Министерство образования и науки Российской Федерации
Федеральное государственное бюджетное образовательное
учреждение высшего образования
«Владимирский государственный университет имени Александра
Григорьевича и Николая Григорьевича Столетовых»

А.А. ВОРОНИН

# БАЗЫ ДАННЫХ И ЭКСПЕРТНЫЕ СИСТЕМЫ. ЧАСТЬ 1

Методические указания для студентов по выполнению
лабораторных работ по дисциплине

Владимир 2023

УДК 004.4, 004.65
ББК 32.972

Рецензент:
Заместитель директора по учебно-методической работе ГАПОУ ВО «Владимирский авиамеханический колледж», преподаватель высшей квалификационной категории
*А.Е. Петров*

**Воронин, А. А.** Базы данных и экспертные системы. Часть 1 [Электронный ресурс] : метод. указания для студентов по выполнению лабораторных работ по дисциплине / А. А. Воронин, Владим. гос. ун-т им. А. Г. и Н. Г. Столетовых. – Изд. 1-е. – Владимир : , 2023. – с. – ISBN . – Электрон. дан. (5 Мб). – 1 электрон. опт. диск (CD-ROM). – Систем. требования: Intel от 1,3 ГГц; Windows XP/7/8/10; Adobe Reader; дисковод CD-ROM. – Загл. с титул. экрана.

Методические указания содержат описания лабораторнх работ, каждое из которых состоит из: теоретической части - минимального объема теоретической информации необходимого для выполнения поставленных задач; пояснений к выполнению работы; перечня заданий или вариантов заданий к работе.

Предназначено для студентов специальностей, изучающих дисциплину «Базы данных и экспертные системы» по программам академического бакалавриата и специалитета.

Ил. 135. Библиогр.: назв.

ISBN

## ОГЛАВЛЕНИЕ

Введение ..... 7

Лабораторная работа №1. ПОСТРОЕНИЕ РЕЛЯЦИОННОЙ СХЕМЫ ДАННЫХ ..... 10

1. Цель работы ..... 10

2. Содержание работы ..... 10

3. Настройка веб-сервера на ОС Windows ..... 10

4. Настройка веб-сервера на ОС Linux ..... 19

5. Работа с MySQL сервером ..... 23

5.1. Подсоединение к серверу и отсоединение от него ..... 23

5.2. Ввод запросов ..... 24

5.3. Создание и использование базы данных ..... 27

5.4. Типы данных столбцов ..... 30

6. Пояснения к выполнению работы ..... 34

7. Варианты заданий к лабораторной работе ..... 40

8. Отчетность по лабораторной работе ..... 43

Лабораторная работа №2. Взаимодействие с базой данных (коннекторы) ..... 44

1. Цель работы ..... 44

2. Содержание работы ..... 44

3. Работе с расширением mysqli ..... 45

4. Работе с расширением php pdo ..... 55

5. Операторы SQL ..... 58

6. Задания к лабораторной работе ..... 65

7. Отчетность по лабораторной работе ..... 65

Лабораторная работа №3. Взаимодействие с базой данных (фреймворки) ..... 66

1. Цель работы ..... 66
2. Содержание работы ..... 66
3. Концепция MVC ..... 67
4. Общие сведения по работе с фреймворком CodeIgniter ... 68
5. Запросы в CodeIgniter ..... 69
6. Контроллеры ..... 71
7. Модели ..... 73
8. Представления (виды) ..... 76
9. Рекомендованная последовательность действий по установке и настройке ..... 78
10. Пример реализации новостной ленты ..... 79
11. Пояснения к выполнению работы ..... 89
12. Отчетность по лабораторной работе ..... 90
Лабораторная работа №4. Взаимодействие с базой данных (фреймворки) ..... 91
1. Цель работы ..... 91
2. Содержание работы ..... 91
3. Установка и настройка среды ..... 91
4. Краткие сведения по работе с объектами Django ..... 98
5. Пояснения к выполнению работы ..... 106
6. Отчетность по лабораторной работе ..... 106
Лабораторная работа №5. Операции над отношениями в реляционных БД ..... 107
1. Цель работы ..... 107
2. Содержание работы ..... 107
3. Запросы объединения (соединения) ..... 107
4. Выборка данных ..... 114

5. Функции ..... 115
6. Задания к лабораторной работе ..... 126
7. Отчетность по лабораторной работе ..... 128
Лабораторная работа №6. Операции над отношениями в реляционных БД (SQL-инъекции) ..... 129
1. Цель работы ..... 129
2. Содержание работы ..... 129
3. Общие сведения ..... 129
4. Задания к лабораторной работе ..... 133
5. Отчетность по лабораторной работе ..... 133
Лабораторная работа №7. Специализированные СУБД ..... 134
1. Цель работы ..... 134
2. Содержание работы ..... 134
3. Общие сведения ..... 134
4. Объекты конфигурации ..... 138
5. Задания к лабораторной работе ..... 143
6. Отчетность по лабораторной работе ..... 143
Лабораторная работа № 8. Базы данных временных рядов (time series database) ..... 144
1. Цель работы ..... 144
2. Содержание работы ..... 144
3. Общие сведения ..... 144
4. Описание работы ..... 145
5. Задания к лабораторной работе ..... 156
6. Отчетность по лабораторной работе ..... 156
Лабораторная работа № 9. Оболочки экспертных систем ..... 157
1. Цель работы ..... 157

2. Содержание работы ..... 157
3. Общие сведения ..... 157
4. Примеры работы с фактами ..... 163
5. Использование правил для вывода результатов ..... 167
6. Задания к лабораторной работе ..... 168
7. Отчетность по лабораторной работе ..... 168
Лабораторная работа № 10. Оболочки экспертных систем (Функции) ..... 169
1. Цель работы ..... 169
2. Содержание работы ..... 169
3. Общие сведения ..... 169
4. Примеры создания пользовательских процедурных и рекурсивных функций ..... 172
5. Работа с функциями ..... 174
6. Задания к лабораторной работе ..... 177
7. Отчетность по лабораторной работе ..... 180
КОНТРОЛЬНЫЕ ВОПРОСЫ ..... 181
БИБЛИОГРАФИЧЕСКИЙ СПИСОК ..... 183

# ВВЕДЕНИЕ

Лабораторные занятия – это одна из разновидностей практического занятия, которые углубляют и закрепляют теоретические знания. На этих занятиях студенты осваивают принципы работы с аппаратным и программным обеспечением, учатся умению наблюдать, оценивать полученные результаты, делать выводы и обобщения.

Основная цель лабораторных работ - овладение техникой эксперимента, выработка умения решать практические задачи путем постановки опыта. В ходе занятия студенты сталкиваются с самостоятельной практической деятельностью в конкретной области.

Выполнение поставленных задач способствует достижению следующих целей:

- углубление и закрепление знания теоретического курса путем практического изучения в лабораторных условиях изложенных в лекциях материалов;
- приобретение навыков в научном экспериментировании, анализе полученных результатов;
- формирование первичных навыков организации, планирования и проведения работ, в том числе и научных исследований.

Типовой порядок проведения лабораторного занятия:

- Вводная часть: входной контроль подготовки студента; знакомство студентов с содержанием работы.
- Основная часть: проведение студентом лабораторной работы; разъяснения (в случае необходимости).
- Заключительная часть: демонстрация результатов выполнения лабораторной работы; оформление отчета о выполнении задания (подготовка материалов, необходимых для подготовки отчета).

Проведения лабораторных занятий предполагает одновременное выполнение работы всеми студентами в рамках группы (подгруппы). Работы выполняются в определенной последовательности. В процессе лабораторного занятия студенты выполняют одну из лабораторных работ (заданий).

