|  |
| --- |
|  |
| |  | | --- | | МИНОБРНАУКИ РОССИИ | | Федеральное государственное. бюджетное образовательное учреждение  высшего образования  **«МИРЭА – Российский технологический университет»**  **РТУ МИРЭА**  **Филиал РТУ МИРЭА в г. Ставрополе** | |
|  |

**МЕТОДИЧЕСКИЕ УКАЗАНИЯ**

**к лабораторным работам и самостоятельной работе**

**по дисциплине «Базы данных» для студентов**

**направления подготовки 09.03.01 Информатика и**

**вычислительная техника**

**Квалификация: специалист**

Ставрополь

Методические указания к выполнению лабораторных и самостоятельных работ составлены в соответствии федеральным государственным образовательным стандартом высшего профессионального образования по специальности 09.03.01 Информатика и вычислительная техника.

Составители: А.В.Горелова, В.И.Дроздова, д.ф.-м.н., профессор

**СОДЕРЖАНИЕ**

[Указания по технике безопасности 5](#_Toc83804616)

[Лабораторная работа №1. АРХИТЕКТУРА РАСПРЕДЕЛЕННЫХ INTERNET](#_Toc83804617)

[ПРИЛОЖЕНИЙ. УСТАНОВКА ПАКЕТА DENWER 7](#_Toc83804618)

[Лабораторная работа №2. ОСНОВНАЯ ИНФОРМАЦИЯ О MYSQL. СОЗДАНИЕ БАЗЫ ДАННЫХ 13](#_Toc83804619)

[Лабораторная работа №3. ОПЕРАТОРЫ И КОМАНДЫ MySQL 19](#_Toc83804620)

[Лабораторная работа № 4. SQL ЗАПРОСЫ. БЕЗУСЛОВНАЯ ВЫБОРКА 29](#_Toc83804621)

[Лабораторная работа № 5. РАСШИРЕННЫЕ SQL-ЗАПРОСЫ. ВЫБОРКА С УСЛОВИЕМ 33](#_Toc83804622)

[Лабораторная работа № 6. РАСШИРЕННЫЕ SQL-ЗАПРОСЫ. ВЫБОРКА ВЫЧИСЛЯЕМЫХ ЗНАЧЕНИЙ 38](#_Toc83804623)

[Лабораторная работа № 7. РАСШИРЕННЫЕ SQL-ЗАПРОСЫ. ВЫБОРКА ИЗ СВЯЗАННЫХ ТАБЛИЦ 41](#_Toc83804624)

[Лабораторная работа № 8. ИСПОЛЬЗОВАНИЕ ГРУППИРОВКИ ДАННЫХ ПРИ ОРГАНИЗАЦИИ SQL- ЗАПРОСОВ 44](#_Toc83804625)

[Лабораторная работа № 9. CОРТИРОВКА 47](#_Toc83804626)

[Лабораторная работа № 10. РАСШИРЕННЫЕ SQL-ЗАПРОСЫ.ИСПОЛЬЗОВАНИЕ КВАНТОРА СУЩЕСТВОВАНИЯ В ЗАПРОСАХ. КОРРЕЛИРОВАННЫЕ И](#_Toc83804627)

[НЕКОРРЕЛИРОВАННЫЕ ПОДЗАПРОСЫ 49](#_Toc83804628)

[Лабораторная работа № 11 МОДИФИКАЦИЯ ДАННЫХ В ТАБЛИЦАХ 52](#_Toc83804629)

[Лабораторная работа № 12 ОПРЕДЕЛЕНИЕ ОТНОШЕНИЙ МЕЖДУ ТАБЛИЦАМИ](#_Toc83804631)

[ПРОВЕРКА УСЛОВИЙ ЦЕЛОСТНОСТИ ДАННЫХ 56](#_Toc83804632)

[Лабораторная работа № 13 ОБЪЕДИНЕНИЕ МНОЖЕСТВ 60](#_Toc83804633)

[Лабораторная работа № 14 СОЕДИНЕНИЕ ОТНОШЕНИЙ. ОПЕРАТОР JOIN 63](#_Toc83804635)

[Лабораторная работа № 15 АДМИНИСТРИРОВАНИЕ БД. ПРИВИЛЕГИИ.](#_Toc83804637)

[ПРЕДСТАВЛЕНИЯ 69](#_Toc83804638)

[Лабораторная работа № 16 СОСТАВЛЕНИЕ ТЕХНИЧЕСКОГО ЗАДАНИЯ НА](#_Toc83804639)

[РАЗРАБОТКУ САЙТА 73](#_Toc83804640)

[Лабораторная работа № 17. ФУНКЦИИ PHP ДЛЯ РАБОТЫ С MySQL 76](#_Toc83804641)

[Лабораторная работа №18. УДАЛЕНИЕ ДАННЫХ И ОБНОВЛЕНИЕ ДАННЫХ В БД 89](#_Toc83804642)

[Лабораторная работа №19. ИЗУЧЕНИЕ PHP CЕССИЙ 92](#_Toc83804644)

[Лабораторная работа № 20. ПЕРЕДАЧА ПЕРЕМЕННЫХ ЧЕРЕЗ URL 96](#_Toc83804646)

[Лабораторная работа №21. ИСПОЛЬЗОВАНИЕ ВНЕШНЕГО ФАЙЛА ДЛЯ ХРАНЕНИЯ И ЗАГРУЗКИ ВНЕШНИХ ДАННЫХ 98](#_Toc83804647)

[Лабораторная работа №22. PHP И ПОЛЯ HTML-ФОРМ: ТЕКСТОВЫЕ ПОЛЯ, ТЕКСТОВАЯ ОБЛАСТЬ 101](#_Toc83804648)

[Лабораторная работа №23. PHP И ПОЛЯ HTML-ФОРМ: ФЛАЖКИ, ПЕРЕКЛЮЧАТЕЛИ 108](#_Toc83804649)

[Лабораторная работа №24. PHP И ПОЛЯ HTML-ФОРМ: СПИСКИ 116](#_Toc83804650)

[Лабораторная работа №25. PHP И ПОЛЯ HTML-ФОРМ: СКРЫТЫЕ ПОЛЯ ФОРМ 120](#_Toc83804651)

[Лабораторная работа № 26. PHP И ПОЛЯ HTML-ФОРМ: ПОЛЯ ВВОДА ПАРОЛЕЙ, КНОПКИ, ЗНАЧЕНИЯ, ВОЗВРАЩАЕМЫЕ ФОРМАМИ, ПРОВЕРКА ОБЯЗАТЕЛЬНЫХ ПОЛЕЙ 124](#_Toc83804652)

[Лабораторная работа №27. ИСПОЛЬЗОВАНИЕ СТАНДАРТНЫХ ОПЕРАТОРОВ ЯЗЫКА PHP ПРИ ОБРАБОТКЕ ДАННЫХ ПОЛЬЗОВАТЕЛЯ ИЗ ФОРМ: БУЛЕВЫ ОПЕРАТОРЫ, IF, ОПЕРАТОРЫ СРАВНЕНИЯ, ЛОГИЧЕСКИЕ ОПЕРАТОРЫ 133](#_Toc83804653)

[Лабораторная работа №28. ИСПОЛЬЗОВАНИЕ СТАНДАРТНЫХ ОПЕРАТОРОВ ЯЗЫКА PHP ПРИ ОБРАБОТКЕ ДАННЫХ ПОЛЬЗОВАТЕЛЯ ИЗ ФОРМ: ОПЕРАТОР SWITCH 142](#_Toc83804654)

[Лабораторная работа № 29. ИСПОЛЬЗОВАНИЕ СТАНДАРТНЫХ ОПЕРАТОРОВ ЯЗЫКА PHP ПРИ ОБРАБОТКЕ ДАННЫХ ПОЛЬЗОВАТЕЛЯ ИЗ ФОРМ: ИСПОЛЬЗОВАНИЕ ЦИКЛОВ 149](#_Toc83804655)

[Лабораторная работа №30. ПРЕДОТВРАЩЕНИЕ КАТАСТРОФ И ВОССТАНОВЛЕНИЕ 154](#_Toc83804656)

[Лабораторная работа № 31. РЕПЛИКАЦИЯ В MySQL 161](#_Toc83804657)

[Лабораторная работа № 32. SQL-команды, относящиеся к репликации. Проблемы и распространённые ошибки MySQL 179](#_Toc83804658)

[СПИСОК РЕКОМЕНДУЕМОЙ ЛИТЕРАТУРЫ 187](#_Toc83804659)

[Приложение 1 188](#_Toc83804660)

# 

**Указания по технике безопасности**

Перед началом занятия необходимо:

1. Проверить правильность установки стола, стула, угол наклона экрана монитора, положение клавиатуры и, при необходимости, произвести корректировку их установки в целях исключения неудобных поз и длительных напряжений тела. Особо обратить внимание на то, что дисплей должен находиться на расстоянии не менее 50 см от глаз (оптимально 60 – 70 см), плоскость его экрана должна быть перпендикулярна направлению взгляда и центр экрана должен быть ниже уровня (или на уровне) глаз.
2. Подготовить к работе рабочее место, разместив необходимые принадлежности таким образом, чтобы исключить их падение и опрокидывание.
3. Проверить работу компьютера, обо всех замеченных нарушениях, неисправностях и поломках немедленно доложить преподавателю или лаборанту.

Запрещается приступать к работе в случае обнаружения несоответствия рабочего места установленным в данном разделе требованиям, а также при невозможности выполнить указанные в данном разделе подготовительные к работе действия.

Обо всех неполадках в работе оборудования необходимо ставить в известность преподавателя или лаборанта. Запрещается самостоятельное устранение любых неисправностей используемого оборудования.

При обнаружении неисправности в работе оборудования (нагревании, появлении искрения, запаха горелой изоляции, появлении посторонних звуков и т.п.) немедленно прекратить работу и сообщить об этом учителю (иному лицу, проводящему занятия) или лаборанту и действовать в соответствии с его указаниями.

По окончании занятия следует:

1. Произвести закрытие всех выполняемых задач.
2. Привести в порядок рабочее место.
3. При обнаружении неисправности мебели, оборудования проинформировать об этом преподавателя и лаборанта.
4. С разрешения преподавателя организованно покинуть кабинет.

# Лабораторная работа №1

# АРХИТЕКТУРА РАСПРЕДЕЛЕННЫХ INTERNET ПРИЛОЖЕНИЙ. УСТАНОВКА ПАКЕТА DENWER

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

**1. Теоретическая часть**

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 1.1):

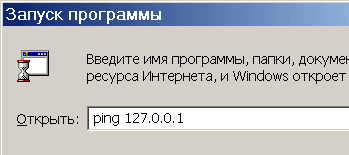


Рисунок 1.1 – Пинг локального сервера

После нажатия Enter вы должны увидеть примерно следующую картину (рисунок.1.2):

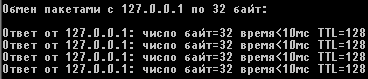


Рисунок 1.2 – Результаты команды «пинг» локального сервера

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 1.3):

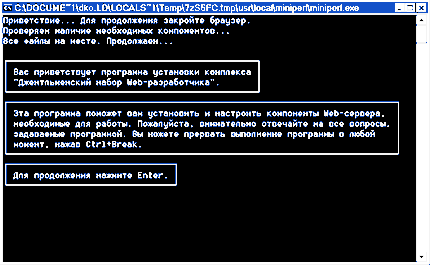


Рисунок 1.3 – Установка Денвера

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 1.4).

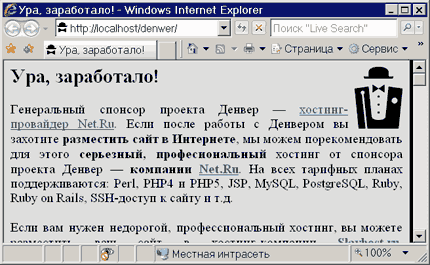


Рисунок 1.4 – Стартовая страница Денвера

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 «DNS-клиент». 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 1.5):

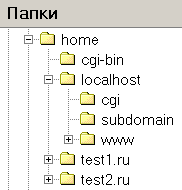


Рисунок 1.5 – Структура папок Девера

* 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 1.6 или 1.7):

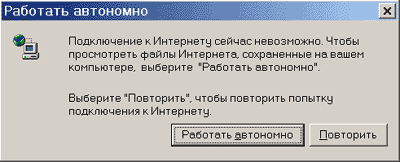


Рисунок 1.6 – Пинг локального сервера

или так:

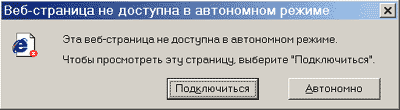


Рисунок 1.7 – Пинг локального сервера

Всегда выбирайте **Подключиться** или **Повторить**.

Ни в коем случае не давайте ему ответ **Автономно.**

Но если ваш **Контроллер удаленного доступа** не реагирует и на ответ **Подключиться** 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 по-другому, 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**.

**2. Материалы и оборудование**

1. Компьютер.
2. Программа MS Visual Studio.

**3. Задания к лабораторной работе**

1. Изучить теоретическую часть к лабораторной работе, изложенную в методических указаниях.

2. Ответить на контрольные вопросы к лабораторной работе, использовать лекции и указанную в них литературу, теоретический материал к лабораторной работе.

3.Выполнить задание**,** согласно своему варианту, изложенное в методических указаниях.

Варианты заданий к лабораторной работе №1:

1.1. Изучить процесс установки пакета Денвер, описанный в разделе «1. Теоретическая часть».

1.2. Установить пакет Денвер.

4. Оформить отчет по лабораторной работе (Приложение 1).

**4. Вопросы к лабораторной работе**

1.1. Какие утилиты входят в базовый пакет Денвер?

1.2. Дайте характеристику понятию «виртуальный хост».

**5. Задания для самостоятельной работы**

1.1. Проанализируйте состав и возможности пакета **Денвер.**

1.2. Изучите особенности установки на базе различных операционных систем.

# Лабораторная работа №2. ОСНОВНАЯ ИНФОРМАЦИЯ О MYSQL. СОЗДАНИЕ БАЗЫ ДАННЫХ

**Цель работы:** научиться создавать и администрировать базу данных mysql.

**1. Теоретическая часть**

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).

Cтруктура 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.

**Поля и их типы в MySQL**

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 2.1- 2.5.

Таблица 2.1- 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 2.2- 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 2.3 - 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 2.4- 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 2.5- 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.

**2. Материалы и оборудование**

1. Компьютер.
2. Программа Denver.

**3. Задания к лабораторной работе**

1. Изучить теоретическую часть к лабораторной работе, изложенную в методических указаниях.

2. Ответить на контрольные вопросы к лабораторной работе, использовать лекции и указанную в них литературу, теоретический материал к лабораторной работе. Варианты заданий к лабораторной работе № 2:

2.1. Запустить Denver

2.2. В браузере перейти по ссылке <http://localhost/Tools/phpMyAdmin/>

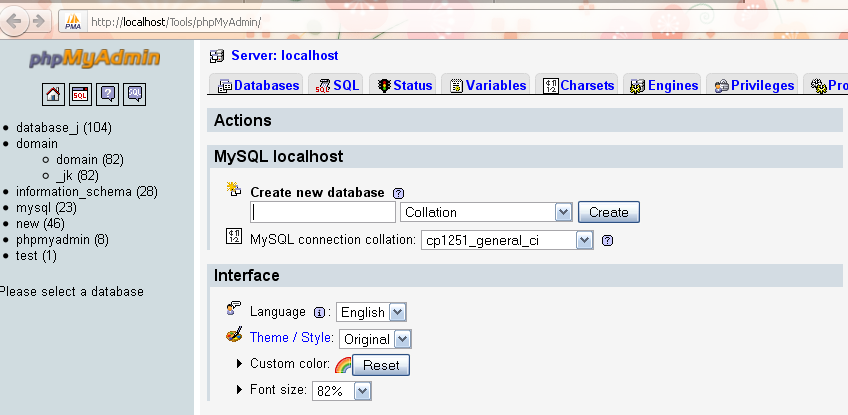


Рисунок 2.1- Окно браузера

2.3. В поле Create new database (рисунок 2.1) ввести имя создаваемой базы данных, в списке MySqlconnection collation выбираем utf8\_general\_ci, нажать кнопку create (рисунок 2.2).

2.4. Изучите поля и типы данных MySQL.

2.5. Создайте нового пользователя базы данных, обладающего всеми привилегиями и установить для него пароль. Воспользуйтесь вкладкой privileges.

4. Оформить отчет по лабораторной работе (Приложение 1).

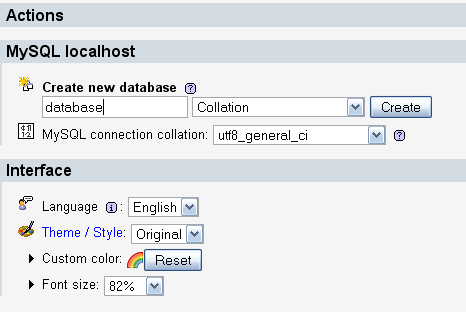


Рисунок 2.2 – Создание БД

**4. Вопросы к лабораторной работе**

2.1. Что такое привилегии?

2.2.Какие привилегии Вы выбрали при создании БД?

2.3.Какие типы данных поддерживает MySQL?

2.4.Каким образом можно представить дату средствами MySQL?

2.5. Операторы SQL для предоставления и отмены привилегий.

2.6. Для чего в MySQL используются бинарные типы данных?

**5. Задания для самостоятельной работы**

2.1. Проанализируйте состав и возможности пакета **Денвер.**

2.2. Создайте пользователя, обладающего ограниченными правами, и задайте ему пароль.

2.3. Изучите особенности представления даты и времени.

# Лабораторная работа №3. ОПЕРАТОРЫ И КОМАНДЫ MySQL

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

**1. Теоретическая часть**

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 по-разному. 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 3.1):

Таблица 3.1- 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:

UPDATE CLIENTS

SET CITY = 'Псков'

WHERE C\_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 3.2. Таблица 3.2 – 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;

SELECT SUM(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;

В этом запросе мы использовали псевдоним столбца TOTALSUM.

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

ADD PRIMARY KEY (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 СHECK (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=19. Как мы потом сможем идентифицировать клиента? Для решения такой проблемы и существуют внешние ключи:

CREATE TABLE

(

/\* Описание полей таблицы \*/

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. Определение внешнего ключа:

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 будут 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().

**2. Материалы и оборудование**

1. Компьютер.
2. Программа Денвер.

**3. Задания к лабораторной работе**

1. Изучить теоретическую часть к лабораторной работе, изложенную в методических указаниях.

2. Ответить на контрольные вопросы к лабораторной работе, использовать лекции и указанную в них литературу, теоретический материал к лабораторной работе.

3.Выполнить задание**,** согласно своему варианту, изложенное в методических указаниях. Варианты заданий к лабораторной работе № 3:

3.1. Создайте базу данных, состоящую из трех таблиц (CLIENTS, ORDERS, TOOLS), заполните ее данными, используя язык запросов SQL (рисунки 3.1 и 3.2). Установите для таблиц ключевые поля, запрет на ввод нулевых (неопределенных) значений. Ввести запрос на вкладке SQL и нажать кнопку Go

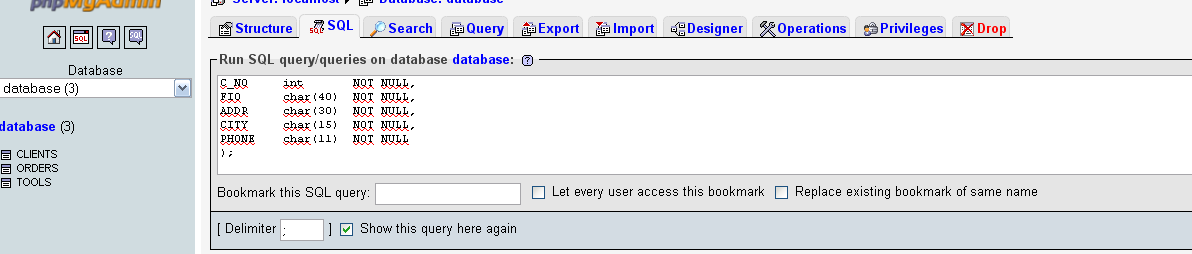


Рисунок 3.1 – Создание БД

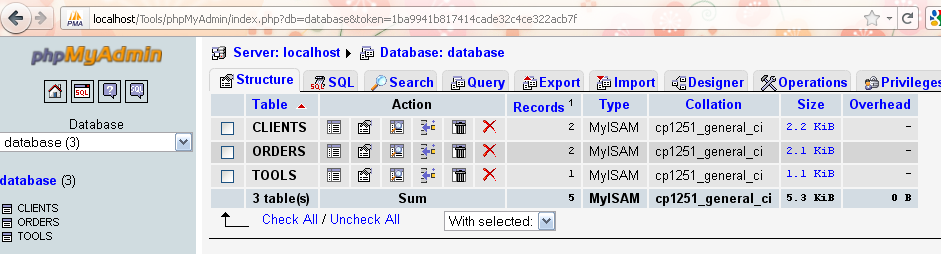


Рисунок 3.2 – Работа с БД

3.2. Проверьте целостность созданной базы данных.

