1\"
width=\"100%\" bgcolor=\"#FFFFE1\">"; echo
"<tr><td>Номер</td><td>Название</td>"; echo
"<td>Цена</td><td>Количество</td></tr>";

// SQL-запрос:
$q = mysql_query ("SELECT * FROM TOOLS");

// Выводимтаблицу: for ($c=0; $c<mysql_num_rows($q); $c++)
{ echo "<tr>";
```

```
$f = mysql_fetch_array($q); echo
"<td>$f[T_NO]</td><td>$f[DSEC]</td><td>$f[PRICE]</td>"; echo
"<td>$f[QTY]</td>";

echo "</tr>";
} echo "</table>";
?>
```

Организуем ввод данных в базу данных.
Сначала создадим форму для ввода данных. Добавим в код файла index.php

```
<br><br><br>
<form action="vvod.php" method="post">
<table width="60%" border="0" >
<tr>
<td width="34%">Номер</td>
<td width="66%"><input name="c_no" type="text" size="13" maxlength="13" /></td>
</tr>
<tr>
<td>ФИО</td>
<td><input name="fio" type="text" size="30" maxlength="30" /></td>
</tr>
<tr>
<td>Адрес</td>
<td><input name="addr" type="text" size="30" maxlength="60" /></td>
</tr>
<tr>
<td>Город</td>
<td><input name="city" type="text" size="8" maxlength="8" /></td>
</tr>
<tr>
<td>Телефон</td>
<td><input name="phone" type="text" size="8" maxlength="8" /></td>
</tr>
<tr>
<td colspan="2"><input name="input" type="submit" value="Зарегистрировать" /></td>
</tr>
</table>
</form>
</table>
```

Создадим скрипт vvod.php, обрабатывающий введенные в форму данные

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.111.ru">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=windows-
1251" />
<title>Запись в базу данных</title>
</head>

<body>
<h1>Запись в базу данных</h1>
<?php

/*if (!$c_no || !$fio || !$addr || !$city || $phone)
{
echo 'Вы ввели не все необходимые сведения. <br>'
.'Пожалуйста, вернитесь на предыдущую страницу и повторите ввод.';
exit;
}
/*if (!get_magic_quotes_gpc())
{
$isbn = addslashes ($isbn);
$author = addslashes ($author);
$title = addslashes ($title);
$price = doubleval ($price);
}*/
// Производим попытку подключения к серверу MySQL:
$host = "localhost";
$user = "ya"; $password = "1111"; if (!mysql_connect($host, $user,
$password))
{
echo "<h2>MySQL Error!</h2>";
exit;
}
$db="database";
// Выбираем базу данных: mysql_select_db($db);
//создание коротких имен переменных
$c_no = $_POST['c_no'];
$fio = $_POST['fio'];
$addr = $_POST['addr'];
$city = $_POST['city'];
$phone = $_POST['phone'];

$query = "insert into CLIENTS values(", $fio, $addr, $city, $phone,")";
```

```
$result = mysql_query($query) or die("Query failed"); echo $result; if ($result) echo $db -> affected_rows. " добавлено.";
?>
</body>
</html>
```

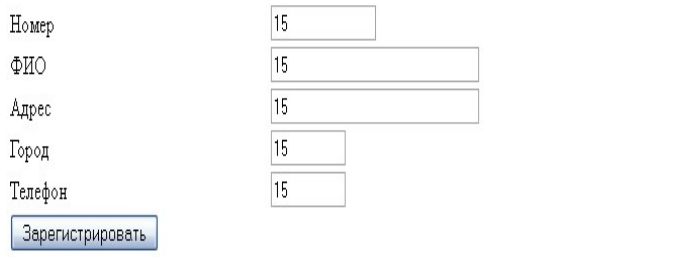
Не забудьте закрыть соединение с базой данных в скрипте index.php: mysql_close();

Проверяем работоспособность скрипта. Результаты представлены в таблицах 6.3а – 6.3д.

Номер	ФИО	Адрес	Город	Номер
1	Иванов И.И.	Вокзальная 3	Псков	09599911100
2	Петров	Мира 29	Екатеринбург	-

Номер	Дата	Клиент	Номер	Точность	цена
1	2020-01-03	1	1	1.00	340.98
1	2010-01-03	1	1	1.00	340.98

Номер	Название	Цена	Количество
1	Клавиатура ABC	340.98	5.00



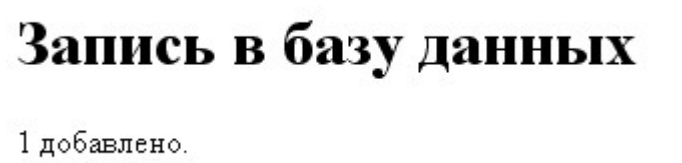


Таблица 6.3-Результаты работы скрипта

а

б

в

г

д

Проверяем, добавлена ли запись в базу данных (таблица 6.4)

Таблица 6.4 – Добавление записей

			C_NO	FIO	ADDR	CITY	PHONE	ZIP
☐	✎	✕	1	Иванов И.И.	Вокзальная 3	Псков	09599911100	*NULL*
☐	✎	✕	2	Петров	Мира 29	Екатеринбург	-	*NULL*
☐	✎	✕	0	15	15	15	15	

Запись добавлена, но, поскольку мы не вписывали в запросе номер для клиента, то добавилось нулевое значение. Логичнее будет изменить поле C_NO, сделав его автоикрементом (рисунок 6.1).

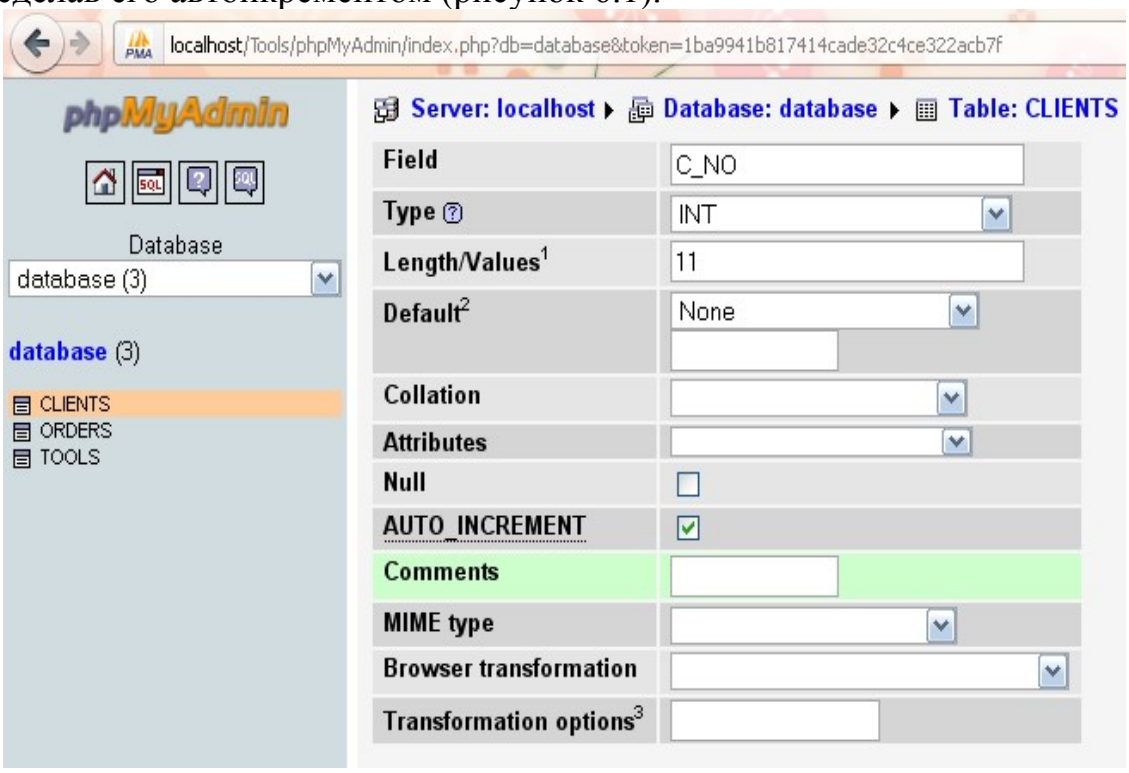


Рисунок 6.1 – Добавление AUTO_INCREMENT

Проверьте, как будет добавляться запись теперь.

Задание

1. Организовать ввод данных для всех таблиц.
2. С помощью web –интерфейса организовать выборку из базы данных по заданному пользователем условию.

Содержание отчета и его форма

Отчет по лабораторной работе должен состоять из:

1) названия лабораторной работы;
2) описание добавления строк в таблицы.

Вопросы для защиты работы:

1. Каким образом осуществляется заполнение таблиц?
2. Для чего используется AUTO_INCREMENT?

Лабораторная работа №7. Удаление данных из БД. Обновление данных в БД

Цель работы: научиться удалять и обновлять записи в таблицах базы данных MySQL.

Теоретическое обоснование

В предыдущей работе вы научились вставлять и запрашивать данные из базы данных (БД). В этой работе мы узнаем, как удалять записи из БД, что значительно проще, чем вставка данных.

Удаление данных с помощью SQL

Синтаксис SQL-оператора удаления записей таков:

DELETEFROMTableNameWHEREcondition

При удалении записи можно использовать уникальное поле AutoNumber в базе данных. В нашей БД это столбец C_NO таблицы Clients. Использование этого уникального идентификатора гарантирует, что удаляется только одна запись. Удалим клиента с номер 3.

```
<html>
<head>
<title>Удаление данных из БД</title>
</head>
<body>
<?php
// Соединение с сервером БД  mysql_connect("localhost", "root", "") or die (mysql_error ());
```

```
// ВыборБд  mysql_select_db("database") or die(mysql_error());  // SQL-оператор, удаляющийзапись
$strSQL = "DELETE FROM CLIENTS WHERE C_NO = 3";
mysql_query($strSQL);
// Закрыть соединение с БД  mysql_close();
?>
<h1>Записьудалена!</h1>
</body>
</html>
```

Помните, что не существует никакой "Recycle Bin" при работе с БД и PHP. Если вы удалили запись, то восстановить её будет невозможно.

Обновление данных с помощью SQL

Синтаксис SQL-оператора обновления полей таблицы:

```
UPDATE TableName SET TableColumn='value' WHERE condition
```

Можно также обновлять несколько ячеек за раз, используя один оператор SQL:

```
UPDATE TableName SET TableColumn1='value1', TableColumn2='value2' WHERE condition
```

Следующий код обновляет FIO в CLIENTS на «Петров» и меняет телефонный номер на 44444444. Прочая информация не изменяются. Можете попробовать изменить другие данные, создав собственные SQL-операторы.

```
<html>
<head>
<title>Обновление данных в БД</title>
</head>
<body>
<?php
// Соединение с сервером БД  mysql_connect("localhost", "root", "") or die (mysql_error ());
// ВыборБД  mysql_select_db("database") or die(mysql_error());
// Построение SQL-оператора
$strSQL = "Update CLIENTS set ";
$strSQL = $strSQL. "FIO= 'Петров', ";
$strSQL = $strSQL. "Phone= '44444444' ";
$strSQL = $strSQL. "Where C_NO = 3";
// SQL-оператор выполняется  mysql_query($strSQL);
// Закрыть соединение с БД  mysql_close();  ?>
<h1>База обновлена!</h1>
```

```
</body>
</html>
```

Задание.

1. Организовать обновление базы данных посредством web интерефейса (кнопкой).
2. Организовать удаление записей базы данных посредством web интерефейса, определенных пользователем.

Содержание отчета и его форма

Отчет по лабораторной работе должен состоять из:

1) названия лабораторной работы;
2) описание удаления строк из таблицы.

Вопросы для защиты работы:

1. Каким образом осуществляется удаление таблиц?
2. Каким образом осуществляется удаление строк?

Лабораторная работа №8. Введение в Php сессии

Цель работы: научиться удалять и обновлять записи в таблицах базы данных MySQL.

Теоретическое обоснование

При посещении сайта вы выполняете различные действия. Переходите с одной страницы на другую. Возможно, заполняете форму или покупаете что-то. Это очень важно учитывать при создании успешных веб-проектов.

Предположим, например, что вы хотите создать сайт, на котором несколько страниц защищены логином и паролем. Чтобы эта защита действовала эффективно, защищённые паролем страницы должны иметь доступ к информации о том, зашёл ли пользователь ранее в систему. Вы должны, иначе говоря, "помнить", что пользователь делал до этого.

Именно об этом наша лабораторная работа - как использовать сессии в PHP для сохранения и получения информации в процессе визита пользователя на наш сайт.

PHP-сессии дают возможность работать с информацией о пользовательской сессии. Вы можете создавать приложения, которые идентифицируют и собирают информацию о пользователях.

Сессии могут начинаться разными способами. Мы не будем углубляться в технические тонкости, а сконцентрируемся на варианте, когда сессия начинается с сохранения значения. Сессия заканчивается (dies), если пользователь не запрашивает страниц в течение какого-то времени (стандартное значение - 20 минут). Разумеется, вы в любой момент можете закончить сессию в вашем скрипте.

Скажем, 50 пользователей просматривают страницы одного сайта, например, веб-магазина. Информацию о том, что у каждого посетителя в корзине, лучше всего сохранить в сессии. Чтобы идентифицировать пользователей, сервер использует уникальные пользовательские идентификаторы/user ID, которые хранятся в куках. Кука – это небольшой текстовый файл, хранимый на компьютере пользователя. Следовательно, сессии часто требуют поддержки кук в браузерах пользователей.

При запросе страницы сохраним текущее время в сессии. Добавим следующую строку в PHP-скрипт:

```
<?php

session_start();
$_SESSION["StartTime"] = date("r");

?>
```

Таким образом, сессия началась. Как сказано выше, каждая сессия получает ID от сервера.

Ваша сессия имеет следующий ID: dae18cd921b51edf04b358d76d15e7e7

В любое время можно вызвать "StartTime" из сессии, введя:

```
<?php
session_start();  echo $_SESSION["StartTime"];
?>
```

что покажет, что страница была запрошена в (в соответствии с временем данного вэб-сервера).

Но интересно, что эта информация остаётся в сессии, даже после выхода со страницы. Эта информация будет сопровождать, пока сессия не завершится.

По умолчанию сессия длится, пока пользователь не закроет окно браузера, и тогда она заканчивается автоматически. Но если вы хотите принудительно завершить сессию, её всегда можно закончить таким образом:

```
<?php
session_destroy();
?>
```

Посмотрим другой пример использования сессий: с паролем. Создадим простейшую систему с логином. Первое, что необходимо, это форма, в которой пользователи могу указывать username и password.

Онаможетвыглядетьтак (index.html):

```
<html>
<head>
<title>Login</title>
</head>
<body>
<form method="post" action="login.php">
<p>Username: <input type="text" name="username" /></p>
<p>Password: <input type="text" name="password" /></p>
<p><input type="submit" value="Войти" /></p>
</form>
</body>
</html>
```

Затемсоздадимфайлlogin.php. Вэтомфайлемыпроверяем, введеныликорректныеusernameиpassword. Если это так, мы начинаем сессию, в которой указано, что пользователь вошёл с корректными username и password.

```
<?php
session_start();
?>
```

```
<html>
<head>
<title>Login</title>
</head>
<body>
<?php
// Проверитькорректностьusernameиpassword if ($_POST["username"] == "1" && $_POST["password"] == "1") {
// Если корректны, устанавливаем значение сессии в YES

$_SESSION["Login"] = "YES"; echo "<h1>Вызашли</h1>"; echo "<p><a href='document.php'>Ссылка на защищённый файл</a><p/>"; } else {
// Если некорректны, устанавливаем сессию в NO session_start();
$_SESSION ["Login"] = "NO"; echo "<h1>Вы зашли НЕкорректно </h1>"; echo "<p><ahref='document.php'>Ссылка на защищённый файл</a><p/>";
}
?>
</body>
</html>
```

При работе с защищёнными файлами мы проверяем, вошёл ли пользователь с корректным логином. Если нет, то пользователь отправляется обратно к логин-форме. Вот как делается эта защита:

```
<?php
// Начать вашу PHP-сессию session_start();
// Если пользователь не зашёл, отправить его/её к логин-форме if ($_SESSION["Login"] != "YES") { header("Location: index.html"); echo "Вы можете получить к нему доступ, только если вошли в систему.";
}
?>
<html>
<head>
<title>Логин</title>
</head> <body>
<h1>Этот документ защищён</h1>
<p></p>
</body>
</html>
```

Задание

1. Организовать сессию в вашей системе без пароля.
2. Организовать сессию вашей системы с паролем и логином.

Содержание отчета и его форма

Отчет по лабораторной работе должен состоять из:

1) названия лабораторной работы;
2) описание сессии без пароля;
3) описание сессиис паролем и логином.

Вопросы для защиты работы:

1. Как использовать сессии в PHP для сохранения и получения информации?
2. Каким образом используются пароль и логин?

Лабораторная работа №9. Передача переменных через URL

Цель работы: научиться использовать значения переменных, передаваемых через URL

Теоретическое обоснование

При работе с PHP часто необходимо передать переменные с одной страницы в другую. Возможно, вас удивляло, почему некоторые URL выглядят так:

http://html.net/page.php?id=1254

Почему после имени страницы стоит знак вопроса? Ответ: символы после знака вопроса это строка HTTP-запроса. Строка HTTP-запроса может содержать как имена переменных, так и их значения. В вышеприведённом примере строка HTTP-запроса содержит переменную "id" со значением "1254".

Вот другой пример: http://html.net/page.php?name=Joe

То есть снова переменная ("name") со значением ("Joe"). Как получить переменную с помощью PHP? Предположим, у вас есть PHP-страница people.php. Можно вызвать её с использованием URL:

people.php?name=Joe

В PHP вы можете получить значение переменной 'name' таким образом:

$_GET["name"]

То есть мы используем$_GET для поиска значения именованной переменной. Попробуем на примере:

```
<html>
<head>
<title>Строка запроса</title>
</head>
<body>
<?php// Значение переменной найдено echo "<h1>Hello ". $ _GET["name"]. "</h1>";
?>
</body>
</html>
```

Попробуйте в этом примере заменить "Joe" вашим собственным в URL и снова вызвать документ. Опишите полученные результаты.

В URL можно передавать и не одну переменную. Разделяя переменные знаком &, можно передавать несколько:

people.php?name=Joe&age=24

Этот URL содержит две переменные: name и age. Как и ранее, можно получит переменные так:

$_GET["name"]
$_GET["age"]

Добавим в пример ещё одну переменную:

```
<html>
<head>
<title>Строказапроса</title>
</head>
<body>
<?php
// Значение имени переменной name найдено  echo "<h1>Hello ". $_GET["name"]. "</h1>";
// Значение имени переменной age найдено  echo "<h1>You are ". $_GET["age"]. " years old </h1>";
?>
</body>
</html>
```

Методика и порядок выполнения работы Изучить передачу переменных по приведенным примерам.

Задание

1. Организовать передачу переменных по приведенным примерам.

Содержание отчета и его форма

Отчет по лабораторной работе должен состоять из:

1) названия лабораторной работы;
2) описание передачи переменныхчерез URL.

Вопросы для защиты работы:

1. Дайте характеристику URL?
2. Каким образом осуществляется передача переменныхчерез URL?

Лабораторная работа №10.Использование внешнего файла для хранения и загрузки внешних данных

Цель работы: научиться организовывать использование внешнего файла для хранения и загрузки внешних данных.

Теоретическое обоснование

Текстовые файлы отлично подходят для хранения разного рода данных. Они не так гибки, как базы данных, но обычно не требую такого количества памяти. Более того, текстовые файлы имеют формат, который читается на большинстве систем.

Для открытия текстового файла используем функцию fopen. Вот её синтаксис:

fopen(filename, mode) где filename – имя открываемого файла.

mode – Mode/Режим может быть "r" (reading/чтение), "w" (writing/запись) или "a" (appending/присоединение). В этом уроке мы будем только читать из файла и, соответственно, используем "r".

Создадим файл unitednations.txt, впишем в него информацию и попробуем открыть его:

```
<?php
// Открыть текстовый файл
$f = fopen("unitednations.txt", "r");
// Закрыть текстовый файл  fclose($f);
?>
```

С помощью функции fgets можно читать строку из текстового файла. Этот метод читает до первого символа переноса строки (но не включая символ переноса строки).

```
<html>
<head>
<title>Чтение из текстовых файлов</title>
</head>
<body>
<?php
$f = fopen("unitednations.txt", "r");
// Читать строку их текстового файла и записать содержимое клиенту
echo fgets($f);   fclose($f);
?>
</body>
</html>
```

Организуем чтение всех строк текстового файла

```
<html>
<head>
<title>Чтение из текстовых файлов</title>
</head>
<body>
<?php
$f = fopen("unitednations.txt", "r");
// Читать построчно до конца файла  while(!feof($f)) {   echo fgets($f). "<br />";
}
fclose($f);
?>
</body>
</html>
```

В этом коде мы выполняем цикл по всем строкам и используем функцию feof (for end-of-file/до конца файла) для проверки достижения конца файла. Если конец не достигнуть, строка записывается. Вместо цикличного прохода по всем строкам мы можем получить тот же результат функцией fread. При работе с очень большими текстовыми файлами помните, что fread использует больше ресурсов, чем fgets. Для маленьких файлов разница в работе несущественна.

Итак, текстовые файлы могут отлично подойти для хранения данных. Это показано на следующем примере, где создаётся простая директория ссылок из содержимого файла unitednations.txt.

В файле систематизировано запишем: название программы, запятая, домен. Как вы, вероятно, могли предположить, в файле с разделением запятыми можно записать куда больше информации. Для получения информации из каждой строки используем массив.

```
<html>
<head>
<title>Чтение из текстовых файлов</title>
</head>
<body>
<?php
$f = fopen("unitednations.txt", "r");
// Читать построчно до конца файла  while (!feof($f)) {
// Создать массив с запятой-разделителем $arrM = explode(",",fgets($f));
// Записать ссылки (получить данные из массива)  echo "<li><a href='http://". $arrM[1]. "'>". $arrM[0]. "</a></li>";
}
```

```
fclose($f);
?>
</body>
</html>
```

Задание

Изучить теоретическое обоснование и приведенные в нем примеры.

Содержание отчета и его форма

Отчет по лабораторной работе должен состоять из:

1) названия лабораторной работы;
2) описание использования текстовых файлов для хранения данных.

.

Вопросы для защиты работы:

1. Как использовать текстовые файлы для хранения данных?
2. Каким образом осуществляется работа с массивами?

Лабораторная работа №11. PHP и поля HTML-форм: текстовые поля, текстовая область, флажки, переключатели, списки

Цель работы: научиться использовать элементы HTML-форм совместно с языком PHP.

Теоретическое обоснование

Использование элемента дизайна форма при разработке Web-сайта является наиболее популярным способом организации интерактивного взаимодействия с его посетителями. С помощью языка HTML можно создавать как простые, так и сложные формы, предполагающие множественный выбор из нескольких вариантов. Формы состоят из одного или несколько полей, в которые пользователь может ввести различную информацию, либо выбрать какую-то опцию. После ввода информации, она передается на сервер, где может обрабатываться различными средствами, в том числе, с помощью языка PHP.

В ниже представленных примерах описывается обработка информации пользователя из форм. К элементам форм относятся однострочное текстовое поле, текстовая область, переключатели, флажки, списки, скрытые поля форм, поля ввода паролей и кнопки. В примерах используются две Webстраницы. Первая страница содержит форму для ввода данных пользователя и имеет расширение .html. Вторая страница обрабатывает введенную информацию средствами языка PHP и имеет расширение .php.

1 Текстовые поля

Текстовые поля являются одними из наиболее известных элементов управления формами. В них пользователь Web-страницы может ввести любую

информацию. Текстовые поля ввода создаются с помощью тега <INPUT>, в котором атрибуту “type” присваивается значение “text”.

Синтаксис:

<INPUT type="text" name="TextBox" size="n" maxlength="m"> где type - тип поля, name - имя поля как элемента формы, size - размер видимой части текстового поля на экране, maxlength - размер поля.

Ниже представлен пример обработки информации из текстового поля. В данном примере присутствует HTML-форма для ввода имени любимого автора. Программа считывает введенную информацию и осуществляет вывод ее на экран.

HTML-код формы для ввода информации представлен ниже.

```
<HTML>
<HEAD>
<TITLE>Пример использования текстового поля</TITLE>
</HEAD>
<BODY>
<FORM method= "GET" action="text.php">
КтоВашлюбимыйавтор?
<INPUT name="Author" type="text">
<BR><BR>
<INPUT type="submit" value= "Отправить">
</FORM>
</BODY>
</HTML>
```

Реализация данного кода представлена на рисунке 11.1

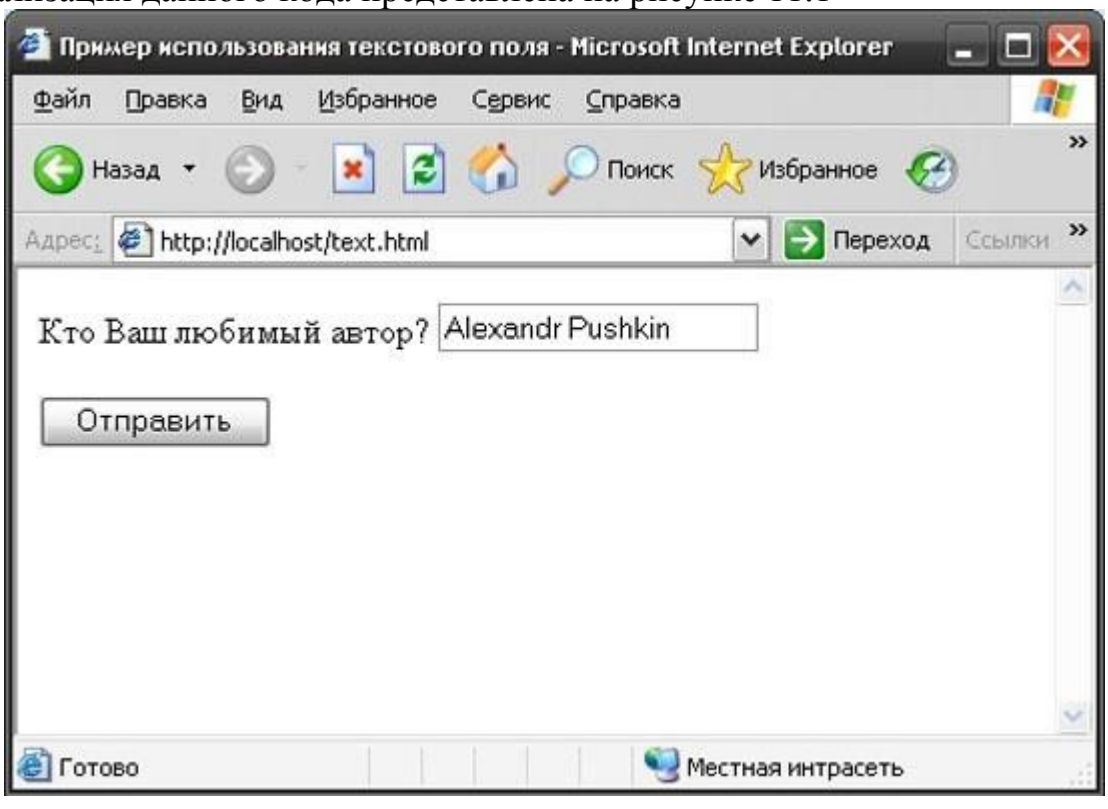


Рисунок 11.1 - Ввод любимого автора

Таким образом, как видно на рисунке 11.1, в данном примере, при загрузке Web-страницы, на экране появляется однострочное текстовое поле, в которое пользователю необходимо ввести информацию и нажать кнопку “Отправить”. После этого подключится обработчик, указанный в атрибуте “action” тега “form”. В данном примере это файл text.php.

Код файла-обработчика представлен ниже.

```
<HTML>
<HEAD>
<TITLE>Пример использования текстового поля</TITLE>
</HEAD>
<BODY>
<B>Ваш любимый автор:<B>
<?php echo $_GET['Author'];
?>
</BODY>
</HTML>
```

Реализация данного кода представлена на рисунке 11.2.

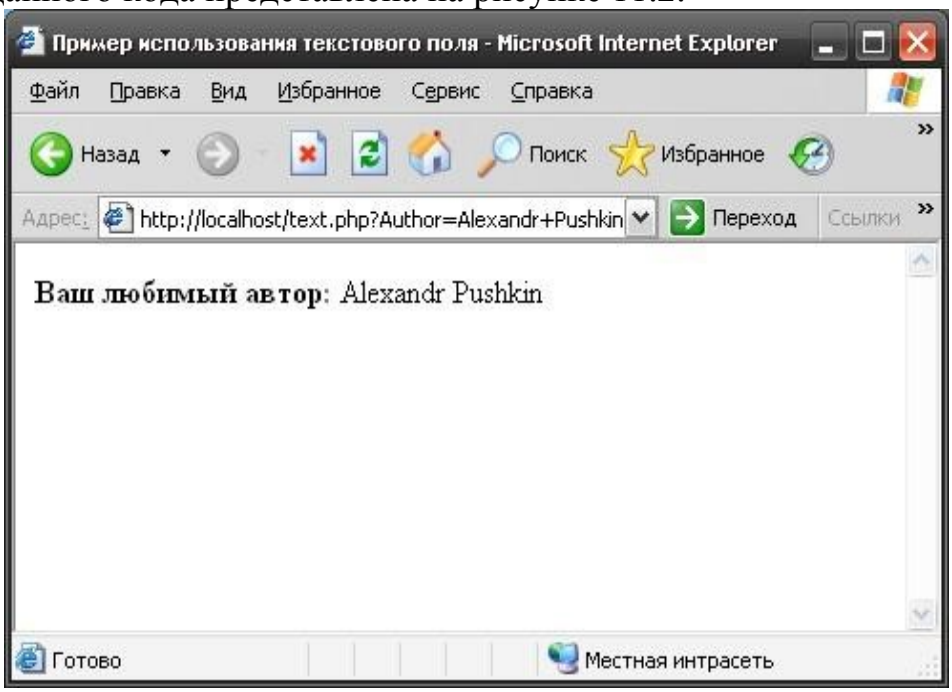


Рисунок 11.2 - Результат обработки введённого автора

Как видно на рисунке 11.2, на экран выводится информация о любимом авторе, то есть “Alexandr Pushkin”. Программа обработки введенных данных text.php, состоит из одной строки PHP-кода: “$_GET ['Author']”, которая отображает содержимое переменной “$Author”. Данная переменная не генерируется в PHP-коде, а автоматически создается, как часть массива “$_GET”, при передаче данных на сервер. При этом “GET” обозначает метод протокола HTTP, с помощью которого передаются данные.

В HTML-файле было создано текстовое поле с именем “Author”. Когда данные из формы передаются Web-серверу, создается массив “$_GET” с элементом “Author”. Также на рисунке 11.2, в строке окна браузера представлен результат запроса, в котором на сервер передается пара “переменная”=“значение”. В данном примере переменной “Author” присваивается значение “Alexandr Pushkin”, то есть на сервер передается пара “Author=Alexandr+Pushkin” и эта строка видна пользователю Webстраницы. Эта строка запроса вверху страницы выглядит следующим образом: “?Author=Alexandr+Pushkin”, так как используется метод GET. Использование метода GET вынуждает браузер отправлять информацию из формы в виде строки запроса, а не скрывать ее в теле HTTP-запроса [4].

Также необходимо отметить, что имена переменных в языке PHP чувствительны к регистру символов. Так переменные “Author” и “author” являются двумя разными переменными. Если, в представленном выше примере вместо переменной “$Author” написать “$author”, то содержимое данной переменной на сервер передаваться не будет и на экране не выведется (рисунок 11.3).

Рисунок 11.3 – Неправильное присваивание имени переменой

2 Текстовая область

Текстовые области предназначены для того, чтобы принимать от пользователя предложения и даже целые строки. Для того чтобы создать текстовую область, следует использовать HTML-тег <TEXTAREA>.

Синтаксис:

<TEXTAREA name="textarea" rows="n" cols="m"> где name - имя поля как элемента формы,

rows - высота поля, cols - ширина поля.

Ниже представлен пример использования текстовой области для ввода информации о любимых Web-сайтах пользователя.

HTML-код формы для ввода информации представлен ниже.

```
<HTML>
<HEAD>
<TITLE>
Пример использования текстовой области
</TITLE>
</HEAD>
<BODY>
<FORM method="POST" action="textarea.php">
ПеречислитеВашилюбимые Web-сайты
<TEXTAREA name="Websites" cols="50" rows="5"> http:// http:// http:// http://
</TEXTAREA>
<BR>
<BR>
<BR>
<INPUT type="submit" value="Отправить">
</FORM>
</BODY>
</HTML>
```

Результат работы данного кода представлен на рисунке 11.4.

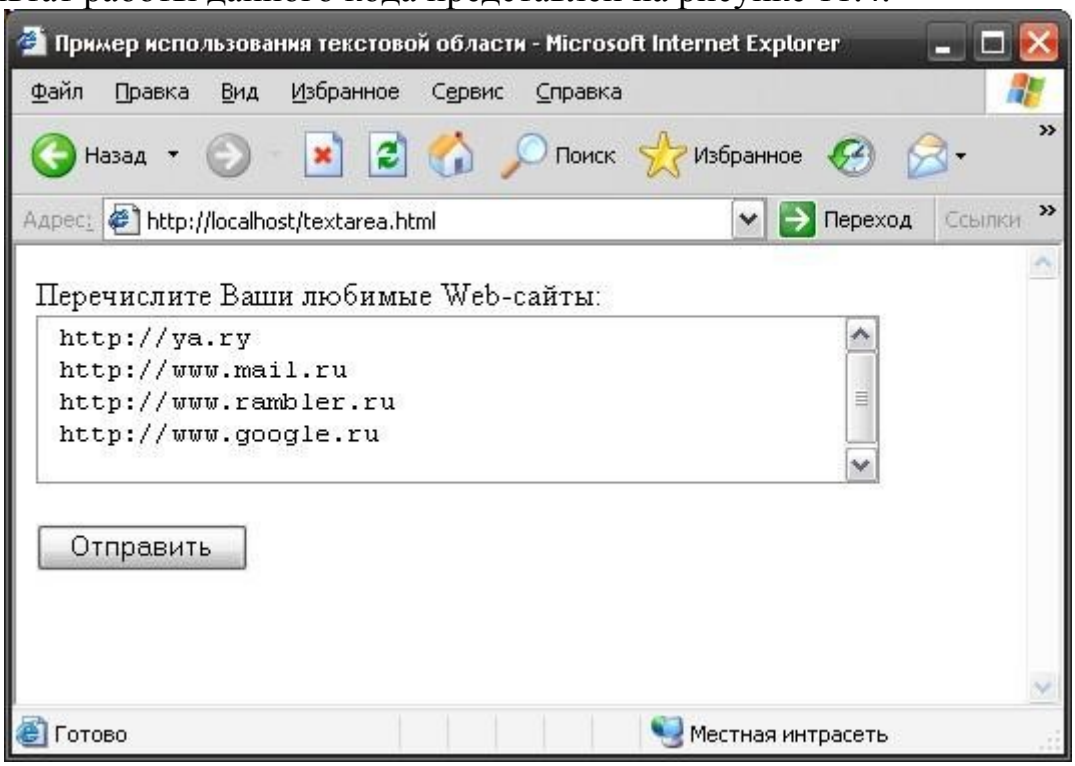


Рисунок 11.4 – Ввод информации в текстовую область

Таким образом, как видно на рисунке 11.4, в данном примере, при загрузке Web-страницы на экране появляется текстовая область, в которую пользователю необходимо ввести информацию и нажать кнопку “Отправить”.

После этого подключится обработчик, указанный в атрибуте “action” тега “form”. В данном примере это файл textarea.php.

Код файла-обработчика представлен ниже.

```
<HTML>
<HEAD>
<TITLE>
Пример использования текстовой области
</TITLE>
</HEAD>
<BODY>
Вашилюбимые Web-сайты:
<?php echo $_POST['Websites']; ?>
</BODY>
</HTML>
```

Результат работы данного кода представлен на рисунке 11.5.

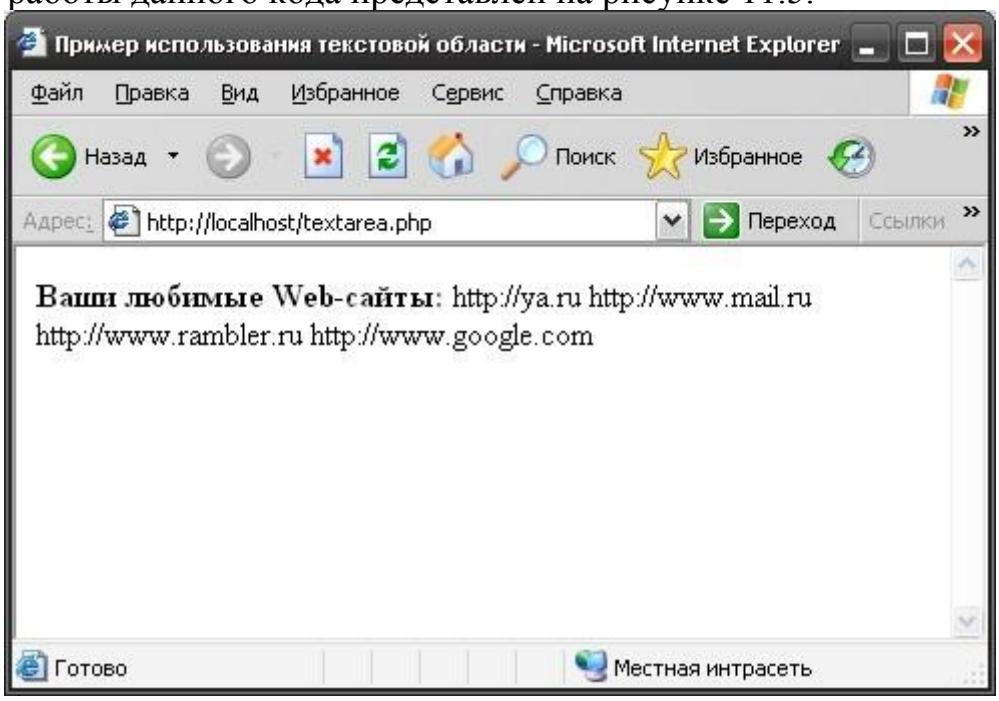


Рисунок 11.5 – Вывод на экран любимых сайтов

Как видно на рисунке 11.5, на сервер передается информация о любимых Web-сайтах пользователя. В данном случае строка запроса не присоединяется, так как в программе был использован метод POST. Использование этого метода необходимо для того, чтобы скрыть информацию из формы при передаче на сервер.

3 Флажки

Флажки являются еще одним элементом управления формами. Они применимы в ситуациях, когда пользователю необходимо ответить на вопрос, требующий строгого однозначного ответа “да” или “нет”, путем установки или снятия “галочки”. Флажки создаются с помощью тега <INPUT>, в котором атрибуту “type” присваивается значение “checkbox”.

Синтаксис:

<INPUT name="checkbox" type="checkbox" checked>

где type - тип поля, name - имя поля как элемента формы, checked - атрибут, который используется для того, чтобы отметить

флажок, как выделенный.

Ниже приведен пример работы с флажками для проверки, является ли пользователь студентом Оренбургского государственного университета.

HTML-код формы для ввода информации представлен ниже.

```
<HTML>
<HEAD>
<TITLE>Пример использования флажка</TITLE>
</HEAD>
<BODY>
<FORM method="POST" action="checkbox.php">
ВыстудентОГУ?
<INPUT name="Choice" type="checkbox">
<BR>
<BR>
<INPUT type="submit" value="Отправить">
</FORM>
</BODY>
</HTML>
```

Реализация данного кода представлена на рисунке 11.6

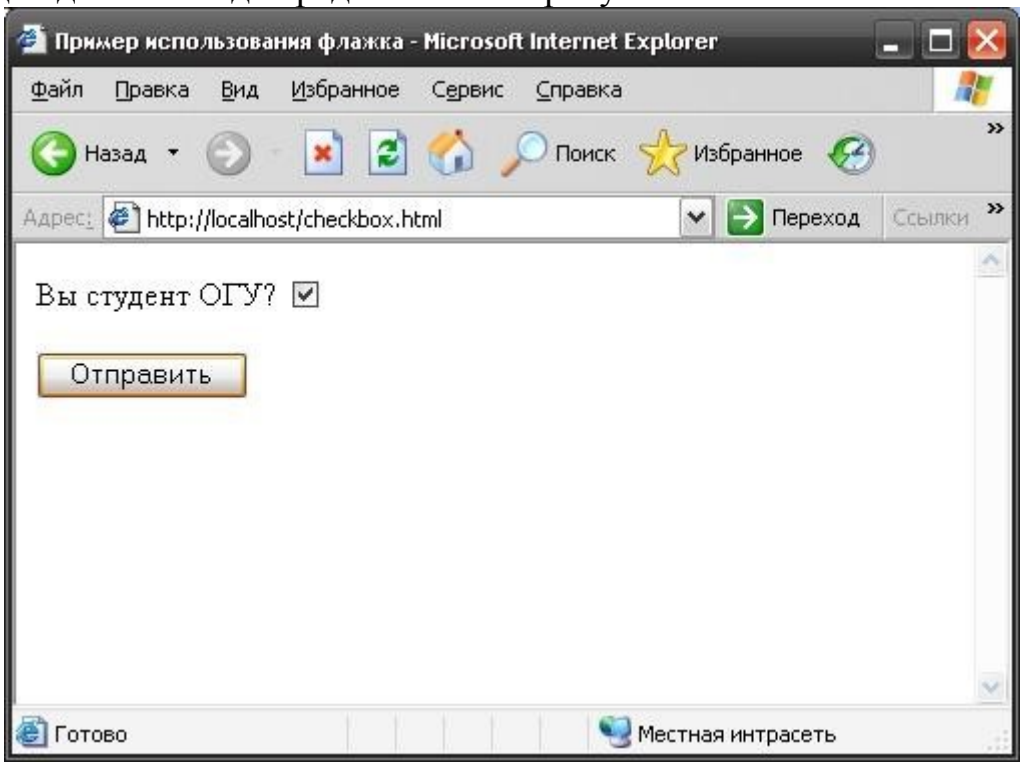


Рисунок 11.6 – Проверка принадлежности к числу студентов ОГУ

Таким образом, как видно на рисунке 11.6, в данном примере, при загрузке Web-страницы на экране появляется флажок, который пользователю необходимо отметить и нажать кнопку “Отправить”.

После этого подключится обработчик, указанный в атрибуте “action” тега “form”. В данном примере это файл checkbox.php.

Код файла-обработчика представлен ниже.

```
<HTML>
<HEAD>
<TITLE> Пример использования флажка</TITLE>
</HEAD>
<BODY>
<?php echo $_POST['Choice'] ;
?>
</BODY>
</HTML>
```

Реализация данного кода представлена на рисунке 11.7.

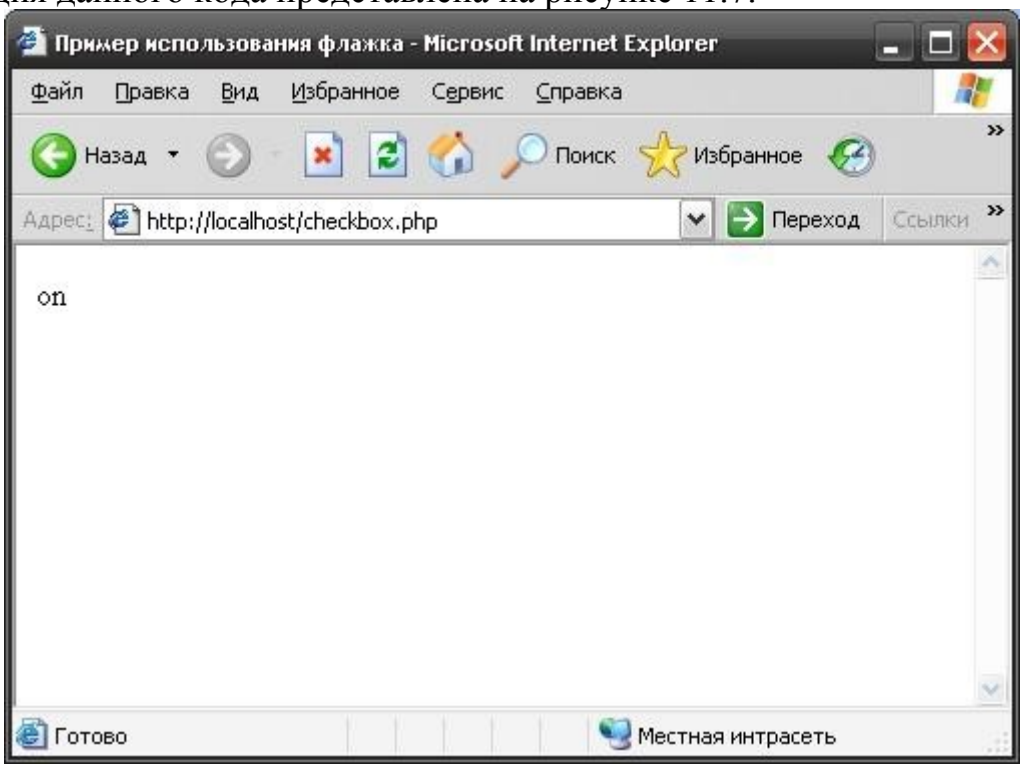


Рисунок 11.7 – Результат проверки

Как видно на рисунке 11.7, из формы на сервер передается не содержимое переменной, а лишь информация о том, был ли выбран данный флажок или нет. Если флажок был установлен пользователем, то значением данной переменной будет “on”. В противном случае, переменная вообще не будет содержать данных. Если же потребуется передать содержимое переменной на сервер, в исходном HTML-файле, в теге <INPUT>, в атрибуте “value” следует указать ее значение [4].

При использовании нескольких флажков, каждый из них является отдельным элементом. Можно выбрать один или несколько флажков, или вообще не выбрать ни одного. При этом если пользователем будут выбраны все флажки, на экран выведется содержимое всех выбранных флажков. Для этого у каждого флажка устанавливается свое значение.

Ниже приведен пример работы с несколькими флажками для выбора языков программирования, которыми владеет пользователь.

HTML-код формы для ввода информации представлен ниже.

```
<HTML>
<HEAD>
<TITLE>Пример использования нескольких флажков</TITLE>
</HEAD>
<BODY>
<P>Выберите язык, на котором Вы программируете:<BR><BR>
<FORM method="POST" action="checkboxes.php">
<INPUT name="Choice1" type="checkbox" value="Delphi">
Вы программируете на Delphi?
<br>
<INPUT name="Choice2" type="checkbox" value="C++/C#">
Вы программируете на C++/C#?
<br>
<INPUT name="Choice3" type="checkbox" value="Assembler">
Вы программируете на Assembler?
<BR>
<BR>
<INPUT type="submit" value="Отправить">
</FORM>
</BODY>
</HTML>
```

Результат работы данного кода представлен на рисунке 11.8.

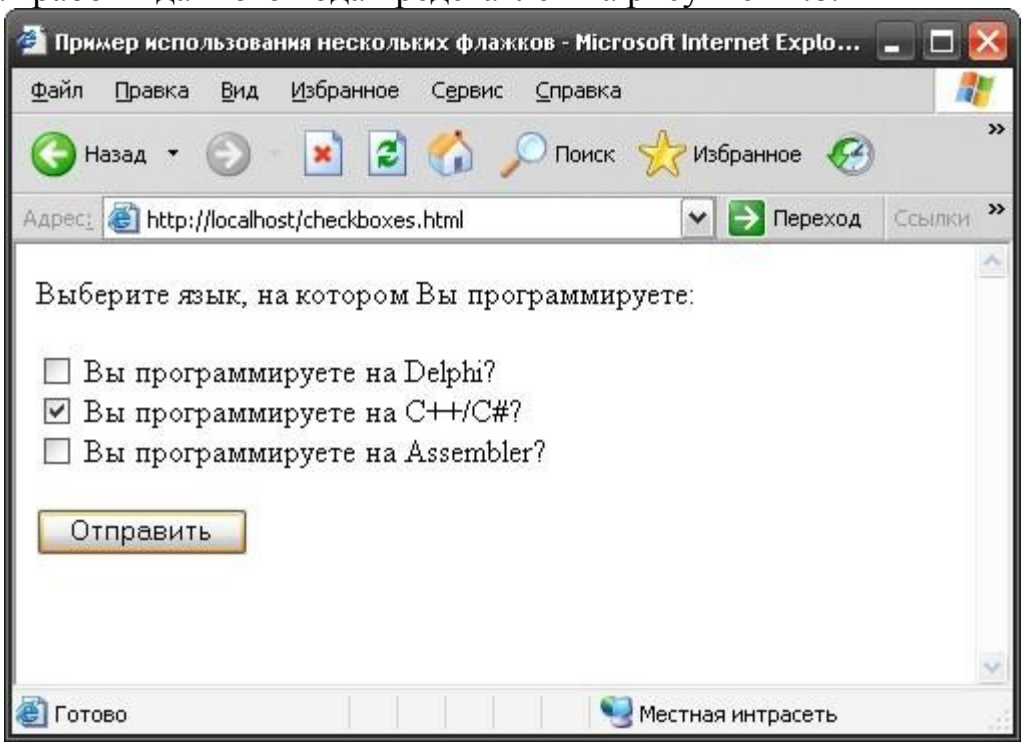


Рисунок 11.8 – Выбор языков программирования

Таким образом, как видно на рисунке 11.8, в данном примере, при загрузке Web-страницы появляются флажки, которые пользователю необходимо отметить и нажать кнопку “Отправить”. После этого подключится обработчик, указанный в атрибуте “action” тега “form”. В данном примере это файл checkboxes.php.

Код файла-обработчика представлен ниже.

```
<HTML>
<HEAD>
<TITLE>
Пример использования нескольких флажков
</TITLE>
</HEAD>
<BODY>
<P>Вывыбралиответ:</P>
<?php echo "$_POST[Choice1]<br>"; echo "$_POST[Choice2]<br>"; echo "$_POST[Choice3]<br>";
?>
</BODY>
</HTML>
```

Результат работы данного примера представлен на рисунке 11.9.

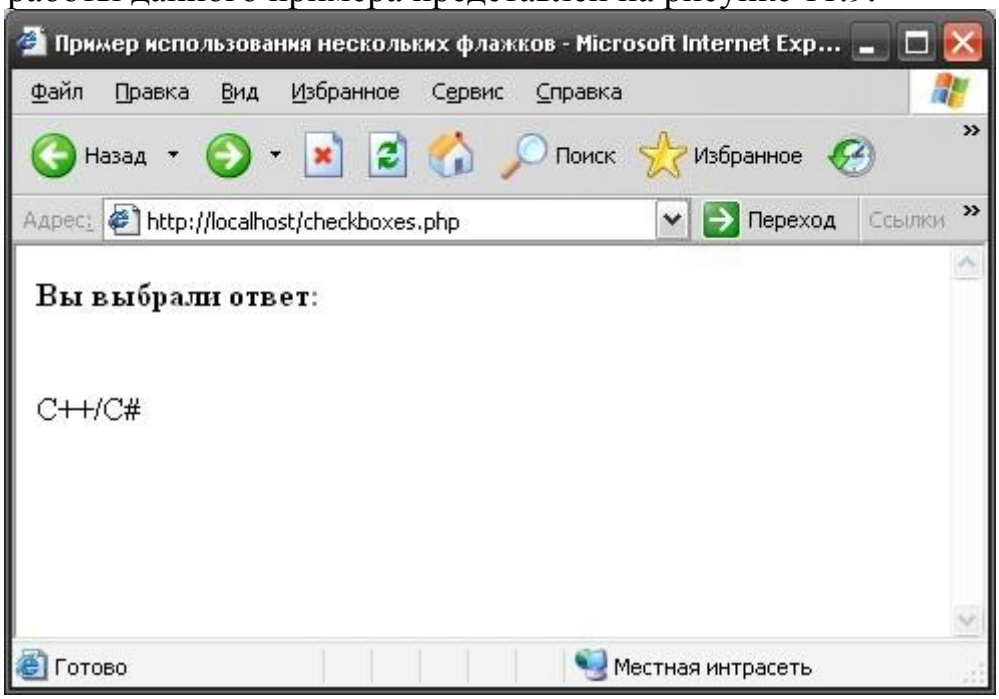


Рисунок 11.9 – Результат выбора языков

Как видно на рисунке 11.9, пользователем был выбран только один флажок и на экран выведено содержимое переменной, равное “C++/C#”. Содержимое остальных переменных выведено не было, так как пользователь не установил другие флажки. Таким образом, в соответствующие PHPпеременные данные передаваться не будут.

4 Переключатели

Переключатели в HTML-формах используются в тех случаях, когда имеется набор возможных ответов, но выбрать можно только один из них. В отличие от флажков, для связи переключателей друг с другом необходимо задать всем переключателям одной группы одинаковое имя.

Переключатели создаются с помощью дескриптора <INPUT>, в котором атрибуту “type” присваивается значение “radio”.

Синтаксис:

<INPUT name="radio" type="radio" checked> где type - тип поля, name - имя поля как элемента формы, checked - атрибут, который используется для того, чтобы отметить

переключатель, как выделенный.

Ниже приведен пример использования переключателей для выбора названия столицы России.

HTML-код формы для ввода информации представлен ниже.

```
<HTML>
<HEAD><TITLE>Пример использования переключателей</TITLE>
</HEAD>
<BODY>
<FORM method="GET" action="radio.php">
Выберите название столицы России
<BR>
<BR>
<INPUT name="Question" type="radio" value="Москва">
Москва
<br>
<INPUT name="Question" type="radio" value="Екатеринбург">
Екатеринбург
<br>
<INPUT name="Question" type="radio" value="Оренбург">
Оренбург
<BR>
<BR>
<INPUT type="submit" value="Отправить">
</FORM>
</BODY>
</HTML>
```

Результат работы данного кода представлен на рисунке 11.10.

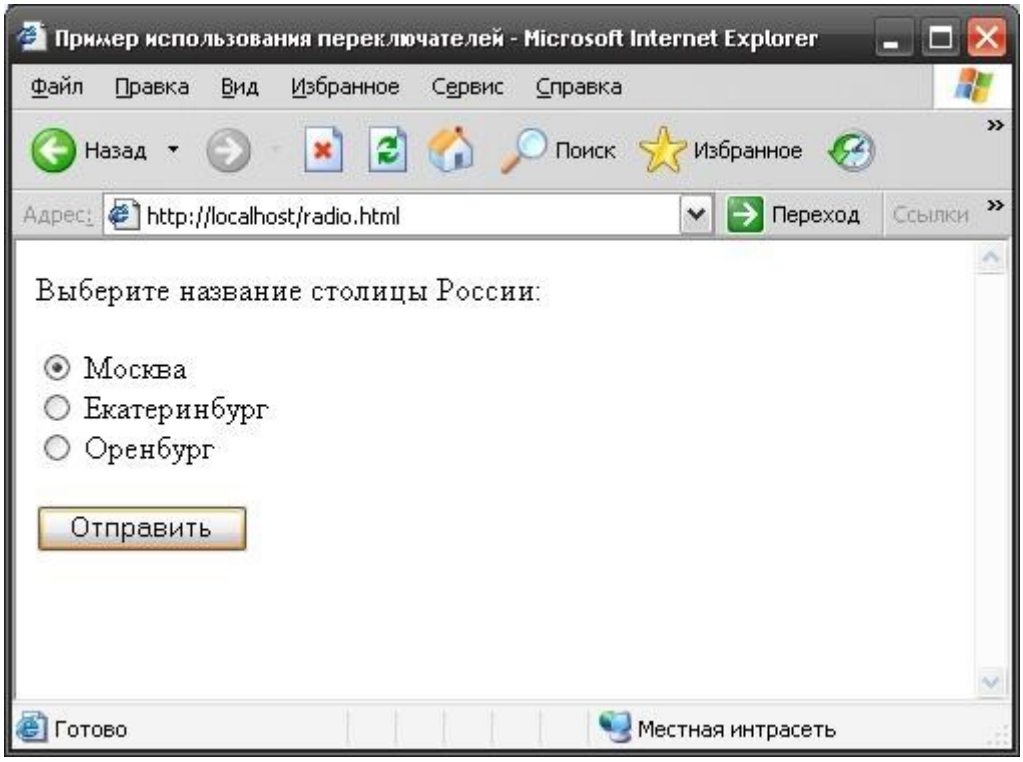


Рисунок 11.10 – Выбор столицы России

Таким образом, как видно на рисунке 11.10, в данном примере, при загрузке Web-страницы на экране появляются переключатели, из которых пользователю необходимо отметить только один и нажать кнопку “Отправить”. После этого подключится обработчик, указанный в атрибуте “action” тега “form”. В данном примере это файл radio.php.

Код файла-обработчика представлен ниже. <HTML>

<HEAD><TITLE> Пример использования переключателей </TITLE>

</HEAD>

<BODY>

<?php echo "Вывыбралиответ: $_GET[Question]"; ?>

</BODY>

</HTML>

Реализация данного кода представлена на рисунке 11.11.

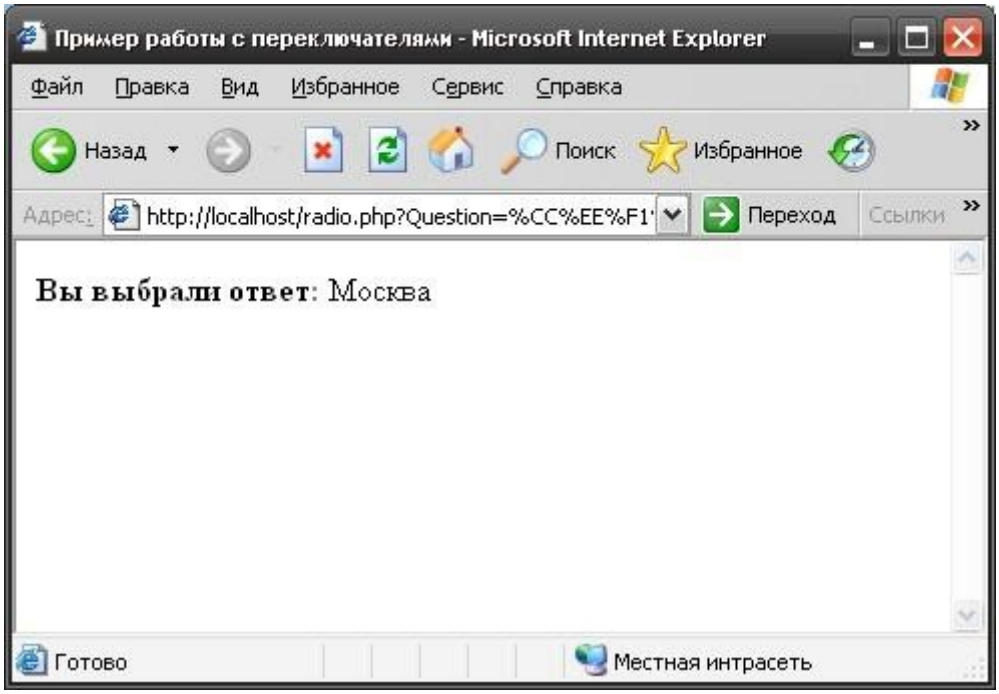


Рисунок 11.11 – Результат выбора столицы

Как видно на рисунке 11.11, на экран выводится значение переменной “$Question”, равное “Москва”. Необходимо отметить, что в коде файла radio.html создается три переключателя, имеющих одинаковое имя, но разные значения. Однако при обработке данных с помощью сценария отображается содержимое только одной переменной, так как может быть только один правильный ответ. В этом заключается основное отличие использования переключателей от флажков [4].

5 Списки

Списки или выпадающие списки представляют собой элементы управления формами, которые обычно отображают несколько объектов. Иногда они имеют стрелку, которая позволяет пользователю перемещаться вниз к дополнительным объектам. В языке HTML списки создаются с помощью тегов <SELECT> и <OPTION>. Тег <SELECT>, создающий список, охватывает несколько дескрипторов <OPTION>, каждый из которых содержит текст, соответствующий объекту в списке.

Синтаксис:

```
<SELECT name="select" multiple size="m">
<OPTION value="value 1">
<OPTION value="value 2">
...
<OPTION value="value n">
</SELECT>
```

где name - имя поля как элемента формы, multiple - множественный выбор элементов списка, size - размер видимой части списка, value - значение переменной.

