HTML-код программы проверки введенных данных представлен ниже.

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

На рисунке 13.4 представлен результат работы данного кода.

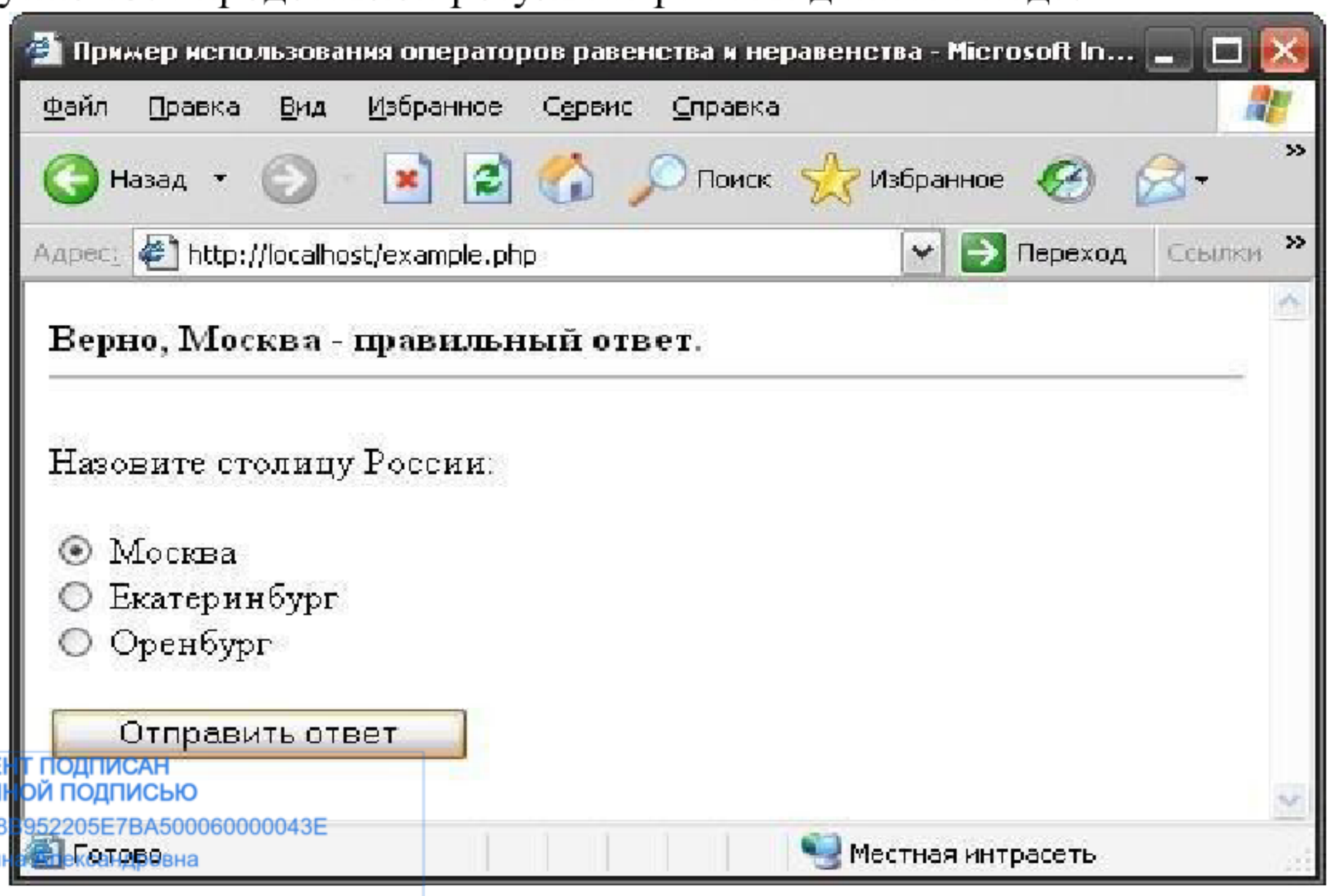

Рисунок13.4 – Проверка на правильность выбора столицы России Таким образом, как видно на рисунке 13.4, в данном примере, при загрузке Web-страницы появляется форма, в которой пользователю необходимо выбрать один из трех вариантов ответа и нажать кнопку “Отправить ответ”. После этого введенный ответ сравнивается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

3 Логические операторы

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

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

HTML -код программы проверки информации пользователя представлен ниже.

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

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

Результат работы данного кода представлен на рисунках 13.5 и 13.6.