Задача на подготовку к лабораторной работе может быть поставлена на лекции, на практическом занятии с таким временным расчетом, чтобы студенты смогли качественно подготовиться к ее проведению.

Результаты выполнения лабораторной работы оформляются студентами в виде отчета. Как правило, отчет состоит из трех частей:

- В первой части указываются наименование и цель выполнения лабораторной работы, приводится краткое описание используемого аппаратного и программного обеспечения, которые использовались в процессе выполнения работы, приводится схема установки, используемой в работе (если использовалась установка).
- Вторая часть отчета посвящается описанию результатов работы (описанию данных, полученных в ходе выполнения работы).
- В третьей части приводятся выводы по работе.

Отчет предоставляется преподавателю для проверки и утверждения. В ходе проверки отчета может проводиться защита результатов работы (пояснения по ходу выполнения работы и/или ответы на вопросы). Качественное оформление отчетной документации является одним из условий получения баллов за выполненную работу.

Оценки за выполнение лабораторных работ учитываются в рейтинг-контроле.

Работы выборочно проверяются на плагиат.

Отсутствие результатов работ и отчетов у студента до выставления итоговой оценки по дисциплине, а также

невозможность их продемонстрировать, является основанием для обнуления баллов за работу (даже в случае, если результаты работы были продемонстрированы ранее).

Методические указания содержат описания лабораторных работ, каждое из которых состоит из:

- теоретической части - минимального объема теоретической информации необходимого для выполнения поставленных задач;
- пояснений к выполнению работы;
- перечьня заданий или вариантов заданий к работе.

# ЛАБОРАТОРНАЯ РАБОТА №1. ПОСТРОЕНИЕ РЕЛЯЦИОННОЙ СХЕМЫ ДАННЫХ

## 1. Цель работы

1.1. Изучение процесса установки и настройки системы управления базами данных (СУБД).

1.2. Изучение команд языка структурированных запросов SQL, необходимых для построения реляционной схемы данных предметной области.

## 2. Содержание работы

2.1. Установить и настроить СУБД.

2.2. По выбранным образцам документов с помощью команд языка SQL провести построение реляционной схемы данных, приведенной в качестве примера, в установленной СУБД.

2.3. Реализовать одну из схем данных приведенных вариантов заданий.

## 3. Настройка веб-сервера на ОС Windows[1]

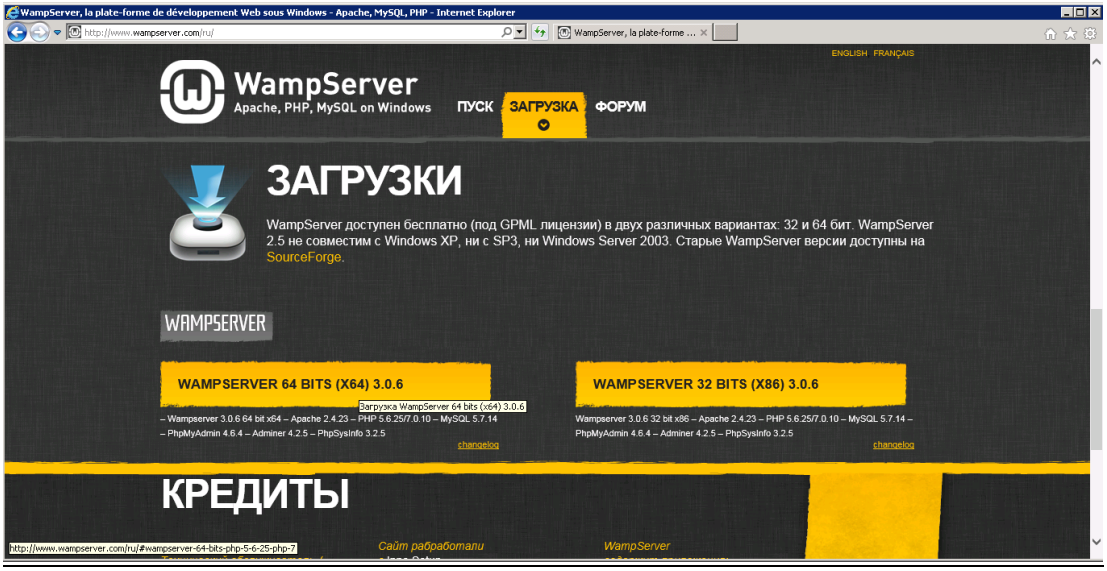
WampServer это среда для web-разработки на Windows. Содержит Apache, php, MySQL и имеет автоматический инсталлятор, что облегчает установку и позволяет настроить простой web-сервер.

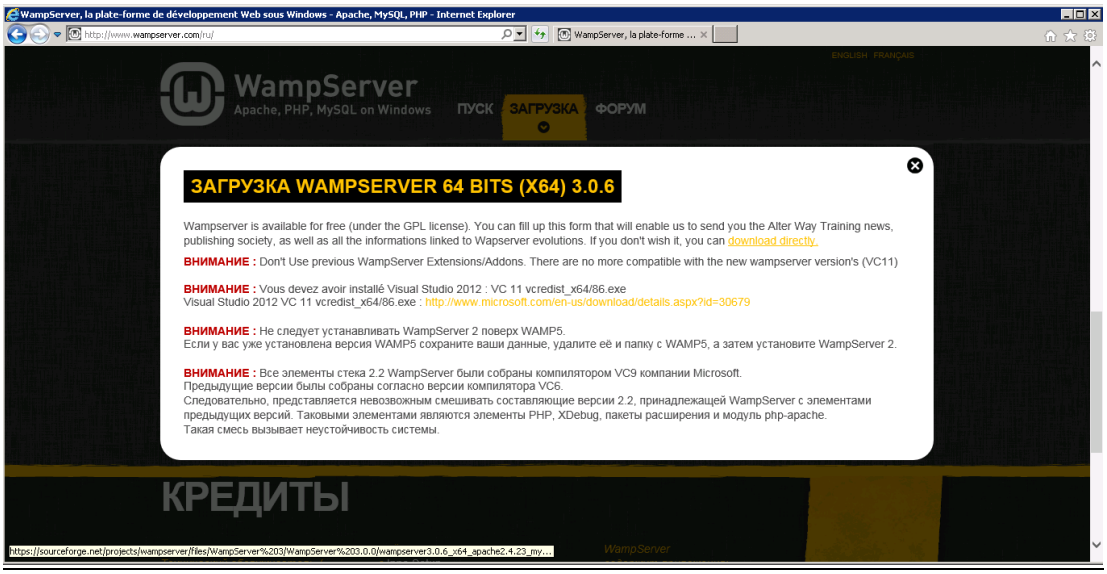
Перед началом установки WampServer необходимо настроить межсетевой экран, указав специфические порты: 80 и 443.

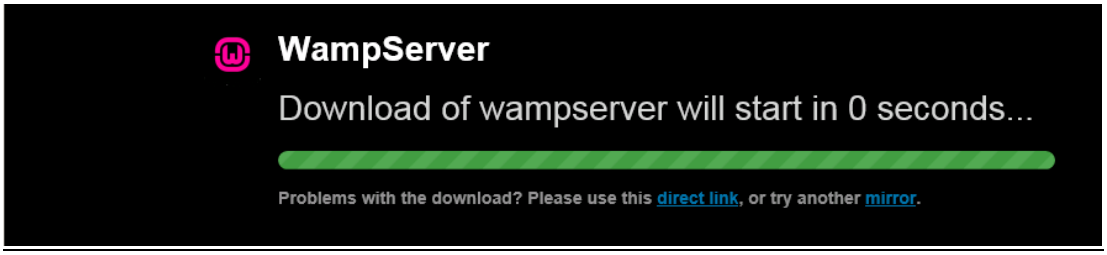
Перед работой с WampServer для начала установим необходимые библиотеки. Для запуска Apache требуются компоненты The Visual C++ Redistributable Packages. Выберите пакет установки в соответствии с разрядностью вашей операционной системы.