Ниже представлен пример использования списков, в котором программа получает от пользователя информацию о желаемой бытовой технике и фирме-производителе. Первый список допускает выбор только одного типа бытовой техники. Во втором допускается множественный выбор фирмы-производителя.

HTML-код формы для выбора информации представлен ниже.

```
<HTML>
<HEAD>
<TITLE>Пример использования списков</TITLE>
</HEAD>
<BODY>
<FORM method="POST" action="listbox.php">
Выберите бытовую технику
<BR>
<BR>
<SELECT name="Tech">
<OPTIONvalue="Стиральнаямашина">Стиральнаямашина</OPTION>
<OPTION value=" Утюг">Утюг</OPTION>
<OPTION value="Микроволновая печь">Микроволновая печь </OPTION>
<OPTION value="Пылесос">Пылесос</OPTION>
</SELECT>
<BR>
<BR>
Выберите фирму-производителя
<BR>
<BR>
<SELECT name="Production[]" multiple size=4>
<OPTION value="LG">LG</OPTION>
<OPTION value=" Samsung">Samsung</OPTION>
<OPTION value=" Panasonic">Panasonic</OPTION>
<OPTION value="Sony">Sony</OPTION>
</SELECT>
<BR>
<BR>
<INPUT TYPE="submit" value="Отправить">
</FORM>
</BODY>
</HTML>
```

Результат работы данного кода представлен на рисунке 11.12.

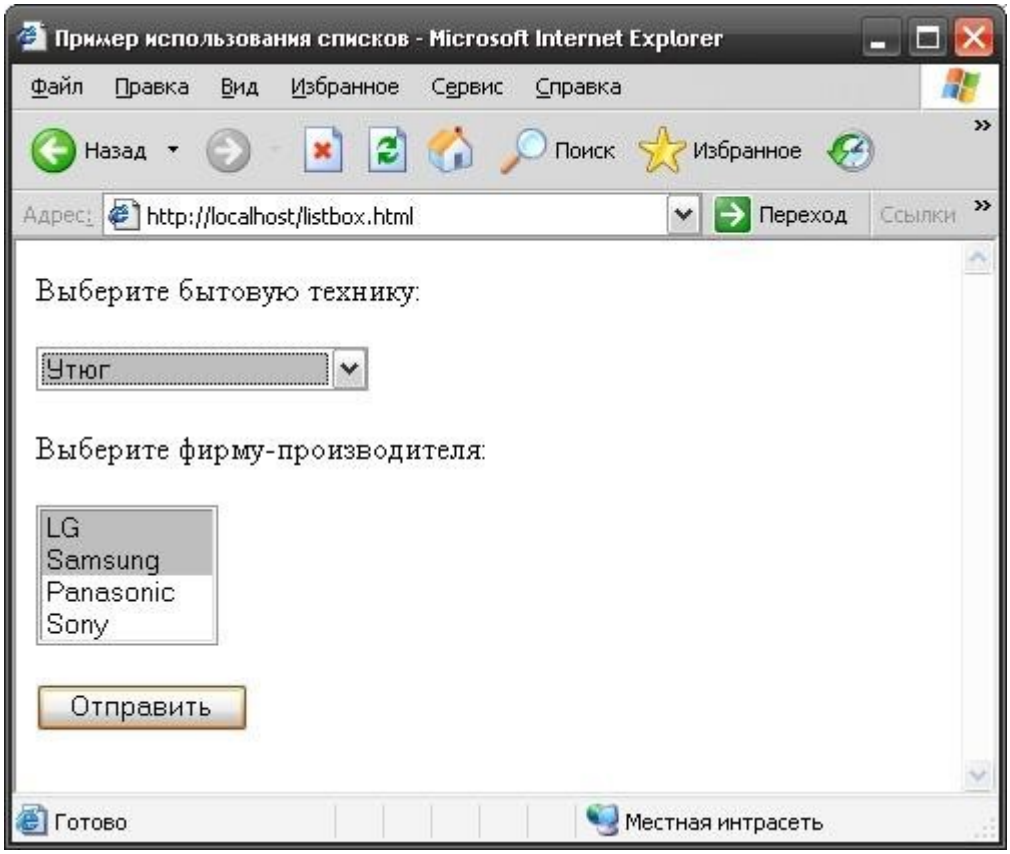


Рисунок 11.12 – Выбор техники и фирмы

Таким образом, как видно на рисунке 11.12, в данном примере, при запуске Web-страницы на экране появляются два списка, в которых необходимо выбрать информацию и нажать кнопку “Отправить”. После этого подключится обработчик, указанный в атрибуте “action” тега “form”. В данном примере это файл listbox.php.

Код файла-обработчика представлен ниже.

```
<HTML>
<HEAD>
<TITLE>Пример использования списков</TITLE>
</HEAD>
<BODY>
<?php echo "Бытоваятехника: $_POST[Tech] "; $Choice0 = $_POST['Production'][0]; $Choice1 = $_POST['Production'][1];
$Choice2 = $_POST['Production'][2]; $Choice3 = $_POST['Production'][3]; echo "<br>Фирма производитель: "; echo "$Choice0 "; echo "$Choice1 "; echo "$Choice2 "; echo "$Choice3 ";
?>
</BODY>
</HTML>
```

Результат работы данного примера представлен на рисунке 11.13.

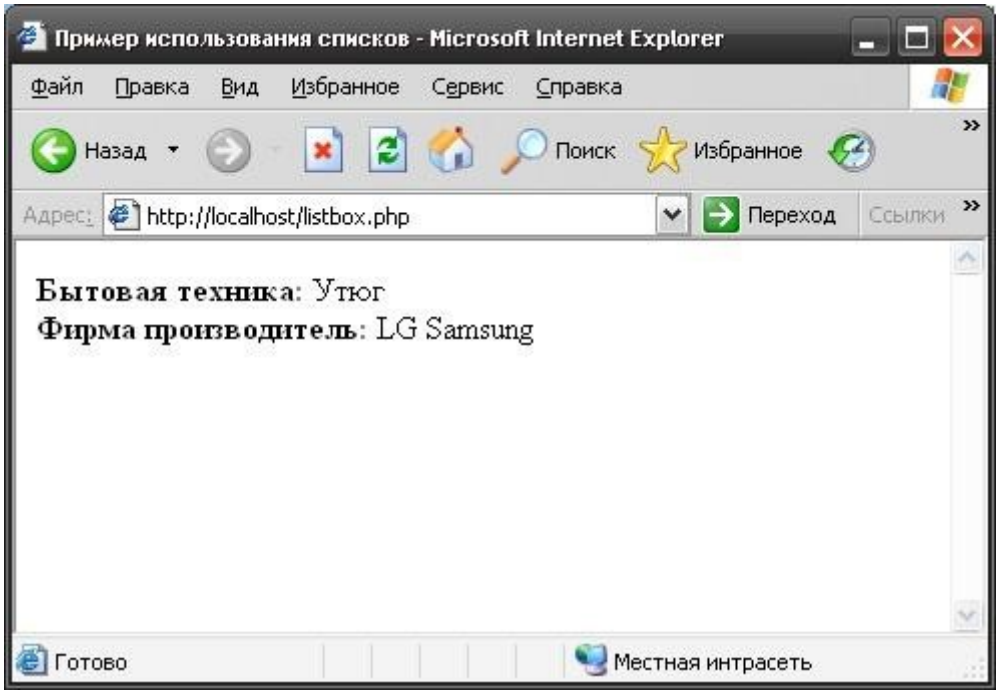


Рисунок 11.13 – Результат выбора техники и фирмы

Как видно на рисунке 11.13, на экран выводятся результаты выбора пользователя. В данном случае он выбрал технику утюг и фирму производителя “LG” и “Samsung”. То есть, при выборе бытовой техники перемененной “$Tech” присвоилось значение “Утюг”. В сценарии listbox.php атрибут “name” связан с PHP-переменной “$_POST ['Tech']”.

При выборе фирмы-производителя атрибуту “name” было присвоено значение “$Production[]”, где квадратные скобки интерпретируют соответствующую переменную как массив. Список содержит четыре пункта, поэтому в массиве будет четыре элемента. Содержимое каждого из них необходимо отобразить.

При работе с массивами в языке PHP индекс массива ссылается на элемент, а нумерация элементов начинается с нуля. Тогда переменная “$Production[0]” будет ссылаться на первый пункт в списке, то есть LG, а переменная “$Production[1]” на второй пункт в списке, то есть Samsung. Элементы “$Production[2]” и “$Production[3]” не содержат ничего, так как пользователь выбрал только два первых пункта списка [4].

Задание

Изучить теоретическое обоснование и приведенные в нем примеры.

Содержание отчета и его форма

Отчет по лабораторной работе должен состоять из:

1) названия лабораторной работы;
2) описание совместного использования PHP и HTML-форм.

Вопросы для защиты работы:

1. Как совместно использовать PHP и текстовые поля HTML-форм?
2. Как совместно использовать PHP и флажки HTML-форм?

3. Как совместно использовать PHP и переключатели HTML-форм?

4.Как совместно использовать PHP и списки HTML-форм?

Лабораторная работа №12. PHP и поля HTML-форм: скрытые поля форм, поля ввода паролей, кнопки, значения, возвращаемые формами, проверка обязательных полей

Цель работы: научиться использовать элементы HTML-форм совместно с языком PHP.

Теоретическое обоснование

1 Скрытые поля форм

При работе с формами может возникать необходимость получения информации, содержащейся на одной Web-странице и передачи ее другой Web-странице, без какого-либо ввода данных пользователем. В теге <INPUT> для этого существует еще один параметр, который позволяет передавать информацию в поле, как если бы оно было текстовым, скрывая, при этом, сам элемент управления и его значение от пользователя. Такое поле называется скрытым полем формы (или скрытым элементом управления) [4].

Действие скрытых полей несколько отличается от действия уже рассмотренных элементов управления. Их можно использовать для отправки информации, хранящейся в PHP-переменных.

Чтобы использовать скрытое поле в PHP-странице, можно написать всю HTML-форму в операторах echo, передающих содержимое PHPпеременных через HTML-теги [4].

Синтаксис:

<INPUT type="hidden" name="hidden" value="message"> где type - тип поля, name - имя поля как элемента формы, value - содержимое поля.

Ниже представлен пример использования скрытых полей форм для чтения содержимого списка и отображения пользовательского выбора на следующей странице.

HTML-код формы для выбора информации представлен ниже.

```
<HTML>
<HEAD><TITLE> Использование скрытых полей ф о р м </TITLE></HEAD>
<BODY>
<?php
$Message1="Иванов";
$Message2="Петров"; $Message3="Сидоров"; echo "<FORM method='GET' action='hidden.php'>"; echo "Кто из следующих персонажей получит приз?"; echo "<SELECT name='ListBox'>"; echo "<OPTION>$Messagel</OPTION>"; echo "<OPTION>$Message2</OPTION>"; echo "<OPTION>$Message3</OPTION>"; echo "</SELECT><br><br>"; echo
```

"<INPUT type='hidden' name='Hidden1' value='$Message1'> "; echo "<input type='hidden' name='Hidden2' value='$Message2'>"; echo "<INPUT type='hidden' name='Hidden3' value='$Message3'>"; echo "<INPUT type='submit' value='Отправить'>"; echo "</FORM>";

?>

</BODY>

</HTML>

Результат работы данного кода представлен на рисунке 12.1.

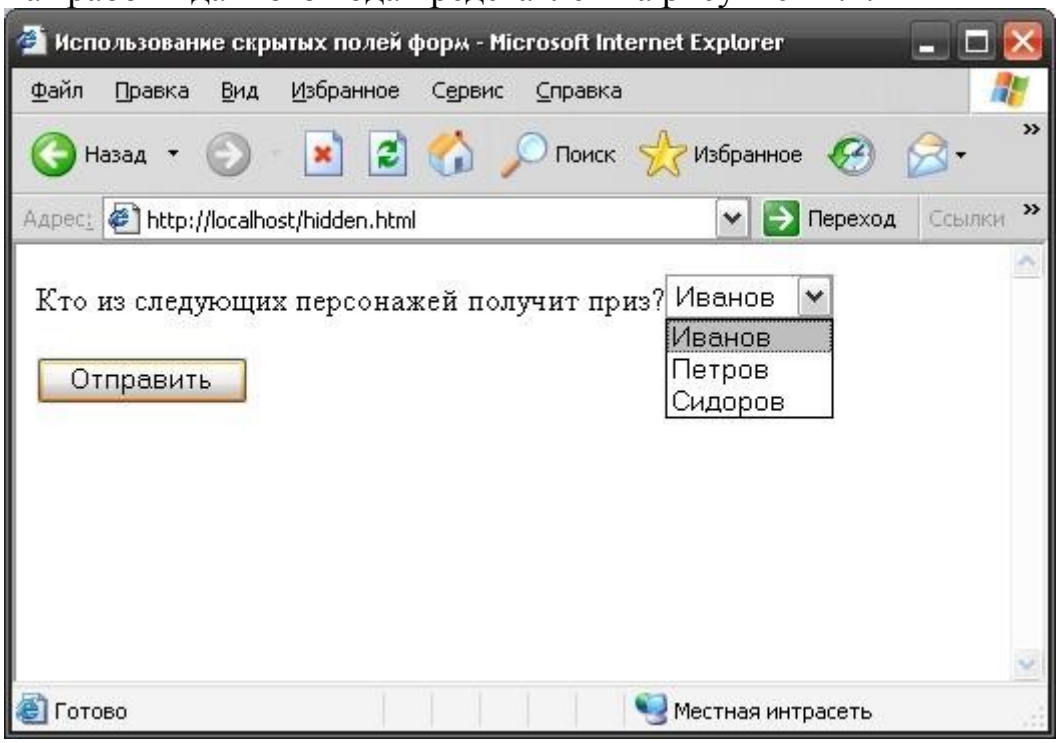


Рисунок 12.1 – Выбор персонажа

Таким образом, как видно на рисунке 12.1, в данном примере, при загрузке Web-страницы на экране появляется список, в котором пользователю необходимо выбрать информацию. При этом форма, отправляющая скрытые поля, содержит элементы данного списка в переменных “$Hidden1”, “$Hidden2”, “$Hidden3”, содержащих текст “Иванов”, “Петров”, “Сидоров” соответственно. После нажатия кнопки “Отправить”, подключается обработчик, указанный в атрибуте “action” тега “form”. В данном примере это файл hidden.php.

Код файла-обработчика представлен ниже.

<HTML>

<HEAD>

<TITLE>Использование скрытых полей форм</TITLE>

</HEAD>

<BODY>

<?php

echo "Было три варианта:
";

echo "$_GET[Hidden1]
"; echo "$_GET[Hidden2]
"; echo "$_GET[Hidden3]
"; echo "
Вывыбрали:
"; echo "$_GET[ListBox]";

?>

</BODY>

</HTML>

Результат работы данного примера представлен на рисунке 12.2.

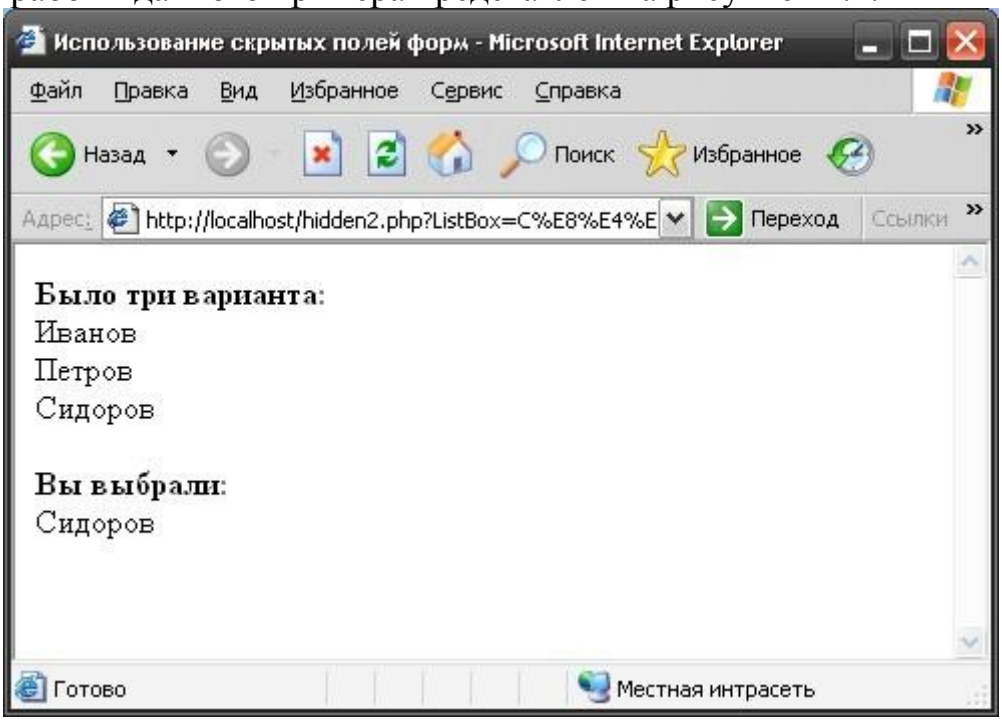


Рисунок 12.2 – Результат выбора персонажа

Как видно на рисунке 12.2, на экран выводятся исходные три варианта для выбора (“Иванов”, “Петров”, “Сидоров”), а затем выводится выбор пользователя. В данном случае это “Сидоров”.

Главная особенность использования этого способа заключается в том, что можно опустить апострофы в именах переменных массива. Для формирования списка создается три переменных. Затем с помощью echoоператоров создается форма. Она ничем не отличается от обычной HTMLформы, за исключением того, что там, где нужны кавычки, следует писать не двойные, а одинарные кавычки, иначе echo-оператор будет работать неправильно.

Вторая PHP-страница отображает содержимое элементов управления, созданных на первой странице. Сначала выводится содержимое трех скрытых полей формы. Это полезно, потому что обычно содержимое всего списка не передается. Последние строки отображают выбранный пользователем пункт.

2 Поля ввода паролей

Поля ввода паролей, по сути, представляют собой текстовые поля, в которых вводимые символы заменяются звездочками. Они хранят и передают информацию так же, как и текстовые поля.

Синтаксис:

Введитепароль:

<INPUT name="password" type="password"> где type - тип поля, name - имя поля как элемента формы.

Обработка полей ввода паролей и текстовых полей ничем не отличается. Следует, однако, отметить, что если для передачи данных из поля такого типа использовать метод GET, то пароль в строке запроса не шифруется и будет видимым. Это не означает, что метод POST является безопасным методом передачи данных. Речь идет только о том, что информация при передаче методом POST непосредственно не показывается пользователю. Чтобы действительно обеспечить безопасность передаваемых данных, необходимо применять какой-либо специальный протокол для реального шифрования данных.

3 Кнопки submit и reset

Кнопка типа Submit используется для передачи данных пользователя на сервер.

Синтаксис:

<INPUT type="submit" name="submit" value="отправить"> где type - тип поля, name - имя поля как элемента формы, value - надпись на кнопке.

Кнопка инициирует отправку данных пользователя из формы на сервер. При нажатии на кнопку “submit”, данные будут передаваться наобработчик, указанный в атрибуте “action” тега “form”. При этом в массиве “$_GET” или “$_POST” будет создана переменная “submit”.

Кнопка типа Reset устанавливает все элементы управления на форме в их первоначальное состояние.

Синтаксис:

<INPUT type="reset" name="reset" value="сброс"> где type - тип поля, name - имя поля как элемента формы, value - надпись на кнопке.

Выбор кнопки “reset ” вернет значение полей в исходное состояние. На практике элементы Submit и Reset используются вместе. При создании сложных форм всегда должна присутствовать кнопка, позволяющая отправить данные из полей формы, и всегда должна быть кнопка очистки содержимого полей.

4 Использование значений, возвращаемых формами, в PHP-сценариях

Выше были рассмотрены все разновидности элементов управления формами и примеры передачи на сервер их содержимого средствами языка PHP. Однако, приводимые до сих пор примеры лишь выводят содержимое элементов управления на другой Web-странице, при этом, не проверяя вводимую информацию на корректность.

Далее представлен пример совместного использования элементов управления формами и основных операторов языка PHP, для создания и обработки формы заявки на получение кредита в банке. Пример состоит из

двух Web-страниц. На первой странице генерируется форма для ввода информации пользователя по получению кредита. Вторая страница принимает переданные значения и выполняет некоторые простые операции над ними, чтобы принять или отклонить заявку на кредит [4].

В данном примере у пользователя запрашивается размер денежной суммы, которую он желает взять в кредит, возраст и размер текущей заработной платы. После этого приложение вычисляет сумму, которую банк может предложить заявителю, на основании данных о возрасте и заработной плате. В конце расчетов выдается либо положительный ответ, в случае, если банк может выдать кредит, либо отрицательный, в обратном случае.

Формула расчета кредита вычисляется по трем переменным:

переменная нормы зарплаты: годовая зарплата пользователя, разделенная на 5; переменная возрастного ценза: возраст пользователя, разделенный на 10, результат округляется в меньшую сторону до ближайшего целого числа, а затем уменьшается на единицу; переменная кредитной нормы: норма зарплаты, умноженная на возрастной ценз [4].

HTML-код формы для ввода информации представлен ниже.

```
<HTML>
<HEAD><TITLE>Заявка на получение кредита</TITLE></HEAD>
<BODY>
<B>Заявка на получение кредита в Alphabank.</B>
<FORM method="POST" action="loan.php"> Имя:
<INPUT name="FirstName" type="text">
Фамилия:
<INPUT name="LastName" type="text">
Возраст:
<INPUT name="Age" type="text" size="3">
<BR>
<BR>
АДРЕС:
<TEXTAREA name="Address" rows="4" cols="40">
</TEXTAREA>
<BR>
<BR>
Укажите размер Вашей текущей заработной платы:
<SELECT NAME="Salary">
<OPTION VALUE=0>ДО $10000</OPTION>
<OPTION VALUE=10000>$10000 - $25000</OPTION>
<OPTION VALUE=25000>$25000 - $50000</OPTION>
<OPTION VALUE=50000>Свыше $50000</OPTION>
</SELECT>
<BR><BR>
Выберите сумму необходимого кредита:<BR><BR>
```

```
<INPUT name="Loan" type="radio" value="1000">$1000 под 8,0% годовых
<br>
<INPUT name="Loan" type="radio" value="5000">$5000 под 11,5% годовых
<br>
<INPUT name="Loan" type="radio" value="10000">$10000 под 15,0% годовых
<BR>
<BR>
<INPUT type="submit" value="Податьзаявку">
<INPUT type="reset" value="Очистить">
</FORM>
</BODY>
</HTML>
```

Реализация данного кода представлена на рисунке 12.3.

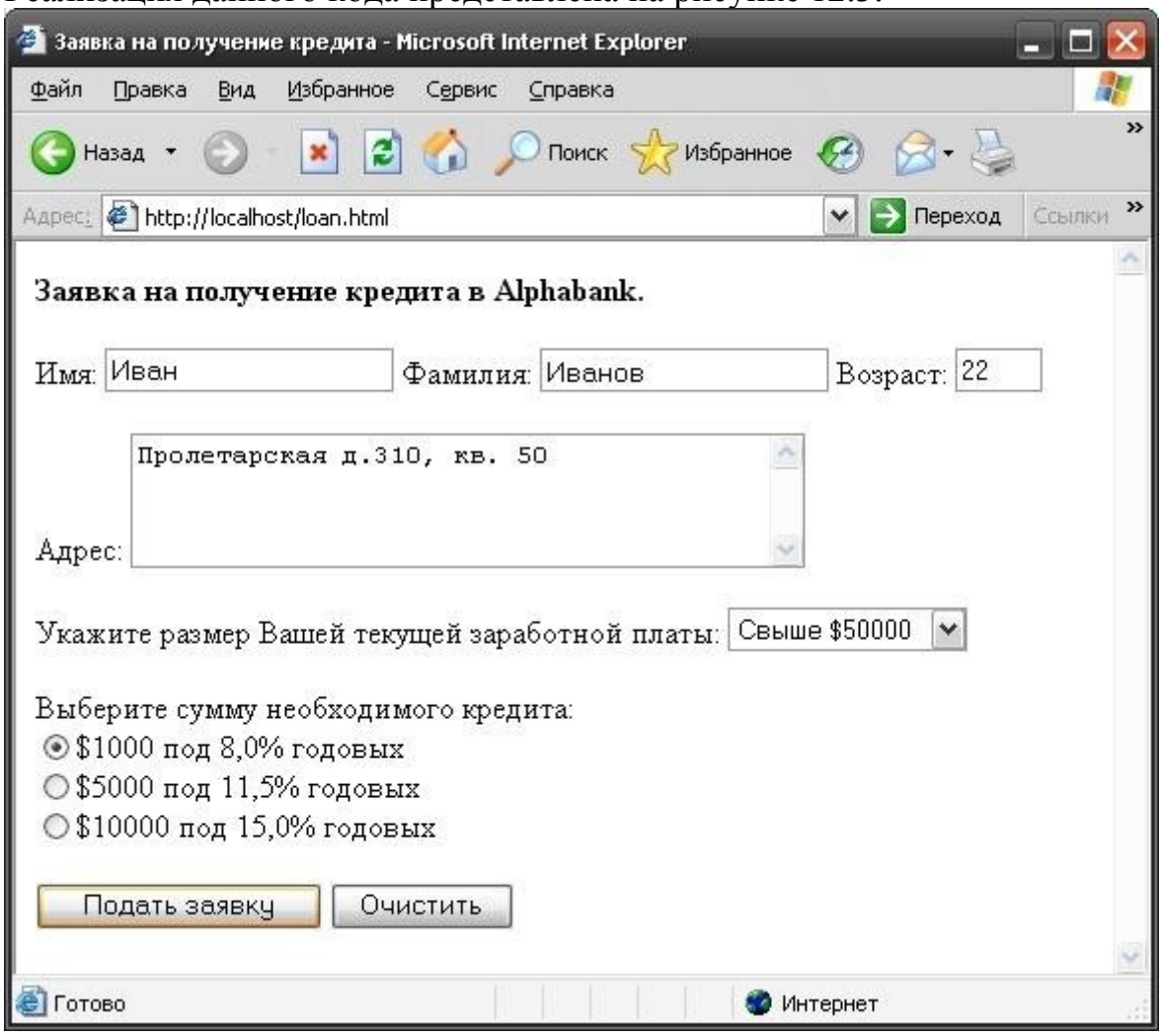


Рисунок 12.3 – Форма заявки на кредит

Таким образом, как видно на рисунке 12.3, в данном примере, при загрузке Web-страницы на экране появляется форма для ввода информации, в которую пользователю необходимо ввести свои данные и нажать кнопку

“Подать заявку”, чтобы отправить данные, либо кнопку “Очистить”, для сброса данных. После этого подключится обработчик, указанный в атрибуте “action” тега “form”. В данном примере это файл loan.php.

Код файла-обработчика представлен ниже. <HTML>

<HEAD><TITLE>Заявка на получение кредита </TITLE></HEAD>

<BODY>

<B> Заявка на получение кредита в Alphabank </B>

<?php

$SalaryAllowance = $_POST['Salary']/5;

$AgeAllowance = ($_POST['Age']/10 - ($_POST['Age']%10)/10)-1; $LoanAllowance = $SalaryAllowance * $AgeAllowance; echo "Запрашиваемыйкредит: $_POST[Loan]
"; echo "Допустимаясуммакредита: $LoanAllowance

"; if ($_POST['Loan'] <= $LoanAllowance) echo "Да, $_POST[FirstName]

$_POST[LastName] , мы удовлетворим Вашу заявку"; if ($_POST['Loan'] > $LoanAllowance) echo "Извините,

$_POST[FirstName] $_POST[LastName], в настоящее время мы не можем принять Вашу заявку";

?>

</BODY>

</HTML>

Реализация данного кода представлена на рисунке 12.4.

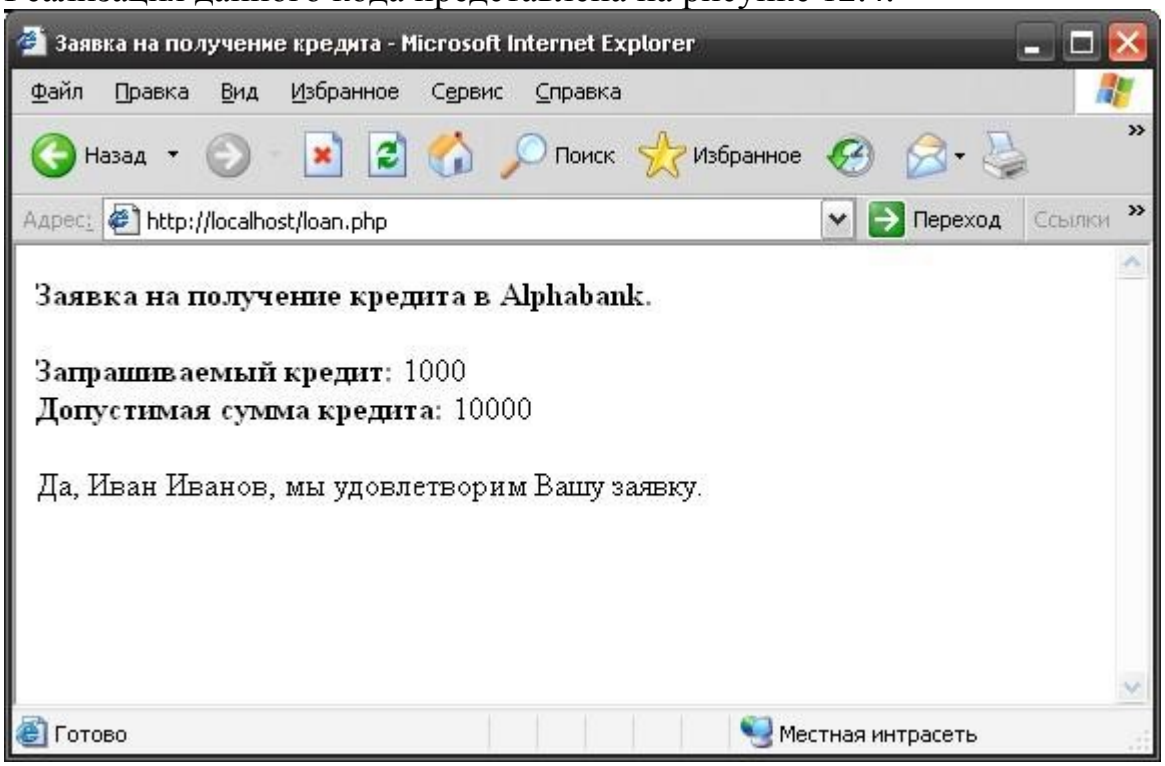


Рисунок 12.4 – Результат подачи заявки

Как видно на рисунке 12.4, на экран выводится информация о заказе пользователя Иванова Ивана, который запрашивает сумму кредита $1000. В программе вычисляется сумма допустимого кредита, в данном случае $10000. При проверке оказалось, что допустимая сумма кредита больше запрашиваемой суммы, то есть 10000>1000. В таком случае, приложение выдает положительный ответ, то есть банк может выдать кредит данному пользователю, что и выводится на экран.

При этом при передаче информации на сервер был использован метод POST, так как этот метод скрывает конфиденциальную информацию от пользователей.

5 Проверка обязательных полей

При обработке данных из форм, пользователи Web-сайта могут вводить в HTML-формы любые данные или не вводить их вовсе, независимо от того, как хорошо сделана форма и что сценарий ожидает получить от пользователя. Главная защита от таких проблем это проверка информации из форм, как на клиентской, так и на серверной стороне. В языке PHP, работающем на стороне сервера, для этого используется стандартный условный оператор [4].

При обработке данных пользователя из форм, при появлении ошибок бывает необходимо прервать обработку данных. Для этого можно использовать любой метод для окончания обработки, но существует более жесткий оператор языка PHP, который останавливает всю работу программы. Это оператор exit. После того, как интерпретатор языка PHP встречает оператор exit, ни HTML-код, ни PHP-код больше не обрабатываются и выполнение скрипта прерывается.

Ниже приведен пример обработки заявок на получение кредита, выполненный более устойчивым по отношению к пользовательским ошибкам, за счет внедрения проверки HTML-формы.

HTML-код формы для ввода информации представлен ниже.

```
<HTML>
<HEAD><TITLE>Заявка на получение кредита</TITLE></HEAD>
<BODY>
<B>Заявка на получение кредита в Alphabank.</B>
<FORM method="POST" action="loan.php">
<INPUT type="hidden" name="posted" value="true"><BR>
Имя: <INPUT name="FirstName" type="text">
Фамилия: <INPUT name="LastName" type="text"> Возраст: <INPUT name="Age" type="text" size="3">
<BR><BR>Адрес:
<TEXTAREA name="Address" rows="4" cols="40"> </TEXTAREA>
<BR><BR> Укажите размер Вашей текущей заработной платы:
<SELECT name="Salary">
<option value=0>До $10000</option>
```

```
<option value=10000>От $10000 до $25000</option>
<option value=25000>От $25000 до $50000</option>
<option value=50000>Свыше $50000</option> </SELECT>
<BR><BR>Выберите сумму необходимого кредита: <BR>
<INPUT name="Loan" type="radio" value="1000">$1000 под 8,0% годовых<BR>
<INPUT name="Loan" type="radio" value="5000">$5000 под 11,5% годовых<BR>
<INPUT name="Loan" type="radio" value="10000">$10000 под 15,0% годовых<BR><BR>
<INPUT type="submit" value="Податьзаявку">
<INPUT type="reset" value="Очистить">
</FORM>
</BODY>
</HTML>
```

Реализация данного кода представлена на рисунке 12.5.

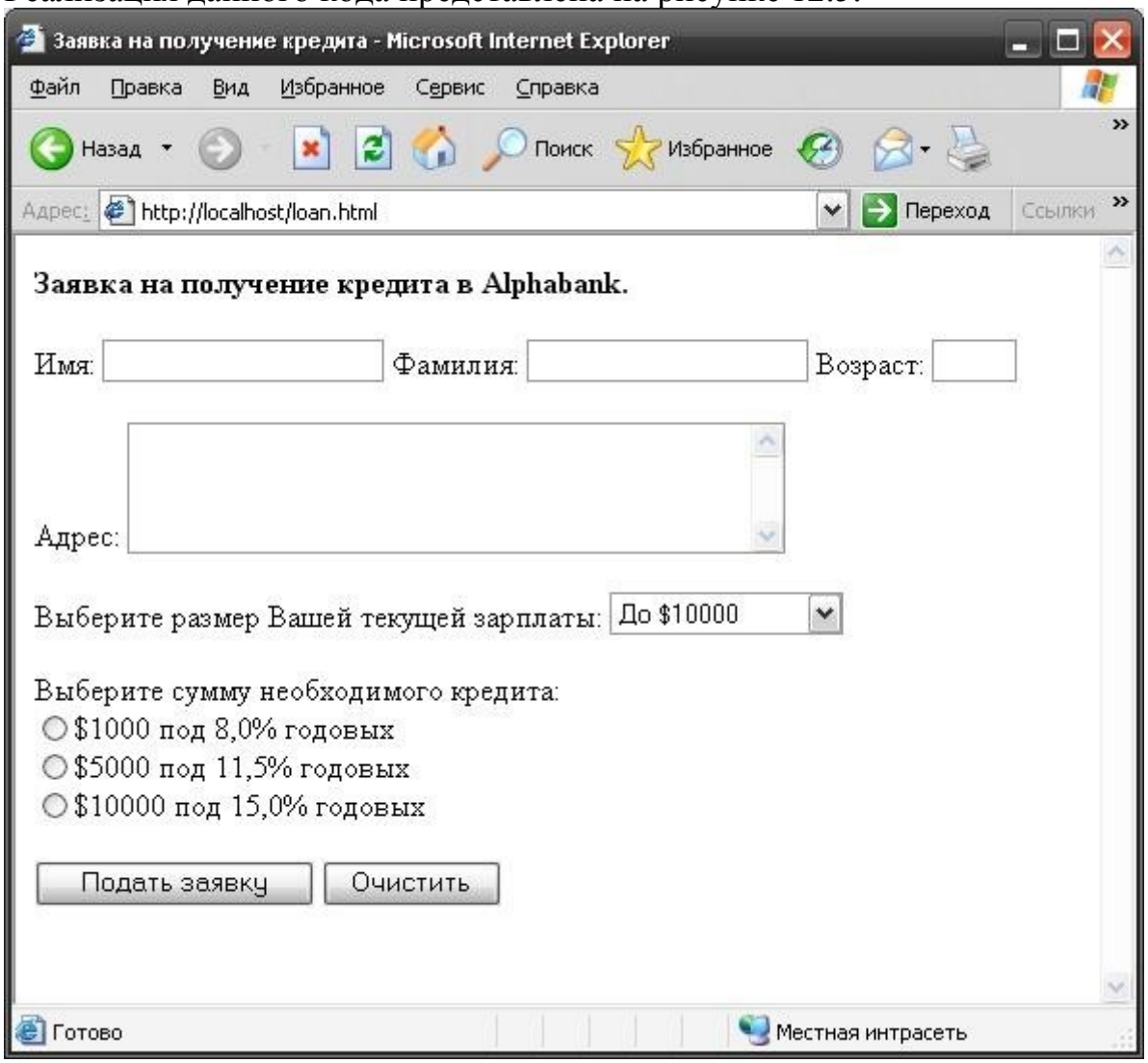


Рисунок 12.5 – Форма заявки на получение кредита

Таким образом, как видно на рисунке 12.5, в данном примере, при загрузке Web-страницы на экране появляется форма для ввода информации, в которую пользователю необходимо ввести свои данные. Затем нужно нажать кнопку “Подать заявку”, чтобы отправить данные, либо “Очистить” для сброса данных. После этого подключится обработчик, указанный в атрибуте “action” тега “form”. В данном примере это файл loan.php.

Код файла-обработчика представлен ниже.

```
<HTML>
<HEAD>
<TITLE>Заявка на получение кредита</TITLE>
</HEAD>
<BODY>
<B>Заявка на получение кредита в Alphabank.</B>
<?php if (isset($_POST['posted']))
{ $age = $_POST['Age'];
$first_name = $_POST['FirstName'];
$last_name = $_POST['LastName'];
$address = $_POST['Address'];
$loan = $_POST['Loan'];
//передача данных из формы на сервер if ($first_name == "" or $last_name == "")
{ echo "Необходимо ввести имя - нажмите кнопку Назад и заполните форму еще раз";
exit; }
//проверка введенного имени пользователя if ($age < 10 or $age > 130)
{ echo "Введен некорректный возраст - нажмите кнопку Назад и заполните форму еще раз";
exit; }
//проверка введенного возраста пользователя if ($address == "")
{ echo " Необходимо ввести адрес - нажмите кнопку Назад и заполните форму еще раз";
exit; }
//проверка введенного адреса пользователя
if ($loan != "1000" and $loan != "5000" and $loan != "10000")
{ echo "Необходимо ввести сумму ссуды - нажмите кнопку Назад и заполните форму еще раз ";
exit; }}
//проверка введенной суммы запрашиваемого кредита
?>
</BODY>
</HTML>
```

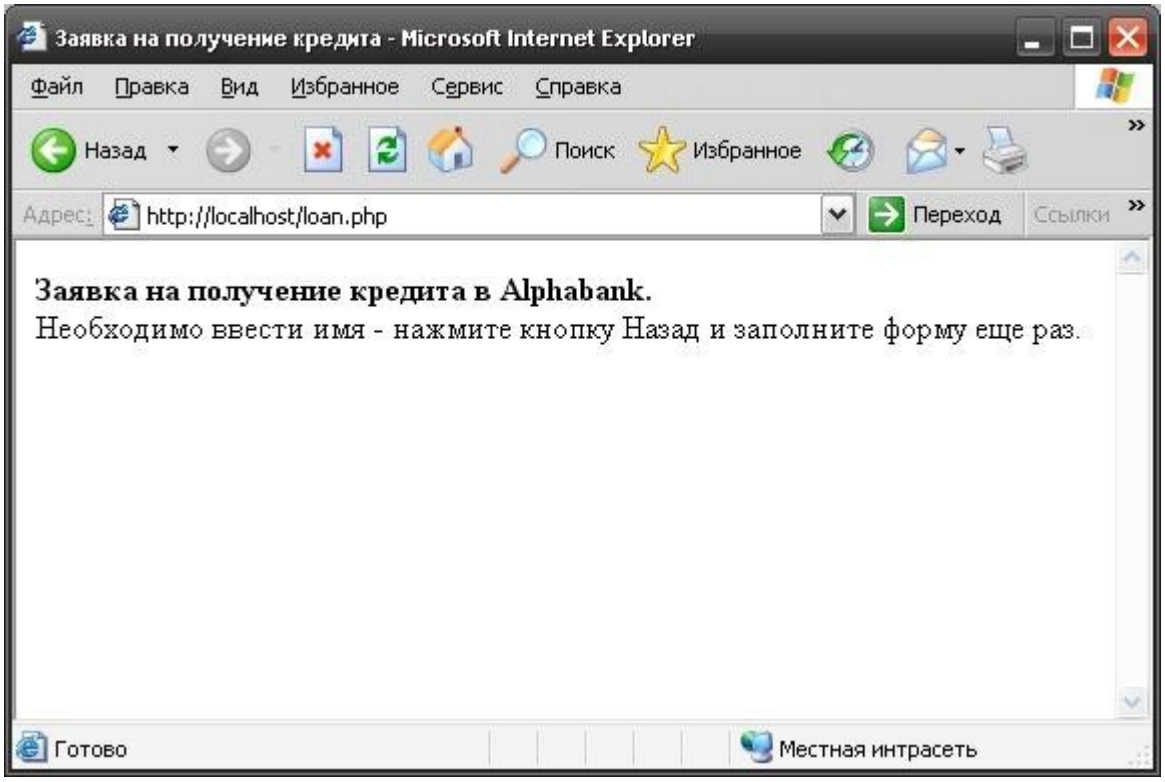


Рисунок 12.6 – Результат подачи заявки в банк. Сообщение об ошибке

Как видно на рисунке 12.6, на экран выводится одно из сообщений об ошибке, так как пользователь не заполнил некоторые поля формы. В данном случае в программе проверяется существование заданной переменной с помощью встроенной функции isset. Также в программе проверяется, не ввел ли пользователь очевидно неверную информацию. Если необходимое условие не соблюдается, то сценарий отображает сообщение об ошибке и завершает работу [4].

Задание

Изучить теоретическое обоснование и приведенные в нем примеры.

Содержание отчета и его форма

Отчет по лабораторной работе должен состоять из:

1) названия лабораторной работы;
2) описание скрытых полей при совместном использованииPHP и HTML-форм;
3) описание полей для ввода паролей при совместном использовании PHP и HTML-форм.

Вопросы для защиты работы:

1. Как совместно использовать PHP и скрытые поля HTML-форм?
2. Как совместно использовать PHP поля ввода паролей HTML-форм?

Лабораторная работа №13. Использование стандартных операторов языка PHP при обработке данных пользователя из форм: булевые операторы, if, операторы сравнения, логические операторы

Цель работы: научиться использовать стандартные операторы языка PHP при обработке данных пользователя из форм.

Теоретическое обоснование

1 Использование булевых операторов и оператора IF

Оператор if, при работе с формами, используется для проверки на правильность введенной информации пользователя. При этом если пользователь случайно сделает ошибку или введет некорректные данные, то программа осуществляет проверку и выводит соответствующее сообщение.

Ниже приведен пример сценария, в котором осуществляется деление двух чисел между собой. При этом необходимо осуществить проверку, чтобы не произошло деление на нуль. То есть, чтобы второе число не было равно нулю. Если осуществится деление на нуль, то сгенерируется ошибка.

HTML-код программы проверки деления чисел представлен ниже.

```
<HTML>
<HEAD>
<TITLE>Пример использования оператора if</TITLE>
</HEAD>
<BODY>
<?php if (isset($_POST['posted']))
{ $first_number = $_POST['first_number']; $second_number =
$_POST['second_number']; if ($second_number == 0) echo
"Нанульделитьнельзя!!!<BR>"; else { $answer = $first_number /
$second_number; echo "Ответ: " . $answer . "<BR>";}
} //проверка введенных чисел пользователя
?>
<B>Деление любого числа (кроме нуля)</B>
<HR align="left" width="300" color="gray"><BR>
<FORM method="POST" action="example.php">
<INPUT type="hidden" name="posted" value="true">
Пожалуйста, введите первое число: <BR>
<INPUT name="first_number" type="text" value="0"><BR>
Пожалуйста, введите второе число: <BR>
<INPUT name="second_number" type="text" value="0">
<BR>
<BR>
<INPUT type="submit" value="Разделить">
</FORM>
</BODY>
</HTML>
```

Пример реализации данного кода приведен на рисунках 13.1 и 13.2.

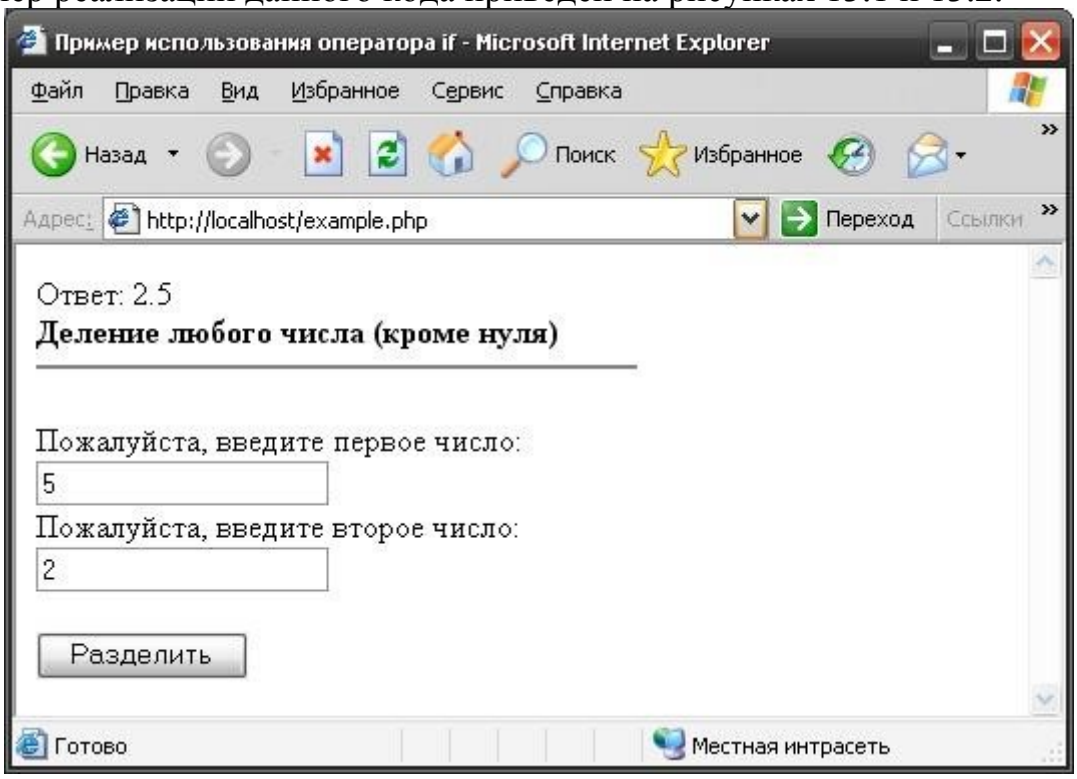


Рисунок 13.1 – Деление на нуль

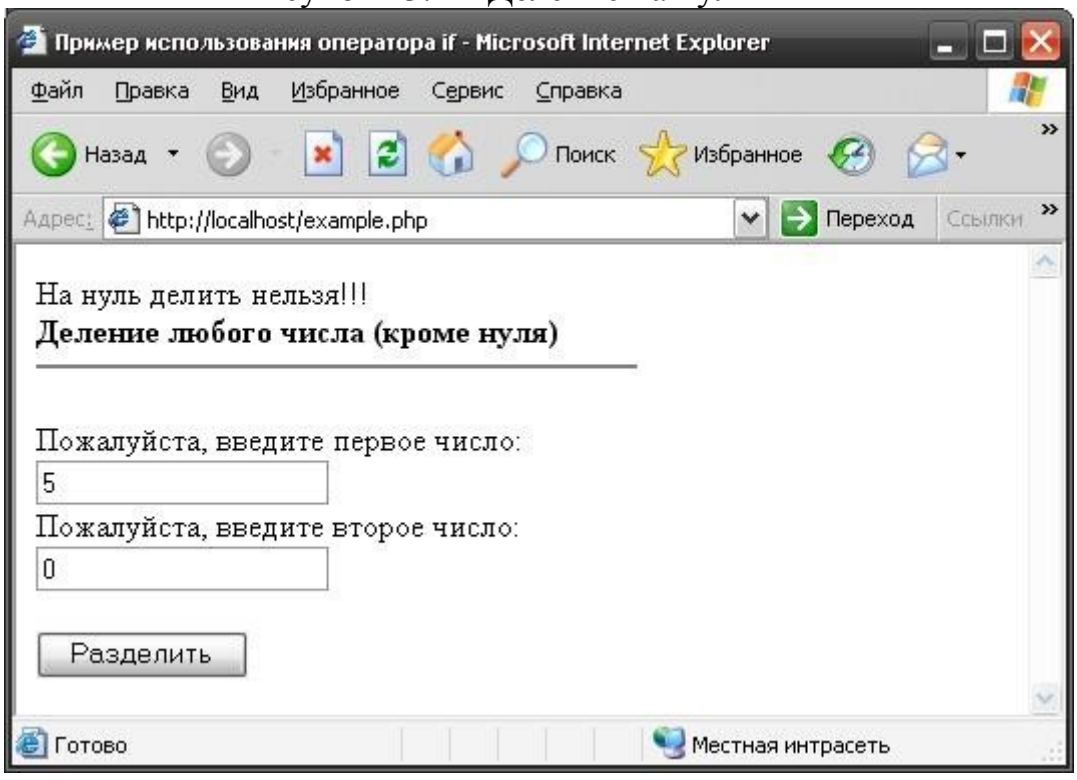


Рисунок 13.2 – Ошибка при делении на нуль

Таким образом, как видно на рисунке 13.1, в данном примере, при загрузке Web-страницы на экране появляется форма для ввода чисел, которые требуется поделить между собой. Затем необходимо нажать кнопку “Разделить”.

В примере, отображенном на рисунке 13.1, пользователь ввел корректные данные, программа выполнила вычисления и вывела результат на экран. При этом пользователю не было выведено никаких сообщений.

На рисунке 13.2 приведен пример обработки некорректных данных. На экран выводится сообщение пользователю “На нуль делить нельзя”, так как в данном примере пользователь ввел нуль в качестве второго значения.

Одна из возможностей уменьшить вероятность возникновения такой ошибки заключается в задании в HTML-коде значений по умолчанию для полей первого и второго чисел, равным не нулю, а единице.

2 Использование операторов сравнения

2.1 Операторы “>” и “<”

Операторы “больше” и “меньше” считаются фундаментальными в математике. В языке PHP они могут использоваться для сравнения любых переменных и констант между собой. При этом в зависимости от результата сравнения, может выполняться определенный набор действий.

Далее представлен пример совместного использования операторов сравнения языка PHP при работе с формами. В данном примере PHPпрограмма “загадывает” число между 1 и 10, а пользователю необходимо угадать это число. Число задается с помощью математической функции rand.

Функция rand используется для генерации случайных чисел в заданных пределах. Этой функции необходимо передать минимальное и максимальное значение, разделенное запятой, после чего она сгенерирует случайное число между двумя этими значениями.

HTML-код программы проверки введенных чисел представлен ниже.

```
<HTML>
<HEAD>
<TITLE>Пример использования операторов сравнения</TITLE>
</HEAD>
<BODY>
<?php if (isset($_POST['posted']))
{
$number = rand(1,10); if ($_POST['guess'] > $number)
{
echo "<B>Ваше число слишком большое.</B>"; echo "<br>Загаданное число:
$number. Вы проиграли. Попробуйте еще раз.<HR>";
} else if ($_POST['guess'] < $number)
{
echo "<B>Ваше число слишком маленькое.</B>"; echo "Загаданное число:
$number. Вы проиграли. Попробуйте еще раз. <HR>";
} else  { echo "<br> Загаданное число: $number. Вывыиграли!<HR>";}
}
```

```
?>
<FORM method="POST" action="example.php"> <INPUT type="hidden" name="posted" value="true">
Угадайтечислоот 1 до 10:
<INPUT name="guess" type="text">
<BR>
<BR>
<INPUT type="submit" value="Отправить">
</FORM>
</BODY>
</HTML>
```

На рисунке 13.3 представлен результат выполнения данного кода.

Таким образом, как видно на рисунке 13.3, в данном примере при загрузке Web-страницы появляется поле формы, в которую пользователь должен ввести число. После этого, с помощью операторов сравнения, осуществляется проверка соответствия введенного и загаданного числа. Как видно на рисунке, загаданное число было равно восьми, а число пользователя четырем, то есть число пользователя слишком маленькое. О чем было выведено соответствующее сообщение.

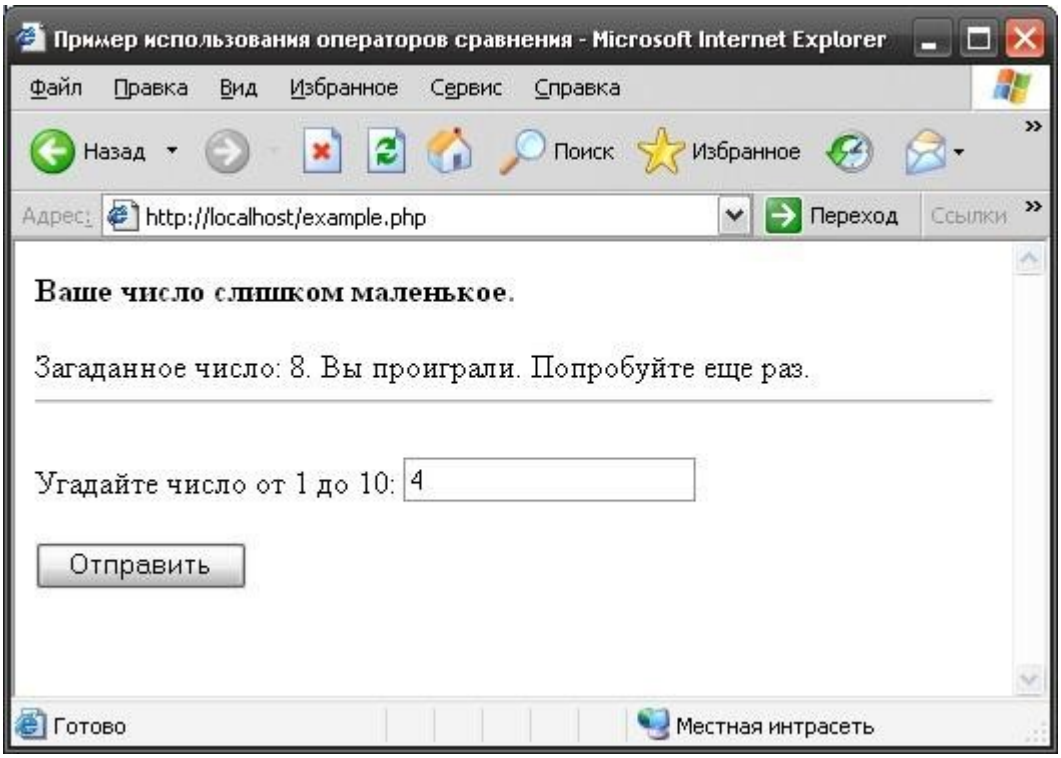


Рисунок 13.3 – Использование операторов сравнения

2.2 Операторы “!=” и “==”

При работе с формами в языке PHP, помимо операторов сравнения, также часто используют операторы неравенства для проверки введенной пользователем информации. Ниже приведен пример использования этих операторов для проверки информации о столице России.

HTML-код программы проверки введенных данных представлен ниже.

```
<HTML>
<HEAD>
<TITLE>Пример использования операторов равенства и неравенства
</TITLE>
</HEAD>
<BODY>
<?php if (isset($_POST['posted'])) { if ($_POST['Question'] == "Москва")
{ echo "<B>Верно, $_POST['Question'] - правильныйответ.</B><HR>";} if
($_POST['Question'] != "Москва") { echo "<B>Неверно, $_POST['Question']
–неправильныйответ.
</B><HR>"; }}
?>
<FORM method="POST" action="example.php">
<INPUT type="hidden" name="posted" value="true">
Назовите столицу России<BR><BR>
<INPUT name="Question" type="radio" value="Москва">
Москва<BR>
<INPUT name="Question" type="radio" value="Екатеринбург">
Екатеринбург<BR>
<INPUT name="Question" type="radio" value="Оренбург">
Оренбург<BR><BR>
<INPUT type="submit" value="Отправитьответ">
</FORM>
</BODY>
</HTML>
```

На рисунке 13.4 представлен результат работы данного кода.

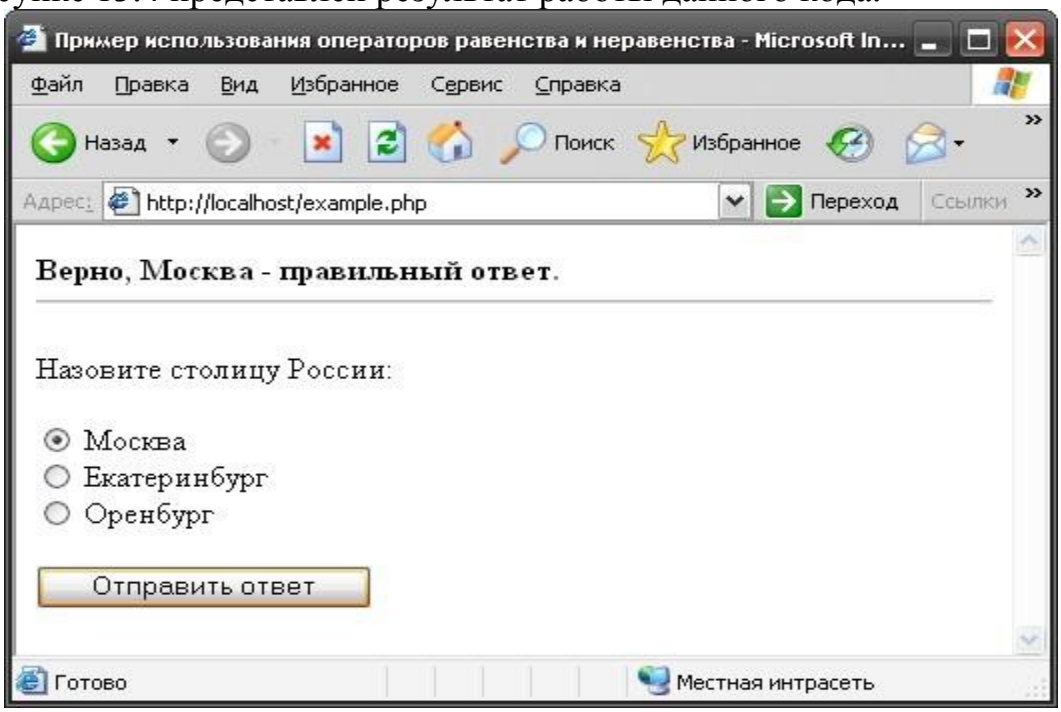


Рисунок13.4 – Проверка на правильность выбора столицы России Таким образом, как видно на рисунке 13.4, в данном примере, при загрузке Web-страницы появляется форма, в которой пользователю необходимо выбрать один из трех вариантов ответа и нажать кнопку “Отправить ответ”. После этого введенный ответ сравнивается с правильным, и выводится соответствующее сообщение. В данном случае пользователю выводится сообщение “Верно, Москва – правильный ответ”, что означает, что он правильно выбрал вариант ответа.

3 Логические операторы

При обработке данных пользователя из форм на стороне сервера также используются логические операторы. К ним относятся такие операторы как операторы логического “И” (and), логического “ИЛИ” (or), логического исключающего “ИЛИ” (xor), а также операторы логического “НЕ”(!).

Ниже приведен пример использования логических операторов для проверки информации о возрасте и наличии водительских прав пользователя для того, чтобы определить, можно ли доверить ему автомобиль напрокат.