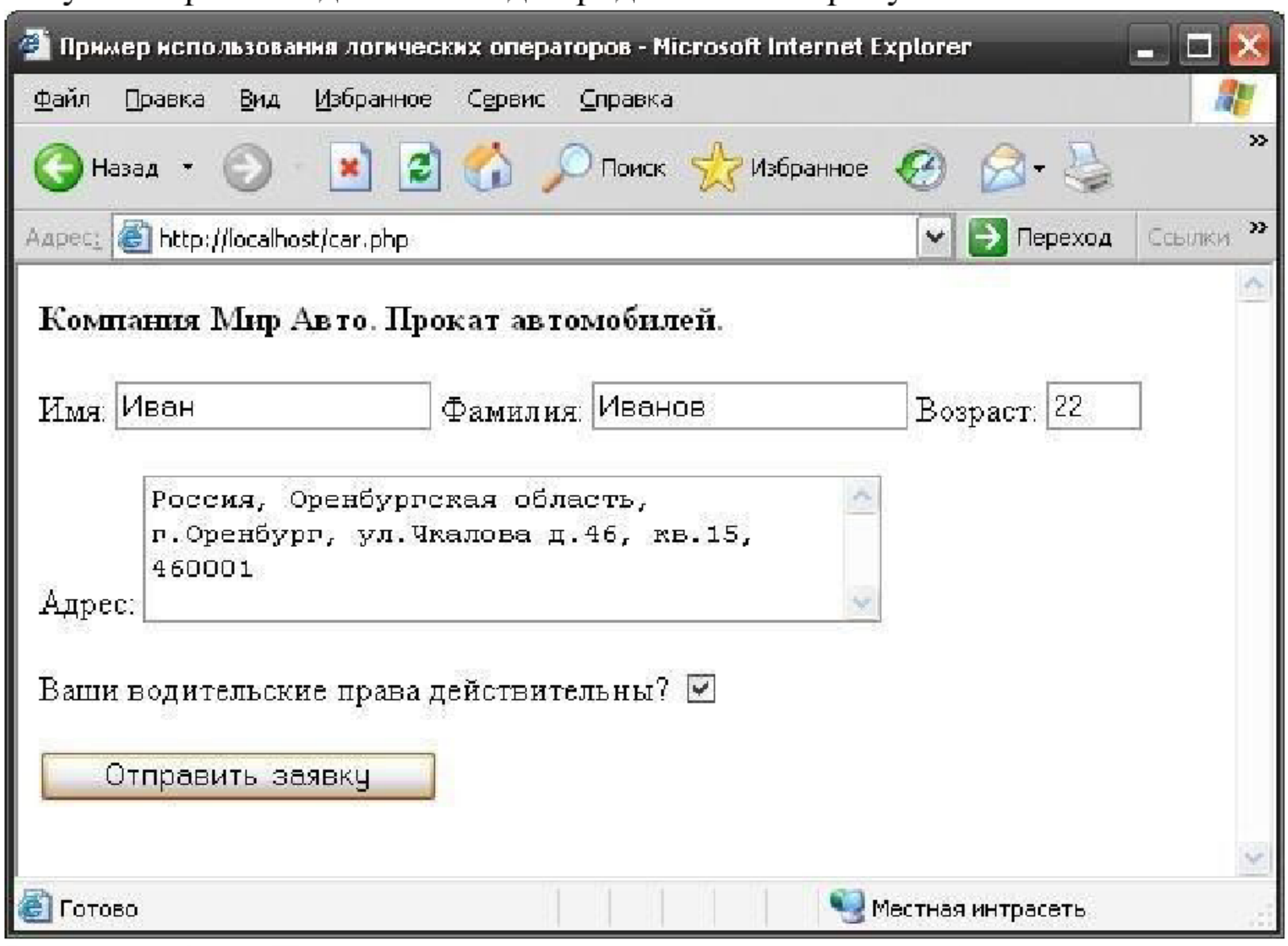

Рисунок 13.5 – Проверка возможности выдачи автомобилей

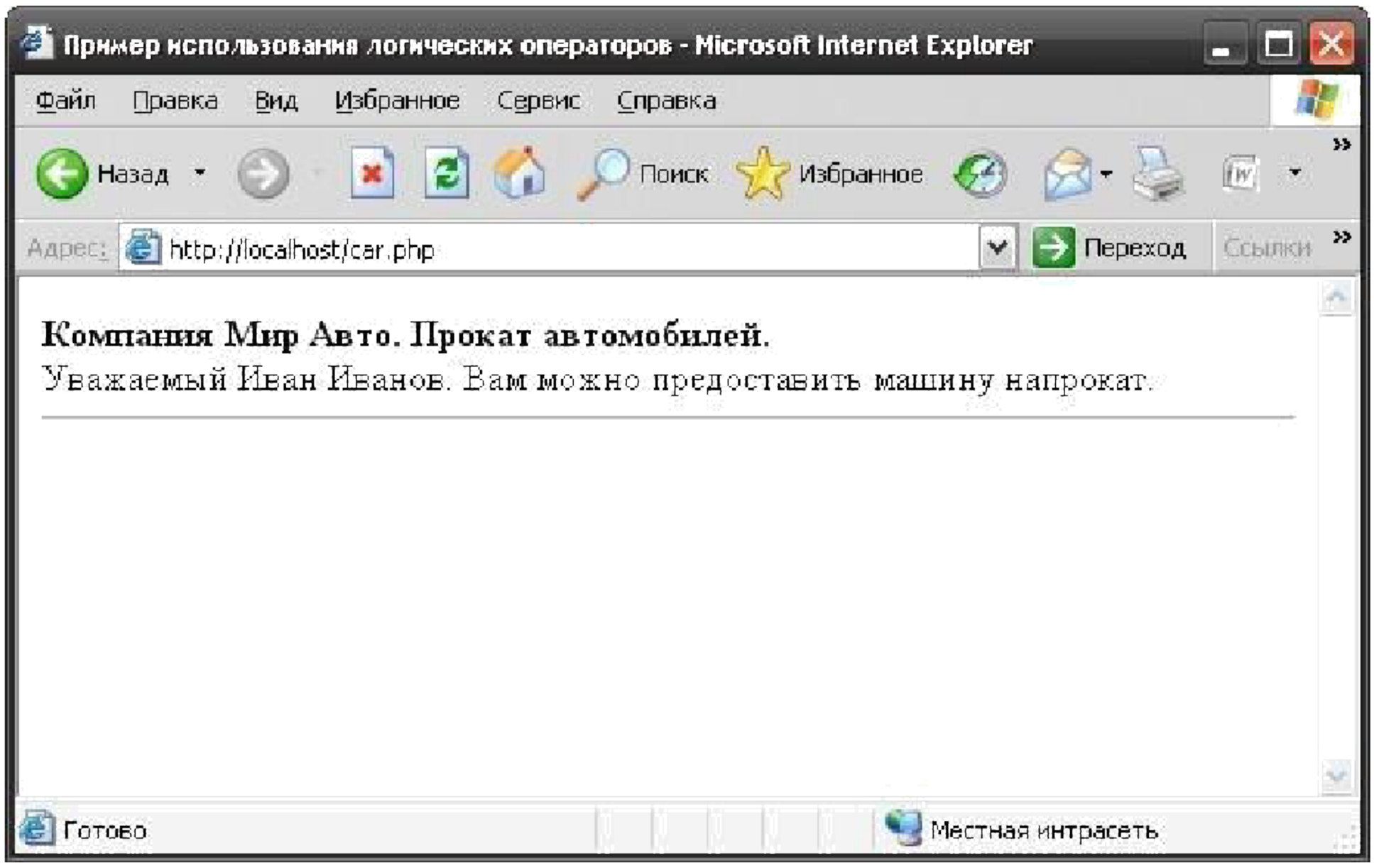

Рисунок 13.6 – Результат проверки

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

Задание

Изучить приведенные в теоретическом обосновании примеры.

Содержание отчета и его форма

Отчет по лабораторной работе должен состоять из:

названия лабораторной работы;

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

Вопросы для защиты работы:

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

Как использовать булевые операторы?

Как использовать операторы сравнения?

Как использовать логические операторы?

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

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

Теоретическое обоснование

1 Оператор SWITCH

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

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

HTML-код программы расчета стоимости путевок представлен ниже.

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

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

Реализация данного кода представлена на рисунке 14.1.

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

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

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

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

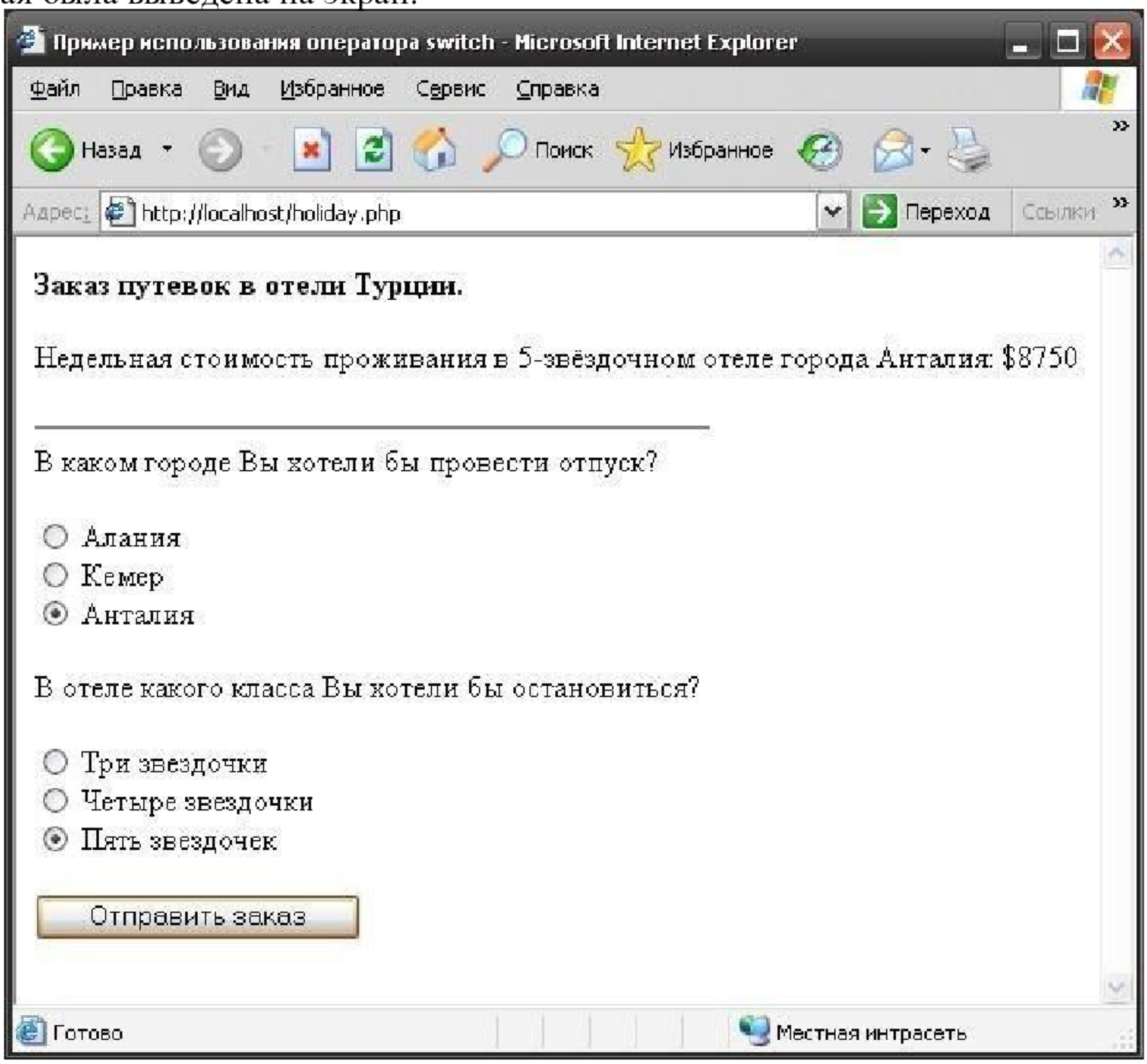

Рисунок 14.1 – Выбор города и класса отеля при заказе путевок

2 Использование циклов при обработке данных пользователя из форм

2.1 Цикл WHILE

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

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

HTML-код программы расчета срока погашения кредита представлен ниже.

```
<HTML>
<HEAD>
<TITLE>Примериспользованияциклawhile</TITLE>
</HEAD>
<BODY>
<B>Заявка на получение кредита Alphabank.</B>
<?php if (isset($_POST['posted'] )) {
$duration = 0; switch ($_POST['loan']) { case "1000"; $interest=8; break;
case "5000"; $interest=11,5; break; case "10000"; $interest=15,0; break; default:
echo "Выневыбралисуммукредита<HR>"; exit; } while ($_POST['loan'] >
0)
{
$duration = $duration + 1;
$monthly = $_POST['month'] - ($_POST['loan']*$interest/100); if ($monthly
<= 0)
{
echo "Чтобы погасить кредит, требуются более крупные
ежемесячныеплатежи<HR>";
exit; }
$_POST['loan'] = $_POST['loan'] - $monthly;
}
echo "Для погашения кредита при процентной ставке $interest
% понадобится $duration месяцев.<HR>"; } ?>

<FORM method="POST" action="loan.php">
<INPUT type="hidden" name="posted"
value="true"><BR> Выберите сумму необходимого
кредита: <BR>
<INPUT name="loan" type="radio"
value="1000"> $1000 под 8,0% годовых<BR>
<INPUT name="loan" type="radio"
value="5000"> $5000 под 11,5% годовых<BR>
<INPUT name="loan" type="radio" value="10000">
$10000 под 15,0% годовых <BR><BR>
Введите сумму ежемесячного платежа в долларах:
<INPUT name="month" type="text" size="5"><BR>
<INPUT type="submit" value="Податьзаявку">
```

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

Реализация данного кода представлена на рисунке 14.2.