4. Оформить отчет по лабораторной работе (Приложение 1).

**4. Вопросы к лабораторной работе**

3.1. Какие атрибуты могут быть первичными ключами?

3.2.Каким образом можно запретить ввод неопределенных значений?

3.3. Для чего используется оператор **CASCADE**? Приведите примеры.

**5. Задания для самостоятельной работы**

3.1.Изучите особенности создания ключевых полей.

3.2. Изучите средства СУБД для поддержания целостности.

3.3. Приведите примеры использования операторов **CASCADE** и **RESTRICT**.

# Лабораторная работа № 4. SQL ЗАПРОСЫ. БЕЗУСЛОВНАЯ ВЫБОРКА

**Цель работы:** научиться использовать SQL запросы для работы с базой данных MySQL.

На лабораторной работе реализуется интерактивная форма обучения – компьютерная симуляция.

**1. Теоретическая часть**

Команды sql для удобства работы разделяются на следующие группы: Команды определения данных (Data Definition Commands); Команды манипуляции данными (Data Мanipulation Commands); Команда выборки данных (Data Query Commands); Команды управления транзакциями (Transaction Control Commands); Команды управления данными (Data Control Commands). Приведем список команд языка sql для каждой из этих групп в таблицах 4.1 - 4.6.

Таблица 4.1 - Команды определения данных (DDL)

|  |  |
| --- | --- |
| **Команда** | **Назначение** |
| ALTER TABLE | Изменяет структуру таблицы |
| CREATE INDEX | Создает индекс |
| CREATE TABLE | Создает таблицу |
| CREATE VIEW | Создает представление |
| DROP | Удаляет таблицу, индекс, представление |

Таблица 4.2 - Команды манипуляции данными- DML

|  |  |
| --- | --- |
| **Команда** | **Назначение** |
| DELETE | Удаляет записи таблицы |
| INSERT | Добавляет записи в таблицу |
| UPDATE | Изменяет данные таблицы |

Таблица 4.3 - Команда выборки данных

|  |  |
| --- | --- |
| **Команда** | **Назначение** |
| SELECT | Выбирает данные из таблиц |

Таблица 4.4 - Команды управления транзакциями

|  |  |
| --- | --- |
| **Команда** | **Назначение** |
| СOMMIT | Делает изменения, проведенные с начала транзакции,  постоянными |
| ROLLBACK | Откатывает все проведенные изменения к точке  сохранения или к началу транзакции |
| SAVEPOINT | Устанавливает контрольную точку, к которой  впоследствии можно будет выполнить откат |

Таблица 4.5 - Команды управления данными

|  |  |
| --- | --- |
| **Команда** | **Назначение** |
| CHECK DATABASE | Проверяет целостность базы данных |
| GRANT | Предоставляет привилегии |
| REVOKE | Удаляет предоставленные ранее привилегии |
| LOCK/UNLOCK | Блокирует данные и отменяет блокировку |

Таблица 4.6 - Программный SQL

|  |  |
| --- | --- |
| **Команда** | **Назначение** |
| declare | Определяет курсор для запроса, задает имя курсора, который соответствует виртуальному набору данных. |
| open | Открыть курсор. Формирует виртуальный набор данных, соответствующий описанию курсора и текущему состоянию БД. |
| fetch | Считать очередную строку, заданную параметром команды из набора данных, соответствующего открытому  курсору. |
| close | Закрыть курсор |
| prepare | Подготовить оператор SQL к динамическому выполнению |
| execute | Выполнить оператор, ранее подготовленный к динамическому выполнению |

Курсоры

Если запрос к таблице возвращает несколько строк (больше одной), то для их обработки используется курсор – указатель во множестве строк, выбранных оператором SELECT. Курсор – средство языка SQL, позволяющее получать построчный доступ к результату запроса. Термин «курсор» означает «текущий набор записей - **cur**rent **s**et **o**f **r**ecords». Оператором DECLARE в динамическом SQL объявляется курсор для запроса, оператором OPEN этот запрос выполняется и выбранные строки выделяются. Оператор FETCH позволяет перемещаться от одной строки результирующего множества к другой.

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 ALLподразумевается 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=R1х R2.

Запрос

SELECT R1.А, R2.В FROM R1, R2

соответствует проекции декартова произведения таблиц R1 и R2 на столбцы A из таблицы R1 и B – из таблицы R2, то есть операции проекция реляционной алгебры.

|  |  |
| --- | --- |
|  |  |

**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.

**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

**2. Материалы и оборудование**

1. Компьютер.

2. Программа Денвер.

**3. Задания к лабораторной работе**

1. Изучить теоретическую часть к лабораторной работе, изложенную в методических указаниях.

2. Ответить на контрольные вопросы к лабораторной работе, использовать лекции и указанную в них литературу, теоретический материал к лабораторной работе.

3.Выполнить задание**,** согласно своему варианту, изложенное в методических указаниях. Варианты заданий к лабораторной работе №4:

4.1. Создать таблицу R1, содержащую не менее 5 столбцов. Составить запрос для выборки всех столбцов таблицы R1 в том же порядке, в котором они находятся в исходной таблице.

4.2. Составить запрос для выборки всех столбцов таблицы R1(А,В,С,D,G) таким образом, чтобы порядок столбцов в выборке изменился следующим образом: D,G, В,С,А.

4.3. Составить запрос для выборки уникальных значений G из таблицы R1(А,В,С,D,G).

4.4. Составить запрос для выборки всех значений G из таблицы R1(А,В,С,D,G).

4. Оформить отчет по лабораторной работе (Приложение 1).

**4. Вопросы к лабораторной работе**

4.1. Как исключить дубликаты строк из результата выборки?

4.2. Как получить Декартово произведение отношений?

**5. Задания для самостоятельной работы**

4.1.Изучите инструментальные средства СУБД MySQL для получения безусловной выборки данных.

# Лабораторная работа № 5. РАСШИРЕННЫЕ SQL-ЗАПРОСЫ. ВЫБОРКА С УСЛОВИЕМ

**Цель работы:** научиться использовать расширенные SQL-запросы для работы с базой данных MySQL.

**1. Теоретическая часть**

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.** 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, можно представить в виде:

**SELECT kod, NAME**

**FROM Table3**

**WHERE CITY=“Москва”**

**AND CREDITLIMIT>200000**

**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)**. Операторы   
**«>(Больше)»** и **«<(Меньше)»** используются в том случае, если задана только нижняя или верхняя граница диапазона, a оператор **Between** **(Между)** – если известны обе границы диапазона.

С помощью SQL-запроса

**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**

**Выборка с использованием оператора IN (принадлежит)**

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)**

**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 cодержит слово “Тoshiba”( таблица 5.1).

Таблица 5.1 – результат выборки по шаблону

|  |  |  |
| --- | --- | --- |
| **kod** | NAME | UNITPRICE |
| 2 | HDD Toshiba 1Tb | 3000.00 |
| 4 | R14 Toshiba 2/Shrink | 557.00 |

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 SQL-запрос, реализующий данное условие примет вид:

**SELECT \* FROM TAB7 WHERE string1 LIKE «%+%S\_**

**2. Материалы и оборудование**

1. Компьютер.
2. Программа Денвер.

**3. Задания к лабораторной работе**

1. Изучить теоретическую часть к лабораторной работе, изложенную в методических указаниях.

2. Ответить на контрольные вопросы к лабораторной работе, использовать лекции и указанную в них литературу, теоретический материал к лабораторной работе.

3.Выполнить задание, согласно своему варианту, изложенное в методических указаниях. Варианты заданий к лабораторной работе № 5:

5.1.Создайте таблицы: «Склад»; «Потребители»; «Журнал товаров»; «Журнал прихода»; «Журнал реализации».

*Таблица «Склад»* характеризуется следующими атрибутами:

* уникальный номер склада;
* название склада;
* адрес склада;
* реквизиты склада (инн, счет в банке);
* телефон склада.

*Таблица «Потребители»* характеризуется атрибутами:

* уникальный код потребителя;
* название потребителя;
* адрес потребителя;
* реквизиты потребителя (инн, счет в банке);
* телефон потребителя;
* факс потребителя.

*Таблица «Журнал товаров»*  описывается с помощью атрибутов

* уникальный ключ товара;
* наименование товара;
* цена 1 товара;
* страна производитель.

*Таблицу «Журнал реализации»* описывают атрибуты:

* уникальный ключ операции;
* уникальный код потребителя;
* уникальный ключ товара;
* номер накладной на реализацию;
* дата реализации товара;
* количество реализованного товара;
* общая стоимость реализованного товара.

*Таблицу «Журнал прихода»* можно описать атрибутами:

* уникальный ключ операции;
* уникальный код поставщика;
* уникальный ключ товара;
* накладной на получение товара.
* дата получения товара;
* количество полученного товара;
* общая стоимость полученного товара.

5.2. Создайте запросы для выборки по шаблону из таблиц «Склад», «Потребители», «Журнал товаров», «Журнал прихода», «Журнал реализации».

5.3. Создайте запросы для выборки по условию с применением операторов IN (принадлежит) и BETWEEN из таблиц «Склад», «Потребители», «Журнал товаров», «Журнал прихода», «Журнал реализации».

4. Оформить отчет по лабораторной работе (Приложение 1).

**4. Вопросы к лабораторной работе**

5.1.Какие логические операторы может содержать SQL-запрос?

5.2. Можно ли по отношению к переменным типа **VARCHAR** или просто **CHAR** применять операторы **«>(Больше)»** и **«<(Меньше)»**?

**5. Задания для самостоятельной работы**

5.1. Изучите инструментальные средства СУБД MySQL для получения выборки данных.

1. **Лабораторная работа № 6. РАСШИРЕННЫЕ SQL-ЗАПРОСЫ. ВЫБОРКА ВЫЧИСЛЯЕМЫХ ЗНАЧЕНИЙ**

**Цель работы:** научиться использовать расширенные SQL-запросы для получения вычисляемых значений.

На лабораторной работе реализуется интерактивная форма обучения – компьютерная симуляция.

**1. Теоретическая часть**

Результирующая таблица может содержать не только поля исходных таблиц, но и результат вычислений над полями таблицы. В выражениях могут быть использованы арифметические операторы и функции. Перечень допустимых функций определяется используемым диалектом языка SQL.

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 6.1. Тип переменной UNITPRICE – Currency.

Таблица 6.1 – Поля таблицы Товары

|  |  |
| --- | --- |
| Kod1 | UNITPRICE |
| 1 | 1000.00 |
| 2 | 2000.00 |
| 3 | 10000.00 |

Необходимо организовать выборку информации из таблицы Товары,указав цены с учетом налога на добавленную стоимость (20%).

SQL-запрос, соответствующий примеру:

**SELECT Kod1, UNITPRICE \* (1 + 0,2) AS ЦЕНА FROM Товары**

Результат выполнения данного запроса представлен в таблице 6.2.

Таблица 6.2 – Результат выборки из таблицы Товары

|  |  |
| --- | --- |
| Kod1 | ЦЕНА |
| 1 | 1200.00 |
| 2 | 2400.00 |
| 3 | 12000.00 |

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, добавить строку **«**Цена указана с учетом НДС», то он примет вид:

**SELECT Kod1, UNITPRICE \* (1 + 0,2) AS ЦЕНА, «Цена указана с учетом НДС» FROM Товары**

Результат выполнения данного запроса представлен в таблице 6.3 .

Таблица 6.3 – Результат выполнения запроса

|  |  |  |
| --- | --- | --- |
| Kod1 | ЦЕНА | Цена указана с учетом НДС |
| 1 | 1200.00 | Цена указана с учетом НДС |
| 2 | 2400.00 | Цена указана с учетом НДС |
| 3 | 12000.00 | Цена указана с учетом НДС |

При выборке с помощью команды **SELECT** можно использовать статистические (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.

**2. Материалы и оборудование**

1. Компьютер.
2. Программа Денвер.

**3. Задания к лабораторной работе**

1. Изучить теоретическую часть к лабораторной работе, изложенную в методических указаниях.

2. Ответить на контрольные вопросы к лабораторной работе, использовать лекции и указанную в них литературу, теоретический материал к лабораторной работе.

3.Выполнить задание**,** согласно своему варианту, изложенное в методических указаниях. Варианты заданий к лабораторной работе №6:

6.1.Составить запрос для выборки цены товара (Таблица «Журнал товаров» из лабораторной работы №5) с кодом Kod1=2 со скидкой 5%.

6.2. Составить запрос для выборки цены товара с кодом Kod1=3 со скидкой 15%.

6.3. Составить запрос для выборки цены товара с кодом Kod1=1 со скидкой 15%.

6.4. Составить запрос для выборки цены товара (с кодом Kod1=1с учетом налога на добавленную стоимость (20%).

6.5. Составить запрос для выборки максимальной цены товара.

6.6. Составить запрос для выборки средней цены товара.

6.7. Составить запрос для выборки суммы всех товаров.

6.8. Составить запрос для выборки минимальной цены товара.

4. Оформить отчет по лабораторной работе (Приложение 1).

**4. Вопросы к лабораторной работе**

6.1. Как определить количество строк в таблице?

6.2. Какой запрос возвращает максимальное значение числового поля?

6.3. Как переименовать столбец в выборке?

6.4. Как добавить константу в результат выборки?

6.5. К каким типам переменных можно применять функцию COUNT(\*)?

6.6. К каким типам переменных можно применять функцию SUM(X)?

6.7. К каким типам переменных можно применять функцию AVG(X)?

6.8. К каким типам переменных можно применять функцию MAX(X)?

6.9. К каким типам переменных можно применять функцию MIN(X)?

**5. Задания для самостоятельной работы**

6.1. Изучите инструментальные средства СУБД MySQL для получения вычисляемых значений.

# Лабораторная работа № 7. РАСШИРЕННЫЕ SQL-ЗАПРОСЫ. ВЫБОРКА ИЗ СВЯЗАННЫХ ТАБЛИЦ

**Цель работы:** изучение синтаксиса расширенных SQL-запросов, которые предназначены для выборки из связанных таблиц

**1. Теоретическая часть**

Способность «соединять» две или более таблицы в одну представляет собой одну из наиболее мощных возможностей реляционных баз данных.

**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 7.1.

Таблица 7.1 – Результат выборки из двух таблиц

|  |  |
| --- | --- |
| **Kod2** | **NAME** |
| 2 | BORLAND C++ |
| 10 | NoSQL СУБД |
| 18 | Книга TURBO C++ |
| 20 | Книга TURBO 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 Macintosh, служит SQL- запрос:

**SELECT ORDSALE.CUSTOMERNO, ORDSALE.FIRSTNAME**

**FROM ORDSALE, GOODS**

**WHERE ORDSALE.STOCK=GOODS.STOCK**

**and GOODS.NAME=“** Macintosh**”**

Результат выполнения данного запроса представлен в таблице 7.2.

Таблица 7.2 – 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 Macintosh.

**SELECT DISTINCT CUSTOMER.FIRSTNAME,**

**CUSTOMER.LASTNAME**

**FROM CUSTOMER, ORDSALE, GOODS**

**WHERE CUSTOMER.CUSTOMERNO=ORDSALE.CUSTOMERNO**

**and ORDSALE.STOCK=GOODS.STOCK**

**and GOODS.NAME=“Macintosh”**

После выполнения данного запроса на экран будет выведена таблица 7.3.

Таблица 7.3 – Результат выборки из трех таблиц

|  |  |
| --- | --- |
| CUSTOMER.FIRSTNAME | CUSTOMER.LASTNAME |
| Иванов | Андрей |
| Петрова | Дарья |
| Сидоров | Максим |

Во многих случаях выборки из нескольких таблиц используются не только для ограничения выборки, но и для объединения данных из нескольких таблиц.

**2. Материалы и оборудование**

1. Компьютер.
2. Программа Денвер.

**3. Задания к лабораторной работе**

1. Изучить теоретическую часть к лабораторной работе, изложенную в методических указаниях.

2. Ответить на контрольные вопросы к лабораторной работе, использовать лекции и указанную в них литературу, теоретический материал к лабораторной работе.

3.Выполнить задание**,** согласно своему варианту, изложенное в методических указаниях. Варианты заданий к лабораторной работе №7:

7.1. В состав реляционной БД входят отношения   
*Журнал товаров (*уникальный ключ товара, наименование товара, цена 1 товара, страна производитель) и

*Журнал прихода(* уникальный ключ операции, уникальный код поставщика, уникальный ключ товара, накладной на получение товара, дата получения товара, количество полученного товара, общая стоимость полученного товара).

Составить SQL-запрос для получения таблицы, содержащей без дублирования все атрибуты отношений *Журнал товаров* и *Журнал прихода*.

7.2. Составить SQL-запрос для получения таблицы, содержащей атрибуты: наименование товара, цена 1 товара, страна производитель, дата получения товара, количество полученного товара, общая стоимость полученного товара.

4. Оформить отчет по лабораторной работе (Приложение 1).

**4. Вопросы к лабораторной работе**

7.1. Какая операция определяет новое отношение как множество кортежей отношения Р или отношения В?

7.2. Какая операция определяет новое отношение как множество кортежей отношения Р, которые являются также кортежами отношения В?

**5. Задания для самостоятельной работы**

7.1. Составьте таблицы, соответствующие экзаменационным ведомостям по трем предметам. Составьте SQL-запрос для получения сводной ведомости и реализуйте с помощью СУБД MySQL.

# Лабораторная работа № 8. ИСПОЛЬЗОВАНИЕ ГРУППИРОВКИ ДАННЫХ ПРИ ОРГАНИЗАЦИИ SQL- ЗАПРОСОВ

**Цель работы:** изучение группировки данных при организации SQL-запросов.

**1. Теоретическая часть**

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.

**2. Материалы и оборудование**

1. Компьютер.
2. Программа Денвер.

**3. Задания к лабораторной работе**

1. Изучить теоретическую часть к лабораторной работе, изложенную в методических указаниях.

2. Ответить на контрольные вопросы к лабораторной работе, использовать лекции и указанную в них литературу, теоретический материал к лабораторной работе.

3.Выполнить задание, согласно своему варианту, изложенное в методических указаниях. Варианты заданий к лабораторной работе №8:

8.1. Используйте отношение *«Журнал реализации»* с атрибутами:

* уникальный ключ операции;
* уникальный код потребителя;
* уникальный ключ товара;
* номер накладной на реализацию;
* дата реализации товара;
* количество реализованного товара;
* общая стоимость реализованного товара.

Составьте запрос для получения таблицы, в которую включены коды товаров, приобретенных более чем одним покупателем.

8.2. Составьте запрос с группировкой для отношения *«Журнал реализации».*

8.3. Составьте запрос с группировкой, чтобы вычислить общий объем покупок для каждого товара.

4. Оформить отчет по лабораторной работе (Приложение 1).

**4. Вопросы к лабораторной работе**

8.1.Для чего используется оператор **HAVING?**

8.2. Сколько значений может принимать агрегатная функция **COUNT** для группы?

8.3. Сколько значений может принимать агрегатная функция **SUM** для группы?

8.4. Сколько значений может принимать агрегатная функция **AVG** для группы?

**5. Задания для самостоятельной работы**

8.1. Составьте SQL-запрос c группировкой и реализуйте с помощью СУБД MySQL.

# Лабораторная работа № 9. CОРТИРОВКА

**Цель работы:** изучение синтаксиса расширенных SQL-запросов, которые предназначены для сортировки данных.

**1. Теоретическая часть**

Порядок строк в реляционных базах данных безразличен.Однако подразумевается возможность сортировки результатов выборки по одному или нескольким произвольно выбранным полям. Для каждого выбранного поля можно 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.** Для таблицы **Table3** необходимо произвести выбор кода (**kod**), имени (**NAME)**  и суммы кредита (**CREDITLIMIT)** всех покупателей, проживающих в Ставрополе, расположив их в порядке убывания.

SQL-запрос 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 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 \* 1.2**

**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.

**2. Материалы и оборудование**

1. Компьютер.
2. Программа Денвер.

**3. Задания к лабораторной работе**

1. Изучить теоретическую часть к лабораторной работе, изложенную в методических указаниях.

2. Ответить на контрольные вопросы к лабораторной работе, использовать лекции и указанную в них литературу, теоретический материал к лабораторной работе.

3.Выполнить задание**,** согласно своему варианту, изложенное в методических указаниях. Варианты заданий к лабораторной работе № 9:

9.1. Для любых трех таблиц, описанных в лабораторной работе №5, составить SQL-запросы с сортировкой по возрастанию и по убыванию. Выполнить сортировку по одному и по двум полям.