Теперь необходимо загрузить установщик WampServer.

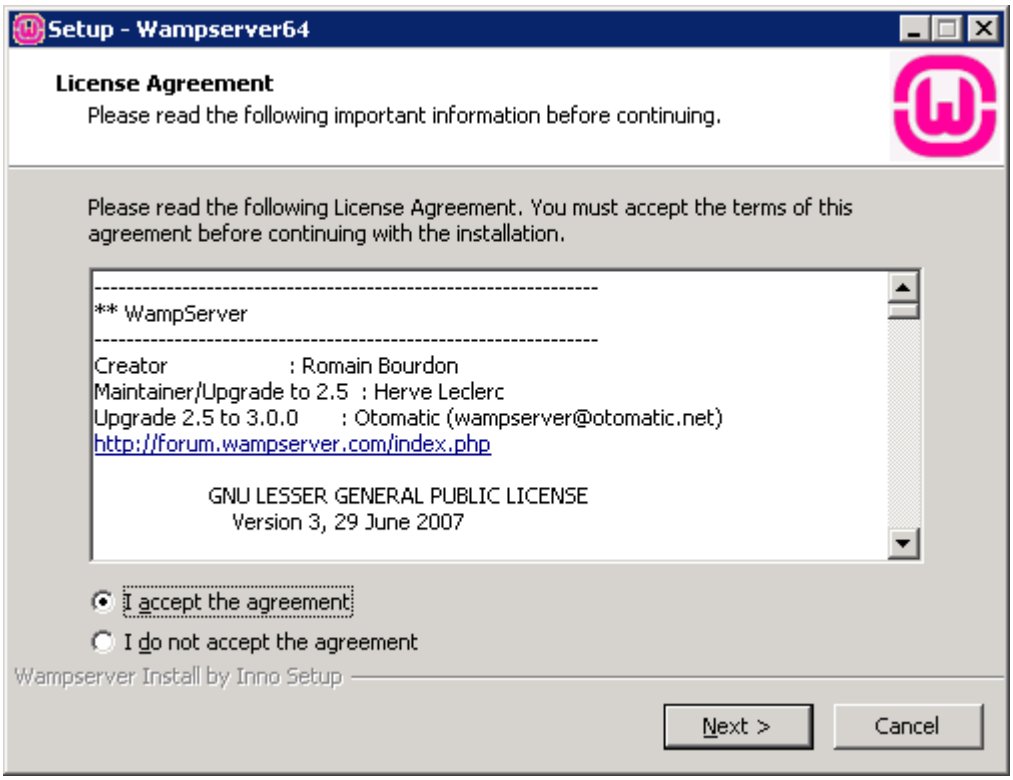
[1] Использованы материалы с сайта https://1cloud.ru