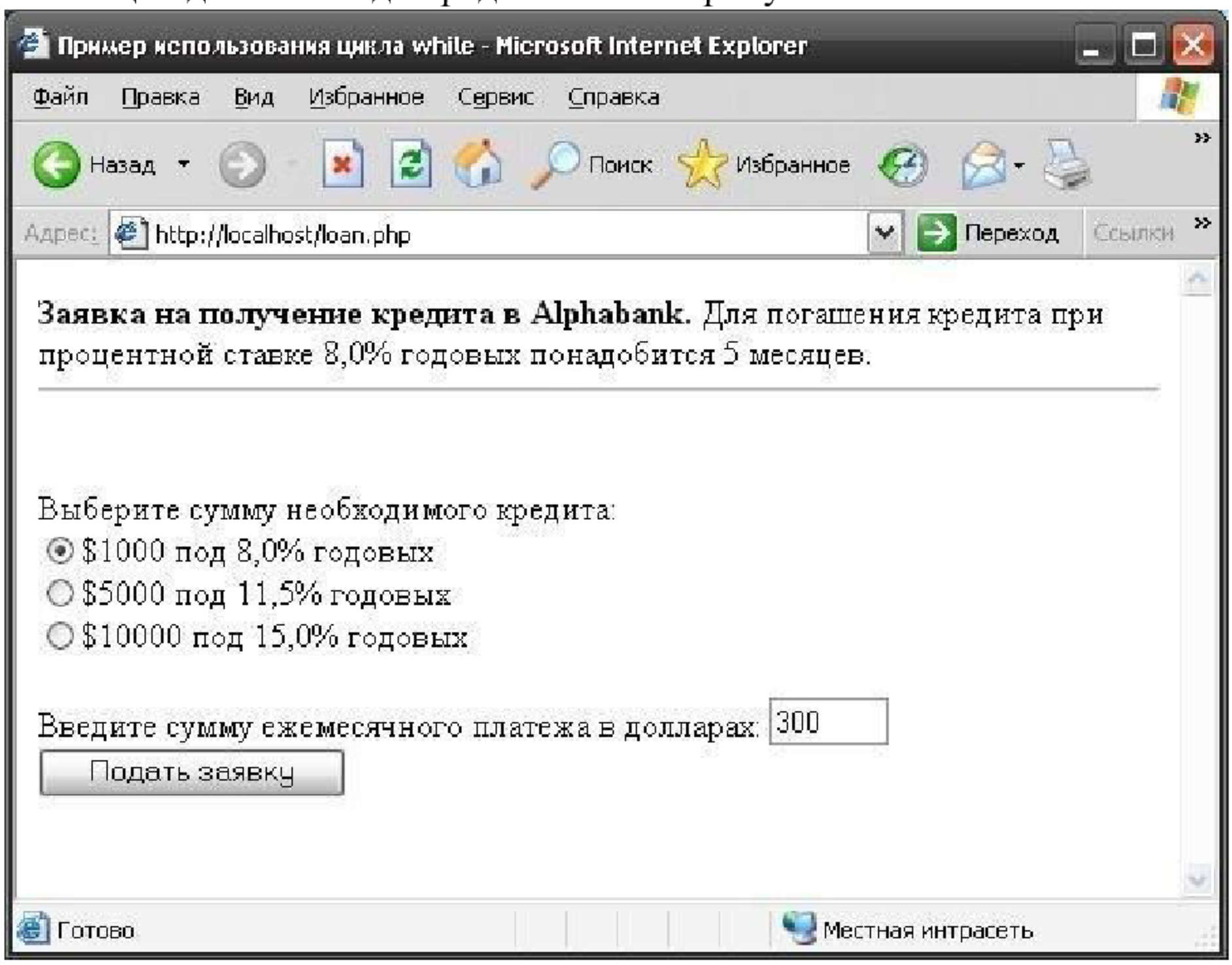

Рисунок 14.2 – Расчет времени погашения кредита в месяцах

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

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

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

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

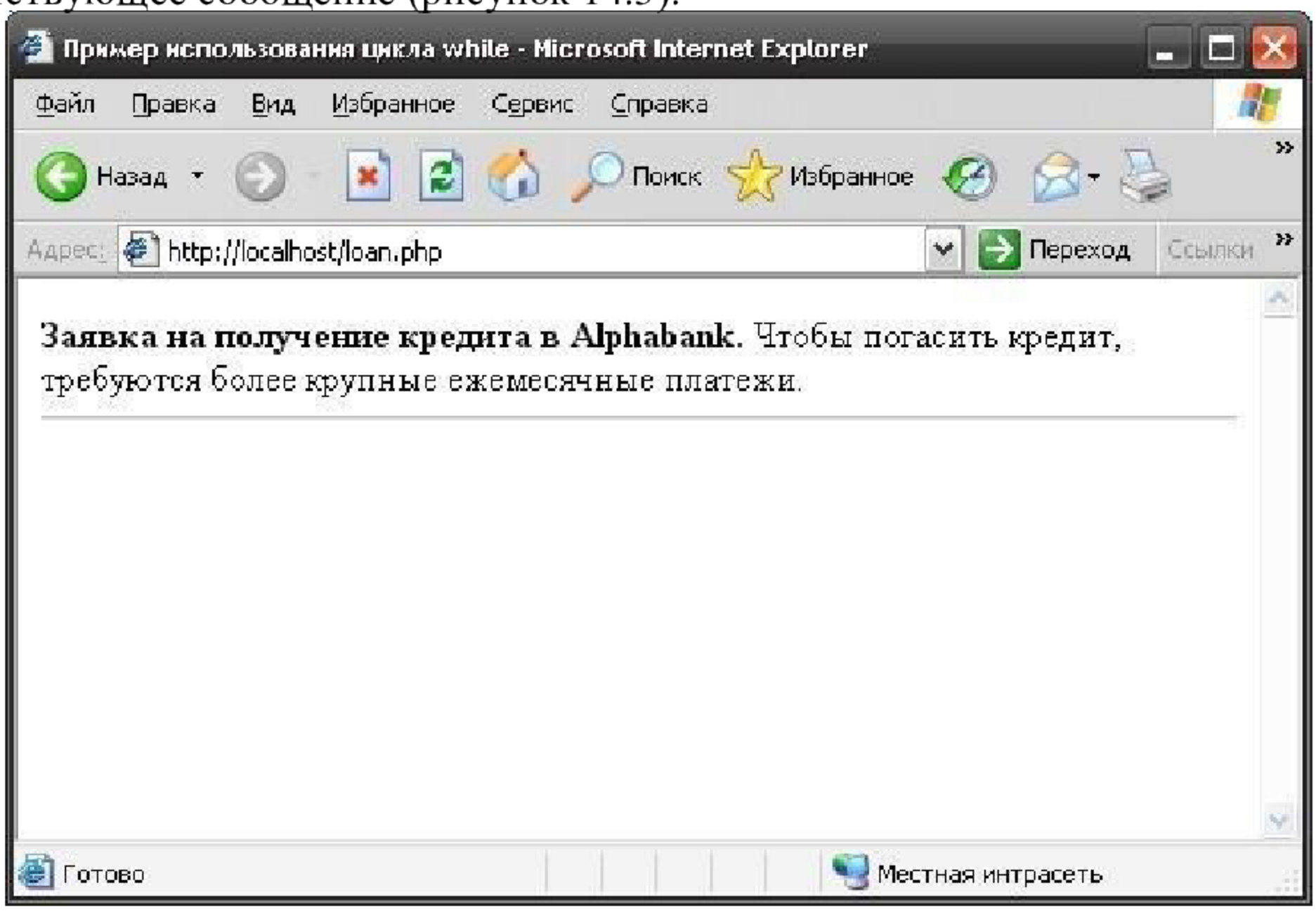

Рисунок 14.3 – Сообщение, выводимое при бесконечном цикле

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

2.2 Цикл DO WHILE

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

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

HTML-код программы проверки вводимого числа представлен ниже.

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

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

Реализация данного кода представлена на рисунке 14.4.