9.2. Для отношения «*Журнал товаров»* составитьSQL-запрос для получения цены товаров со скидкой 7%. Выборку упорядочить по возрастанию по вычисленному полю.

9.3. Для отношения «*Журнал товаров»* составитьSQL-запрос для получения цены товаров с учетом НДС т.е. с наценкой 20%. Выборку упорядочить по убыванию по вычисленному полю.

4. Оформить отчет по лабораторной работе (Приложение 1).

**4. Вопросы к лабораторной работе**

9.1. В каком порядке выводятся строки результирующей таблицы без указания критерия упорядочивания данных?

9.2. Можно ли осуществить сортировку по вычисляемым значениям?

9.3. Каким образом осуществляется сортировка по алфавиту?

**5. Задания для самостоятельной работы**

9.1**.** Составьте SQL-запрос c сортировкой и реализуйте его с помощью СУБД MySQL.

# Лабораторная работа № 10

# РАСШИРЕННЫЕ SQL-ЗАПРОСЫ. ИСПОЛЬЗОВАНИЕ КВАНТОРА СУЩЕСТВОВАНИЯ В ЗАПРОСАХ. КОРРЕЛИРОВАННЫЕ И НЕКОРРЕЛИРОВАННЫЕ ПОДЗАПРОСЫ

**Цель работы:** изучение синтаксиса расширенных SQL-запросов, содержащих операторы EXISTS и NOT EXISTS.

**1. Теоретическая часть**

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.

**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.

**2. Материалы и оборудование**

1. Компьютер.
2. Программа Денвер.

**3. Задания к лабораторной работе**

1. Изучить теоретическую часть к лабораторной работе, изложенную в методических указаниях.

2. Ответить на контрольные вопросы к лабораторной работе, использовать лекции и указанную в них литературу, теоретический материал к лабораторной работе.

3.Выполнить задание**,** согласно своему варианту, изложенное в методических указаниях. Варианты заданий к лабораторной работе №10:

10.1.В состав реляционной БД входят отношения

CUSTOMER (CUSTOMERNO, FIRSTNAME, LASTNAME, CITY, CREDITLIMIT) и ORDSALE (STOCK, CUSTOMERNO, NAME). Составить SQL-запрос с помощью оператора EXISTS для выборки фамилий покупателей, которым был продан товар с наименованием (NAME), содержащим символы “HDD”.10.2.В состав реляционной БД входят отношения

CUSTOMER (CUSTOMERNO, FIRSTNAME, LASTNAME, CITY, CREDITLIMIT) и ORDSALE (STOCK, CUSTOMERNO, NAME). Составить SQL-запрос с помощью оператора EXISTS для выборки фамилий покупателей, которым не был продан товар с наименованием (NAME), содержащим символы “HDD”.10.3.В состав реляционной БД входит отношение  
CUSTOMER(CUSTOMERNO, FIRSTNAME, LASTNAME, CITY, CREDITLIMIT). Составить SQL-запрос для выборки проживающих в Москве покупателей, имеющих кредит (CREDITLIMIT) больше среднего с указанием кода покупателя (CUSTOMERNO), фамилии и кредита (выборку упорядочить в порядке убывания кредита).

10.4. В состав реляционной БД входит отношение  
CUSTOMER(CUSTOMERNO, FIRSTNAME, LASTNAME, CITY, CREDITLIMIT). Составить SQL-запрос для выборки проживающих в Москве покупателей, имеющих кредит (CREDITLIMIT) меньше среднего с указанием кода покупателя (CUSTOMERNO), фамилии и кредита (выборку упорядочить в порядке убывания кредита).

4. Оформить отчет по лабораторной работе (Приложение 1).

**4. Вопросы к лабораторной работе**

10.1. Какой подзапрос называют коррелированным?

10.2. Какой подзапрос называют некоррелированным?

10.3. Каким образом без оператора **EXISTS** получить такой же результат, как при выполнении запроса:

SELECT NAME FROM tab10 WHERE EXISTS (SELECT \* FROM tab11 WHERE tab10.kod= tab11.kod and.STOCK = “Macintosh”)?

**5. Задания для самостоятельной работы**

10.1. Для связанных отношений *«Журнал товаров» и «Журнал реализации»* составить запросы для выборки наименования товара, приобретенного потребителем, у которого уникальный код потребителя равен 11. Один из вариантов запроса составить с использованием квантора, а другой вариант запроса без использования квантора. Результаты выборки, полученные разными способами должны быть одинаковыми.

*Атрибуты отношения «Журнал товаров»:*

- уникальный ключ товара;

* наименование товара;
* цена 1 товара;
* страна производитель.

*Атрибуты отношения «Журнал реализации»*:

* уникальный ключ операции;
* уникальный код потребителя;
* уникальный ключ товара;
* номер накладной на реализацию;
* дата реализации товара;
* количество реализованного товара;

- общая стоимость реализованного товара.

# Лабораторная работа № 11

**РАСШИРЕННЫЕ SQL-ЗАПРОСЫ. МОДИФИКАЦИЯ, УДАЛЕНИЕ И   
ДОБАВЛЕНИЕ ДАННЫХ**

**Цель работы:** изучение синтаксиса расширенных SQL-запросов, содержащих операторы для модификации, удаления и добавления данных.

**1. Теоретическая часть**

## 

## **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 =“Macintosh4”,**

**UNITPRICE = UNITPRICE+10000**

**WHERE NAME =“Macintosh”**

будет изменено название товара «Macintosh» на «Macintosh4» и увеличена стоимость на 10000.

**Модификация множества записей.** При изменении значений во множестве записей таблицы условие, определяемой при помощи оператора **WHERE,** должно удовлетворятьвсему множеству. Используя **SQL**-запрос

**UPDATE tab14**

**SET CREDITLIMIT= CREDITLIMIT\*2**

**WHERE CITY=“Ставрополь”**

можно получить список всех покупателей, проживающих в Ставрополе, с увеличенной в 2 раза суммой кредита.

**Модификация с подзапросом.** Модификация с подзапросом используется в том случае, если необходимо произвести изменить значения 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%.

**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** – **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 – SQL**-запрос, иллюстрирующий удаление из базы множества записей

**DELETE FROM tab12 WHERE STOCK=34**

После выполнения данного **SQL**-запроса из таблицы **tab12** будутудаленывсе записи, в которых поле **STOCK** равно 34.

Пример – **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 – 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.

**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  **– SQL-**запрос для вставки одной записи в базу данных

**INSERT INTO tab15**

**(kod, NAME, UNITPRICE, CATEGORY)**

**VALUES**

**(1001,”Macintosh”, 30000.00, 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 «Macintosh»,с кодом равным 1001, стоимостью 30000000 и категорией, равной 2.

Пример  **– SQL-**запрос для вставки одной записи в базу данных

**INSERT INTO tab15 VALUES (1001,”Macintosh”, 30000.00, 2)**

С помощью данного **SQL**-запроса в таблицу **tab15** также добавляется новая строка, содержащая информацию о товаре «Macintosh».

Отсутствие полей эквивалентно перечислению списка всех полей таблицы в порядке слева направо так, как они были определены при создании таблицы.

**2. Материалы и оборудование**

1. Компьютер.
2. Программа Денвер.

**3. Задания к лабораторной работе**

1. Изучить теоретическую часть к лабораторной работе, изложенную в методических указаниях.

2. Ответить на контрольные вопросы к лабораторной работе, использовать лекции и указанную в них литературу, теоретический материал к лабораторной работе.

3.Выполнить задание**,** согласно своему варианту, изложенное в методических указаниях.

Для отношения*«Журнал реализации» выполнить удаление, добавление и модификацию записей. Атрибуты отношения*:

* уникальный ключ операции;
* уникальный код потребителя;
* уникальный ключ товара;
* номер накладной на реализацию;
* дата реализации товара;
* количество реализованного товара;

- общая стоимость реализованного товара.

4. Оформить отчет по лабораторной работе (Приложение 1).

**4. Вопросы к лабораторной работе**

11.1. Каким образом осуществить модификацию записей с помощью SQL?

11.2. Каким образом осуществить добавление записей с помощью SQL?

11.3. Каким образом осуществить удаление записей с помощью SQL?

**5. Задания для самостоятельной работы**

11.1. Добавьте записи в отношение *«Журнал реализации»* таким образом, чтобы дата реализации товара была равна текущей дате.

# Лабораторная работа № 12

# ОПРЕДЕЛЕНИЕ ОТНОШЕНИЙ МЕЖДУ ТАБЛИЦАМИ. ПРОВЕРКА УСЛОВИЙ ЦЕЛОСТНОСТИ ДАННЫХ

**Цель работы:** познакомить студентов с основными типами отношений между таблицами в реляционных базах данных; научить определять отношения между сформированными таблицами и осуществлять проверку условий целостности данных в базе.

**1. Теоретическая часть**

Cовременные СУБДподдерживают четыре типа отношений между таблицами: **1:1; 1:М; М:1; M:N**.

Отношения **M:N** возникают между двумя таблицами в том случае, если:

1) одна запись из первой таблицы может быть связана более чем с одной записью из второй таблицы;

2) одна запись из второй таблицы может быть связана более чем с одной записью первой из таблицы.

Отношение **1:1** возникает между двумя таблицами в базе данных, если каждая запись одной таблицы соответствует только одной записи в другой таблице.

Отношения **1:M** или **M:1** возникают между двумя таблицами, если одна запись в одной из таблиц базы данных соответствует более одной записи в другой таблице.

С помощью СУБД можно устанавливать постоянные логические отношения между таблицами. Логическая связь таблиц является иерархической. При определении отношений одна таблица является родительской, а другая дочерней. Для поддержки связей обе таблицы должны содержать одинаковые наборы атрибутов, по которым они связаны. В родительской таблице это первичный ключ (**PRIMARY KEY**), а в дочерней таблице этот же набор атрибутов называют внешним ключом отношения (**FOREIGN KEY**). В дочерней таблице множество кортежей, определяемых внешним ключом, связано с единственным кортежем родительской таблицы. Для получения первичного ключа используют индекс **PRIMARY**. Ключевые поля, по которым осуществляется связь таблиц, должны иметь одинаковый тип данных.

**Определение условий целостности данных**

Для реляционных баз данных существуют ограничения целостности трех видов: по сущностям; по ссылкам и целостность, определяемая пользователем.

Для поддержания первого вида целостности СУБД контролирует уникальность первичного ключа и вводит запрет на неопределенные значения первичного ключа.

Целостность по ссылкам обеспечивается механизмом каскадирования изменений первичного ключа, при котором согласованно изменяются также и внешние ключи, или механизмом, который отвергает изменения первичного ключа, если на него есть ссылки в дочерних таблицах.

Для поддержания третьего вида целостности СУБД располагает специальными средствами, которые обеспечивают контроль вводимых данных на уровне поля. Целостность, определяемая пользователем, соответствует набору ограничений, характерных для данной предметной области. Можно ограничить уникальность какого-либо поля, его принадлежность заданному интервалу значений или принадлежность перечисленному множеству. Пользовательские функции для контроля данных можно задать в виде хранимой процедуры.

Для создания первичных и внешних ключей можно использовать команду CREATE TABLE а также команду ALTER TABLE. Приведем синтаксис команды ALTER TABLE:

ALTER TABLE имя таблицы

ADD PRIMARY KEY(имя поля, имя поля ...)

ADD FOREIGN KEY имя внешнего ключа

(имя поля, имя поля ...)

REFERENCES имя ссылочной таблицы

[ON DELETE

RESTRICT ⎜CASCADE ⎜SET NULL]

Эта команда использует три различных варианта поддержания ссылочной целостности. При выборе параметра RESTRICT вводятся ограничения. Команда не будет выполнена при наличии ссылочных записей.

При выборе параметра CASCADE согласованно удаляются также ссылочные записи.

SET NULL означает установление неопределенных значений, т.е. пустых полей таблицы.

Пример. Для обеспечения целостности покупателей в базе данных создадим первичный ключ для таблицы CUSTOMER и внешние ключи для всех таблиц, которые содержат код покупателя:

ALTER TABLE CUSTOMER

ADD PRIMARY KEY(CUSTOMERNO)

ALTER TABLE ORDSALE

ADD FOREIGN KEY CUSTNO (CUSTOMERNO)

REFERENCES CUSTOMER

ON DELETE RESTRICT

Если указан параметр RESTRICT, то удалить покупателя в таблице CUSTOMER можно только в том случае, если он не указан ни в какой другой таблице, имеющей ссылку на таблицу CUSTOMER. В этом случае для удаления покупателя из таблицы CUSTOMER предварительно необходимо удалить все записи в таблице все записи в таблице ORDSALE, содержащие ссылки на этот номер покупателя. В некоторых случаях желательно, чтобы удаление главной таблицы приводило к удалению подчиненных. Для этого используется фраза: ON DELETE CASCADE.

**2. Материалы и оборудование**

1. Компьютер.
2. Программа Денвер.

**3. Задания к лабораторной работе**

1. Изучить теоретическую часть к лабораторной работе, изложенную в методических указаниях.

2. Ответить на контрольные вопросы к лабораторной работе, использовать лекции и указанную в них литературу, теоретический материал к лабораторной работе.

3.Выполнить задание**,** согласно своему варианту, изложенное в методических указаниях.

Для связанных отношений *«Журнал товаров» и «Журнал реализации»* записать команду, предусматривающую каскадирование при удалении записей.

*Атрибуты отношения «Журнал товаров»:*

- уникальный ключ товара;

* наименование товара;
* цена 1 товара;
* страна производитель.

*Атрибуты отношения «Журнал реализации»*:

* уникальный ключ операции;
* уникальный код потребителя;
* уникальный ключ товара;
* номер накладной на реализацию;
* дата реализации товара;
* количество реализованного товара;

- общая стоимость реализованного товара.

4. Оформить отчет по лабораторной работе (Приложение 1).

**4. Вопросы к лабораторной работе**

12.1. Каким образом осуществить каскадирование при удалении записей с помощью SQL?

12.2. Каким образом осуществить ввод неопределенных значений с помощью SQL?

12.3. Каким образом с помощью SQL осуществить модификацию данных с использованием каскадирования?

**5. Задания для самостоятельной работы**

12.1.Составить команду обновления данных с помощью SQL, предусматривающую каскадирование изменений первичного ключа, при котором согласованно изменяются также и внешние ключи.

# Лабораторная работа № 13

**ОБЪЕДИНЕНИЕ, ПЕРЕСЕЧЕНИЕ И РАЗНОСТЬ ОТНОШЕНИЙ. ОПЕРАТОРЫ UNION, INTERSECT, EXEPT**

**Цель работы:** изучение синтаксиса теоретико-множественных операций.

**1. Теоретическая часть**

**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-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.

**Пример**. **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**. 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 1, добавить строку **OR UNITPRICE < 500**, то его результирующая таблица будет аналогична результирующей таблице, соответствующей запросу 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.

**2. Материалы и оборудование**

1. Компьютер.
2. Программа Денвер.

**3. Задания к лабораторной работе**

1. Изучить теоретическую часть к лабораторной работе, изложенную в методических указаниях.

2. Ответить на контрольные вопросы к лабораторной работе, использовать лекции и указанную в них литературу, теоретический материал к лабораторной работе.

3.Выполнить задание**,** согласно своему варианту, изложенное в методических указаниях.

Варианты заданий к лабораторной работе № 13:

13.1. В состав реляционной БД входят отношения  
CUSTOMER ( CUSTOMERNO, FIRSTNAME, LASTNAME, CITY, CREDITLIMIT) и ORDSALE ( STOCK, CUSTOMERNO, NAME). Составить SQL-запрос с помощью оператора UNION для выбора фамилий покупателей, которые имеют кредит (CREDITLIMIT), превышающий 10000, а также фамилий покупателей, которым был продан товар с наименованием (NAME), содержащим символы “DVD”.

13.2. В состав реляционной БД входят отношения R1(код\_покупателя, фамилия, имя, отчество, город) и R2(код\_товара, наименование, цена, количество, сумма, код\_покупателя). Составить SQL-запрос для получения фамилий, имен и отчеств покупателей, которым продан товар с наименованием «Intel» и покупателей, которым продан товар с наименованием, содержащим в конце последовательность символов«HDD». Использовать оператор UNION.

4. Оформить отчет по лабораторной работе (Приложение 1).

**4. Вопросы к лабораторной работе**

13.1. При каких условиях можно объединить две таблицы?

13.2. Какие условия соответствуют совместимости таблиц по объединению?

13.3. Можно ли объединить таблицы различной арности?

**5. Задания для самостоятельной работы**

13.1.Рассмотреть самостоятельно реализацию теоретико-множественных операций пересечения и разности.

# Лабораторная работа № 14

# СОЕДИНЕНИЕ ОТНОШЕНИЙ. ОПЕРАТОР JOIN

**Цель работы:** изучение соединений.

**1. Теоретическая часть**

Соединение – одна из замечательных операций реляционной алгебры, позволяющая получать выборку данных из нескольких таблиц сразу. В языке SQL этой операции соответствует оператор JOIN.

**Соединение** отношений R1 и R2:



обозначается символом «», под которым на месте нижнего индекса записывают условие соединения. В условии используют обозначения: θ *—* арифметический оператор сравнения (<,=,**>,≥,≤,≠**); k—арность отношения R1; i и j—номера столбцов соответственно в отношениях R1 и R2*.*

Если θ является арифметическим оператором равенства, то операцию называют эквисоединением.

**Эквисоединение**

Рассмотрим пример соединения отношений с поименованными столбцами R11(А,В,С) и R12(D,Е). Вычислим эквисоединение при условии, что элементы столбцов В и D равны.



Отношения R11(А,В,С) и R12(D,Е) представлены в таблицах 14.1 и 14. 2.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Таблица 14.1 - R11(А,В,С) | | |  | Таблица 14.2. - R12(D,Е) | |
| A | B | C |  | D | Е |
| а | б | с |  | а | и |
| а | и | р |  | е | к |
| д | е | ж |  |  |  |

Промежуточное действие – вычисление декартова произведения: R11\*R12 .  Результат представим в таблице 14.3.

Таблица 14.3 - R11\*R12

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| А | В | С | D | Е |
| 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 В и D равны. Результат содержит одну строку, представленную в таблице 14.4.

Таблица 14.4 – R13

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| A | B | C | Д | Е |
| д | е | ж | е | к |

**Естественное соединение** отношений R1 и R2*.* Эта операция применяется только тогда, когда в отношениях R1 и R2имеются одинаковые столбцы.Естественное соединение - это комбинация двух отношений по общим атрибутам. Условие реализации R1 ∩ R2 ≠ Ø.

Пусть отношения R1 и R2 имеют соответственно схемы

где имена A1,A2,...,Ak 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:



R1.A1—имя столбца отношения R1×R2, соответствующего столбцу А1 в отношении R1; R2.A2 — имя столбца отношения R1×R2, соответствующего столбцу А1 в отношении R2. Запись «имя\_таблицы**.**имя\_атрибута», например, R1.A1 называется «уточненное имя атрибута».

Пример. Получим естественное соединение отношений ВЕДОМОСТЬ\_1(Фамилия, алгебра) и ВЕДОМОСТЬ\_2(Фамилия, геометрия). Отношения представлены в таблицах 14.5 и 14.6. В этих отношениях имеется один совпадающий столбец. Условие естественного соединения – в одноименных столбцах атрибуты должны принимать совпадающие значения.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Таблица 14.5 -ВЕДОМОСТЬ\_1 | |  | Таблица 14.6 - ВЕДОМОСТЬ\_2 | |
| **Фамилия** | **алгебра** |  | **Фамилия** | **геометрия** |
| Иванов | отлично |  | Иванов | хорошо |
| Петров | хорошо |  | Петров | отлично |

Получим декартово произведение отношений ВЕДОМОСТЬ\_1 и ВЕДОМОСТЬ\_2. Результат представим в таблице 14.14.

Таблица 14. 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 14. 8.

Таблица 14. 8 – результат селекции

σ 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 8 нужно устранить избыточность. Для этого выполним проекцию на все разные столбцы. Результат – в таблице 14. 9.

Таблица 14.9 – 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.

**Пример 1. Естественное соединение.**

Рассмотрим базу данных, содержащую две таблицы, представленные на рисунке 14.1.