HTML-код программы проверки информации пользователя представлен ниже.

```
<HTML>
<HEAD>
<TITLE>Пример использования логических операторов</TITLE>
</HEAD>
<BODY>
<B>Компания Мир Авто. Прокатавтомобилей.</B>
<?php if (isset($_POST['posted'])) { if ($_POST['age'] > 20 and $_POST['license'] == "on") { echo "Уважаемый $_POST['first_name'] $_POST['last_name']. Вам можно
предоставить машину напрокат.<HR>";}
if ($_POST['age'] < 21 or $_POST['license'] == "") {
echo "Уважаемый $_POST['first_name'] $_POST['last_name']. К сожалению, мы не можем предоставить Вам машину напрокат.<HR>";}}
else {
?>
<FORM method="post" action="car.php">
<INPUT type="hidden" name="posted" value="true">
Имя: <INPUT name="first_name" type="text">
Фамилия: <INPUT name="last_name" type="text">
Возраст: <INPUT name="age" type="text"size="3"><BR><BR>
Адрес: <TEXTAREA name="address" rows=4 cols=40> </TEXTAREA><BR><BR>
Ваши водительские права действительны?
<INPUT name="license" type="checkbox"><BR><BR>
<INPUT type="submit" value="Отправитьзаявку">
```

```
</FORM>
</BODY>
</HTML>
<?php
}
?>
```

Результат работы данного кода представлен на рисунках 13.5 и 13.6.

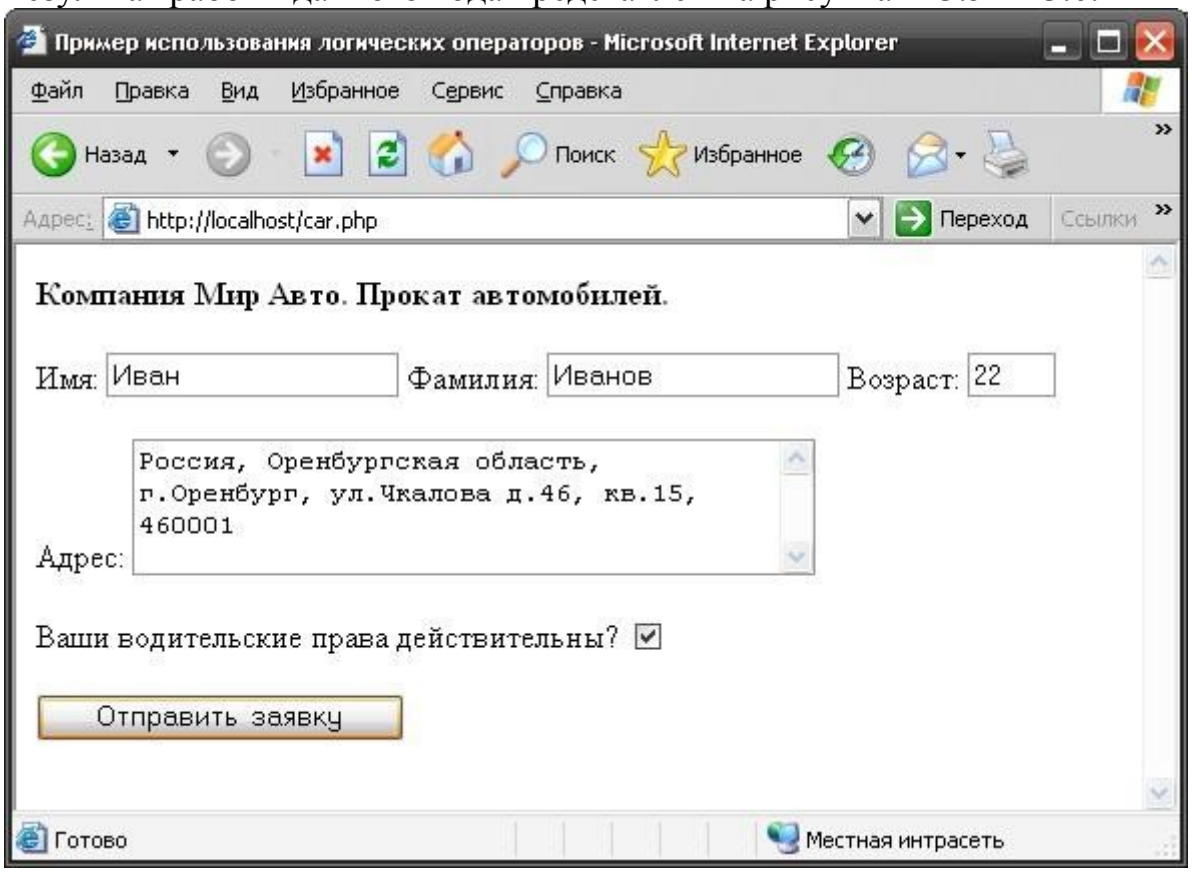


Рисунок 13.5 – Проверка возможности выдачи автомобилей

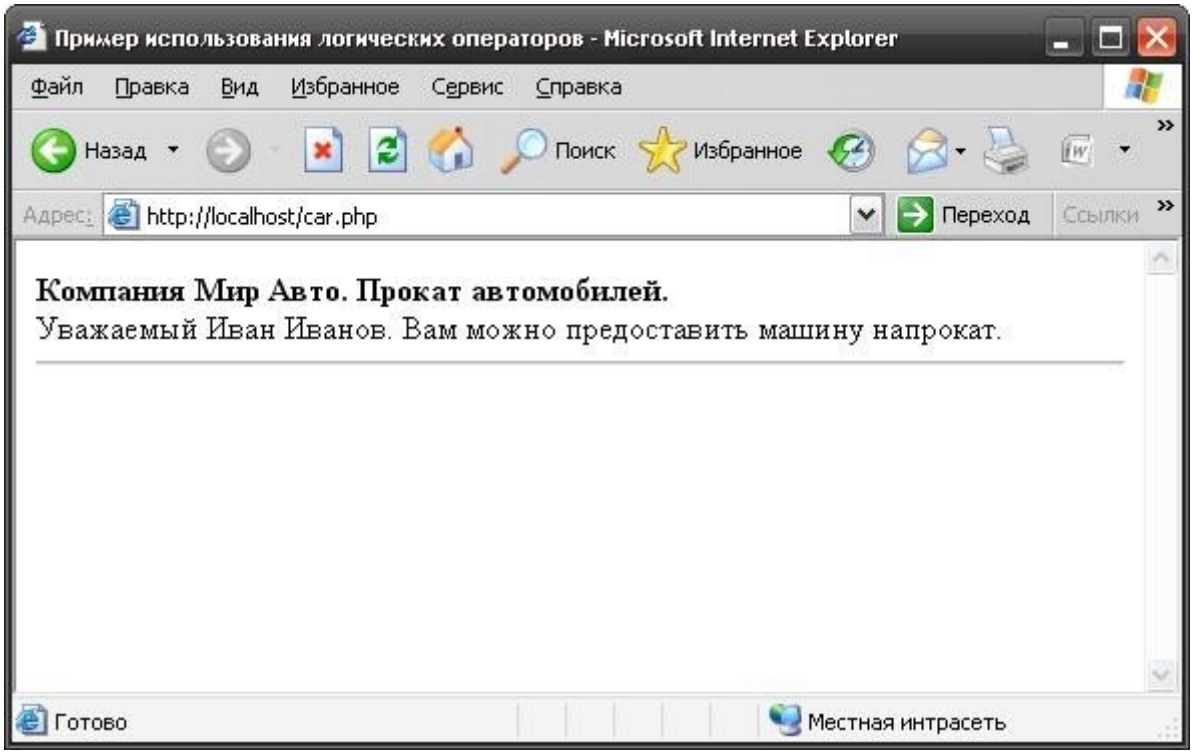


Рисунок 13.6 – Результат проверки

Чтобы собрать дополнительную информацию, используется более сложная по сравнению с уже рассмотренными HTML-форма. В коде используется оператор else, позволяющий отображать форму, только если она не была отправлена пользователем, потому что как только пользователь отправил заявку, нет необходимости отображать форму снова.

Задание

Изучить приведенные в теоретическом обосновании примеры.

Содержание отчета и его форма

Отчет по лабораторной работе должен состоять из:

1) названия лабораторной работы;
2) описание стандартных операторов языка PHP при обработке данных; 3) примеры заполнения форм с операторами if, операторами сравнения, логическими операторами.

Вопросы для защиты работы:

1. Как использовать стандартные операторы языка PHP при обработке данных пользователя из форм?
2. Как использовать булевые операторы?
3. Как использовать операторы сравнения?
4. Как использовать логические операторы?

Лабораторная работа №14. Использование стандартных операторов языка PHP при обработке данных пользователя из форм: оператор SWITCH, использование циклов

Цель работы: научиться использовать стандартные операторы языка PHP при обработке данных пользователя из форм.

Теоретическое обоснование

1 Оператор SWITCH

Наряду с другими операторами цикла, в языке PHP также используется оператор switch для проверки данных пользователя из форм.

Ниже представлен пример использования оператора switch для обработки заказа путевок в Турцию. В зависимости от того, какой был выбран город и какого класса отель, рассчитывается стоимость путевки пользователя.

HTML-код программы расчета стоимости путевок представлен ниже.

```
<HTML>
<HEAD>
<TITLE>Пример использования оператора switch</TITLE>
</HEAD>
<BODY>
<B>Заказ путевок в отели Турции.</B>
<BR><BR>
<?php if (isset($_POST['posted'])) {
$price = 500;
$starmodifier = 3;
$citymodifier = 1;
$destination = $_POST['destination'];
$destgrade = $_POST['destination'] . $_POST ['grade'];
switch($destgrade){ case "Kimerthree":
$citymodifier = 2;
$price = $price * $citymodifier; echo "Недельная стоимость проживания в 3-звёздочном отеле города
Кемер: $price"; break; case "Kimerfour": $citymodifier = 2;
$starmodifier = 4;
$price = $price * $citymodifier * $starmodifier; echo "Недельная стоимость проживания в 4-звёздочном отеле города
Кемер: $price"; break; case "Kimerfive": $citymodifier = 2;
$starmodifier = 5;
$price = $price * $citymodifier * $starmodifier; echo "Недельная стоимость проживания в 5-звёздочном отеле города
Кемер: $price"; break; case "Antaliathree":
$citymodifier = 3.5;
$price = $price * $citymodifier; echo "Недельная стоимость проживания в 3-звёздочном отеле города
Анталия: $price"; break; case "Antaliafour": $citymodifier = 3.5;
$starmodifier = 4;
```

```
$price = $price * $citymodifier * $starmodifier; echo "Недельная стоимость проживания в 4-звёздочном отеле города
Анталия: $price"; break; case "Antaliafive": $citymodifier = 3.5;
$starmodifier = 5;
$price = $price * $citymodifier * $starmodifier; echo "Недельная стоимость проживания в 5-звёздочном отеле города
Анталия: $price"; break; case "Alaniathree":
$price = $price * $citymodifier; echo "Недельная стоимость проживания в 3-звёздочном отеле города
Алания: $price"; break; case "Alaniafour":
$starmodifier = 4;
$price = $price * $citymodifier * $starmodifier; echo "Недельная стоимость проживания в 4-звёздочном отеле города
Алания: $price"; break; case "Alaniafive":
$starmodifier = 5;
$price = $price * $citymodifier * $starmodifier; echo "Недельная стоимость проживания в 5-звёздочном отеле города
Алания: $price"; break; default:
echo "Выберитеснова"; break; }}
?>
<FORM method="POST" action="holiday.php">
<INPUT type="hidden" name="posted" value="true">
<HR align="left" width="300" color="gray">
В каком городе Вы хотели бы провести отпуск? <BR><BR>
<INPUT name="destination" type="radio" value="Alania">Алания
<BR>
<INPUT name= "destination" type="radio" value="Kimer">Кемер
<BR>
<INPUT name="destination" type="radio"
value="Antalia">Анталия<BR><BR>
В отеле какого класса Вы хотели бы остановиться? <BR><BR>
<INPUT name="grade" type="radio" value="three">Тризвездочки
<BR>
<INPUT name="grade" type="radio" value="four">Четырезвездочки<BR>
<INPUT name="grade" type="radio" value="five">Пятьзвездочек
<BR><BR>
<INPUT type="submit" value="Отправитьзаявку">
</FORM>
</BODY>
</HTML>
```

Реализация данного кода представлена на рисунке 14.1.

Таким образом, как видно на рисунке 14.1, в данном примере, при загрузке Web-страницы на экране появляется форма заказа путевки, в которой пользователю необходимо указать город, в котором он хотел бы провести

отпуск, а также класс отеля. При нажатии на кнопку “Отправить заказ”, рассчитывается стоимость путевки, в зависимости от выбора пользователя. После этого стоимость путевки выводится на экран.

При этом обрабатываются все возможные варианты. Три варианта городов, умноженные на три класса отеля. К таким вариантам относятся: город Алания и трехзвездочный, четырехзвездочный и пятизвездочный отели. Также, город Кемер и трехзвездочный, четырехзвездый и пятизвездочный отели. И, наконец: город Анталия и трехзвездочный, четырехзвездочный, пятизвездочный отели.

В данном примере[4] пользователь указал город Анталия и пятизвездочный класс отеля. В соответствии с этим выбором рассчиталась стоимость путевки, равная $8750, включающая недельное проживание, которая была выведена на экран.

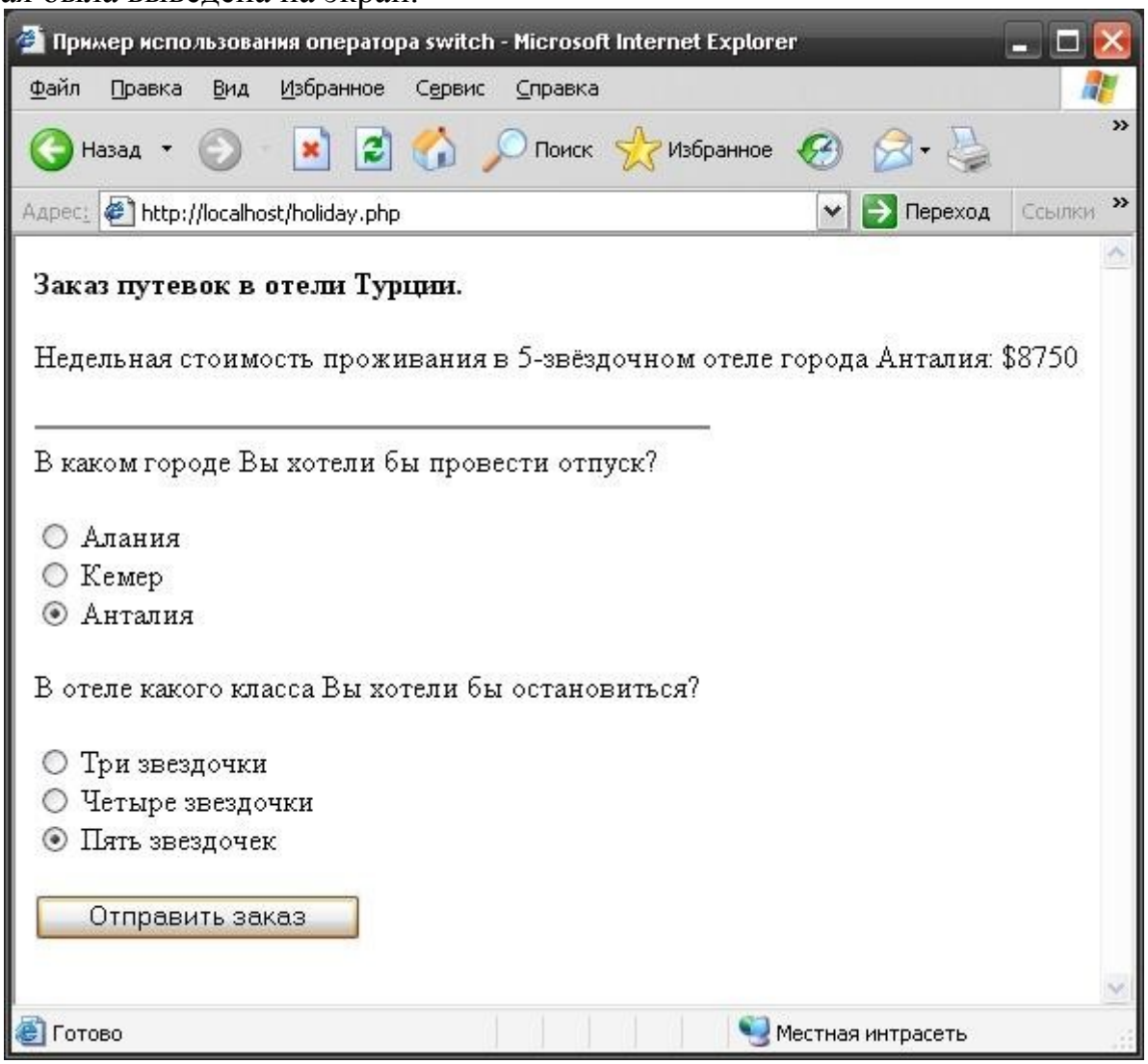


Рисунок 14.1 – Выбор города и класса отеля при заказе путевок

2 Использование циклов при обработке данных пользователя из форм

2.1 Цикл WHILE

Оператор while является одним из популярных операторов цикла, используемым при обработке данных пользователя из форм в языке PHP.

Ниже представлен пример обработки заявки на получение кредита в банке Alphabank. При этом пользователю предлагается выбрать сумму кредита, а также ввести сумму, которую он планирует вносить ежемесячно в счет погашения кредита. После этого в программе рассчитывается период, в течение которого пользователь сможет погасить выбранный кредит.

HTML-код программы расчета срока погашения кредита представлен ниже.

```
<HTML>
<HEAD>
<TITLE>Примериспользованияциклawhile</TITLE>
</HEAD>
<BODY>
<B>Заявка на получение кредита Alphabank.</B>
<?php if (isset($_POST['posted'] )) {
$duration = 0; switch ($_POST['loan']) { case "1000"; $interest=8; break;
case "5000"; $interest=11,5; break; case "10000"; $interest=15,0; break; default:
echo "Выневыбралисуммукредита<HR>"; exit; } while ($_POST['loan'] >
0)
{
$duration = $duration + 1;
$monthly = $_POST['month'] - ($_POST['loan']*$interest/100); if ($monthly
<= 0)
{
echo "Чтобы погасить кредит, требуются более крупные
ежемесячныеплатежи<HR>";
exit; }
$_POST['loan'] = $_POST['loan'] - $monthly;
}
echo "Для погашения кредита при процентной ставке $interest%
понадобится $duration месяцев.<HR>"; }
?>
<FORM method="POST" action="loan.php">
<INPUT type="hidden" name="posted" value="true"><BR>
Выберите сумму необходимого кредита:
<BR>
<INPUT name="loan" type="radio" value="1000">
$1000 под 8,0% годовых<BR>
<INPUT name="loan" type="radio" value="5000">
$5000 под 11,5% годовых<BR>
<INPUT name="loan" type="radio" value="10000">
$10000 под 15,0% годовых <BR><BR>
Введите сумму ежемесячного платежа в долларах:
<INPUT name="month" type="text" size="5"><BR>
<INPUT type="submit" value="Податьзаявку">
```

```
</FORM>
</BODY>
</HTML>
```

Реализация данного кода представлена на рисунке 14.2.

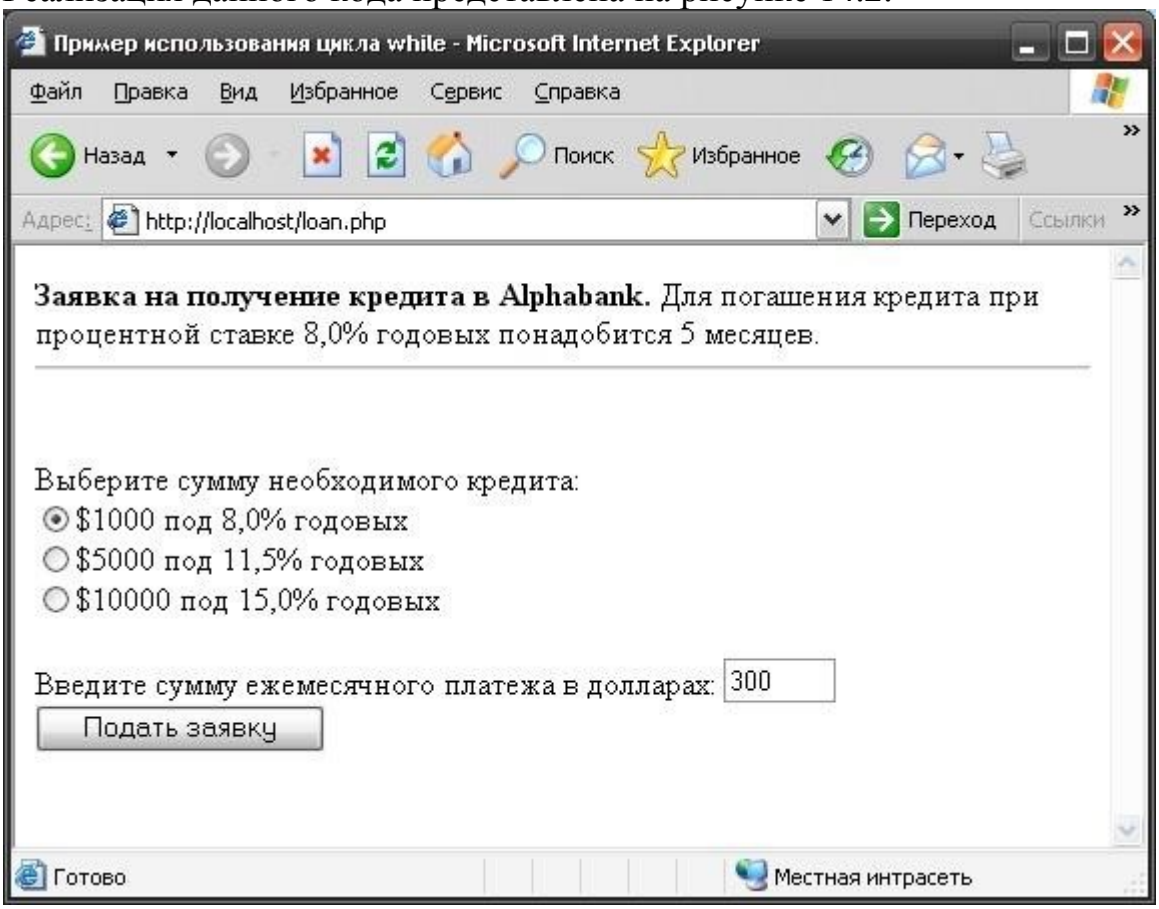


Рисунок 14.2 – Расчет времени погашения кредита в месяцах

Таким образом, как видно на рисунке 14.2, в данном примере, при загрузке Web-страницы на экране появляется форма для расчета периода выплаты кредита. В данной форме пользователю необходимо выбрать сумму кредита, а также ввести предполагаемую сумму ежемесячного платежа. После этого нажать кнопку “Подать заявку”. После обработки заявки на экран выведется сообщение, в котором пользователь сможет узнать, сколько ему месяцев необходимо будет выплачивать кредит.

Как видно на рисунке 14.2, пользователь выбрал сумму кредита $1000 под 8% годовых и сумму ежемесячного платежа $300. В результате программа выдала ответ, что, при заданных параметрах, пользователю понадобится 5 месяцев, чтобы погасить выбранный кредит.

При этом необходимо отметить, что если пользователь введет слишком маленькую сумму ежемесячного платежа, ему выводится соответствующее сообщение “Чтобы погасить кредит, требуются более крупные ежемесячные платежи”. Так как, при этом условии цикл будет продолжаться бесконечно долго, потому что условие “сумма кредита” больше нуля всегда будет верным.

Поэтому, если создается условие для бесконечного цикла, следует выйти из цикла с помощью команды exit, предварительно отправив пользователю соответствующее сообщение (рисунок 14.3).

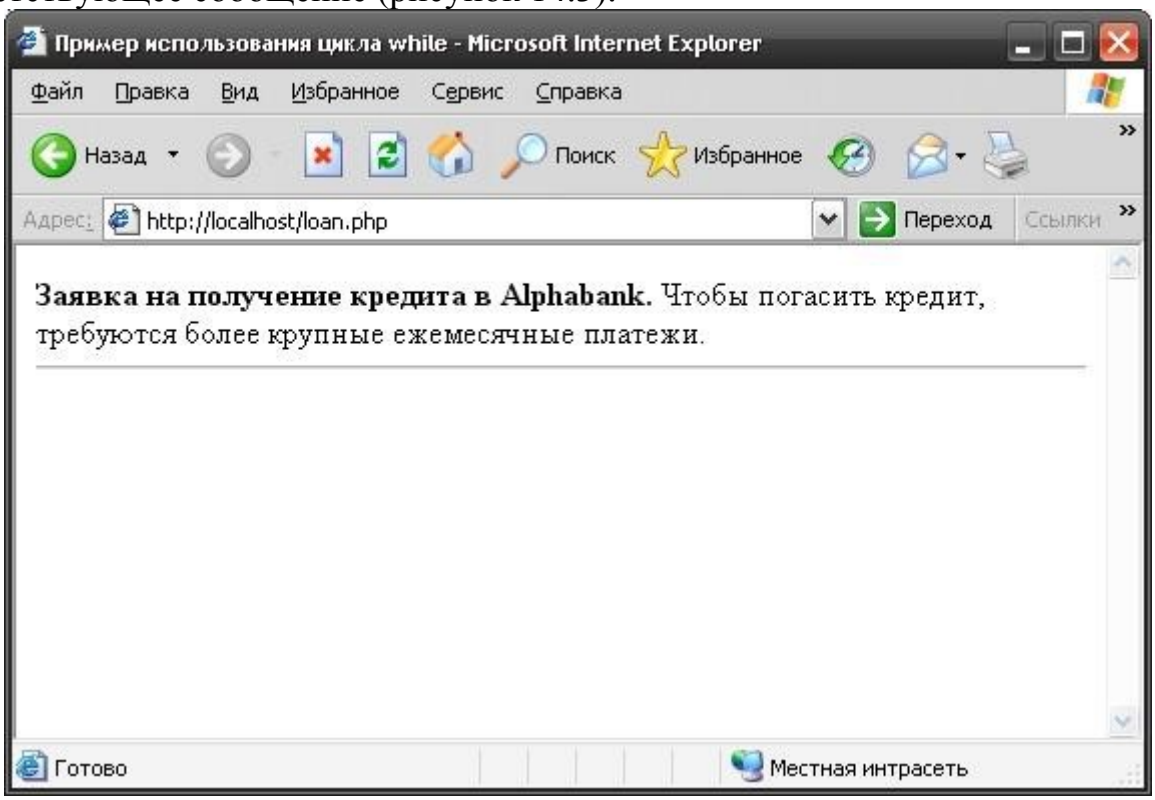


Рисунок 14.3 – Сообщение, выводимое при бесконечном цикле

Таким образом, если ли в программе возникает бесконечный цикл, браузер может, в конце концов, зависнуть, а если не предпринимать определенных мер, то зависнет и Web-сервер. Это означает, что необходимо позаботиться о том, чтобы бесконечные циклы не возникали.

2.2 Цикл DO WHILE

Оператор do while работает аналогично while, за исключением того, что условие проверяется в конце цикла, а не в начале. В этом заключается важное отличие оператора do while: код в фигурных скобках выполняется, по крайней мере, один раз, даже если условие ложно.

Ниже представлен пример использования оператора do while для проверки вводимого числа пользователя на предмет того, является ли оно простым. При этом простым является число, которое делится без остатка только на себя и на единицу.

HTML-код программы проверки вводимого числа представлен ниже.

```
<HTML>
<HEAD>
<TITLE>Примериспользованияцикладowhile</TITLE>
</HEAD>
<BODY>
<?php if (isset($_POST['posted']))
```

```
{ $count=2; do { $remainder = $_POST['guess'] % $count;
$count = $count + 1; } while ($remainder != 0 and $count <
$_POST['guess']); if (($count < $_POST['guess']) || ($_POST['guess'] == 0))
{echo "<B>Введенное число не является простым.<B><HR>";} else
{ echo "<B>Введенное число является простым.</B>HR>";} }
?>
<FORM method="POST" action="example.php"> <INPUT type="hidden"
name="posted" value="true">
Введитечисло:
<INPUT name="guess" type="text"><BR><BR>
<INPUT type="submit" value="Проверитьчисло">
<BR>
</FORM>
</BODY>
</HTML>
```

Реализация данного кода представлена на рисунке 14.4.

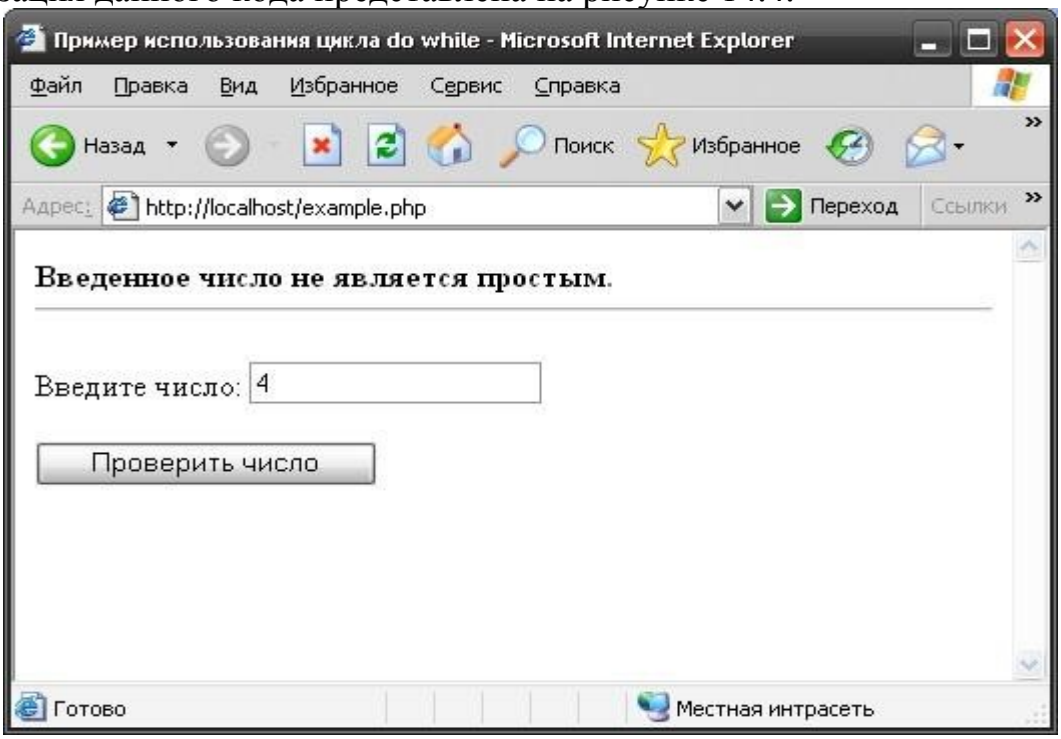


Рисунок 14.4 – Проверка простоты числа

Таким образом, как видно на рисунке 14.4, в данном примере, при загрузке Web-страницы, на экране появляется форма для ввода числа пользователя. После нажатия кнопки “Проверить число” программа проверяет, является ли это чело простым и выводит соответствующее сообщение. В данном примере пользователь ввел число четыре, которое не является простым, о чем пользователю было выведено соответствующее сообщение.

2.3 Цикл FOR

Оператор for также используется в языке PHP для обработки данных пользователя из форм.

Далее представлен пример использования цикла for для динамического ввода имен детей пользователей, в зависимости от того, сколько у него детей.

В программе создается динамическая форма, которая принимает от пользователя некоторое число, использует это число для установки количества элементов управления, отображаемых на второй странице, а затем на третьей странице отображает содержимое этих элементов управления.

HTML-код программы для ввода информации о детях пользователя представлен ниже.

```
<HTML>
<HEAD>
<TITLE>Примериспользованияциклаfor</TITLE>
</HEAD>
<BODY>
<?php if(isset($_POST['posted']))
{ echo "<form method='POST' action='dynamic.php'>"; for ($counter = 0; $counter < $_POST['number']; $counter++)
{ $offset = $counter + 1; echo "<br>Введитеимя $offset-го ребенка<br>"; echo "<input name='child[]' type='text'>";} echo "<br>Для продолжения нажмите кнопку<<br>"; echo "<input type='submit' value='Далее'>"; echo "<input type='hidden' name='posted01' value='true'></FORM>";} else { if (isset($_POST['posted01']))
{ $count=0;
echo "<B>Имена Ваших детей: </B>"; do { $childs_name=$_POST['child'][$count]; echo"<br>$childs_name"; $checkempty = $childs_name; $count = $count + 1; } while ($checkempty != "");
if ($count == 1)
{ echo "Введите другое число"; } }
?>
<HR>
<FORM method="POST" action="examlpe.php"> <INPUT type="hidden" name="posted" value="true">
СколькоуВасдетей?
<INPUT name="number" type="text"><BR><BR>
<INPUT type="submit" value="Отправитьчисло"><BR>
</FORM>
<?php } ?>
</BODY>
</HTML>
```

Результат работы данного примера представлен на рисунках 14.5, 14.6 и14.7.

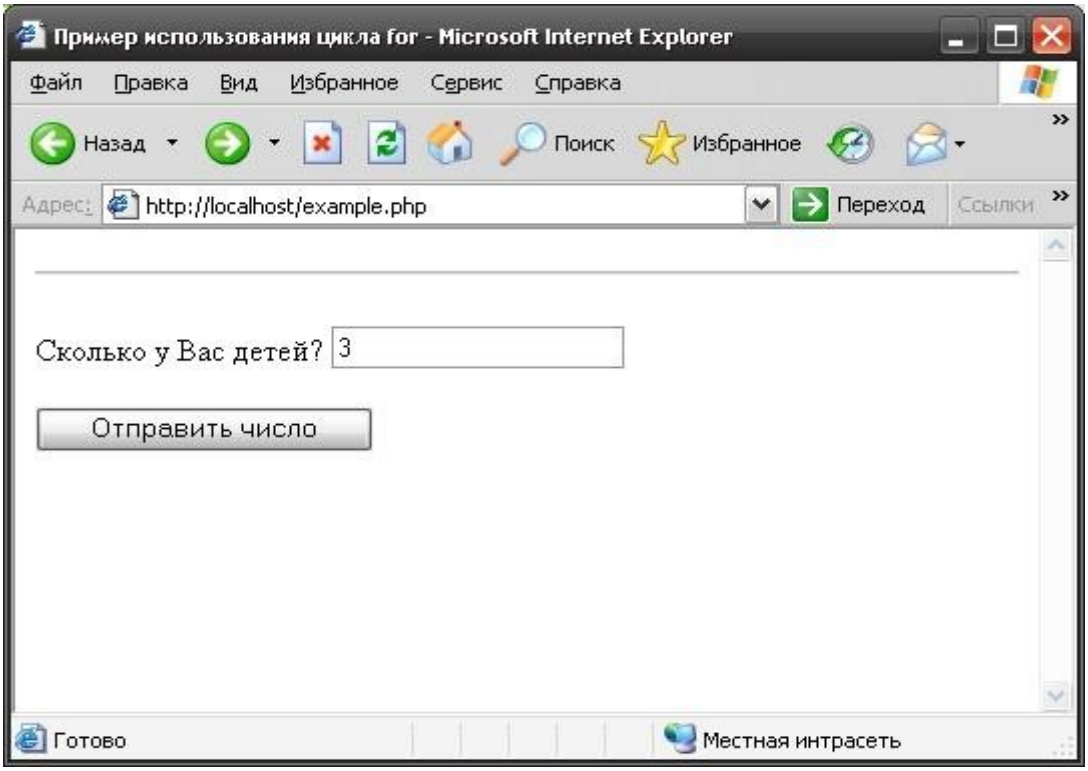


Рисунок 14.5 – Форма ввода количества детей

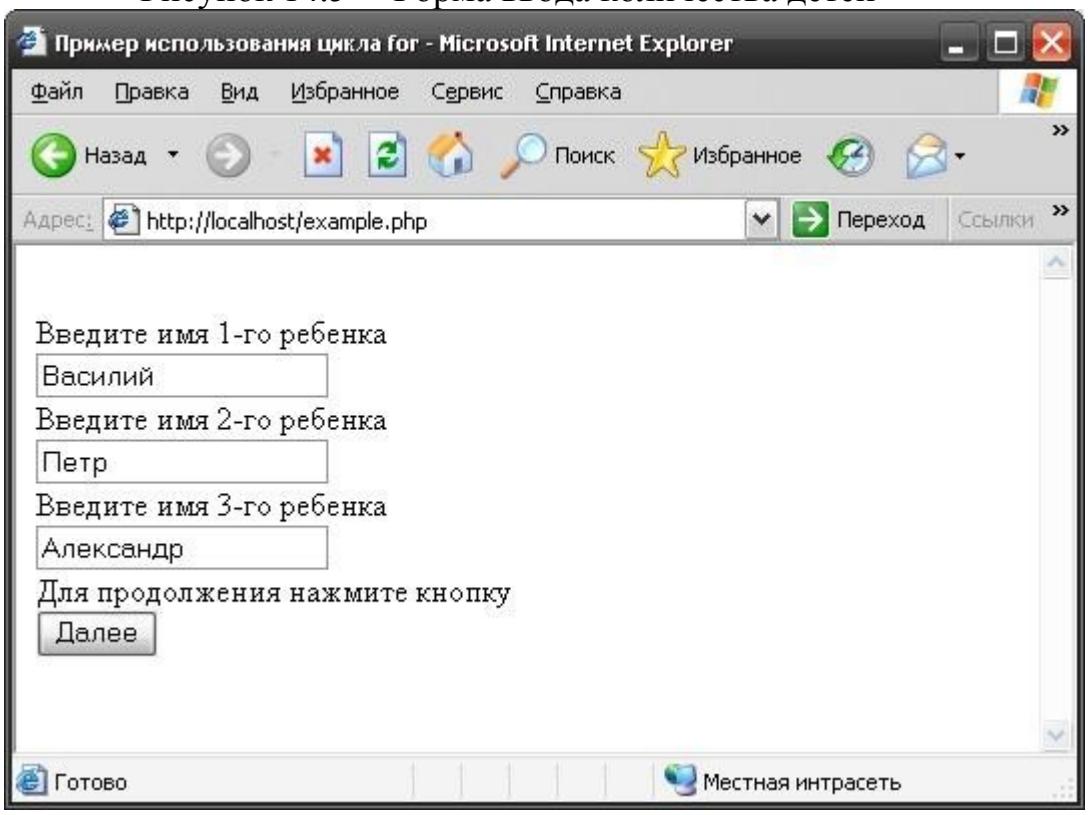


Рисунок 14.6 – Динамическая форма ввода имён детей

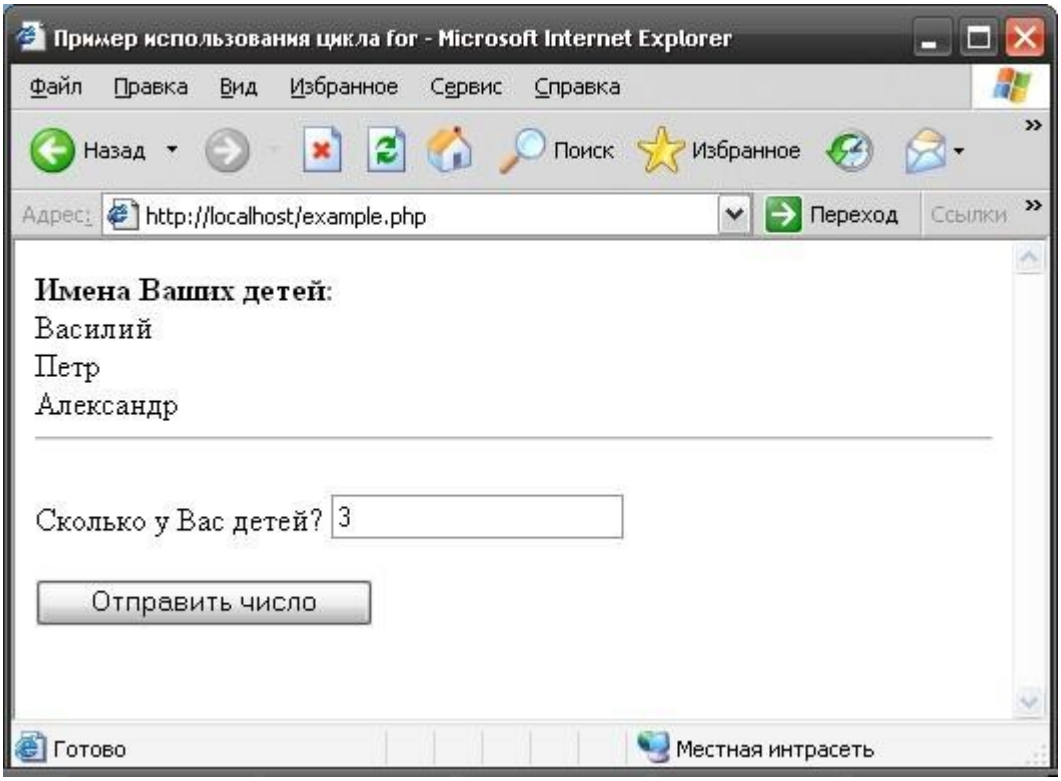


Рисунок 14.7 – Результат ввод имён детей

Таким образом, как видно на рисунке 14.5, в данном примере при загрузке Web-страницы появляется форма, в которую пользователю требуется ввести число детей. После нажатия кнопки “Отправить число” данное число передается сценарию. Программа, таким образом, сообщает браузеру, что необходимо создать форму, состоящую из того количества полей, которые указал пользователь.

После этого на экране отображается несколько текстовых полей, количество которых соответствует значению, указанному пользователем. В каждое из этих полей пользователю необходимо ввести имя очередного ребенка и нажать кнопку “Далее” (рисунок 14.6). Однако если у пользователя нет детей, то отображаться текстовые поля не будут. В результате программа выводит имена детей на экран (рисунок 14.7).

Как видно из рисунков 14.5 - 14.7, пользователь ввел количество детей, равное трем. После этого, на следующей странице появилось три текстовых поля, в которые пользователь ввел имена детей: “Василий”, “Петр”, “Александр”. На третьей странице на экран выводятся эти имена детей.

Таким образом, циклы в языке PHP являются мощным средством для обработки данных из форм. Так, если в данном примере пользователь ввел бы большое число детей, то без использования циклов в программе пришлось бы использовать большое количество строк для считывания и вывода.

Задание

Изучить примеры, приведенные в теоретическом обосновании.

Изменить вводимую информацию (например, количество и имена детей).

Содержание отчета и его форма

Отчет по лабораторной работе должен состоять из:

1) названия лабораторной работы;
2) описание примеров использования оператора SWITCH при работе с формами;
3) описание примеров использования циклов.

Вопросы для защиты работы:

1. Как использовать оператор SWITCH?
2. Каким образом используются циклы при работе с формами?

Лабораторная работа №15. Предотвращение катастроф и восстановление

Цель работы: научиться использовать инструменты предотвращения катастроф с БД

Теоретическое обоснование

1 Резервное копирование баз данных

Поскольку таблицы MySQL хранятся в виде файлов, то резервное копирование выполняется легко. Чтобы резервная копия была согласованной, выполните на выбранных таблицах LOCK TABLES, а затем FLUSH TABLES для этих таблиц. При этом требуется блокировка только на чтение; поэтому другие потоки смогут продолжать запросы на таблицах в то время, пока будут создаваться копии файлов из каталога базы данных. Команда FLUSH TABLE обеспечивает гарантию того, что все активные

индексные страницы будут записаны на диск прежде, чем начнется резервное копирование.

Если есть необходимость провести резервное копирование на уровне SQL, то можно воспользоваться SELECT INTO OUTFILE или BACKUP TABLE [4].

Существует еще один способ создать резервную копию базы данных - использовать программу mysqldump или сценарий mysqlhotcopy . Для этого нужно выполнить следующие действия:

Сделать полное резервное копирование баз данных: shell> mysqldump --tab=/path/to/some/dir --opt --full

или

shell> mysqlhotcopy database /path/to/some/dir

Можно также просто скопировать табличные файлы (файлы '*.frm', '*.MYD' и '*.MYI') в тот момент, когда сервер не проводит никаких обновлений. Этот метод используется в сценарии mysqlhotcopy.

Если mysqld выполняется, остановить его, и затем запустить с опцией -log-update[=file_name]. В файлах журнала обновлений находится информация, необходимая для того, чтобы повторить в базе данных последовательность изменений, внесенных с момента выполнения mysqldump.

Если дело дошло до восстановления, сначала надо попробовать восстановить таблицы с помощью REPAIR TABLE илиmyisamchk -r - это должно сработать в 99,9% случаев. Если myisamchk не даст результата, попробуйте применить следующую процедуру (эти действия применимы только в случае, если MySQL запускался с --log-update:

Восстановите исходный вариант по копии, сделанной в mysqldump.

Выполните следующую команду, чтобы повторить обновления из бинарного журнала:

shell> mysqlbinlog hostname-bin.[0-9]* | mysql

Если используется журнал обновлений, то можно применить: shell> ls -1 -t -r hostname.[0-9]* | xargs cat | mysql

ls используется для того, чтобы расположить все файлы журнала

обновлений в правильном порядке.

Можно проводить избирательное резервное копирование посредством SELECT * INTO OUTFILE 'file_name' FROM tbl_name, а восстановление - при помощи LOAD DATA INFILE 'file_name' REPLACE ... Чтобы избежать повторения записей, в таблице должен быть первичный или уникальный ключ. Ключевое слово REPLACE задает замену старых записей новыми в случае, когда новая запись в значении уникального ключа повторяет старую.

Если в системе, где выполняется резервное копирование, возникают проблемы с производительностью, то решить их можно, установив репликацию и выполняя резервное копирование на подчиненном сервере вместо головного.

Пользователи файловой системы Veritas могут поступить следующим образом:

Изклиента (или Perl) выполнить: FLUSH TABLES WITH READ LOCK.

Из другого shell выполнить: mount vxfs snapshot.

Из первого клиента выполнить: UNLOCK TABLES.

Скопировать файлы из образа.

Демонтировать образ.

2 Синтаксис BACKUP TABLE

BACKUP TABLE tbl_name[,tbl_name...] TO '/path/to/backup/directory'

Копирует в каталог резервного копирования тот минимум табличных файлов, который достаточен для восстановления таблицы. На данный момент работает только для таблиц MyISAM. Для таблиц MyISAM копирует файлы '.frm'(определений) и '.MYD' (данных). Индексные файлы могут быть реконструированы по этим двум.

В процессе резервного копирования будет установлена блокировка чтения отдельно для каждой таблицы на время ее копирования. Если

необходимо сделать резервное копирование в виде мгновенного образа нескольких таблиц, необходимо сначала запросить LOCK

TABLES установки блокировки чтения для каждой таблицы в группе.

Команда возвращает таблицу (15.1) со следующими столбцами:

Таблица 15.1 – Результат выполнения блокировки

Столбец	**Значение**
Table	Имя таблицы
Op	Всегда "backup"
Msg_type	Одно из значений status, error, info или warning.
Msg_text	Само сообщение.

Заметим, что BACKUP TABLE доступна только в версии MySQL 3.23.25 и выше.

3 Синтаксис RESTORE TABLE

RESTORE TABLE tbl_name[,tbl_name...] FROM '/path/to/backup/directory'

Восстанавливает таблицу(ы) из резервной копии, созданной с помощью BACKUP TABLE. Существующие таблицы не перезаписываются: при попытке восстановления поверх существующей таблицы будет выдана ошибка. Восстановление занимает больше времени, нежели BACKUP - из-за необходимости повторного построения индекса. Чем больше в таблице будет ключей, тем больше времени заберет реконструкция. Эта команда, так же как и BACKUP TABLE, в настоящее время работает только для таблиц MyISAM.

Команда возвращает таблицу (15.2) со следующими столбцами:

Таблица 15.2 – Результат выполнения RESTORETABLE

Столбец	**Значение**
Table	Имя таблицы
Op	Всегда "restore"
Msg_type	Одно из значений status, error, info или warning.
Msg_text	Само сообщение.

4 Синтаксис CHECK TABLE

CHECK TABLE tbl_name[,tbl_name...] [option [option...]]

option = QUICK | FAST | MEDIUM | EXTENDED | CHANGED

CHECK TABLE работает только на таблицах MyISAM и InnoDB. На таблицах типа MyISAM команда эквивалентна запуску на таблице myisamchk -m table_name.

Если опция не указана, используется MEDIUM.

Проверяет таблицу(ы) на наличие ошибок. Для таблиц MyISAM обновляется статистика ключей. Команда возвращает таблицу (15.3) со следующими столбцами:

Таблица 15.3 – Результат выполнения CHECK TABLE

Столбец	Значение
Table	Имя таблицы.
Op	Всегда "check".
Msg_type	Одно из значений status, error, info, или warning.
Msg_text	Само сообщение.

Заметим, что по каждой проверяемой таблице может быть выдано много строк информации. Последняя строка будет представлять Msg_type status и, как правило, должна содержать OK. Если выдается что-либо отличное от OK и Not checked, то обычно следует провести ремонт таблицы[4]. Not checked свидетельствует о том, что указанный для таблицы тип (TYPE) не нуждается в проверке.

Различные типы проверки представлены в таблице15.4.

Таблица 15.4 – Операторы, используемые для выполнения проверки

Тип	Действия
QUICK	Не сканировать строки для проверки на неправильные связи.
FAST	Проверять только таблицы, которые не были корректно закрыты.
CHANGED	Проверять только таблицы, которые изменились со времени последней проверки или не были закрыты корректно.
MEDIUM	Сканировать строки для проверки того, что уничтоженные связи в порядке. При этом также подсчитывается ключевая контрольная сумма для строки и сравнивается с подсчитанной контрольной суммой для ключей.
EXTENDED	Выполнить полный просмотр ключа для всех ключей для каждой строки. Успех такой проверки гарантирует 100%
	отсутствие противоречий в таблице, но на проверку уйдет немало времени!

Для таблиц MyISAM с динамическими размерами при запуске

проверки всегда выполняется проверка MEDIUM. Для строк со статическими размерами мы пропускаем сканирование строк для QUICK и FAST, поскольку повреждение строк происходит крайне редко.

Проверочные опции можно сочетать:

CHECK TABLE test_table FAST QUICK;

Эта команда просто вызовет быструю проверку таблицы для выявления того, была ли она закрыта корректно.

Примечание: в некоторых случаях CHECK TABLE изменяет таблицу!

Это происходит, если таблица помечена как 'поврежденная/corrupted' или 'не закрытая корректно/not closed properly', а CHECK TABLE не находит никаких проблем в таблице. В этом случае CHECK TABLE отметит в таблице, что с ней все нормально.

Если таблица повреждена, то, скорее всего, проблема в индексах, а не в данных. Проверки всех типов обеспечивают всестороннюю проверку индексов и тем самым должны обнаруживать большинство ошибок.

Если проверяется таблица, с которой предположительно все нормально, то можно опустить проверочные опции или указать опцию QUICK. Последнюю возможность следует использовать в случае ограничений по времени и тогда, когда можно пойти на риск (очень незначительный), что QUICK пропустит ошибку в файле данных. (В большинстве случаев MySQL должен найти - при нормальной работе - любые ошибки в файле с данными.Если ошибки найдены, то таблица будет отмечена как 'поврежденная/corrupted', и в таком случае ее нельзя будет использовать, пока она не будет исправлена.)

FAST и CHANGED главным образом предназначены для использования в сценариях (например, для запуска из cron), если необходимо время от времени проверять таблицы. В большинстве случаев следует отдавать предпочтение FAST передCHANGED (иначе надо поступать только в случае, когда возникает подозрение, что найдена ошибка в самом кодеMyISAM).

Прибегать к EXTENDED следует только тогда, когда после выполнения нормальной проверки для таблицы по-прежнему выдаются странные ошибки при попытке MySQL обновить строку или найти строку по ключу (что очень маловероятно в случае успеха нормальной проверки!).

Некоторые проблемы, о которых сообщается при проверке таблицы, нельзя исправить автоматически:

Found row where the auto_increment column has the value 0. Это означает, что в таблице есть строка, где индексированный столбец AUTO_INCREMENT содержит значение 0 (строку, в которой столбец AUTO_INCREMENTимеет значение 0, можно создать, явно установив столбец в 0 командой UPDATE). Это само по себе не является ошибкой, но может вызвать неприятности, если понадобится сделать дамп таблицы или восстановить ее или выполнить над ней ALTER TABLE. В этом случае столбец с атрибутом AUTO_INCREMENT изменит значение в соответствии с

правилами для столбцов AUTO_INCREMENT, что может вызвать проблемы, подобные ошибке дублирования ключа. Чтобы избавиться от предупреждения, просто выполните команду UPDATE для установки в столбце значения, отличного от 0.

5Синтаксис REPAIR TABLE

REPAIR TABLE tbl_name[,tbl_name...] [QUICK] [EXTENDED] [USE_FRM]

REPAIRTABLE работает только на таблицах типа MyISAM и эквивалентна выполнению на таблице myisamchk -rtable_name.

При обыкновенной работе запускать эту команду не приходится, но если случится катастрофа, то с помощью REPAIR TABLE практически наверняка удастся вернуть все данные из таблицы MyISAM. Если таблицы сильно повреждены, то следует постараться выяснить, что послужило этому причиной[4].

REPAIR TABLE ремонтирует таблицу, которая, возможно, повреждена.

Команда возвращает таблицу (15.5) со следующими столбцами:

Таблица 15.5 – Выполнение REPAIR TABLE

Столбец	**Значение**
Table	Имя таблицы
Op	Всегда "repair"
Msg_type	Одно из значений status, error, info или warning.
Msg_text	Само сообщение.

Заметим, что по каждой ремонтируемой таблице может быть выдано много строк информации. Последняя строка будет представлять Msg_type status и, как правило, должна содержать OK. Если выдается что-либо отличное от OK, то следует попробовать исправить таблицу с помощью myisamchk -o, поскольку в REPAIR TABLE пока реализованы не все опциимyisamchk. В скором будущем мы сделаем команду более гибкой.

Если указан QUICK, то MySQL будет пытаться сделать REPAIR только индексного дерева.

Если используется EXTENDED, то MySQL будет создавать индекс строка за строкой вместо создания по одному индексу единоразово с помощью сортировки; такая техника может работать лучше сортировки для ключей фиксированной длины, если речь идет о хорошо сжимаемых ключах типа char() большой длины.

Что касается MySQL 4.0.2, то тут для REPAIR существует режим USE_FRM. Используйте его, если отсутствует файл '.MYI'или поврежден его заголовок. В этом режиме MySQL воссоздаст таблицу, используя информацию из файла '.frm'. Этот вид исправления в myisamchk недоступен.

Задание

Изучить инструментарии предотвращения катастроф и резервного копирования.

Содержание отчета и его форма

Отчет по лабораторной работе должен состоять из:

1) названия лабораторной работы;
2) описание примеров использования LOCK TABLES/UNLOCK TABLES, BACKUPTABLE, RESTORETABLE, CHECKTABLE, REPAIR TABLE.

Вопросы для защиты работы:

1. Для чего используетсяоператорLOCKTABLES/UNLOCKTABLES?
2. Для чего используется BACKUPTABLE?
3. Для чего используется RESTORETABLE?
4. Для чего используется CHECKTABLE?
5. Для чего используетсяREPAIRTABLE?

Лабораторная работа №16.Репликация в MySQL

Цель работы: научиться настраивать репликацию вMySQL

Теоретическое обоснование

К числу преимуществ, которые обеспечивает репликация, относится повышение скорости и надежности. Чтобы обеспечить надежность, можно установить две системы и при возникновении проблем с головным сервером переключаться на резервную копию. Для увеличения скорости можно перенаправлять те запросы, которые не обновляют данные, на сервер с копиями. Разумеется, это даст эффект лишь в том случае, если запросы, не обновляющие данные, преобладают, но, как правило, чаще всего так и бывает.

MySQL, начиная с версии 3.23.15, поддерживает односторонний внутренний механизм репликации. Один сервер действует как головной, а другие - как подчиненные. Обратите внимание: один сервер может играть роль головного в одной паре и подчиненного - в другой. Головной сервер содержит двоичный журнал обновлений и индексный файл двоичных журналов для протоколирования ротации двоичных журналов. Подчиненный сервер при соединении уведомляет головной о том, в каком состоянии он находится, начиная от последнего обновления, которое было успешно опубликовано на подчиненный сервер. После этого подчиненный сервер принимает обновления, а затем блокируется и ждет, пока головной сервер не сообщит о новых обновлениях.

Обратите внимание: при реплицировании базы данных все обновления этой базы данных должны производиться через головной сервер.

Еще одно преимущество использования механизма репликации заключается в том, что можно иметь "живую" резервную копию системы, выполняя резервное копирование не на головном, а на подчиненном сервере.

Репликация в MySQL основывается на том, что все изменения базы данных (обновления, удаления и т.д.) протоколируются в двоичном журнале на сервере, а подчиненный сервер читает сохраненные запросы из двоичного журнала головного сервера и выполняет эти запросы на своей копии данных.

Очень важно понимать, что двоичный журнал - это просто запись, начатая с фиксированного момента времени (с момента, когда вы включаете ведение записей в двоичном журнале). При установке каждого из подчиненных серверов нужно будет скопировать с головного сервера все данные, существовавшие на нем к моменту начала ведения записей в двоичном журнале. Если подчиненный сервер будет запущен с данными, не соответствующими тем, которые содержались на головном сервере **к моменту запуска двоичного журнала**, на подчиненном сервере может произойти сбой.

Начиная с версии 4.0.0, для записи данных на подчиненный сервер можно использовать команду LOAD DATA FROM MASTER. Обратите внимание: подчиненные серверы версии 4.0.0 не могут связываться с

головными серверами версии 3.23, но подчиненные серверы версии 4.0.1 и более поздних - могут. Подчиненный сервер версии 3.23 не может общаться с головным сервером версии 4.0.

Учтите, что команда LOAD DATA FROM MASTER в настоящее время работает только если все таблицы на головном сервере имеют тип MyISAM, и для них будет установлена глобальная блокировка чтения, чтобы не допустить никаких записей во время передачи таблиц от головного сервера к подчиненному. Данное ограничение носит временный характер. Оно обусловлено тем, что мы еще не реализовали горячее резервное копирование таблиц без блокировок. Это ограничение мы снимем для следующих ветвей версии 4.0 - как только будет реализовано горячее резервное копирование, которое позволит команде LOAD DATA FROM MASTER работать без

блокирования обновлений на головном сервере.

Из-за вышеупомянутого ограничения рекомендуется использовать команду LOAD DATA FROM MASTER только в тех случаях, если набор данных на головном сервере относительно невелик или если для головного сервера допустима длительная блокировка чтения. Скорость выполнения команды LOAD DATA FROM MASTER для разных систем может быть различной, поэтому для грубой оценки времени выполнения команды можно считать, что для передачи 1 Мб данных требуется 1 секунда. Это приблизительно соответствует случаю, когда и головной, и подчиненный серверы эквивалентны Pentium с тактовой частотой 700 МГц и связаны сетью с пропускной способностью 100 Мбит/с, а размер индексного файла равен примерно половине размера файла данных. Разумеется, такая прикидка дает лишь грубую приближенную оценку и в случае каждой конкретной системы потребуются свои допущения.

После того как подчиненный сервер будут правильно сконфигурирован и запущен, он должен легко соединиться с головным сервером и ожидать обработки обновлений. Если головной сервер завершит работу или подчиненный сервер потеряет связь с головным, подчиненный сервер будет пытаться установить соединение каждый раз по истечении интервала времени, указанного в опции master-connect-retry (в секундах) до тех пор,

пока не установится подсоединение и не продолжится прослушивание обновлений.

Каждый подчиненный сервер отслеживает события с момента разрыва. Головной сервер не имеет никакой информации о том, сколько существует подчиненных серверов, и какие из них обновлены последними данными в любой момент времени.

Ниже приводится список поддерживаемых и не поддерживаемых при репликации функций:

Реплицирование будет выполнено правильно при использовании значений AUTO_INCREMENT, LAST_INSERT_ID() иTIMESTAMP.

Если в обновлениях присутствует функция RAND(), реплицирование будет выполнено некорректно. При реплицировании обновлений с функцией

RAND() применяйте RAND(some_non_rand_expr). В качестве аргумента (some_non_rand_expr - некоторое не случайное выражение) для функции RAND() можно, например, использовать функцию UNIX_TIMESTAMP().