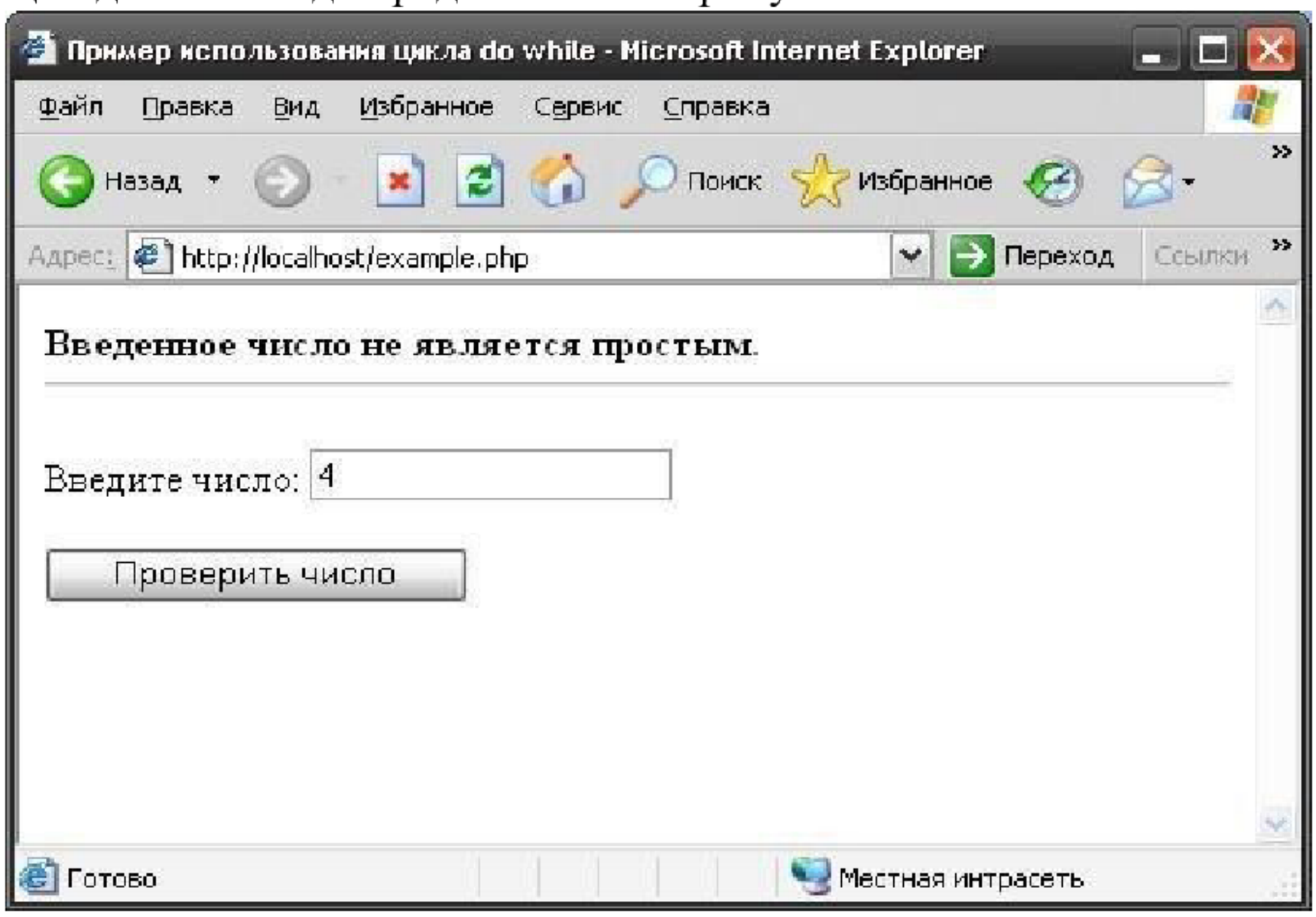

Рисунок 14.4 – Проверка простоты числа

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

2.3 Цикл FOR

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

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

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

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

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

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

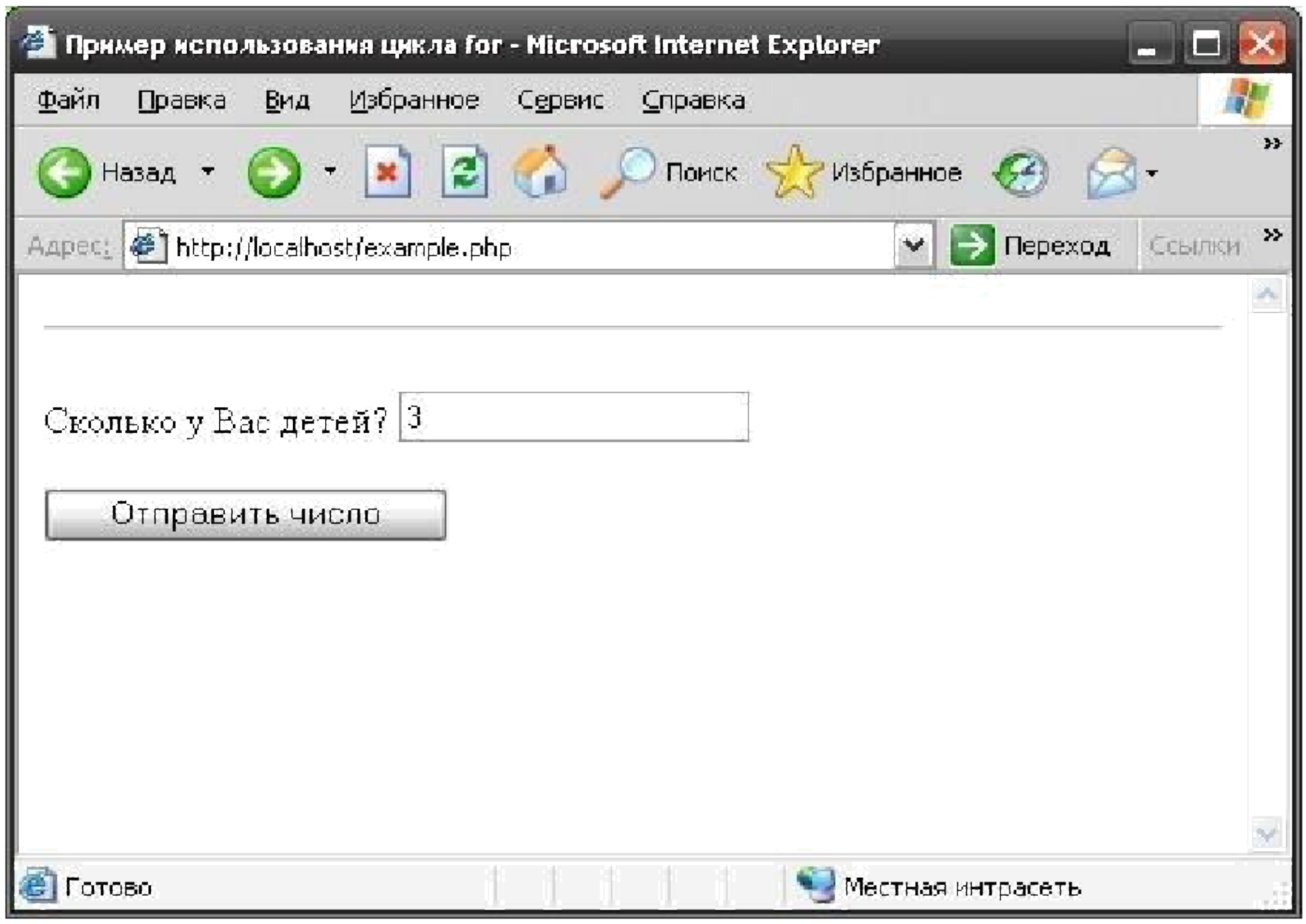

Рисунок 14.5 – Форма ввода количества детей

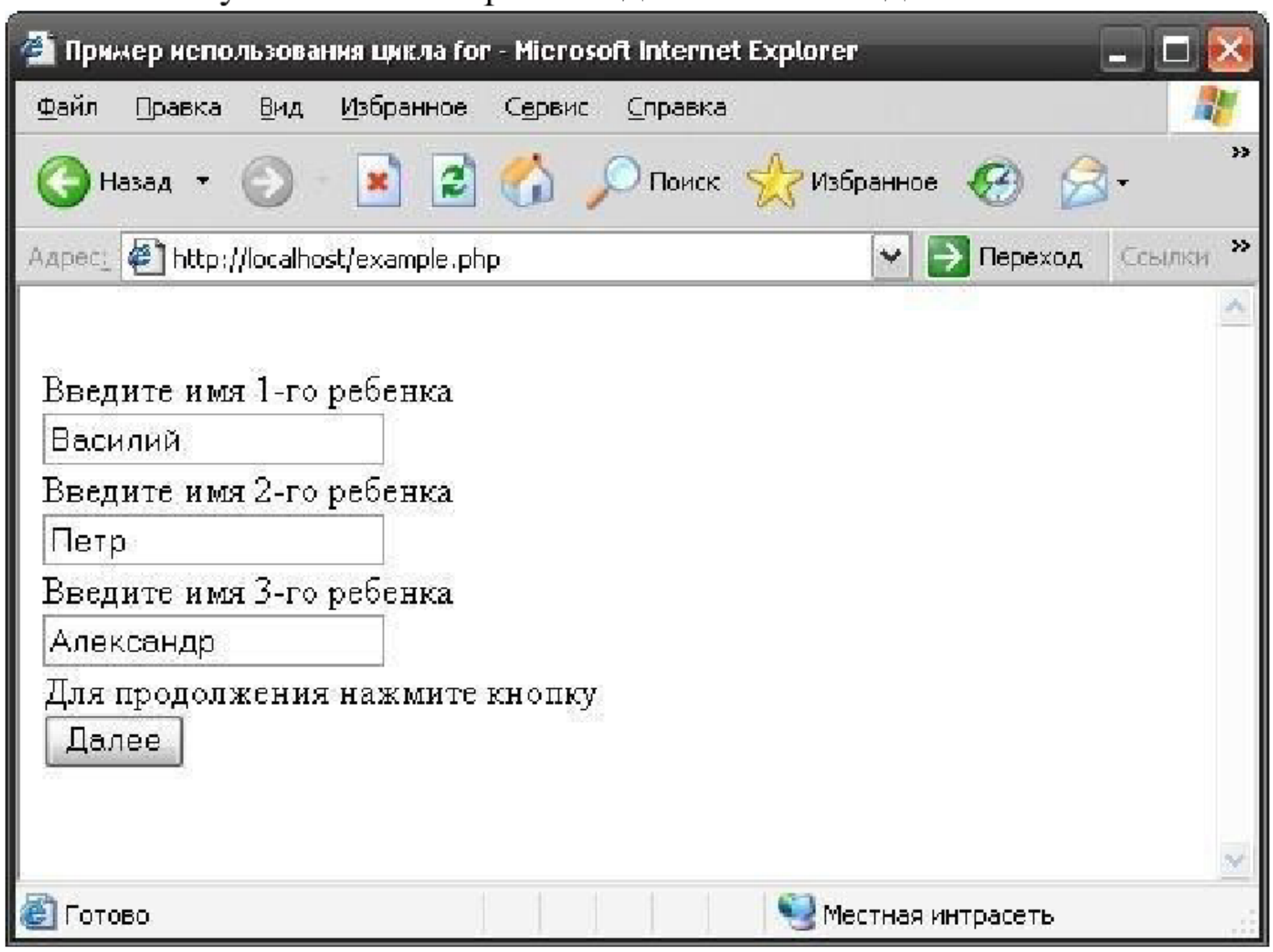

Рисунок 14.6 – Динамическая форма ввода имён детей

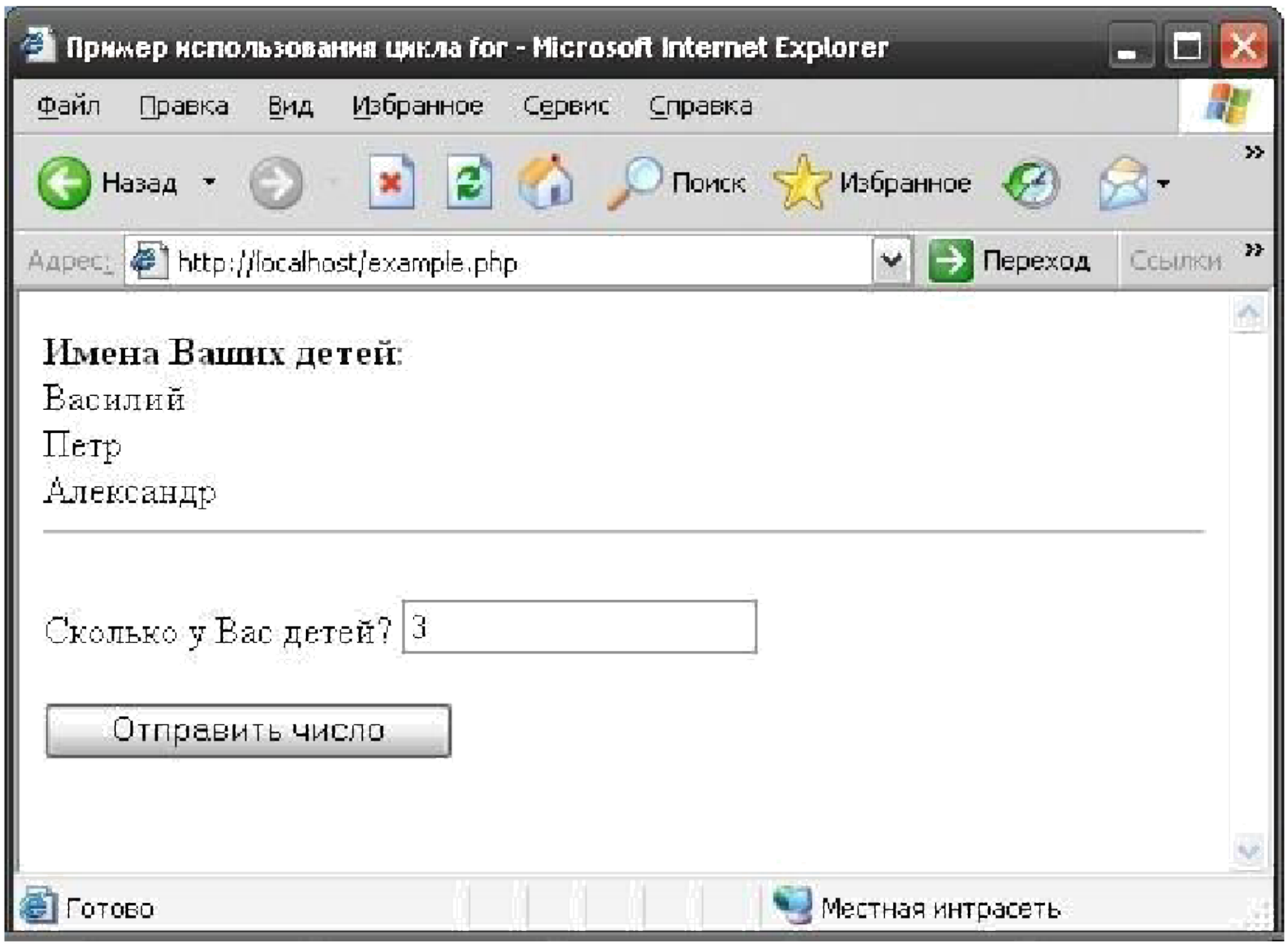

Рисунок 14.7 – Результат ввод имён детей

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

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

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

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

Задание

Изучить примеры, приведенные в теоретическом обосновании.