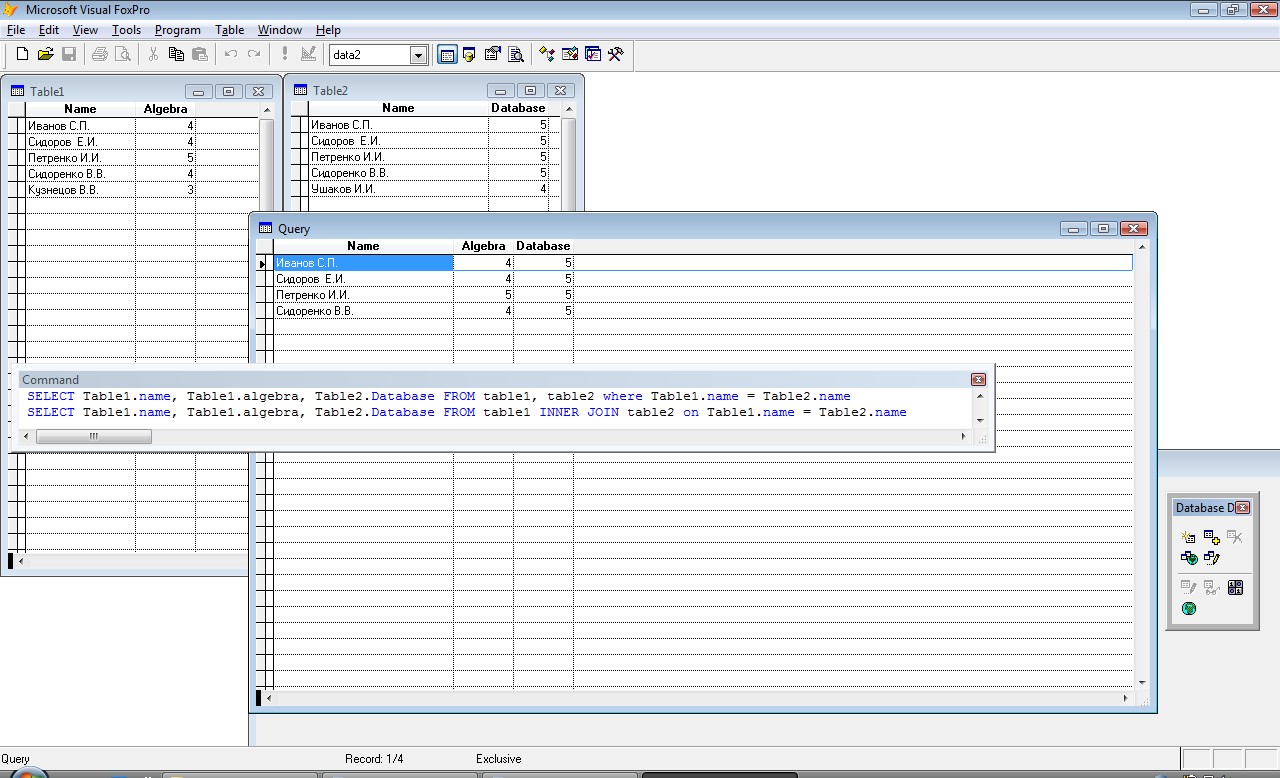


Рисунок 14.1 – Получение естественного соединения

Отношение Table1 представляет собой ведомость по алгебре – Table1(Name, Algebra) и отношение Table2(Name,Database) представляет собой ведомость по БД. Условие применения естественного соединения выполняется, так как у таблиц имеется общий столбец – Name. Естественное соединение таблиц можно получить разными способами, показанными на рисунке 14.1. 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

**Пример 2. Левое соединение**.

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 14.2. 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.

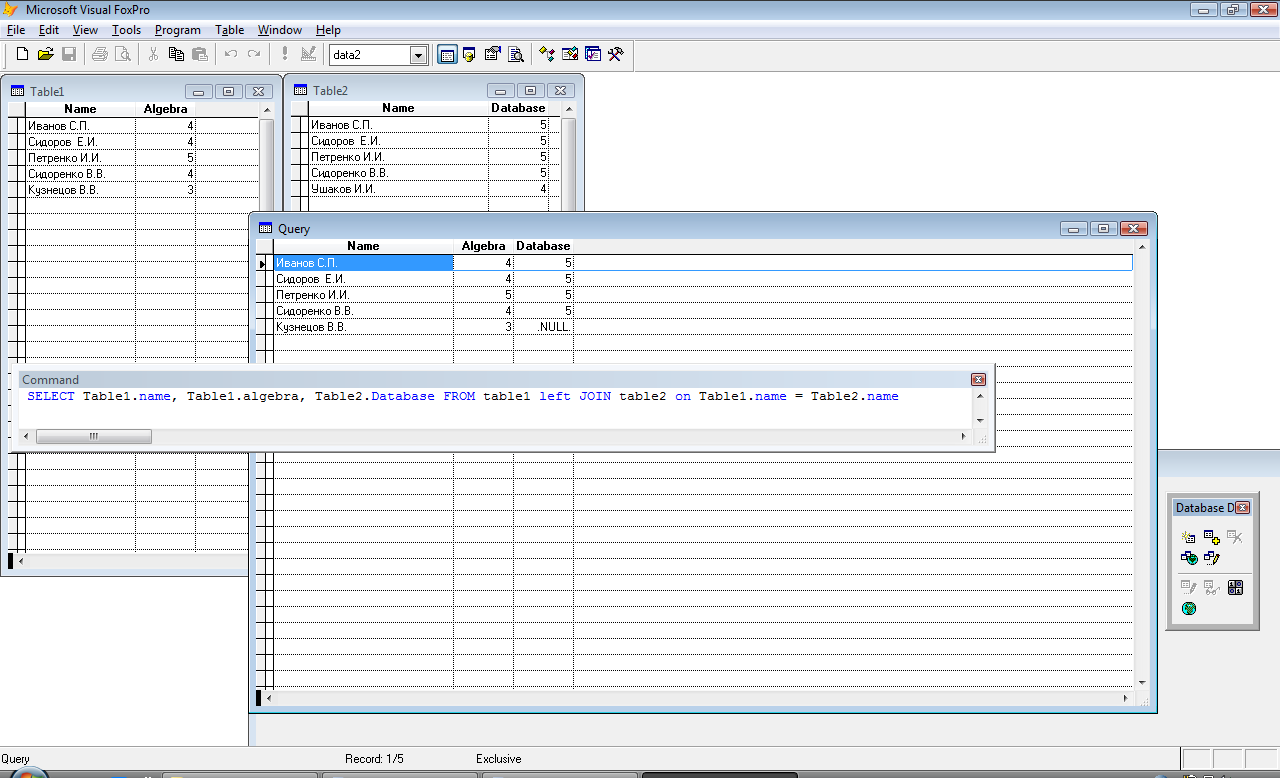


Рисунок 14.2 – Получение левого соединения

**Пример 3. Правое соединение**

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).

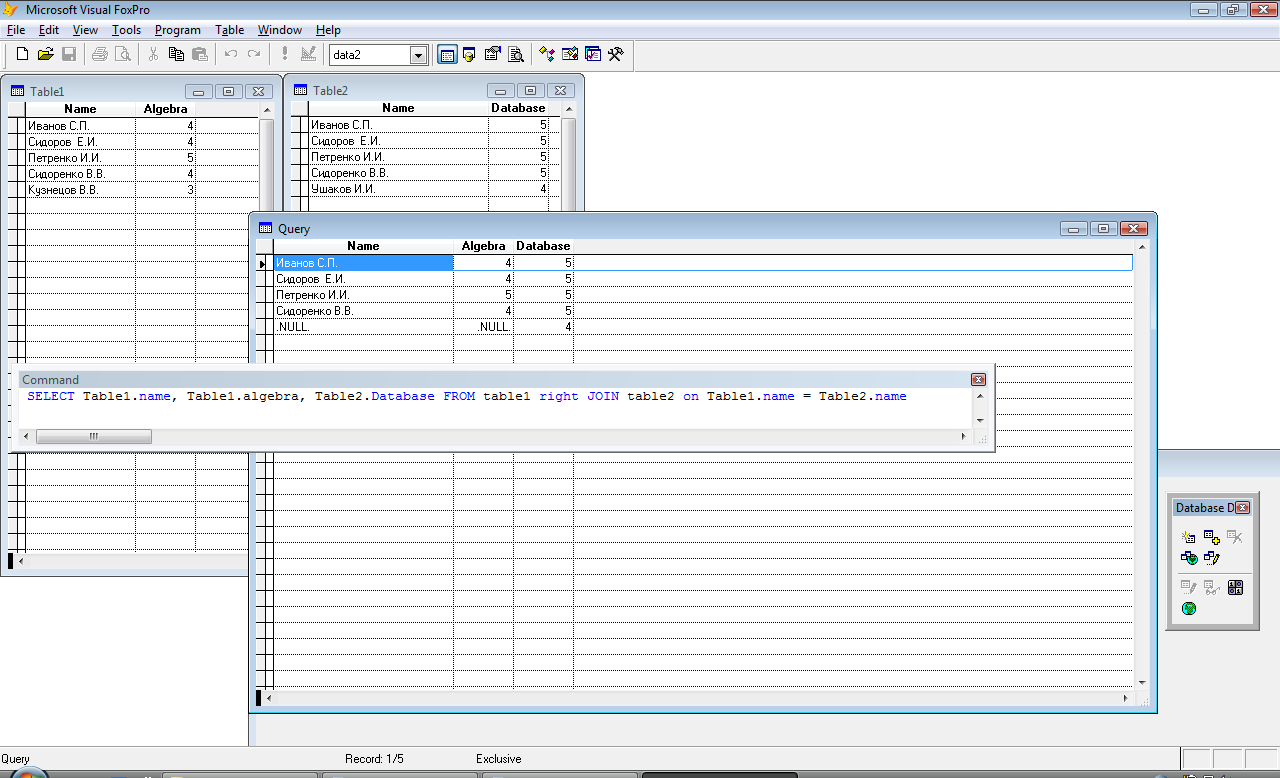


Рисунок 14.3 – Получение правого соединения

**2. Материалы и оборудование**

1. Компьютер.
2. Программа Денвер.

**3. Задания к лабораторной работе**

1. Изучить теоретическую часть к лабораторной работе, изложенную в методических указаниях.

2. Ответить на контрольные вопросы к лабораторной работе, использовать лекции и указанную в них литературу, теоретический материал к лабораторной работе.

3.Выполнить задание**,** согласно своему варианту, изложенное в методических указаниях.

Варианты заданий к лабораторной работе № 14: Изучить возможность получения неопределенных значений и привести пример выборки с получением .NULL.

4. Оформить отчет по лабораторной работе (Приложение 1).

**4. Вопросы к лабораторной работе**

14.1. К каким отношениям можно применить операцию «естественное соединение»?

14.2. В каких случаях в результате выборки получаются неопределенные значения?

**5. Задания для самостоятельной работы**

14.1. В деканат сданы ведомости группы по трем предметам (алгебра, геометрия, история), представляющие собой таблицы. Ведомости соответствуют схемам отношений:  
Алгебра (№ п/п, ФИО, номер зачетной книжки, оценка);

Геометрия (№ п/п, ФИО, номер зачетной книжки, оценка);

История (№ п/п, ФИО, номер зачетной книжки, оценка).

Получить сводную ведомость в виде отношения   
Ведомость(№ п/п, ФИО, номер зачетной книжки, алгебра.оценка, геометрия.оценка, история.оценка) с помощью СУБД и языка SQL.

# Лабораторная работа № 15

# АДМИНИСТРИРОВАНИЕ БД. ПРИВИЛЕГИИ. ПРЕДСТАВЛЕНИЯ

**Цель работы:** изучение операторов GRANT и REVOKE.

**1. Теоретическая часть**

**Привилегии**

Термин «привилегии» при работе с БД соответствует набору полномочий для выполнения определенных действий с объектами БД, к которым пользователь имеет право доступа. Управление доступом входит в обязанности администратора баз данных. Все привилегии можно разделить на системные и объектные. Рассмотрим операторы языка SQL, которые применяются для управления доступом к БД.

В системах клиент/сервер доступ к базе данных могут получать только зарегистрированные в системе пользователи. Для регистрации новых пользователей используется команда GRANT:

**GRANT** привилегия

**TO** имя пользователя

[ **IDENTIFIED BY** пароль ]

Перечень возможных системных привилегий в таблице 15.1.

Таблица 15.1 - Системные привилегии

|  |  |
| --- | --- |
| CONNECT (соединять, связывать, быть согласованным) | Имея эту привилегию, пользователь может подключаться к базе данных и работать с объектами, к которым он имеет привилегии по доступу. Пользователь может создавать представления, синонимы и межтабличные связи. |
| RESOURCE (ресурсы) | Эта привилегия разрешает пользователю создавать объекты базы данных, включая таблицы и индексы. |
| DBA  (АБД) | Эта привилегия разрешает пользователю выполнять действия администратора базы данных. |

Пример:

GRANT DBA TO SYSADM IDENTIFIED BY SYSTEM

**Предоставление доступа к отдельным таблицам**

Кроме предоставления общих привилегий пользователю, можно определить уточненные привилегии на объект БД, например, таблицу или даже на ее отдельные поля. Для этого используется следующий вариант команды GRANT:

**GRANT [SELECT] [,INSERT] [,DELETE] [,UPDATE** (список столбцов) **]**

**ON** имя таблицы ⎜ имя представления

**TO** имя пользователя ⎜ **PUBLIC**

**[WITH GRANT OPTION]**

SQL поддерживает два аргумента для команды **GRANT,** которые имеют специальное значение. Это **ALL PRIVILEGES** (или ALL) и **PUBLIC.**

**ALL PRIVILEGES** (или ALL) заменяет список всех привилегий. **PUBLIC** означает, чтодоступом пользуются все пользователи.

Команда **WITH GRANT OPTION** позволяет пользователю передавать привилегии другим пользователям.

Примеры:

**1) GRANT ALL ON students TO Ivanov WITH GRANT OPTION**

Эта команда предоставляет все полномочия по работе с таблицей (или представлением) students пользователю Иванову, он имеет право передавать привилегии другим пользователям.

**2) GRANT UPDATE**(f\_name, l\_name, city) **, SELECT ON** Customer **TO PUBLIC**

Эта команда предоставляет возможность всем пользователям делать запросы к таблице Customer и модифицировать столбцы f\_name, l\_name, city.

**3) Grant read access ON library TO PUBLIC**

Эта команда предоставляет возможность всем пользователям получать доступ «только для чтения» к таблице library.

Отмена ранее предоставленных привилегий осуществляется по команде **REVOKE**, имеющей следующий синтаксис:

**REVOKE** список операций ⎜ **ALL PRIVILEGES**

**ON** имя \_объекта

**FROM** список пользователей⎜ **PUBLIC**

**[cascade⎜restrict]**

Применение команды **cascade** отменяет все привилегии, назначенные во фразе **WITH GRANT OPTION.**

Пример:

**REVOKE ALL PRIVILEGES ON** tab1

**FROM** user4 **restrict**

операция будет отвергнута, если user4 передал привилегии еще кому-то.

**Представления**

Термин «Представление – VIEW» часто используют как синоним термина «Курсор».

Представление является виртуальной таблицей, которая не существует физически, но пользователь может просматривать и выбирать данные из представления как из обычной таблицы. Для создания представлений используется команда CREATE VIEW, которая содержит имя представления и условие выборки из базовых таблиц.

Представления можно использовать для ограничения доступа к отдельным столбцам (вертикальное представление) или к строкам (горизонтальное представление) базовой таблицы. Возможно, создание и вертикально-горизонтального представления для ограничения доступа к столбцам и строкам одновременно.

Вертикальное представление соответствует операции «проекция» реляционной алгебры и осуществляется путем выбора столбцов π1,2(EMPLOYEE). Синтаксис команды:

**CREATE VIEW** tabel

**AS SELECT** T\_NUM, NAME

**FROM** EMPLOYEE

Горизонтальное представление соответствует операции «селекция» реляционной алгебры и осуществляется путем выбора строк:

Пример, представление “Незавершенные поставки”:

**CREATE VIEW** 0\_S\_NOT

**AS SELECT \***

**FROM** ORDSALE

WHERE SALEDATE=NULL

При создании представления запрос сохраняется в каталоге. В конструкции запроса можно использовать выборки из нескольких таблиц, но нельзя использовать фразы UNION & ORDER BY.

Для удаления представления используется команда **DROP VIEW**.

**2. Материалы и оборудование**

1. Компьютер.
2. Программа Денвер.

**3. Задания к лабораторной работе**

1. Изучить теоретическую часть к лабораторной работе, изложенную в методических указаниях.

2. Ответить на контрольные вопросы к лабораторной работе, использовать лекции и указанную в них литературу, теоретический материал к лабораторной работе.

3.Выполнить задание**,** согласно своему варианту, изложенное в методических указаниях.

Варианты заданий к лабораторной работе № 15:

15.1. Предоставить Петрову привилегии для выполнения всех операций с таблицей R22 с правом передачи привилегий.

15.2. Отменить привилегии, предоставленные в п.1.

15.3. Предоставить всем пользователям привилегию делать запросы к отношению R33.

15.4. Предоставить всем пользователям привилегию редактировать столбец «Фамилия» в отношении R33.

4. Оформить отчет по лабораторной работе (Приложение 1).

**4. Вопросы к лабораторной работе**

15.1. Как отменить привилегии, переданные по команде **WITH GRANT OPTION**?

15.2. Как отменить все привилегии, полученные Сидоровым?

**5. Задания для самостоятельной работы**

15.1. Изучите возможности предоставления привилегий с помощью опции WITH GRANT OPTION. Создайте таблицу и предоставьте пользователям привилегию самостоятельно делать записи в столбце ФИО.

# Лабораторная работа № 16

# СОСТАВЛЕНИЕ ТЕХНИЧЕСКОГО ЗАДАНИЯ НА РАЗРАБОТКУ САЙТА

**Цель работы:** научиться разрабатывать техническое задание на разработку сайта.

**1. Теоретическая часть**

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.

**2. Материалы и оборудование**

1. Компьютер.
2. Программа Денвер.

**3. Задания к лабораторной работе**

1. Изучить теоретическую часть к лабораторной работе, изложенную в методических указаниях.

2. Ответить на контрольные вопросы к лабораторной работе, использовать лекции и указанную в них литературу, теоретический материал к лабораторной работе.

3.Выполнить задание**,** согласно своему варианту, изложенное в методических указаниях.

Варианты заданий к лабораторной работе № 16: Составить техническое задание на разработку сайта для информационной системы по варианту, согласованному с преподавателем.

4.Оформить отчет по лабораторной работе (Приложение 1).

**4. Вопросы к лабораторной работе**

16.1. Назовите, какой раздел технического задания можно считать основным и почему.

16.2. Какую информацию должны содержать остальные разделы?

16.3.В чем основная сложность разработки технического задания?

**5. Задания для самостоятельной работы**

16.1. Составить техническое задание на разработку информационной системы для предметной области, соответствующей тематике курсового проекта по дисциплине «Базы данных».

# Лабораторная работа № 17. Функции PHP для работы с MySQL

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

На лабораторной работе реализуется интерактивная форма обучения – компьютерная симуляция.

**1. Теоретическая часть**

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. Синтаксис данной функции:

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](http://php.su/prepare/?phpini). 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()](http://php.su/functions/?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()](http://php.su/functions/?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()](http://php.su/functions/?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 веб-сервера 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])](http://php.su/functions/?mysql_select_db) 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)](http://php.su/functions/?mysql_errno);
* [mysql\_error(int $id)](http://php.su/functions/?mysql_error);

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()](http://php.su/functions/?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()](http://php.su/functions/?mysql_result) - получить необходимый элемент из набора записей;
* [mysql\_fetch\_array()](http://php.su/functions/?mysql_fetch_array) - занести запись в массив;
* [mysql\_fetch\_row()](http://php.su/functions/?mysql_fetch_row) - занести запись в массив;
* [mysql\_fetch\_assoc()](http://php.su/functions/?mysql_fetch_assoc) - занести запись в ассоциативный массив;
* [mysql\_fetch\_object()](http://php.su/functions/?mysql_fetch_object) - занести запись в объект.

Также можно определить количество содержащихся записей и полей в результате запроса. Функция [mysql\_num\_rows()](http://php.su/functions/?mysql_num_rows) позволяет узнать, сколько записей содержит результат запроса:

$q = mysql\_query("SELECT \* FROM mytable");

echo "В таблице mytable ".mysql\_num\_rows($q)." записей";

Запись состоит из полей (колонок). С помощью функции [mysql\_num\_fields()](http://php.su/functions/?mysql_num_fields) можно узнать, сколько полей содержит каждая запись результата:

$q = mysql\_query("SELECT \* FROM mytable");

echo "В таблице 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)](http://php.su/functions/?mysql_result);

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] <br>";

}

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 результах SQL-запросов.

* Функция [mysql\_field\_name(int $result, int $offset)](http://php.su/functions/?mysql_field_name) 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)](http://php.su/functions/?mysql_field_type) возвращает тип поля с номером $offset в результате $result (номер задается относительно результата, а не таблицы);
* Функция [mysql\_field\_flags(int $result, int $offset)](http://php.su/functions/?mysql_field_flags) возвращает пречисленные через пробел флаги (модификаторы), которые имеются у поля с номером $offset. Перечислим все поддерживаемые MySQL флаги (Таблица 17.1).