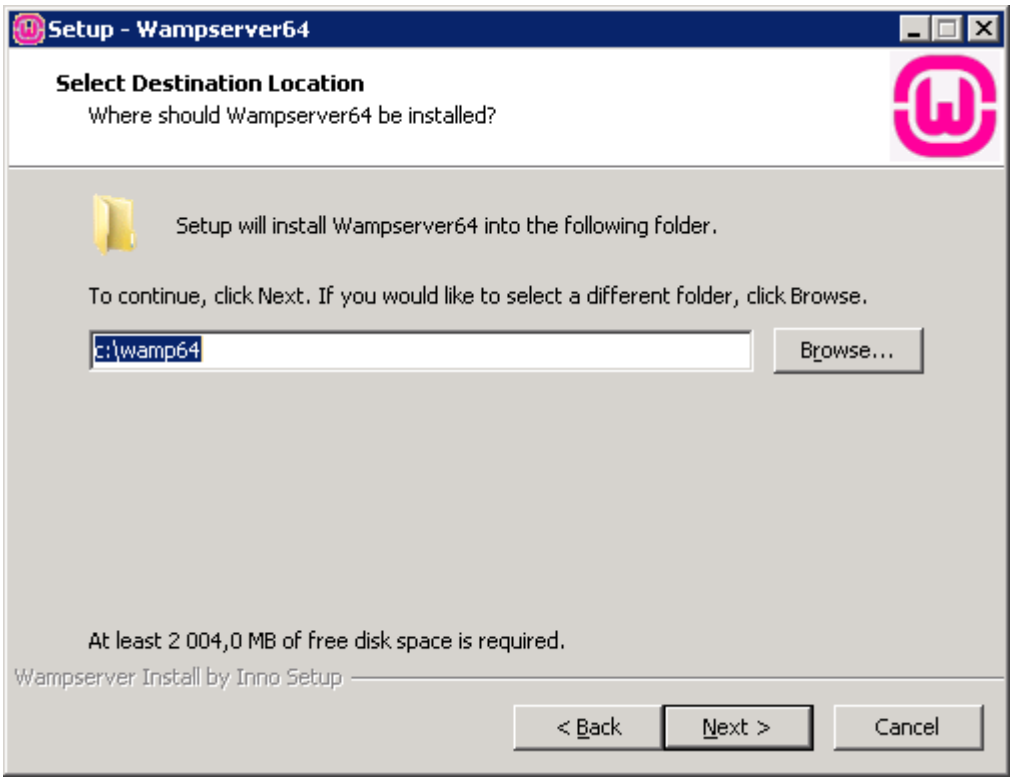


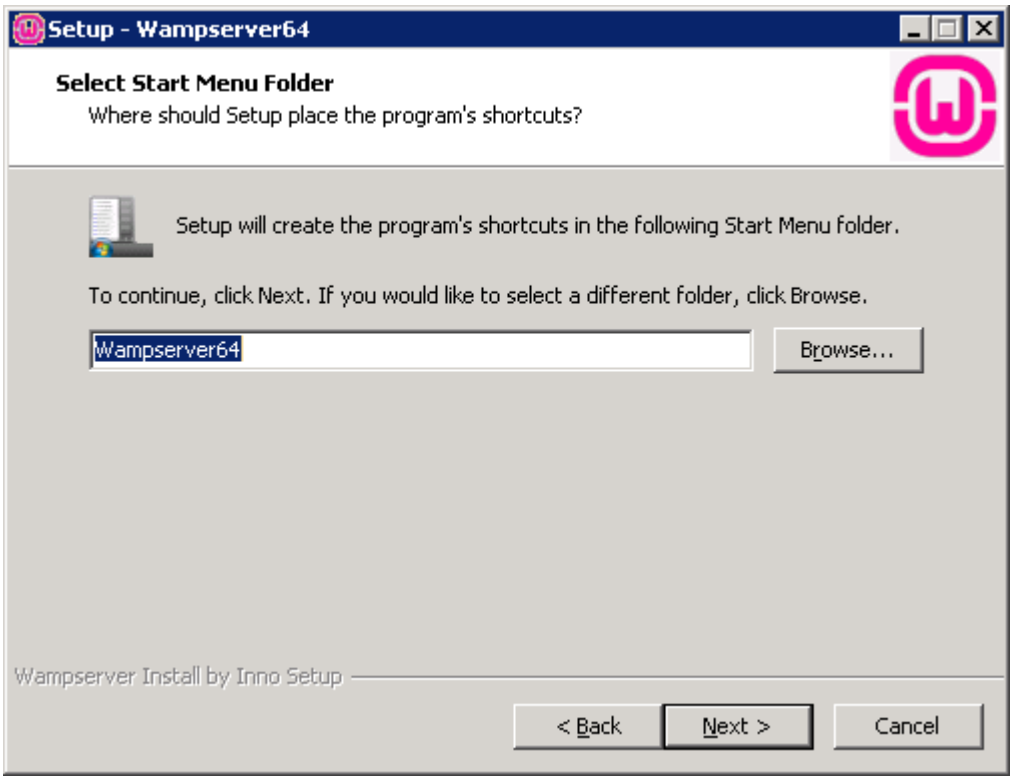



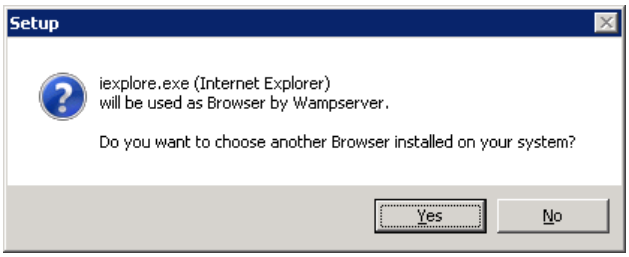

Перейдите в папку загрузки и запустите на исполнение скачанный файл.


Setup - Wampserver64
License Agreement
Please read the following important information before continuing.
Please read the following License Agreement. You must accept the terms of this agreement before continuing with the installation.
** WampServer
Creator : Romain Bourdon
Maintainer/Upgrade to 2.5 : Herve Leclerc
Upgrade 2.5 to 3.0.0 : Otomatic (wampserver@otomatic.net)
http://forum.wampserver.com/index.php
GNU LESSER GENERAL PUBLIC LICENSE
Version 3, 29 June 2007
I accept the agreement
I do not accept the agreement
Wampserver Install by Inno Setup
Next >
Cancel

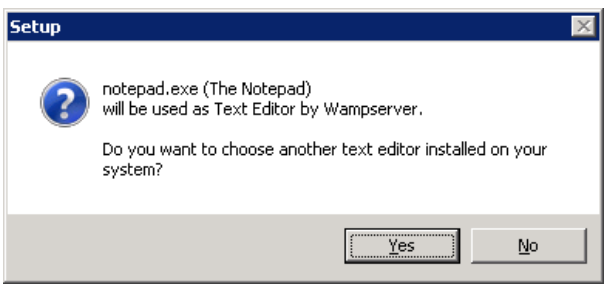

Setup - Wampserver64
Select Destination Location
Where should Wampserver64 be installed?
Setup will install Wampserver64 into the following folder.
To continue, click Next. If you would like to select a different folder, click Browse.
c:\wamp64
Browse...
At least 2 004,0 MB of free disk space is required.
Wampserver Install by Inno Setup
< Back
Next >
Cancel



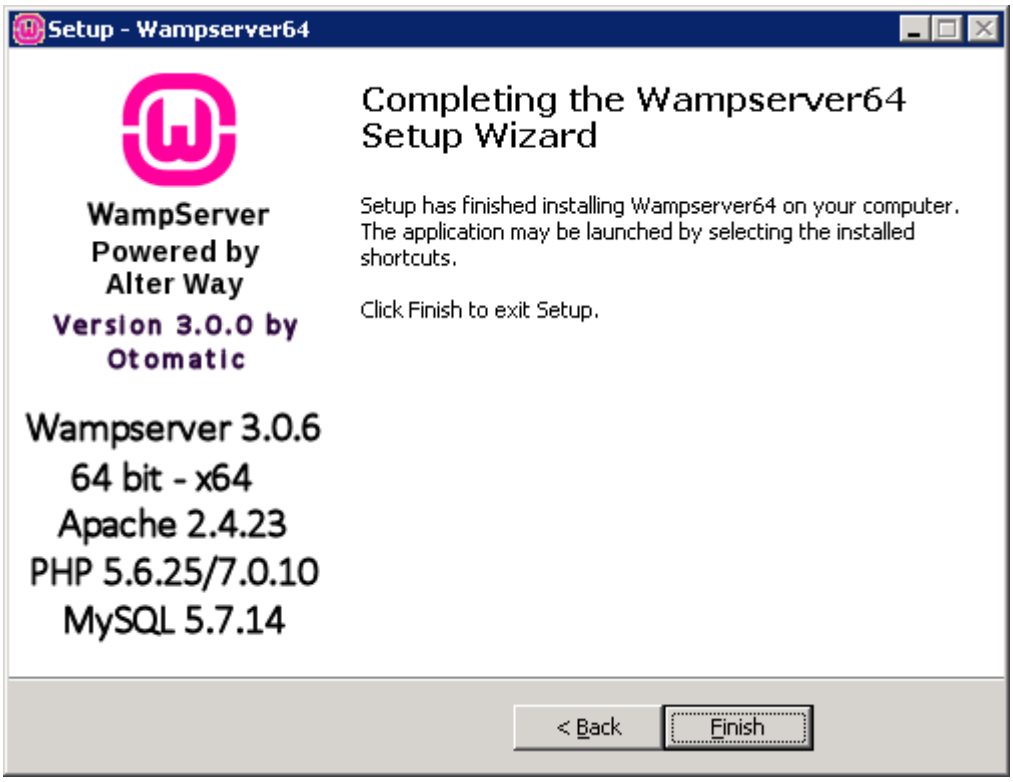

Выберите браузер по умолчанию, например Internet Explorer.




Выберите текстовый редактор по умолчанию, например The Notepad.