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

Содержание отчета и его форма

Отчет по лабораторной работе должен состоять из:

названия лабораторной работы;

описание примеров использования оператора SWITCH при работе с формами;

описание примеров использования циклов.

Вопросы для защиты работы:

Как использовать оператор SWITCH?

Каким образом используются циклы при работе с формами?

Лабораторная работа №15. Предотвращение катастроф и восстановление

Цель работы: научиться использовать инструменты предотвращения катастроф с БД

Теоретическое

обоснование 1 Резервное копирование баз данных

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

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

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

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

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

или

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

Можно также просто скопировать табличные файлы

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

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

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

Восстановите исходный вариант по копии, сделанной в mysqldump.

Выполните следующую команду, чтобы повторить обновления из бинарного журнала:

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

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

ls используется для того, чтобы расположить все файлы

журнала обновлений в правильном порядке.

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

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

Пользователи файловой системы Veritas могут поступить следующим образом:

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

Из другого shell выполнить: mount vxfs snapshot.

Из первого клиента выполнить: UNLOCK TABLES.

Скопировать файлы из образа.

Демонтировать образ.

2 Синтаксис BACKUP TABLE

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

Копирует в каталог резервного копирования тот минимум табличных файлов, который достаточен для восстановления таблицы. На данный момент работает только для таблиц MyISAM. Для таблиц MyISAM копирует файлы

'.frm'(определений) и '.MYD' (данных). Индексные файлы могут быть реконструированы по этим двум.

процессе резервного копирования будет установлена блокировка чтения отдельно для каждой таблицы на время ее копирования. Если необходимо сделать резервное копирование в виде мгновенного образа нескольких таблиц, необходимо сначала запросить LOCK

TABLES установки блокировки чтения для каждой таблицы в группе. Команда возвращает таблицу (15.1) со следующими столбцами: Таблица 15.1 – Результат выполнения блокировки

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

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

3 Синтаксис RESTORE TABLE

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

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

Команда возвращает таблицу (15.2) со следующими столбцами:

Таблица 15.2 – Результат выполнения RESTORETABLE

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

4 Синтаксис CHECK TABLE

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

option = QUICK | FAST | MEDIUM | EXTENDED | CHANGED

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

Если опция не указана, используется MEDIUM.

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

Таблица 15.3 – Результат выполнения CHECK TABLE

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

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

Различные типы проверки представлены в таблице15.4.

Таблица 15.4 – Операторы, используемые для выполнения проверки

Тип	Действия
QUICK	Не сканировать строки для проверки на неправильные связи.
FAST	Проверять только таблицы, которые не были корректно закрыты.
CHANGED	Проверять только таблицы, которые изменились со времени последней проверки или не были закрыты корректно.
MEDIUM	Сканировать строки для проверки того, что уничтоженные связи в порядке. При этом также подсчитывается ключевая контрольная сумма для строки и сравнивается с подсчитанной контрольной суммой для ключей.

EXTENDED	Выполнить полный просмотр ключа для всех ключей для каждой строки. Успех такой проверки гарантирует 100%
	отсутствие противоречий в таблице, но на проверку уйдет немало времени!

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

Проверочные опции можно сочетать:

CHECK TABLE test_table FAST QUICK;

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

Примечание: в некоторых случаях CHECK TABLE изменяет таблицу!

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

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

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

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

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

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

Found row where the auto_increment column has the value 0. Это означает,

что в таблице есть строка, где индексированный столбец AUTO_INCREMENT содержит значение 0 (строку, в которой столбец AUTO_INCREMENTимеет значение 0, можно создать, явно установив столбец 0 командой UPDATE). Это само по себе не является ошибкой, но может вызвать неприятности, если понадобится сделать дамп таблицы или восстановить ее или выполнить над ней ALTER TABLE. В этом случае столбец с атрибутом AUTO_INCREMENT изменит значение в соответствии с правилами для столбцов AUTO_INCREMENT, что может вызвать проблемы, подобные ошибке дублирования ключа. Чтобы избавиться от предупреждения, просто выполните команду UPDATE для установки в столбце значения, отличного от 0.

5Синтаксис REPAIR TABLE

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

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

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

REPAIR TABLE ремонтирует таблицу, которая, возможно, повреждена. Команда возвращает таблицу (15.5) со следующими столбцами: Таблица 15.5 – Выполнение REPAIR TABLE

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

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

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

Если используется EXTENDED, то MySQL будет создавать индекс строка за строкой вместо создания по одному индексу единоразово с помощью сортировки; такая техника может работать лучше сортировки для ключей

фиксированной длины, если речь идет о хорошо сжимаемых ключах типа char() большой длины.

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

Задание

Изучить инструментарии предотвращения катастроф и резервного копирования.

Содержание отчета и его форма

Отчет по лабораторной работе должен состоять из:

названия лабораторной работы;

описание примеров использования LOCK TABLES/UNLOCK TABLES, BACKUPTABLE, RESTORETABLE, CHECKTABLE, REPAIR TABLE.

Вопросы для защиты работы:

Для чего используетсяоператорLOCKTABLES/UNLOCKTABLES?
Для чего используется BACKUPTABLE?
Для чего используется RESTORETABLE?
Для чего используется CHECKTABLE?
Для чего используетсяREPAIRTABLE?

Лабораторная работа №16.Репликация в MySQL

Цель работы: научиться настраивать репликацию вMySQL

Теоретическое обоснование

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

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

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

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

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

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

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

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

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

блокирования обновлений на головном сервере.

Из-за вышеупомянутого ограничения рекомендуется использовать команду LOAD DATA FROM MASTER только в тех случаях, если набор данных на головном сервере относительно невелик или если для головного сервера допустима длительная блокировка чтения. Скорость выполнения команды LOAD DATA FROM MASTER для разных систем может быть различной, поэтому для грубой оценки времени выполнения команды можно считать, что для передачи 1 Мб данных требуется 1 секунда. Это приблизительно соответствует случаю, когда и головной, и подчиненный серверы эквивалентны Pentium с тактовой частотой 700 МГц и связаны сетью 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.

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

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

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

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

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

нагрузки путем посылки запросов на выборки различным подчиненным серверам.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Отдельныетаблицымогутисключатьсяизрепликацииприпомощиопции r

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

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

Методика и порядок выполнения работы

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

16.1-16-23.

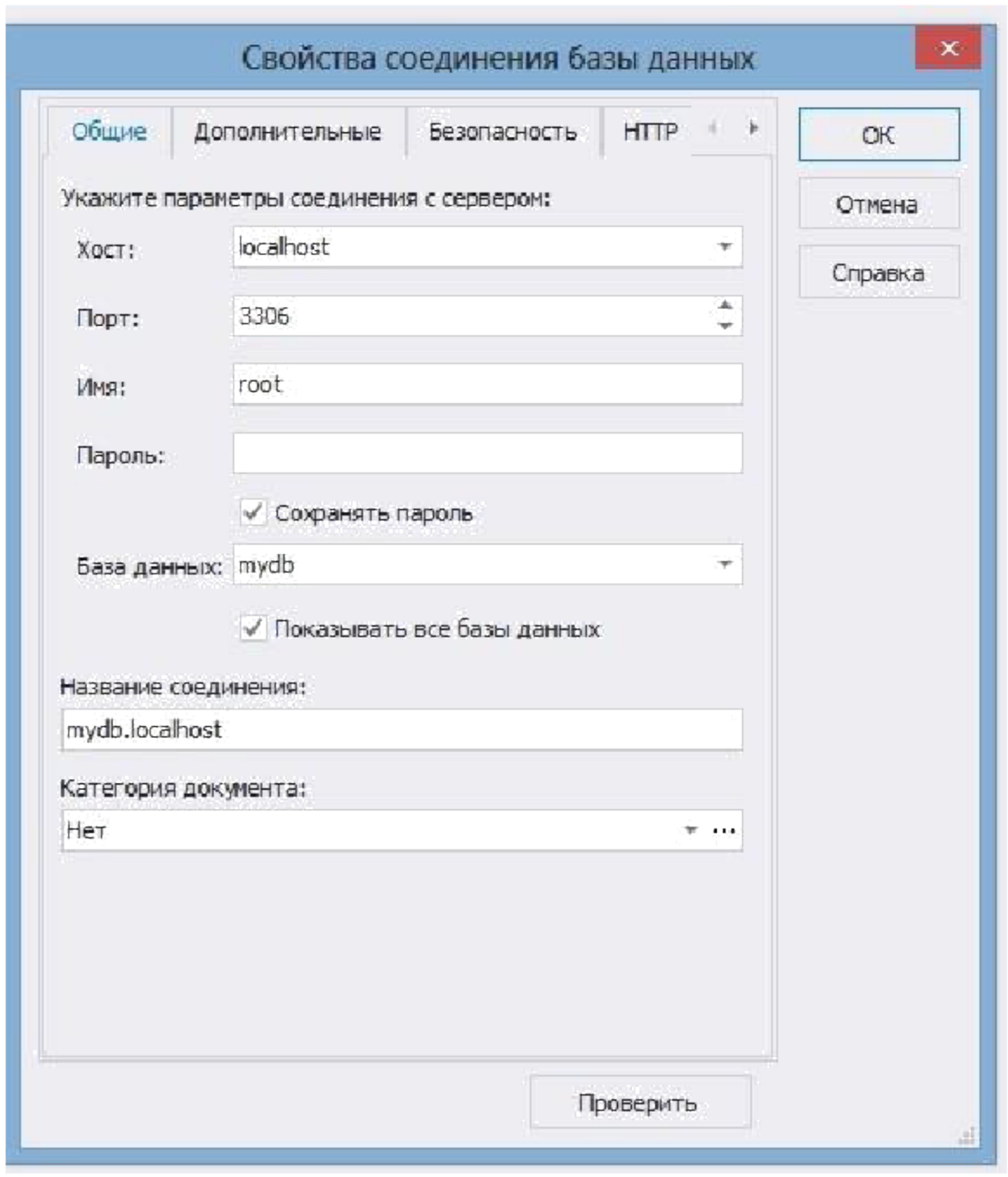

Рисунок 16.1 – Этап 1

Создаем тестовую базу данных «mydb».