Таблица 17.1 – 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>";

?>

**2. Материалы и оборудование**

1. Компьютер.
2. Программа DENVER.

**3. Задания к лабораторной работе**

1. Изучить теоретическую часть к лабораторной работе, изложенную в методических указаниях.

2. Ответить на контрольные вопросы к лабораторной работе, использовать лекции и указанную в них литературу, теоретический материал к лабораторной работе.

3.Выполнить задание**,** согласно своему варианту, изложенное в методических указаниях. Варианты заданий к лабораторной работе № 17:

17.1. Создать скрипт. Для этого создать 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);

// Выводим заголовок таблицы:

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");

// Выводим таблицу:

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>";

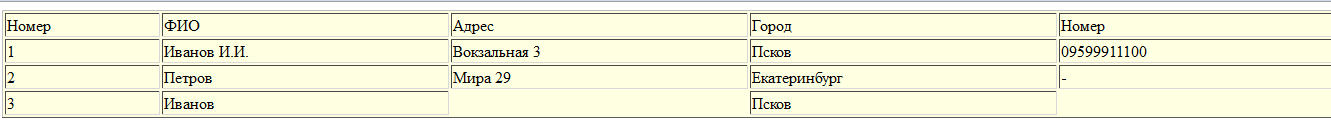
}

echo "</table>";

?>

В результате получаем таблицу 17.2.

Таблица 17.2 – Таблица CLIENTS



Для вывода записей полей всех таблиц выполнить:

$f = mysql\_fetch\_array($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>";

// Выводим заголовок таблицы Заказы:

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");

// Выводим таблицу:

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>";

// Выводим заголовок таблицы Инструменты:

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");

// Выводим таблицу:

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.w19.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 17.3а – 17.3д.

Таблица 17.3-Результаты работы скрипта

|  |  |
| --- | --- |
| а  б  в  г |  |
| д |  |

Проверяем, добавлена ли запись в базу данных (таблица 17.4)

Таблица 17.4 – Добавление записей



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 17.5).

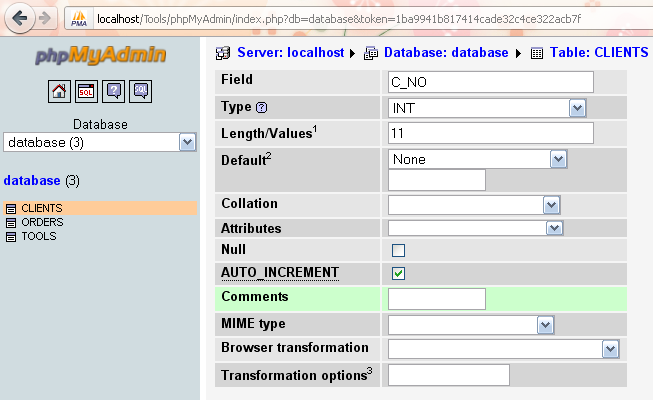
****

Рисунок 17.5 – Добавление AUTO\_INCREMENT

Проверьте, как будет добавляться запись теперь.

17.2. Организовать ввод данных для всех таблиц.

4. Оформить отчет по лабораторной работе (Приложение 1).

**4. Вопросы к лабораторной работе**

17.1. Каким образом осуществляется заполнение таблиц?

17.2. Для чего используется AUTO\_INCREMENT?

17.3. В чем разница между постоянным соединением и обыкновенным соединением с MySQL?

17.4. Опишите отличие функций mysql\_connect() и [mysql\_pconnect()](http://php.su/functions/?mysql_pconnect)

**5. Задания для самостоятельной работы**

17.1. Проанализируйте возможности скриптов**.**

17.2. С помощью web –интерфейса необходимо организовать выборку из базы данных по заданному пользователем условию.

17.3. Проверьте работоспособность скрипта, обрабатывающего введенные в форму данные.

# Лабораторная работа №18.

# УДАЛЕНИЕ ДАННЫХ И ОБНОВЛЕНИЕ ДАННЫХ В БД

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

На лабораторной работе реализуется интерактивная форма обучения – компьютерная симуляция.

**1. Теоретическая часть**

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:

DELETE FROM Table Name WHERE condition

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 19.

<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, удаляющий запись

$strSQL = "DELETE FROM CLIENTS WHERE C\_NO = 3";

mysql\_query($strSQL);

// Закрыть соединение с БД

mysql\_close();

?>

<h1>Запись удалена!</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 информаци 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 ());

// Выбор БД

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>

**3. Задания к лабораторной работе**

1. Изучить теоретическую часть к лабораторной работе, изложенную в методических указаниях.

2. Ответить на контрольные вопросы к лабораторной работе, использовать лекции и указанную в них литературу, теоретический материал к лабораторной работе.

3.Выполнить задание**,** согласно своему варианту, изложенное в методических указаниях. Варианты заданий к лабораторной работе № 18:

18.1. Организовать обновление базы данных посредством web интерфейса (кнопкой).

4. Оформить отчет по лабораторной работе (Приложение 1).

**4. Вопросы к лабораторной работе**

18.1. Каким образом осуществляется удаление таблиц?

18.2. Каким образом осуществляется удаление строк?

18.3. Как использовать сессии в PHP для сохранения и получения информации?

18.4. Каким образом используются пароль и логин?

18.5. В чем разница между операторами DELETE и UPDATE?

**5. Задания для самостоятельной работы**

18.1. Организовать удаление записей базы данных, определенных пользователем, посредством web интерфейса.

# Лабораторная работа №19.

# Изучение PHP cессий

**Цель работы:** научиться работать с сессиями **PHP.**

На лабораторной работе реализуется интерактивная форма обучения – компьютерная симуляция.

**1. Теоретическая часть**

**Введение в PHP cессии**

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.

[РНР-сессии](http://www.php.net/manual/en/book.session.php) 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 РНР-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 сохраняться, 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. Она может выглядеть так (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>

Затем создадим файл login.php. В этом файле мы проверяем, введены ли корректные username и password. 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

// Проверить корректность username и password

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 РНР-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>

**2. Материалы и оборудование**

1. Компьютер.
2. Программа MS Visual Studio.

**3. Задания к лабораторной работе**

1. Изучить теоретическую часть к лабораторной работе, изложенную в методических указаниях.

2. Ответить на контрольные вопросы к лабораторной работе, использовать лекции и указанную в них литературу, теоретический материал к лабораторной работе.

3.Выполнить задание**,** согласно своему варианту, изложенное в методических указаниях. Варианты заданий к лабораторной работе № 19:

19.1. Организовать сессию в вашей системе без пароля.

4. Оформить отчет по лабораторной работе (Приложение 1).

**4. Вопросы к лабораторной работе**

19.1. Сравните сессии без пароля и сессии с паролем и логином.

19.2. Для чего используются сессии?

**5. Задания для самостоятельной работы**

19.1. Организовать сессию вашей системы с паролем и логином.

# Лабораторная работа № 20. ПЕРЕДАЧА ПЕРЕМЕННЫХ ЧЕРЕЗ URL

**Цель работы:** научиться использовать значения переменных, передаваемых через URL

**1. Теоретическая часть**

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](http://php.net/manual/en/reserved.variables.get.php) 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 получить переменные так:

$\_GET["name"]

$\_GET["age"]

Добавим в пример ещё одну переменную:

<html>

<head>

<title>Строка запроса</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>

**2. Материалы и оборудование**

1. Компьютер.
2. Программа Денвер.

**3. Задания к лабораторной работе**

1. Изучить теоретическую часть к лабораторной работе, изложенную в методических указаниях.

2. Ответить на контрольные вопросы к лабораторной работе, использовать лекции и указанную в них литературу, теоретический материал к лабораторной работе.

3.Выполнить задание**,** согласно своему варианту, изложенное в методических указаниях. Варианты заданий к лабораторной работе № 20:

20.1. Изучить передачу переменных по приведенным примерам.

20.2. Организовать передачу нескольких переменных.

4. Оформить отчет по лабораторной работе (Приложение 1).

**4. Вопросы к лабораторной работе**

20.1. Характеристика URL.

20.2. Каким образом осуществляется передача переменных через URL?

20.3. Описание синтаксиса URL.

**5. Задания для самостоятельной работы**

20.1.Изучить синтаксис URL для передачи переменных.

20.2. Организовать передачу трех переменных.

# Лабораторная работа №21. ИСПОЛЬЗОВАНИЕ ВНЕШНЕГО ФАЙЛА ДЛЯ ХРАНЕНИЯ И ЗАГРУЗКИ ВНЕШНИХ ДАННЫХ

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

На лабораторной работе реализуется интерактивная форма обучения – компьютерная симуляция.

**1. Теоретическая часть**

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

Для открытия текстового файла используем функцию [fopen](http://php.net/manual/en/function.fopen.php). 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](http://php.net/manual/en/function.fgets.php) 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](http://php.net/manual/en/function.feof.php) (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](http://php.net/manual/en/function.fread.php). При работе с очень большими текстовыми файлами помните, что [fread](http://php.net/manual/en/function.fread.php) использует больше ресурсов, чем [fgets](http://php.net/manual/en/function.fgets.php). 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](http://ru.html.net/tutorials/php/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>

**2. Материалы и оборудование**

1. Компьютер.
2. Программа Денвер.

**3. Задания к лабораторной работе**

1. Изучить теоретическую часть к лабораторной работе, изложенную в методических указаниях.

2. Ответить на контрольные вопросы к лабораторной работе, использовать лекции и указанную в них литературу, теоретический материал к лабораторной работе.

3.Выполнить задание**,** согласно своему варианту, изложенное в методических указаниях. Варианты заданий к лабораторной работе № 21:

21.1. Изучить теоретическое обоснование и приведенные в нем примеры.

21.2. Составить программу для загрузки внешних данных с использованием функции [feof](http://php.net/manual/en/function.feof.php) (for end-of-file/до конца файла).

21.3. Составить программу для загрузки внешних данных с использованием функции [fread](http://php.net/manual/en/function.fread.php).

21.4. Сравнить результаты использования функций [feof](http://php.net/manual/en/function.feof.php) (for end-of-file/до конца файла) и [fread](http://php.net/manual/en/function.fread.php).

4. Оформить отчет по лабораторной работе (Приложение 1).

**4. Вопросы к лабораторной работе**

21.1. Как использовать текстовые файлы для хранения данных?

21.2. Каким образом осуществляется работа с массивами?

21.3. Описание функции fopen(filename, mode)

**5. Задания для самостоятельной работы**

21.1. Проанализируйте использование циклов для загрузки внешних данных из текстовых файлов.

21.2.Создайте массив для загрузки внешних данных из текстовых файлов.

# Лабораторная работа №22. PHP И ПОЛЯ HTML-ФОРМ: ТЕКСТОВЫЕ ПОЛЯ, ТЕКСТОВАЯ ОБЛАСТЬ

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

**1. Теоретическая часть**

Использование элемента дизайна форм 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 Web-страницы. Первая страница содержит форму для ввода данных пользователя и имеет расширение .html. Вторая страница обрабатывает введенную информацию средствами языка PHP и имеет расширение .php.

**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">

Кто Ваш любимый автор?

<INPUT name="Author" type="text">

<BR><BR>

<INPUT type="submit" value= "Отправить">

</FORM>

</BODY>

</HTML>

Реализация данного кода представлена на рисунке 22.1

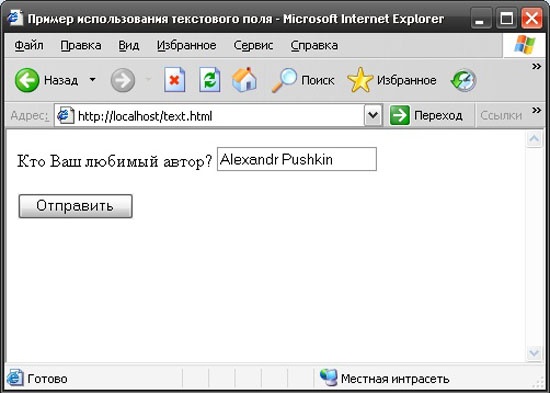


Рисунок 22.1 - Ввод любимого автора

Таким образом, как видно на рисунке 22.1, 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 22.2.

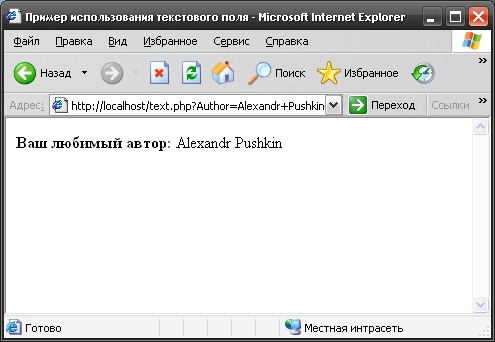


Рисунок 22.2 - Результат обработки введённого автора

Как видно на рисунке 22.2, на экран выводится информация о любимом авторе, то есть “Alexandr Pushkin”. Программа обработки введенных данных text.php, состоит из одной строки РНР-кода: “$\_GET ['Author']”, которая отображает содержимое переменной “$Author”. Данная переменная не генерируется в РНР-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 22.2, 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 Web-страницы. 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 РНР 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 22.3).

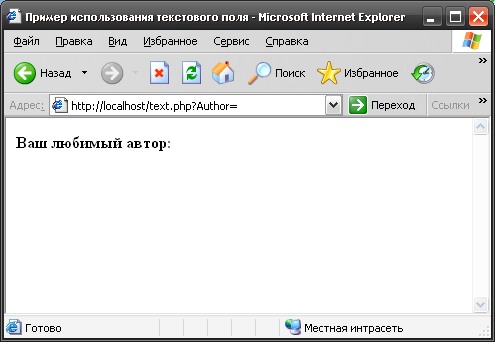


Рисунок 22.3 – Неправильное присваивание имени переменой

**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">

Перечислите Ваши любимые 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 22.4.

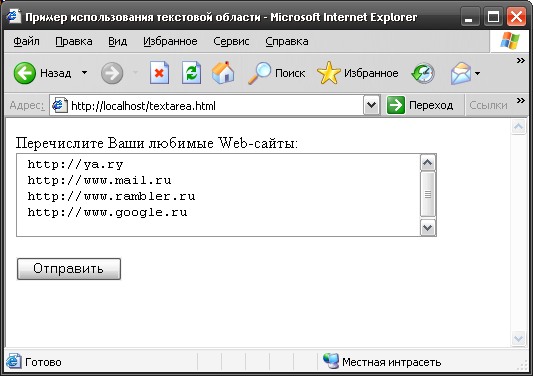


Рисунок 22.4 – Ввод информации в текстовую область

Таким образом, как видно на рисунке 22.4, 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>

Ваши любимые Web-сайты:

<?php

echo $\_POST['Websites'];

?>

</BODY>

</HTML>

Результат работы данного кода представлен на рисунке 22.5.

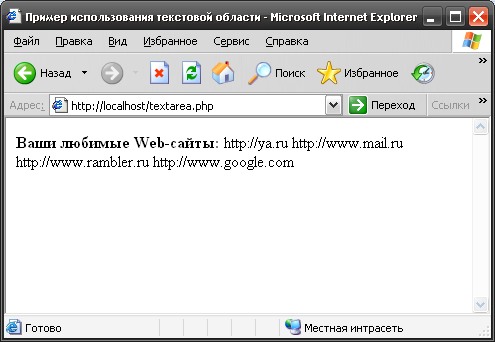


Рисунок 22.5 – Вывод на экран любимых сайтов

Как видно на рисунке 22.5, 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.

**2. Материалы и оборудование**

1. Компьютер.
2. Программа Денвер.

**3. Задания к лабораторной работе**

1. Изучить теоретическую часть к лабораторной работе, изложенную в методических указаниях.

2. Ответить на контрольные вопросы к лабораторной работе, использовать лекции и указанную в них литературу, теоретический материал к лабораторной работе.

3.Выполнить задание**,** согласно своему варианту, изложенное в методических указаниях. Варианты заданий к лабораторной работе № 22:

22.1. Изучить теоретическое обоснование и приведенные в нем примеры.

22.2. Составить программу для вывода информации в текстовую область.

4. Оформить отчет по лабораторной работе (Приложение 1).

**4. Вопросы к лабораторной работе**

22.1. Как использовать текстовые файлы для хранения данных?

22.2. Как использовать текстовую область для хранения данных?

**5. Задания для самостоятельной работы**

22.1. Проанализируйте использование данных из текстовых файлов.

# Лабораторная работа №23. PHP И ПОЛЯ HTML-ФОРМ: ФЛАЖКИ, ПЕРЕКЛЮЧАТЕЛИ

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

**1. Теоретическая часть**

**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">

Вы студент ОГУ?

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

<BR>

<BR>

<INPUT type="submit" value="Отправить">

</FORM>

</BODY>

</HTML>

Реализация данного кода представлена на рисунке 23.1

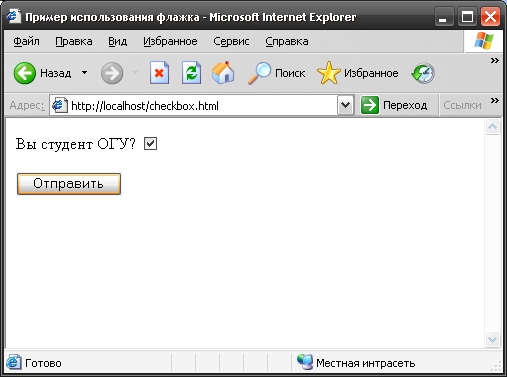


Рисунок 23.1 – Проверка принадлежности к числу студентов ОГУ

Таким образом, как видно на рисунке 23.1, 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 23.2.

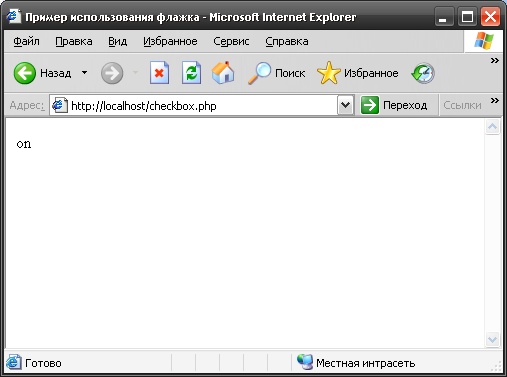


Рисунок 23.2 – Результат проверки

Как видно на рисунке 23.2, 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 23.3.

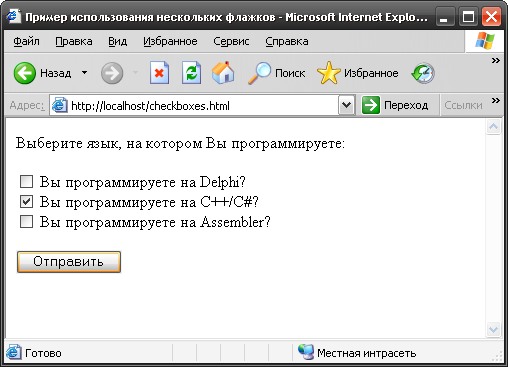


Рисунок 23.3 – Выбор языков программирования

Таким образом, как видно на рисунке 23.3, 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>Вы выбрали ответ:</P>

<?php

echo "$\_POST[Choice1]<br>";

echo "$\_POST[Choice2]<br>";

echo "$\_POST[Choice3]<br>";

?>

</BODY>

</HTML>

Результат работы данного примера представлен на рисунке 23.4.

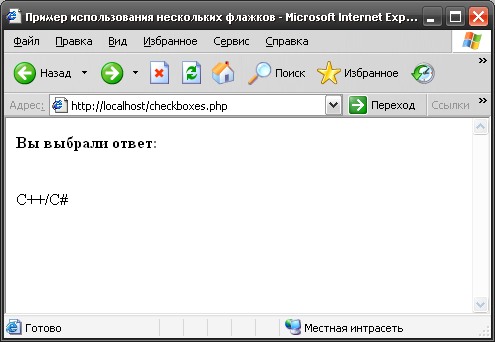


Рисунок 23.4 – Результат выбора языков

Как видно на рисунке 23.4, 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 PHP-переменные данные передаваться не будут.

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 23.5.

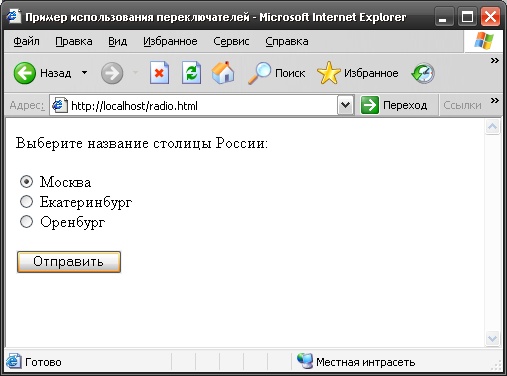


Рисунок 23.5 – Выбор столицы России

Таким образом, как видно на рисунке 23.5, 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 "Вы выбрали ответ: $\_GET[Question]";

?>

</BODY>

</HTML>

Реализация данного кода представлена на рисунке 23.6.