На головном и подчиненном серверах следует использовать одинаковый набор символов (--default-character-set). В противном случае могут возникать ошибки дублирующихся ключей на подчиненном сервере, поскольку ключ, который считается уникальным на головном сервере, может не быть таковым при использовании другого набора символов.

В MySQL 3.23 команда LOAD DATA INFILE будет выполнена корректно, если файл во время выполнения обновления будет находиться на головном сервере. Команда LOAD LOCAL DATA INFILE будет проигнорирована. В MySQL 4.0 это ограничение не присутствует - все разновидности команды LOAD DATA INFILE реплицируются правильно.

Запросы на обновление, в которых используются пользовательские переменные, являются не безопасными для репликации (пока).

Команды FLUSH не записываются в двоичный журнал и поэтому не копируются на подчиненный сервер. Проблем при этом не возникает, поскольку команды FLUSH ничего не изменяют. Однако это означает, что при непосредственном, без использования оператора GRANT, обновлении таблиц привилегий MySQL и при последующем реплицировании базы данных привилегий mysql нужно выполнить команду FLUSH

PRIVILEGES на подчиненных серверах, чтобы новые привилегии вступили в силу.

Временные таблицы, начиная с версии 3.23.29, реплицируются корректно, за исключением случая, когда при прекращении работы подчиненного сервера (не только потока подчиненного сервера) некоторые временные таблицы остаются открытыми и используются в последующих обновлениях. Для решения этой проблемы перед прекращением работы подчиненного сервера выполните команду SLAVE STOP, проверьте, чтобы переменнаяSlave_open_temp_tables содержала значение 0, затем выполните mysqladmin shutdown. Если значение переменнойSlave_open_temp_tables не 0, перезапустите поток подчиненного сервера при помощи команды SLAVE START и проверьте, не улучшилась ли ситуация теперь. Эта проблема будет решаться более изящно, но придется подождать MySQL 4.0. В более ранних версиях при использовании временных таблиц репликации не выполняются должным образом - в таких случаях мы рекомендуем либо обновить версию MySQL, либо перед выполнением запросов, использующих временные таблицы, выполнить команду SET SQL_LOG_BIN=0 на своих клиентах.

MySQL поддерживает лишь один головной и много подчиненных серверов. В 4.x будет добавлен алгоритм голосования, обеспечивающий автоматическое изменение головного сервера, если что-либо будет выполняться неправильно при текущем головном сервере. Будут также введены процессы 'агента', которые помогут выполнять распределение

нагрузки путем посылки запросов на выборки различным подчиненным серверам.

Начиная с версии 3.23.26, стало безопасно соединять серверы циклическими соединениями головной-подчиненный с включенной опцией log-slave-updates. Однако обратите внимание: при таком способе установки многие запросы не будут выполняться корректно, если только в коде вашего клиента не предусмотрена обработка потенциальных проблем, которые могут случаться при обновлениях, происходящих в различной последовательности на различных серверах. Это означает, что если вы сделаете установку следующим образом:

A -> B -> C -> A, то такая установка будет работать только в том случае, если выполняются непротиворечивые обновления между таблицами. Другими словами, при вставке данных на серверах A и C нельзя вставлять на сервере A строку, которая может иметь ключ, противоречащий строке, вставляемой на сервере C. Также нельзя обновлять одинаковые строки на двух серверах, если имеет значение порядок обновлений. Обратите внимание: в версии 3.23.26 изменился формат журнала. Таким образом, если версия подчиненного сервера меньше 3.23.26, сервер не сможет считывать записи из журнала.

Если запрос на подчиненном сервере вызывает ошибку, поток подчиненного сервера завершится, и в файле '.err'появится соответствующее сообщение. После этого нужно будет вручную установить соединение с подчиненным сервером, исправить причину ошибки (например, обращение к несуществующей таблице) и затем выполнять SQL-команду SLAVE START (доступна в версии 3.23.16). При использовании версии 3.23.15 потребуется перезапустить сервер.

Если соединение с головным сервером прервется, подчиненный сервер попытается сразу же восстановить его, и затем в случае неудачи будет повторять попытки через установленное в опции master-connectretry количество секунд (по умолчанию 60). По этой причине безопасно выключить головной сервер и после этого перезапустить его через некоторое время. Подчиненный сервер будет также разрешать проблемы, возникающие при аварийных отключениях электричества в узлах сети.

Завершение работы подчиненного сервера (корректное) также является безопасным, поскольку при этом отслеживаются события, начиная от момента остановки сервера. Но в случае некорректного отключения сервера могут возникать проблемы, особенно, если дисковый кэш не был синхронизирован перед "смертью" системы. Для того чтобы значительно повысить эффективность своей системы обеспечения отказоустойчивости, целесообразно приобрести хороший UPS (источник бесперебойного питания).

Если головной сервер слушает нестандартный порт, это нужно будет указать также в параметре master-port в файле 'my.cnf'.

В версии 3.23.15 все таблицы и базы данных могут быть реплицированы. Начиная с версии 3.23.16, появилась возможность ограничить репликацию набором баз данных при помощи директив replicatedo-db в файле 'my.cnf';

можно также исключить набор баз данных из репликации при помощи директив replicate-ignore-db. Обратите внимание,: в версиях MySQL до 3.23.23, имелась ошибка, из-за которой команда LOAD DATA INFILE выполнялась некорректно, если она применялась к базе данных, исключенной из репликации.

Начиная с версии 3.23.16, команда SET SQL_LOG_BIN = 0 будет выключать ведение записей о репликации в журналах (двоичных) на головном сервере, а команда SET SQL_LOG_BIN = 1 - включать такое ведение записей. Для выполнения этих команд нужно иметь привилегию SUPER (в MySQL 4.0.2 и выше) или PROCESS (в более ранних версиях MySQL).

Начиная с версии 3.23.19 можно убрать мусор, оставшийся после неоконченной репликации (если ее выполнение пошло не должным образом), и начать все сначала, используя команды FLUSH MASTER и FLUSH SLAVE.

В версии 3.23.26 эти команды переименованы в RESET MASTER и RESET SLAVE соответственно - чтобы сделать понятным их назначение. Тем не менее, старые варианты FLUSH все еще работают - для обеспечения совместимости.

Начиная с версии 3.23.23 можно заменять головные серверы и корректировать точку положения в журнале репликации при помощи команды CHANGE MASTER TO.

Начиная с версии 3.23.23 можно при помощи опции binlog-ignoredb уведомлять головной сервер о том, что обновления в некоторых базах данных не должны отражаться в двоичном журнале.

Начиная с версии 3.23.26, можно использовать опцию replicate-rewritedb для уведомления подчиненного сервера о том, что он должен применить обновления базы данных на головном сервере к базе данных с другим именем на подчиненном сервере.

Начиная с версии 3.23.28 можно использовать команду PURGE MASTER LOGS TO 'имя-журнала', чтобы избавиться от старых журналов без завершения работы подчиненного сервера.

Из-за того, что по своей природе таблицы MyISAM являются нетранзакционными, может случиться так, что запрос обновит таблицу только частично и возвратит код ошибки. Это может произойти, например, при вставке нескольких строк, одна из которых нарушает ограничение ключа, или в случае, когда длинный запрос обновления "убивается" после обновления некоторых строк. Если такое случится на головном сервере, то поток подчиненного сервера завершит работу и будет ждать, пока администратор базы данных не примет решение о том, что делать в этом случае (если только код ошибки не является легитимным и в результате выполнения запроса не будет сгенерирована ошибка с тем же кодом). Если такой способ проверки правильности кода ошибки нежелателен, начиная с версии 3.23.47, некоторые (или все) ошибки могут быть замаскированы при помощи опции slave-skip-errors.

Отдельныетаблицымогутисключатьсяизрепликацииприпомощиопции r

eplicate-do-table/replicate-ignore-tab илиопции replicate-wild-do-table/replicate-

wild-ignore-table. Однако в настоящее время наличие определенных конструктивных неточностей в некоторых довольно редких случаях может приводить к неожиданным результатам. Протокол репликации явно не уведомляет подчиненный сервер о том, какие таблицы должны быть изменены запросом, поэтому подчиненному серверу требуется анализировать запрос, чтобы узнать это. Чтобы избежать лишнего синтаксического анализа, для которого требуется прерывать выполнение запросов, исключение таблицы в настоящее время реализуется путем посылки запроса к стандартному анализатору MySQL для упрощенного синтаксического анализа. Если анализатор обнаружит, что таблица должна игнорироваться, выполнение запроса будет остановлено и выдано сообщение об успехе. Этот подход несколько неэффективен, при его применении чаще возникают ошибки и, кроме того, имеются две известные ошибки в версии 3.23.49. Первая может возникнуть из-за того, что поскольку анализатор автоматически открывает таблицу при анализе некоторых запросов, игнорируемая таблица должна существовать на подчиненном сервере. Другая ошибка заключается в том, что при частичном обновлении игнорируемой таблицы поток подчиненного сервера не заметит, что таблица должна игнорироваться, и приостановит процесс репликации. Несмотря на то что вышеупомянутые ошибки концептуально очень просто исправить, для этого придется изменить достаточно много кода, что поставит под угрозу состояние стабильности ветви 3.23. Если описанные случаи непосредственно имеют отношение к вашему приложению (а это довольно редкий случай) - используйте опцию slave-skip-errors, чтобы дать указание серверу продолжать репликации, игнорируя эти ошибки.

Методика и порядок выполнения работы

1. Запускаем головной сервер и подключаемся к нему.Для подключения используем программуDevart dbForge Studio for MySQL, а не интерфейс phpMyAdmin. Этапы выполнения работы показаны на рисунках

16.1-16-23.

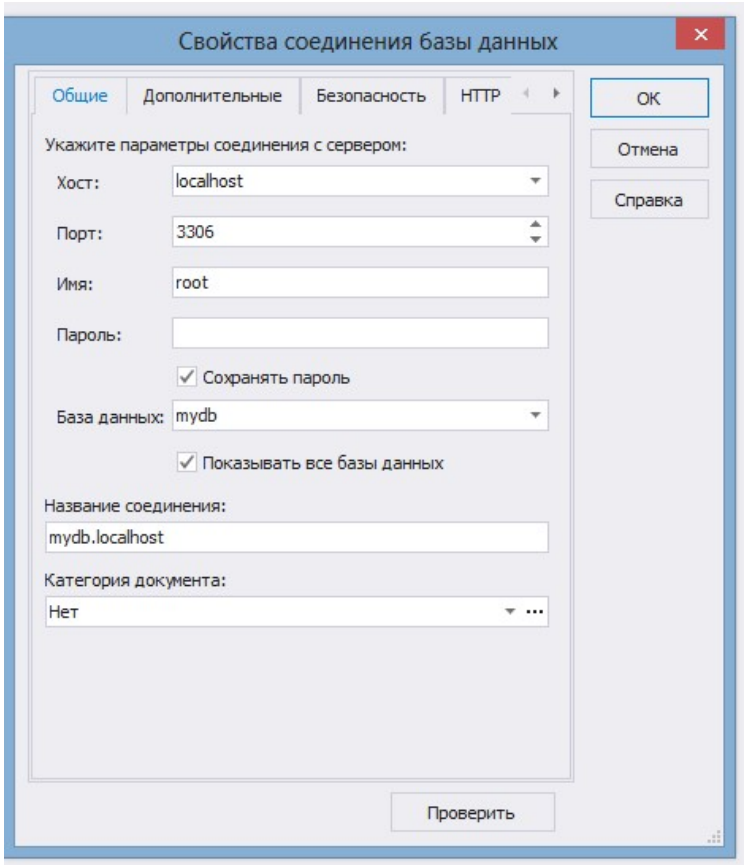


Рисунок 16.1 – Этап 1

2. Создаем тестовую базу данных «mydb».

3. Создаем в базе данных тестовую таблицу «mytable» с двумя полями – числовым (уникальным) и текстовым. Добавляем новую строку с текстом, например, «OldData», рисунок 16.2.

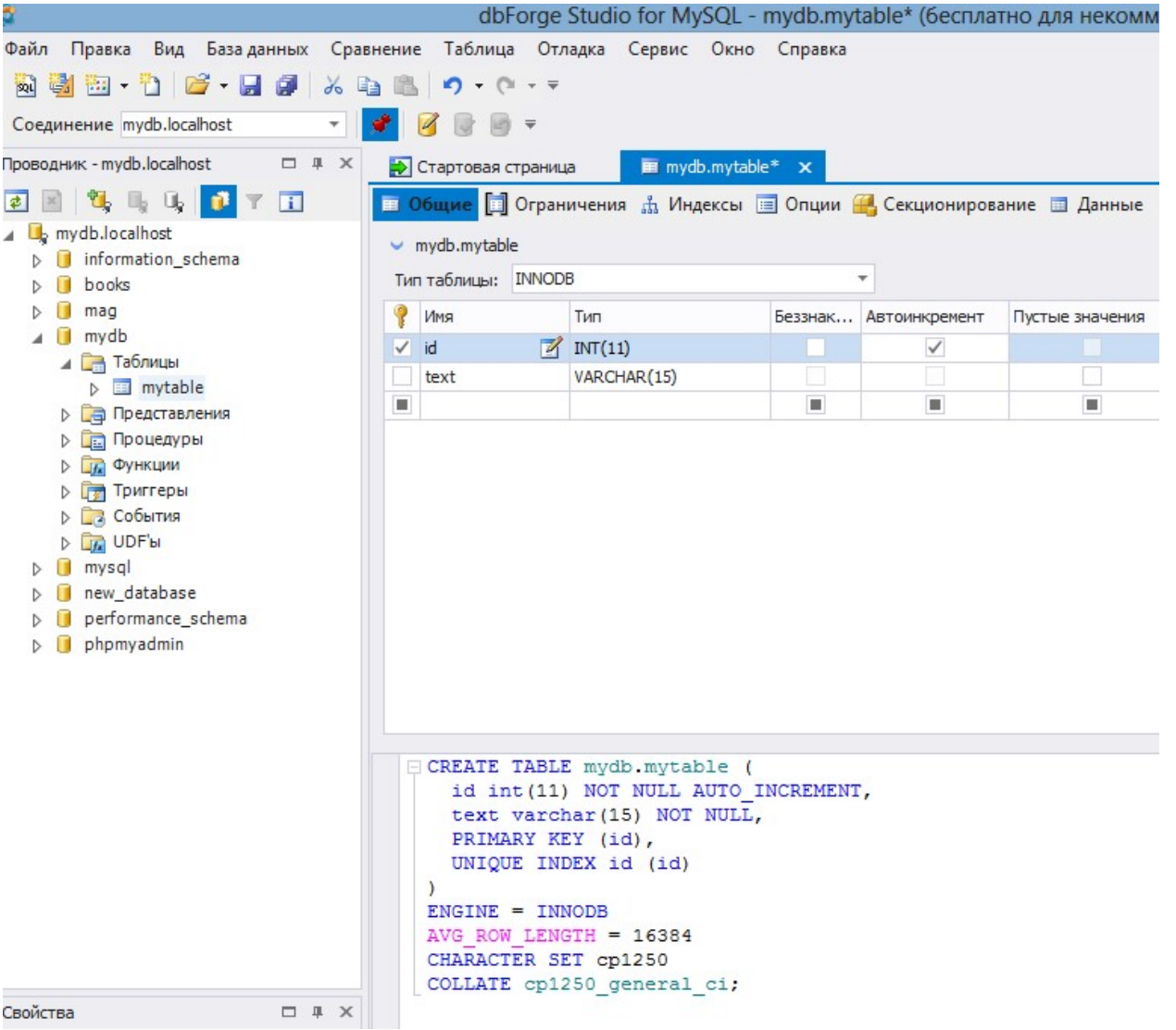


Рисунок 16.2 – Этап 3

4. Делаем резервную копию БД, рисунок 16.3. Не забываем поставить галочку «Включать выражение CREATEDATABASE».

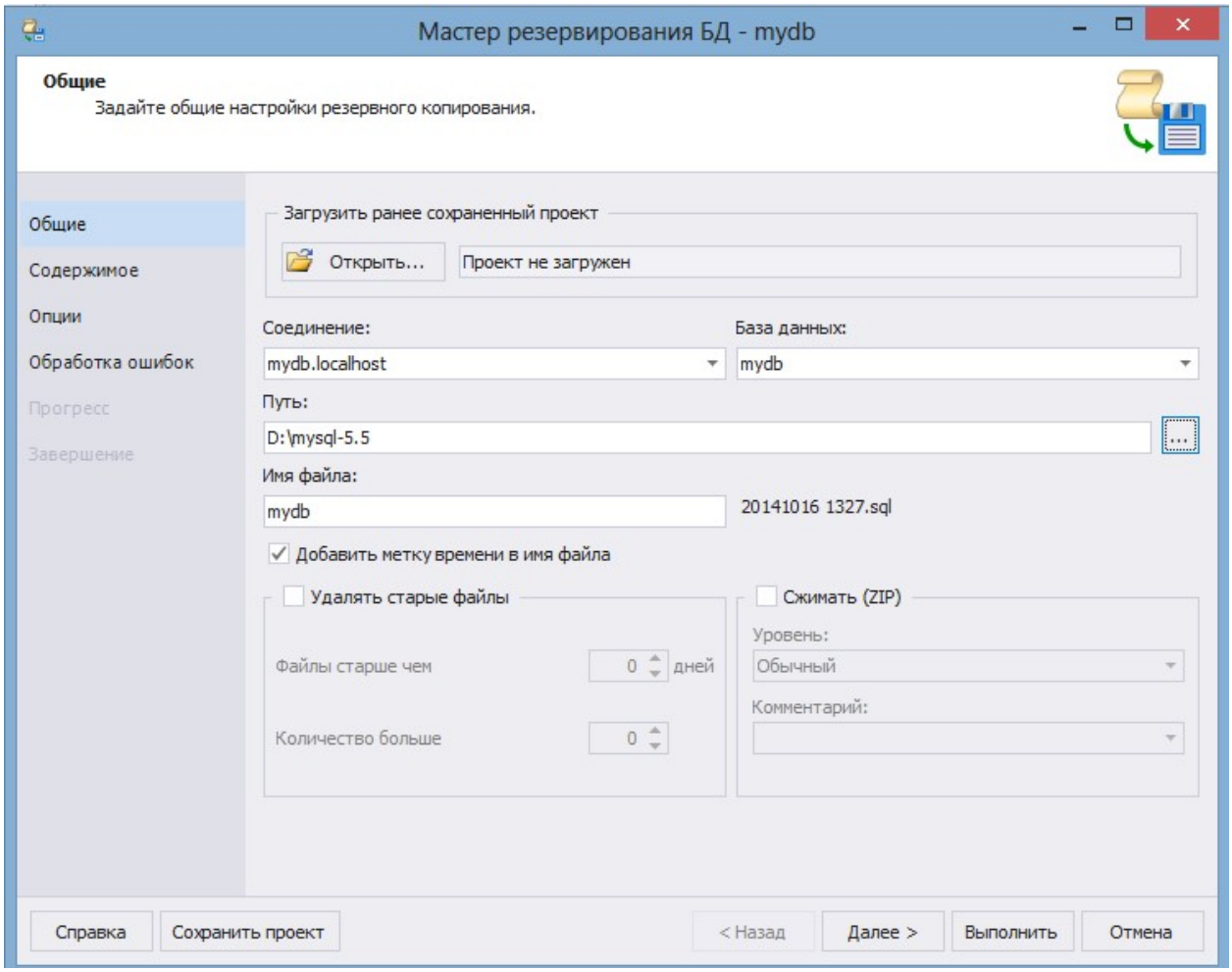


Рисунок 16.3 – Этап 4

5. Создаем пользователя для репликации «RepUser» с паролем «reppass»(рисунок 16.4) и даем ему привилегию Replication Slave, рисунок 16.5.

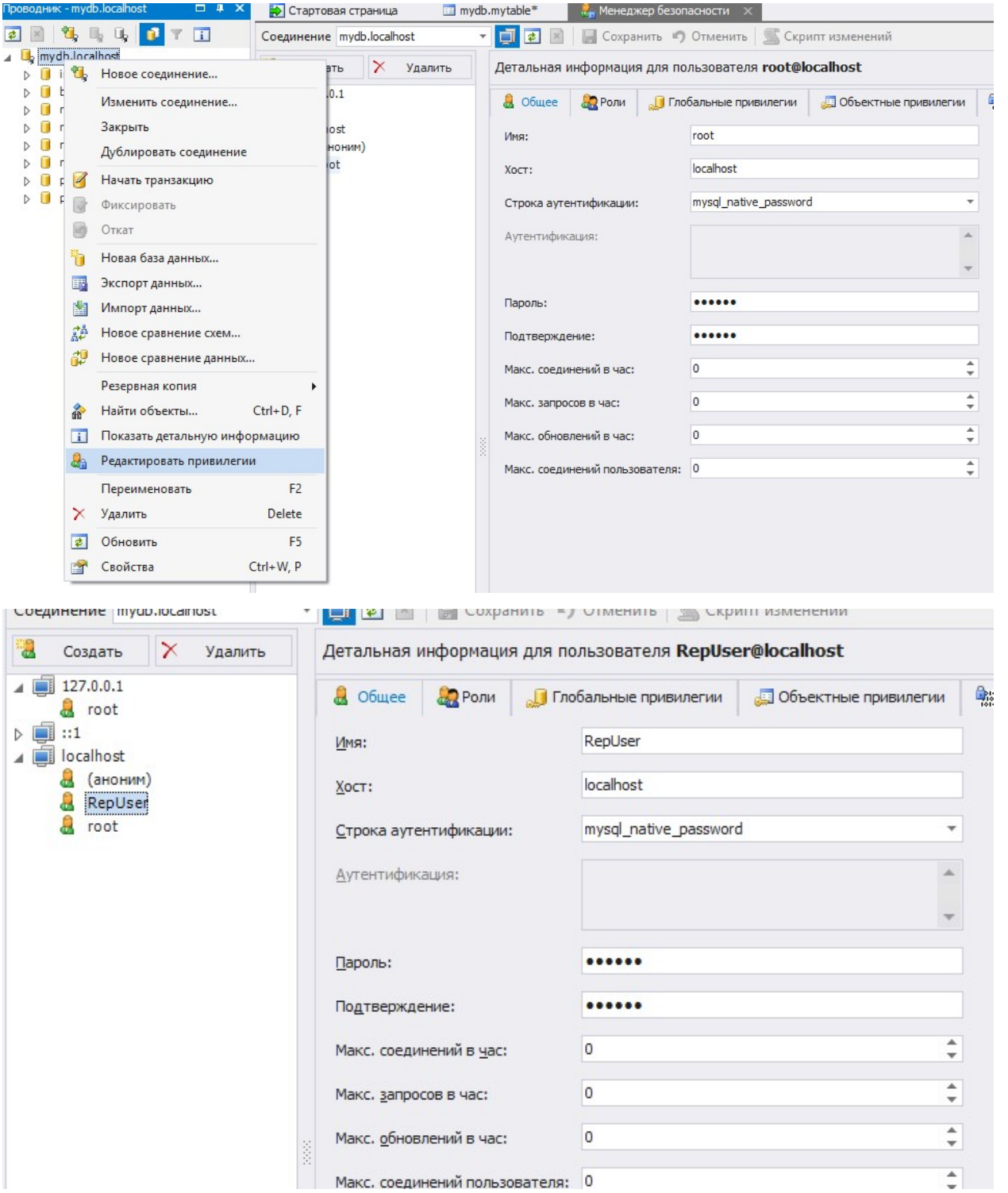


Рисунок 16.4 – Создание пользователя с паролем

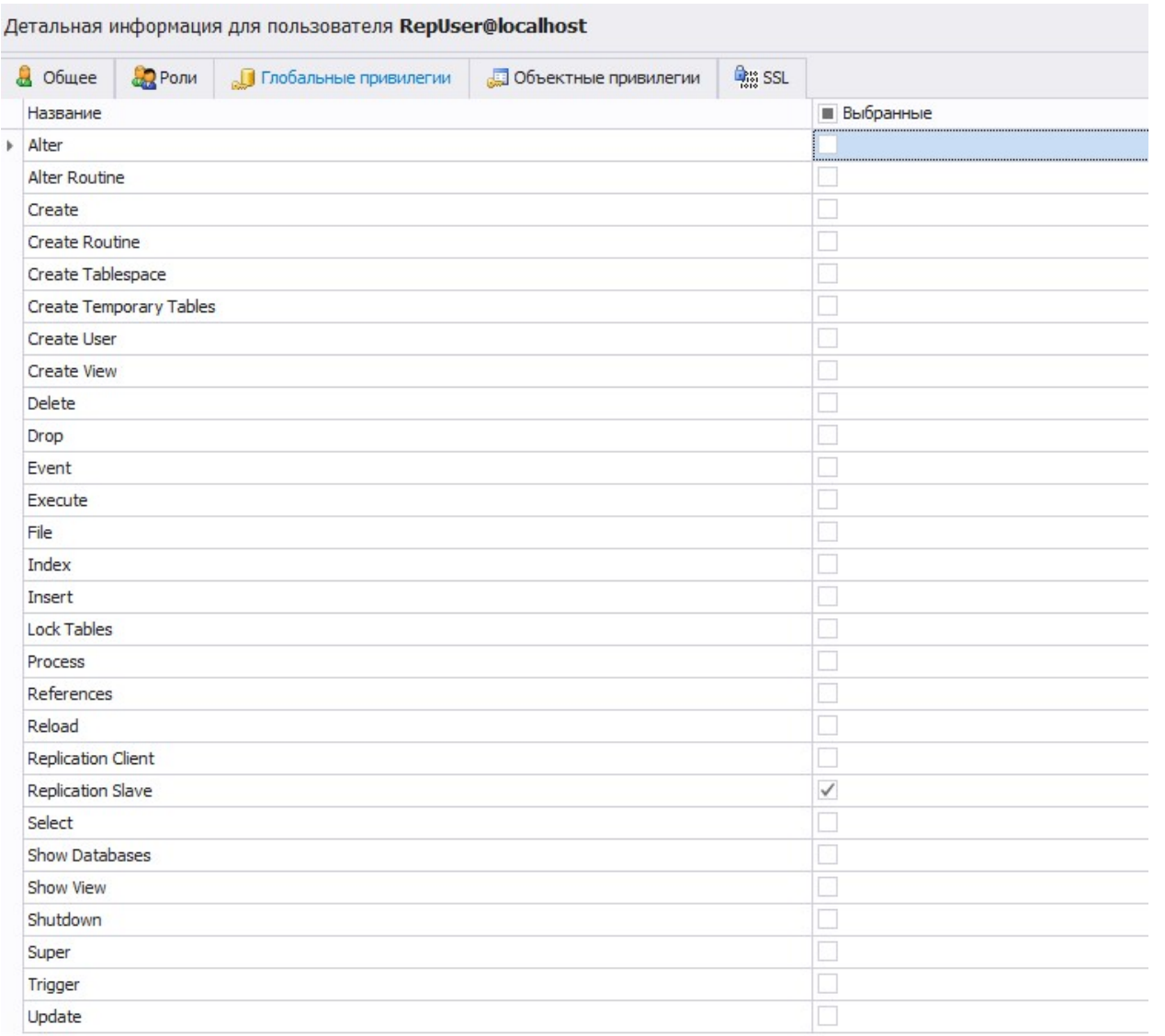


Рисунок 16.5 – Этап 5

6. Останавливаем головной сервер.

7. Запускаем подчиненный сервер и подключаемся к нему. Запускаем файл StartMySQL, рисунок 16.6.

Поскольку Мы пока не меняли настройки порта подчиненного сервера, то подключаться можно с теми же параметрами что и к главному. То есть к порту 3306.

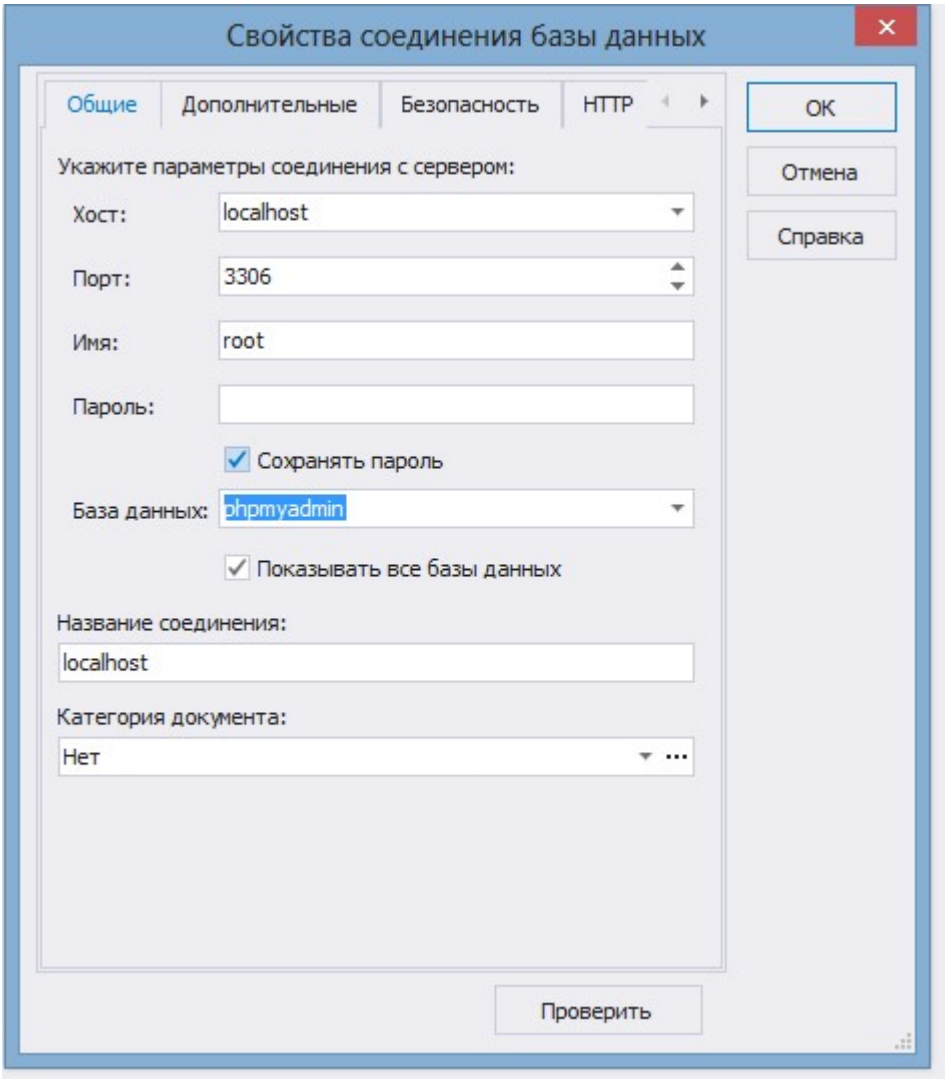


Рисунок 16.6 – Этап 7

8. Восстанавливаем базу данных «mydb» из архива, рисунок 16.7.

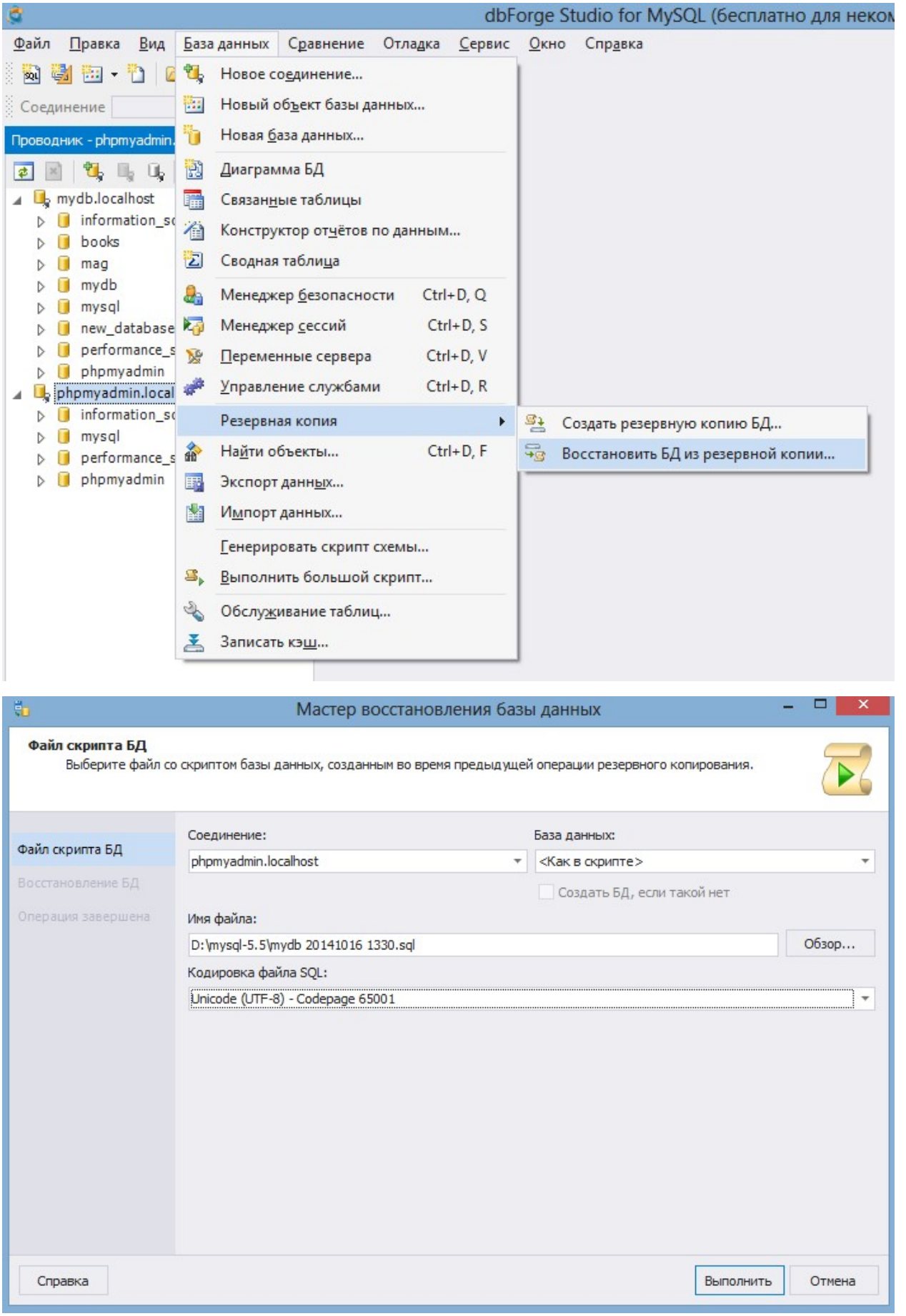


Рисунок 16.7 – Этап 8

9. Останавливаем подчиненный сервер (StopMySQL).

10. Настраиваем головной сервер.

Строка server-id = 1 уже присутствует в конфигурационном файле. А строку log-bin=mysql-bin надо раскомментировать. Файл «my.ini» раздел *[mysqld]*

server-id = 1 log-bin=mysql-bin

С этого момента головной сервер будет записывать все изменения в базах данных, создание новых баз данных, создание пользователей.

11. Настраиваем подчиненный сервер. Изменяем порт на 3307, например, в двух местах и изменяем идентификатор сервера на 2, например.

Файл «my.ini»

[client] port = 3307

[mysqld]

port = 3307

server-id = 2

12. Запускаем головной сервер

13. Запускаем подчиненный сервер

14. Запускаем командную строку и переходим в папку bin подчиненного сервера

15. Запускаем из командной строки утилиту mysql.exe с параметрами порта (3307), хоста (127.0.0.1), и пользователя (root). Таким образом, мы подключимся к подчиненному серверу, рисунок 16.8. *mysql.exe --host=127.0.0.1 --port=3307 --user=root*

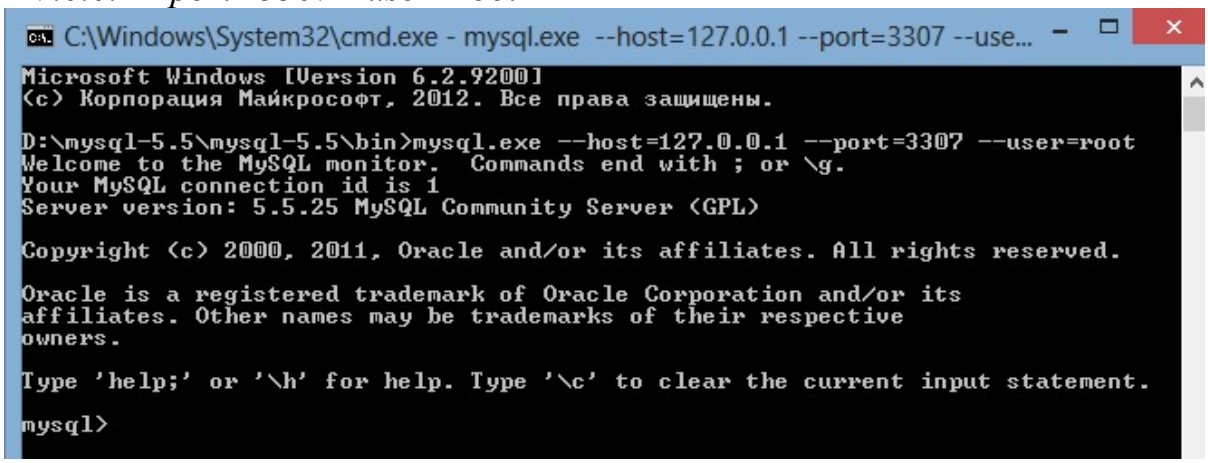


Рисунок 16.8 – Этап 15

16. Настраиваем репликацию. Нам нужно прописать имя головного сервера, порт, пользователя и пароль Для этого настраиваем параметры MASTER -MASTER_HOST, MASTER_PORT, MASTER_USER и MASTER_PASSWORD. Для этого вводим команду:

CHANGE MASTER TO MASTER_HOST='127.0.0.1',
MASTER_PORT=3306, MASTER_USER='RepUser',
MASTER_PASSWORD='reppass';

Команду не обязательно вводить одной строкой. Командный интерпретатор определяет окончание ввода по символу «;», рисунок 16.9.

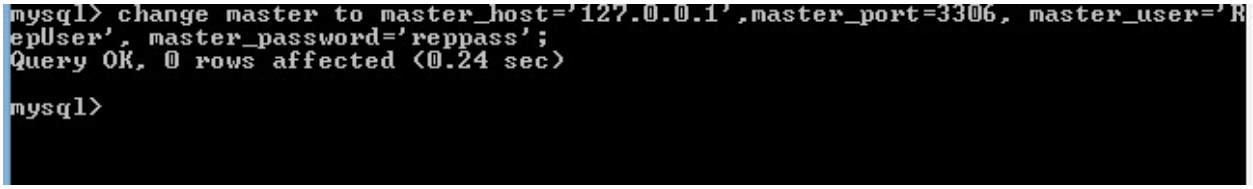


Рисунок 16.9 – Этап 16

17. Запускаем репликацию командой
START SLAVE;
18. Проверяем состояние репликации (рисунок 16.10)
SHOW SLAVE STATUS \G

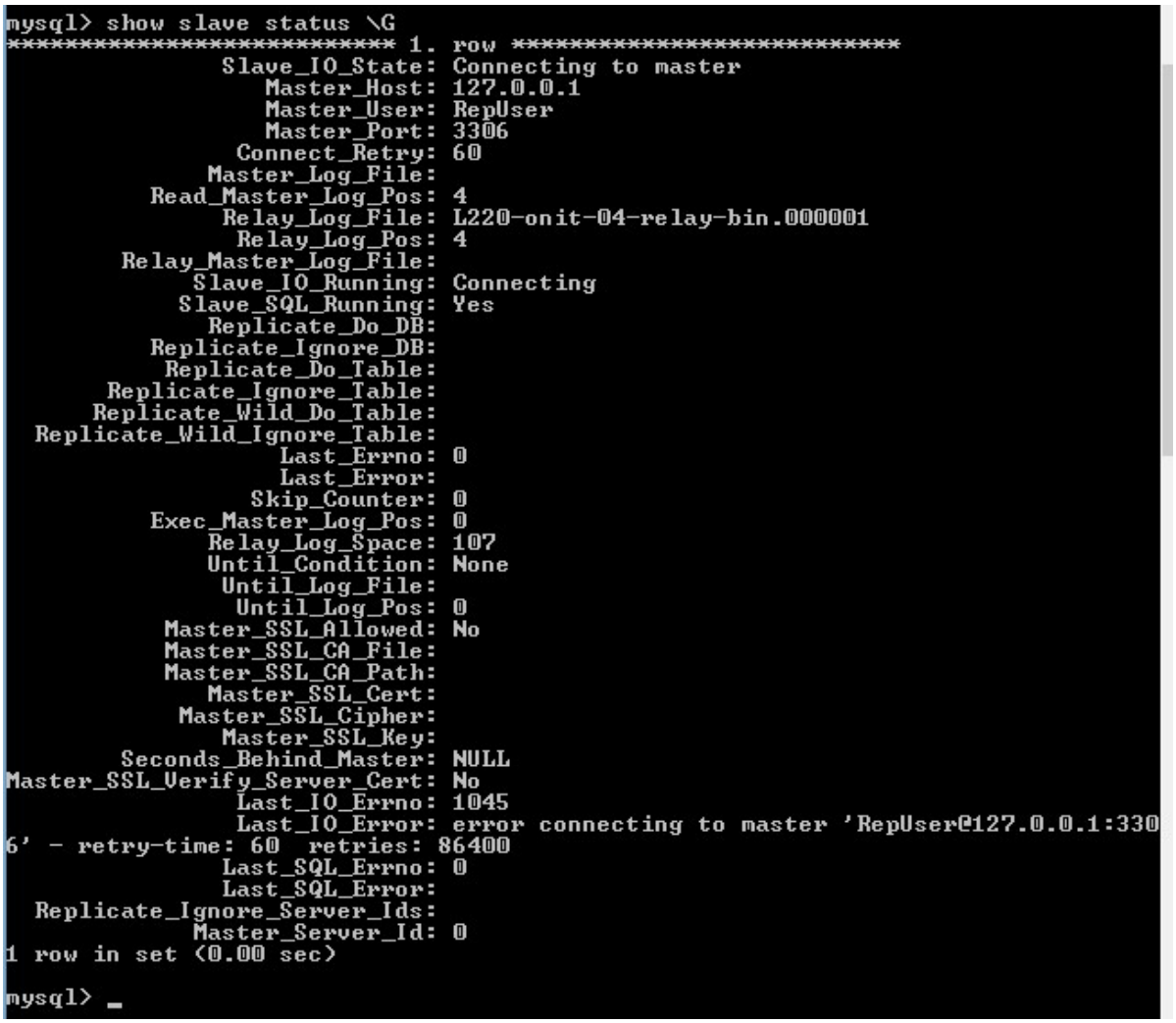


Рисунок 16.10 – Этап 18

19. Смотрим список баз данных, рисунок 16.11.
SHOW DATABASES;

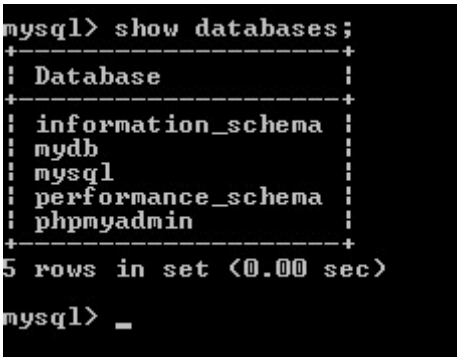


Рисунок 16.11 – Этап 19

20. Не закрывая консоль подключаемся к головному серверу и создаем в головном сервере новую базу данных «newdb»

21. Добавляем в старую базу данных «mydb» в таблицу «mytable» новую строку, например «NewData», рисунок 16.12.

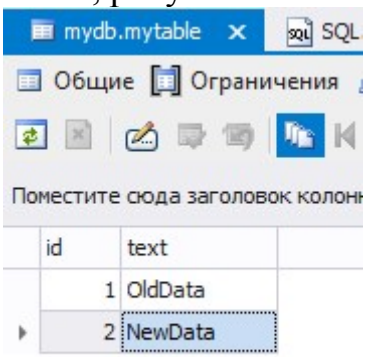


Рисунок 16.12 – Этап 21

22. В консоли, которую мы не закрыли, проверяем наличие новой базы данных. Для этого смотрим список баз данных, рисунок 16.13.

SHOW DATABASES;

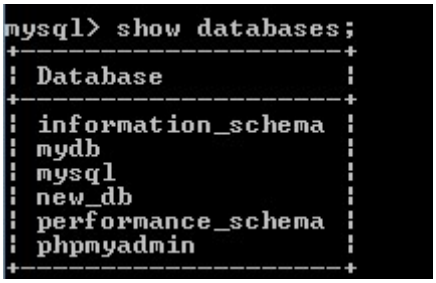


Рисунок 16.13 – Этап 22

Как мы видим новая база данных «newdb» появилась в списке 23. Проверяем наличие новой строки в старой базе данных «mydb».

Сначала подключаемся к базе, рисунок 16.14.

USEmydb;

Рисунок 16.14 – Этап 23, подключение к БД

Выводим список строк, рисунок 16.15:

*SELECT * FROMmytable*

Рисунок 16.15 – Этап 23

Как мы видим, на подчиненном сервере в таблице появилась новая строка.

24. Выход из командного интерпретатора mysql.exe осуществляется командой \q

\q

25. Выход из командного интерпретатора mysql.exe осуществляется командой exit

Exit

26. Оба сервера можно остановить. Теперь их можно запускать и останавливать в произвольном порядке. Головной сервер при любых изменениях в базе данных будет записывать все изменения в специальный файл, а подчиненный сервер при удачном подключении к главному выполнит все эти изменения у себя.

Обратить внимание. Для функционирования схемы необходимо не только скопировать базы данных на новый сервер, но и скопировать пользователей с их привилегиями. Вновь созданные пользователи будут копироваться на подчиненный сервер автоматически. Например, пользователь RepUser, которого мы создали ДО включения логгирования изменений на главном сервере, на подчиненном сервере отсутствует.

Задание

Организовать репликацию Вашей базы данных, разработанной в предыдущих лабораторных работах. Проверить ее работоспособность.

Содержание отчета и его форма

Отчет по лабораторной работе должен состоять из:

1) названия лабораторной работы;
2) описание примеров репликации.

Вопросы для защиты работы:

1. Каким образом осуществляется репликация?

Лабораторная работа №17.SQL-команды, относящиеся к репликации

Цель работы: научиться управлять репликациями вMySQL.

Теоретическое обоснование

Управление репликацией производится командами SQL. Ниже приводится краткое описание команд:

Команда	Описание
SLAVE START	Запускает поток подчиненного сервера (подчиненный сервер)
SLAVE STOP	Завершает поток подчиненного сервера (подчиненный сервер)
SET SQL_LOG_BIN=0	Блокирует ведение записей в журналах обновлений, если пользователь имеет привилегию SUPER. В противном случае ничего не выполняет (головной сервер)
SET SQL_LOG_BIN=1	Отменяет блокировку ведения записей в журналах обновлений, если пользователь имеет привилегиюSUPER. В противном случае ничего не выполняет (головной сервер)
SET SQL_SLAVE_SKIP_COUNTER=n	Пропускает последующие n событий на головном сервере. Опция допустима, если поток подчиненного сервера не запущен, в противном случае будет выдана ошибка. Полезна для восстановления после сбоев репликации.
RESET MASTER	Удаляет все двоичные журналы, перечисленные в индексном файле, и делает индексный файл двоичных журналов пустым. Для версий ниже 3.23.26 используйте команду FLUSH SLAVE (головной сервер)
RESET SLAVE	Заставляет подчиненный сервер "забыть" свою точку положения репликации в журналах головного сервера. Для версий ниже 3.23.26 эта команда называется FLUSH SLAVE (подчиненный сервер)

LOAD TABLE tblname FROM MASTER	Загружает копию таблицы из головного на подчиненный сервер. Используется главным образом для отладки команды LOAD DATA FROM MASTER, но некоторые "пользователи-

	гурманы" могут найти ей и другие применения. Если вы относите себя к числу обычных, не отягощенных хакерскими амбициями пользователей, данную опцию применять не стоит (подчиненный сервер).
LOAD DATA FROM MASTER	Присутствует в версиях выше 4.0.0. Создает образ головного сервера и копирует его на подчиненный сервер. Обновляет значения MASTER_LOG_FILE и MASTER_LOG_POS таким образом, чтобы подчиненный сервер начинал репликацию из конкретной позиции. Будет обрабатывать ограничения таблиц и баз данных, указанные в опциях replicate-*. При этом, пока происходит создание образа, могут использоваться лишь таблицы MyISAM и требуется глобальная блокировка чтения на головном сервере. В будущем планируется обеспечить работу этой команды с таблицами InnoDB и устранить необходимость глобальной блокировки чтения при помощи интерактивного резервного копирования, не требующего блокировки.
CHANGE MASTER TO master_def_list	Заменяетпараметрыголовногосерверазначениями, заданнымивсписке master_def_list, иперезапускаетпотокподчиненногосервера. master_def_list –этосписоксразделителем-запятой, содержащийзначения master_def, где master_def – одноизследующихзначений: MASTER_HOST, MASTER_USER, MASTER_PASSWORD, MASTER_PORT, MASTER_CONNECT_RETRY, MASTER_LOG_FILE, MASTER_LOG_POS. Например:

	CHANGE MASTER TO MASTER_HOST='master2.mycompany.com', MASTER_USER='replication', MASTER_PASSWORD='bigs3cret', MASTER_PORT=3306, MASTER_LOG_FILE='master2-bin.001', MASTER_LOG_POS=4;
	Следует указывать только те значения, которые подлежат изменению. Не указанные значения останутся неизменными, за исключением тех случаев, когда изменяется хост или порт. В этом

	случае подчиненный сервер считает, что поскольку изменяется хост или порт, головной сервер становится другим. Следовательно, старые значения и точки положения в журнале будут автоматически заменены на значение пустой строки и 0 соответственно (начальные значения). Обратите внимание: если подчиненный сервер перезапускается, он сохраняет "память" о своем последнем головном сервере. Если это нежелательно, можно перед перезапуском удалить файл 'master.info' - тогда подчиненный сервер будет считывать информацию о своем головном сервере из файла 'my.cnf' или из командной строки. Эта команда используется для настройки подчиненного сервера при наличии образа головного сервера, а также записей из журнала и сдвига головного сервера, которые соответствуют образу. Можновыполнитькоманду
	CHANGE MASTER TO MASTER_LOG_FILE='log_name_on_master', MASTER_LOG_POS=log_offset_on_master
	на подчиненном сервере после восстановления образа (подчиненный сервер)
SHOW MASTER STATUS	Выводит информацию о состоянии головного сервера, исходя из информации в двоичных журналах (головной сервер)

SHOW SLAVE HOSTS	Присутствует в версии 4.0.0 и выше. Выводит список подчиненных серверов, связанных в текущее время с головным сервером (подчиненный сервер)
SHOW SLAVE STATUS	Выводит информацию о состоянии существенных параметров потока подчиненного сервера (головной сервер)
SHOW MASTER LOGS	Присутствует только начиная с версии 3.23.28. Выводит список двоичных журналов головного сервера. Эту команду следует использовать перед вызовом команды PURGE MASTER LOGS TO для определения того, какие из журналов можно удалить (головной сервер)
SHOW BINLOG EVENTS [IN 'logname'] [FROM pos] [LIMIT [offset,] rows	

] @tab Показывает события в двоичном журнале обновлений. Преимущественно применяется для тестирования/отладки, но может также использоваться и для обычных клиентов, по какой-либо причине нуждающихся в чтении содержимого двоичных журналов (головной сервер).	

SHOW NEW MASTER FOR SLAVE WITH MASTER_LOG_FILE='logfile' AND MASTER_LOG_POS=pos AND MASTER_LOG_SEQ=log_seq AND MASTER_SERVER_ID=server_id	Эта команда используется, когда подчиненному серверу, связанному с головным сервером, который, возможно, является "мертвым" или недоступным, нужно отключить репликации на другом подчиненном сервере, связанном с тем же головным сервером. Команда возвратит пересчитанные координаты репликации, и вывод этой команды может использоваться в последующей командеCHANGE MASTER TO. Обычным пользователям данная команда, как правило, никогда не понадобится: она главным образом служит для внутреннего использования в отказобезопасном репликационном коде. В будущем возможны изменения синтаксиса опции, если будет найден более интуитивно понятный способ описания этой операции.
PURGE MASTER LOGS TO 'logname'	Присутствует начиная с версии 3.23.28. Удаляет все журналы репликации, которые перечислены в индексном файле журналов до передаваемого журнала, и удаляет их из индексного файла журналов. Таким образом передаваемый журнал становится первым в индексном файле журналов. Пример:
	PURGE MASTER LOGS TO 'mysql-bin.010'
	Эта команда не выполнит никаких действий и возвратит ошибку, если имеется активный подчиненный сервер, который в текущее время читает данные из одного из журналов, который должен быть удален. Однако если имеется бездействующий подчиненный сервер и

	происходит удаление одного из журналов, который он хочет прочитать, то после того, как подчиненный сервер "поднимется", он станет неспособным к репликации. Команда может быть безопасно выполнена на подчиненных серверах во время процесса репликации - не нужно останавливать процесс. Сначала необходимо проверить все подчиненные серверы при помощи команды SHOW SLAVE STATUS, чтобы увидеть, какой журнал используется, затем вывести список журналов головного сервера при помощи команды SHOW MASTER LOGS, найти самый ранний журнал среди всех подчиненных серверов (если все подчиненные серверы получили последние обновления, это будет последний журнал в списке), сделать резервные копии всех журналов, которые должны быть удалены (необязательно), и очистить все до целевого журнала.

Задание

Изучить назначение команд, относящихся к репликации БД.

Содержание отчета и его форма

Отчет по лабораторной работе должен состоять из:

1) названия лабораторной работы;
2) описание примеров использования команд репликации.

Вопросы для защиты работы:

1. Какие команды используются для репликации данных?

Лабораторная работа №18. Проблемы и распространённые ошибки MySQL

Цель работы: научиться работать с СУБДMySQL в режиме клиентсервер.

Теоретическое обоснование

Как определить, чем вызваны проблемы? При возникновении проблемы прежде всего следует обнаружить ее источник - программу или элемент оборудования.

Если присутствует один из следующих симптомов, то проблема, скорее всего, связана с аппаратным обеспечением (с памятью, материнской платой, процессором или жестким диском) либо с ядром:

- Не работает клавиатура. Обычно ее работоспособность можно проверить по реакции на нажатие Caps Lock. Если индикатор Caps Lock не меняется, то клавиатуру необходимо заменить (прежде чем это сделать, следует попробовать перезагрузить компьютер и проверить все кабели, подключенные к клавиатуре).
- Не перемещается курсор мыши.
- Машина не отвечает на ping-запросы удаленной машины.
- Различные не связанные между собой программы не работают, как надо.
- Система неожиданно перезагрузилась (дефектная программа пользовательского уровня **никогда** не должна быть способна вызвать отказ системы).

В этом случае необходимо начать с проверки всех кабелей и запуска диагностических средств для проверки аппаратуры.

Следует также проверить, нет ли патчей, обновлений, сервисных пакетов (service pack) для используемой операционной системы, при помощи которых вы, возможно, могли бы решить проблемы. Кроме того, следует удостовериться, что у вас установлены достаточно свежие версии библиотек (таких как glibc). Для раннего обнаружения проблем хорошо использовать машину с ECC-памятью.

В случае блокировки клавиатуры положение можно исправить, если войти на свою машину с другой машины и выполнить на своей машине kbd_mode -a.

Исследуйте свой системный журнальный файл ('/var/log/messages' или т.п.) на предмет причин возникающих проблем. Если есть основания полагать, что проблема - в MySQL, то следует также изучить журнальные файлы MySQL.

Если вы считаете, что аппаратные проблемы отсутствуют, то следует попробовать обнаружить вызывающую проблемы программу. Попробуйте с помощью top, ps, taskmanager или подобной программы проверить, какая программа забирает все ресурсы процессора или блокирует машину.

Проверьте с помощью top, df или подобной программы, нет ли нехватки памяти, дискового пространства, дескрипторов для открытия файлов или каких-либо других критических ресурсов.

Если проблема связана с бесконтрольным процессом, то всегда можно попробовать уничтожить его. Если он не хочет уничтожаться, то, вероятно, существует ошибка в операционной системе.

Если после изучения всех возможных причин вы сделали вывод, что источником проблемы является именно MySQL-сервер или клиент, то следует

сделать отчет об ошибке для команды поддержки. В отчете об ошибке постарайтесь дать очень подробное описание поведения системы и свое мнение по поводу происходящего. Следует также объяснить, почему вы считаете, что проблемы вызывает именно MySQL. Примите во внимание все ситуации, описанные в данном разделе. Опишите все проблемы в точности так, как они наблюдаются при исследовании системы. При помещении в отчет для всего вывода программ и/или их сообщений об ошибках и/или подобной информации из журнальных файлов используйте метод "вырезать и вставить". При обращении к группе поддержки MySQL необходимо

детально описать, какая именно программа не работает, и какие симптомы вы наблюдали! Нам доводилось получать много отчетов об ошибках, где просто утверждалось, что "система не работает", - такие отчеты не давали никакой информации о характере возможной проблемы.

Если программа сбоит, то всегда полезно выяснить:

Не вызвала ли данная программа ошибки сегментации (core dump)?

Не забирает ли программа все ресурсы процессора? Проверьте с помощью top. Дайте программе немного поработать - возможно, она занимается сложными вычислениями.

Если проблемы вызваны именно сервером mysqld, то можно ли выполнить mysqladmin -u root ping или mysqladmin -u root processlist?

Что сообщает клиентская программа (попробуйте поработать, например, с mysql) при попытке соединиться с MySQL? Происходит ли заклинивание клиента? Выдает ли программа какой-нибудь вывод?

Коды ошибки представлены в таблице 18.1. Какой код из них вы получите, зависит от ОС.

Таблица 18.1 – Коды ошибок

Код ошибки	**Описание**
CR_SERVER_GONE_ERROR	Клиент не может послать запрос серверу.
CR_SERVER_LOST	Клиент не получил ошибки при передаче запроса серверу, но он не получил также полного ответа (или хоть какого-то ответа) на запрос.

Ошибка будет также выдана, если кто-нибудь уничтожит выполняющийся поток посредством kill номер потока.

Проверить, что MySQL на ходу, можно, запустив mysqladmin version и изучив время работы (uptime). Если проблема в аварийном завершении mysqld, то необходимо сосредоточиться на поиске причины аварии. В этом случае следует сначала проверить, не будет ли уничтожен MySQL снова при повторном задании запроса.

Эти ошибки будут также выдаваться при посылке серверу неверного или слишком длинного запроса. Если mysqldполучает неправильный или

слишком большой пакет, то сервер предполагает, что с клиентом что-то не так, и закрывает соединение. Если необходимо выполнять объемные запросы (например, при работе с большими столбцами типа BLOB), можно увеличить предельный размер запроса, запустив mysqld с опцией O max_allowed_packet=# (по умолчанию 1 Мб). Дополнительная память выделяется по требованию, так что mysqld будет использовать больше памяти только в случае, когда выдан большой запрос или когда mysqld должен возвратить большую строку результата.

Задание

Изучить распространённые ошибки и методы их решения.

Содержание отчета и его форма

Отчет по лабораторной работе должен состоять из:

1) названия лабораторной работы;
2) описание проблем и распространенных ошибок MySQL.

Вопросы для защиты работы:

1. Какие проблемы могут возникнуть при работе с MySQL?

Список рекомендуемой литературы

а) основная литература:

1. Агальцов В.П. Базы данных. В 2 книгах. Книга 2 Распределенные и удаленные базы данных. Учебник. М.:ИД ФОРУМ НИЦ Инфра-М, 2013. – 2013 (электронная библиотечная система znanium.com)
2. Справочное руководство по MySQL. [Электронный ресурс]: http://www.php.su/mysql/manual/
3. http://www.intuit.ru
4. http://www.citforum.ru
5. http://guardmag.com

б) дополнительная литература:

6. PHP, MySQL, XML: программирование для Интернета.

Автор: Бенкен Е.С. Издательство: БХВ-Петербург, 2007 г. 314 с.