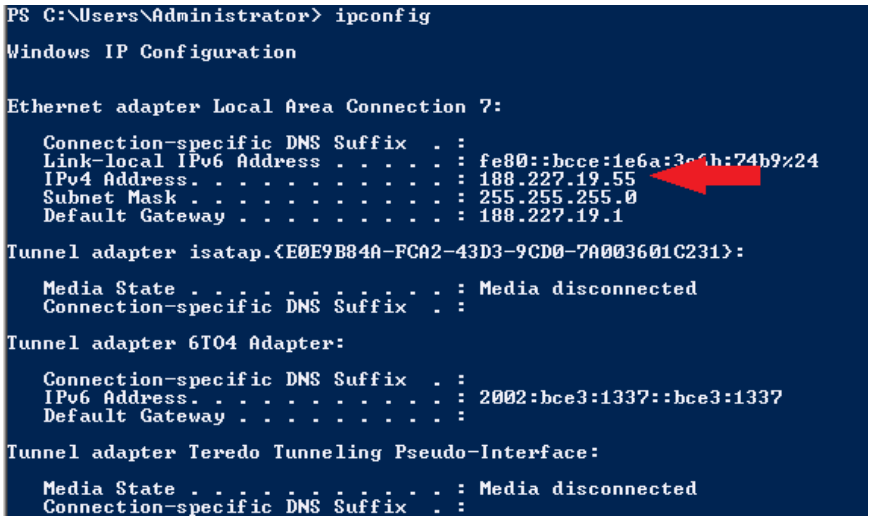


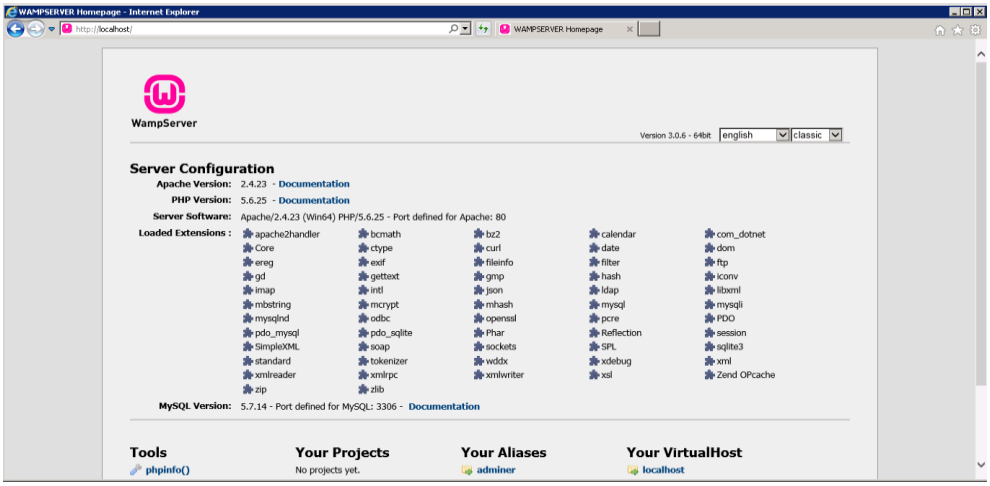
Установка завершена.




На рабочем столе появится иконка Wampserver. После запуска в системном трее появится зеленый значок запущенного сервера.


В браузере перейдите по ссылке http://localhost/ или вашему ip адресу для проверки (узнать IP адрес можно в настройках сетевого адаптера или командой ipconfig в командной строке).

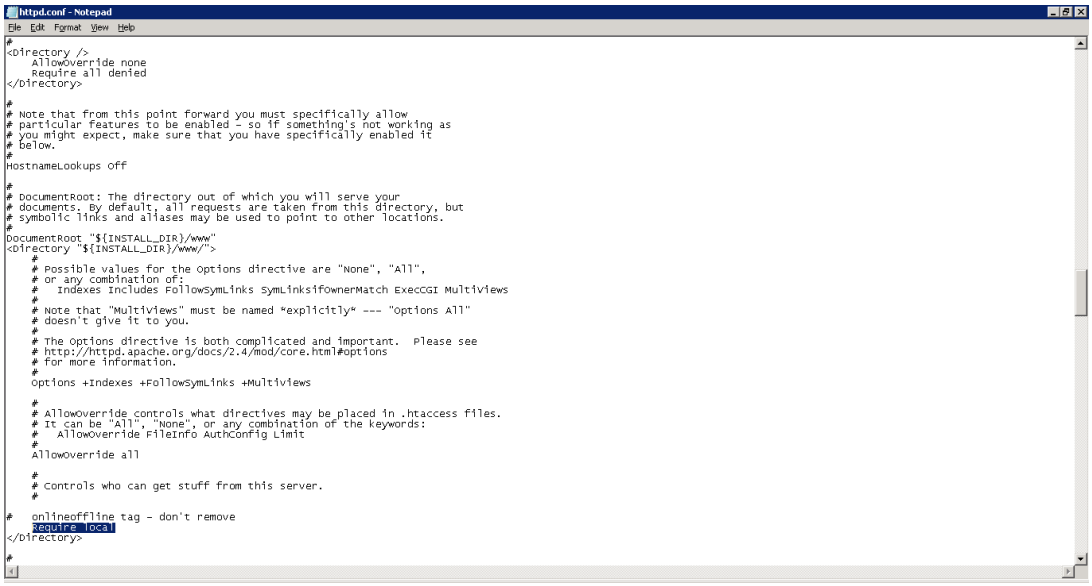


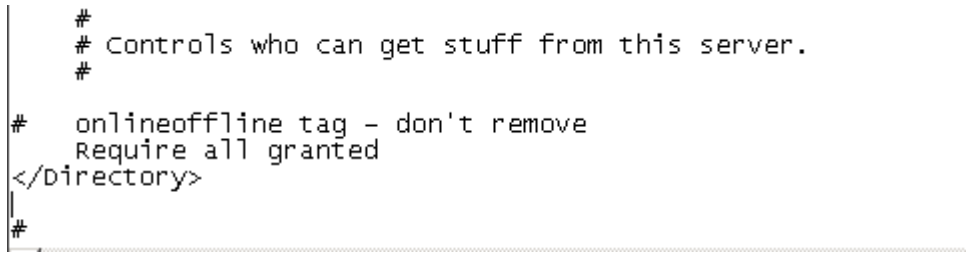




Доступ с других компьютеров к установленному серверу возможен по IP адресу компьютера или доменному имени компьютера. Для этого необходимо сконфигурировать Apache[2].

Откройте файл conf\httpd.conf в каталоге установки Apache с помощью текстового редактора. С помощью поиска найдите первое вхождение строки “Require local”, и замените параметр “local” на “all granted”.

[2] Далее приведены команды настройки для версии Apache 2.4. Для настройки Apache версии 2.2 необходимо использовать другие команды. Уточняйте в документации.



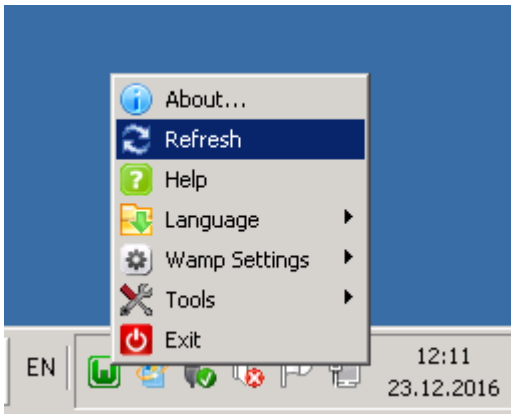




Сохраните изменения и закройте файл.

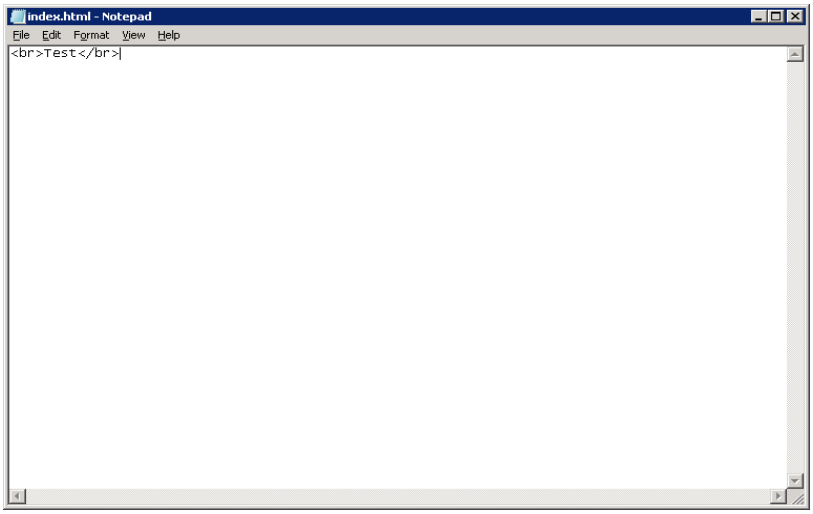
Откройте файл конфигурации виртуальных хостов conf\extra\httpd-vhosts.conf и замените содержимое тега Directory на следующие параметры:

```
Options FollowSymLinks
AllowOverride All
Order deny,allow
allow from all
```

Сохраните изменения и закройте файл. Перезапустите WampServer нажав правой кнопкой мыши в нижней панели на значок сервера и выберите пункт “Refresh”.