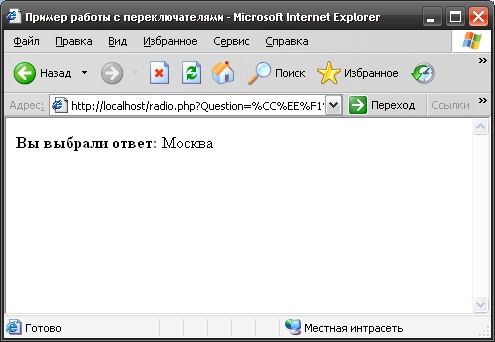


Рисунок 23.6 – Результат выбора столицы

Как видно на рисунке 23.6, 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].

**2. Материалы и оборудование**

1. Компьютер.
2. Программа Денвер.

**3. Задания к лабораторной работе**

1. Изучить теоретическую часть к лабораторной работе, изложенную в методических указаниях.

2. Ответить на контрольные вопросы к лабораторной работе, использовать лекции и указанную в них литературу, теоретический материал к лабораторной работе.

3.Выполнить задание**,** согласно своему варианту, изложенное в методических указаниях. Варианты заданий к лабораторной работе № 23:

23.1.Изучить теоретическое обоснование и приведенные в нем примеры.

23.2. Разработать форму с совместным использованием PHP и текстовых полей HTML-форм.

23.3. Разработать форму с совместным использованием PHP и флажков HTML-форм.

4. Оформить отчет по лабораторной работе (Приложение 1).

**4. Вопросы к лабораторной работе**

23.1. Как совместно использовать PHP и текстовые поля HTML-форм?

23.2. Как совместно использовать PHP и флажки HTML-форм?

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

**5. Задания для самостоятельной работы**

23.1. Разработать форму с совместным использованием PHP и переключателей HTML-форм.

# Лабораторная работа №24. PHP И ПОЛЯ HTML-ФОРМ: СПИСКИ

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

**1. Теоретическая часть**

**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">

<OPTION value="Стиральная машина">Стиральная машина</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 24.1.

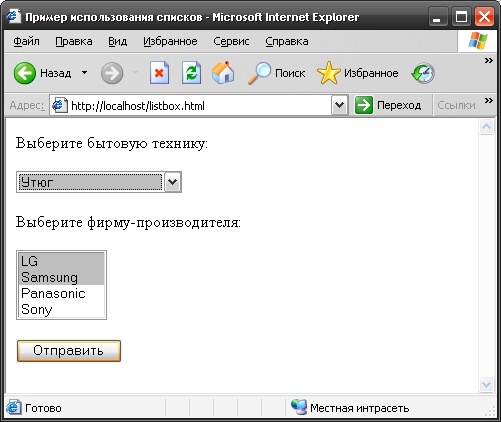


Рисунок 24.1 – Выбор техники и фирмы

Таким образом, как видно на рисунке 24.1, 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 24.2.

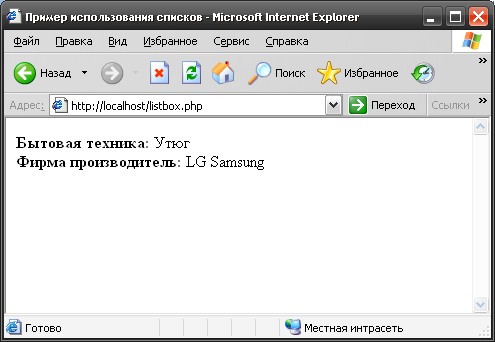


Рисунок 24.2 – Результат выбора техники и фирмы

Как видно на рисунке 24.2, 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].

**2. Материалы и оборудование**

1. Компьютер.
2. Программа Денвер.

**3. Задания к лабораторной работе**

1. Изучить теоретическую часть к лабораторной работе, изложенную в методических указаниях.

2. Ответить на контрольные вопросы к лабораторной работе, использовать лекции и указанную в них литературу, теоретический материал к лабораторной работе.

3.Выполнить задание**,** согласно своему варианту, изложенное в методических указаниях. Варианты заданий к лабораторной работе № 24:

24.1.Изучить теоретическое обоснование и приведенные в нем примеры.

24.2. Разработать форму с совместным использованием PHP и списков HTML-форм.

4. Оформить отчет по лабораторной работе (Приложение 1).

**4. Вопросы к лабораторной работе**

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

24.2. Как создать формы, предполагающие множественный выбор из нескольких вариантов?

**5. Задания для самостоятельной работы**

24.1. Проанализируйте средства создания форм, предполагающие множественный выбор из нескольких вариантов.

24.2. Разработайте форму для интерактивной работы пользователя, в которой будет реализован множественный выбор из нескольких вариантов.

# Лабораторная работа №25. PHP И ПОЛЯ HTML-ФОРМ: СКРЫТЫЕ ПОЛЯ ФОРМ

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

**1. Теоретическое обоснование**

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 РНР-переменных.

Чтобы использовать скрытое поле в PHP-странице, можно написать всю HTML-форму в операторах echo, передающих содержимое PHP-переменных 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 форм </TITLE></HEAD>

<BODY>

<?php

$Message1="Иванов";

$Message2="Петров";

$Message3="Cидоров";

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 25.1.

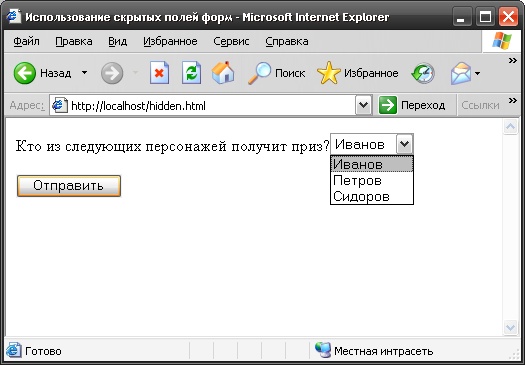


Рисунок 25.1 – Выбор персонажа

Таким образом, как видно на рисунке 25.1, 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:<br>";

echo "$\_GET[Hidden1]<br>";

echo "$\_GET[Hidden2]<br>";

echo "$\_GET[Hidden3]<br>";

echo "<br>Вывыбрали:<br>";

echo "$\_GET[ListBox]";

?>

</BODY>

</HTML>

Результат работы данного примера представлен на рисунке 25.2.

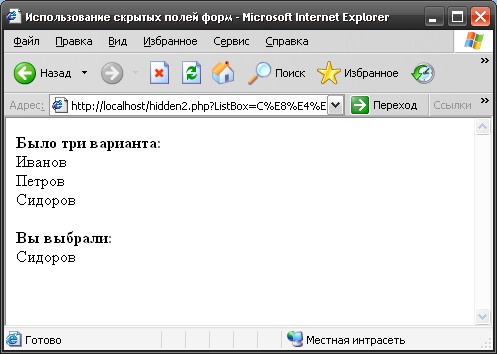


Рисунок 25.2 – Результат выбора персонажа

Как видно на рисунке 25.2, 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 echo-операторов создается форма. Она ничем не отличается от обычной HTML-формы, 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. Материалы и оборудование**

1. Компьютер.
2. Программа Денвер.

**3. Задания к лабораторной работе**

1. Изучить теоретическую часть к лабораторной работе, изложенную в методических указаниях.

2. Ответить на контрольные вопросы к лабораторной работе, использовать лекции и указанную в них литературу, теоретический материал к лабораторной работе.

3.Выполнить задание**,** согласно своему варианту, изложенное в методических указаниях. Варианты заданий к лабораторной работе № 25:

25.1.Изучить теоретическое обоснование и приведенные в нем примеры.

25.2.Создайте форму с использованием скрытых полей.

4. Оформить отчет по лабораторной работе (Приложение 1).

**4. Вопросы к лабораторной работе**

25.1. Как совместно использовать PHP и скрытые поля HTML-форм?

25.2. Характеристика скрытых элементов управления .

**5. Задания для самостоятельной работы**

25.1. Проанализируйте:

1) описание скрытых полей при совместном использовании PHP и HTML-форм;

2) описание полей для ввода паролей при совместном использовании PHP и HTML-форм.

25.2. Создайте форму с использованием скрытых полей, в которой будет реализован множественный выбор из нескольких вариантов.

# Лабораторная работа № 26. PHP И ПОЛЯ HTML-ФОРМ: ПОЛЯ ВВОДА ПАРОЛЕЙ, КНОПКИ, ЗНАЧЕНИЯ, ВОЗВРАЩАЕМЫЕ ФОРМАМИ, ПРОВЕРКА ОБЯЗАТЕЛЬНЫХ ПОЛЕЙ

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

**1.Теоретическое обоснование**

1.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:

Введите пароль:

<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.

2. 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 на обработчик, 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.

3. Использование значений, возвращаемых формами, в PHP-сценариях

Выше были рассмотрены все разновидности элементов управления формами и примеры передачи на сервер их содержимого средствами языка РНР. 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="Подать заявку">

<INPUT type="reset" value="Очистить">

</FORM>

</BODY>

</HTML>

Реализация данного кода представлена на рисунке 26.1.

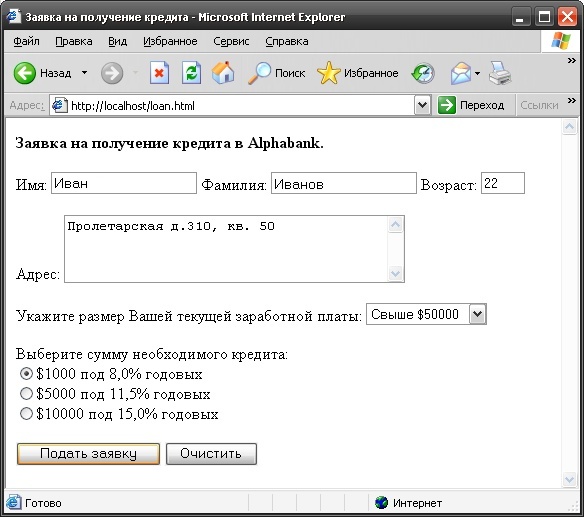


Рисунок 26.1 – Форма заявки на кредит

Таким образом, как видно на рисунке 26.1, 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>

<BR><BR>

<?php

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

$AgeAllowance = ($\_POST['Age']/10 - ($\_POST['Age']%10)/10)-1;

$LoanAllowance = $SalaryAllowance \* $AgeAllowance;

echo "Запрашиваемый кредит: $\_POST[Loan]<br>";

echo "Допустимая сумма кредита: $LoanAllowance<br><br>";

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 23.2.

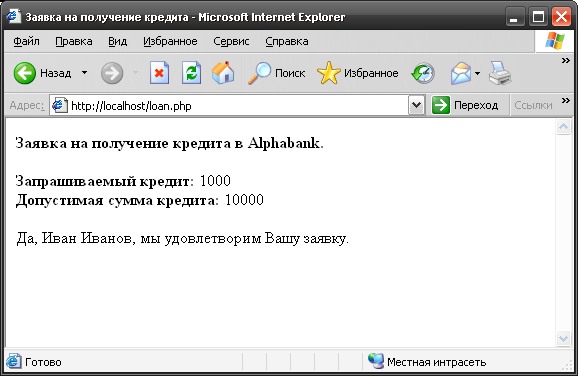


Рисунок 26.2 – Результат подачи заявки

Как видно на рисунке 26.2, 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.

4. 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>Oт $10000 до $25000</option>

<option value=25000>Oт $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="Подать заявку">

<INPUT type="reset" value="Очистить">

</FORM>

</BODY>

</HTML>

Реализация данного кода представлена на рисунке 26.3.

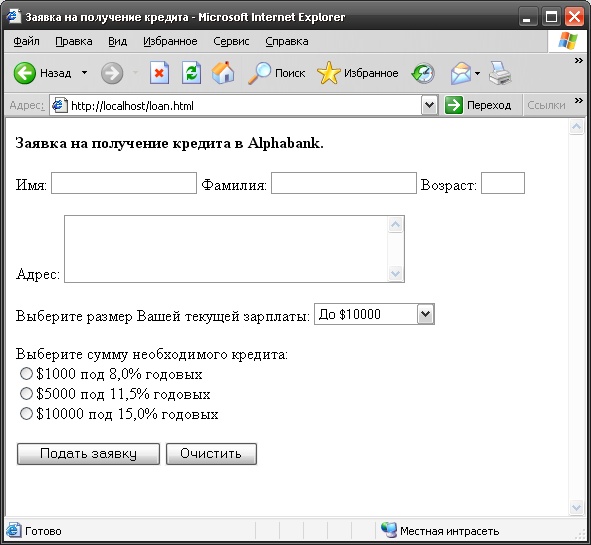


Рисунок 26.3 – Форма заявки на получение кредита

Таким образом, как видно на рисунке 26.3, 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>

На рисунке 26.4 показано одно из сообщений об ошибке, которое выводится на экран.

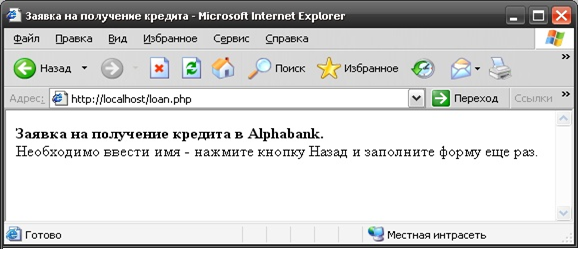


Рисунок 26.4 – Результат подачи заявки в банк. Сообщение об ошибке

Как видно на рисунке 26.4, 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].

**2. Материалы и оборудование**

1. Компьютер.
2. Программа Денвер.

**3. Задания к лабораторной работе**

1. Изучить теоретическую часть к лабораторной работе, изложенную в методических указаниях.

2. Ответить на контрольные вопросы к лабораторной работе, использовать лекции и указанную в них литературу, теоретический материал к лабораторной работе.

3.Выполнить задание**,** согласно своему варианту, изложенное в методических указаниях. Варианты заданий к лабораторной работе № 26:

26.1.Изучить теоретическое обоснование и приведенные в нем примеры.

26.2. Разработать форму, содержащую кнопки submit и reset.

26.3. Разработать форму, содержащую поля ввода паролей.

4. Оформить отчет по лабораторной работе (Приложение 1).

**4. Вопросы к лабораторной работе**

26.1. Как совместно использовать PHP поля ввода паролей HTML-форм?

26.2. Для чего используется оператор EXIT языка PHP?

**5. Задания для самостоятельной работы**

26.1. Разработать форму, использующую значения, возвращаемые PHP-сценариями.

26.2.Создайте форму с использованием оператора exit.

# Лабораторная работа №27. ИСПОЛЬЗОВАНИЕ СТАНДАРТНЫХ ОПЕРАТОРОВ ЯЗЫКА PHP ПРИ ОБРАБОТКЕ ДАННЫХ ПОЛЬЗОВАТЕЛЯ ИЗ ФОРМ: БУЛЕВЫ ОПЕРАТОРЫ, IF, ОПЕРАТОРЫ СРАВНЕНИЯ, ЛОГИЧЕСКИЕ ОПЕРАТОРЫ

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

**1. Теоретическая часть**

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 "На нуль делить нельзя!!!<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 27.1 и 27.2.

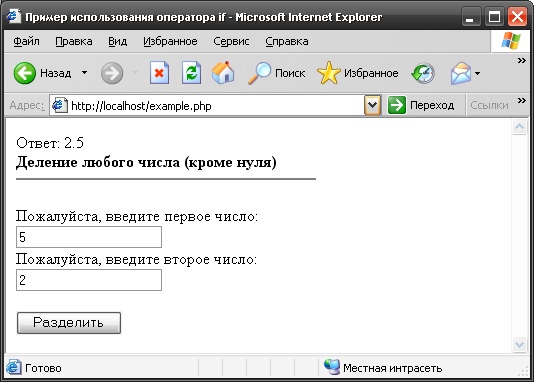


Рисунок 27.1 – Деление на нуль

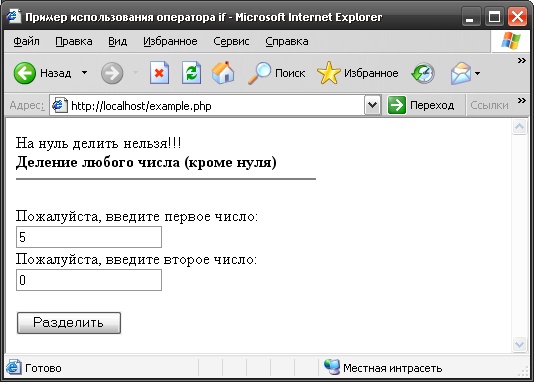


Рисунок 27.2 – Ошибка при делении на нуль

Таким образом, как видно на рисунке 27.1, 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 27.1, пользователь ввел корректные данные, программа выполнила вычисления и вывела результат на экран. При этом пользователю не было выведено никаких сообщений.

На рисунке 27.2 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 PHP-программа “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>Baшe число слишком большое.</B>";

echo "<br>Загаданное число:

$number. Вы проиграли. Попробуйте еще раз.<HR>";

}

else if ($\_POST['guess'] < $number)

{

echo "<B>Baшe число слишком маленькое.</B>";

echo "Загаданное число:

$number. Вы проиграли. Попробуйте еще раз. <HR>";

}

else

{

echo "<br> Загаданное число:

$number. Вы выиграли!<HR>";}

}

?>

<FORM method="POST" action="example.php">

<INPUT type="hidden" name="posted" value="true">

Угадайте число от 1 до 10:

<INPUT name="guess" type="text">

<BR>

<BR>

<INPUT type="submit" value="Отправить">

</FORM>

</BODY>

</HTML>

На рисунке 27.3 представлен результат выполнения данного кода.

Таким образом, как видно на рисунке 27.3, 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.

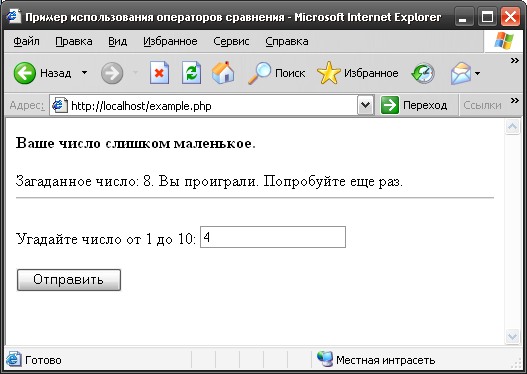


Рисунок 27.3 – Использование операторов сравнения

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'] – правильный ответ.</B><HR>";}

if ($\_POST['Question'] != "Москва") {

echo "<B>Неверно, $\_POST['Question'] – неправильный ответ.

</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="Отправить ответ">

</FORM>

</BODY>

</HTML>

На рисунке 27.4 представлен результат работы данного кода.

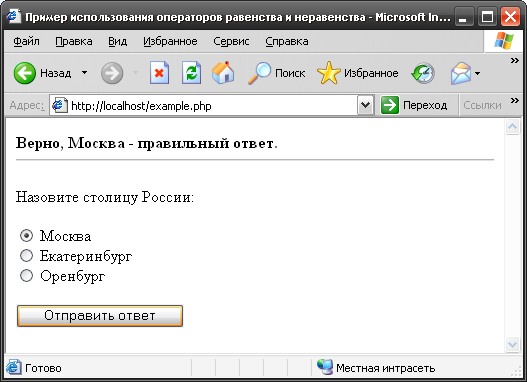


Рисунок 27.4 – Проверка на правильность выбора столицы России

Таким образом, как видно на рисунке 27.4, 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. Прокат автомобилей.</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="Отправить заявку">

</FORM>

</BODY>

</HTML>

<?php

}

?>

Результат работы данного кода представлен на рисунках 27.5 и 27.6.

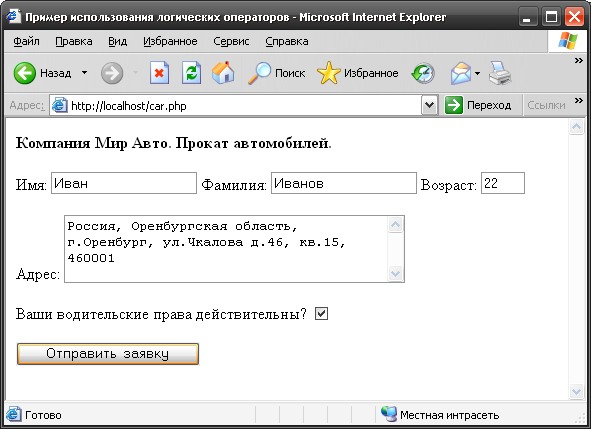


Рисунок 27.5 – Проверка возможности выдачи автомобилей

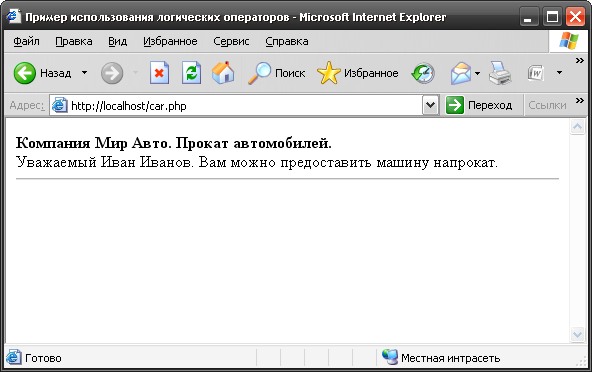


Рисунок 27.6 – Результат проверки

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.