7. Максимов, Е.М Базы данных в системах управления производственными процессами: учебное пособие [Текст] / Максимов Е.М., Бахтадзе Н.Н.: Издательство Московского государственного открытого университета, 2011 г. 160 с.
8. Туманов, В.Е. Проектирование хранилищ данных для систем бизнес-аналитики: учебное пособие [Текст] / Туманов В.Е.: ИнтернетУниверситет Информационных Технологий; БИНОМ. Лаборатория знаний,

2010 г.

9. Чекмарев, Ю.В. Локальные вычислительные сети: Учебное пособие [Текст] / Ю.В. Чомаев. ДМК Пресс, 2009 г.

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное образовательное учреждение высшего образования
«СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Пятигорский институт (филиал) СКФУ

Управление данными

Методические указания к выполнению курсовой работы

Направление подготовки 09.03.02 «Информационные системы и технологии»

Пятигорск, 2025

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

«История исследований систем баз данных – это, по сути, история развития приложений, достигших исключительной производительности и оказавших потрясающее воздействие на экономику. Достижения в исследованиях баз данных стали основой фундаментальных разработок коммуникационных систем, транспорта и логистики, финансового менеджмента, систем с базами знаний, а также большого количества гражданских и военных приложений. Они послужили фундаментом значительного прогресса в ведущих областях науки – от информатики до биологии». Эта цитата материалов международного семинара по системам баз данных отражает исключительную важность их применения, с одной стороны, как основы практически любой информационной системы, с другой стороны, как катализатора научных достижений не только в области программного обеспечения, но и во многих других областях науки.

Поэтому изучение дисциплины «Управление данными» бакалаврами направления 09.03.02 «Информационные системы и технологии» является одним из важнейших этапов их специальной подготовки, а также подготовки к последующему изучению курсов «Технологии обработки информации», «Корпоративные информационные системы», «Предметно-ориентированные информационные системы», «Методы и средства проектирования информационных систем и технологий», «Управление данными в сетях», а также при подготовке выпускной квалификационной работы.

Курсовое проектирование по дисциплине «Управление данными» осуществляется в течение всего семестра, в котором изучается данная дисциплина, и предполагает самостоятельное проектирование студентом базы данных (БД) в среде одной из промышленных систем управления базами данных (СУБД).

1. ЦЕЛЬ И ЗАДАЧИ

Курсовая работа является самостоятельной учебно-исследовательской работой студента и представляет собой логически завершенное и оформленное научное исследование. Цель курсового проектирования – формирование у студента навыков научно-исследовательской работы, повышение уровня его профессиональной (теоретической и практической) подготовки, углубление знаний по учебной дисциплине, развитие интереса и навыков самостоятельной работы с научной и справочной литературой.

Целью его выполнения курсовой работы является закрепление, углубление и обобщение теоретических знаний и практических навыков, выработка навыков самостоятельной работы по разработке баз данных и информационных систем. Для реализации этой цели тематикой и требованиями к выполнению курсовой работы предусматривается выполнение основных этапов проектирования и создания баз данных, включая формулировку цели и задач проектирования, изучение предметной области, моделирование данных, разработку средств обеспечения защиты данных от несанкционированного доступа, проектирование интерфейса и выходных форм.

Выполнение курсовой работы имеет целью расширение знаний студентов, обучение методам теоретического анализа явлений и закономерностей науки, отработку навыков самостоятельного применения теоретических знаний к комплексному решению профессиональных задач, использования справочной литературы, методов математической обработки экспериментальных данных, компьютерных технологий.

Задание к курсовой работы по дисциплине «Управление данными» содержит теоретическую часть и практическое задание. При выполнении теоретического задания студентом должны решаться следующие задачи:

- приобретение новых теоретических знаний в соответствии с темой работы и заданием руководителя;
- умение систематизировать, обобщать и логично излагать концепции, альтернативные точки зрения по исследуемой проблеме;
- развитие учебно-исследовательских и методических навыков, необходимых для системного научного анализа изучаемого явления; - совершенствование профессиональной подготовки.

При выполнении практического задания решаются задачи:

- приобретение практических навыков обследования предметной области, концептуального, логического и физического проектирования базы данных;
- освоение средств поддержания целостности БД; - изучение языка запросов SQL.

В процессе курсового проектирования у студентов формируются и закрепляются следующие умения и навыки.

Умения:

- анализировать информационную среду предметной области и устанавливать структурное представление и взаимосвязи с другими компонентами информационного пространства, информационные потоки, систематизировать документооборот; существующий рынок аппаратного и программного обеспечения, классифицировать существующие информационные системы, определять направления их развития;
- формулировать запросы к базам данных на естественном языке, с помощью абстрактных реляционных языков и на языке SQL;
- разрабатывать информационно-логическую, функциональную и объектно-ориентированную модели информационной системы, модели данных информационных систем.

Навыки:

- работы с программно-техническими средствами проектирования БД в интерактивном режиме;
- методами и средствами представления данных о предметной области.

2. ФОРМУЛИРОВКА ЗАДАНИЯ

Курсовая работа выполняется на общую тему: «Разработка приложения для управления данными в базе данных (для заданной предметной области) средствами Visual Studio»

Задание на курсовую работу состоит из двух частей: теоретической и практической.

Теоретическая часть

1. Анализ тенденций развития отдельных направлений управления данными.

Практическая часть

2.1. Разработка базы данных
2.2. Разработка приложения для управления данными
2.2.1.Добавление записей
2.2.2. Удаление записей
2.2.3Редактирование записей
2.3. Разработка экранных форм и отчетов

2. Разработка локальной реляционной базы данных.

2.1. Обследование предметной области.
2.2. Концептуальное проектирование.
2.2.1. Перечень сущностей (обосновать список).
2.2.2. Перечень атрибутов.
2.3. Инфологическое проектирование БД.
2.3.1. Модель “сущность-связь”.
2.3.2. Классификация связей.

2.4. Реляционная модель БД.
2.4.1. Функциональные зависимости между атрибутами.
2.4.2. Выбор ключей.
2.4.3. Нормализация отношений.
2.5. Даталогическое проектирование БД.
2.5.1. Состав таблиц БД.
2.5.2. Средства поддержания целостности.
2.6. Запросы к БД.
2.7. Разработка механизмов защиты данных от несанкционированного доступа.
2.8. Требования к техническому обеспечению.
2.9. Инструкция по использованию БД.
2.9.1. Вызов программы.
2.9.2. Экранные формы.
2.9.3. Описание отчетов.

2.1. Тематика теоретических заданий на курсовой проект

Теоретическое задание состоит в анализе тенденций развития отдельных направлений управления данными.

Примерные темы вариантов для выполнения теоретического задания

1. Стандартизация языка SQL. Стандарт SQL/MM (Multimedia).
2. Темпоральные базы данных.
3. Стандартизация языка SQL. SQL/XML
4. Дедуктивные базы данных.
5. Стандартизация языка SQL. SQL/JRT (Java Routines and Types).
6. Обзор Российского рынка СУБД.
7. Разработка новых архитектур СУБД.
8. Средства управления доступом.
9. Администрирование БД.
10. Гипертекстовые БД.
11. Объектно-ориентированные БД.
12. Коммерческие БД.
13. Средства управления транзакциями.
14. Распределенная обработка данных.
15. Параллельная обработка данных.
16. Сжатие информации в БД.
17. Фрактальные методы сжатия информации.
18. Файловые и бесфайловые методы организации данных.
19. Тенденции развития методов организации данных.
20. Оптимизация запросов.
21. Тенденции построения файловых систем.
22. Странично-сегментная организации данных.
23. OLTP-технология.
24. OLAP-технология.
25. Методы анализа данных в фактографических системах.

26. Управление данными в системах Data Grids.
27. Облачные вычисления (Cloud computing).
28. NoSQL СУБД.
29. Перспективы развития СУБД.
30. Арендная модель инфраструктурных платформ.
31. Проблемы стандартизации СУБД.
34. СУБД ведущих фирм-производителей.
35. Языковые средства управления данными.
36. Глубинный анализ данных.
37. Архитектура хранилищ данных.
38. Методы распределенной обработки данных.
39. Быстродействие и включение измерения времени в СУБД (In memory DB).
40. Эволюция технологий управления данными.

2.2 Предметные области для выполнения практической части курсового проекта

Практическая часть курсовой работы представляет собой разработку реляционной базы данных для заданной предметной области, а также приложения для управления данными в этой БД. Примеры предметных областей и атрибуты их характеризующие приведены ниже.

Вариант 1 БД Ресторана

Таблицы:	Сотрудники (Код сотрудника, ФИО, Возраст, Пол, Адрес, Телефон, Паспортные данные, Код должности)[10 записей]. Должности (Код должности, Наименование должности, Оклад, Обязанности, Требования)[5 записей]. Склад (Код ингредиента, Наименование ингредиента, Дата выпуска, Объём, Срок годности, Стоимость, Поставщик)[10 записей]. Меню (Код блюда, Наименование блюда, Код ингредиента 1, Объём ингредиента 1, Код ингредиента 2, Объём ингредиента 2, Код ингредиента 3, Объём ингредиента 3, Стоимость, Время приготовления)[10 записей]. Заказ (Дата, Время, ФИО заказчика, Телефон, Код блюда 1, Код блюда 2, Код блюда 3, Стоимость, Отметка о выполнении, Код сотрудника)[10 записей].
Запросы:	Отдел кадров (Связывает таблицы "Сотрудники" и "Должности" по полю "Код должности"). Меню (Связывает таблицы "Меню" и "Склад" по полям "Код ингредиента", "Код ингредиента 1", "Код ингредиента 2" и "Код ингредиента 3"). Заказ (Связывает таблицы "Заказ", "Меню" и "Сотрудники" по полям "Код блюда", "Код блюда 1", "Код блюда 2", "Код блюда 3" и "Код сотрудника").

Вариант 2. БД Банка

Таблицы:	Сотрудники (Код сотрудника, ФИО, Возраст, Пол, Адрес, Телефон, Паспортные данные, Код должности)[10 записей]. Должности (Код должности, Наименование должности, Оклад, Обязанности, Требования)[5 записей].

	Вклады (Код вклада, Наименование вклада, Минимальный срок вклада, Минимальная сумма вклада, Код валюты, Процентная ставка, Дополнительные условия)[5 записей]. Валюта (Код валюты, Наименование, Обменный курс)[3 записи]. Вкладчики (ФИО вкладчика, Адрес, Телефон, Паспортные данные, Дата вклада, Дата возврата, Код вклада, Сумма вклада, Сумма возврата, Отметка о возврате вклада, Код сотрудника)[10 записей].
Запросы:	Отдел кадров (Связывает таблицы "Сотрудники" и "Должности" по полю "Код должности"). Вклады (Связывает таблицы "Вклады" и "Валюта" по полю "Код валюты"). Вкладчики (Связывает таблицы "Вкладчики", "Вклады" и "Сотрудники" по полям "Код вклада" и "Код сотрудника").

Вариант 3. БД Больницы.

Таблицы:	Сотрудники (Код сотрудника, ФИО, Возраст, Пол, Адрес, Телефон, Паспортные данные, Код должности)[10 записей]. Должности (Код должности, Наименование должности, Оклад, Обязанности, Требования)[5 записей]. Лекарства (Код лекарства, Наименование, Показания, Противопоказания, Упаковка, Стоимость)[5 записей]. Болезни (Код болезни, Наименование, Симптомы, Продолжительность, Последствия, Код лекарства 1, Код лекарства 2, Код лекарства 3)[10 записей]. Пациенты (ФИО пациента, Возраст, Пол, Адрес, Телефон, Дата обращения, Код болезни, Код сотрудника, Результат лечения)[10 записей].
Запросы:	Отдел кадров (Связывает таблицы "Сотрудники" и "Должности" по полю "Код должности"). Болезни (Связывает таблицы "Болезни" и "Лекарства" по полю "Код лекарства", "Код лекарства 1", "Код лекарства 2" и "Код лекарства 3"). Пациенты (Связывает таблицы "Пациенты", "Болезни" и "Сотрудники" по полям "Код болезни" и "Код сотрудника").

Вариант 4. БД Гостиницы.

Таблицы:	Сотрудники (Код сотрудника, ФИО, Возраст, Пол, Адрес, Телефон, Паспортные данные, Код должности)[10 записей]. Должности (Код должности, Наименование должности, Оклад, Обязанности, Требования)[5 записей]. Номера (Код номера, Наименование, Вместимость, Описание, Стоимость, Код сотрудника)[5 записей]. Услуги (Код услуги, Наименование, Описание, Стоимость)[5 записей]. Клиенты (ФИО, Паспортные данные, Дата заселения, Дата выезда, Код номера, Код услуги 1, Код услуги 2, Код услуги 3, Стоимость, Код сотрудника)
Запросы:	Отдел кадров (Связывает таблицы "Сотрудники" и "Должности" по полю "Код должности"). Номера (Связывает таблицы "Сотрудники" и "Номера" по полю "Код сотрудника"). Клиенты (Связывает таблицы "Клиенты", "Номера", "Услуги" и "Сотрудники" по полям "Код номера", "Код услуги", "Код услуги 1", "Код

	услуги 2", "Код услуги 3" и "Код сотрудника").

Вариант 5. БД МВД.

Таблицы:	Сотрудники (Код сотрудника, ФИО, Возраст, Пол, Адрес, Телефон, Паспортные данные, Код должности, Код звания)[10 записей]. Должности (Код должности, Наименование должности, Оклад, Обязанности, Требования)[5 записей]. Звания (Код звания, Наименование, Надбавка, Обязанности, Требования)[5 записей]. Виды преступлений (Код вида преступления, Наименование, Статья, Наказание, Срок)[5 записей]. Преступники (Номер дела, ФИО, Дата рождения, Пол, Адрес, Код вида преступления, Код пострадавшего, Состояние, Код сотрудника)[10 записей]. Пострадавшие (Код пострадавшего, ФИО, Дата рождения, Пол, Адрес)[5 записей].
Запросы:	Отдел кадров (Связывает таблицы "Сотрудники", "Должности" и "Звания" по полям "Код должности" и "Код звания"). Преступники (Связывает таблицы "Преступники", "Виды преступлений", "Пострадавшие" и "Сотрудники" по полям "Код вида преступления", "Код пострадавшего" и "Код сотрудника").

Вариант 6. БД Аэропорта.

Таблицы:	Сотрудники (Код сотрудника, ФИО, Возраст, Пол, Адрес, Телефон, Паспортные данные, Код должности)[10 записей]. Должности (Код должности, Наименование должности, Оклад, Обязанности, Требования)[5 записей]. Самолёты (Код самолёта, Марка, Вместимость, Грузоподъемность, Код типа, Технические характеристики, Дата выпуска, Налётано часов, Дата последнего ремонта, Код сотрудника)[5 записей]. Типы самолётов (Код типа, Наименование, Назначение, Ограничения). Экипажи (Код экипажа, Налётано часов, Код сотрудника 1, Код сотрудника 2, Код сотрудника 3)[5 записей]. Рейсы (Код рейса, Дата, Время, Откуда, Куда, Код экипажа, Код самолёта, Время полёта)[5 записей]. Билеты (ФИО пассажира, Паспортные данные, Место, Код рейса, Цена)
Запросы:	Отдел кадров (Связывает таблицы "Сотрудники" и "Должности" по полю "Код должности"). Самолёты (Связывает таблицы "Самолёты", "Типы самолётов" и "Сотрудники" по полям "Код типа" и "Код сотрудника") Экипажи (Связывает таблицы "Экипажи" и "Сотрудники" по полям "Код сотрудника" "Код сотрудника 1", "Код сотрудника 2" и "Код сотрудника 3") Рейсы (Связывает таблицы "Рейсы", "Самолёты" и "Экипажи" по полям "Код экипажа" и "Код самолёта") Билеты (Связывает таблицы "Билеты" и "Рейсы" по полю "Код рейса")

Вариант 7 БД Видео проката.

Таблицы:	Сотрудники (Код сотрудника, ФИО, Возраст, Пол, Адрес, Телефон, Паспортные данные, Код должности)[10 записей]. Должности (Код должности, Наименование должности, Оклад, Обязанности, Требования)[5 записей].

	Жанры (Код жанра, Наименование жанра, Описание)[5 записей]. Кассеты (Код кассеты, Наименование фильма, Год создание, Производитель, Страна, Главный актёр, Дата записи, Код жанра, Цена)[10 записей]. Клиенты (ФИО, Адрес, Телефон, Паспортные данные, Дата взятия, Дата возврата, Отметка об оплате, Отметка о возврате, Код кассеты 1, Код кассеты 2, Код кассеты 3, Код сотрудника)[10 записей].
Запросы:	Отдел кадров (Связывает таблицы "Сотрудники" и "Должности" по полю "Код должности"). Кассеты (Связывает таблицы "Кассеты" и "Жанры" по полю "Код жанра"). Кассеты на руках (Связывает таблицы "Клиенты", "Кассеты" и "Сотрудники" по полям "Код кассеты", "Код кассеты 1", "Код кассеты 2", "Код кассеты 3" и "Код сотрудника").

Вариант 8. БД Библиотеки.

Таблицы:	Сотрудники (Код сотрудника, ФИО, Возраст, Пол, Адрес, Телефон, Паспортные данные, Код должности)[10 записей]. Должности (Код должности, Наименование должности, Оклад, Обязанности, Требования)[5 записей]. Издательства (Код издательства, Наименование, Город, Адрес)[5 записей]. Жанры (Код жанра, Наименование, Описание) [5 записей]. Книги (Код книги, Наименование, Автор, Код издательства, Год издания, Код жанра) [10 записей]. Читатели (Код читателя, ФИО, Дата рождения, Пол, Адрес, Телефон, Паспортные данные) [10 записей]. Выданные книги (Код книги, Код читателя, Дата выдачи, Дата возврата, Отметка о возврате, Код сотрудника) [10 записей].
Запросы:	Отдел кадров (Связывает таблицы "Сотрудники" и "Должности" по полю "Код должности"). Каталог (Связывает таблицы "Книги", "Издательства" и "Жанры" по полям "Код издательства" и "Код жанра"). Книги на руках (Связывает таблицы "Выданные книги", "Книги", "Читатели" и "Сотрудники" по полям "Код книги", "Код читателя" и "Код сотрудника")

Вариант 9. БД Радиостанции.

Таблицы:	Сотрудники (Код сотрудника, ФИО, Возраст, Пол, Адрес, Телефон, Паспортные данные, Код должности)[10 записей]. Должности (Код должности, Наименование должности, Оклад, Обязанности, Требования)[5 записей]. Исполнители (Код исполнителя, Наименование, Описание)[5 записей]. Жанры (Код жанра, Наименование, Описание)[5 записей]. Записи (Код записи, Наименование, Код исполнителя, Альбом, Год, Код жанра, Дата записи, Длительность, Рейтинг)[10 записей]. График работы (Дата, Код сотрудника, Время 1, Код записи 1, Время 2, Код записи 2, Время 3, Код записи 3)[10 записей].
Запросы:	Отдел кадров (Связывает таблицы "Сотрудники" и "Должности" по полю "Код должности"). Музыкальный архив (Связывает таблицы "Записи", "Исполнители" и "Жанры" по полям "Код исполнителя" и "Код жанра"). Сетка вещания (Связывает таблицы "График работы", "Сотрудники" и

	"Записи" по полям "Код сотрудника", "Код записи", "Код записи 1", "Код записи 2" и "Код записи 3").

Вариант 10. БД Таксопарка.

Таблицы:	Сотрудники (Код сотрудника, ФИО, Возраст, Пол, Адрес, Телефон, Паспортные данные, Код должности)[10 записей]. Должности (Код должности, Наименование должности, Оклад, Обязанности, Требования)[5 записей]. Марки (Код марки, Наименование, Технические характеристики, Стоимость, Специфика)[5 записей]. Тарифы (Код тарифа, Наименование, Описание, Стоимость)[5 записей]. Дополнительные услуги (Код услуги, Наименование, Описание услуги, Стоимость)[5 записей]. Автомобили (Код автомобиля, Код марки, Регистрационный номер, Номер кузова, Номер двигателя, Год выпуска, Пробег, Код сотрудника-шофёра, Дата последнего ТО, Код сотрудника-механика, Специальные отметки)[10 записей]. Вызовы (Дата, Время, Телефон, Откуда, Куда, Код тарифа, Код услуги, Код автомобиля, Код сотрудника-оператора)[10 записей].
Запросы:	Отдел кадров (Связывает таблицы "Сотрудники" и "Должности" по полю "Код должности"). Автопарк (Связывает таблицы "Автомобили", "Марки" и "Сотрудники" по полю "Код марки" и "Код сотрудника"). Список вызовов (Связывает таблицы "Вызовы", "Тарифы", "Услуги", "Автомобили" и "Сотрудники" по полю "Код тарифа", "Код услуги", "Код автомобиля" и "Код сотрудника-диспетчера").

Вариант 11. БД Туристического агентства.

Таблицы:	Сотрудники (Код сотрудника, ФИО, Возраст, Пол, Адрес, Телефон, Паспортные данные, Код должности)[10 записей]. Должности (Код должности, Наименование должности, Оклад, Обязанности, Требования)[5 записей]. Виды отдыха (Код вида, Наименование, Описание, Ограничения)[5 записей]. Отели (Код отеля, Наименование, Страна, Город, Адрес, Телефон, Количество звёзд, Контактное лицо)[10 записей]. Дополнительные услуги (Код услуги, Наименование, Описание, Цена) [5 записей]. Клиенты (Код клиента, ФИО, Дата рождения, Пол, Адрес, Телефон, Паспортные данные)[5 записей]. Путёвки (Дата начала, Дата окончания, Продолжительность, Код отеля, Код вида, Код услуги 1, Код услуги 2, Код услуги 3, Код клиента, Код сотрудника, Отметка о бронировании, Отметка об оплате)[10 записей].
Запросы:	Отдел кадров (Связывает таблицы "Сотрудники" и "Должности" по полю "Код должности"). Список путёвок (Связывает таблицы "Путёвки", "Отели", "Виды отдыха", "Дополнительные услуги", "Клиенты" и "Сотрудники" по полям "Код отеля", "Код вида", "Код услуги", "Код услуги 1", "Код услуги 2", "Код услуги 3", "Код клиента" и "Код сотрудника").

Вариант 12. БД Страховой компании.

Таблицы:	Сотрудники (Код сотрудника, ФИО, Возраст, Пол, Адрес, Телефон,

	Паспортные данные, Код должности)[10 записей]. Должности (Код должности, Наименование должности, Оклад, Обязанности, Требования)[5 записей]. Риски (Код риска, Наименование, Описание, Средняя вероятность)[5 записей]. Виды полисов (Код вида полиса, Наименование, Описание, Условия, Код риска 1, Код риска 2, Код риска 3)[5 записей]. Группы клиентов (Код группы, Наименование, Описание)[5 записей]. Клиенты (Код клиента, ФИО, Дата рождения, Пол, Адрес, Телефон, Паспортные данные, Код группы)[10 записей]. Полисы (Номер полиса, Дата начала, Дата окончания, Стоимость, Сумма выплаты, Код вида полиса, Отметка о выплате, Отметка об окончании, Код клиента, Код сотрудника)[10 записей].
Запросы:	Отдел кадров (Связывает таблицы "Сотрудники" и "Должности" по полю "Код должности"). Риски полисов (Связывает таблицы "Виды полисов" и "Риски" по полям "Код риска", "Код риска 1", "Код риска 2", "Код риска 3"). Список клиентов (Связывает таблицы "Клиенты" и "Группы клиентов" по полю "Код группы"). Список полисов (Связывает таблицы "Полисы", "Виды полисов", "Клиенты" и "Сотрудники" по полям "Код вида полиса", "Код клиента" и "Код сотрудника").

Вариант 13. БД Брачного агентства.

Таблицы:	Сотрудники (Код сотрудника, ФИО, Возраст, Пол, Адрес, Телефон, Паспортные данные, Код должности)[10 записей]. Должности (Код должности, Наименование должности, Оклад, Обязанности, Требования)[5 записей]. Знаки зодиака (Код знака, Наименование, Описание)[5 записей]. Отношения (Код отношения, Наименование, Описание)[5 записей]. Национальности (Код национальности, Наименование, Замечания)[5 записей]. Дополнительные услуги (Код услуги, Наименование, Описание, Цена)[5 записей]. Клиенты (Код клиента, ФИО, Пол, Дата рождения, Возраст, Рост, Вес, Количество детей, Семейное положение, Вредные привычки, Хобби, Описание, Код знака, Код отношения, Код национальности, Адрес, Телефон, Паспортные данные, Информация о партнёре)[10 записей]. Услуги (Код клиента, Дата, Код услуги 1, Код услуги 2, Код услуги 3, Стоимость, Код сотрудника)[10 записей].
Запросы:	Отдел кадров (Связывает таблицы "Сотрудники" и "Должности" по полю "Код должности"). Список клиентов (Связывает таблицы "Клиенты", "Знаки зодиака", "Отношения" и "Национальности" по полям "Код знака", "Код отношения" и "Код национальности"). Список услуг (Связывает таблицы "Услуги", "Клиенты", "Дополнительные услуги" и "Сотрудники" по полям "Код клиента", "Код услуги", "Код услуги 1", "Код услуги 2", "Код услуги 3" и "Код сотрудника").

Вариант 14. БД Сервис центра.

Таблицы:	Сотрудники (Код сотрудника, ФИО, Возраст, Пол, Адрес, Телефон,

	Паспортные данные, Код должности)[10 записей]. Должности (Код должности, Наименование должности, Оклад, Обязанности, Требования)[5 записей]. Запчасти (Код запчасти, Наименование, Функции, Цена)[5 записей]. Ремонтируемые модели (Код модели, Наименование, Тип, Производитель, Технические характеристики, Особенности)[5 записей]. Виды неисправностей (Код вида, Код модели, Описание, Симптомы, Методы ремонта, Код запчасти 1, Код запчасти 2, Код запчасти 3, Цена работы)[5 записей]. Обслуживаемые магазины (Код магазина, Наименование, Адрес, Телефон)[5 записей]. Заказы (Дата заказа, Дата возврата, ФИО заказчика, Серийный номер, Код вида неисправности, Код магазина, Отметка о гарантии, Срок гарантии ремонта, Цена, Код сотрудника)[10 записей].
Запросы:	Отдел кадров (Связывает таблицы "Сотрудники" и "Должности" по полю "Код должности"). Список неисправностей (Связывает таблицы "Виды неисправностей", "Ремонтируемые модели" и "Запчасти" по полям "Код модели", "Код запчасти", "Код запчасти 1", "Код запчасти 2", "Код запчасти 3"). Список заказов (Связывает таблицы "Заказы", "Виды неисправностей", "Обслуживаемые магазины" и "Сотрудники" по полям "Код вида неисправности", "Код магазина" и "Код сотрудника").

Вариант 15. БД Школы.

Таблицы:	Сотрудники (Код сотрудника, ФИО, Возраст, Пол, Адрес, Телефон, Паспортные данные, Код должности)[10 записей]. Должности (Код должности, Наименование должности, Оклад, Обязанности, Требования)[5 записей]. Ученики (ФИО, Дата рождения, Пол, Адрес, ФИО отца, ФИО матери, Код класса, Дополнительная информация) [10 записей]. Классы (Код класса, Код сотрудника-класного руководителя, Код вида, Количество учеников, Буква, Год обучения, Год создания)[5 записей]. Виды классов (Код вида, Наименование, Описание)[5 записей]. Предметы (Код предмета, Наименование, Описание, Код сотрудника-учителя)[10 записей]. Расписание (Дата, День недели, Код класса, Код предмета, Время начала, Время окончания)[10 записей].
Запросы:	Отдел кадров (Связывает таблицы "Сотрудники" и "Должности" по полю "Код должности"). Список учеников (Связывает таблицы "Ученики" и "Классы" по полю "Код класса"). Список классов (Связывает таблицы "Классы", "Виды классов" и "Сотрудники" по полям "Код вида" и "Код сотрудника"). Список предметов (Связывает таблицы "Предметы" и "Сотрудники" по полю "Код сотрудника"). Расписание занятий (Связывает таблицы "Расписание", "Классы" и "Предметы" по полям "Код класса" и "Код предмета").

Вариант 16. БД Транспортной компании.

Таблицы:	Сотрудники (Код сотрудника, ФИО, Возраст, Пол, Адрес, Телефон, Паспортные данные, Код должности)[10 записей].

	Должности (Код должности, Наименование должности, Оклад, Обязанности, Требования)[5 записей]. Виды автомобилей (Код вида автомобиля, Наименование, Описание)[5 записей]. Марки автомобилей (Код марки, Наименование, Технические характеристики, Описание) [5 записей]. Виды грузов (Код вида груза, Наименование, Код вида автомобиля для транспортировки, Описание)[5 записей]. Грузы (Код груза, Наименование, Код вида груза, Срок годности, Особенности)[5 записей]. Автомобили (Код автомобиля, Код марки, Код вида автомобиля, Регистрационный номер, Номер кузова, номер двигателя, Год выпуска, Код сотрудника-водителя, Дата последнего ТО, Код сотрудника-механика)[5 записей]. Рейсы (Код автомобиля, Заказчик, Откуда, Куда, Дата отправления, Дата прибытия, Код груза, Цена, Отметка об оплате, Отметка о возвращении, Код сотрудника)[10 записей].
Запросы:	Отдел кадров (Связывает таблицы "Сотрудники" и "Должности" по полю "Код должности"). Транспортировка (Связывает таблицы "Виды грузов" и "Виды автомобилей" по полю "Код вида автомобиля"). Перевозимые грузы (Связывает таблицы "Грузы" и "Виды грузов" по полю "Код вида груза"). Автопарк (Связывает таблицы "Автомобили", "Марки автомобилей", "Виды автомобилей" и "Сотрудники" по полям "Код марки", "Код вида автомобиля" и "Код сотрудника"). Заказы (Связывает таблицы "Рейсы", "Автомобили", "Грузы" и "Сотрудники" по полям "Код автомобиля", "Код груза" и "Код сотрудника").

Вариант 17. БД Проката автомобилей.

Таблицы:	Сотрудники (Код сотрудника, ФИО, Возраст, Пол, Адрес, Телефон, Паспортные данные, Код должности)[10 записей]. Должности (Код должности, Наименование должности, Оклад, Обязанности, Требования)[5 записей]. Марки автомобилей (Код марки, Наименование, Технические характеристики, Описание) [5 записей]. Дополнительные услуги (Код услуги, Наименование, Описание, Цена)[5 записей]. Автомобили (Код автомобиля, Код марки, Регистрационный номер, Номер кузова, Номер двигателя, Год выпуска, Пробег, Цена автомобиля, Цена дня проката, Дата последнего ТО, Код сотрудника-механика, Специальные отметки, Отметка о возврате)[10 записей]. Клиенты (Код клиента, ФИО, Пол, Дата рождения, Адрес, Телефон, Паспортные данные) [5 записей]. Прокат (Дата выдачи, Срок проката, Дата возврата, Код автомобиля, Код клиента, Код услуги 1, Код услуги 2, Код услуги 3, Цена проката, Отметка об оплате, Код сотрудника)[10 записей].

Вариант 18. БД Оптового склада.

Таблицы:	Сотрудники (Код сотрудника, ФИО, Возраст, Пол, Адрес, Телефон, Паспортные данные, Код должности)[10 записей].

	Должности (Код должности, Наименование должности, Оклад, Обязанности, Требования)[5 записей]. Товары (Код товара, Код типа, Производитель, Наименование, Условия хранения, Упаковка, Срок годности) [10 записей]. Типы товаров (Код типа, Наименование, Описание, Особенности) [5 записей]. Поставщики (Код поставщика, Наименование, Адрес, Телефон, Код поставляемого товара 1, Код поставляемого товара 2, Код поставляемого товара 3) [5 записей]. Заказчики (Код заказчика, Наименование, Адрес, Телефон, Код потребляемого товара 1, Код потребляемого товара 2, Код потребляемого товара 3) [5 записей]. Склад (Дата поступления, Дата заказа, Дата отправки, Код товара, Код поставщика, Код заказчика, Способ доставки, Объём, Цена, Код сотрудника) [10 записей].
Запросы:	Отдел кадров (Связывает таблицы "Сотрудники" и "Должности" по полю "Код должности"). Список товаров (Связывает таблицы "Товары" и "Типы товаров" по полю "Код типа"). Список поставщиков (Связывает таблицы "Поставщики" и "Товары" по полям "Код товара", "Код поставляемого товара 1", "Код поставляемого товара 2" и "Код поставляемого товара 3"). Список заказчиков (Связывает таблицы "Заказчики" и "Товары" по полям "Код товара", "Код потребляемого товара 1", "Код потребляемого товара 2" и "Код потребляемого товара 3"). Заказы (Связывает таблицы "Склад", "Товары", "Поставщики", "Заказчики" и "Сотрудники" по полям "Код товара", "Код поставщика", "Код заказчика" и "Код сотрудника").

Вариант 19. БД Строительной компании.

Таблицы:	Сотрудники (Код сотрудника, ФИО, Возраст, Пол, Адрес, Телефон, Паспортные данные, Код должности)[10 записей]. Должности (Код должности, Наименование должности, Оклад, Обязанности, Требования)[5 записей]. Виды работ (Код вида, Наименование, Описание, Цена работы, Код материала 1, Код материала 2, Код материала 3)[5 записей]. Материалы (Код материала, Наименование, Упаковка, Описание, Цена) [5 записей]. Бригады (Код бригады, Код сотрудника 1, Код сотрудника 2, Код сотрудника 3) [5 записей]. Заказчики (Код заказчика, ФИО, Адрес, Телефон, Паспортные данные)[5 записей]. Заказы (Код заказчика, Код вида работ, Код бригады, Стоимость, Дата начала, Дата окончания, Отметка о завершении, Об оплате, Код сотрудника) [10 записей].
Запросы:	Отдел кадров (Связывает таблицы "Сотрудники" и "Должности" по полю "Код должности"). Список работ (Связывает таблицы "Виды работ" и "Материалы" по полям "Код материала", "Код материала 1", "Код материала 2" и "Код материала 3"). Список бригад (Связывает таблицы "Бригады" и "Сотрудники" по полям

	"Код сотрудника", "Код сотрудника 1", "Код сотрудника 2" и "Код сотрудника 3"). Список заказов (Связывает таблицы "Заказы", "Виды работ", "Бригады" и "Сотрудники" по полям "Код вида", "Код бригады" и "Код сотрудника").

Вариант 20. БД Риэлтерской фирмы.

Таблицы:	Сотрудники (Код сотрудника, ФИО, Возраст, Пол, Адрес, Телефон, Паспортные данные, Код должности)[10 записей]. Должности (Код должности, Наименование должности, Оклад, Обязанности, Требования)[5 записей]. Виды услуг (Код вида услуги, Наименование, Описание, Цена)[5 записей]. Виды квартир (Код вида, Наименование, Описание)[5 записей]. Продавцы (Код продавца, ФИО, Пол, Дата рождения, Адрес проживания, Телефон, Паспортные данные, Код вида квартиры, Адрес квартиры, Количество комнат, Площадь, Отметка о раздельном санузле, Отметка о наличии телефона, Цена, Дополнительная информация)[10 записей]. Покупатели (Код покупателя, ФИО, Пол, Дата рождения, Адрес проживания, Телефон, Паспортные данные, Код вида квартиры, Количество комнат, Площадь, Отметка о раздельном санузле, Отметка о наличии телефона, Цена, Дополнительные пожелания)[10 записей]. Договоры (Дата заключения, Код продавца, Код покупателя, Сумма сделки, Стоимость услуг, Код вида услуги, Код сотрудника)[10 записей].
Запросы:	Отдел кадров (Связывает таблицы "Сотрудники" и "Должности" по полю "Код должности"). Продажа (Связывает таблицы "Продавцы" и "Виды квартир" по полю "Код вида квартиры"). Покупка (Связывает таблицы "Покупатели" и "Виды квартир" по полю "Код вида квартиры"). Заключённые договора (Связывает таблицы "Договоры", "Продавцы", "Покупатели", "Услуги" и "Сотрудники" по полям "Код продавца", " Код покупателя", " Код услуги" и "Код сотрудника".

Вариант 21. БД Рекламного агентства.

Таблицы:	Сотрудники (Код сотрудника, ФИО, Возраст, Пол, Адрес, Телефон, Паспортные данные, Код должности)[10 записей]. Должности (Код должности, Наименование должности, Оклад, Обязанности, Требования)[5 записей]. Виды рекламы (Код вида, Наименование, Описание) [5 записей]. Дополнительные услуги (Код услуги, Наименование, Описание, Стоимость) [5 записей]. Места расположения (Код места, Наименование, Расположение, Код вида, Описание, Стоимость) [10 записей]. Заказчики (Код заказчика, ФИО, Адрес, Телефон) [10 записей]. Заказы (Дата заказа, Дата начала, Дата окончания, Код заказчика, Код места, Код услуги 1, Код услуги 2, Код услуги 3, Стоимость, Отметка об оплате, Код сотрудника) [10 записей].
Запросы:	Отдел кадров (Связывает таблицы "Сотрудники" и "Должности" по полю "Код должности"). Список мест (Связывает таблицы "Места расположения" и "Виды рекламы" по полю "Код вида"). Список заказов (Связывает таблицы "Заказы", "Заказчики", "Места

	расположения", "Дополнительные услуги" и "Сотрудники" по полям "Код заказчика", "Код места", "Код услуги", "Код услуги 1", "Код услуги 2", "Код услуги 3" и "Код сотрудника").

Вариант 22. БД Компьютерной фирмы.

Таблицы:	Сотрудники (Код сотрудника, ФИО, Возраст, Пол, Адрес, Телефон, Паспортные данные, Код должности)[10 записей]. Должности (Код должности, Наименование должности, Оклад, Обязанности, Требования)[5 записей]. Виды комплектующих (Код вида, Наименование, Описание)[15 записей]. Комплектующие (Код комплектующего, Код вида, Марка, Фирма производитель, Страна производитель, Дата выпуска, Характеристики, Срок гарантия, Описание, Цена)[15 записей]. Заказчики (Код заказчика, ФИО, Адрес, Телефон)[10 записей]. Услуги (Код услуги, Наименование, Описание, Стоимость)[5 записей]. Заказы (Дата заказа, Дата исполнения, Код заказчика, Код комплектующего 1, Код комплектующего 2, Код комплектующего 3, Доля предоплаты, Отметка об оплате, Отметка об исполнении, Общая стоимость, Срок общей гарантии, Код услуги 1, Код услуги 2, Код услуги 3, Код сотрудника)[10 записей].
Запросы:	Отдел кадров (Связывает таблицы "Сотрудники" и "Должности" по полю "Код должности"). Список комплектующих (Связывает таблицы "Комплектующие" и "Виды комплектующих" по полю "Код вида"). Список заказов (Связывает таблицы "Заказы", "Заказчики", "Комплектующие", "Услуги" и "Сотрудники" по полям "Код заказчика", "Код комплектующего", "Код комплектующего 1", "Код комплектующего 2", "Код комплектующего 3", "Код услуги", "Код услуги 1", "Код услуги 2", "Код услуги 3" и "Код сотрудника").

Вариант 23. БД ГИБДД.

Таблицы:	Сотрудники (Код сотрудника, ФИО, Возраст, Пол, Адрес, Телефон, Паспортные данные, Код должности, Код звания)[10 записей]. Должности (Код должности, Наименование должности, Оклад, Обязанности, Требования)[5 записей]. Звания (Код звания, Наименование, Надбавка, Обязанности, Требования)[5 записей]. Марки автомобилей (Код марки, Наименование, Фирма производитель, Страна производитель, Дата начала производства, Дата окончания производства, Характеристики, Категория, Описание)[10 записей]. Водители (Код водителя, ФИО, Дата рождения, Адрес, Паспортные данные, Номер водительского удостоверения, Дата выдачи удостоверения, Дата окончания удостоверения, Категория удостоверения, Описание, Код сотрудника)[15 записей]. Автомобили (Код автомобиля, Код водителя, Код марки, Регистрационный номер, Номер кузова, Номер двигателя, Номер техпаспорта, Дата выпуска, Дата регистрации, Цвет, Технический осмотр, Дата технического осмотра, Описание, Код сотрудника)[15 записей]. Автомобили в угоне (Дата угона, Дата обращения, Код автомобиля, Код водителя, Обстоятельства угона, Отметка об нахождении, Дата нахождения, Код сотрудника)[5 записей].

Запросы:	Отдел кадров (Связывает таблицы "Сотрудники", "Должности" и "Звания" по полям "Код должности" и "Код звания"). Список автомобилей (Связывает таблицы "Автомобили", "Марки автомобилей", "Водители" и "Сотрудники" по полям "Код марки", "Код водителя" и "Код сотрудника"). Список угонов (Связывает таблицы "Автомобили в угоне", "Автомобили" и "Водители" по полям "Код автомобиля" и "Код водителя").

Вариант 24. БД Кинотеатра.

Таблицы:	Сотрудники (Код сотрудника, ФИО, Возраст, Пол, Адрес, Телефон, Паспортные данные, Код должности)[10 записей]. Должности (Код должности, Наименование должности, Оклад, Обязанности, Требования)[5 записей]. Жанры (Код жанра, Наименование, Описание)[5 записей]. Фильмы (Код фильма, Наименование, Код жанра, Длительность, Фирма производитель, Страна производитель, Актёры, Возрастные ограничения, Описание)[10 записей]. Репертуар (Код сеанса, Дата, Время начала, Время окончания, Цена билета)[10 записей]. Места (Код сеанса, Номер места, Занятость, Код сотрудника)[15 записей].
Запросы:	Отдел кадров (Связывает таблицы "Сотрудники" и "Должности" по полю "Код должности"). Список фильмов (Связывает таблицы "Фильмы" и "Жанры" по полю "Код жанра"). Билеты (Связывает таблицы "Места", "Репертуар" и "Сотрудники" по полям "Код сеанса" и "Код сотрудника").

Вариант 25 БД Автосалона.

Таблицы:	Сотрудники (Код сотрудника, ФИО, Возраст, Пол, Адрес, Телефон, Паспортные данные, Код должности)[10 записей]. Должности (Код должности, Наименование должности, Оклад, Обязанности, Требования)[5 записей]. Производители (Код производителя, Наименование, Страна, Адрес, Описание, Код сотрудника)[5 записей]. Дополнительное оборудование (Код оборудования, Наименование, Характеристики, Цена)[5 записей]. Тип кузова (Код типа кузова, Название, Описание)[5 записей]. Автомобили (Код автомобиля, Марка, Код производителя, Код типа кузова, Дата производства, Цвет, Номер кузова, Номер двигателя, Характеристики, Код оборудования 1, Код оборудования 2, Код оборудования 3, Цена, Код сотрудника)[10 записей]. Заказчики (ФИО, Адрес, Телефон, Паспортные данные, Код автомобиля, Дата заказа, Дата продажи, Отметка о выполнении, Отметка об оплате, Процент предоплаты, Код сотрудника)[10 записей].
Запросы:	Отдел кадров (Связывает таблицы "Сотрудники" и "Должности" по полю "Код должности"). Каталог автомобилей (Связывает таблицы "Автомобили", "Производители", "Тип кузова", "Дополнительное оборудование" и "Сотрудники" по полям "Код производителя", "Код типа кузова", "Код оборудования", "Код оборудования 1", "Код оборудования 2", "Код оборудования 3" и "Код сотрудника").

	Список заказов (Связывает таблицы "Заказчики", "Автомобили" и "Сотрудники" по полям "Код автомобиля" и "Код сотрудника").

3. СТРУКТУРА РАБОТЫ

По своей структуре курсовая работа по дисциплине «Управление данными» состоит из введения, теоретической и практической частей, заключения, приложений и списка использованной литературы.

Примерный план курсового проекта:

ВВЕДЕНИЕ

1. ТЕОРЕТИЧЕСКАЯ ЧАСТЬ

1.1 Анализ тенденций развития отдельных направлений управления данными по вариантам).

2. ПРАКТИЧЕСКАЯ ЧАСТЬ

2.1 Обследование предметной области.
2.2 Техническое задание на разработку базы данных
2.3 Концептуальное проектирование.
2.3.1 Перечень сущностей (обосновать список).
2.3.2 Перечень атрибутов.
2.4 Инфологическое проектирование БД.
2.4.1 Модель “сущность-связь”.
2.4.2 Классификация связей.
2.5.Реляционная модель БД.
2.5.1 Функциональные зависимости между атрибутами.
2.5.2 Выбор ключей.
2.5.3 Нормализация отношений.
2.6 Даталогическое проектирование БД.
2.6.1 Состав таблиц БД.
2.6.2 Средства поддержания целостности.
2.7 Запросы к БД.
2.8 Разработка механизмов защиты данных от несанкционированного доступа.
2.9 Требования к техническому обеспечению.
2.10 Инструкция по использованию БД.
2.10.1 Вызов программы.
2.10.2 Экранные формы.
2.10.3 Описание отчетов.

ЗАКЛЮЧЕНИЕ

СПИСОК ИСПОЛЬЗУЕМОЙ ЛИТЕРАТУРЫ

ПРИЛОЖЕНИЯ

3.1 Введение к курсовому проекту

Во введении к курсовому проекту обосновывается актуальность темы, формулируется цель, а также приводятся краткие сведения о содержании пояснительной записки с разбивкой по разделам. Во введении необходимо отразить следующую информацию: наименование предприятия (истинное или

вымышленное), для которого разрабатывается база данных (БД), наименование предметной области, назначение разработки БД, место разрабатываемой БД в общей системе управления предприятием, требования заказчика к разрабатываемой БД, чья точка зрения используется при проектировании, перечисление пользователей БД, права пользователей, описание (перечисление) общетехнических и общесистемных программных средств. Примерный объем введения – не более 3 страниц машинописного текста.

3.2 Теоретическая часть курсового проекта

Теоретическая часть курсового проекта по дисциплине «Управление данными» содержит аналитический обзор тенденций развития систем управления данными, составленный на основе анализа научноисследовательской литературы и ресурсов Internet.

3.3 Практическая часть курсового проекта

В практическую часть курсового проекта по дисциплине «Управление данными» включены девять подразделов:

- обследование предметной области;
- концептуальное проектирование – выделение сущностей и атрибутов;
- инфологическое проектирование БД;
- разработка реляционной модель БД;
- даталогическое проектирование БД;
- запросы к БД;
- разработка механизмов защиты данных от несанкционированного доступа;
- требования к техническому обеспечению; - инструкция по использованию БД.

3.3.1 Обследование предметной области

В этом разделе необходимо указать область применения базы данных, то есть провести анализ предметной области, определив ее объекты и связи между ними.

При выборе состава и структуры предметной области возможны два подхода: функциональный и предметный.

Функциональный подход реализует принцип движения «от задач» и применяется, когда определен комплекс задач, для обслуживания которых создается информационная система. В этом случае можно выделить минимальный необходимый набор объектов предметной области, которые должны быть описаны.

В предметном подходе объекты предметной области определяются с таким расчетом, чтобы их можно было использовать при решении множества разнообразных, заранее не определенных задач.

При необходимости можно разработать словарь терминов предметной области. Здесь же необходимо указать источники информации, которые были использованы при анализе предметной области и информационных

потребностей пользователей; перечислить бизнес-процессы, для поддержки которых разрабатывается БД. А также провести анализ входных и выходных документов.

Обследование предметной области в процессе проектирования реляционной базы данных служит основой для разработки технического задания.

Пример. Компьютерная фирма ООО «Сатурн+» оказывает услуги по прокату художественных фильмов на CD- и DVD-носителях. Необходимо спроектировать информационную подсистему «Прокат», содержащую в себе реляционную база данных для учета выдачи носителей посетителям.

Пользователями подсистемы в данной фирме являются менеджер первого звена, менеджер по работе с клиентами, работник склада и бухгалтер. За аренду фильма начисляется плата, за просрочку возврата – пеня. Новые фильмы фирма приобретает у оптовых поставщиков, заключая с ними долговременные контракты. Менеджер по работе с клиентами, и работник склада регулярно готовят отчеты для руководства за определенный период времени об изменении спроса, о поступлении новых лент, о ценах на оптовом рынке, о финансовых показателях.

В функции менеджера по работе с клиентами входит контроль за возвратом носителей, не допуская их выдачу тем, кто просрочил срок аренды.

Функциональный анализ предметной области, лежащей в основе информационной подсистемы «Прокат», позволяет выделить 4 основные функции, которые будет выполнять данная система (рисунок 2.1). Каждый блок функций соответствует одному из пользователей и будет выполняться на его рабочем месте.

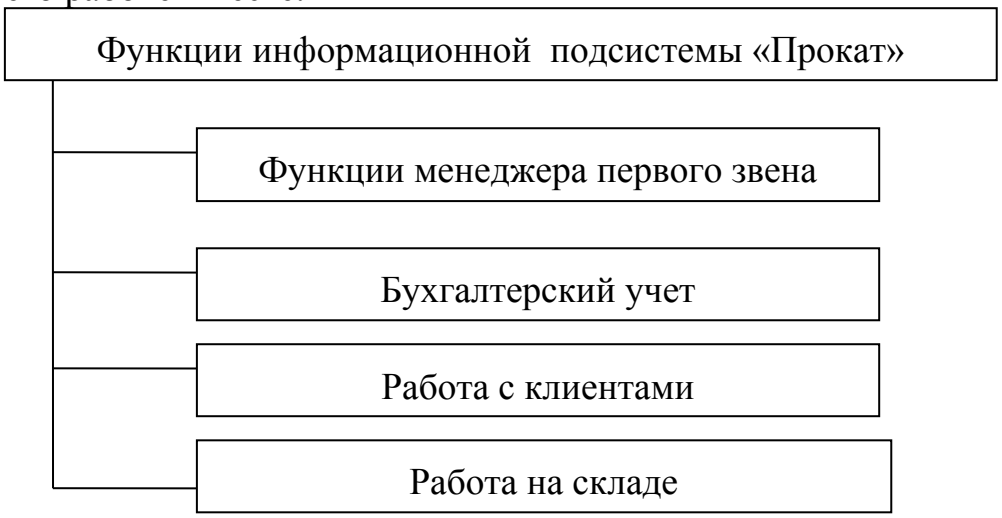


Рисунок 3.1 – Начало разработки функциональной диаграммы

На следующем этапе выполняется декомпозиция информационной подсистемы, то есть проводится анализ каждой функции. При выполнении процесса декомпозиции каждая из функций разбивается на более мелкие и составляется список тех документов, которые обрабатывает пользователь, разобьем функцию на более мелкие. На рисунке 2.2 представлена

декомпозиция функции «Работа с клиентами». В нижней половине каждого блока перечислены документы, которые нужны для выполнения данной функции. Родительский блок содержит также должность ответственного за функцию лица.

На основании предложенных функциональных диаграмм можно разработать техническое задание на проектирование базы данных, являющейся частью информационной подсистемы «Прокат».

База данных «Прокат» разрабатывается на основании рекомендации руководителя организации для автоматизации деятельности компании ООО «Сатурн+».

База данных «Прокат» предназначена для учета выдаваемых клиентам товаров, платежей и формирования всей необходимой документации и отчетов. Для ускорения обслуживания, работа может вестись с применением технологий штрихового кодирования и членских карточек. База данных должна иметь развитую систему разделения доступа сотрудников к различным функциям и отчетам.

Рисунок 3.2 – Детализация фрагмента диаграммы для функции

«Работа с клиентами»

На основании предложенных функциональных диаграмм можно разработать техническое задание на проектирование базы данных, являющейся частью информационной подсистемы «Прокат».

3.3.2 Разработка технического задания

На техническое задание существует стандарт ГОСТ 19.201-78 «Техническое задание. Требования к содержанию и оформлению». В соответствии с этим стандартом техническое задание должно содержать следующие разделы:

- введение;
- основания для разработки;
- назначение разработки;
- требования к программе или программному изделию;
- требования к программной документации;
- технико-экономические показатели;
- стадии и этапы разработки;
- порядок контроля и приемки.

При необходимости допускается в техническое задание включать приложения.

Пример технического задания. Требуется разработать техническое задание на разработку базы данных «Прокат».

1. ВВЕДЕНИЕ

Настоящее техническое задание распространяется на разработку базы данных «Прокат».

База данных «Прокат» предназначена для учета платежей и выдаваемых клиентам товаров, формирования всей необходимой документации и отчетов. Для ускорения обслуживания работа может вестись с применением технологий штрихового кодирования и членских карточек. База данных должна иметь развитую систему разделения доступа сотрудников к различным функциям и отчетам.

Разрабатываемая программа позволит автоматизировать работу с клиентами компьютерной фирмы ООО «Сатурн+» .

2. ОСНОВАНИЕ ДЛЯ РАЗРАБОТКИ

Программа разрабатывается на основе учебного плана кафедры «Информационные системы и технологии», утвержденного на заседании кафедры «___»_____201_г., протокол № ___. База данных «Прокат» разрабатывается на основании рекомендации руководителя организации ООО «Сатурн+» для автоматизации деятельности компании.

3. НАЗНАЧЕНИЕ

Основным назначением базы данных является помощь сотрудникам ООО «Сатурн+» при выполнении функций:

- регистрация клиентов;
- ввод товаров;
- автоматическое определение клиентов и товаров;
- быстрый поиск товаров по названию, жанру, ценовой категории, стране, режиссеру...;
- прием оплаты от клиентов, договора с клиентом и т.п.; - составление различных отчетов.

4.ТРЕБОВАНИЯ К ПРОГРАММЕ ИЛИ ПРОГРАММНОМУ ИЗДЕЛИЮ

4.1.Требования к функциональным характеристикам

4.1.1. Программа должна обеспечивать возможность выполнения следующих функций:

- инициализация системы, которая включает ввод информации о клиентах фирмы, имеющихся носителях и т.п.;
- ввод и корректировка текущей информации о деятельности фирмы;
- хранение информации о деятельности фирмы;
- получение сведений о текущем состоянии деятельности фирмы.

4.1.2. Исходные данные:

~ информация о клиентах фирмы;
~ каталог имеющихся носителей;
~ сотрудники фирмы;
~ договора с поставщиками;
~ договора с клиентами.

4.1.3. Результаты:

- информация о клиентах фирмы, имеющих задолженности по аренде носителей;
- информация о клиентах фирмы, не имеющих задолженности по аренде носителей;
- информация о поставке новых носителей;
- поиск товаров по названию, жанру, ценовой категории, стране, режиссеру

4.2. Требования к надежности

4.2.1.Предусмотреть контроль вводимой информации.

4.2.2.Предусмотреть блокировку некорректных действий пользователя при работе с системой.

4.2.3.Обеспечить целостность хранимой информации.

4.2.4. Обеспечить разграничение доступа к функциям системы для сотрудников пункта проката.

4.3. Требования к составу и параметрам технических средств

4.3.1.Система должна работать на IBM совместимых персональных компьютерах.

4.3.2.Минимальная конфигурация:

~ тип процессора - Pentium и выше;

~ объем оперативного запоминающего устройства - 32 Мб и более.

4.4. Требования к информационной и программной совместимости

Система должна работать под управлением семейства операционных систем Win 32 (Windows 95, Windows 98, Windows 2000, Windows NT и т. п.).

5. ТРЕБОВАНИЯ К ПРОГРАММНОЙ ДОКУМЕНТАЦИИ

5.1.Разрабатываемые программные модули должны быть самодокументированы, т. е. тексты программ должны содержать все необходимые комментарии.

5.2.Программная система должна включать справочную информацию о работе и подсказки пользователю.

5.3.В состав сопровождающей документации должны входить:

5.3.1.Пояснительная записка, содержащая описание разработки.

5.3.2. Руководство пользователя.

5.3.3.Схема структурная базы данных.

5.3.4. Формы интерфейса пользователя. База данных должна иметь простой и интуитивно-понятный, настраиваемый интерфейс пользователя.

3.3.3 Концептуальное проектирование

В процессе концептуального проектирования базы данных производится выбор информационных объектов, составляется перечень сущностей и атрибутов.

Для выделения информационных объектов в проектируемой базе данных необходимо соблюдать следующие условия:

1. Выявить документы, использующиеся фирмой в своей деятельности, и их реквизиты (поля).
2. Определить функциональную зависимость между реквизитами для каждого документа в отдельности.
3. Выбрать все зависимые реквизиты и указать для каждого из них все его ключевые (один или несколько), т.е. те, от которых он зависти функционально полно.
4. Сгруппировать реквизиты, зависимые от одинаковых ключевых реквизитов. Полученные группы и будут являться информационным объектом.

Совокупность полей – реквизитов выделенного объекта должна отвечать требованиям нормализации, в том числе:

- информационный объект должен содержать уникальный ключ, который может быть как простым, так и составным;
- все остальные (описательные или дополнительные) реквизиты должны быть независимыми друг от друга;
- все реквизиты, входящие в составной ключ, также должны быть независимыми;
- каждый описательный реквизит должен функционально полно зависеть от ключа, т.е. каждому значению ключа должно соответствовать только одно значение описательного реквизита;

- при составном ключе описательный реквизит должен зависеть целиком от всей совокупности реквизитов, образующих ключ;
- каждый описательный реквизит не должен зависеть от ключа транзитивно, через другой промежуточный реквизит.

Проведенный анализ предметной области «Прокат» позволяет выделить следующие сущности: «Клиент», «Носитель», «Заказ» и «Сотрудник».

Сущность «Клиент» может быть описана следующем перечнем атрибутов:

- код клиента,
- фамилия, имя, отчество клиента;
- паспортные данные;
- адрес;
- телефон;
- категория

Сущность «Носители» характеризуется атрибутами:

- код носителя;
- вид носителя;
- содержание носителя (художественный фильм или программное обеспечение);
- наименование;
- аннотация;
- обложка;
- стоимость проката.

Сущность «Сотрудник» можно описать с помощью атрибутов:

- код сотрудника;
- фамилия, имя, отчество сотрудника;
- личный пароль.

Центральной в проектируемой базе данных «Прокат» является сущность «Заказ», которая может быть описана с помощью атрибутов:

- код заказа;
- дата заказа;
- код клиента;
- код носителя;
- код сотрудника;
- дата возврата;
- стоимость услуги.

Первичными (уникальными) ключами для сущностей, входящих в базу данных «Прокат» являются:

- сущность «Клиент» – код клиента;
- сущность «Носитель» – код носителя; - сущность «Сотрудник» – код сотрудника; - сущность «Заказ» – код заказа.

Для сущности «Заказ» уникальные ключи «код клиента», «код носителя» и «код сотрудника» являются вторичными, и с их помощью осуществляется связь между всеми выбранными сущностями.

3.3.4 Инфологическое проектирование

В данном подразделе курсового проекта приводится алгоритм разработки ER-диаграммы (диаграммы «сущность – связь») разработанной модели предметной области, рассматривается классификация бинарных связей между сущностями, описывается моделирование связи «один-комногим». Рассматривается перечень атрибутов, описывающих, идентифицирующих или моделирующих свойства сущностей.

Основными задачами инфологического проектирования при разработке реляционных баз данных являются описание предметной области и формирование взгляда на предметную область с точки зрения пользователей базы данных.

Инфологическая модель предметной области включает:

- описание ее структуры и динамики;
- характер информационных потребностей пользователей в терминах, понятных пользователю и не зависимых от реализации баз данных.

Это описание выражается в терминах не отдельных объектов предметной области и связей между ними, а их типов, связанных с ними ограничений целостности и процессов, которые приводят к переходу предметной области из одного состояния в другое.

Основными подходами к созданию инфологической модели предметной области являются: функциональный подход к проектированию базы данных; предметный подход к проектированию базы данных; проектирование с использованием метода «сущность-связь»

Функциональный подход к проектированию базы данных. Этот метод реализует принцип «от задач» и применяется для обслуживания информационных потребностей тогда, когда известны функции некоторой группы лиц и|или комплекса задач, для которых создаётся рассматриваемая база данных.

Предметный подход к проектированию базы данных. Предметный подход к проектированию базы данных применяется в тех случаях, когда у разработчиков есть чёткое представление о самой предметной области и о том, какую именно информацию они хотели бы хранить в базе данных, а структура запросов не определена или определена не полностью. Тогда основное внимание должно быть уделено всестороннему обследованию предметной области и наиболее адекватному её отображению в базе данных с учётом самого широкого спектра информационных запросов к ней.

Проектирование с использованием метода «сущность-связь». Метод «сущность-связь» (entity-relation, ER-method) является комбинацией двух предыдущих и обладает достоинствами обоих.

Проектировщик разбивает предметную область на ряд локальных областей, каждая из которых включает в себя информацию, достаточную для обеспечения запросов отдельной группы будущих пользователей или решения

отдельной задачи. Каждое локальное представление моделируется отдельно, затем они объединяются.