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

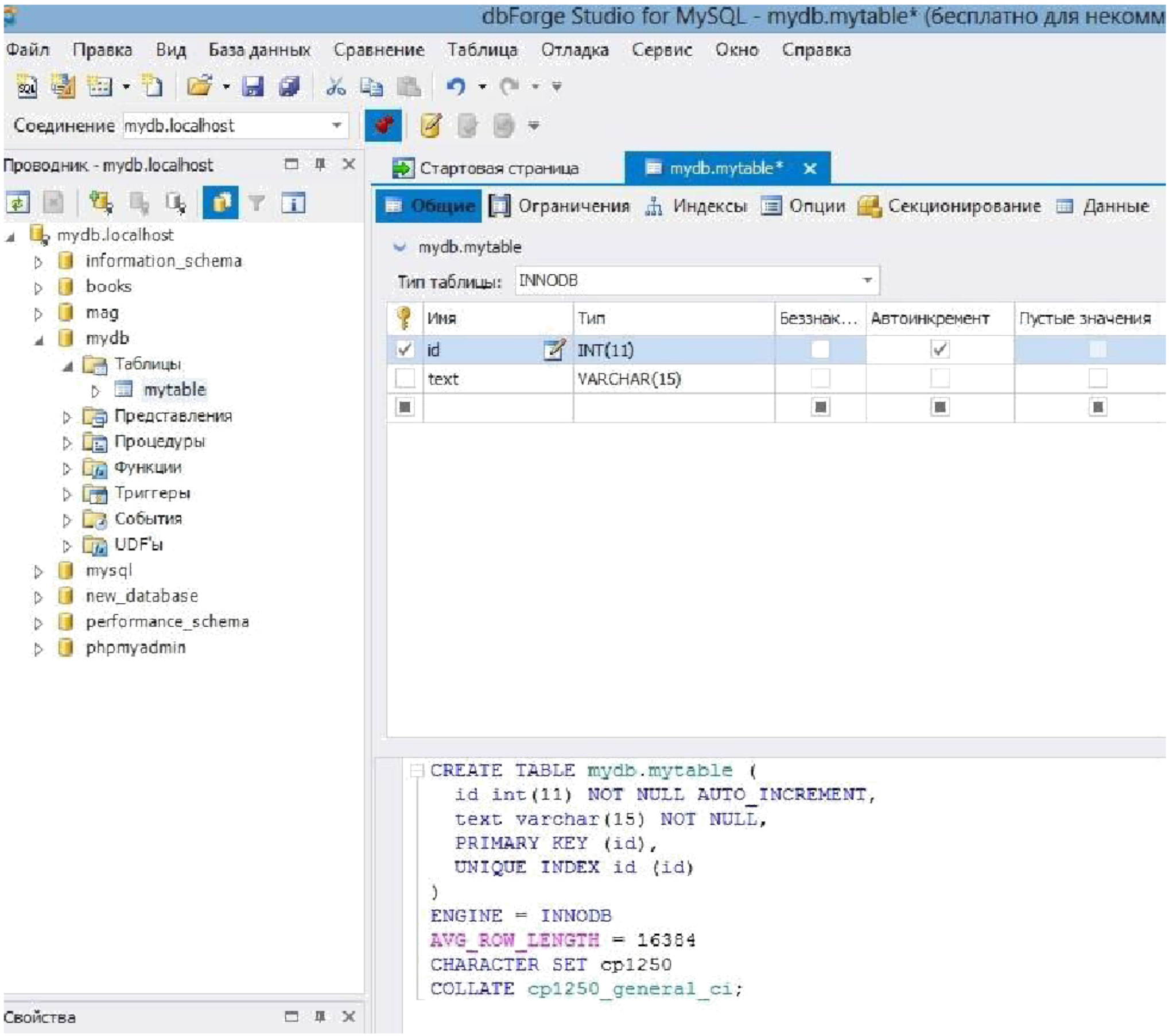

Рисунок 16.2 – Этап 3

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

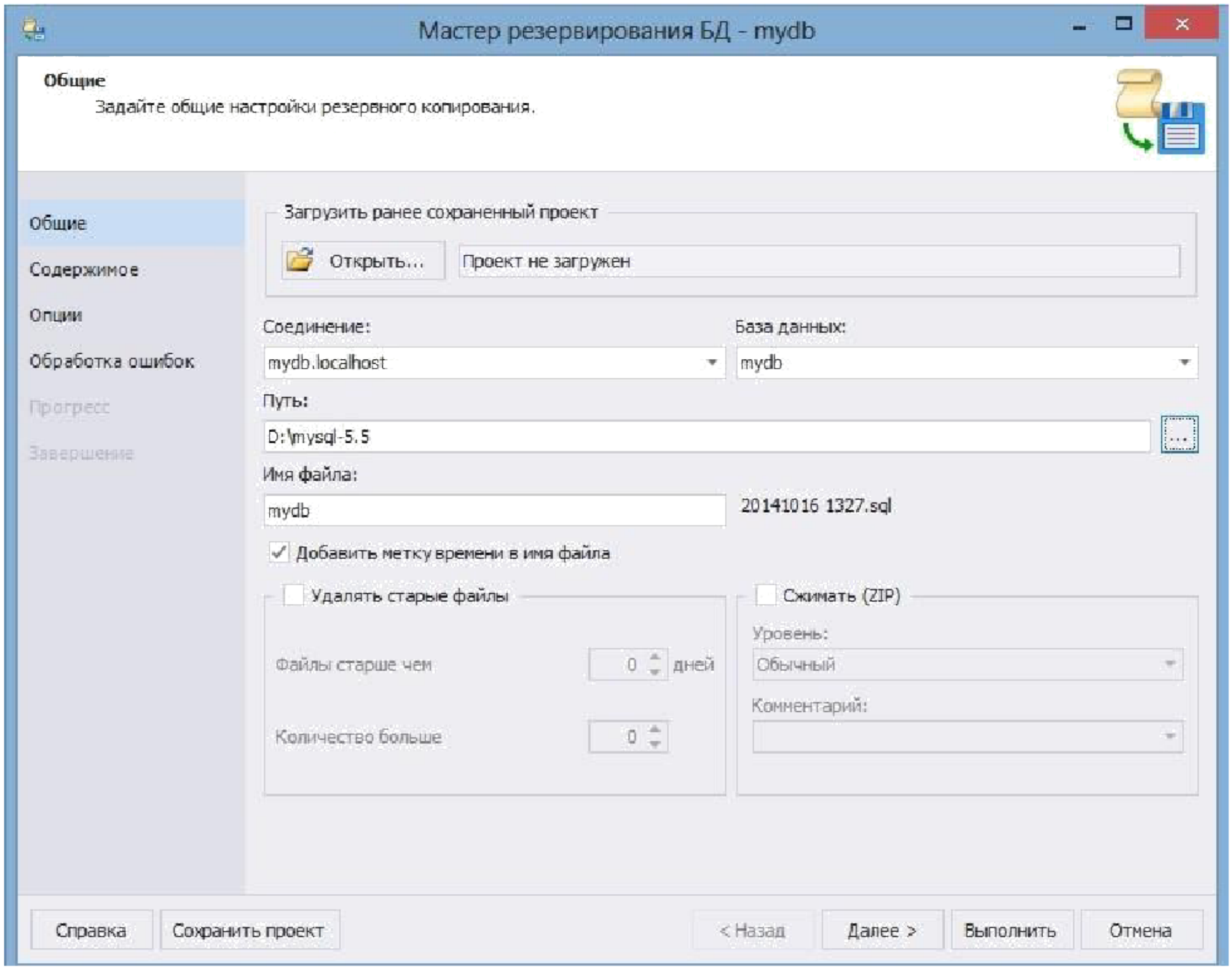

Рисунок 16.3 – Этап 4

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

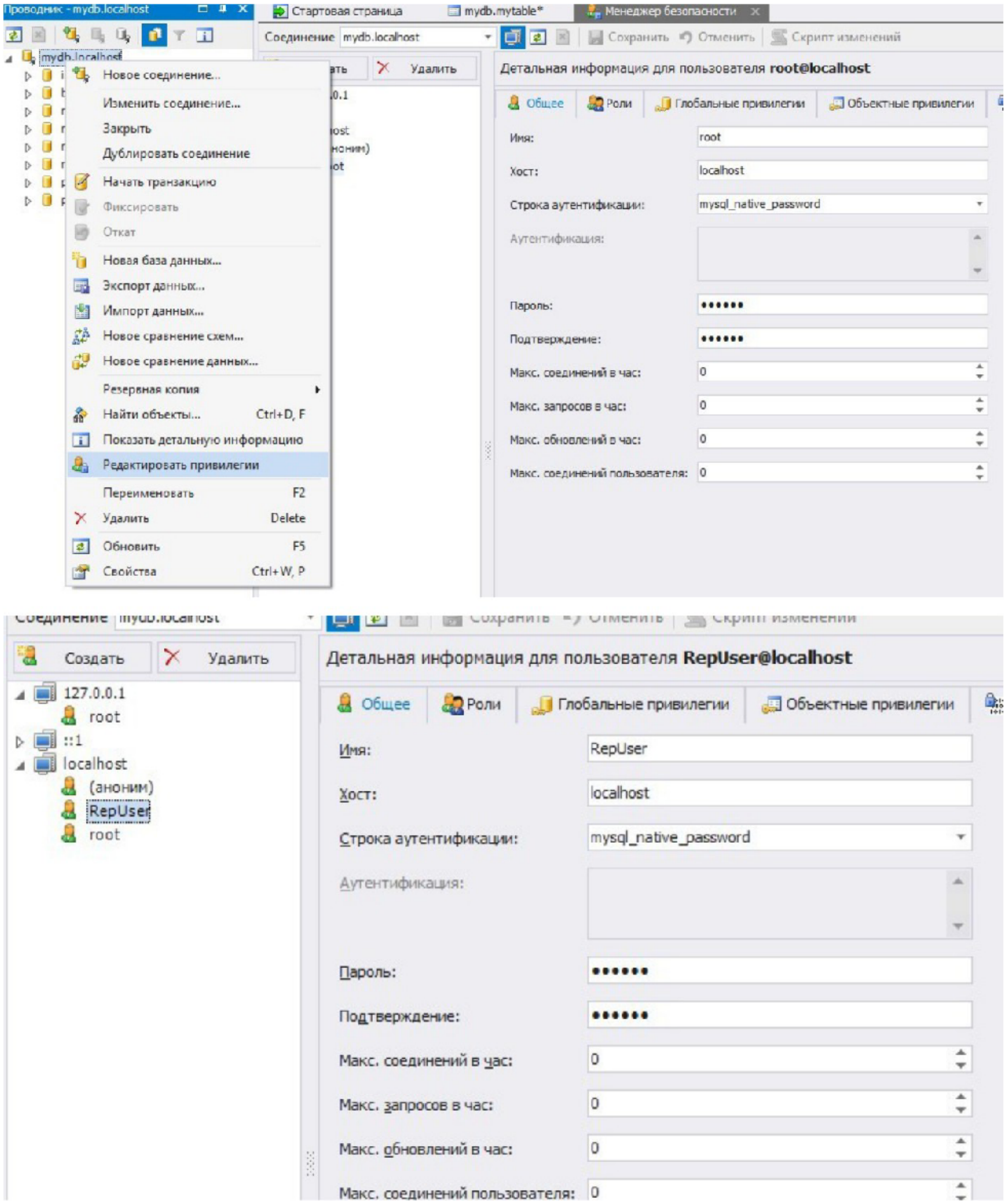

Рисунок 16.4 – Создание пользователя с паролем

Детальная информация для пользователя **RepUser@localhost**

Общее | Роли | Глобальные привилегии | Объектные привилегии | SSL

Название	Выбранные
Alter	☐
Alter Routine	☐
Create	☐
Create Routine	☐
Create Tablespace	☐
Create Temporary Tables	☐
Create User	☐
Create View	☐
Delete	☐
Drop	☐
Event	☐
Execute	☐
File	☐
Index	☐
Insert	☐
Lock Tables	☐
Process	☐
References	☐
Reload	☐
Replication Client	☐
Replication Slave	☑
Select	☐
Show Databases	☐
Show View	☐
Shutdown	☐
Super	☐
Trigger	☐
Update	☐

Рисунок 16.5 – Этап 5

Останавливаем головной сервер.