**2. Материалы и оборудование**

1. Компьютер.
2. Программа Денвер.

**3. Задания к лабораторной работе**

1. Изучить теоретическую часть к лабораторной работе, изложенную в методических указаниях.

2. Ответить на контрольные вопросы к лабораторной работе, использовать лекции и указанную в них литературу, теоретический материал к лабораторной работе.

3.Выполнить задание**,** согласно своему варианту, изложенное в методических указаниях.

27.1. Изучить приведенные в теоретическом обосновании примеры.

27.2.Разработать форму с использованиемоператоров сравнения.

27.3.Разработать форму с использованиемоператоров неравенства для проверки введенной пользователем информации.

27.4.Разработать форму с использованиембулевых операторов и оператора IF.

4. Оформить отчет по лабораторной работе (Приложение 1).

**4. Вопросы к лабораторной работе**

27.1. Как использовать стандартные операторы языка PHP при обработке данных пользователя из форм?

27.2. Как использовать булевые операторы?

27.3. Как использовать операторы сравнения?

27.4. Как использовать логические операторы?

27.5. Пример использования сценария, содержащего стандартные операторы языка PHP.

**5. Задания для самостоятельной работы**

27.1. Проанализируйте описание стандартных операторов языка PHP при обработке данных.

27.2.Создать формы с использованием операторов if, сравнения и логических операторов.

# Лабораторная работа №28. ИСПОЛЬЗОВАНИЕ СТАНДАРТНЫХ ОПЕРАТОРОВ ЯЗЫКА PHP ПРИ ОБРАБОТКЕ ДАННЫХ ПОЛЬЗОВАТЕЛЯ ИЗ ФОРМ: ОПЕРАТОР SWITCH

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

**1. Теоретическая часть**

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 = 19.5;

$price = $price \* $citymodifier;

echo "Недельная стоимость проживания в 3-звёздочном отеле города Анталия: $price";

break;

case "Antaliafour":

$citymodifier = 19.5;

$starmodifier = 4;

$price = $price \* $citymodifier \* $starmodifier;

echo "Недельная стоимость проживания в 4-звёздочном отеле города Анталия: $price";

break;

case "Antaliafive":

$citymodifier = 19.5;

$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 "Выберите снова";

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">Три звездочки

<BR>

<INPUT name="grade" type="radio" value="four">Четыре звездочки<BR>

<INPUT name="grade" type="radio" value="five">Пять звездочек

<BR><BR>

<INPUT type="submit" value="Отправить заявку">

</FORM>

</BODY>

</HTML>

Реализация данного кода представлена на рисунке 28.1.

Таким образом, как видно на рисунке 28.1, 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.

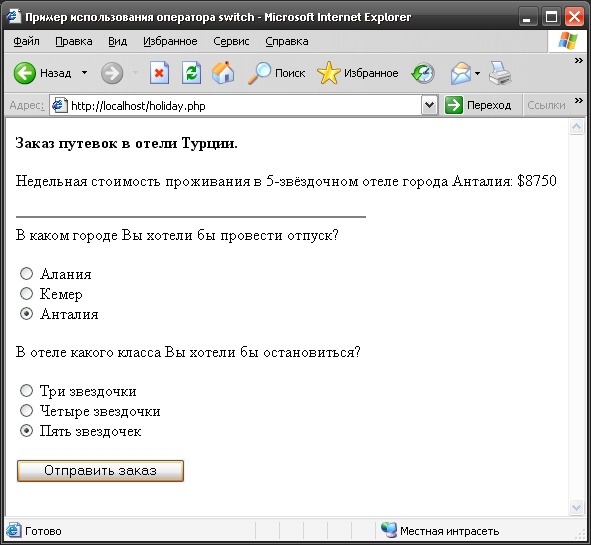


Рисунок 28.1 – Выбор города и класса отеля при заказе путевок

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>Пример использования цикла while</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 "Вы не выбрали сумму кредита<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

ежемесячные платежи<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="Подать заявку">

</FORM>

</BODY>

</HTML>

Реализация данного кода представлена на рисунке 28.2.

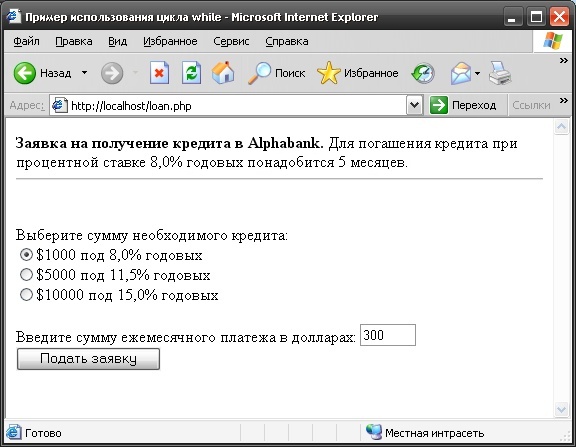


Рисунок 28.2 – Расчет времени погашения кредита в месяцах

Таким образом, как видно на рисунке 28.2, 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 28.2, 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 28.3).

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.

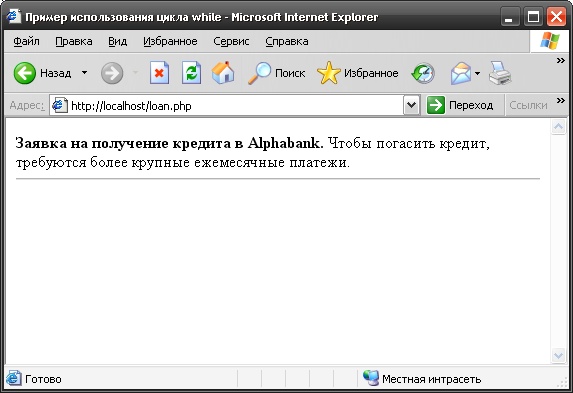


Рисунок 28.3 – Сообщение, выводимое при бесконечном цикле

**2. Материалы и оборудование**

1. Компьютер.
2. Программа Денвер.

**3. Задания к лабораторной работе**

1. Изучить теоретическую часть к лабораторной работе, изложенную в методических указаниях.

2. Ответить на контрольные вопросы к лабораторной работе, использовать лекции и указанную в них литературу, теоретический материал к лабораторной работе.

3.Выполнить задание**,** согласно своему варианту, изложенное в методических указаниях. Варианты заданий к лабораторной работе № 25:

28.1. Изучить приведенные в теоретическом обосновании примеры.

28. 2.Изменить в примерах вводимую информацию (например, название страны).

28.3. Разработать форму с оператором SWITCH.

4. Оформить отчет по лабораторной работе (Приложение 1).

**4. Вопросы к лабораторной работе**

28.1. Как использовать оператор SWITCH?

28.2.охарактеризуйте операторы PHP, используемые при обработке данных пользователя из форм.

**5. Задания для самостоятельной работы**

28.1. Проанализируйте описание стандартных операторов PHP, используемых для создания циклов при обработке данных.

28.2. Разработать форму с оператором цикла WHILE.

# Лабораторная работа № 29. ИСПОЛЬЗОВАНИЕ СТАНДАРТНЫХ ОПЕРАТОРОВ ЯЗЫКА PHP ПРИ ОБРАБОТКЕ ДАННЫХ ПОЛЬЗОВАТЕЛЯ ИЗ ФОРМ: ИСПОЛЬЗОВАНИЕ ЦИКЛОВ

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

**1. Теоретическая часть**

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>Пример использования цикла do while</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">

Введите число:

<INPUT name="guess" type="text"><BR><BR>

<INPUT type="submit" value="Проверить число">

<BR>

</FORM>

</BODY>

</HTML>

Реализация данного кода представлена на рисунке 29.1.

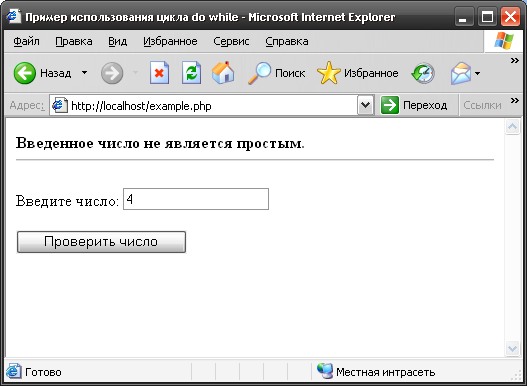


Рисунок 29.1 – Проверка простоты числа

Таким образом, как видно на рисунке 29.1, 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.

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>Пример использования цикла for</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>Введите имя $offset-гo 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">

Сколько у Вас детей?

<INPUT name="number" type="text"><BR><BR>

<INPUT type="submit" value="Отправитьчисло"><BR>

</FORM>

<?php } ?>

</BODY>

</HTML>

Результат работы данного примера представлен на рисунках 29.2, 29.3 и 29.4.

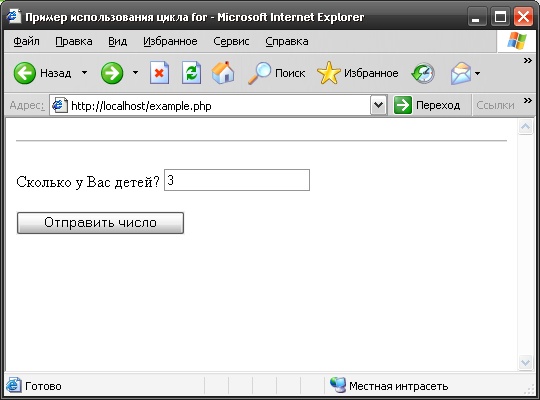


Рисунок 29.2 – Форма ввода количества детей

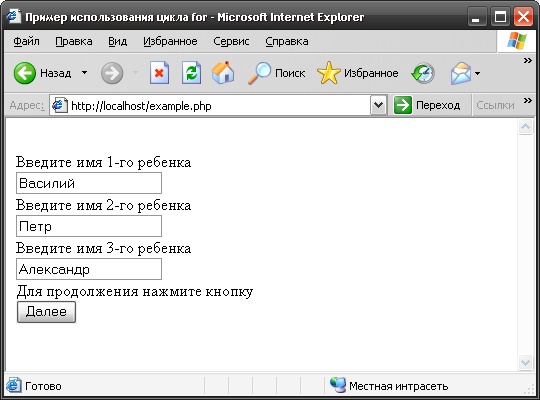


Рисунок 29.3 – Динамическая форма ввода имён детей

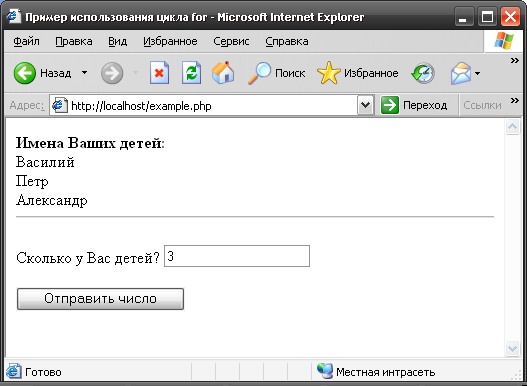


Рисунок 29.4 – Результат ввод имён детей

Таким образом, как видно на рисунке 29.2, 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 29.3). Однако если у пользователя нет детей, то отображаться текстовые поля не будут. В результате программа выводит имена детей на экран.

Как видно из рисунков 29.2 - 29.4, 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.

**2. Материалы и оборудование**

1. Компьютер.
2. Программа MS Денвер.

**3. Задания к лабораторной работе**

1. Изучить теоретическую часть к лабораторной работе, изложенную в методических указаниях.

2. Ответить на контрольные вопросы к лабораторной работе, использовать лекции и указанную в них литературу, теоретический материал к лабораторной работе.

3.Выполнить задание**,** согласно своему варианту, изложенное в методических указаниях. Варианты заданий к лабораторной работе № 26:

29.1. Изучить приведенные в теоретическом обосновании примеры.

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

29.3. Разработать форму с применением операторов цикла.

4. Оформить отчет по лабораторной работе (Приложение 1).

**4. Вопросы к лабораторной работе**

29.1. Каким образом используются циклы при работе с формами?

29.2. Примеры использования операторов цикла.

**5. Задания для самостоятельной работы**

29.1. Проанализируйте возможности операторов DO WHILE и FOR при обработке данных.

29.2.Создать формы с использованием операторов цикла DO WHILE и FOR.

### Лабораторная работа №30. ПРЕДОТВРАЩЕНИЕ КАТАСТРОФ И ВОССТАНОВЛЕНИЕ

**Цель работы:** научиться использовать инструменты предотвращения катастроф с БД

**1. Теоретическая часть**

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 или myisamchk -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:

Из клиента (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 (30.1) со следующими столбцами:

Таблица 30.1 – 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 19.219.25 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 (30.2) со следующими столбцами:

Таблица 30.2 – Результат выполнения RESTORE TABLE

|  |  |
| --- | --- |
| **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 (30.3) со следующими столбцами:

Таблица 30.3 – 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 таблице 30.4.

Таблица 30.4 – 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 перед CHANGED (иначе надо поступать только в случае, когда возникает подозрение, что найдена ошибка в самом коде MyISAM).

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\_INCREMENT имеет 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.

5 Синтаксис REPAIR TABLE

REPAIR TABLE tbl\_name[,tbl\_name...] [QUICK] [EXTENDED] [USE\_FRM]

REPAIR TABLE работает только на таблицах типа MyISAM и эквивалентна выполнению на таблице myisamchk -r table\_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 (30.5) со следующими столбцами:

Таблица 30.5 – 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 опции myisamchk. 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.

**2. Материалы и оборудование**

1. Компьютер.
2. Программа Денвер.

**3. Задания к лабораторной работе**

1. Изучить теоретическую часть к лабораторной работе, изложенную в методических указаниях.

2. Ответить на контрольные вопросы к лабораторной работе, использовать лекции и указанную в них литературу, теоретический материал к лабораторной работе.

3.Выполнить задание**,** согласно своему варианту, изложенное в методических указаниях. Варианты заданий к лабораторной работе № 27:

30.1. Изучить приведенные в теоретическом обосновании примеры.

30.2. полное резервное копирование ранее разработанной базы данных.

30.3.Выполнить проверку ранее разработанных таблиц с помощью оператора CHECK TABLE.

4. Оформить отчет по лабораторной работе (Приложение 1).

**4. Вопросы к лабораторной работе**

30.1. Для чего используется оператор LOCK TABLES/UNLOCK TABLES?

30.2. Для чего используется BACKUP TABLE?

30.3. Для чего используется RESTORE TABLE?

30.4. Для чего используется CHECK TABLE?

30.5. Для чего используется REPAIR TABLE?

30.6. Приведите пример использования средств резервного копирования.

**5. Задания для самостоятельной работы**

30.1. Проанализируйте использование операторов LOCK TABLES/UNLOCK TABLES, BACKUP TABLE, RESTORE TABLE, CHECK TABLE, REPAIR TABLE.

30.2.Создать формы с использованием операторов LOCK TABLES/UNLOCK TABLES, BACKUP TABLE, RESTORE TABLE, CHECK TABLE, REPAIR TABLE.if.

# Лабораторная работа № 31. [РЕПЛИКАЦИЯ В MySQL](http://www.php.su/mysql/manual/?page=Replication)

**Цель работы:** научиться настраивать репликацию в MySQL

**1. Теоретическая часть**

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 19.219.15, 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 19.23, но подчиненные серверы версии 4.0.1 и более поздних - могут. Подчиненный сервер версии 19.23 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 19.23 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 19.219.29, 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 переменная Slave\_open\_temp\_tables содержала значение 0, затем выполните mysqladmin shutdown. Если значение переменной Slave\_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 19.219.26, 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 19.219.26 изменился формат журнала. Таким образом, если версия подчиненного сервера меньше 19.219.26, 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 19.219.16). При использовании версии 19.219.15 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-connect-retry 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 19.219.15 все таблицы и базы данных могут быть реплицированы. Начиная с версии 19.219.16, появилась возможность ограничить репликацию набором баз данных при помощи директив replicate-do-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 19.219.23, имелась ошибка, из-за которой команда LOAD DATA INFILE выполнялась некорректно, если она применялась к базе данных, исключенной из репликации.

Начиная с версии 19.219.16, 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 19.219.19 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 19.219.26 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 19.219.23 можно заменять головные серверы и корректировать точку положения в журнале репликации при помощи команды CHANGE MASTER TO.

Начиная с версии 19.219.23 можно при помощи опции binlog-ignore-db уведомлять головной сервер о том, что обновления в некоторых базах данных не должны отражаться в двоичном журнале.

Начиная с версии 19.219.26, можно использовать опцию replicate-rewrite-db 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 19.219.28 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 19.219.47, некоторые (или все) ошибки могут быть замаскированы при помощи опции slave-skip-errors.

Отдельные таблицы могут исключаться из репликации при помощи опции replicate-do-table/replicate-ignore-tab или опции 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 19.219.49. 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 19.219. 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.

Пример репликации:

1. Запускаем головной сервер и подключаемся к нему. Для подключения используем программу Devart dbForge Studio for MySQL, а не интерфейс phpMyAdmin. Этапы выполнения работы показаны на рисунках 31.1-31.23.

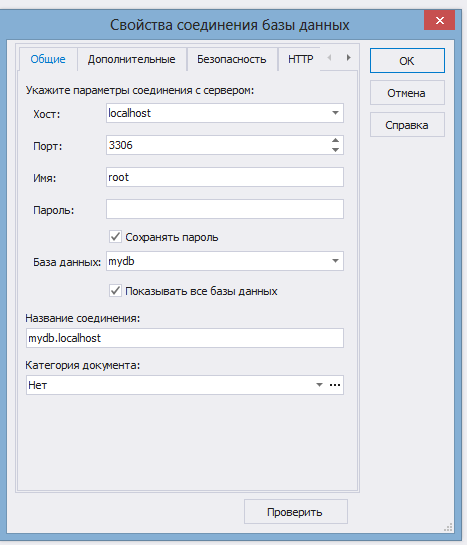


Рисунок 31.1 – Этап 1

1. Создаем тестовую базу данных «mydb».
2. Создаем в базе данных тестовую таблицу «mytable» с двумя полями – числовым (уникальным) и текстовым. Добавляем новую строку с текстом, например, «OldData», рисунок 31.2.

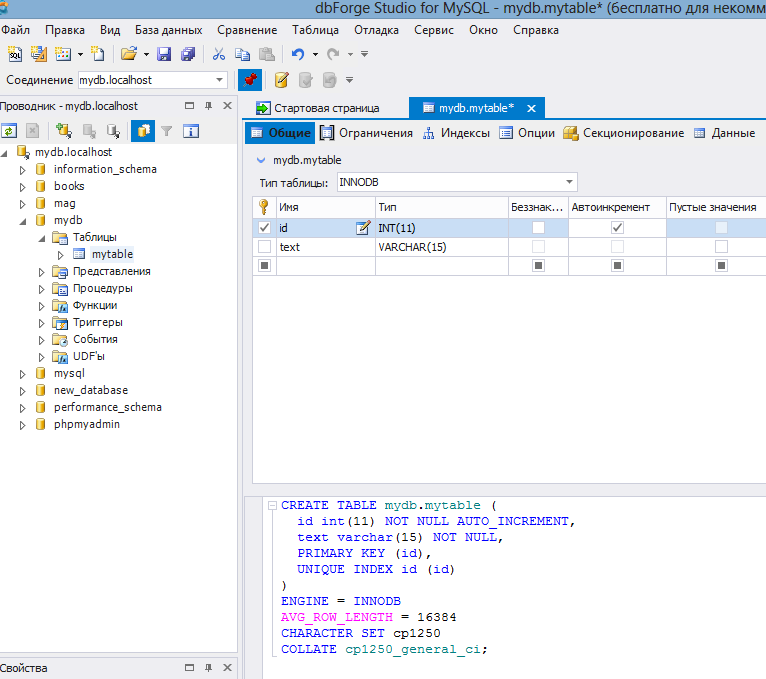


Рисунок 31.2 – Этап 3

1. Делаем резервную копию БД, рисунок 31.3. Не забываем поставить галочку «Включать выражение CREATE DATABASE».