Диаграмма «сущность-связь» - Entity-Relationship Diagram в англоязычной литературе обозначается термином «ERD». ERD – это графическое представление структуры предметной области в терминах модели ER (сущностей-связей). На диаграммах именованные прямоугольники обозначают множества сущностей, а именованные ромбы – множества связей сущностей. Дуги, связывающие прямоугольники и ромбы могут быть ориентированными.

На рисунке 3.3 показана диаграмма «сущность-связь» (ERD) с тремя сущностями: «Клиент», «Сотрудник» и «Заказ». Для получения диаграммы «сущность-связь» для всей базы данных «Прокат» необходимо добавить еще одну сущность «Носитель».

Впервые модель «сущность-связь» (ERD) была предложена Петеров Пин-Шен Ченом в 1976 году. На использовании различных разновидностей ERD основано большинство современных подходов к проектированию баз данных.

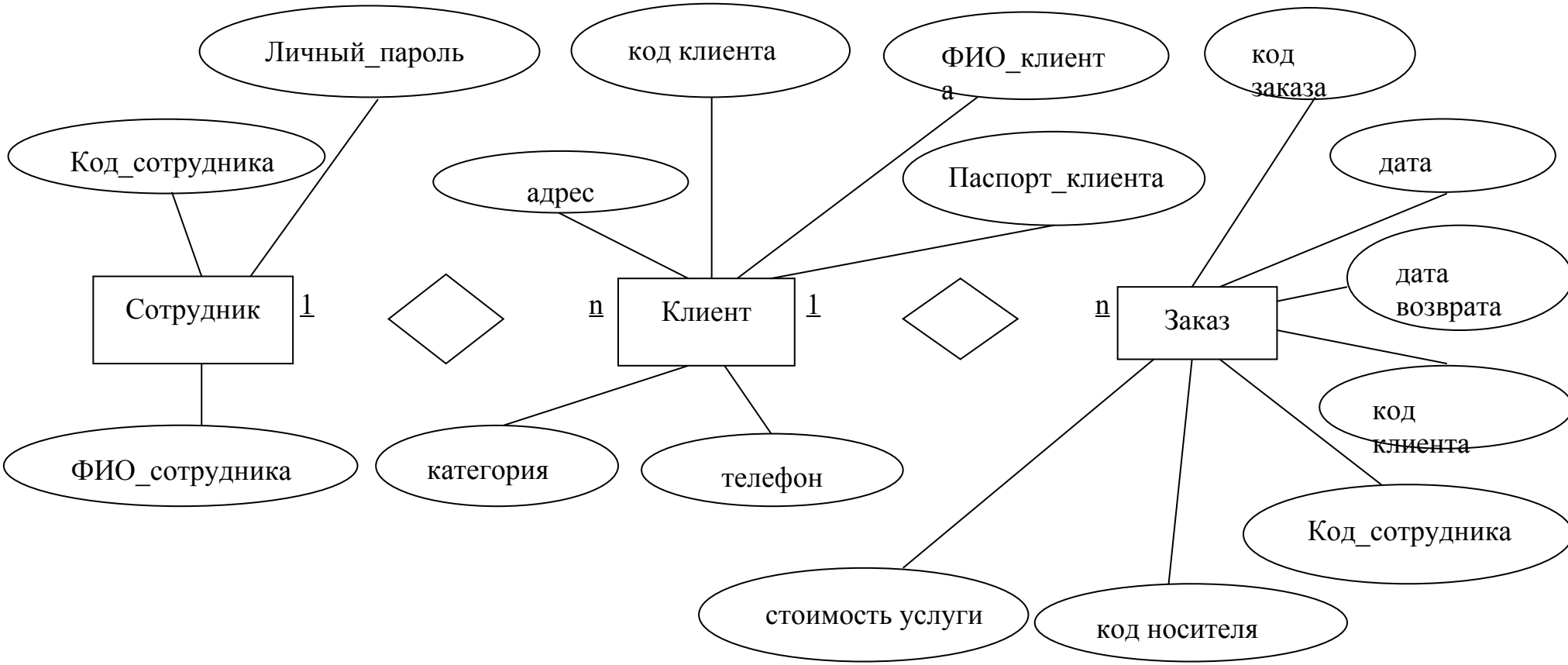


Рисунок 3.3 – ERD с тремя сущностями

Когда в фирму приходит новый клиент, он проходит процедуру регистрации, после чего его заносят в базу данных клиентов проката и вручают ему членскую карточку. На этой членской карточке присутствует уникальный идентификационный номер клиента. С помощью этого номера оператор видео проката быстро идентифицирует клиента и получает полную картину о нем.

Все товары также пронумерованы. Каждому товару присваивается уникальный номер. Этикетка с этим кодом наклеивается на каждый товар, который выставляется на стеллаж. Клиент может взять любой товар, рассмотреть его и, если товар ему понравился, подойти с ним к продавцу. По коду товара оператор очень быстро находит товар в программе, получает от клиента деньги и отдает ему товар. Таким образом, вся процедура обслуживания клиента занимает несколько секунд.

При построении инфологической модели на основе метода «сущностьсвязь» выделяются три основных типа связей межу сущностями:

- «один – к – одному»,
- «один – ко – многим»,
- «многие – ко – многим».

На основании проведенного концептуального проектирования можно сделать вывод о том, что между всеми сущностями в проектируемой базе «Прокат» данных связь между объектами осуществляется по типу «один – ко – многим».

Так один сотрудник может обслуживать много клиентов, но при этом клиент может сделать много заказов.

3.3.5 Реляционная модель БД

При переходе от концептуальной модели предметной области к схеме БД (например, реляционной) используются три различных подхода, которые подробно описаны в списке использованных источников.

При первом подходе преобразование осуществляется вручную.

Второй подход основан на автоматизированной компиляции концептуальной модели предметной области в схему БД, в результате которой создается реляционная база данных в третьей нормальной форме. Наиболее популярным программным продуктом для автоматизированного проектирования БД является CASE-средство ERwin фирмы Platinum.

Третий подход основан на применении СУБД, использующих семантические модели данных.

В рамках курсового проекта по дисциплине «Управление данными» используется первый подход. При этом сущности модели «сущность-связь» соответствует отношение (таблица) реляционной БД. Связи между сущностями типа «один – к – одному (1:1)» и «один – ко – многим (1:М)» соответствуют связям между отношениями. Атрибуты, которые идентифицируют, определяют или моделируют сущности в модели

«сущность-связь» соответствуют атрибутам отношений. Сложнее моделируется связь типа «M : N» или «многие-ко-многим». Такой тип связи может быть представлен вспомогательным отношением и двумя функциональными связями 1:M и M:1. Рассмотрим пример, в котором имеются 2 сущности: «Преподаватель» и «Дисциплина». Если преподаватель ведет занятия по нескольким дисциплинам, а занятия по одной и той же дисциплине могут вести разные преподаватели, то связь между сущностями «Преподаватель» и «Дисциплина» соответствует типу «многие-ко-многим». Представление связи типа «M : N» показано на рисунке 2.4. Сущности «Преподаватель» и «Дисциплина» преобразуются в отношения. Связь типа «M : N» преобразуется в дополнительное отношение.

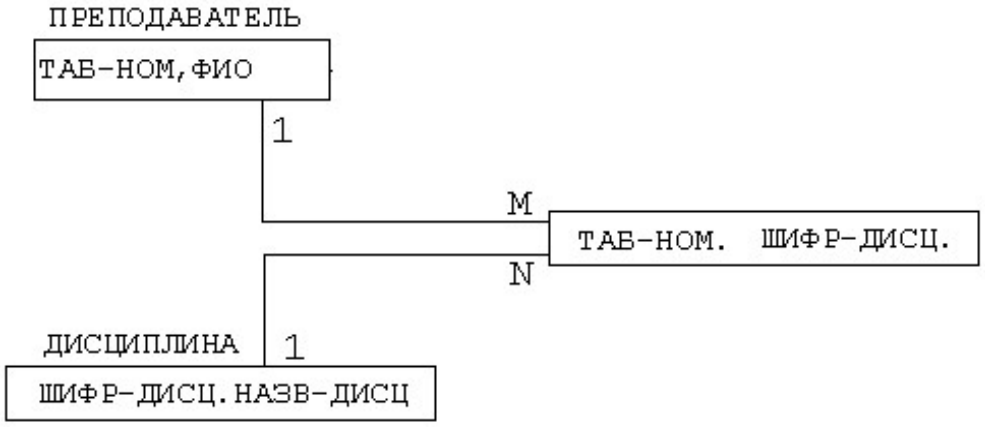


Рисунок 3.4 - Представление связи типа «M : N»

Нормализация баз данных основана на функциональных зависимостях между атрибутами отношений. Функциональную зависимость атрибута Y от X обозначают с помощью записи: X→Y. На практике используются первая, вторая и третья нормальные формы. В рамках курсового проекта необходимо разработать реляционную базу данных в третьей нормальной форме.

Выполнение требования первой нормальной формы необходимо для того, чтобы работали SQL-запросы. Схема отношения R находится в первой нормальной форме, когда все атрибуты атомарны (неделимы). Обычно имя столбца, состоящее из нескольких слов, записывают одним словом, например, код_заказа, дата_заказа, код_носителя, так как пробел в имени столбца будет нарушать работу запросов. Последующая нормализация, связанная с ограничением функциональных зависимостей в схеме отношения, необходима для уменьшения нежелательных характеристик БД по отношению к включению, удалению, модификации.

Вторая нормальная форма запрещает включать в отношение частичные функциональные зависимости. Если в отношении все ключи простые (т.е. состоят из одного атрибута), то отношение, которое находится в первой нормальной форме, находится одновременно и во второй нормальной форме. Проверять выполнение второй нормальной формы необходимо, если первичный ключ отношения является составным (т.е. состоит из нескольких

атрибутов). Схема отношения R находится во второй нормальной форме, если она соответствует первой нормальной форме, и каждый атрибут, не входящий в первичный ключ в этом отношении, полностью зависит от ключа этого отношения R. Атрибут А частично зависит от К, если К есть составной ключ отношения R, $К_1$ ⊂ К, и одновременно А зависит от ключа, т.е. К → А и от части ключа, т.е. $К_1$ → А.

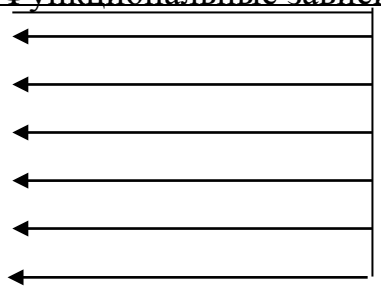
Третья нормальная форма запрещает функциональные зависимости непервичных (т.е. не входящих в первичный ключ) атрибутов от других непервичных атрибутов. Схема отношения R находится в третьей нормальной форме, если она соответствует второй нормальной форме и каждый атрибут, не являющийся первичным в этом отношении, нетранзитивно зависит от каждого возможного ключа.

Атрибут А называется транзитивно зависимым от К в отношении R на множестве функциональных зависимостей F, если существует атрибут Y в этом же отношении Y ⊂ R, такой что К → Y, Y → А (при этом К не зависит от Y). Чтобы привести схему отношения к третьей нормальной форме обычно производят декомпозицию отношения R на несколько отношений, чаще на два, таким образом, чтобы ни в одном из отношений не было транзитивных зависимостей. В приведенном примере зависимость К → А является транзитивной, поэтому при декомпозиции эти атрибуты (К и А) должны попасть в разные отношения.

Рассмотрим схему отношения «Заказ» из базы данных «Прокат». В схеме отношения Заказ (Код_заказа, дата_заказа, код_клиента, код_носителя, код_сотрудника, дата_возврата, стоимость_услуги) все атрибуты атомарны. Это позволяет сделать вывод о том, что требования первой нормальной формы выполняются.

Проведем анализ функциональных зависимостей между атрибутами в отношении «Заказ» из базы данных «Прокат». В таблице 2.1 показана зависимость всех атрибутов от атрибута «Код_заказа». Первичным ключом в отношении «Заказ» может быть только атрибут «Код_заказа», так как именно от этого атрибута функционально зависят все остальные атрибуты. Так как первичный ключ «Код_заказа» отношения «Заказ» является простым, то из этого следует, что в отношении нет частичных функциональных зависимостей, поэтому схема отношения находится во второй нормальной форме. Необходимо далее провести анализ предметной области с целью выявления и других функциональных зависимостей, если они есть. Если при анализе схемы отношения не выявлена зависимость непервичных атрибутов от других непервичных атрибутов, то можно сделать вывод об отсутствии транзитивных зависимостей в отношении «Заказ» и о том, что требования третьей нормальной формы выполняются.

Таблица 3.1 – Функциональные зависимости между атрибутами сущности «Заказ»

Наименование атрибутов	Функциональные зависимости
Код_заказа дата_заказа код_клиента код_носителя код_сотрудника дата_возврата стоимость_услуги	

При разработке реляционной базы данных необходимо проанализировать функциональные зависимости между атрибутами всех отношений и выбрать идентифицирующие атрибуты, которые в реляционной модели данных используются в качестве первичных ключей.

После этого необходимо нормализовать отношения, исключив частичные и транзитивные функциональные зависимости. Проверить соответствие схем отношений требованиям второй и третьей нормальной формы.

Логическая связь отношений в реляционной БД является иерархической. Одно из отношений является основным (родительским или PARENT TABLE), а другое – подчиненным (дочерним или CHILD TABLE).

Для поддержания связей оба отношения должны содержать одинаковые наборы атрибутов (ключи), по которым они связаны. В основном отношении это первичный ключ (PRIMARY KEY), который однозначно определяет кортеж основного отношения. В подчиненном отношении этот же набор атрибутов называют внешним ключом отношения (FOREIGN KEY). В подчиненном отношении множество кортежей, определяемых внешним ключом, связано с единственным кортежем основного отношения. Условие связи отношений:

PARENT TABLE. PRIMARY KEY= CHILD TABLE. FOREIGN KEY

3.3.6 Даталогическая модель БД

3.3.6.1. Состав таблиц

При разработке даталогической модели реляционной базы данных необходимо определить состав таблиц. Затем для каждого поля таблицы указать размер поля (в количестве символов), тип данных. Для первичных ключей необходимо ввести запрет неопределенных значений. Для остальных полей возможность запрета неопределенных значений определяется семантикой предметной области. Пример описания состава таблицы приведен в таблице 3.2.

Таблица 3.2 – Состав таблицы «Клиент»

Наименование атрибутов	Тип поля	Размер поля	Допустимость неопределенных значений

Код_клиента	Integer		NOT NULL
ФИО_клиента	Character	30	
Паспортные_данные	Character	20	
адрес телефон категория	Character	20	
	Character	15	
	Character	15	

Для создания первичных и внешних ключей отношений используются индексы или индексные выражения, подробно описанные в методических указаниях к лабораторным работам. После определения состава отношений (таблиц) и описания их полей (по аналогии с таблицей 3.2) необходимо установить постоянные отношения между таблицами, т.е. связи по ключам. Для этого необходимо выбрать родительские (управляющие) таблицы, на первичные ключи которых ссылаются другие (дочерние) таблицы.

3.3.6.2 Средства поддержания целостности

Важнейшим вопросом, возникающим при проектировании и эксплуатации систем обработки данных, является обеспечение целостности данных. Нарушение целостности может быть связано не только с ошибками оператора при вводе данных, но и со сбоями в работе системы. Отказ в работе системы может возникнуть, когда одни операции были выполнены, а другие, связанные с ними определенными ограничениями операции, не успели выполниться. Например, удалена запись о сотруднике, а не удалены относящиеся к ней записи о его детях и т. п. Нарушение целостности возникает при вводе или корректировке данных, а также при выполнении реляционных операций (проекция, соединение).

Решение проблемы целостности заключается в обеспечении в любой момент времени правильности данных в базе данных, для этого используют правила проверки достоверности данных гарантирующие, что недействительные данные не попадут в таблицы БД. Целостность данных обеспечивается набором специальных предложений, называемых ограничениями целостности. Ограничения целостности, присущие той или иной предметной области, должны быть выявлены при обследовании и зафиксированы в инфологической модели. Для того чтобы обеспечить целостность при выполнении реляционных операций, необходимо, чтобы соблюдались определенные правила при их выполнении, а также чтобы файлы были правильно спроектированы.

Целостность (от англ. integrity – нетронутость, неприкосновенность, сохранность, целостность) – это правильность данных в любой момент времени.

Но цель по поддержанию целостности может быть достигнута лишь в определенных пределах: СУБД не может контролировать правильность каждого отдельного значения, вводимого в базу данных. Например, нельзя

обнаружить, что вводимое значение 5 (представляющее номер дня недели) в действительности должно быть равно 3. Поддержание целостности базы данных может рассматриваться как защита данных от неверных и случайных действий персонала или разрушений. Ограничения целостности определяются в большинстве случаев особенностями предметной области, хотя могут отражать и чисто информационные характеристики. Например, в проектируемой базе данных «Прокат» примером одного из ограничений может служить запрет на ввод будущей даты.

Ограничения целостности могут относиться к разным информационным объектам базы данных, в том числе атрибутам (полям), кортежам (строкам, записям), отношениям (таблицам, файлам), связям между файлами и так далее. Современные СУБД имеют ряд средств для поддержания целостности баз данных.

Выделяют три группы правил поддержания целостности:

- Целостность по сущностям.
- Целостность по ссылкам.
- Целостность, определяемая пользователем.

Целостность по сущностям состоит в следующем: Не допускается, чтобы первичный ключ сущности принимал неопределенное значение. Неопределенные значения соответствуют пустым клеткам таблиц. Запрет неопределенных значений обозначают как: «NOT NULL». Ключевое слово NULL может обозначать в базах данных различные понятия: неопределенное значение, ложь (в логическом поле), нуль (в числовом поле). При попытке ввода новой строки с неопределенным первичным ключом операция ввода отвергается средствами СУБД.

Первичный ключ должен быть уникален. При попытке ввода двух разных строк с одинаковыми первичными ключами операция ввода отвергается средствами СУБД. Таким образом, целостность по сущностям, которая ограничивает возможные значения первичного ключа, поддерживается автоматически средствами СУБД.

Внешний ключ может принимать неопределенные значения. СУБД не отвергает ввод строк с неопределенными значениями внешнего ключа. В логически связанных таблицах значение внешнего ключа дочерней таблицы должно быть либо:

- равно значению первичного ключа родительской таблицы;
- полностью неопределенным.

Целостность по ссылкам определяет механизм каскадирования изменений первичного ключа отношения, при котором согласовано изменяются логически связанные с первичным внешние ключи, или механизм, который отвергает эти изменения.

Таким образом, для каждого внешнего ключа проектировщик базы данных должен специфицировать не только поле или комбинацию полей, составляющих этот внешний ключ, и основную таблицу, которая идентифицируется этим ключом, но также и ответы на вопросы:

- что должно происходить при попытке ОБНОВЛЕНИЯ первичного ключа родительской таблицы, на которую ссылается некоторый внешний ключ?
- что должно происходить при попытке УДАЛЕНИЯ первичного ключа родительской таблицы, на которую ссылается некоторый внешний ключ?

На эти вопросы можно дать три ответа (по выбору проектировщика):

1. КАСКАДИРОВАНИЕ. Операция обновления (или удаления) «каскадируется» с тем, чтобы обновить (удалить) также и внешний ключ в дочерней таблице. (На лабораторных работах для каскадирования обновления (и удаления) используется оператор CASCADE).
2. ОГРАНИЧЕНИЕ. Обновляются (или удаляются) первичные ключи лишь тех записей, на которые нет ссылок в дочерних таблицах. При этом используется оператор RESTRICT.
3. УСТАНОВЛЕНИЕ неопределенных значений (NULL-значений). Внешний ключ устанавливается в неопределенное значение, а затем обновляется первичный ключ родительской таблицы.

Целостность, определяемая пользователем. Для любой конкретной базы данных существует ряд дополнительных специфических правил, которые относятся к ней одной и определяются разработчиком. Чаще всего контролируется:

- уникальность тех или иных атрибутов,
- диапазон значений (обычно используется для числовых полей, различают открытые и закрытые диапазоны; первые фиксируют значение только одной из границ, а вторые фиксируют ебе границы; например, экзаменационная оценка от 2 до 5;
- принадлежность набору значений или *задание домена* (пол "М" или "Ж").

Ограничения целостности – это утверждения о допустимых значениях отдельных информационных единиц и связях между ними.

Ограничение целостности устанавливает правила на уровне баз данных, определяя набор проверок для таблиц системы. Эти проверки автоматически выполняются всякий раз, когда вызывается оператор вставки, модификации или удаления данных в таблице. Если какие-либо ограничения нарушены, операторы отменяются.

Для полей в таблицах, входящих в структуру базы данных, в основном используются следующие виды ограничений.

1. *Тип и формат поля*.
2. *Признак непустого поля*. Характеризует недопустимость пустого значения поля в БД. Например, поля со сведениями о сотрудниках («фамилия», «имя», «отчество», «оклад») обязательно должны иметь какоето значение, а в поле «ученая степень» может отсутствовать значение.

В том случае если речь идёт об ограничениях целостности, относящихся к кортежу, то имеется в виду ограничение на значение всей строки,

рассматриваемой как единое целое, или ограничения на соотношения значений отдельных полей в пределах одной строки. Естественным ограничением является требование уникальности каждой строки таблицы.

По определению в реляционном отношении не может быть одинаковых кортежей. Однако не все реляционные СУБД обеспечивают автоматическое соблюдение этого ограничения. Пример ограничения на соотношения полей внутри одного кортежа: значение поля «дата возврата» не должно превышать «дату заказа».

Если встроенные возможности СУБД не позволяет контролировать подобные ограничения целостности, то надо написать универсальную программу (создать процедуру), позволяющую это делать.

Видом ограничения является также *запрет на обновление*. Он может относиться как к отдельному полю, так и к записи или целой таблице. В некоторых СУБД существует запрет на корректировку ключевого поля.

Запрет на обновление может быть связан с технологией обработки данных или со спецификой предметной области. Например, если описывается объект «Личность», то его атрибуты дата и место рождения являются постоянными и не могут меняться. Для соответствующих полей в БД рекомендуется задавать запрет на обновление. При этом запрет на обновление может относиться как к отдельному полю, так и ко всей записи и или файлу.

3.3.7 Запросы к базе данных

В шестом подразделе практической части курсового проекта по дисциплине «Управление данными» следует привести не менее 10 запросов всех типов, реализуемых средствами СУБД и средствами языка SQL. Составить запросы с коррелированными и некоррелированными подзапросами. Предварительно они должны быть сформулированы на естественном языке и выражены в терминах реляционной алгебры или реляционного исчисления. Примеры возможных запросов к базе данных и способы их реализации приведены в приложении 3.

3.3.8 Разработка механизмов защиты данных от несанкционированного доступа

При разработке механизмов защиты данных в разрабатываемой базе необходимо проанализировать состав обслуживающего персонала, который будет работать с ней. Рассмотреть привилегии, предоставляемые пользователям для работы с базой данных, таблицами, представлениями.

Целью управления доступом к объектам базы данных является ограничение действий, проводимых зарегистрированным в системе пользователем. При определении прав доступа администратор БД решает, какие действия в системе может выполнять пользователь, а также какие операции разрешено выполнять приложениям, запущенным от имени

пользователя. Управление доступом предназначено для предотвращения действий пользователя, которые способны нанести вред базе данных.

Под управлением доступом понимается возможность субъекта (сущность, определяющая пользователя при работе в системе) проводить действия над объектом (как базы данных целиком, так и ее компонентов). Различаются три метода управления доступом: дискреционный, обязательный, ролевой.

Дискреционный контроль предполагает, что владелец объекта сам может указывать, кто имеет доступ к объекту и вид этого доступа.

Для управления доступа к объектам дискреционный контроль доступа использует идентификационную информацию объекта и список доступа, который содержит субъекты и ассоциированные с ними типы доступа. Во время запроса доступа к объекту, система производит поиск субъекта в списке прав доступа объекта, и, в случае, если субъект там присутствует и разрешённый тип доступа включает требуемый тип, разрешает доступ. В противном случае запрещает доступ.

Благодаря своей гибкости, дискреционный контроль доступа широко используется. Одним из значительных недостатков данного вида контроля является отсутствие полной гарантии того, что информация не станет доступна другим субъектам, которые не имеют к ней доступа. Причина этого кроется в том, что субъект, имеющий право чтения информации может без уведомления владельца объекта передать её другим субъектам, не имеющим такого права. Дискреционная модель контроля доступа не накладывает ограничений на дальнейшее распространение информации после того, как субъект её получил.

Помимо этого, к недостаткам можно отнести ещё одну особенность дискреционной модели контроля доступа: объекты в системе принадлежат субъектам, которые настраивают доступ к ним для других. Но на практике, в большинстве случаев, данные в системе принадлежат всей системе, а не отдельным субъектам. Информационная система является наиболее распространённым примером.

Различают закрытую и открытую систему дискреционного контроля. Под закрытой системой понимается такая система, в которой изначально объект никому не доступен и список разрешений описывается в списке прав доступа, а открытая – та, в которой к объектам все имеют полный доступ, а в списке доступа описывается список ограничений.

Обязательный контроль доступа осуществляет управление доступом, опираясь на элементы самой системы. Каждому из элементов системы назначается уровень безопасности, который описывает важность этого объекта и ущерб, причинённый при разглашении информации в этом объекте. Уровень доверия субъекту является уровнем его безопасности. Все уровни безопасности являются членами определённой иерархии, то есть каждый уровень безопасности включает себя и уровни, находящиеся ниже.

Доступ субъекта к структурному элементу базы данных (таблице, запросу и т. д.), а также самой базе данных предоставляется в случае выполнения определённого условия отношения между уровнями безопасности субъекта и объекта. Доступ на выполнение операции даётся при выполнении определённых условий. Доступ на чтение даётся в случае, если уровень безопасности субъекта включает в себя уровень безопасности объекта. Доступ на запись даётся в случае, если уровень безопасности субъекта включается в уровень безопасности объекта. При выполнении этих условий гарантируется, что данные высокоуровневых объектов не попадут в низкоуровневые объекты.

Однако в такой модели существуют два момента, ставящие под вопрос её непротиворечивость.

1. Пользователь нижнего уровня может записывать информацию в объекты верхних уровней. Он может переписать существующий объект своим собственным и информация будет потеряна. Такой недостаток можно устранить, запретив запись на более верхние уровни. При такой схеме доступ на чтение будет даваться, если уровень безопасности субъекта включается в себя уровень безопасности объекта, а доступ на чтение будет даваться в случае, если уровень безопасности субъекта будет равняться уровню безопасности объекта.
2. Пользователи с более высоким уровнем не могут изменять объекты с более низки уровнем. Этот недостаток можно устранить, позволив пользователю при доступе к объектам выступать от имени субъектов с различными уровнями.

Часто, кроме обеспечения безопасности, также требуется обеспечение её достоверности. Чем выше уровень доверия объекта, тем выше его достоверность. Чем выше уровень безопасности субъекта, тем достовернее информация, которую он вносит в систему. Для такой модели описанные выше правила должны быть изменены: доступ на запись даётся в случае, если уровень безопасности субъекта включает в себя уровень безопасности объекта, а доступ чтение даётся в случае, если уровень безопасности субъекта включается в уровень безопасности объекта. То есть критерии просто поменялись местами.

Вместе с использованием уровней безопасности можно использовать категории. В таком случае, кроме уровня безопасности, каждому объекту и субъекту можно назначить список категорий, к которым он относится. Категории объекта используются для описания тех областей, в которых он используется. Категории субъекта описывают области, в которых он работает. Такая система позволит детальнее управлять доступом в системе.

В классических моделях разграничения доступа права на выполнение определённых операций над объектом должны быть прописаны для каждого пользователя или группы пользователей. Благодаря разделению понятий «роль» и «пользователь», можно разбить задачу на две части: определение роли пользователя и определение прав доступа к объекту для роли. Такой

подход позволяет упростить процесс администрирования, так как при изменении области ответственности пользователя, требуется лишь убрать у него старые роли и назначить новые, соответствующие его текущим обязанностям. Эта же процедура потребует массу усилий при переназначении новых прав, если права доступа определялись напрямую между пользователями и объектами.

Посредством построения иерархии ролей, систему ролей можно настроить так, чтобы она точнее отображала реальные бизнес процессы. Каждая роль, вместе со своими привилегиями, может ещё наследовать привилегии других ролей. Это тоже упростит администрирование системы

Ролевая модель позволяет пользователю регистрироваться в системе с наименьшей ролью для выполнения требуемых задач. Пользователям, у которых множество ролей, не всегда требуются все привилегии для выполнения определённой задачи.

Принцип наименьшей привилегии обеспечивает достоверность данных в системе. Необходимо давать пользователю только те разрешённые привилегии, которые ему нужны для выполнения определённой задачи. Для этого следует выяснить цель задачи, привилегии, необходимые для её выполнения и ограничить этим набором привилегии пользователя. Запрещение привилегий, не требуемых для выполнения текущей задачи, не позволит обойти политику безопасности системы.

Разделение обязанностей также является одним из важных принципов в системе управления доступом. Не редки ситуации, когда ряд определённых действий не может быть выполнен одним и тем же человеком для предотвращения мошенничеств, например, операции создания и подтверждения платежа. Система ролевого управления доступом позволит решить эту задачу без особых затруднений.

Модель состоит из сущностей пользователя, роли и привилегии, где пользователь – человек либо программа запущенная им, роль – вид деятельности человека в организации, а привилегия – разрешение доступа к объектам системы. У пользователя может быть несколько ролей, а одна роль может принадлежать нескольким пользователям. Также и несколько привилегий могут принадлежать одной роли, а несколько ролей – иметь одну привилегию.

Сессия, которую активизирует пользователь для выполнений задачи, позволяет отнести пользователя к множеству ролей. В этот момент система определяет роли и привилегии, требующиеся пользователю для выполнения задачи, и запрещает все остальные.

В проектируемой базе данных «Прокат» можно рассматривать следующие механизмы доступа как к базе данных целиком, так к ее компонентам:

- парольная защита:
- разграничение прав пользователей (ролевая защита).

Пользователями базы данных «Прокат» могут быть менеджер первого звена, менеджер по работе с клиентами и администратор базы данных.

Из всех перечисленных пользователей наиболее широким спектром полномочий обладает администратор, который может выполнять все операции с базой данных и ее компонентами.

3.3.9 Требования к техническому обеспечению

В этом разделе следует сформулировать требования к системе БД, например, определить быстродействие системы, требования к аппаратному обеспечению, объему оперативной памяти и жесткого диска.

При описании технического обеспечения базы данных необходимо соблюдать следующие требования:

- к видам технических средств, в том числе к видам комплексов технических средств, программно-технических комплексов и других комплектующих изделий, допустимых к использованию;
- к функциональным, конструктивным и эксплуатационным характеристикам средств технического обеспечения системы.

Техническое обеспечение системы должно максимально и наиболее эффективным образом использовать существующие в фирме средства компьютерной и вычислительной техники.

Для эффективного функционирования базы данных «Прокат» необходимы персональные компьютеры, являющиеся рабочими местами менеджера первого звена, менеджеров по работе с клиентами и администратора, которые могут обладать следующими техническими характеристиками:

– процессор – Intel Core i3 380M;
– объем оперативной памяти – 6 Гб;
- жесткий диск объемом не менее 500 Гб;
- дисковая подсистема – 40 Гб;
- устройство чтения компакт-дисков (DVD-ROM); - сетевой адаптер – 100 Мбит.

3.3.10 Инструкция по использованию базы данных

При разработке инструкции для пользователя базы данных необходимо описать, как производится вызов программы, и как характеризуются ее основные функции. В этом разделе приводятся справочные сведения о разработанной базе данных, описание экранных форм и отчетов.

Руководство пользователя — один из основных программных документов. Основная задача документа состоит в том, чтобы обеспечить пользователям возможность самостоятельно решать все основные задачи, на которые нацелена программа.

Руководство пользователя должно обеспечить пользователям возможность в полном объеме самостоятельно освоить и применять программу и регламентируется стандартами ГОСТ 34.201-89, РД 50-34.69890, IEEE 1063-2001.

Руководство пользователя содержит полное описание программы с точки зрения ее целевого применения. В руководстве пользователя обязательно должны быть описаны:

- назначение базы данных;
- основные задачи и возможности;
- способ отражения предметной области в базе данных;
- пользовательский интерфейс базы данных;
- порядок решения основных пользовательских задач;
- все функции программы и порядок их применения;
- пользовательская настройка программы;
- проблемы при использовании и способы их решения.

При документировании небольших программ в руководство пользователя часто включают инструкции по установке, настройке, администрированию, обновлению и прочему обслуживанию программы.

В зависимости от особенностей программы и целевой аудитории руководство пользователя по способу изложения материала может приближаться к учебнику или, наоборот, к справочнику. Порядок изложения материала в руководстве пользователя определяется *пользовательской перспективой* программы.

Если программа представляет собой инструмент, позволяющий решать практические задачи из некоторого конечного набора, то в руководстве приводят типовые процедуры решения каждой из них. Каждое из этих действий можно разложить на последовательные элементарные шаги, во всяком случае, для типичных ситуаций. Руководство пользователя, построенное по принципу пользовательских задач, напоминает учебник, хотя, как правило, лишено присущего учебникам методического аппарата: проверочных заданий, вопросов, упражнений.

Если программа представляет собой среду, в пределах которой пользователь может решать задачи, поставленные им самостоятельно, руководство пользователя должно быть ближе к справочнику.

Если программа является инструментом, с помощью которого пользователь контролирует состояние того или иного объекта, например, промышленной установки, то руководство пользователя строится по принципу таблицы: сообщение программы - реакция или возможные реакции пользователя.

Если пользователь применяет программу для решения задач в нетривиальных предметных областях, в руководство пользователя настоятельно рекомендуется включить концептуальный раздел. В нем должен быть описан реализованный в программе способ представления объектов реального мира, чтобы пользователь хорошо понимал, с какими из них и на каком уровне абстракции он может работать.

В каждом конкретном случае структура руководства будет в основном определяться особенностями описываемой программы. Обычно структура руководства пользователя содержит следующие разделы:

Общие сведения.
Установка и первоначальная настройка.
Основные понятия и определения.
Интерфейс пользователя.
Работа с программой.
Пользовательская настройка.
Сообщения об ошибках.

Единый раздел «Работа с программой» часто заменяют несколькими последовательными разделами, описывающими крупные группы пользовательских задач или функций.

3.3.11 Заключение

Заключение к курсовому проекту должно содержать выводы о проделанной работе. Примерный объем заключения 1 – 2 страницы.

3.3.12 Список используемой литературы

В списке используемой литературы приводятся данные об литературных источниках информации (монографии, учебники, статьи, справочники и т. д.), а также ссылки на электронные издания по тематике курсовой работе и Internet-адреса соответствующих сайтов. Содержание списка нумеруется по мере упоминания в пояснительной записке. Пример оформления литературных источников приведен в разделе «Оформление пояснительной записки».

3.3.13 Приложения

Пояснительная записка к курсовому проекту по дисциплине «Управление данными» может содержать перечисленные приложения:

- словарь терминов, используемых в данной предметной области;
- формы входных документов;
- формы выходных документов;
- экранные формы;
- распечатки реализованных запросов;
- распечатки реализованных отчетов; - тексты программ.

4. ОБЩИЕ ТРЕБОВАНИЯ К НАПИСАНИЮ И ОФОРМЛЕНИЮ РАБОТЫ

Выполнение курсовой работы и подготовка его к защите включает этапы:

1) выбор и оформление темы курсовой работы;
2) разработка и утверждение задания на курсовую работу;
3) реализация этапов проектирования информационной подсистемы;
4) оформление пояснительной записки к курсовой работе.

Тематика курсовых работ должна быть актуальной, соответствовать современному состоянию и перспективам развития информационных систем и средств сбора, передачи и отображения информации в различных предметных областях. При выборе темы необходимо учитывать следующие положения:

1. Актуальность проблемы и ее значимость для практической деятельности базовой организации.

2. Интересы, склонности, имеющийся задел научно-исследовательской работы студента во время предшествующего обучения, а также перспектив его будущей профессиональной деятельности.

3. Возможность использования полученных результатов и дальнейшего развития темы при выполнении дипломного проектирования.

4. Наличие специальной научной литературы для теоретического обоснования проблемы.

Тема курсового проекта должна быть краткой, отражать его основное содержание. В названии темы должны быть указаны область деятельности и объект, на которые ориентирован проект. Закрепление темы курсового проекта осуществляется по предоставлению кафедры и утверждается приказами по директорату. Темы курсовых работ (проектов) утверждаются на заседании кафедры, ведущей дисциплины , по которым учебными планами предусмотрены курсовые работы (проекты), в течение 2-х недель после начала семестра. Выписка из протокола заседания кафедры передается в дирекцию института, где формируется распоряжение об утверждении тем курсовых работ (проектов). В течение 10 дней после выхода распоряжения об утверждении тем курсовых работ (проектов) специалист по учебно-методической работе дирекции вносит темы работ в автоматизированную базу университета.

После этого руководителем разрабатывается задание на курсовой проект, включающее аналитическую и практическую часть разработки, исходные данные, описывающие предметную область для которой нужно разработать локальную базу данных.

Руководитель проекта регулярно, не реже одного раза в месяц, проводит консультации, на которых рекомендует студенту необходимую литературу, справочные и методические материалы, стандарты и типовые решения, а также проверяет ход ее выполнения, сохраняя за студентом полную самостоятельность.

Задание на курсовой проект оформляется в соответствии с требованиями, утвержденными в ВУЗе (приложение 1). Титульный лист оформляется по образцу, приведенному в приложении 2.

4.1 Оформление пояснительной записки

Пояснительная записка к курсовому проекту относится к текстовым документам и должна соответствовать требованиям:

- ГОСТ 2.105-95 ЕСКД. Общие требования к текстовым документам;
- ГОСТ 2.106-96 ЕСКД «Текстовые документы»

Содержание пояснительной записки к курсовому проекту следует делить на разделы, подразделы и пункты. В случае значительного объема и разнохарактерного содержания записку следует делить на части (ГОСТ 2.105-95).

Каждый из разделов должен иметь свой порядковый номер в пределах всей пояснительной записки. Нумерация разделов осуществляется арабскими цифрами без точки, начиная с абзацного отступа. Нумерация подразделов осуществляется внутри соответствующего раздела. Номер подраздела состоит из номеров раздела и подраздела, разделенных точкой. В конце номера пункта точка не ставится, например:

1 Типы и основные размеры

1.1

1.2 Нумерация пунктов первого раздела документа

1.3

2 Технические требования

2.1

2.2 Нумерация пунктов второго раздела документа

2.3

Текст каждого подраздела должен быть разбит по смысловому значению на абзацы. Параметры форматирования абзацев текста: выравнивание – по ширине, отступ первой строки – 1,25 см, межстрочный интервал – полуторный. Текст пояснительной записки распечатывается шрифтом Times New Roman, обычного начертания и высотой 14 пунктов.

Если документ имеет подразделы, то нумерация пунктов должна быть в пределах подраздела и номер пункта должен состоять из номеров раздела, подраздела и пункта, разделенных точками, например:

3 Методы испытаний

3.1 Аппараты, материалы и реактивы

3.1.1

3.1.2 Нумерация пунктов первого подраздела третьего раздела

3.1.3

документа

3.2 Подготовка к испытанию

3.2.1

3.2.2 Нумерация пунктов второго подраздела третьего раздела

3.2.3

документа

Если раздел или подраздел состоит из одного пункта, он также нумеруется.

Если текст документа подразделяется только на пункты, они тоже нумеруются порядковыми номерами в пределах документа. Пункты, при необходимости, могут быть разбиты на подпункты, которые должны иметь порядковую нумерацию в пределах каждого подпункта, например: 4.2.1.1, 4.2.1.2, 4.2.1.3, и т. д. Внутри пунктов или подпунктов могут быть приведены перечисления. Перед каждой позицией перечисления следует ставить дефис

или при необходимости ссылки в тексте документа на одно из перечислений, строчную букву, после которой ставится скобка. Для дальнейшей детализации перечислений необходимо использовать арабские цифры, после которых ставится скобка, запись производится с абзацного отступа, как показано на примере.

Пример

а) _____

б) _____

1) _____

2) _____

в) _____

Каждый пункт, подпункт и перечисление записывают с абзацного отступа.

Разделы, подразделы должны иметь заголовки. Пункты, как правило, заголовков не имеют. Заголовки должны четко и кратко отражать содержание разделов, подразделов. Заголовок следует печатать с прописной буквы без точки в конце, не подчеркивая. Переносы слов в заголовках не допускаются. Если заголовок состоит из двух предложений, их разделяют точкой. Расстояние между заголовком и текстом при электронном наборе должно составлять два полуторных интервала. Каждый раздел текстового документа рекомендуется начинать с нового листа (страницы).

Слово «Содержание» записывают в виде заголовка (симметрично тексту) с прописной буквы. Наименования, включенные в содержание, записывают строчными буквами, начиная с прописной буквы.

В конце текстового документа перед листом регистрации изменений допускается приводить список литературы, которая была использована при его составлении. Выполнение списка и ссылки на него в тексте – по ГОСТ 7.32. Список литературы включают в содержание документа.

Нумерация страниц документа и приложений, входящих в состав этого документа, должна быть сквозная.

Текст пояснительной записки должен быть кратким, четким и не допускать различных толкований.

При изложении обязательных требований в тексте пояснительной записки необходимо применяться слова «должен», «разрешается только», «следует», «необходимо», «требуется чтобы», «не допускается», «запрещается», «не следует». При изложении в ПЗ других положений следует применять слова – «могут быть», «как правило», «при необходимости», «может быть», «в случае» и т.д.

Формулы в тексте пояснительной записки нумеруют сквозной нумерацией арабскими цифрами, например, (1), (2) и т. д. Допускается нумеровать формулы в пределах раздела. Формулы в приложении, например в приложении А, нумеруют следующим образом (А.1), (А.2) и т. д. Допускается нумерация формул в пределах раздела, например (3.1) – раздел 3

формула 1. Ссылки на формулы в тексте пояснительной записки рекомендуется выполнять следующим образом:…как показано в формуле (5) …;…как следует из соотношений (2) – (6) …

Примечания помещают непосредственно после текста, графического материала или таблиц, к которым относятся эти примечания, и печатают с прописной буквы с абзаца. Если примечание одно, то после слова «Примечание» ставится тире и примечание тоже печатается с прописной буквы. Одно примечание не нумеруется. Несколько примечаний нумеруют по порядку арабскими цифрами. Примечания к таблице помещают в конце таблицы под линией, обозначающей окончание таблицы.

Иллюстрации могут быть размещены как по тексту пояснительной записки, так и в конце нее. Иллюстрации следует нумеровать арабскими цифрами сквозной нумерацией. Если рисунок один, то он обозначается Рисунок 1. Иллюстрация в приложении обозначается Рисунок А.3 (приложение А, рисунок 3). Допускается нумеровать иллюстрации в пределах раздела, например, Рисунок 1.1 (первый раздел, рисунок один), Рисунок 3.5 (третий раздел. рисунок пять). Ссылки на рисунки в тексте пояснительной записки выполняются следующим образом: …в соответствии с рисунком 2…., …в соответствии с рисунком 1.2…. Подрисуночный текст оформляется следующим образом: Рисунок 1 – Детали прибора В конце подрисуночного текста точка не ставится.

Приложения в пояснительной записке являются информационными (носят справочный характер). Каждое приложение начинается с новой страницы с указанием наверху посредине страницы слова «Приложение» и его обозначения. Под ним в скобках для информационного приложения пишут слово «рекомендуемое» или «справочное». Приложение должно иметь заголовок, который записывают симметрично относительно текста с прописной буквы отдельной строкой. Приложения обозначают заглавными буквами русского алфавита, начиная с А, за исключением букв Ё, З, Й, О, Ч, Ь, Ы, Ъ. После слова «Приложение» следует буква, обозначающая его последовательность. Допускается обозначение приложений буквами латинского алфавита, за исключением букв I и O. В случае полного использования букв русского и латинского алфавитов допускается обозначать приложения арабскими цифрами. Если в пояснительной записке одно приложение, оно обозначается «ПРИЛОЖЕНИЕ А». Приложения выполняются на листах формата А4 (допускаются форматы А3, А4×3, А4×4, А2 и А1).

Цифровой материал оформляют в виде таблиц. Пример оформления таблицы приведен ниже.

Графу «Номер по порядку» в таблицу включать не допускается. Нумерация граф таблицы арабскими цифрами допускается в случаях: 1) когда в тексте есть ссылки на них; 2) при делении таблицы на части;3) при перенесении части на следующую страницу. Если в тексте пояснительной записки одна таблица, она обозначается «Таблица 1» или «Таблица А.1» (в

приложении А). Допускается нумерация таблиц в пределах раздела, например, «Таблица 1.2».

Ссылки на таблицы в тексте пояснительной записки: … как показано в таблице 1…, … как следует из таблицы 2.5…

Таблица 5 – Сводная ведомость

(конец страницы)

Продолжение таблицы 5

5. ПОСЛЕДОВАТЕЛЬНОСТЬ ВЫПОЛНЕНИЯ ЗАДАНИЯ

Содержание основных этапов подготовки курсового проекта по дисциплине «Управление данными» представлено в таблице 5.1.

Таблица 5.1 – Основные этапы подготовки курсового проекта

Наименование этапа работы	Срок выполнения
Получение задания на курсовое проектирование	2 неделя семестра
Выполнение теоретического задания	3-9 недели семестра
Предварительное обследование предметной области и оформление его результатов	5 неделя семестра
Составление технического задания на разработку базы данных	6 неделя семестра
Инфологическое проектирование	7 неделя семестра
Преобразование ER-модели в реляционную	8 неделя семестра
Даталогическое проектирование, загрузка базы данных, тестирование и отладка	9 неделя семестра
Разработка запросов	10-12 недели семестра
Создание справочной системы	13 неделя семестра
Создание входных и выходных форм	14 неделя семестра

Оформление пояснительной записки	15 неделя семестра
Защита проекта	17 неделя семестра

6. КРИТЕРИИ ОЦЕНИВАНИЯ ПРОЕКТА

Оценка «отлично» при защите курсового проекта выставляется в том случае, если он свободно ориентируется в исследуемой предметной области и может дать четкие развернутые ответы на все поставленные преподавателем вопросы.

Оценка «хорошо» при защите курсового проекта может быть выставлена, если студент хорошо ориентируется в предметной области, рассмотренной в курсовом проекте, и может дать ответы на поставленные вопросы.

Оценка «удовлетворительно» может быть выставлена, если студент плохо ориентируется в предметной области, для которой выполнен курсовой проект, и им были даны ответы не на все вопросы, сформулированные в ходе защиты проекта, преподавателем.

Оценка «неудовлетворительно» при защите курсового проекта может быть выставлена, если:

1) студент вообще не ориентируется в той предметной области, для которой выполнен курсовой проект;
2) не может дать ответы на вопросы, заданные преподавателем;
3) две курсовые работы выполнены на одну и ту же тему.

Если студент на защите получает оценку «неудовлетворительно», то тема курсового проекта изменяется. Изменение темы курсового проекта происходит и в том случае, когда несколько пояснительных записок абсолютно одинаковы.

7. ПОРЯДОК ЗАЩИТЫ ПРОЕКТА

Пояснительная записка сдается на проверку руководителю работы в срок не менее чем за 10 дней до защиты. После проверки руководитель либо допускает студента к защите, либо возвращает проект на доработку.
Программная реализация обязательно прилагается на диске и демонстрируется руководителю.

Порядок защиты курсового проекта и состав комиссии утверждается на заседании кафедры. Студент делает доклад (около 5 минут), в котором кратко излагает результаты проектирования, демонстрирует графическую часть проекта и разработанную программную реализацию.

СПИСОК РЕКОМЕНДУЕМОЙ ЛИТЕРАТУРЫ

Основная литература

1. Швецов, В.И. Базы данных Электронный ресурс : учебное пособие / В.И. Швецов. - Базы данных,2019-12-01. - Москва : Интернет-Университет Информационных Технологий (ИНТУИТ), 2016. - 218 c. - Книга находится в базовой версии ЭБС IPRbooks.
2. Крис, Фиайли . SQL Электронный ресурс / Фиайли Крис ; пер. А. В. Хаванов. - SQL,2019-04-19. - Саратов : Профобразование, 2017. - 452 c. - Книга находится в премиум-версии ЭБС IPR BOOKS. - ISBN 978-5-44880103-7.
3. Дроздова, В. И. (Северо-Кавказский федеральный университет). Управление данными : учеб. пособие : Направление подготовки 320400.62-Информационные системы и технологии. Бакалавриат / В. И.Дроздова ; Сев.-Кав. федер. ун-т- Ставрополь : СКФУ, 2013. - 170 с.

Дополнительная литература

1.Агальцов, В. П. Базы данных : учебник / В. П. Агальцов, Кн.2, Распределенные и удаленные базы данных. - М. : Форум : ИНФРА-М, 2014. - 272 с. : ил. - (Высшее образование). - Гриф: Доп. УМО. - Библиогр.: с. 260. - ISBN 978-5-8199-0394-0. - ISBN 978-5-16-003526-0

2.Карпова, И. П. Базы данных : курс лекций и материалы для практических занятий : учебное пособие / И. П. Карпова. - СПб. [и др.] : Питер, 2013. - 240 с. : ил., табл. ; 24 см. - (Учебное пособие). - Гриф: Рек. - Библиогр.: с. 233-234. - ISBN 978-5-496-00546-3

3.Кузнецов, С.Д.
    Введение в реляционные базы данных Электронный ресурс : учебное пособие / С.Д. Кузнецов. - Введение в реляционные базы данных,2021-01-23. - Москва : Интернет-Университет Информационных Технологий (ИНТУИТ), 2016. - 247 c. - Книга находится в базовой версии ЭБС IPRbooks. - ISBN 5-9556-00028-0

Internet-ресурсы

http://www.intuit.ru – национальный открытый университет «Интуит»
http://www.citforum.ru – центр информационных технологий «Море(!) аналитической информации!»

ПРИЛОЖЕНИЕ 1

УТВЕРЖДАЮ
Заведующий кафедрой
_____ (название кафедры) _____
(ФИО)

Институт _____
Кафедра _____
Направление (специальность) _____

Профиль (специализация) _____

ЗАДАНИЕ на курсовую работу (проект)
студента _____
(фамилия, имя, отчество)
по дисциплине _____

1. Тема работы
_____
_____

2. Цель
_____
_____

3. Задачи_____
_
_____
_____
_____
_____
_____

4. Перечень подлежащих разработке вопросов:
а) по теоретической части_____
_____
_____
_____

б) по аналитической части _____
_____

_____
_____

5. Исходные данные:
а) по литературным источникам_____
_____
_____

б) по вариантам, разработанным преподавателем_____
_____
_____

в)
иное_____
_____
_____

6. Список рекомендуемой литературы
_____
_____
_____
_____
_____
_____

7. Контрольные сроки представления отдельных разделов курсового проекта: 25 % - _____“___” _____ 20_ г.
50 % - _____“___” _____ 20_ г.
75 % - _____“___” _____ 20_ г.
100 % - _____“___” _____ 20_ г.

8. Срок защиты студентом курсового проекта “___” _____ 20_ г.

Дата выдачи задания “___” _____ 20_ г.

Руководитель курсового проекта _____ _____ _____
(ученая степень, звание) (личная подпись) (инициалы, фамилия)

Задание принял(а) к исполнению студент(ка) _____ формы обучения ___ курса ___ группы _____ _____
(личная подпись) (инициалы, фамилия)

ПРИЛОЖЕНИЕ 2

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ И РОССИЙСКОЙ ФЕДЕРАЦИИ
ФГАОУ ВО «СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»
Пятигорский институт (филиал) СКФУ
КАФЕДРА Систем управления и информационных технологий

КУРСОВОЙ ПРОЕКТ по дисциплине
«Управление данными» на тему:
«_____ »

Выполнил:
студент__ курса группы _ направления_____
_____ очной формы обучения
Фамилия, имя, отчество

_____
(подпись)

Руководитель работы:

_____
(ФИО, должность, кафедра)

Работа допущена к защите_____ _____
(подпись руководителя) (дата)

Работа выполнена и защищена с оценкой _____ Дата защиты_____

Члены комиссии: _____ _____ _____
(должность) (подпись) (И.О. Фамилия) _____
_____ _____
_____ _____ _____
_____ _____ _____

Пятигорск, 20___ г.

ПРИЛОЖЕНИЕ 3

1 Безусловная выборка значений

В СУБД Visual FoxPro вводить SQL-запросы можно непосредственно в командном окне (Command window) или в окне дизайнера запросов (Query Designer). Для формирования запросов на языке **SQL** используется конструкция **SELECT**. Результатом выполнения запроса является таблица, которая хранится во временном буфере базы данных. Выбранные данные можно использовать для просмотра, формирования графиков или печати отчетов. Длина строки на языке **SQL** до 255 символов, регистр не имеет значения. Синтаксис команды **SELECT:**

SELECT [ALL DISTINCT] список выбираемых полей
FROM список таблиц
[WHERE условие выборки или соединения]
[GROUP BY список полей по условию группировки
[HAVING условие выборки группы]]
[ORDER BY список полей, по которым упорядочить вывод]

При формировании запросов можно использовать уточненные имена полей (например, CUSTOMER.CUSTOMERNO, т.е. Имя_таблицы.
Имя_поля).
Ключевое слово **ALL** подразумевается по умолчанию.

Для выборки всех полей таблицы в том же порядке, что и в таблице, используется следующая форма запроса:

SELECT*FROM таблица.

Запрос вида

SELECT*FROM R1, R2

соответствует декартову произведению таблиц **R1** и **R2**, т. е **R=R1x R2.**

Запрос

SELECT R1.A, R2.B FROM R1, R2

соответствует проекции декартова произведения таблиц **R1** и **R2** на столбцы **A** из таблицы **R1** и **B** – из таблицы **R2**, то есть $\mathbf{R = \pi_{R1.A,\ R2.B}(R1x\ R2).}$

2 Простая выборка

При использовании механизма простой выборки предполагается, что в результате ее выполнения на экран будет выведен некоторый диапазон значений. Например, результатом выполнения запроса

SELECT kod FROM R1

является столбец **kod** из таблицы **R1**.

При выполнении данного запроса в результат выборки будут включены все дубликаты строк.

3 Выборка уникальных значений

Чтобы исключить дубликаты строк из результата выборки используется ключевое слово DISTINCT. Примером запроса, исключающего дублирование записей, является

SELECT DISTINCT kod FROM R1

4 Выборка вычисляемых значений

При выполнении запросов в СУБД **Visual FoxPro** может осуществляться не только выбор из таблицы ранее введенных значений, но и получение данных, отсутствующих в исходной таблице. Для выборки вычисляемых значений в **Visual FoxPro** имеются встроенные функции и арифметические операторы. При организации выборки информации вычисления могут производится по одному или нескольким полям исходной таблицы.

Чтобы включить в запрос функцию поля или выражение, необходимо:

1) в окне конструктора запросов активизировать вкладку **Fields**;
2) с помощью манипулятора «мышь» или клавиши **Tab** перейти в поле

«Functions and expressions»

3) в этом поле ввести выражение для вычисления или нажать кнопку вызова построителя выражения, расположенную с правой стороны поля,
4) в окне диалога **Expression Builder** создать выражение для вычисляемого поля.
5) нажать кнопку **Add** для переноса данного выражения в список выходных полей запроса.

Пример 4.1. Пусть в некоторой базе данных содержится таблица **Товары**, в которой цена (**UNITPRICE)** указана без учёта налога на добавленную стоимость (НДС). Поля таблицы **Товары** представлены в таблице 4.1. Тип переменной **UNITPRICE – Currency.**

Таблица 4.1 – Поля таблицы **Товары**

Kod1	**UNITPRICE**
1	1000.0000
2	2000.0000
3	10000.0000

Необходимо организовать выборку информации из таблицы **Товары,** указав цены с учетом налога на добавленную стоимость (18%).

SQL-запрос, соответствующий примеру 4.1:

SELECT Kod1, UNITPRICE * (1 + 0,18) AS ЦЕНА FROM Товары

Результат выполнения данного запроса представлен в таблице 4.2. Таблица 4.2 – Результат выборки из таблицы **Товары**

Kod1	**ЦЕНА**
1	1180.0000
2	2360.0000
3	11800.0000

В запросе после ключевого слова **AS** записано новое название столбца таблицы.

Применение конструкции **SELECT** в формировании запросов с вычисляемыми полями позволяет использовать в них не только арифметические выражения, но и простые имена полей. Кроме того, в результаты выборки можно добавить константы.

Если в SQL-запрос, соответствующий примеру 1, добавить строку **«Цена указана с учетом НДС»**, то он примет вид:

SELECT Kod1, UNITPRICE * (1 + 0,18) AS ЦЕНА, «Цена указана с учетом НДС» FROM Товары

Результат выполнения данного запроса представлен в таблице 3.

Таблица 4.3 – Результат выполнения запроса

Kod1	**ЦЕНА**	Цена указана с учетом НДС
1	1180.0000	Цена указана с учетом НДС
2	2360.0000	Цена указана с учетом НДС
3	11800.0000	Цена указана с учетом НДС

При выборке с помощью команды **SELECT** можно использовать агрегатные функции:

1) **MIN(X)** – вычисляет минимальное значение из множества X;
2) **MAX(X)** – вычисляет максимальное значение из множества X;
3) **AVG(X)** – вычисляет среднее арифметическое из множества значений X;
4) **SUM(X)** – вычисляет сумму значений множества X; 5) **COUNT(X)** – определяет число элементов множества X.

Примеры использования этих функций:

1) **SELECT COUNT(*) FROM GOODS** – создает выборку, состоящую из одной строки и одного поля, содержащего количество всех строк таблицы
GOODS (товары);
2) **SELECT MAX(Цена), MIN(Цена), AVG(Цена) FROM GOODS** – создает выборку, состоящую из одной строки и трех полей, содержащих минимальное значение цены, максимальное значение цены и её среднее значение.

5 Выборка с условием

Для задания условия выборки в SQL-запросе используется команда (ключевое слово) **WHERE**. Условие, следующее за ключевым словом **WHERE**, может включать:

1) арифметические операторы сравнения: **=, <>, > , <, >=, <=;**
2) логические операторы – **AND, OR, NOT**;
3) скобки, определяющие порядок вычислений.

При выполнении условия выборки числа сравниваются алгебраически: отрицательные числа считаются меньше, чем положительные, независимо от

их абсолютной величины. Строки сравниваются с их представлением в коде **ANSI**. При сравнении двух строк, имеющих разные длины, предварительно более короткая строка дополняется справа пробелами для того, чтобы строки имели одинаковую длину.

Пример 5.1. Пусть существует некоторая база данных, в которой имеется таблица **Table3**, содержащая информацию о поставщиках, покупателях, товарах. Предполагается, что покупатель может приобретать товар в кредит. Произвести из данной базы выбор всех кодов (**kod**) и фамилий (**NAME)** покупателей, которые находятся в Москве и имеют кредит(**CREDITLIMIT)** более 200 000.

С помощью языка SQL запрос, соответствующий примеру 1, можно представить в виде:

```
SELECT kod, NAME
FROM Table3
WHERE CITY=“Москва”
AND CREDITLIMIT>200000
```

6 Выборка с упорядочением

Вкладка **Ordered By** в окне дизайнера запросов позволяет управлять порядком расположения записей в результирующей таблице. Для упорядочивания выделите указателем (курсором) поля, которые будут определять порядок сортировки выбранных данных, и перенесите их последовательно в список **Ordering criteria**. Для каждого выбранного поля можно с помощью переключателя (кнопки) **Order options** установить критерий упорядочивания по возрастанию **(Ascending)** или по убыванию **(Descending)**.

Порядок сортировки записей результирующей таблицы определяется порядком следования полей в списке **Ordering criteria** и критерием упорядочивания отдельных полей.

Пример 6.1. Для таблицы **Table3** из примера 4.1 необходимо произвести выбор кода (**kod**), имени (**NAME)** и суммы кредита (**CREDITLIMIT)** всех покупателей, проживающих в Ставрополе, расположив их в порядке убывания.