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

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

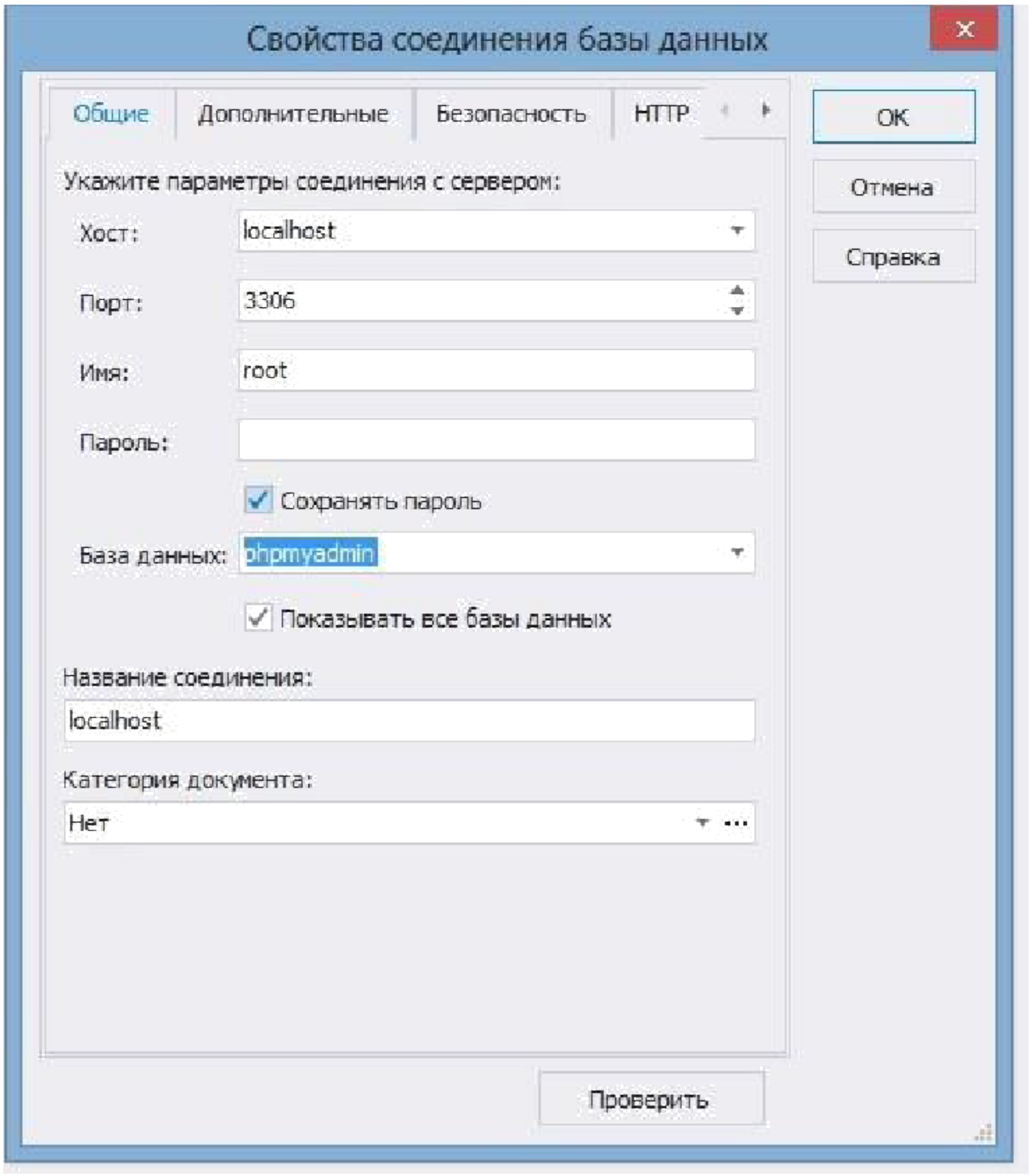

Рисунок 16.6 – Этап 7

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

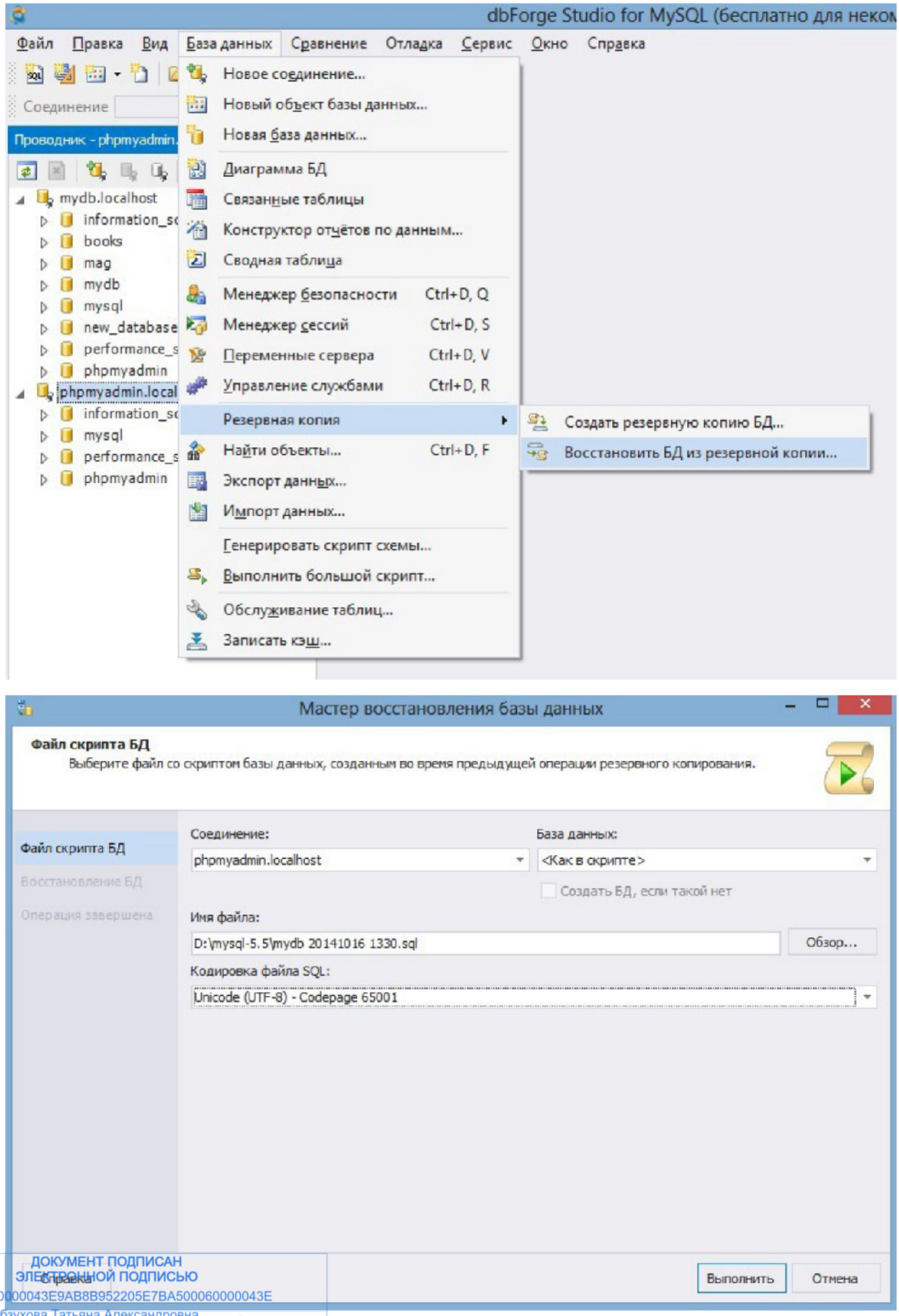

Рисунок 16.7 – Этап 8

Останавливаем подчиненный сервер (StopMySQL).