В папке www создайте файл index.html и отредактируйте с помощью текстового редактора, добавив содержимое на языке разметки HTML. Сохраните изменения и закройте файл.

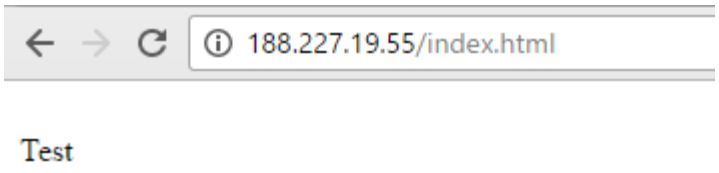



Далее, с другого компьютера в браузере попытайтесь перейти по адресу:

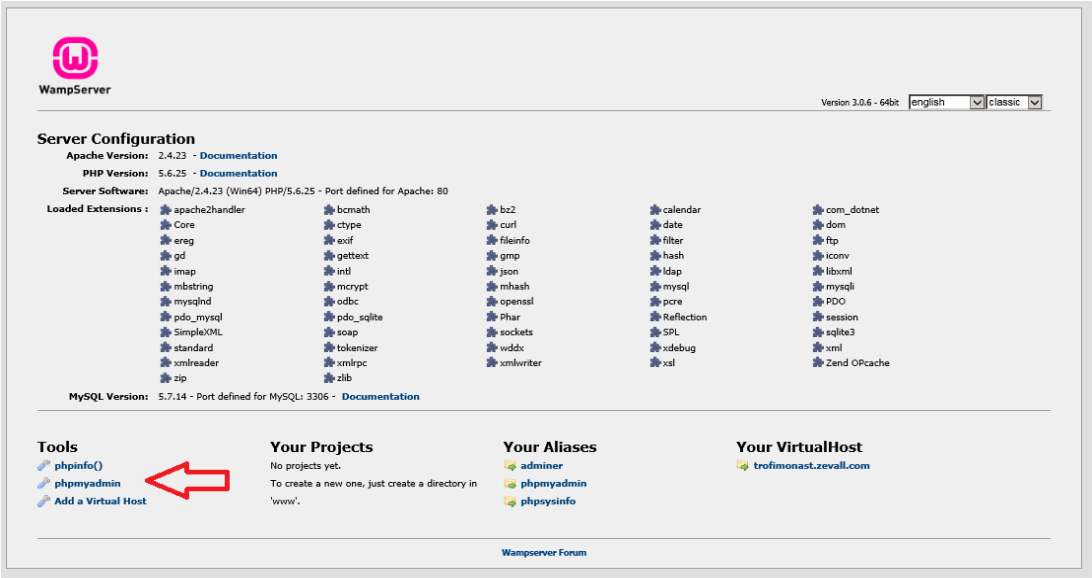
```
<ip-address>/index.html
```

где <ip-address> - ip-адрес вашего веб-сервера.

Например:




Манипуляции с базами данных осуществляются через phpMyAdmin. На своем локальном хосте на главной странице WampServer в меню “Tools” выберете “phpmyadmin”, или перейдите по ссылке http://localhost/phpmyadmin/ в браузере.




В открывшемся окне введите значения логина и пароля по умолчанию:

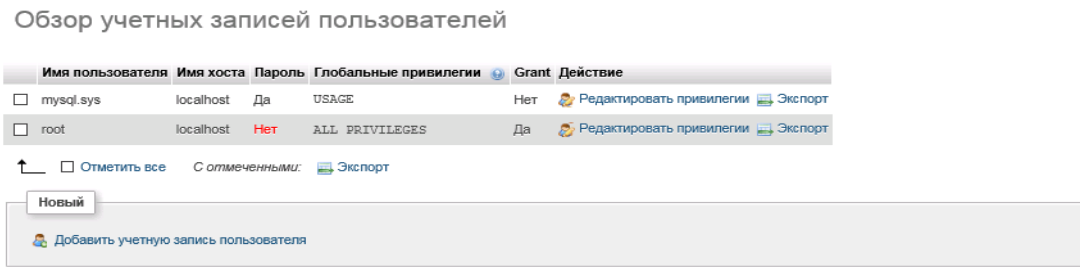
Логин: *root*

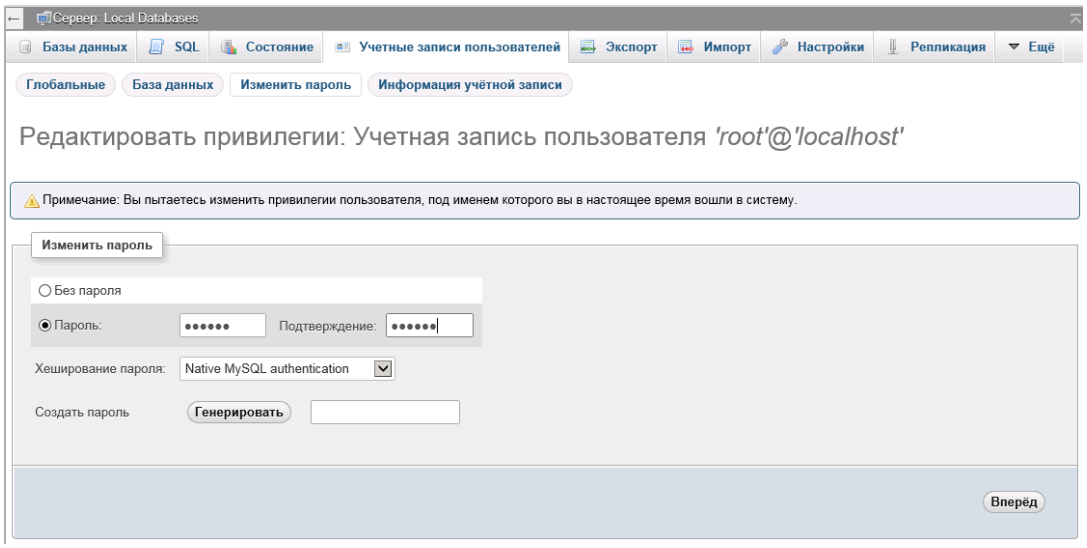
Пароль: *оставьте поле пустым*

После авторизации вы можете изменить пароль для пользователя root, а также добавить новых пользователей. Для этого перейдите во вкладку “Учетные записи пользователей”.

Для добавления пользователя перейдите по ссылке “Добавить учетную запись пользователя”. Чтобы изменить пароль для уже

существующих пользователей выберите действие “Редактировать привилегии”, далее “Изменить пароль”.




Сохраните изменения и перезапустите WampServer нажав правой кнопкой мыши в нижней панели на значок сервера и выберите пункт “Refresh”.

## 4. Настройка веб-сервера на ОС Linux[3]

Стек LAMP - это набор открытого программного обеспечения, который используется для создания веб-серверов. Аббревиатура является набором первых букв в названии технологий Linux, Apache, MySQL и PHP.

Причина, по которой LAMP называют стеком, заключается в том, что каждый уровень зависит от его базового (нижнего) уровня.

[3] Использованы материалы с сайта https://1cloud.ru. На примере ОС Ubuntu.

Ваша операционная система Ubuntu - это базовый уровень. Затем Apache, ваш веб-демон находится поверх вашей ОС. Затем база данных хранит всю информацию, обслуживаемую вашим веб-демоном, а язык сценариев Php используется для управления и отображения всех данных, а также для взаимодействия с пользователем.