SQL-запрос, реализующий пример 6.1, имеет вид:

```
SELECT kod, NAME, CREDITLIMIT FROM Table3
WHERE CITY=“Ставрополь”
ORDER BY CREDITLIMIT DESC
```

В выборках без указания критерия упорядочивания данных результирующая таблица будет упорядочена в соответствии с внутренними алгоритмами их осуществления. Их примера 6.1 следует, что результат выборки может быть организован в определенной последовательности. Упорядочение данных в выборке может осуществляться по любому полю результирующей таблицы:

имя поля[упорядочение]

[,имя поля [упорядочение]] ..., где аргумент **«упорядочение»** может принимать значение **ASC(возрастание)** или **DESC (убывание)**. По умолчанию устанавливается значение **ASC**. В качестве аргументов имя поля могут использоваться только поля результирующей таблицы. Поэтому недопустима следующая конструкция:

```
SELECT kod, NAME, CREDITLIMIT
FROM Table3
ORDER BY CITY
```

Для идентификации полей, по которым осуществляется упорядочивание, можно использовать не только наименования полей результирующей таблицы, но и их номера (номер поля указывает порядковую позицию данного поля в результирующей таблице запроса). Благодаря этому можно упорядочить результат на основе вычисляемых полей, которые не обладают именами. Например, результатом выполнения запроса:

```
SELECT Kod1, UNITPRICE * (1 + 0,18)
FROM Товары
```

ORDER BY 2 будет являться результирующая таблица, во второй столбец которой будет помещена информация о стоимости товаров с учетом НДС. Записи в выборке будут упорядочены по второму столбцу.

7 Выборка с использованием оператора BETWEEN

Для организации выборки информации из базы данных, принадлежащей некоторому диапазону, в **SQL**-запросах используются операторы: **«>(Больше)»; «<(Меньше)»** и **Between (Между)**. Выбор каждого из этих операторов осуществляется в окне мастера запросов из списка **Criteria (Критерий)**, расположенного во вкладке **Filter (Фильтр)**. Операторы **«>(Больше)»** и **«<(Меньше)»** используются в том случае, если задана только нижняя или верхняя граница диапазона, а оператор **Between (Между)** – если известны обе границы диапазона.

С помощью SQL-запроса, набранного в командном окне,

```
SELECT kod, NAME, UNITPRICE
FROM Table4
WHERE UNITPRICE BETWEEN 20000 AND 100000
```

на экран будет выведена таблица, содержащая сведения о товарах, стоимость которых (**UNITPRICE)** находится в диапазоне от 20000 до 100000 включительно. Оператор **BETWEEN** возвращает истинные значения и для граничных точек.

Кроме того, при организации выборки из таблицы значений, не принадлежащих некоторому диапазону, может быть использовано условие **NOT BETWEEN (не принадлежит диапазону между величинами)**, например:

```
SELECT kod, NAME, UNITPRICE
FROM Table4
WHERE UNITPRICE  NOT BETWEEN 20000 AND 100000
```

8 Выборка с использованием оператора IN (принадлежит)

СУБД **Visual FoxPro** позволяет при создании запросов формировать несколько условий. В том случае, если все задаваемые условия накладываются на одно поле, их можно разместить в одной строке. Для этих целей используется оператор **IN (принадлежит)**.

В частности, результатом выполнения SQL-запроса, приведенного ниже

```
SELECT kod, NAME, UNITPRICE
FROM Table4
```

WHERE UNITPRICE IN (100000, 200000, 500000) является таблица, содержащая информацию о товарах, цена которых равна 100000, 200000 или 500000.

Оператор **IN** является краткой записью условия, представляющего собой последовательность отдельных сравнений, соединенных между операторами **OR (или)**. Предыдущая конструкция **SELECT** эквивалентна следующей конструкции:

```
SELECT kod, NAME, UNITPRICE
FROM Table4
WHERE UNITPRICE=100000
OR UNITPRICE=200000
OR UNITPRICE=500000
```

Как и в случае с оператором **BETWEEN** можно также использовать конструкцию **NOT IN (не принадлежит)**, например:

```
SELECT kod, NAME, UNITPRICE
FROM Table4
WHERE UNITPRICE NOT IN (100000, 200000, 500000)
```

9 Выборка с использованием шаблонов

Использование шаблонов языка SQL расширяет возможности выборки информации из базы данных, для которой пользователь не помнит точные значения полей или названия полей длинные и пользователь хочет сократить время их набора. Например, с помощью запроса

```
SELECT kod, NAME, UNITPRICE
FROM Table4
WHERE NAME LIKE “T%”
```

пользователь может выбрать из базы данных все товары, наименование которых начинается с буквы “T”(таблица 9.1).

Таблица 9.1 – Результат выборки по шаблону

kod	NAME	UNITPRICE
2	Toshiba 1200	600.00
4	Turbo Pascal	200.00

Шаблоны в языке SQL описываются с помощью оператора **LIKE**, который может быть представлен в виде:

имя поля LIKE строковая константа

Результат выполнения оператора **LIKE** принимает значение «истина», если значение в указанном поле соответствует образцу, указанному аргументом «строковая константа», символы которой интерпретируются следующим образом:

1) _ (пробел или подчеркивание) – любой одиночный символ;
2) % (процент) – произвольная последовательность символов.
3) Все другие символы обозначают сами себя. Специальный смысл специальных символов отменяет знак «\».

Например, совокупность символов «**%**» означает процент, а не последовательность символов.

В приведенном примере конструкция **SELECT** будет осуществлять выборку записей из таблицы **Table4**, для которых значение в поле **NAME** начинается с буквы «Т» и содержит далее любую последовательность символов. Например, если из таблицы **TAB7** необходимо выбрать все строки, в которых поле string1 содержит «**+**», а предпоследняя буква «S», то SQLзапрос, реализующий данное условие примет вид:

SELECT * FROM TAB7 WHERE string1 LIKE «%+%S_».

10 Выборка из связанных таблиц

Способность «соединять» две или более таблицы в одну представляет собой одну из наиболее мощных возможностей реляционных баз данных.

В СУБД Visual FoxPro при создании многотабличного запроса в окно конструктора запросов добавляются все участвующие в выборке таблицы и определяются условия их объединения. Таблицу в окно конструктора запросов при организации выборки из нескольких связанных таблиц можно добавить одним из способов:

1) выполните команду **F10 → Query → Add Table**; 2) нажмите кнопку **Add Table**.

После этого на экран будет выведено диалоговое окно **Add Table or View**. В этом окне следует выбрать необходимые для формирования запроса таблицы, затем нажать кнопку **Add**. Таким образом, выбранные таблицы будут размещены в окне конструктора запросов. Если между участвующими в запросе таблицами в базе данных установлены постоянные отношения, то в окне конструктора запросов эта связь будет отображаться в виде линии, соединяющей таблицы, а на вкладке **Join** появится запись, содержащая условие объединения таблиц.

Простое соединение

Простое соединение как один способов выбора информации из нескольких связанных таблиц предполагает, что на выбираемую пользователем информацию не накладывается никаких дополнительных условий. Например, с помощью **SQL**-запроса

SELECT Tab8.kod2, Tab9.NAME
FROM Tab8, Tab9

WHERE Tab8.STOCK= Tab9.STOCK будет получен список кодов (kod2) и наименований (NAME) проданных покупателям товаров, представленный в таблице 10.1.

Таблица 10.1 – Результат выборки из двух таблиц

Kod2	NAME
2	BORLAND C++
10	PARADOX for Windows
18	Книга TURBO C++
20	Книга C#

В данном случае использованы уточнённые имена полей, т.е. ссылки на поля после ключевого слова **WHERE** уточнены именами содержащих их таблиц. Два поля **STOCK** являются ключами, с помощью которых установлены постоянные отношения между таблицами.

Соединение с дополнительным условием

При выполнении выборки информации из базы данных можно использовать механизм соединения с дополнительным условием, которое ограничивает выбираемую пользователем информацию. Использование выборки на основе соединения с дополнительным условием возможно только в том случае, если таблицы, включенные в запрос, связаны по одному и тому же ключевому полю. Например, для выбора из базы данных кодов и фамилий покупателей, которым проданы компьютеры Lenovo, служит SQL- запрос:

```
SELECT ORDSALE.CUSTOMERNO, ORDSALE.FIRSTNAME
FROM ORDSALE, GOODS
WHERE ORDSALE.STOCK=GOODS.STOCK
AND GOODS.NAME=“ Lenovo”
```

Результат выполнения данного запроса представлен в таблице 10.2. Таблица 10.2 – Результат выборки из двух таблиц с условием

ORDSALE.CUSTOMERNO	FIRSTNAME
2	Иванов
2	Иванов
11	Петров
16	Сидоров
19	Кузнецов

Соединение трех таблиц

Для выборки данных из трех таблиц в программном коде после ключевого слова **WHERE** необходимо указать два условия связи таблиц. Пример **SQL**-запроса для выбора информации из трех связанных таблиц**:** вывести на экран фамилии и имена всех покупателей, которые приобрели Lenovo.

```
SELECT DISTINCT CUSTOMER.FIRSTNAME,
CUSTOMER.LASTNAME
```

FROM CUSTOMER, ORDSALE, GOODS
WHERE CUSTOMER.CUSTOMERNO=ORDSALE.CUSTOMERNO
AND ORDSALE.STOCK=GOODS.STOCK
AND GOODS.NAME=“ Lenovo”

После выполнения данного запроса на экран будет выведена таблица 10.3.

Во многих случаях выборки из нескольких таблиц используются не только для ограничения выборки, но и для объединения данных из нескольких таблиц.

Таблица 10.3 – Результат выборки из трех таблиц

CUSTOMER.FIRSTNAME	CUSTOMER.LASTNAME
Иванов	Андрей
Петрова	Дарья
Сидоров	Максим

11 Использование группировки данных при организации запросов

Группировка данных при организации запросов используется для объединения нескольких строк, включаемых в запрос, в одну.

Для выполнения группировки данных в **SQL**-запросах используется оператор **GROUP BY**. Оператор **GROUP BY** перекомпоновывает данные, включаемые в результирующую таблицу. Указание на группировку данных в **SQL**-запросе осуществляется после ключевого слова **FROM**. Сам процесс группировки данных при выполнении выборки информации из базы данных в разделы или группы заключается в объединении в одну группу всех строк, которые имеют одно и то же значение поле, указанное после оператора **GROUP BY**.

Далее, к каждой группе применяется конструкция **SELECT**. Каждое из выражений, включаемое в конструкцию оператора **SELECT**, должно принимать единственное значение для группы. Это выражение может быть либо самим полем, указанным в операторе **GROUP BY**, либо арифметическим выражением, включающим это поле, либо константой, либо такой функцией как **SUM**, которая оперирует всеми значениями данного поля в группе и сводит эти значения к единственному значению.

Строки таблицы можно группировать по любой комбинации ее полей. Если поле, по значению которого осуществляется группировка, содержит какие-либо неопределенные значения, то каждое из них порождает отдельную группу.

Использование конструкции **GROUP BY** не предполагает одновременного применения в **SQL**-запросах и оператора **ORDER BY.** Для упорядочения результата, полученного после выполнения **SQL**-запроса, конструкцию **ORDER BY** <имя поля> необходимо разместить после оператора, указывающего на группировку данных.

Пусть требуется вычислить общий объем покупок для каждого товара, т. е. для каждого товара необходимо определить код этого товара и общий объем покупок. **SQL**-запрос, соответствующий данному условию, можно представить виде:

```
SELECT kod, SUM(QUANT)
FROM ORDSALE
GROUP BY kod
```

Использование группировки данных совместно с условием

Использование группировки при выборе информации из базы данных выводит на экран все поля, значения которых совпадают. Однако на практике эта особенность группировки затрудняет работу пользователя с базой данных, так как после выполнения запроса выводятся лишние записи, значения которых в данный момент не нужны. Для ограничения выборки с упорядочиванием в этом случае используется оператор **WHERE**.

Например, **SQL**-запрос

```
SELECT kod, SUM(QUANT)
FROM ORDSALE
WHERE CUSTOMERNO <> 23
```

GROUP BY kod служит для выборки информации о товаре, проданном покупателям кроме покупателя с кодом 23.

Строки, не удовлетворяющие условию **WHERE**, исключаются перед группировкой данных.

Конструкции **GROUP BY** свойственно ограничение, которое заключается в том, что она работает только на одном уровне. Невозможно разбить каждую из групп на группы более низкого уровня, а затем применить некоторую стандартную функцию, например, **SUM** или **AVERAGE** на каждом уровне группировки.

Использование HAVING

Оператор **HAVING** используется для ограничения записей, участвующих в группировке, его нельзя использовать отдельно от конструкции **GROUP BY**. Оператор **HAVING** используется для того, чтобы исключать группы так же, как **WHERE** используется для исключения записей. Выражение после конструкции **HAVING** должно принимать единственное для группы значение. В частности, результатом выполнения запроса

```
SELECT kod FROM ORDSALE GROUP BY kod HAVING COUNT(*)>1
```

является таблица, в которую включены коды товаров, приобретенных более чем одним покупателем.

12 Использование квантора существования в запросах

Квантор существования является понятием, заимствованным из формальной логики. Его смысл заключается в следующем: при задании квантора существования говорят о том, что существует некоторая переменная.

В языке SQL квантор существования заменяется оператором **EXISTS (существует)**.

Пусть символ **«X»** обозначает некоторую произвольную переменную. Тогда в формальной логике оператор с примененным квантором существования **EXISTS X** (предикат, зависящий от X) принимает значение «истина» тогда и только тогда, когда «предикат, зависящий от X» имеет значение «истина» при каком-либо значении переменной X. Если переменная X может быть любым целым числом, принадлежащим интервалу от 1 до 10, то предикат: **EXISTS X (X < 5)** принимает значение «истина», тогда как предикат **EXISTS X(X < 0)** принимает значение «ложь».

В языке **SQL** предикат с квантором существования представлен может быть представлен выражением вида:

EXISTS (SELECT * FROM …)

Выражение считается истинным только тогда, когда результат вычисления подзапроса, представленного с помощью **SELECT * FROM ...,** является непустым множеством. Выражение истинно тогда и только тогда, когда существует какая-либо запись в таблице, указанной во фразе **FROM** подзапроса, которая удовлетворяет условию **WHERE** этого подзапроса.

Пример **SQL-** запроса с квантором существования для выбора фамилий покупателей, которым продан компьютер «Macintosh»:

SELECT NAME FROM tab10
WHERE EXISTS (SELECT * FROM tab11
WHERE tab10.kod= tab11.kod AND.STOCK = “Macintosh”)

Фактически любой запрос, который может быть выражен с использованием оператора **IN**, альтернативным образом может быть сформулирован с помощью **EXISTS**.

Можно сконструировать отрицание существования, используя **NOT EXISTS.** С помощью SQL-запроса

SELECT NAME FROM tab10
WHERE NOT EXISTS (SELECT * FROM tab11
WHERE tab10.kod= tab11.kod AND.STOCK = “Macintosh”) можно получить информацию о покупателях, которые не купили «Macintosh»**.**

Заключенный в скобки подзапрос, входящий в конструкцию **EXISTS** не обязательно использует конструкцию **SELECT * ...** . В конструкции **SELECT** можно также указать имя поля, т.е. использовать предложение вида: **SELECT имя_поля FROM ...** . Операторы **EXISTS** и **NOT EXISTS** всегда помещаются перед подзапросом.

13 Объединение множеств

Объединением двух множеств называется множество всех элементов, принадлежащих какому-либо одному или обоим множествам. Поскольку отношение является множеством записей, то можно построить объединение двух отношений. Результатом будет отношение, состоящее из всех строк,

входящих в какое-либо одно или в оба сразу отношения. Однако строки этих двух отношений должны быть совместимы по объединению.

В языке **SQL** две таблицы совместимы по объединению и к ним может быть применен оператор объединения **UNION** тогда и только тогда, когда:

1) они имеют одинаковое число полей, например m;

2) для всех i (i ● 1,m) i-е поле первой таблицы и i-е поле второй таблицы имеют в точности одинаковый тип данных.

В **SQL**-92 реализованы некоторые операции реляционной алгебры в явном виде с помощью операторов: **UNION, INTERSECT, EXCEPT, JOIN**.

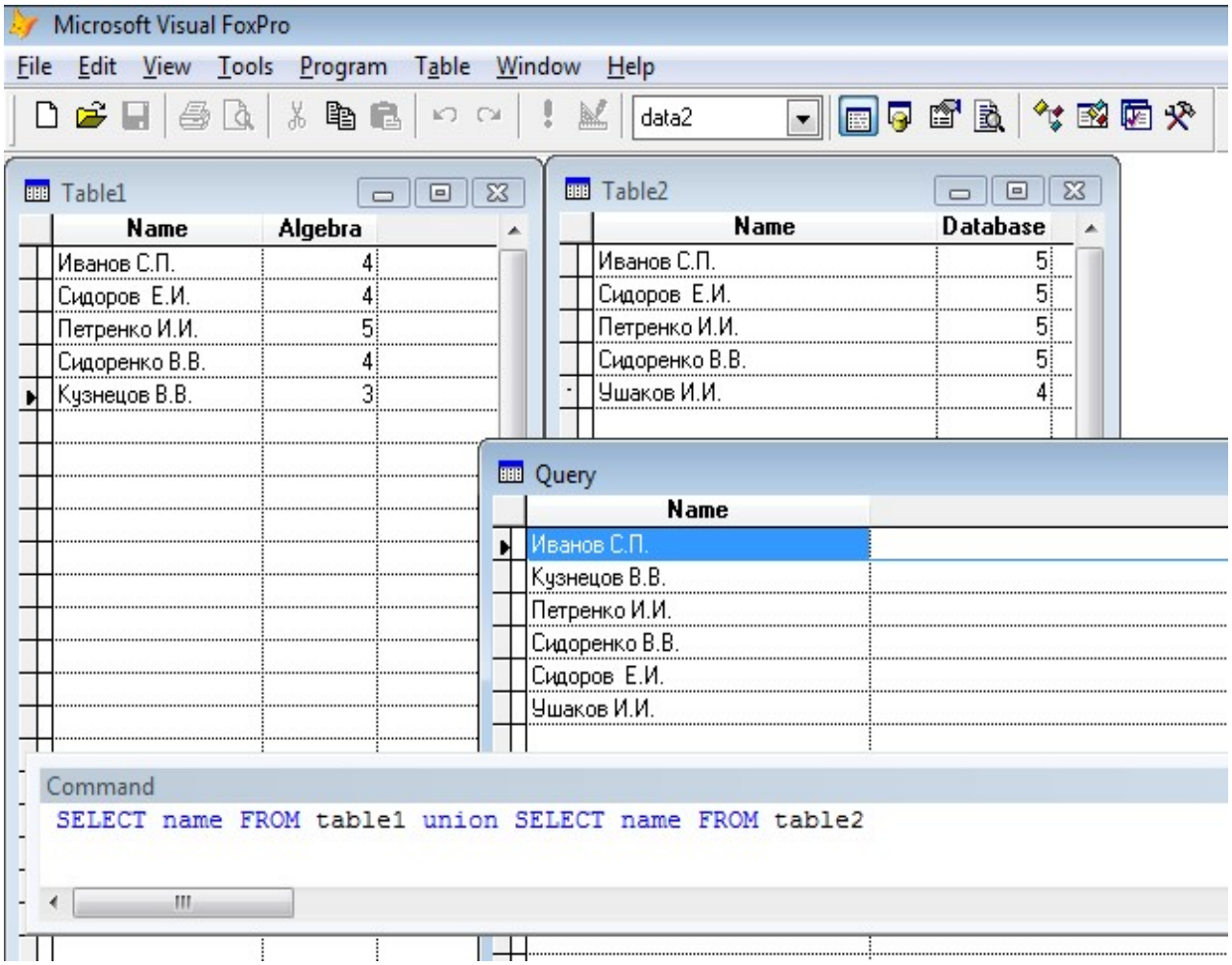
По отношению к таблицам операторы объединения **(UNION)**, пересечения **(INTERSECT)** и разности **(EXCEPT)** можно применять только в случае объединительной совместимости.

На рисунке 13.1 показано применение оператора UNION для объединения двух таблиц, полученное с помощью **СУБД MS Visual FoxPro.** В командном окне показан текст запроса:

SELECT name FROM table1 UNION Select name FROM table2

Результат выполнения запроса находится в окне Query. Состав таблиц показан в верхней части окна. В запросе нет ключевого слова ALL, поэтому в выборке нет повторений.

Для применения оператора **UNION** требуется совместимость таблиц по объединению. СУБД проверяет эту совместимость формально. Пользователи должны проверять и по смыслу. НЕЛЬЗЯ ОБЪЕДИНЯТЬ ДАННЫЕ ОДНОГО ТИПА, РАЗНЫЕ ПО СМЫСЛУ!



унок 13.1 – Объединение таблиц, полученное средствами **СУБД MS Visual FoxPro**

Следующий пример показывает результат применения оператора UNION в случае, когда формальная совместимость по объединению имеется, но фактически столбцы из разных таблиц отображают разные данные. Их совмещение в едином столбце не имеет смысла. Поэтому слева от рисунка имеется надпись «неверно».

НЕВЕРНО

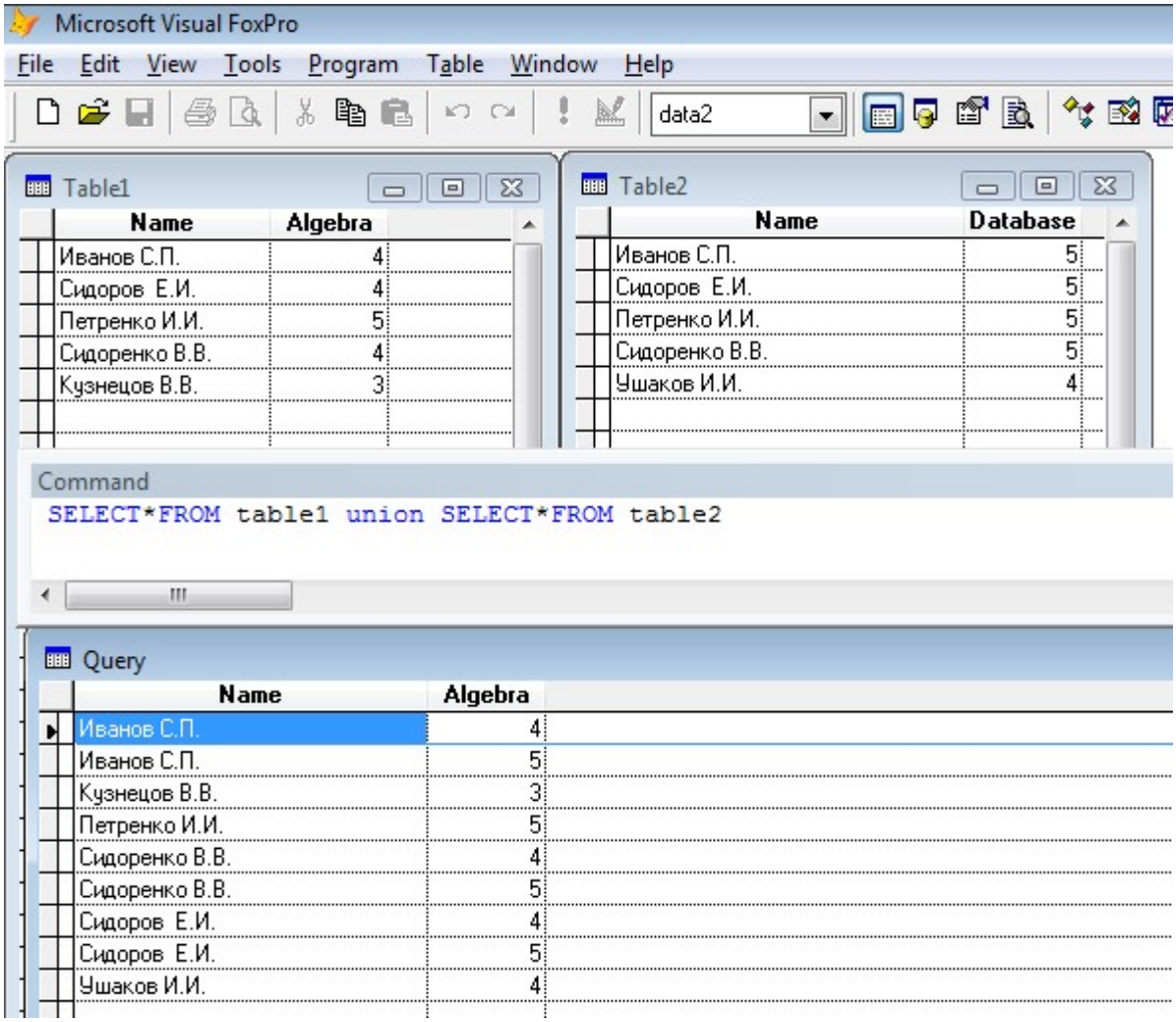


Рисунок 13.2 – Неверное применение оператора **UNION**

Пример 13.1. **SQL**-запрос

SELECT kod FROM tab12 WHERE UNITPRICE>1000
UNION
SELECT kod FROM tab13 WHERE CUSTOMERNO=23

предназначен для выбора кодов товаров, которые имеют стоимость более 1000, либо приобретаются покупателем с кодом 23 (либо и то, и другое).

Использование оператора **UNION** исключает из результатов выборки повторяющиеся значения. Если это не устраивает пользователя, то вместо оператора **UNION** используют **UNION ALL**.Тогда повторы не будут исключены из выборки.

Оператором **UNION** можно соединить любое количество конструкций **SELECT**.

Пример 13.2. С помощью **SQL**-запроса

SELECT kod FROM tab12 WHERE UNITPRICE>1000
UNION
SELECT kod FROM tab13 WHERE CUSTOMERNO=23
UNION

SELECT kod FROM tab12 WHERE UNITPRICE< 500 пользователь может получить информацию о кодах товаров, которые имеют стоимость более 1000, либо приобретены покупателем с кодом 23, либо имеют цену менее 500.

Если к запросу, приведенному в примере 13.1, добавить строку **OR UNITPRICE < 500**, то его результирующая таблица будет аналогична результирующей таблице, соответствующей запросу из примера 13.2.

Оператор **ORDER BY** в запрос с использованием оператора **UNION** может входить только в последнее предложение **SELECT**. При использовании критерия упорядочивания используются номера полей результирующей таблицы. Пример с включением константы в результирующую таблицу:

```
SELECT kod,"Стоимость товара >1000$" FROM tab12
WHERE UNITPRICE>1000
UNION
SELECT kod, "Товар куплен покупателем 23" FROM tab13
WHERE CUSTOMERNO=23 ORDER BY 2,1
```

Другой синтаксис оператора объединения:

```
(SELECT*FROM STOCK) UNION (SELECT*FROM ORDSALE)
```

Еще один вариант запроса:

```
SELECT*FROM (TABLE tab12 UNION TABLE tab13)
```

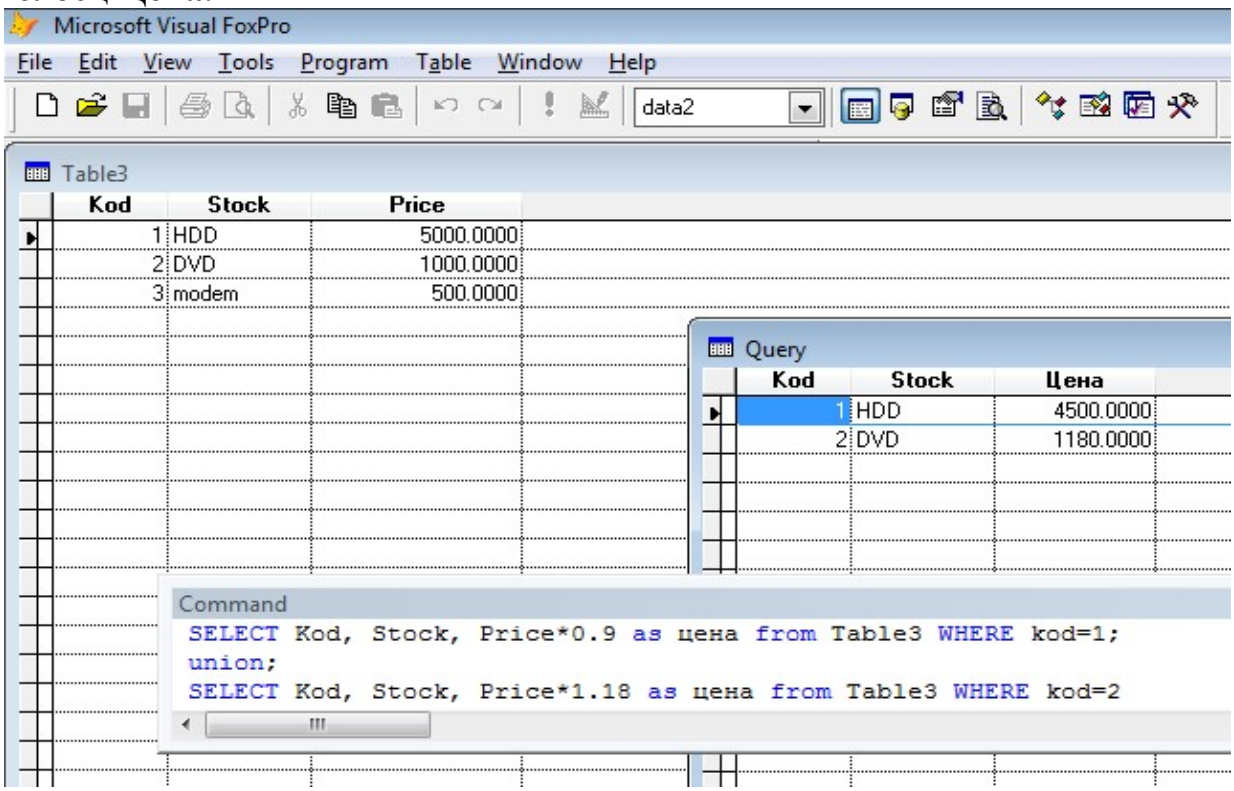
Если эти операции необходимо применить к отдельным столбцам, то используют команду **CORRESPONDING BY (имя_поля)**. В этом случае объединительная совместимость таблиц не требуется, но обязательно указанные поля должны иметь одинаковый тип данных.

Использование оператора UNION при выборке вычисляемых значений

При выполнении запросов может осуществляться не только выбор из таблицы ранее введенных в нее значений, но и получение данных, отсутствующих в исходной таблице. При этом условия выборки для различных полей и строк исходной таблицы могут быть различными, что приводит к необходимости составления различных запросов. В том случае, когда полученный результат требуется представить в виде одной таблицы, требуется выполнить объединение множеств. В этом случае используется оператор **UNION**, а результаты выполнения различных запросов должны быть совместимы по объединению.

На рисунке 13.3 в СУБД **Visual FoxPro** показано применение оператора **UNION** для объединения двух запросов, полученное с помощью **СУБД MS Visual FoxPro**. Текст запроса приведен в командном окне. Результат выполнения запроса находится в окне **Query**. Состав таблицы показан в верхней части окна. Первый запрос вычисляет цену товара с кодом kod=1 со скидкой 10%. Второй запрос вычислят цену товара с кодом kod=2 с наценкой

18% (Это налог на добавленную стоимость). Столбец **Price** переименован в столбец **Цена**.



исунок 13.3 – Применение оператора **UNION** при выборке вычисляемых значений

Тип переменной **PRICE – Currency.** По умолчанию данный тип переменной настроен на получения десятичного числа с 4 знаками после запятой (рисунок 13.3, окно **Query)**. Однако данный тип переменной позволяет настроить формат вывода десятичного числа таким образом, чтобы после десятичной точки было два знака, что соответствует национальной валюте России.

SQL-запрос с оператором UNION может объединять произвольное количество запросов. Добавим к запросу, представленному на рисунке 13.3, еще одно условие, вычислим цену товара с кодом Kod=3 со скидкой 5%. Тогда для получения результатов выборки в виде одной таблицы потребуется SQL-запрос:

SELECT Kod, Stock, Price * 0.9 AS цена FROM Table3
WHERE Kod=1
UNION
SELECT Kod, Stock, Price * 1.18 AS цена FROM Table3
WHERE Kod=2
UNION
SELECT Kod, Stock, Price * 0,95 AS цена FROM Table3
WHERE Kod=3

Получение вычисляемых значений и правила вычислений могут определяться условиями, которые проверяются не только в таблице, содержащей числовые данные для вычислений, но и в других таблицах.

Пример 13.3. Пусть в базе данных содержится таблица Т**овары(kod**, **price)** и таблица **Клиенты(kod_c, name, , kod).** Чтобы вычислить цены товаров из таблицы **Товары**, одновременно предоставив клиентам из Москвы скидку 10%, а клиентам из Ставрополя скидку 5% можно использовать запрос с подзапросом:

```
SELECT Товары. kod, Товары. price*0.9 as цена,  Клиенты.city
FROM Товары, Клиенты
WHERE EXISTS(SELECT*from Клиенты WHERE city="Москва" AND
Товары.kod=Клиенты.kod)
UNION
 SELECT Товары. kod, Товары. price *0.95 as цена, Клиенты.city
FROM Товары, Клиенты
WHERE EXISTS(SELECT*from клиенты WHERE city="Ставрополь"
AND Товары.kod=Клиенты.kod) AND Товары.kod=Клиенты.kod
```

14 Модификация данных в таблицах

С помощью конструкций языка **SQL** можно не только производить поиск информации в базе данных, но и выполнять основные действия над таблицами:

1) добавлять информацию в таблицу;
2) модифицировать данные в таблице;
3) удалять информацию из таблицы.

В языке SQL для модификации данных в таблицах используется конструкция **UPDATE**, которая имеет следующий синтаксис:

```
UPDATE таблица
SET поле=выражение [,поле=выражение] …
[WHERE условие]
```

В результате выполнения этой конструкции все записи в таблице, которые удовлетворяют условию, обновляются в соответствии с оператором присвоения **«поле = выражение»**.

При использовании конструкции **UPDATE** может использоваться только одна таблица. При использовании подзапросов для модификации данных результат выборки должен возвращать только одно значение, а не несколько.

Модификация единственной записи. Для каждой записи, которая должна быть обновлена, т. е. для каждой записи, которая удовлетворяет условию **WHERE**, или для всех записей, если фраза **WHERE** опущена, ссылки во фразе **SET** на поля этой записи обозначают значения этих полей до их модификации. Например, с помощью **SQL**-запроса

```
UPDATE tab12
SET NAME =“HDD4”,
UNITPRICE = UNITPRICE+10000.00
WHERE NAME =“HDD”
```

будет изменено название товара «HDD» на «HDD4» и увеличена стоимость на 10000.00.

Модификация множества записей. При изменении значений во множестве записей таблицы условие, определяемой при помощи оператора **WHERE,** должно удовлетворять всему множеству. Используя **SQL**-запрос

UPDATE tab14
SET CREDITLIMIT= CREDITLIMIT*2
WHERE CITY=“Ставрополь”

можно для всех покупателей, проживающих в Ставрополе увеличить в 2 раза сумму кредита.

Модификация с подзапросом. Модификация с подзапросом используется в том случае, если необходимо произвести изменения полей в связанных таблицах. В частности, с помощью **SQL**-запроса

UPDATE tab14
SET UNITPRICE = 0.8*UNITPRICE
WHERE “Мичуринск” = (SELECT CITY FROM tab15
WHERE tab15.CUSTOMERNO= tab14.CUSTOMERNO) можно вывести на экран список всех покупателей, проживающих в Мичуринске, уменьшив стоимость, приобретенного ими товара на 20%.

15 Удаление данных

Для удаления данных из базы в языке SQL используется конструкция **DELETE**, которая имеет следующий синтаксис:

DELETE FROM таблица [WHERE условие]

В результате выполнения конструкции удаляются все записи, которые удовлетворяют условию.

С помощью конструкции **DELETE** из базы данных может быть удалена одна запись, множество записей, все записи из одной таблицы; одна или множество записей из нескольких связанных таблиц.

Пример 15.1 – **SQL**-запрос, иллюстрирующий удаление одной записи из базы данных

DELETE FROM tab12
WHERE CUSTOMERNO=23

В результате его выполнения будет удален из базы данных покупатель со значением кода, равным «23».

Пример 15.2 – SQL-запрос, иллюстрирующий удаление из базы множества записей

DELETE FROM tab12 WHERE STOCK=34

После выполнения данного **SQL**-запроса из таблицы **tab12** будут удалены все записи, в которых поле **STOCK** равно 34.

Пример 15.3 – **SQL**-запрос, иллюстрирующий удаление всех записей из таблицы **ORDSALE:**

DELETE FROM ORDSALE

В результате выполнения этой операции таблица будет доступна для дальнейшей работы, однако, она будет пустой. Удаление из таблицы всех записей не приводит к уничтожению таблицы.

SQL-запрос с подзапросом используют в том случае, если необходимо произвести удаление записей из нескольких связанных таблиц, принадлежащих одной базе данных.

Пример 15.4 – Удаление с подзапросом **DELETE FROM tab12**
WHERE “Киев”= (SELECT CITY FROM tab13
WHERE tab13.CUSTOMERNO= tab12.CUSTOMERNO)

После выполнения данного запроса из базы данных будет удалена информация о покупках всех клиентов, проживающих в Киеве.

16 Добавление записей

Для добавления записей в базу данных с помощью языка **SQL** используется конструкция **INSERT**, которая имеет два варианта синтаксиса:
INSERT INTO таблица [(поле [,поле] ...)] VALUES (константа [,константа]...) или
INSERT INTO таблица [(поле [,поле] ...)] подзапрос

В первом варианте в таблицу вставляется запись, имеющая заданные значения для указанных полей, причем i-я константа в списке констант соответствует i-у полю в списке полей. Во втором варианте формируется подзапрос, представляющий собой множество записей, которые добавляются в таблицу. При этом i-е поле результата подзапроса соответствует i-у полю в списке полей добавляемой таблицы. В обоих случаях отсутствие списка полей эквивалентно использованию всех полей таблицы.

С помощью конструкции **INSERT** в базу данных можно вставить одну запись, множество записей.

Пример 16.1 – SQL-запрос для вставки одной записи в базу данных
INSERT INTO tab15
(kod, NAME, UNITPRICE, CATEGORY)
VALUES
(1001,”Lenovo”,3000000,2)

В результате выполнения приведенного выше запроса будет создана новая запись для товара с заданным номером, наименованием, стоимостью и категорией товара. В частности, в таблицу **tab15** будет добавлен товар «Lenovo», с кодом равным 1001, стоимостью 30000000 и категорией, равной 2.

Пример 16.2 – SQL-запрос для вставки одной записи в базу данных
INSERT INTO tab15 VALUES (1001,”Pentium”,3000000,2)
С помощью данного **SQL**-запроса в таблицу **tab15** также добавляется новая строка, содержащая информацию о товаре «Pentium».

Отсутствие полей эквивалентно перечислению списка всех полей таблицы в порядке слева направо так, как они были определены при создании таблицы.

17 Запросы по дате

При работе с базой данных у пользователя часто возникает проблема поиска информации за определенный промежуток времени. Для выполнения в СУБД **Visual FoxPro** запросов по дате можно воспользоваться следующими способами:

1) с помощью форм;
2) с помощью конструктора запросов; 3) с помощью «мастера запросов».

Чтобы создать запрос с помощью конструктора запросов необходимо активизировать вкладку **Filter (Фильтр)** и выбрать из списка **Field Name (Имя поля)** поле, содержащее дату. Затем в списке **Criteria (Критерий)** выбрать значение **Between (Между)**.

После перехода на поле **«Example (Образец)»** можно вводить начальную и конечную даты интервала. Для ввода дат используется следующий формат: **CTOD(12.12.01), CTOD(12.12.04)**.

Для поиска по дате можно использовать экранную форму, приведенную на рисунке 17.1.

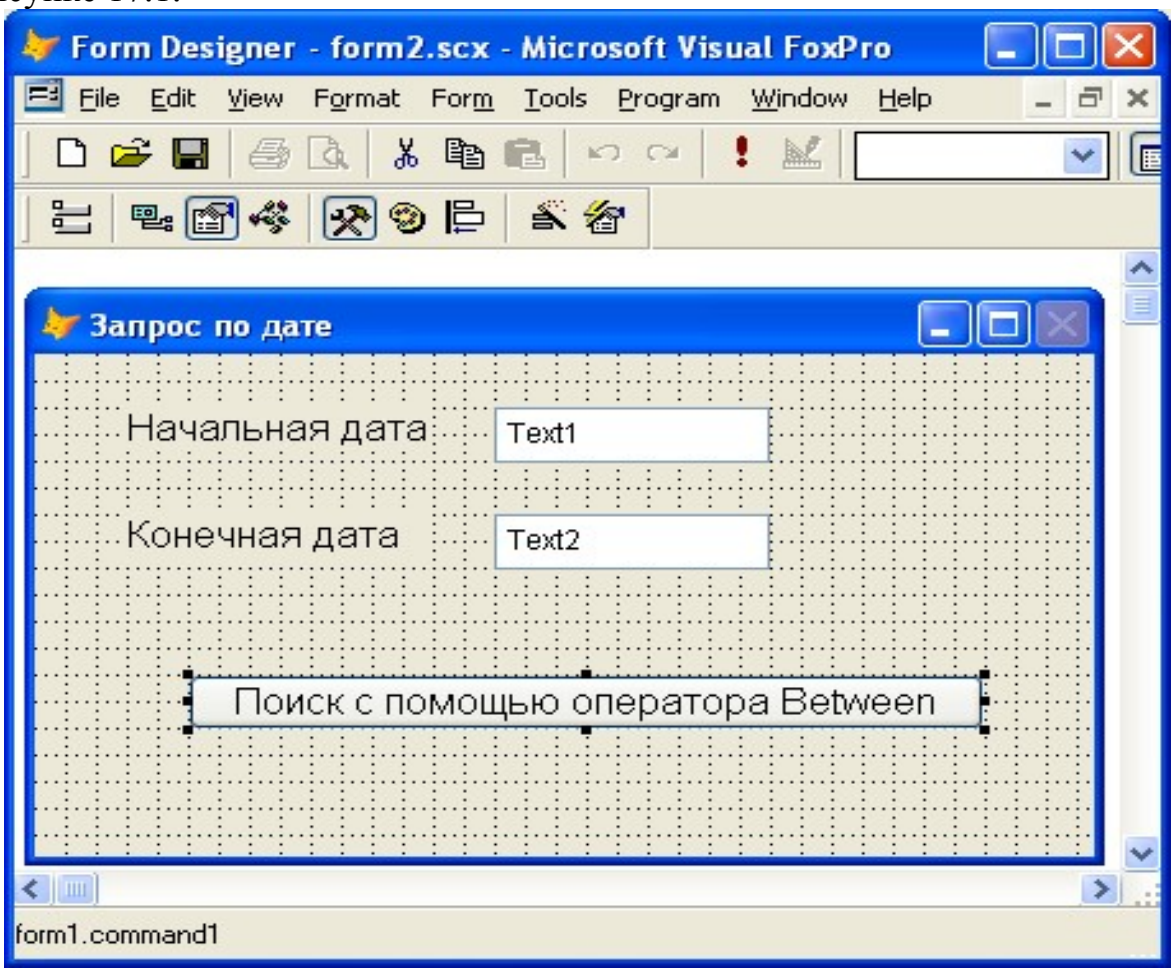
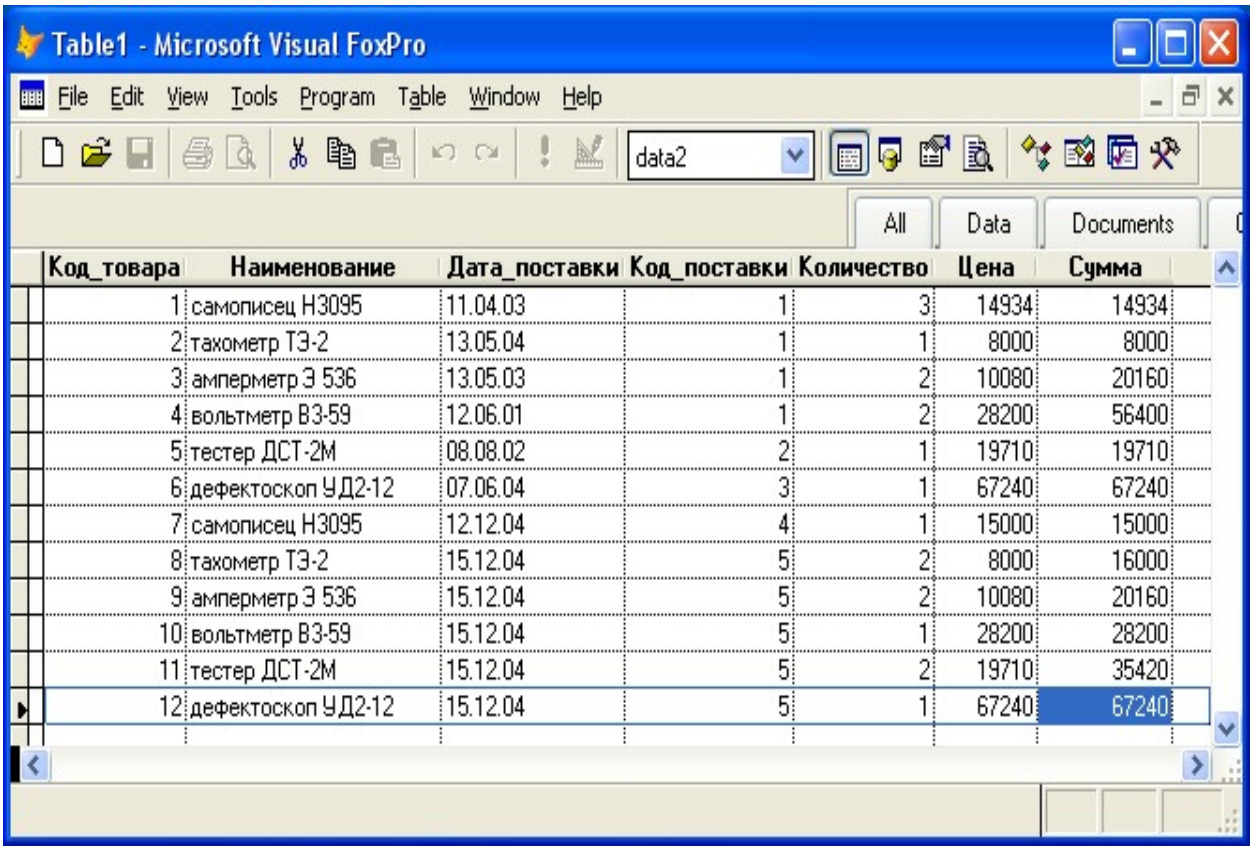


Рисунок 17.1 – Экранная форма для поиска информации по дате

В качестве среды окружения для спроектированной формы нужно выбрать таблицу, содержащую данные типа «Date». Например, таблицу Table1, представленную на рисунке 17.2.



Код_товара	Наименование	Дата_поставки	Код_поставки	Количество	Цена	Сумма
1	самописец Н3095	11.04.03	1	3	14934	14934
2	тахометр ТЭ-2	13.05.04	1	1	8000	8000
3	амперметр Э 536	13.05.03	1	2	10080	20160
4	вольтметр В3-59	12.06.01	1	2	28200	56400
5	тестер ДСТ-2М	08.08.02	2	1	19710	19710
6	дефектоскоп УД2-12	07.06.04	3	1	67240	67240
7	самописец Н3095	12.12.04	4	1	15000	15000
8	тахометр ТЭ-2	15.12.04	5	2	8000	16000
9	амперметр Э 536	15.12.04	5	2	10080	20160
10	вольтметр В3-59	15.12.04	5	1	28200	28200
11	тестер ДСТ-2М	15.12.04	5	2	19710	35420
12	дефектоскоп УД2-12	15.12.04	5	1	67240	67240

Рисунок 17.2 – Экранная форма **Table1**

Для организации поиска информации по дате в командном окне кнопки «Поиск с помощью оператора **Between»** для процедуры **Click** введите программный код, показанный на рисунке 17.3. В нем функция **Alltrim()** удаляет все пробелы из строки, функция **CTOD()** преобразует тип переменной в переменную типа **DATE.**

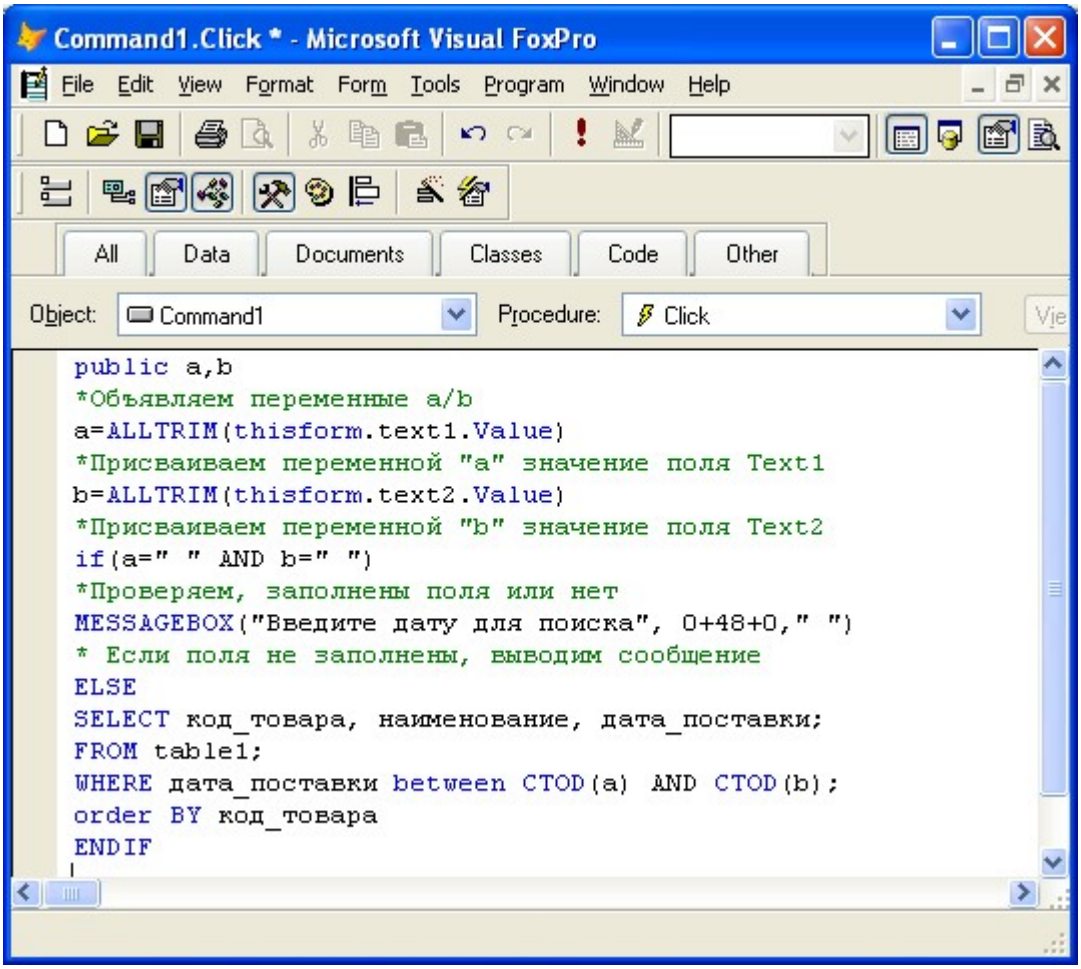


Рисунок 17.3 – Программный код для кнопки «Поиск с помощью оператора **Between»**

18. Соединение отношений. Оператор JOIN

Соединение отношений R_1 и R_2:

$$R = R_1 \underset{i\theta j}{\bowtie} R_2 = \sigma_{i\theta kj}(R_1 * R_2),$$

обозначается символом «$\underset{i\theta j}{\bowtie}$», под которым на месте нижнего индекса записывают условие соединения, в котором θ — арифметический оператор сравнения (<,=,>,≠,≤,≥); k—арность отношения R_1; i и j—номера столбцов соответственно в отношениях R_1 и R_2.

Если θ является арифметическим оператором равенства, то операцию называют эквисоединением.

Пример 18.1. Рассмотрим пример соединения отношений с поименованными столбцами R_{11}(A,B,C) и R_{12}(D,E). Вычислим эквисоединение при условии, что элементы столбцов B и D равны.

$$R_{13} = R_{11} \underset{B=D}{\bowtie} R_{12} = \sigma_{B=D}(R_{11} * R_{12}),$$

Отношения R_{11}(A,B,C) и R_{12}(D,E) представлены в таблицах 18.1 и 18. 2.

Таблица 18.1 - R_{11}(A,B,C)

Таблица 18.2. - R_{12}(D,E)

A	B	C
а	б	с
а	и	р
д	е	ж

D	E
а	и
е	к

Промежуточное действие – вычисление декартова произведения: $R_{11}*R_{12}$. Результат представим в таблице 18.3.

Таблица 18.3 - $R_{11}*R_{12}$

A	B	C	D	E
а	б	с	а	и
а	и	р	а	и
д	е	ж	а	и
а	б	с	е	к
а	и	р	е	к
д	е	ж	е	к

Следующее действие – селекция строк при условии, что элементы столбцов B и D равны. Результат содержит одну строку, представленную в таблице 18.4.

Таблица 18.4 – R_{13}

A	B	C	Д	E
д	е	ж	е	к

Естественное соединение отношений R_1 и R_2. Эта операция применяется только тогда, когда в отношениях R_1 и R_2 имеются одинаковые столбцы. Естественное соединение - это комбинация двух отношений по общим атрибутам. Условие реализации $R_1 \cap R_2 \neq Ø$.

Пусть отношения R_1 и R_2 имеют соответственно схемы

$R_1(A_1,A_2,...,A_k,B_1,B_2,...,B_n)$, $R_2(A_1,A_2,...,A_k,C_1,C_2,...,C_m)$, где имена $A_1,A_2,...,A_k$ у обоих отношений совпадают, а остальные различаются (для упрощения совпадающие имена размещены в начале, но они, конечно, могут быть записаны в любом другом порядке). Естественное соединение равно:

$$R = \pi_{A1,...,Ak,B1,...,Bn,C1,...,Cm}(\sigma_{R1.A1=R2.A1,...,R1.Ak=R2.Ak}(R_1 * R_2))$$

$R_1.A_1$—имя столбца отношения $R_1 \times R_2$, соответствующего столбцу A_1 в отношении R_1; $R_2.A_2$ — имя столбца отношения $R_1 \times R_2$, соответствующего столбцу A_1 в отношении R_2. Запись «имя_таблицы**.**имя_атрибута», например, $R_1.A_1$ называется «уточненное имя атрибута».

Пример 18.2. Получим естественное соединение отношений ВЕДОМОСТЬ_1(Фамилия, алгебра) и ВЕДОМОСТЬ_2(Фамилия, геометрия). Отношения представлены в таблицах 18.5 и 18.6. В этих отношениях имеется

один совпадающий столбец. Условие естественного соединения – в одноименных столбцах атрибуты должны принимать совпадающие значения.

Таблица 18.5 -ВЕДОМОСТЬ_1

Фамилия	**алгебра**
Иванов	отлично
Петров	хорошо

Таблица 18.6 - ВЕДОМОСТЬ_2

Фамилия	**геометрия**
Иванов	хорошо
Петров	отлично

Получим декартово произведение отношений ВЕДОМОСТЬ_1 и ВЕДОМОСТЬ_2. Результат представим в таблице 18.7.

Таблица 18. 7 – отношение ВЕДОМОСТЬ_1 * ВЕДОМОСТЬ_2

Фамилия	**алгебра**	**Фамилия**	**геометрия**
Иванов	отлично	Иванов	хорошо
Петров	хорошо	Иванов	хорошо
Иванов	отлично	Петров	отлично
Петров	хорошо	Петров	отлично

Следующий шаг – селекция. Выбираем строки, удовлетворяющие условию: ВЕДОМОСТЬ_1.фамилия= ВЕДОМОСТЬ_2.фамилия. Получим таблицу 18. 8.

Таблица 18. 8 – Результат селекции

? ВЕДОМОСТЬ_1.фамилия= ВЕДОМОСТЬ_2.фамилия (ВЕДОМОСТЬ_1 * ВЕДОМОСТЬ_2)

Фамилия	**алгебра**	**Фамилия**	**геометрия**
Иванов	отлично	Иванов	хорошо
Петров	хорошо	Петров	отлично

Для получения окончательного результата в таблице 18.8 нужно устранить избыточность. Для этого выполним проекцию на все разные столбцы. Результат – в таблице 18. 9.

Таблица 18.9 – результат вычисления естественного соединения отношений ВЕДОМОСТЬ_1 и ВЕДОМОСТЬ_2

Фамилия	**алгебра**	**геометрия**
Иванов	отлично	хорошо
Петров	хорошо	отлично

Для реализации соединения в языке SQL используется оператор **JOIN.** В СУБД **Visual FoxPro** также используется оператор **JOIN.** Для получения естественного соединения в **Visual FoxPro** используется оператор **INNER JOIN.**

Пример 18.3. Рассмотрим базу данных, содержащую две таблицы, представленные на рисунке 18.1. Таблица Table1 представляет собой ведомость по алгебре – **Table1(Name, Algebra)** и таблица Table.2 представляет собой ведомость БД – **Table2(Name,Database).** Условие применения естественного соединения выполняется, так как у таблиц имеется общий столбец – **Name.** Естественное соединение таблиц можно получить разными способами, показанными на рисунке 18.1. В командном окне представлены два различных запроса, выполнение которых дает одинаковый результат, представленный в окне **Query**. Один из запросов содержит оператор **INNER JOIN:**

SELECT Table1.name, Table1.Algebra, Table2.Database FROM Table1, Table2 WHERE Table1.name= Table2.name

Другой запрос:

SELECT Table1.name, Table1.Algebra, Table2.Database
FROM Table1 INNER JOIN Table2 ON Table1.name= Table2.name

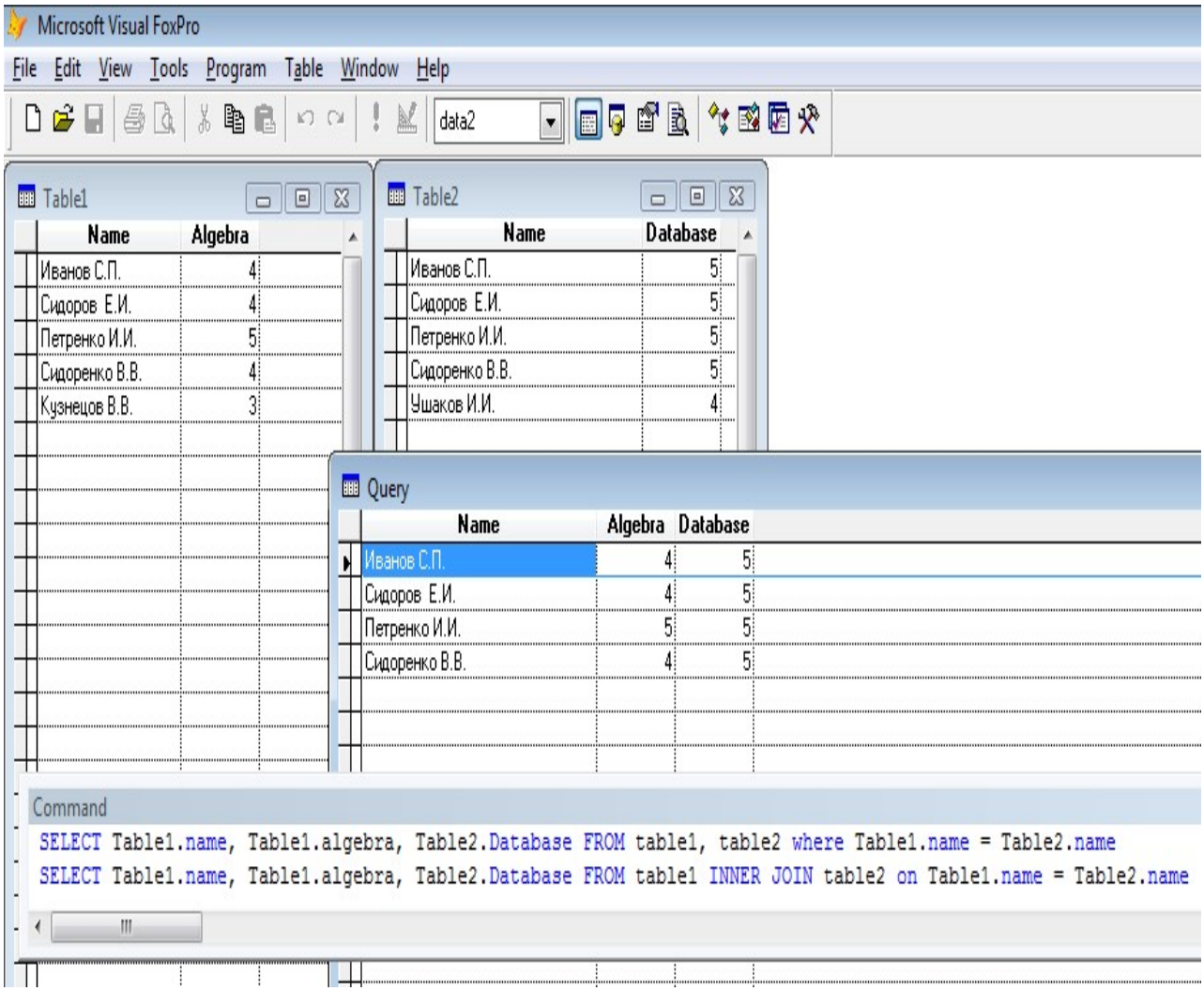


Рисунок 18.1 – Получение естественного соединения

Естественное соединение, полученное в результате выполнения любого из приведенных запросов, показано в окне Query на рисунке 18.1.