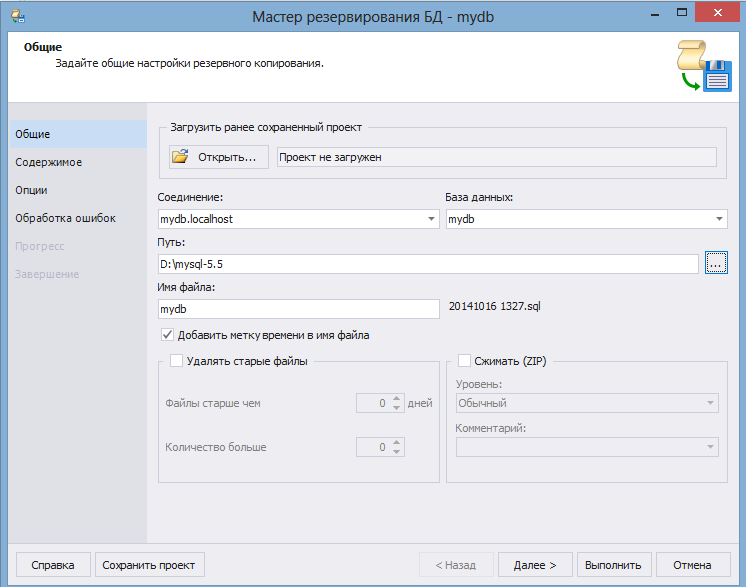
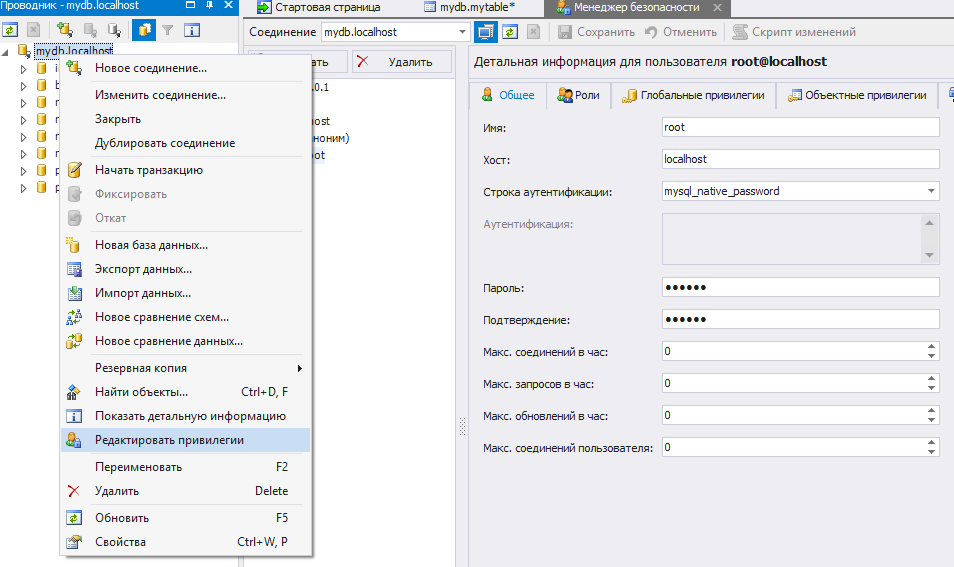


Рисунок 31.3 – Этап 4

1. Создаем пользователя для репликации «RepUser» с паролем «reppass» ( рисунок 31.4) и даем ему привилегию Replication Slave, рисунок 31.5.



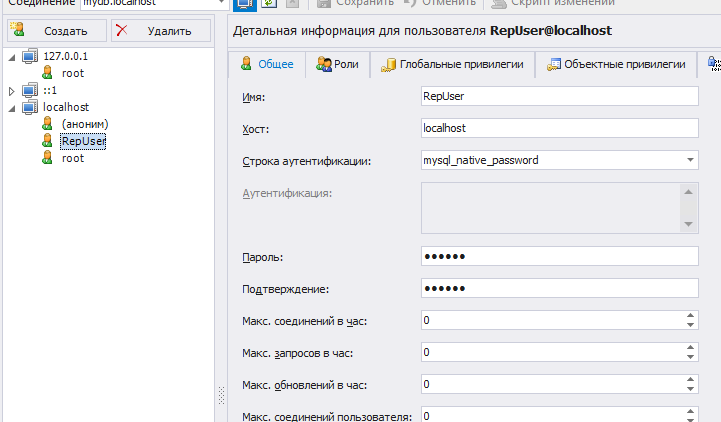


Рисунок 31.4 – Создание пользователя с паролем

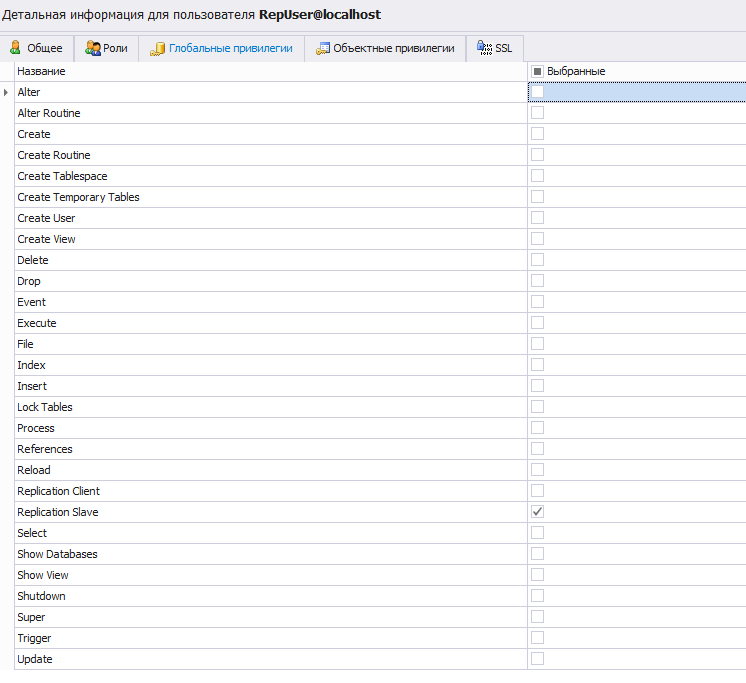


Рисунок 31.5 – Этап 5

1. Останавливаем головной сервер.
2. Запускаем подчиненный сервер и подключаемся к нему. Запускаем файл StartMySQL, рисунок 31.6.

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.

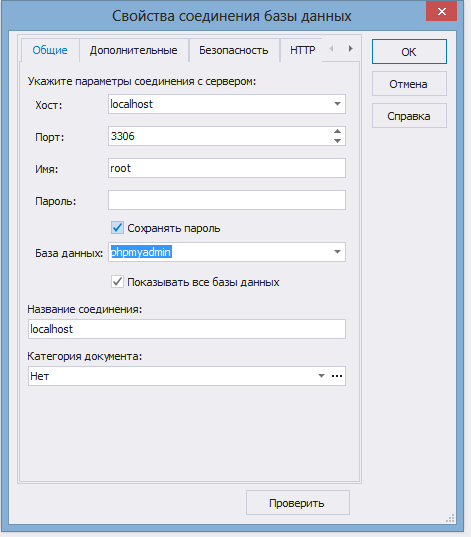
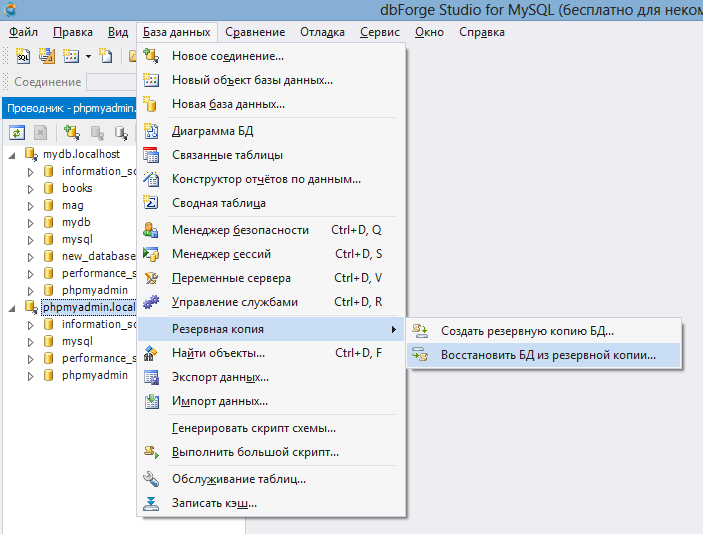


Рисунок 31.6 – Этап 7

1. Восстанавливаем базу данных «mydb» из архива, рисунок 31.14.



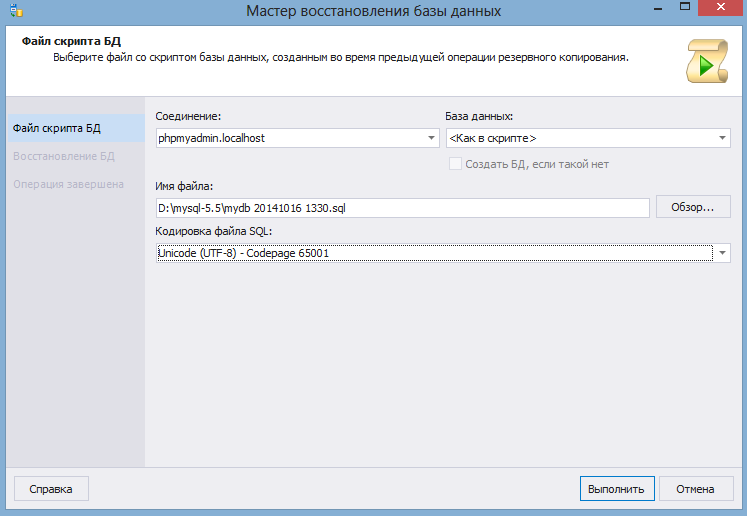


Рисунок 31.7 – Этап 8

1. Останавливаем подчиненный сервер (StopMySQL).
2. 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.

1. 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

1. Запускаем головной сервер
2. Запускаем подчиненный сервер
3. Запускаем командную строку и переходим в папку bin подчиненного сервера
4. Запускаем из командной строки утилиту mysql.exe с параметрами порта (3307), хоста (1214.0.0.1), и пользователя (root). Таким образом, мы подключимся к подчиненному серверу, рисунок 31.8.

*mysql.exe --host=1214.0.0.1 --port=3307 --user=root*

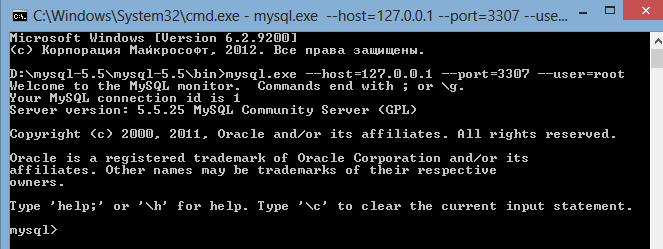


Рисунок 31.8 – Этап 15

1. 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=’1214.0.0.1’, MASTER\_PORT=3306, MASTER\_USER=’RepUser’, MASTER\_PASSWORD=’reppass’;*

Команду не обязательно вводить одной строкой. Командный интерпретатор определяет окончание ввода по символу «;», рисунок 31.9.

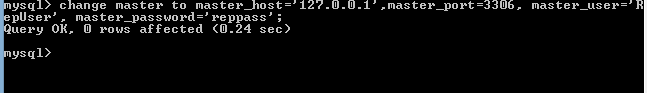


Рисунок 31.9 – Этап 16

1. Запускаем репликацию командой

*START SLAVE;*

1. Проверяем состояние репликации (рисунок 31.10)

*SHOW SLAVE STATUS \G*

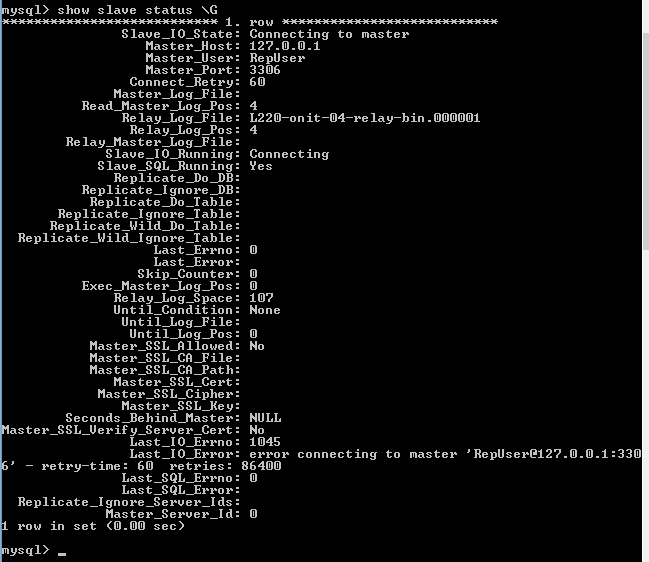


Рисунок 31.10 – Этап 18

1. Смотрим список баз данных, рисунок 31.11.

*SHOW DATABASES;*

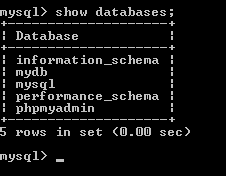


Рисунок 31.11 – Этап 19

1. Не закрывая консоль подключаемся к головному серверу и cоздаем в головном сервере новую базу данных «newdb»
2. Добавляем в старую базу данных «mydb» в таблицу «mytable» новую строку, например «NewData», рисунок 31.12.

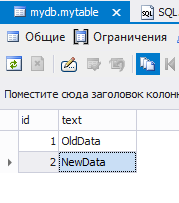


Рисунок 31.12 – Этап 21

1. В консоли, которую мы не закрыли, проверяем наличие новой базы данных. Для этого смотрим список баз данных, рисунок 31.13.

*SHOW DATABASES;*

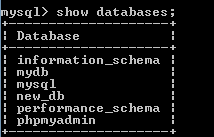


Рисунок 31.13 – Этап 22

Как мы видим новая база данных «newdb» появилась в списке

1. Проверяем наличие новой строки в старой базе данных «mydb».

Сначала подключаемся к базе, рисунок 31.14.

*USEmydb;*



Рисунок 31.14 – Этап 23, подключение к БД

Выводим список строк, рисунок 31.15:

*SELECT \* FROMmytable*



Рисунок 31.15 – Этап 23

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

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

*\q*

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

*Exit*

1. 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.

**2. Материалы и оборудование**

1. Компьютер.
2. Программа Денвер.

**3. Задания к лабораторной работе**

1. Изучить теоретическую часть к лабораторной работе, изложенную в методических указаниях.

2. Ответить на контрольные вопросы к лабораторной работе, использовать лекции и указанную в них литературу, теоретический материал к лабораторной работе.

3.Выполнить задание**,** согласно своему варианту, изложенное в методических указаниях. Варианты заданий к лабораторной работе № 31:

31.1. Выполнить [репликацию базы данных MySQL](http://www.php.su/mysql/manual/?page=Replication), разработанной в предыдущих лабораторных работах.

31.2.Проверить работоспособность базы данных из задания 31.1.

4. Оформить отчет по лабораторной работе (Приложение 1).

**4. Вопросы к лабораторной работе**

31.1. Каким образом осуществляется репликация?

31.2. Для чего применяется репликация? Пример использования.

**5. Задания для самостоятельной работы**

31.1. Проанализируйте средства СУБД для репликации данных.

31.2. Проанализируйте ошибки, возникающие при обмене данными между головным и подчиненным серверами.

# Лабораторная работа № 32. SQL-команды, относящиеся к репликации. Проблемы и распространённые ошибки MySQL

**Цель работы:** научиться управлять репликациями в MySQL.

**1. Теоретическая часть**

Управление репликацией производится командами SQL. В таблице 32.1 приводится краткое описание команд:

Таблица 32.1 – Команды SQL

|  |  |
| --- | --- |
| **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 привилегию SUPER. 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 19.219.26 используйте команду FLUSH SLAVE (головной сервер) |
| RESET SLAVE | Заставляет подчиненный сервер "забыть" свою точку положения репликации в журналах головного сервера. Для версий ниже 19.219.26 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 | Заменяет параметры головного сервера значениями, заданными в списке master\_def\_list, и перезапускает поток подчиненного сервера. master\_def\_list – это список с разделителем-запятой, содержащий значения master\_def, где master\_def – одно из следующих значений: 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. Можно выполнить команду  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 19.219.28. 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 19.219.28. 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. |

**Проблемы и распространённые ошибки MySQL**

[Как определить, чем вызваны проблемы](http://www.php.su/mysql/manual/" \l "What_is_crashing)? 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 32.2. Какой код из них вы получите, зависит от ОС.

Таблица 32.2 – 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 mysqld получает 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 Mб). 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.

**2. Материалы и оборудование**

1. Компьютер.
2. Программа Денвер.

**3. Задания к лабораторной работе**

1. Изучить теоретическую часть к лабораторной работе, изложенную в методических указаниях.

2. Ответить на контрольные вопросы к лабораторной работе, использовать лекции и указанную в них литературу, теоретический материал к лабораторной работе.

3.Выполнить задание**,** согласно своему варианту, изложенное в методических указаниях.

Варианты заданий к лабораторной работе № 29:

32.1. Изучить приведенные в теоретическом обосновании примеры.

32.1. Изучить назначение команд, относящихся к репликации БД.

32.3. Проанализировать распространённые ошибки MySQL.

4. Оформить отчет по лабораторной работе (Приложение 1).

**4. Вопросы к лабораторной работе**

32.1. Какие команды используются для репликации данных?

32.2. Какие проблемы могут возникнуть при работе с MySQL?

32.3. Пример использования репликации.

**5. Задания для самостоятельной работы**

32.1. Изучить распространённые ошибки и методы их решения.

32.2. Привести примеры устранения ошибок.

# СПИСОК РЕКОМЕНДУЕМОЙ ЛИТЕРАТУРЫ

***а) основная литература:***

1. Базы данных. В 2-х кн. Кн. 2. Распределенные и удаленные базы данных: Учебник / В.П. Агальцов. - М.: ИД ФОРУМ: НИЦ Инфра-М, 2019. - 272 с.: ил.; 60x90 1/16. - (Высшее образование). (переплет) ISBN 978-5-8199-0394-0, 2000 экз. <http://znanium.com/catalog.php?bookinfo=372740>

2. Мартишин С.А., Симонов В.Л., Храпченко М.В. Базы данных. Практическое применение СУБД SQL и NoSOL-типа для применения проектирования информационных систем // ИД ФОРУМ, 2018. 3. http://guardmag.com

3. Справочное руководство по MySQL. [Электронный ресурс]: <http://www.php.su/mysql/manual/>

4. Защита информации в базах данных - <http://bourabai.ru/dbt/dbms/13.htm>

***б) дополнительная литература:***

1. Новиков Б.А., Горшкова Е.А. Основы технологии баз данных. Учебное пособие/ Изд-во ДМК Пресс, 2019, 240 с. ISBN 9785940748205

2. Карпова, Т.С. Базы данных: модели, разработка, реализация: учебное пособие / Т.С. Карпова. - 2-е изд., исправ. - Москва: Национальный Открытый Университет «ИНТУИТ», 2016. - 241 с.: ил.; То же [Электронный ресурс]. - URL: http://biblioclub.ru/index.php?page=book&id=429003.

# Приложение 1

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| |  |  |  |  | | --- | --- | --- | --- | | |  | | --- | |  | | МИНОБРНАУКИ РОССИИ | | Федеральное государственное бюджетное образовательное учреждение  высшего образования  **«МИРЭА – Российский технологический университет»**  **РТУ МИРЭА**  **Филиал РТУ МИРЭА в г. Ставрополе** | | | | |
|  | | |
| Кафедра \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  *(название кафедры* *полностью)* | | |

|  |  |
| --- | --- |
| **ОТЧЕТ ПО ЛАБОРАТОРНЫМ РАБОТАМ** | |
| **по дисциплине** | |
| **«**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**»**  *(наименование дисциплины)* | |
| Выполнил студент группы \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  *(учебная группа)* | *Фамилия И.О.* |
| Принял \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  *должность, звание, ученая степень* | *Фамилия И.О.* |
|  |  |

|  |  |  |
| --- | --- | --- |
| Лабораторные работы выполнены | «\_\_»\_\_\_\_\_\_\_201\_\_ г. | *(подпись студента)* |
|  |  |  |
| «Зачтено» | «\_\_»\_\_\_\_\_\_\_201\_\_ г. | *(подпись руководителя)* |

Ставрополь 20\_\_

**А.В.Горелова, В.И.Дроздова**

**Методические указания**

**к лабораторным работам по дисциплине**

**"Базы данных"**

**для студентов направления**

**«Информатика и вычислительная техника»**