LAMP стал де-факто стандартом разработки. Сегодня продукты, составляющие стек LAMP, по умолчанию включены почти во все дистрибутивы Linux, и вместе они создают мощную платформу для веб-приложений.

Исходная аббревиатура LAMP породила ряд других связанных акронимов, которые используют основной фокус первоначальной комбинации технологий для создания многофункциональных веб-сайтов. Некоторые из этих связанных веб-стеков включают LAPP, MAMP и BAMP.

Аналогом LAMP для Windows Server является WAMP.

Установка нужного программного обеспечения LAMP стека происходит в привилегированном режиме, для этого необходимо авторизоваться на сервере Ubuntu под логином администратора.

На сегодняшний день веб-сервер Apache является самым популярным веб-сервером в мире. Чтобы поставить Apache для LAMP вам необходимо подключиться к своему серверу и ввести следующие команды:

```
sudo apt-get update
sudo apt-get install apache2
```

Для тестирования успешной установки сервера, перейдите в браузере по следующей ссылке, указав IP-адрес вашего компьютера:

`http://<ip-адрес_сервера>/`

Например:

`http://5.101.77.24/`

MySQL — это система управления базами данных (СУБД), которая используется для организации и получения контента. Сегодня в LAMP стеке MySQL заменяется на популярный аналог - MariaDB. MariaDB имеет те же возможности, что и MySQL, и при этом является полностью бесплатной.

*Примечание: если вы предпочитаете использовать базу данных MySQL в Ubuntu или ваш проект работает именно с ней, вам нужно будет добавить необходимые репозитории.*

Установите пакеты из главного репозитория Ubuntu с помощью менеджера пакетов:

```
sudo apt-get install mariadb-server mariadb-client
```

После успешной установки выполните настройки безопасности и удалите некоторые значения по умолчанию. Приведите в действие скрипт:

```
sudo mysql_secure_installation
```

Здесь MariaDB попросит вас задать пароль супер-пользователя БД. Затем система попросит ответить на несколько вопросов (они служебные и на каждый из них без лишних проблем можно ответить Yes).

После этого конфигурирование базы данных завершено, и можно переходить к следующему шагу настройки LAMP стека - установке PHP. Для установки компонента обработки программного кода PHP[4] установите пакеты из главного репозитория с помощью менеджера пакетов. В разных версия ОС доступны разные версии языка (например):

```
sudo apt-get install php7.0-mysql php7.0-curl php7.0-
json php7.0-cgi php7.0 libapache2-mod-php7.0 php7.0-xml
```

[4] Текущая версия php – 7.4

или

```
sudo apt-get install php5 libapache2-mod-php5 php5-
mcrypt
```

В процессе установки потребуется 1 или 2 раза ответить ‘Yes’ на вопросы системы.

Для повышения функциональности PHP вы можете дополнительно поставить некоторые модули. В большинстве случаем выбор модулей определяется системой управления контентом, устанавливаемой на веб-сервер (перечень необходимых модулей смотрите в документации к ней).

Если вы решите установить какой-либо из модулей, то это можно сделать с помощью специальной команды:

```
sudo apt-get install <имя пакета>
```

После завершения этого шага LAMP-стек будет полностью установлен на вашем сервере. Чтобы удостовериться в том, что LAMP поставлен без ошибок, можно создать тестовую страницу.

Создаем файл с нужным расширением:

```
vi /var/www/html/info.php
```

Внесите в него следующую строку и сохраните:

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

Затем сохранить и выйти.

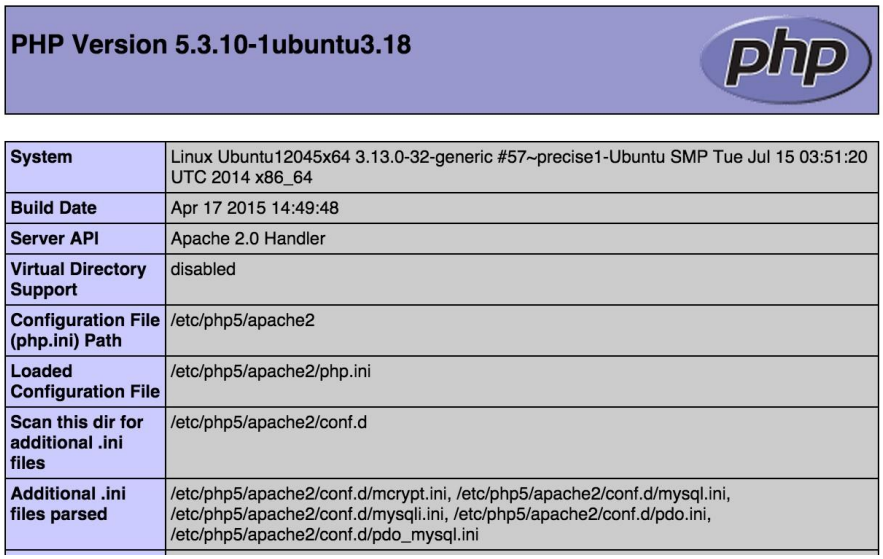
Чтобы изменения стали актуальны, нужно заново запустить веб-сервер:

```
service apache2 restart
```

Затем перейдите в браузере по следующей ссылке, указав IP-адрес вашего сервера:

```
http://<IP-адрес_сервера>/info.php
```

Ожидаемый результат:



**PHP Version 5.3.10-1ubuntu3.18**

| | |
|---|---|
| **System** | Linux Ubuntu12045x64 3.13.0-32-generic #57~precise1-Ubuntu SMP Tue Jul 15 03:51:20 UTC 2014 x86_64 |
| **Build Date** | Apr 17 2015 14:49:48 |
| **Server API** | Apache 2.0 Handler |
| **Virtual Directory Support** | disabled |
| **Configuration File (php.ini) Path** | /etc/php5/apache2 |
| **Loaded Configuration File** | /etc/php5/apache2/php.ini |
| **Scan this dir for additional .ini files** | /etc/php5/apache2/conf.d |
| **Additional .ini files parsed** | /etc/php5/apache2/conf.d/mcrypt.ini, /etc/php5/apache2/conf.d/mysql.ini, /etc/php5/apache2/conf.d/mysqli.ini, /etc/php5/apache2/conf.d/pdo.ini, /etc/php5/apache2/conf.d/pdo_mysql.ini |

## 5. Работа с MySQL сервером

### 5.1. Подсоединение к серверу и отсоединение от него[5]

При подключении к серверу с помощью mysql обычно нужно ввести имя пользователя MySQL и, в большинстве случаев, пароль:

```
shell> mysql -h host -u user -p
Enter password: ********
```

Если сервер запущен не на том компьютере, с которого вы вошли в систему, необходимо также указать имя хоста. Если все сработает, на экране должна появиться следующая информация и метка командной строки mysql>:

[5] Материал с сайта http://www.mysql.ru

```
shell> mysql -h host -u user -p
Enter password: ********

Welcome to the MySQL monitor. Commands end with ; or \g.
Your MySQL connection id is 459 to server version:
3.22.20a-log

Type 'help' for help.

mysql>
```

Метка обозначает, что программа mysql готова к вводу команд.

После установки соединения можно в любой момент отключиться от сервера, набрав в командной строке mysql> команду QUIT:

```
mysql> QUIT
Bye
```

Отсоединиться от сервера можно и при помощи сочетания клавиш Control-D.

### 5.2. Ввод запросов

В этом разделе разъясняются основные принципы ввода команд; на примере нескольких запросов вы можете поближе познакомиться с работой mysql.

Подсоединитесь к серверу.