Пример 18.4. Левое соединение включает все строки левой таблицы (по отношению к оператору **LEFT JOIN**) и только те строки из правой таблицы, которые удовлетворяют условию соединения (Table1.name= Table2.name).

Запрос для получения левого соединения:

Select Table1.name, Table1.Algebra, Table2.Database
FROM Table1 LEFT JOIN Table2 ON Table1.name= Table2.name

Результат выполнения этого запроса представлен на рисунке 18.2. В рассмотренном примере в правой таблице Table2 нет строчки, где поле Table2.name = «Кузнецов В.В.», поэтому в результате выполнения запроса появляется неопределенное значение NULL.

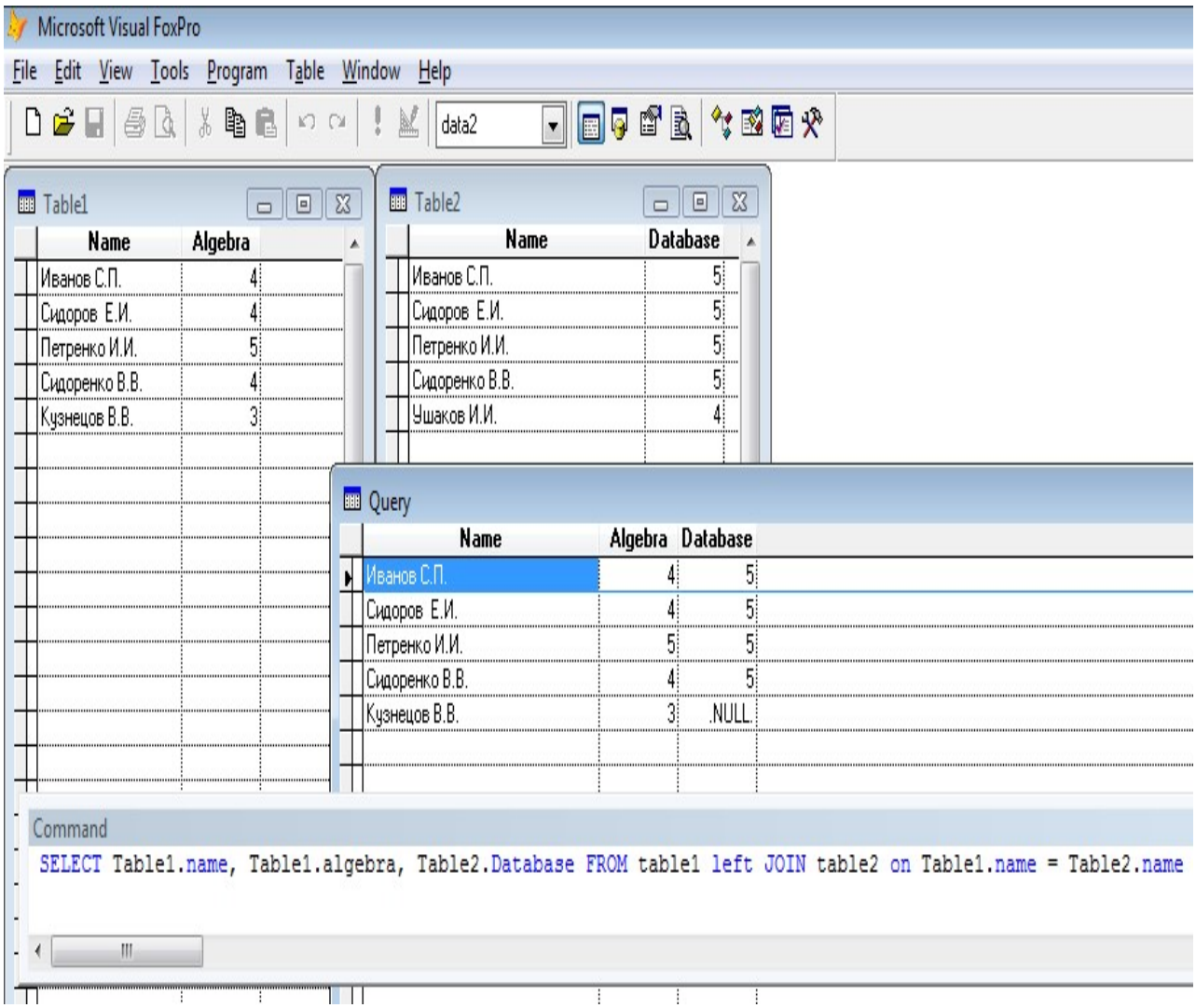


Рисунок 18.2 – Получение левого соединения

Пример18.5. Правое соединение включает все строки правой таблицы (по отношению к оператору RIGHT JOIN) и только те строки из левой таблицы, которые удовлетворяют условию соединения (Table1.name=Table2.name). Запрос для получения правого соединения

Select Table1.name, Table1.Algebra, Table2.Database FROM Table1 RIGHT JOIN Table2 ON Table1.name= Table2.name

Результат выполнения этого запроса представлен на рисунке 18.3.

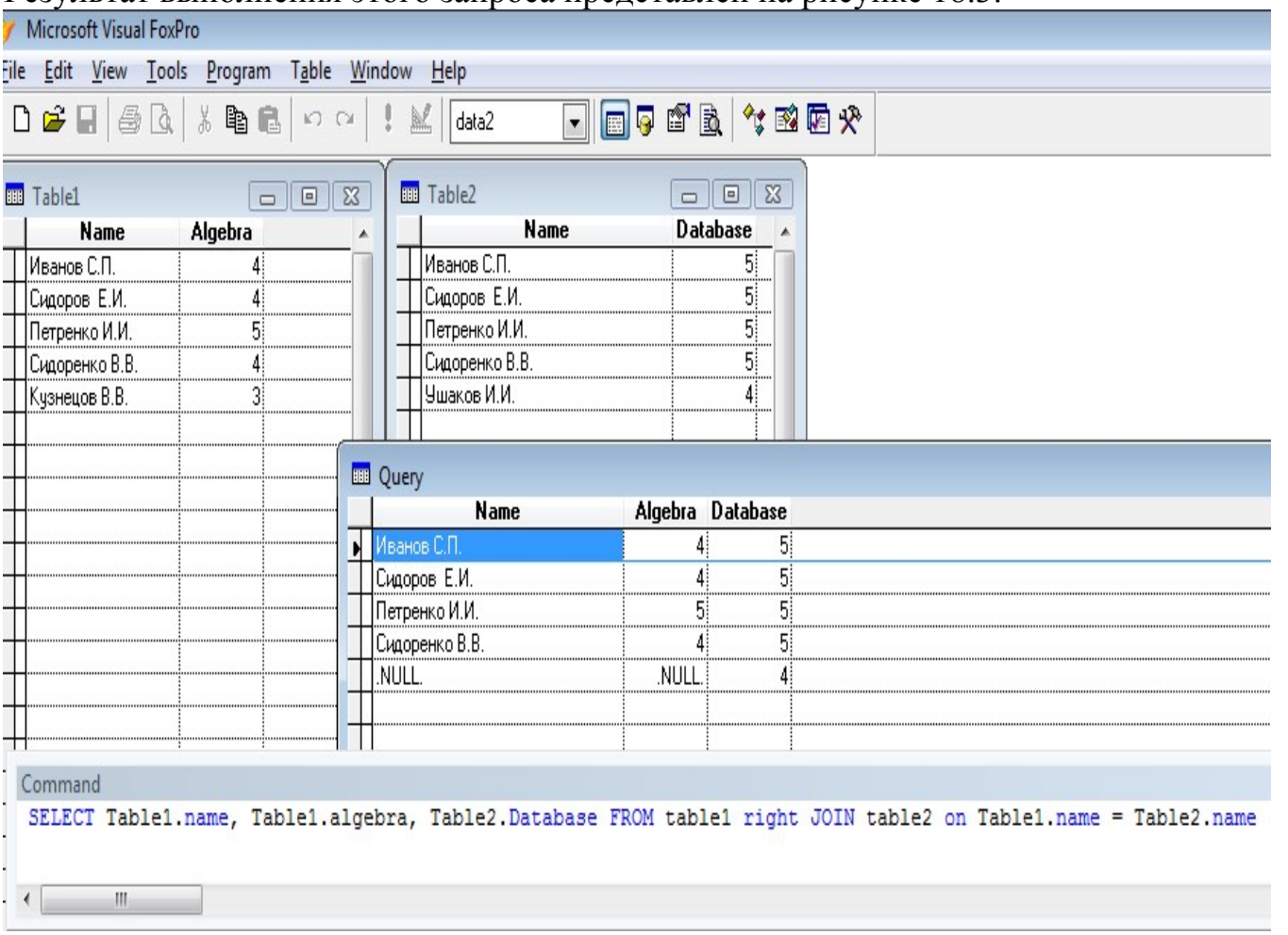


Рисунок 18.2 – Получение правого соединения

19. Коррелированные и некоррелированные подзапросы

В некоторых случаях необходимо выбрать данные из таблицы, основываясь на результатах дополнительных выборок из этой же таблицы. Такие выборки называются коррелированными. Некоррелированным называется подзапрос, который не зависит ни от какого внешнего запроса. Некоррелированный подзапрос не зависит от строки, которую рассматривает внешний запрос. Внешним или главным запрос называется запрос, в котором содержатся все подзапросы. Синтаксис подзапроса такой же, как и у внешнего запроса. Подзапрос записывают в скобках.

Рассмотрим рисунок 19.1, на котором представлен пример получения коррелированного подзапроса. В левой части рисунка показана таблица Table3. В командном окне записан запрос с подзапросом, который использован для выборки товаров, у которых цена меньше, чем средняя. В правой части рисунка показан результат выполнения запроса (Query).

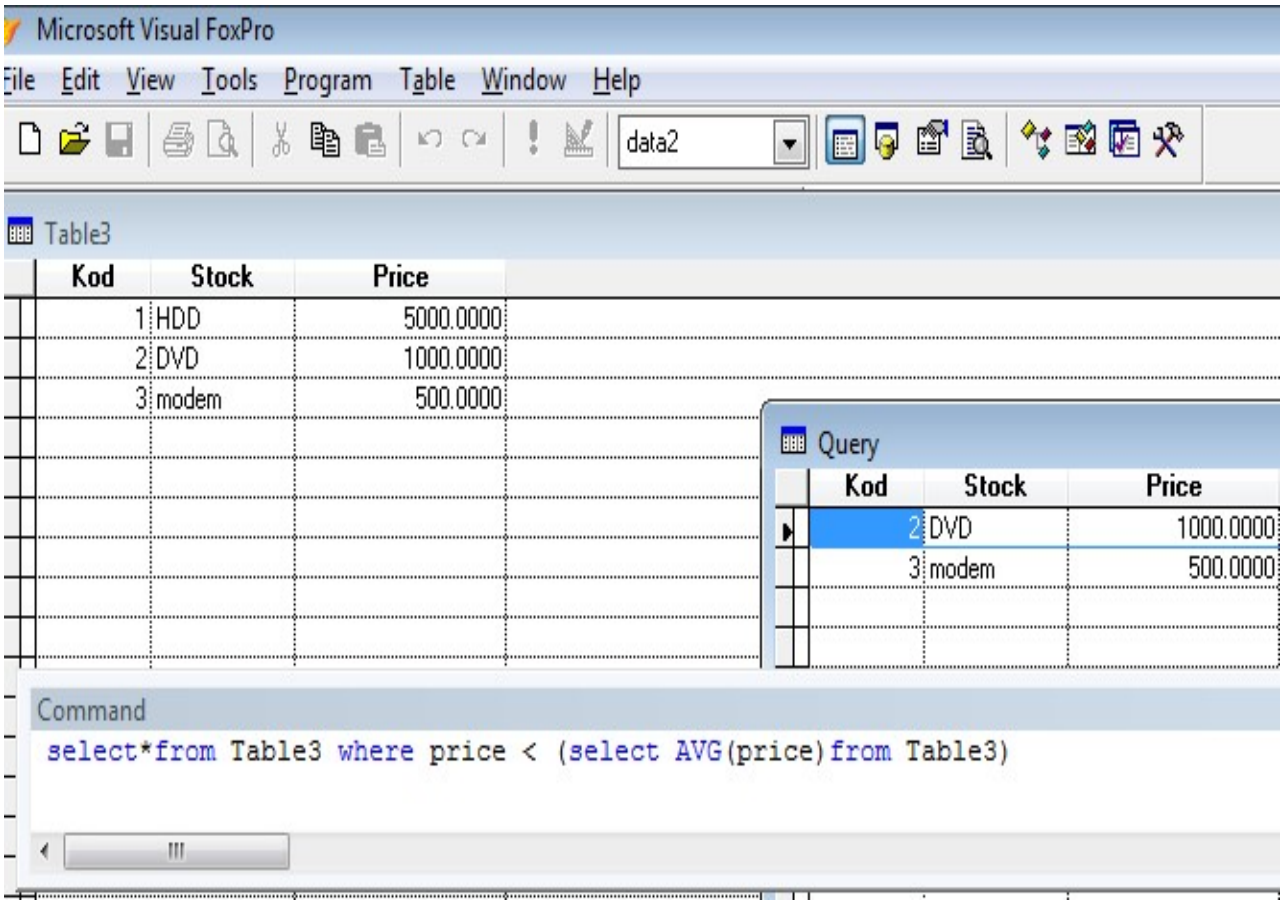


Рисунок 20.1 – Получение коррелированного подзапроса

20. Вычисления в таблицах. Команда CALCULATE

В СУБД Visual FoxPro команда

CALCULATE eExpressionList[Scope] [FOR Expression1] [WHILE Expression2 [TO VarList | TO ARRAY ArrayName] [NOOPTIMIZE] [IN nWorkArea | cTableAlias] выполняет финансовые и статистические операции с полями таблиц или с выражениями, включающими поля, используя в качестве элементов *eExpressionList* приведенные в таблице 20.1 функции.

Таблица 21.1- Функции, употребляемые в eExpressionList команды **CALCULATE**

ФУНКЦИЯ	ОПИСАНИЕ
AVG(nExpression)	Вычисляет среднее арифметическое всех значений nExpression
CNT() или COUNT()	Возвращает число записей, удовлетворяющих условиям, заданным Scope, FOR и WHILE
MAX(eExpression)	Возвращает максимальное изо всех значений eExpression. Тип eExpression может быть Character, Date, DateTime, Currency и любой числовой

MIN(eExpression)	Возвращает минимальное изо всех значений *eExpression.* Тип eExpression может быть *Character, Date, DateTime, Currency* и любой числовой
NPV(nExpression1, nExpression2 [, nExpression3])	Вычисляет величину чистой приведенной стоимости инвестиции, используя ставку дисконтирования *nExpressionl*, а также стоимости *(nExpressionl)* будущих выплат (отрицательные значения) и поступлений (положительные значения). Ставка дисконтирования *nExpressionl* задается не в процентах, а в виде числа от 0 до 1; *nExpression2* задает поле или выражение с именем поля или числовое выражение, значения которого интерпретируются как будущие выплаты или поступления. *nExpressionl* - необязательное начальное вложение капитала. Не включается, если предполагается произвести начальное вложение в конце первого периода; такое начальное вложение задается в первой записи поля, причем в виде отрицательного числа
STD(nExpression)	Вычисляет стандартное (средне квадратичное) отклонение величин, возвращаемых nExpression. Средне квадратичное отклонение *STD* величин $x_1, x_2, ..., x_n$ от a вычисляется по формуле $STD=\sqrt{\frac{(x_1 - a)^2 + (x_2 - a)^2 + ... + (x_n - a)^2}{n}}$
VA*R(nExpression)*	Вычисляет дисперсию D величин, возвращаемых nExpression. Она равна квадрату стандартного отклонения: D — STD^2.

Помимо перечисленных в таблице функций, выражения *eExpressibnList* могут содержать и иные функции. Однако функции таблицы не могут быть аргументами иных функций. Так, можно записать команду calculate Avg(Sqrt(Price)), но следующая команда недопустима: calculate Sqrt(Avg(Price))

В результатах, возвращаемых функциями, учитываются только записи, удовлетворяющие условиям, заданным *Scope,* **FOR** и **WHILE.** Поля, содержащие **NULL,** при вычислениях **CALCULATE** не учитываются.

Пример 20.1. Вычисляются средняя и максимальная цены книг, а также стандартное отклонение цен по данным, имеющимся в таблице Books. Результаты сохраняются в массиве *calcResults.* В последний элемент массива заносится количество вовлеченных в вычисления записей.

```
SET TALK OFF
SELECT BOOKS
```

&& Записываем результаты в массив calcResults calculate Avg(Price), Max(Price), Std(Price), Cnt() to array calcResults
&& Печать результата
DISPLAY MEMORY LIKE CALCRESULTS
Возможный результат:
CALCRESULTS
(1) N 195.71 (195.71428571)
(2) N 210.00 (210.00000000)
(3) N 10.33 (10.32630878) (4) N 14 (14.00000000)

Министерство науки и высшего образования Российской Федерации
Государственное образовательное учреждение высшего образования
«Северо-Кавказский федеральный университет»
Пятигорский институт (филиал) СКФУ

МЕТОДИЧЕСКИЕ РЕКОМЕНДАЦИИ

К САМОСТОЯТЕЛЬНОЙ РАБОТЕ СТУДЕНТОВ

по дисциплине «Управление данными»

Направление 09.03.02 «Информационные системы и технологии»
Квалификация выпускника – бакалавр

Пятигорск, 2025 г.

СОДЕРЖАНИЕ

Введение

Цели и задачи дисциплины - изучение программных, математических, технических, алгоритмических и лингвистических методов и средств, направленных на сбор, хранение, обработку и выдачу информации средствами СУБД. Задачей дисциплины "Управление данными" является не только знакомство студентов с программными средствами реализации информационных систем, но получение ими навыков практической работы с локальными базами данных. Целью курса является также рассмотрение перспектив развития СУБД, связанных интеллектуальной обработкой данных, с обработкой и хранением мультимедиадокументов, применением распределенной обработки данных, а также приобретение практических навыков обследования предметной области, концептуального, логического и физического проектирования базы данных. Задачей курса является также воспитание коммуникационной готовности студентов к работе в области информационного обмена.

1. Место дисциплины в структуре ООП

Содержание дисциплины определено ФГОС ВО по направлению подготовки "Информационные системы и технологии".

Основные понятия банков данных и знаний; информация и данные; предметная область банка данных; роль и место банков данных в информационных системах; пользователи банков данных; преимущества централизованного управления данными; база данных как информационная модель предметной области; система управления базой данных (СУБД); администратор базы данных; архитектура банка данных; инфологическое проектирование базы данных; выбор модели данных; иерархическая, сетевая и реляционная модели данных, их типы структур, основные операции и ограничения; представление структур данных в памяти ЭВМ; современные тенденции построения файловых систем; обзор промышленных СУБД; тенденции развития банков данных.

В результате освоения дисциплины студент должен

знать:

- методы предпроектного обследования;
- методы рабочего проектирования;
- основные положения теории баз данных, хранилищ данных;
- методы выбора исходных данных для проектирования;
- методы обеспечения безопасности и целостности данных информационных систем и технологий;

уметь:

- проводить анализ предметной области,
- выявлять информационные потребности и разрабатывать требования к информационным системам и базам данных;
- формулировать запросы к базам данных на естественном языке, с помощью абстрактных реляционных языков и на языке SQL;
- выбирать исходные данные для проектирования;
- обеспечивать безопасность и целостность данных; владеть:
- методами проектирования баз данных;
- методами обеспечения безопасности и целостности данных;
- методами предпроектного обследования;
- методами выбора исходных данных для проектирования.

2. 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 выполнение лабораторного задания.

Подготовленный материал, практическое задание оформляются в конспекте.

Самостоятельные занятия (СЗ) являются одной из активных форм обучения.

Самостоятельные занятия по дисциплине «Управление данными» имеют целью:

- закрепить и углубить знания, полученные студентами на лекциях и в процессе лабораторных занятий;
- привить практические навыки при разработке, эксплуатации и исследовании информационных систем.

Самостоятельные занятия проводятся в специализированных аудиториях, оборудованных СВТ и возможностью пользования Интернет-ресурсами, в библиотеке, а также дома.

Предлагаемые методические рекомендации содержат информацию для студентов, необходимую при подготовке и проведении лабораторных занятий по дисциплине «Управление данными».

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

Название раздела (темы)	Цель	Форма контроля СРС	Задания для СРС	Требования к представлению и оформлению результатов СРС	Рекомендуемая литература
Подготовка к лабораторны м занятиям	изучить создание и ведение баз данных, команды SQL для создания запросов	индивиду альное собеседов ание	Сформул ировать ответы на контроль ные вопросы к лаборатор ным работам	Устно ответить на контрольные вопросы к лабораторны м работам	-Дроздова, В. И. (СевероКавказский федеральный университет). Управление данными : учеб. пособие : Направление подготовки 320400.62-Информационные системы и технологии. Бакалавриат / В. И.Дроздова ; Сев.-Кав. федер. ун-т- Ставрополь : СКФУ, 2013. - 170 с. -Крис, Фиайли . SQL Электронный ресурс / Фиайли Крис ; пер. А. В. Хаванов. - SQL,2019-04-19. - Саратов : Профобразование, 2017. - 452 с. - Книга находится в премиум-версии ЭБС IPR BOOKS. - ISBN 978-5-44880103-7 -Швецов, В.И. Базы данных Электронный ресурс : учебное пособие / В.И. Швецов. - Базы данных,2019-12-01. - Москва : Интернет-Университет Информационных Технологий (ИНТУИТ), 2016. - 218 с. - Книга находится в базовой версии ЭБС IPRbooks. -Агальцов, В. П. Базы данных : учебник / В. П. Агальцов, Кн.2, Распределенные и удаленные базы данных. - М. : Форум : ИНФРА-М,
Подготовка к лекционным занятиям	изучить основные понятия БД и СУБД	индивиду альное собеседов ание	сформули ровать и письменн о ответить на вопросы для контроля владения компетен циями данного раздела программ ы	письменно ответить на вопросы для контроля владения компетенция ми вопросы данного раздела программы	
Самостоятель ное изучение тем "Бесфайловая организация данных" и "Оптимизаци	изучить новые технологии управления данными и организаци ю БД	индивиду альное собеседов ание	сформули ровать и письменн о ответить на вопросы	письменно ответить на вопросы для контроля владения компетенция ми вопросы	
я запросов" (Сроки выполнения – 6 семестр,7,8 недели. Форма контроля - опрос)	средствами СУБД		для контроля владения компетен циями данного раздела	данного раздела	2014. - 272 с. : ил. - (Высшее образование). - Гриф: Доп. УМО. - Библиогр.: с. 260. - ISBN 978-5-8199-0394-0. - ISBN 978-5-16-003526-0 - Карпова, И. П. Базы данных : курс лекций и материалы для практических занятий : учебное пособие / И. П. Карпова. - СПб. [и др.] : Питер, 2013. - 240 с. : ил., табл. ; 24 см. - (Учебное

Выполнение курсового проекта (6 семестр)	изучить основные методы поиска информации	Защита курсового проекта	Разработка локальной реляционной базы данных (по вариантам)	Представить пояснительную записку, оформленную согласно методическим указаниям к курсовому проекту	пособие). - Гриф: Рек. - Библиогр.: с. 233-234. - ISBN 978-5-496-00546-3 - Кузнецов, С.Д. Введение в реляционные базы данных Электронный ресурс : учебное пособие / С.Д. Кузнецов. - Введение в реляционные базы данных,2021-01-23. - Москва : Интернет-Университет Информационных Технологий (ИНТУИТ), 2016. - 247 с. - Книга находится в базовой версии ЭБС IPRbooks. - ISBN 5-9556-00028-0

4.1. Примерная тематика самостоятельной работы студентов

Темы для самостоятельного изучения (или для подготовки докладов) по дисциплине «Управление данными»:

1. Сетевые технологии для обслуживания мобильного пользователя.
2. Частотные каналы Wi-Fi.
3. Облачные сервисы IBM.
4. Сервисы Apple – Amazon.
5. Сервисы Facebook, Adobe.
6. Информационно-поисковые системы.
7. Автоматизированный поиск графической информации.
8. Семантический Web и платформа XML.
9. Автоматизированное извлечение знаний из текста.
10. Гипертекстовые информационные системы.

5. План-график выполнения СРС

№ №	Название раздела	Срок сдачи результатов
5 семестр		
1	Раздел 1. Общая характеристика систем управления данными	36

2	Раздел 2. Реляционные базы данных	36
	Итого	72
6 семестр		
3	Раздел 3. Тенденции развития систем управления данными	38
4	Раздел 4. Организация данных	38
	Итого	76

6. Организация контроля знаний студентов

6.1. Формы контроля знаний студентов

Контроль и оценка знаний, умений и навыков студентов осуществляется на лабораторных занятиях, консультациях, при сдаче экзамена. В ходе контроля знаний преподаватель оценивает понимание студентом содержания дисциплины «Управление данными», его способность анализировать развитие информационных систем и технологий.

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; ~ защита лабораторной работы;

~ проверка конспектов лекций.

Устный контрольный опрос студентов проводится на лекциях (и лабораторных занятиях). По его результатам преподаватель оценивает качество подготовки студента к занятию.

На лабораторных занятиях знания и практические навыки студентов оцениваются по 4-балльной системе. Полученные оценки выставляется 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.

Экзамен, как итоговый контроль знаний студентов имеет целью проверить и оценить учебную работу студентов, уровень полученных знаний и практических навыков.

Экзамен проводится в 8 триместре после защиты всех лабораторных работ в объеме учебной программы.

7. Рекомендации по работе 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», которые прямо или косвенно относятся к изучаемой теме.

При изучении литературы и источников 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».

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 - взаимоопрос 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

7.1.1. Основная литература:

Основная литература

1.Швецов, В.И. Базы данных Электронный ресурс : учебное пособие / В.И. Швецов. - Базы данных,2019-12-01. - Москва : Интернет-Университет Информационных Технологий (ИНТУИТ), 2016. - 218 c. - Книга находится в базовой версии ЭБС IPRbooks.

2.Крис, Фиайли . SQL Электронный ресурс / Фиайли Крис ; пер. А. В. Хаванов. - SQL,2019-04-19. - Саратов : Профобразование, 2017. - 452 c. - Книга находится в премиум-версии ЭБС IPR BOOKS. - ISBN 978-5-44880103-7

3.Дроздова, В. И. (Северо-Кавказский федеральный университет). Управление данными : учеб. пособие : Направление подготовки 320400.62-Информационные системы и технологии. Бакалавриат / В. И.Дроздова ; Сев.Кав. федер. ун-т- Ставрополь : СКФУ, 2013. - 170 с.

7.1.2. Дополнительная литература:

- Агальцов, В. П. Базы данных : учебник / В. П. Агальцов, Кн.2, Распределенные и удаленные базы данных. - М. : Форум : ИНФРА-М, 2014. -
272 с. : ил. - (Высшее образование). - Гриф: Доп. УМО. - Библиогр.: с. 260. -
ISBN 978-5-8199-0394-0. - ISBN 978-5-16-003526-0

- Карпова, И. П. Базы данных : курс лекций и материалы для практических занятий : учебное пособие / И. П. Карпова. - СПб. [и др.] : Питер, 2013. - 240 с. : ил., табл. ; 24 см. - (Учебное пособие). - Гриф: Рек. - Библиогр.: с. 233-234. - ISBN 978-5-496-00546-3

- Кузнецов, С.Д. Введение в реляционные базы данных Электронный ресурс : учебное пособие / С.Д. Кузнецов. - Введение в реляционные базы данных,2021-01-23. - Москва : Интернет-Университет Информационных Технологий (ИНТУИТ), 2016. - 247 c. - Книга находится в базовой версии ЭБС IPRbooks. - ISBN 5-9556-00028-0

7.1.3. Интернет-ресурсы:
http://www.citforum.ru/database/
http://www.acm.org/sigmod/publications.html
http://www.acm.org/sigmod/databaseSoftware/index.html

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.

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное образовательное учреждение
высшего образования
«СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Пятигорский институт (филиал) СКФУ

МЕТОДИЧЕСКИЕ УКАЗАНИЯ ПО ВЫПОЛНЕНИЮ КОНТРОЛЬНОЙ РАБОТЫ ПО ДИСЦИПЛИНЕ УПРАВЛЕНИЕ ДАННЫМИ

Направление подготовки	**09.03.02 Информационные системы и технологии**
Квалификация выпускника	Бакалавр

Пятигорск, 2025

Содержание

Введение

Методические указания содержат перечень вариантов заданий для контрольных работ, требования к оформлению контрольных работ и пример выполнения задания. Теоретической основой подготовки специалиста являются знания в области информатики, вычислительной систем.

1. Цель, задачи и реализуемые компетенции

Методические указания составлены с учетом требований стандарта высшего образования по дисциплине: «Управление данными».

Целью изучения дисциплины является приобретение знаний об архитектурах банков данных, основных моделях баз данных, языках описания и манипулирования данными; современных методах проектирования реляционных баз данных, одной из конкретных СУБД MS SQL Server.

Задачи изучения дисциплины: формирование представлений об архитектуре банка данных, моделях баз данных; основных конструкциях языков описания и манипулирования данными, этапах проектирования баз данных, методах проектирования реляционных баз данных, подходах к составлению приложений для базы данных, способах обеспечения защиты и целостности баз данных.

2. Формулировка задания и его объем

Контрольная работа включат в себе пять заданий по темам, которые нужно изучить самостоятельно и по ним подготовить контрольную работу. Результаты выполнения контрольной работы предоставляются в электронном виде. Объем контрольной работы составляет 15-20 печатных листов формата А4.

Вариант задания выбирается по двум последним цифрам зачетной книжки из каждого предложенного задания.

3. Общие требования к написанию и оформлению работы

Контрольная работа выполняется и сдается в электронном виде на CD/CDRW носителе. На конверте необходимо указать название дисциплины, ФИО студента, факультет, номер группы, шифр зачетной книжки, № варианта задания, и список всех созданных в ходе выполнения задания файлов.

Приведенный в конце методических указаний список литературы может использоваться студентами при выполнении контрольной работы.

4. Варианты заданий для студентов заочной формы обучения

Задание 1. Разработка структуры БД, ввод и корректировка БД.

Порядок выполнения работы:

1. создать таблицу и ввести в нее данные,
2. выполнить просмотр и корректировку созданной структуры и введенных данных,
3. осуществлять редактирование, вставку и удаление записей таблицы посредством меню,

Каждый студент выполняет вариант согласно порядковому номеру в журнале.

Количество записей, введенных в базу данных, должно быть не меньше 15.

Варианты заданий

Вариант 1

О проданных товарах имеется следующая информация:

- артикул товара (5 знаков),
- наименование товара (20 знаков),
- дата начала продажи товара,
- количество проданного товара (4 знака),
- цена за единицу товара (в тенге и тиынах 6 знаков)

Вариант 2

О расходовании электроэнергии на заводах имеется следующая информация:

- номер завода (4 знака),
- ф.и.о. директора (25 знаков),
- ф.и.о. главного энергетика (25 знаков),
- дата отчета,

- количество израсходованной электроэнергии (в тыс. квт*ч, 5 знаков)

Вариант 3

Даны сведения о заболеваемости студентов:

- наименование группы (8 знаков),
- факультет (6 знаков),
- дата медосмотра,
- фамилия студента (25 знаков),
- код диагноза (5 знаков)

Вариант 4

Даны сведения о рабочих завода:

- код предприятия (4 знака),
- номер цеха (2 знака),
- табельный номер рабочего (5 знаков),
- дата поступления на работу,
- пол (1 знак)

Вариант 5

Имеется следующая информация о результатах отладки программ для курсовых проектов:

- шифр группы (8 знаков),
- наименование дисциплины (20 знаков),
- дата отчета,
- количество студентов в группе (2 знака),
- число отлаженных программ (2 знака)

Вариант 6

Имеется следующая информация о наличии технических средств на добычных участках:

- дата отчета,
- номер участка (2 знака),
- количество комбайнов (2 знака),
- количество бункеров-перегружателей (2 знака),

- количество самоходных вагонов (2 знака)

Вариант 7

Имеется следующая информация о расходовании топлива в автоколоннах:

- номер колонны (2 знака),
- количество автомашин (2 знака),
- марка автомашины (10 знаков),
- дата отчета,
- расход топлива (3 знака)

Вариант 8

Имеется следующая информация о городских маршрутах:

- тип маршрута (15 знаков - троллейбус, трамвай, автобус),
- код маршрута (1 знак),
- дата начала эксплуатации,
- номер маршрута (2 знака),
- число подвижного состава (3 знака)

Вариант 9

Имеется следующая информация об авиарейсах:

- номер рейса (8 знаков),
- наименование маршрута(20 знаков),
- дата ввода в эксплуатацию,
- марка самолета (5 знаков),
- стоимость билета (в долларах - 3 знака)

Вариант 10

Имеется следующая информация о сотрудниках магазина ЦУМ:

- шифр магазина (2 знака),
- наименование отдела (20 знаков),
- код отдела (2 знака),
- количество продавцов в отделе (2 знака),
- средняя заработная плата по отделу (в долларах 3 знака).

Вариант 11

Имеется следующая информация о студентах, занимающихся спортом:

- фамилия студента (25 знаков),
- шифр группы (8 знаков),
- код вида спорта (2 знака),
- наименование вида спорта (20 знаков),
- разряд (1 знак).

Вариант 12

Имеется следующая информация о прохождении студентами практики на предприятиях:

- страна (20 знаков),
- название города (20 знаков),
- наименование предприятия (20 знаков),
- код предприятия (4 знака),
- шифр группы (8 знаков),
- ф.и.о. студента (20 знаков).

Вариант 13

Имеется следующая информация о пропусках занятий студентами:

- шифр группы (8 знаков),
- фамилия студента (20 знаков),
- пропущено часов (3 знака),
- оправдано часов (3 знака).

Вариант 14

Имеется следующая информация о сдаче студентами сессии:

- сессия (зимняя, весенняя - 10 знаков),
- шифр группы (8 знаков),
- количество студентов, сдавших сессию досрочно (3 знака),
- количество студентов, не сдавших сессию в срок (3 знака),
- количество студентов, сдавших сессию вовремя (3 знака).

Вариант 15

Имеется следующая информация о породах собак:

- фамилия владельца (25 знаков),
- код породы (3 знака),
- наименование породы (25 знаков),
- количество медалей (2 знака),
- наименование клуба (25 знаков).

Вариант 16

Имеется следующая информация о ПЭВМ:

- код предприятия (5 знаков),
- наименование предприятия (20 знаков),
- тип компьютера (20 знаков),
- количество компьютеров (2 знака),
- страна-изготовитель (25 знаков).

Вариант 17

Имеется следующая информация о предметах, изучаемых в семестре:

- шифр группы (8 знаков),
- количество предметов (2 знака),
- количество экзаменов (2 знака),
- количество зачетов (2 знака),
- количество курсовых проектов (1 знак).

Вариант 18

Имеется следующая информация о сделанных покупках:

- наименование покупки (25 знаков),
- наименование магазина (25 знаков),
- количество (5 знаков),
- цена (5 знаков).

Вариант 19

Имеется следующая информация об операциях, сделанных в больнице:

- фамилия больного (25 знаков),
- фамилия врача (25 знаков),
- наименование операции (25 знаков),
- код операции (2 знака),
- номер палаты (2 знака).

Вариант 20

Имеется следующая информация об использовании ПЭВМ в ауд.430:

- номер машины (2 знака),
- тип компьютера (20 знаков),
- дата (8 знаков),
- время загрузки (2 знака).

Вариант 21

Имеется следующая информация о распределении студентов:

- фамилия студента (25 знаков),
- шифр группы (8 знаков),
- наименование предприятия (25 знаков),
- код предприятия (4 знака),
- средний балл в приложении к диплому (2 знака).

Вариант 22

Имеется следующая информация о кооперативах в нашем городе:

- фамилия председателя (25 знаков),
- шифр кооператива (3 знака),
- наименование кооператива (25 знаков),
- тип (4 знака - строительный, швейный и т.д.),
- годовая прибыль (7 знаков).

Вариант 23

Имеется следующая информация о составе групп:

- фамилия студента (25 знаков),
- шифр группы (8 знаков),

- количество экзаменов, сданных на 5 (1 знак),

- количество экзаменов, сданных на 4 (1 знак),

- количество экзаменов, сданных на 3 (1 знак).

Вариант 24

Имеется следующая информация о погоде на квартал:

- общее количество дней (3 знака),

- количество солнечных дней (3 знака),

- количество пасмурных дней (3 знака),

- фамилия метеоролога (25 знаков),

- средняя температура (2 знака).

Вариант 25

Имеется следующая информация о распределении мест на теннисных соревнованиях:

- фамилия теннисиста (25 знаков):

- шифр группы (2 знака),

- количество выигрышей (3 знака),

- количество проигрышей (3 знака),

- место, занимаемое в общем списке (3 знака).

Контрольные вопросы:

1. Основные функции СУБД.
2. Что такое БД?
3. Назначение таблиц?
4. Управление таблицами
5. Отношения между таблицами
6. Команды добавления, удаления, поиска, перемещения по записям.
7. Отличие простого индекса от составного. Различные типы индексов.

Контрольные задания для СРС.

1. Создание простых и составных индексов.

2. Перемещение по таблице и поиск данных по определенному критерию.

Задание 2. Организация циклов. Индексные файлы.

Порядок выполнения работы:

1. Создать две таблицы и ввести в них данные, используя маски ввода, свойство значения по умолчанию согласно варианту (в качестве ключа назначьте данные, выделенные символом подчеркивания).
2. Определите условия целостности и достоверности вводимых данных.

Варианты заданий

Вариант 1

Код организации, наименование организации

Код организации, дата отгрузки, кол-во товара, артикул товара, номер накладной

Вариант 2

Код цеха, наименование цеха, ФИО начальника

Таб. номер, код цеха, кол-во отработанных часов

Вариант 3

Код сбербанка, наименование сбербанка

Таб. номер сотрудника, Код сбербанка, Лицевой счет, % перечисления

Вариант 4

Код цеха, наименование цеха, ФИО начальника

Таб. Номер, код цеха, кол-во отработанных дней

Вариант 5

Код организации, наименование организации, ФИО экспедитора, номер машины, марка машины

№ накладной, дата, код организации, артикул товара, кол-во товара, признак (отгрузка=1/прием=0)

Вариант 6

Дата продажи, дата отправления, ФИО пассажира, кол-во билетов, № рейса, номер билета

Вариант 7

Код продукта, наименование продукта

Код блюда, код продукта, кол-во продукта

№ заказа, код блюда, кол-во порций, ФИО

Вариант 8

Код работника, ФИО получателя платежа, адрес получателя, процент отчислений

Код цеха, наименование

Вариант 9

Индекс издания, Наименование издания, цена

Код подписчика, индекс издания, дата подписки, период подписки, кол-во экземпляров

Вариант 10

№ накладной, код товара, кол-во, дата, код склада

Код склада, наименование, ФИО кладовщика

Вариант 11

Дата поступления, код товара, № склада, кол-во, инвент.номер

№ склада, наименование, ФИО кладовщика

Вариант 12

Код книги, наименование, ФИО автора

Код читателя, код книги, дата выдачи, дата возврата

Вариант 13

Код читателя, ФИО, адрес, № телефона, дата рождения

Код книги, код читателя, дата выдачи, дата возврата

Код автора, ФИО автора

Вариант 14

Код пользователя, ФИО, адрес, пол

Код пользователя, Код фильма, дата выдачи, дата возврата, сумма залога

Вариант 15

№ накладной, дата отгрузки, код водителя, кол-во товара, код товара

Код товара, цена за ед. измерения, ед. измерения

Вариант 16

Код пользователя, Код диска, дата выдачи, дата возврата

Код пользователя, ФИО, адрес, телефон, пол, сумма залога

Вариант 17

Код врача, ФИО врача, специальность

Дата назначения, код врача, код лекарства, ФИО больного

Вариант 18

№ накладной, код организации, код товара, кол-во

Код товара, наименование, ед. измерения, цена

Код банка, наименование банка

Вариант 19

Код врача, ФИО врача, специальность врача

Код пациента, код врача, дата приема, время приема

Вариант 20

Код жильца, ФИО, адрес

Код жильца, код издания, кол-во экземпляров

Код автора, наименование автора

Вариант 21

Код закройщика, ФИО закройщика

Код изделия, наименование изделия, цена, тип изделия (верхняя одежда, легкое платье…)

Вариант 22

Код страхуемого лица, ФИО, адрес, код страховки, дата страховки, код страховщика

Код страховщика, ФИО

Вариант 23

Код жильца, дата взноса, период взноса, сумма, код КСК

Код КСК, наименование, ФИО бухгалтера

Вариант 24

Код налоговой инспекции, наименование

Код налоговой инспекции, код налогоплательщика, дата

Вариант 25

Номер автомобиля, код марки автомобиля, номер двигателя, номер шасси, код цвета

Код цвета, наименование цвета

Код марки автомобиля, наименование

Контрольные вопросы:

1. Понятие индекса. Виды индексов.
2. Способы сортировки информации.
3. Основные типы связей между таблицами.
4. Что такое ключевые поля?
5. Как установить (разорвать) связи между таблицами?
6. Способы определения условий целостности.
7. Механизмы определение условий достоверности данных.
8. Для чего используется представление данных?
9. Способы редактирования данных при помощи представлений.

Контрольные задания для СРС.

1. На основании таблицы создать представление для редактирования одного из полей таблицы в режиме просмотра.

Задание 3. Объекты и управление объектами СУБД. Работа с таблицами БД.

Порядок выполнения работы:

1. Создать форму, содержащую элементы управления и реализовать с помощью них просмотр и редактирование данных из задания №1;
2. Осуществить экспорт одной из таблиц вашей БД в файл формата Microsoft Excel;
3. Создать какую-либо таблицу в Microsoft Excel и проимпортировать эти данные в вашу базу данных;
5. Внедрить объект OLE (по вашему усмотрению) в таблицу БД;

Варианты заданий

Вариант 1

Вывести значения всех полей базы данных на экран. Подсчитать итог: общее количество товара и его стоимость.

Вариант 2

Вывести значения всех полей базы данных на экран.
Подсчитать итог: общее количество заводов и расход энергии.

Вариант 3

Вывести значения всех полей базы данных на экран.

Подсчитать итог: общее количество больных студентов и общее число дней нетрудоспособности.

Вариант 4

Вывести значения всех полей базы данных на экран.
Подсчитать итог: количество женщин и мужчин.

Вариант 5

Вывести значения всех полей базы данных на экран.
Подсчитать итог: общее количество студентов и общее число отлаженных программ.

Вариант 6

Вывести значения всех полей базы данных на экран.

Подсчитать итог: общее количество каждого вида оборудования.

Вариант 7

Вывести значения всех полей базы данных на экран.
Подсчитать итог: общее количество автомашин и расход топлива.

Вариант 8

Вывести значения всех полей базы данных на экран. Подсчитать итог: общее количество маршрутов и подвижного состава.

Вариант 9

Вывести значения всех полей базы данных на экран.
Подсчитать итог: общее количество рейсов и самолетов.

Вариант 10

Вывести значения всех полей базы данных на экран.
Подсчитать итог: количество продавцов и общую сумму месячной заработной платы.

Вариант 11

Вывести значения всех полей базы данных на экран.
Подсчитать итог: количество студентов, занимающихся спортом и количество видов спорта.

Вариант 12

Вывести значения всех полей базы данных на экран.
Подсчитать итог: количество стран или количество городов и количество студентов на практике.

Вариант 13

Вывести значения всех полей базы данных на экран.
Подсчитать итог: количество студентов и количество пропущенных часов.

Вариант 14

Вывести значения всех полей базы данных на экран.

Подсчитать итог: количество студентов и количество групп.

Вариант 15

Вывести значения всех полей базы данных на экран.
Подсчитать итог: количество пород и общее количество собак.

Вариант 16

Вывести значения всех полей базы данных на экран.
Подсчитать итог: количество предприятий и компьютеров.

Вариант 17

Вывести значения всех полей базы данных на экран.
Подсчитать итог: количество групп и среднее количество экзаменов.

Вариант 18

Вывести значения всех полей базы данных на экран. Подсчитать итог: общее количество покупок и их стоимость.

Вариант 19

Вывести значения всех полей базы данных на экран.
Подсчитать итог: общее количество больных и врачей.

Вариант 20

Вывести значения всех полей базы данных на экран.
Подсчитать итог: количество машин и общее время загрузки.

Вариант 21

Вывести значения всех полей базы данных на экран.

Подсчитать итог: общее количество студентов и предприятий.

Вариант 22

Вывести значения всех полей базы данных на экран.

Подсчитать итог: количество кооперативов и общая годовая прибыль.

Вариант 23

Вывести значения всех полей базы данных на экран.
Подсчитать итог: количество групп и студентов.

Вариант 24

Вывести значения всех полей базы данных на экран.

Подсчитать итог: количество метеорологов и среднее количество пасмурных дней.

Вариант 25

Вывести значения всех полей базы данных на экран.

Подсчитать итог: количество спортсменов и общее количество игр.

Контрольные вопросы:

1. Что такое объекты в MS SQL Server?
2. Чем отличаются друг от друга контейнеры и средства управления?
3. Какими путями можно создавать объекты?
4. Как осуществляется доступ к свойствам объекта?
5. Каким образом можно связать объект с данными?
6. Что могут содержать в себе контейнеры?
7. Какие объекты менее доступны для модификации и менее гибки?
8. Объясните понятия метода и события объекта?
9. Основные способы работы с файлами других форматов данных?
10. С какими форматами и базами данных может взаимодействовать?
11. Как связать OLE-объект с объектом на сервере, какие способы существуют?
12. Как осуществляется связывание объектов?
13. Как осуществляется внедрение объектов?
14. Как осуществляется экспорт объектов?
15. Как осуществляется импорт объектов?
16. Назначение макроса. Способы его создания

Контрольные задания для СРС.

1. Осуществить экспорт таблицы вашей БД в файл формата Microsoft Word;
2. Создать макрос.

Задание 4. Запросы в SQL. Работа с запросами

Порядок выполнения работы:

Используя базу данных, созданную в задания №1:

1. Создать запрос на выборку, согласно индивидуальному варианту
2. Создайте запрос по своему усмотрению с вычисляемым полем на основании данных, взятых из созданных вами ранее таблиц.

Варианты заданий

Вариант 1

Результат запроса должен содержать всю имеющуюся информацию о товарах, проданных после определённой в запросе даты.

Вариант 2

Результат запроса должен содержать информацию о номерах заводов, расход электроэнергии на которых превысил определённый лимит, и о количестве израсходованной электроэнергии.

Вариант 3

Результат запроса должен содержать все имеющиеся сведения о студентах, прошедших медосмотр в определённый промежуток времени.

Вариант 4

Результат запроса должен содержать все имеющиеся сведения о рабочих – мужчинах, поступивших на работу позднее определённого срока.

Вариант 5

Результат запроса должен содержать информацию о результатах отладки программ по какой-либо одной дисциплине.

Вариант 6

Результат запроса должен содержать дату отчета, номер участка и количество комбайнов для всех участков, данные отсортировать по участкам.

Вариант 7

Результат запроса должен содержать номер колонны, марку автомашины и расход топлива только тех колонн, расход топлива в которых превысил какое-то определенное значение.

Вариант 8

Результат запроса должен содержать все имеющиеся данные о маршрутах, эксплуатация которых началась раньше определённого срока.

Вариант 9

Результат запроса должен содержать информацию об авиарейсах, кроме стоимости билета, которые были введены в эксплуатацию раньше определенного срока.

Вариант 10

Результат запроса должен содержать всю имеющуюся информацию о сотрудниках магазина, работающих в отделах, где количество продавцов больше 1.

Вариант 11

Результат запроса должен содержать всю имеющуюся информацию о студентах, занимающихся спортом и имеющих первый спортивный разряд.

Вариант 12

Результат запроса должен содержать всю имеющуюся информацию о прохождении студентами практики за пределами Казахстана.

Вариант 13

Результат запроса должен содержать всю имеющуюся информацию о студентах, разница между пропущенными и оправданными часами которых больше определённого числа.

Вариант 14

Результат запроса должен содержать всю имеющуюся информацию о сдаче студентами весенней сессии.

Вариант 15

Результат запроса должен содержать всю имеющуюся информацию о собаках определённой породы.

Вариант 16

Результат запроса должен содержать всю имеющуюся информацию о ПЭВМ на предприятии, имеющем определённый код.

Вариант 17

Результат запроса должен содержать всю имеющуюся информацию о предметах для определённой группы.

Вариант 18

Результат запроса должен содержать всю имеющуюся информацию о покупках, сделанных в каком-либо одном магазине.

Вариант 19

Результат запроса должен содержать всю имеющуюся информацию об операциях, каким-либо одним врачом.

Вариант 20

Результат запроса должен содержать всю имеющуюся информацию об использовании ПЭВМ в ауд.430, время загрузки которых превышает 90 минут.

Вариант 21

Результат запроса должен содержать всю имеющуюся информацию о распределении студентов какой-либо одной группы.

Вариант 22

Результат запроса должен содержать всю имеющуюся информацию о кооперативах, прибыль которых превысила определённую сумму.

Вариант 23

Результат запроса должен содержать всю имеющуюся информацию о студентах, которые сдали на 3 больше 2-х экзаменов.

Вариант 24

Результат запроса должен содержать всю имеющуюся информацию о погоде за кварталы, солнечных дней в которых было больше определённого количества.

Вариант 25

Результат запроса должен содержать всю имеющуюся информацию о теннисистах, занимающих три первых места в турнирной таблице.

Контрольные вопросы:

1. Для чего предназначен SQL?
2. В каком случае игнорируется опция TO команды SELECT?
3. Объясните разницу между клиентом и сервером.
4. Чем отличаются локальные и удалённые виды?
5. Назначение запросов?
6. Основные виды запросов?
7. Как можно создать запрос?
8. Как определяются заголовки столбцов, используемых в работе с запросом?
9. Может ли вид включать в себя другие виды?
10. Для чего используются вычисляемые поля?
11. Как произвести итоговые вычисления по группам записей?

Контрольные задания для СРС.

1. Создать запрос по своему усмотрению на основании данных, взятых из созданных ранее таблиц с расчетом итоговых значений.

Задание 5. Работа с формами, отчётами, меню и панелями инструментов.

Порядок выполнения работы:

Используя базу данных, созданную в лабораторной работе №1, необходимо:

1. Отредактировать созданную форму, (должны быть реализованы следующие функции: ввод записей в уже существующую таблицу, просмотр данных, корректировка отдельных полей, удаление).
2. Создать меню, дублирующее панель инструментов на форме.
3. Создать отчет согласно индивидуальному варианту в табличном виде на основании таблицы, созданной в лабораторной работе 1.

Отчеты должны содержать заголовок, дату и время создания, сопроводительный текст (автор отчета), нумерацию страниц и вычисление итоговых результатов, где это возможно.

Варианты заданий

Вариант 1

Отчёт должен содержать все данные о товарах и итоговую сумму продаж.

Вариант 2

Отчёт должен содержать все данные о расходовании электроэнергии на заводах, и итоговую сумму расходованной электроэнергии.

Вариант 3

Отчёт должен содержать все данные о заболеваемости студентов и количество студентов, прошедших медосмотр по каждому факультету.

Вариант 4

Отчёт должен содержать все данные о рабочих и количество рабочих в каждом цехе.

Вариант 5

Отчёт должен содержать все данные о результатах отладки программ и число отлаженных программ в каждой группе.

Вариант 6

Отчёт должен содержать все данные о наличии технических средств на добычных участках и общее количество комбайнов, бункеров-перегружателей и самоходных вагонов.

Вариант 7

Отчёт должен содержать все данные о расходовании топлива в автоколоннах и итоговую сумму расхода топлива.

Вариант 8

Отчёт должен содержать все данные о городских маршрутах и общее количество троллейбусов, трамваев и автобусов.

Вариант 9

Отчёт должен содержать все данные об авиарейсах и количество самолётов каждой марки в отдельности, совершающих рейсы.

Вариант 10

Отчёт должен содержать все данные о сотрудниках магазина ЦУМ и общее количество продавцов.

Вариант 11

Отчёт должен содержать все данные о студентах, занимающихся спортом и их общее количество.

Вариант 12

Отчёт должен содержать все данные о прохождении студентами практики на предприятиях и количество студентов, проходящих практику на каждом предприятии.

Вариант 13

Отчёт должен содержать все данные о пропусках занятий студентами и количество пропусков по каждой группе.

Вариант 14

Отчёт должен содержать все данные о сдаче студентами сессии и количество студентов, сдавших каждую сессию досрочно.

Вариант 15

Отчёт должен содержать все данные о породах собак и количество собак каждой породы.

Вариант 16

Отчёт должен содержать все данные о ПЭВМ и общее количество компьютеров каждого типа.

Вариант 17

Отчёт должен содержать все данные о предметах, изучаемых в семестре

Вариант 18

Отчёт должен содержать все данные о сделанных покупках и общее количество покупок каждого наименования.

Вариант 19

Отчёт должен содержать все данные об операциях, сделанных в больнице и количество операций каждого наименования.

Вариант 20

Отчёт должен содержать все данные об использовании ПЭВМ в ауд.430 и количество компьютеров каждого типа.

Вариант 21

Отчёт должен содержать все данные о распределении студентов и количество студентов по каждому предприятию.

Вариант 22

Отчёт должен содержать все данные о кооперативах в нашем городе и количество кооперативов каждого типа.

Вариант 23

Отчёт должен содержать все данные о составе групп и количество студентов в каждой группе.

Вариант 24

Отчёт должен содержать все данные о погоде на квартал, общее количество солнечных дней во всех кварталах, общее количество пасмурных дней во всех кварталах.

Вариант 25

Отчёт должен содержать все данные о распределении мест на теннисных соревнованиях.

Контрольные вопросы:

1. Назначение форм.
2. Способы создания форм.
3. В чем заключается различие между формами и представлением данных в режиме таблицы?
4. Что такое среда данных?
5. Что такое элементы управления?
6. Основные типы элементов управления?
7. Как определяются свойства элементов управления?
8. Как добавить в форму элемент управления?
9. Назначение и типы отчетов.
10. Способы создания отчетов.
11. Типы полос конструктора отчетов и их назначение.
12. Вычисляемые и итоговые поля в отчетах
13. Группировка данных в отчетах
14. Что включает в себя система меню?
15. Как определяются процедуры для пунктов меню?
16. Отличие «горячих клавиш» меню от клавиш ускоренного действия

Контрольные задания для СРС.

1. Создать отчет в свободной форме на основании одного из запросов, созданных в задание 3.

5. План-график выполнения задания

Дата получения задания	Дата предоставления выполненного задания
Установочная сессия.	Экзаменационная сессия за две недели до начала.

6. Критерии оценивания работы

Оценка «отлично» выставляется студенту, если он продемонстрировал глубокие, исчерпывающие знания и творческие способности в понимании, изложении и использовании учебно-программного материала; логически последовательные, содержательные, полные, правильные и конкретные ответы на все поставленные вопросы и дополнительные вопросы преподавателя; свободное владение основной и дополнительной литературой, рекомендованной учебной программой.

Оценка «хорошо» выставляется студенту, если он продемонстрировал твердые и достаточно полные знания всего программного материала, правильное понимание сущности и взаимосвязи рассматриваемых процессов и явлений; последовательные, правильные, конкретные ответы на поставленные вопросы при свободном устранении замечаний по отдельным вопросам; достаточное владение литературой, рекомендованной учебной программой.

Оценка «удовлетворительно» выставляется студенту, если он продемонстрировал твердые знания и понимание основного программного материала; правильные, без грубых ошибок ответы на поставленные вопросы при устранении неточностей и несущественных ошибок в освещении отдельных положений при наводящих вопросах преподавателя; недостаточное владение литературой, рекомендованной учебной программой.

Оценка «неудовлетворительно» выставляется студенту, если он продемонстрировал неправильные ответы на основные вопросы, допущены грубые ошибки в ответах, непонимание сущности излагаемых вопросов; неуверенные и неточные ответы на дополнительные вопросы.

7. Порядок защиты работы

Защита контрольной работы проводится в виде научного дискоса с презентацией выполненных заданий, в соответствии с графиком защиты. После доклада студенту задаются вопросы как преподавателем, так и студентами группы.

В процессе защиты своей работы студент делает доклад продолжительностью 7-10 минут. Доклад должен быть предварительно подготовлен студентом. Лучшее впечатление производит доклад, в форме пересказа, без зачтения текста, которым следует пользоваться только для уточнения цифрового материала. Студент должен свободно ориентироваться в своей работе.

В выступлении необходимо корректно использовать демонстрационные материалы, которые усиливают доказательность выводов и облегчают восприятие доклада студента. Они оформляются в виде презентации в системе Power Point.

8. Учебно-методическое и информационное обеспечение дисциплины

8.1. Перечень основной и дополнительной литературы, необходимой для освоения дисциплины

8.1.1. Перечень основной литературы:

1. Васильев, В. В. Информационные технологии в библиотечном деле : учеб.-метод. пособие / Отв. ред. серии О. Р. Бородин. – М. : Либерея-Бибинформ, 2013. – 368 с.
2. Ларсон, Б. Разработка бизнес-аналитики в Microsoft SQL Server. – СПб. – М. – Н. Новгород : Питер, 2011. – 688 с.
3. Макин, Дж. К. Проектирование серверной инфраструктуры баз данных Microsoft SQL SERVER : учебный курс : пер. с англ. / Дж. К. Макин, Майк Хотек. – М. : Русская редакция, 2012. – 560 с.

8.1.2. Перечень дополнительной литературы:

1. Избачков, Ю. С. Информационные системы: учебник / Ю. С. Избачков, В. Н. Петров. - 3-е изд. - СПб.: Питер, 2011. - 656 с.
2. Томас, О. Оптимизация и администрирование баз данных Microsoft SQL Server 2008 : официальное пособие для самоподготовки : пер. с англ. : учебный курс Microsoft. – М. : Русская редакция, 2012. – 601 с.
3. Основы проектирования и разработки реляционных баз данных : (Спец. 075200 – Компьютерная безопасность) : учеб. пособие / авт.-сост.: О. М. Лепешкин, Д. Л. Осипов ; Федеральное агентство по образованию, Ставроп. гос. ун-т. – Ставрополь : Изд-во СГУ, 2010. – 203 с.

8.2. Перечень учебно-методического обеспечения самостоятельной работы обучающихся по дисциплине:

1. Методические указания по выполнению лабораторных работ по дисциплине «Управление данными».
2. Методические указания по выполнению контрольной работы по дисциплине «Управление данными».
3. Методические рекомендации для студентов по организации самостоятельной работы по дисциплине «Управление данными».
4. Методические указания по выполнению курсового проекта по дисциплине «Управление данными».

8.3. Перечень ресурсов информационно-телекоммуникационной сети «Интернет», необходимых для освоения дисциплины:

1. http://www.intuit.ru – сайт дистанционного образования в области информационных технологий
2. http://e.lanbook.com/books/element.php?pl1_cid=25&pl1_id=5196 – ЭБС издательства «Лань». Соколова Ю.С., Жулева С.Ю. Разработка приложений в среде Delphi. В 2 частях.
3. http://www.intuit.ru/ – сайт дистанционного образования в области информационных технологий
4. https://msdn.microsoft.com/ru-ru/library/bb545450.aspx – сайт Библиотека Microsoft SQL Server - MSDN