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

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

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

Запускаем головной сервер

Запускаем подчиненный сервер

Запускаем командную строку и переходим в папку bin подчиненного сервера

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

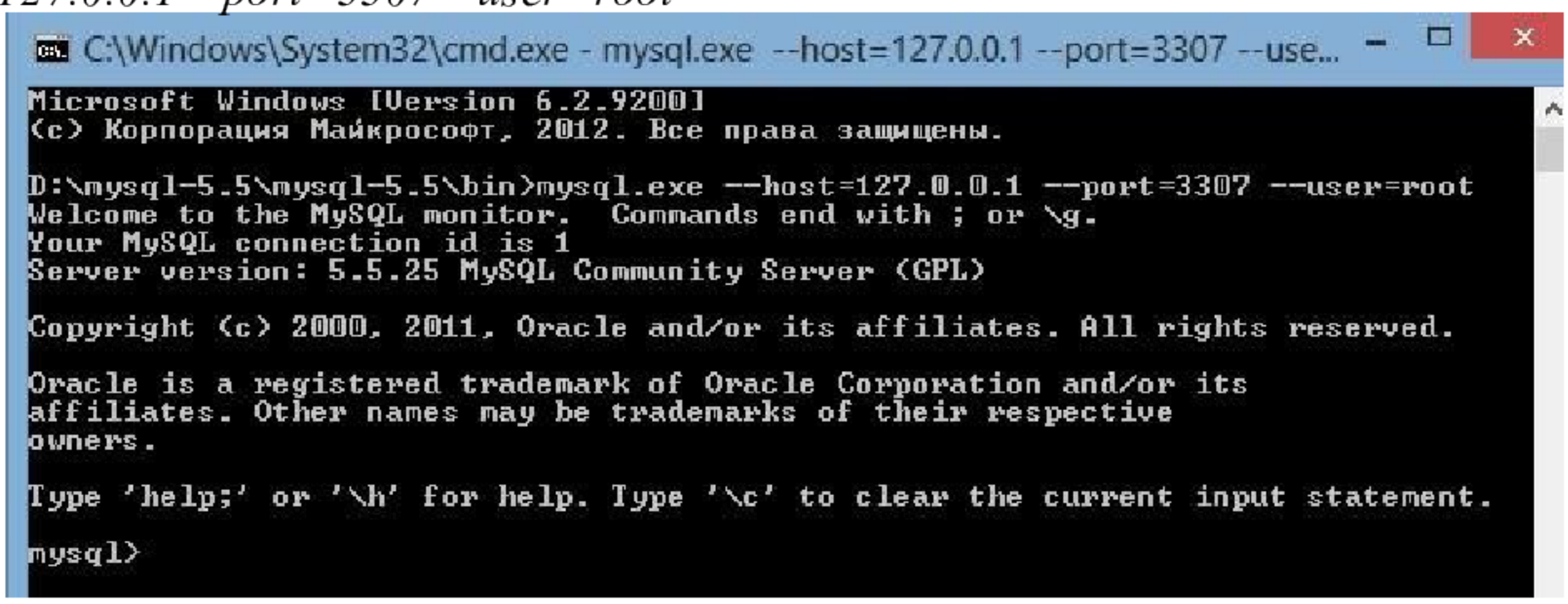

Рисунок 16.8 – Этап 15

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

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

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

```
mysql> change master to master_host='127.0.0.1',master_port=3306, master_user='R
epUser', master_password='reppass';
Query OK, 0 rows affected (0.24 sec)

mysql>
```

Рисунок 16.9 – Этап 16

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

```
mysql> show slave status \G
*************************** 1. row ***************************
               Slave_IO_State: Connecting to master
                  Master_Host: 127.0.0.1
                  Master_User: RepUser
                  Master_Port: 3306
                Connect_Retry: 60
              Master_Log_File:
          Read_Master_Log_Pos: 4
               Relay_Log_File: L220-onit-04-relay-bin.000001
                Relay_Log_Pos: 4
        Relay_Master_Log_File:
             Slave_IO_Running: Connecting
            Slave_SQL_Running: Yes
              Replicate_Do_DB:
          Replicate_Ignore_DB:
           Replicate_Do_Table:
       Replicate_Ignore_Table:
      Replicate_Wild_Do_Table:
  Replicate_Wild_Ignore_Table:
                   Last_Errno: 0
                   Last_Error:
                 Skip_Counter: 0
          Exec_Master_Log_Pos: 0
              Relay_Log_Space: 107
              Until_Condition: None
               Until_Log_File:
                Until_Log_Pos: 0
           Master_SSL_Allowed: No
           Master_SSL_CA_File:
           Master_SSL_CA_Path:
              Master_SSL_Cert:
            Master_SSL_Cipher:
               Master_SSL_Key:
        Seconds_Behind_Master: NULL
Master_SSL_Verify_Server_Cert: No
                Last_IO_Errno: 1045
                Last_IO_Error: error connecting to master 'RepUser@127.0.0.1:330
6' - retry-time: 60  retries: 86400
               Last_SQL_Errno: 0
               Last_SQL_Error:
  Replicate_Ignore_Server_Ids:
             Master_Server_Id: 0
1 row in set (0.00 sec)

mysql> _
```

Рисунок 16.10 – Этап 18

Смотрим список баз данных, рисунок 16.11.
SHOW DATABASES;

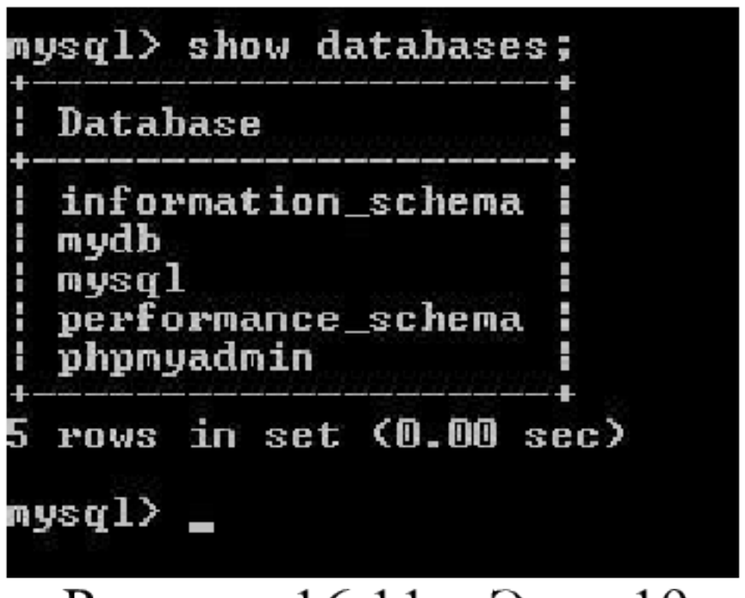

Рисунок 16.11 – Этап 19

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

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

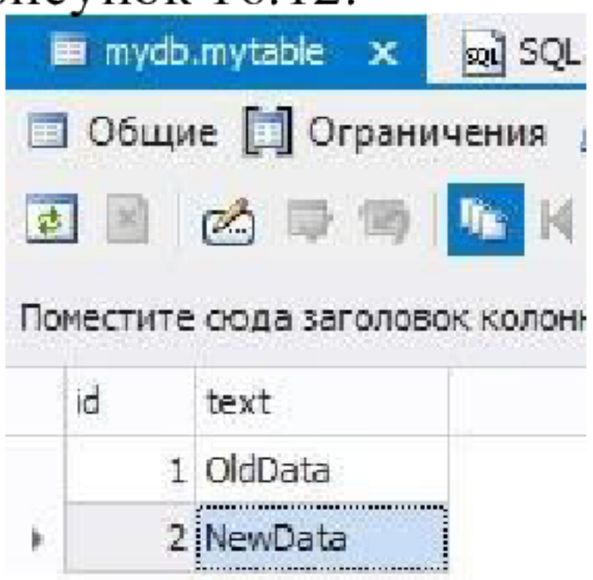

Рисунок 16.12 – Этап 21

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

SHOW DATABASES;

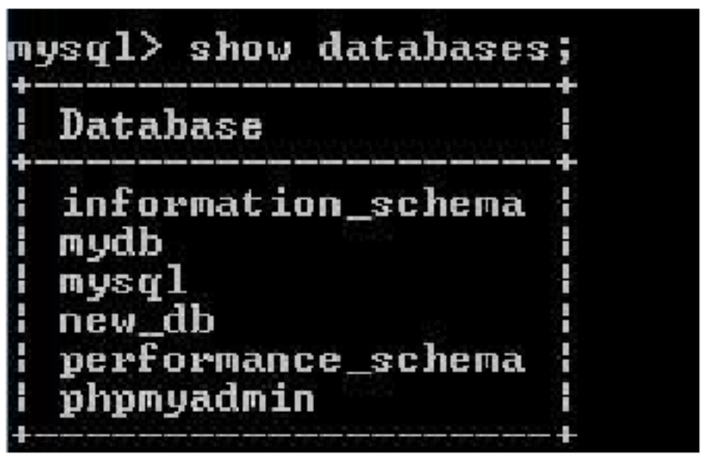

Рисунок 16.13 – Этап 22

Как мы видим новая база данных «newdb» появилась в списке 23.

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

Сначала подключаемся к базе, рисунок 16.14.

USEmydb;

Рисунок 16.14 – Этап 23, подключение к БД

Выводим список строк, рисунок 16.15:
*SELECT * FROMmytable*

```
mysql> Use mydb;
Database changed
mysql> select * from mytabl
    -> ;
+----+----------+
| id | text     |
+----+----------+
|  1 | OldData  |
|  2 | NewData  |
+----+----------+
2 rows in set (0.00 sec)
```

Рисунок 16.15 – Этап 23

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

Выход из командного интерпретатора mysql.exe осуществляется командой \q

\q

Выход из командного интерпретатора mysql.exe осуществляется командой exit

Exit

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

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

Задание

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

Содержание отчета и его форма

Отчет по лабораторной работе должен состоять из:
названия лабораторной работы;
описание примеров репликации.

Вопросы для защиты работы:

1. Каким образом осуществляется репликация?

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

Цель работы: научиться управлять репликациями вMySQL.

Теоретическое обоснование

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

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

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

	CHANGE MASTER TO MASTER_HOST='master2.mycompany.com', MASTER_USER='replication', MASTER_PASSWORD='bigs3cret', MASTER_PORT=3306, MASTER_LOG_FILE='master2-bin.001', MASTER_LOG_POS=4;
	Следует указывать только те значения, которые подлежат изменению. Не указанные значения останутся неизменными, за исключением тех случаев, когда изменяется хост или порт. В этом случае подчиненный сервер считает, что поскольку изменяется хост или порт, головной сервер становится другим. Следовательно, старые значения и точки положения в журнале будут автоматически заменены на значение пустой строки и 0 соответственно (начальные значения). Обратите внимание: если подчиненный сервер перезапускается, он сохраняет "память" о своем последнем головном сервере. Если это нежелательно, можно перед перезапуском удалить файл 'master.info' - тогда подчиненный сервер будет считывать информацию о своем головном сервере из файла 'my.cnf' или из командной строки. Эта команда используется для настройки подчиненного сервера при наличии образа головного сервера, а также записей из журнала и сдвига головного сервера, которые соответствуют образу. Можновыполнитькоманду CHANGE MASTER TO MASTER_LOG_FILE='log_name_on_master', MASTER_LOG_POS=log_offset_on_master на подчиненном сервере после восстановления образа (подчиненный сервер)
SHOW MASTER STATUS	Выводит информацию о состоянии головного сервера, исходя из информации в двоичных журналах (головной сервер)
SHOW SLAVE	Присутствует в версии 4.0.0 и выше.