Ниже приведена простая команда, запрашивающая у сервера информацию об его версии и текущей дате. Введите ее в командной строке mysql> и нажмите Enter:

```
mysql> SELECT VERSION(), CURRENT_DATE;
+--------------+--------------+
| VERSION()    | CURRENT_DATE |
+--------------+--------------+
| 3.22.20a-log | 1999-03-19   |
```

```
+--------------+--------------+
1 row in set (0.01 sec)
mysql>
```

Этот запрос иллюстрирует следующие особенности mysql:

Команда обычно состоит из SQL-выражения, за которым следует точка с запятой.

Когда пользователь вводит команду, mysql отправляет ее серверу для выполнения и выводит на экран сначала результаты, а затем - новую строку mysql>, что означает готовность к выполнению новых команд.

mysql выводит результаты работы запроса в виде таблицы (строк и столбцов). В первой строке этой таблицы содержатся заголовки столбцов, а в следующих строках - собственно результаты. Обычно заголовками столбцов становятся имена, полученные из таблиц базы. Если же извлекается не столбец таблицы, а значение выражения (как это происходит в приведенном выше примере), mysql дает столбцу имя запрашиваемого выражения.

mysql сообщает количество возвращаемых строк и время выполнения запроса, что позволяет в некоторой степени составить представление о производительности сервера.

Для ввода ключевых слов можно использовать любой регистр символов.

Следующий запрос демонстрирует использование mysql в качестве калькулятора:

```
mysql> SELECT SIN(PI()/4), (4+1)*5;
+-------------+---------+
| SIN(PI()/4) | (4+1)*5 |
+-------------+---------+
|    0.707107 |      25 |
+-------------+---------+
```

В одну строку можно поместить и несколько команд. Но каждая из них должна заканчиваться точкой с запятой:

```
mysql> SELECT VERSION(); SELECT NOW();
+--------------+
| VERSION()    |
+--------------+
| 3.22.20a-log |
+--------------+
+---------------------+
| NOW()               |
+---------------------+
| 1999-03-19 00:15:33 |
+---------------------+
```

Для mysql признаком завершения выражения является точка с запятой, а не конец строки (ввод данных будет прекращен только поле точки с запятой). Пример выражения, занимающего несколько строк:

```
mysql> SELECT
    -> USER()
    -> ,
    -> CURRENT_DATE;
+--------------------+--------------+
| USER()             | CURRENT_DATE |
+--------------------+--------------+
| joesmith@localhost | 1999-03-18   |
+--------------------+--------------+
```

Если вы решите отменить исполнение набираемой команды, наберите \c.

В таблице приведены все возможные варианта вида метки командной строки и соответствующие им состояния mysql:

| Метка | Значение |
|---|---|
| mysql> | Ожидание новой команды. |
| -> | Ожидание следующей строки многострочной команды. |
| '> | Ожидание следующей строки, сбор строкового выражения, начинающегося с одиночной кавычки (`"). |

| | |
|---|---|
| "> | Ожидание следующей строки, сбор строкового выражения, начинающегося с двойной кавычки (`""). |

В MySQL строки можно заключать как в одинарные (`"), так и в двойные (`"") кавычки.

### 5.3. Создание и использование базы данных

В этом разделе вы научитесь:

- создавать базу данных
- создавать таблицу
- записывать в таблицу данные
- извлекать данные из таблицы различными способами
- работать с несколькими таблицами сразу

Узнать, какие базы существуют в настоящее время на сервере, можно при помощи команды SHOW (пример):

```
mysql> SHOW DATABASES;
+----------+
| Database |
+----------+
| mysql    |
| test     |
| tmp      |
+----------+
```

Если база данных test существует, попробуйте обратиться к ней:

```
mysql> USE test
Database changed
```

В команде USE, как и QUIT, точка с запятой не нужна. Команда USE отличается от остальных и кое-чем еще: она должна задаваться одной строкой.

Создать базу данных можно следующей командой:

```
mysql> CREATE DATABASE menagerie;
```

В Unix имеет значение регистр символов в именах баз данных (в отличие от ключевых слов SQL), так что в этой ОС вам всегда придется называть свою базу menagerie, а не Menagerie, MENAGERIE или еще как-нибудь. Это же правило распространяется и на имена таблиц (в Windows данное ограничение не действует, однако при обращении к базам и таблицам в пределах одного запроса, тем не менее, можно использовать только один регистр).

При создании базы данных она автоматически не выбирается; выбирать ее нужно отдельно:

```
mysql> USE menagerie
Database changed
```

Получить список таблиц базы данных можно командой SHOW TABLES:

```
mysql> SHOW TABLES;
Empty set (0.00 sec)
```

При помощи команды CREATE TABLE определим структуру новой таблицы[6]:

```
mysql> CREATE TABLE pet (name VARCHAR(20), owner VAR-
CHAR(20),
-> species VARCHAR(20), sex CHAR(1), birth DATE, death
DATE);
```

Тип VARCHAR отлично подойдет для хранения имени животного, имени владельца и названия вида, так как длина данных этого типа может варьироваться. Конечно, длины таких столбцов вовсе не должны совпадать и не должны быть равны 20 - можно

[6] В таблице pet (домашнее животное) следующие поля: имя (name), имя владельца (owner), вид (species), пол (sex), дата рождения (birth) и дата смерти (death).

выбрать любое значение в пределах от 1 до 255 (если при выборе длины столбца вы ошибетесь, и при работе с базой окажется, что столбец маловат, можно будет исправить ошибку при помощи команды ALTER TABLE).

Пол животного можно обозначать несколькими способами, например буквами "m" и "f", или словами male (мужской) и female (женский). С буквами "m" и "f" будет проще.

Применение типа данных DATE для хранения дат рождения и смерти вполне очевидно.

Теперь, когда таблица создана, команда SHOW TABLES должна вывести следующее:

```
mysql> SHOW TABLES;
+---------------------+
| Tables in menagerie |
+---------------------+
| pet                 |
+---------------------+
```

Проверить, правильно была ли таблица создана в соответствии с планом, можно при помощи команды DESCRIBE:

```
mysql> DESCRIBE pet;
+---------+-------------+------+-----+---------+-------+
| Field   | Type        | Null | Key | Default | Extra |
+---------+-------------+------+-----+---------+-------+
| name    | varchar(20) | YES  |     | NULL    |       |
| owner   | varchar(20) | YES  |     | NULL    |       |
| species | varchar(20) | YES  |     | NULL    |       |
| sex     | char(1)     | YES  |     | NULL    |       |
| birth   | date        | YES  |     | NULL    |       |
| death   | date        | YES  |     | NULL    |       |
+---------+-------------+------+-----+---------+-------+
```

При добавлении одиночных записей используется команда INSERT:

```
mysql> INSERT INTO pet
```

```
    -> VALUES ('Puffball','Diane','hamster','f','1999-03-
30',NULL);
```

Обратите внимание на то, что здесь строковые выражения и даты представлены в виде ограниченных кавычками строк. Кроме того, в команде INSERT отсутствующие данные можно прямо заменять на NULL.

### 5.4. Типы данных столбцов

MySQL поддерживает несколько типов столбцов, которые можно разделить на три категории: числовые типы данных, типы данных для хранения даты и времени и символьные (строковые) типы данных.

Ниже перечислены типы столбцов, поддерживаемые MySQL. В описаниях используются следующие обозначения:

- M - Указывает максимальный размер вывода. Максимально допустимый размер вывода составляет 255 символов.
- D - Употребляется для типов данных с плавающей точкой и указывает количество разрядов, следующих за десятичной точкой. Максимально возможная величина составляет 30 разрядов, но не может быть больше, чем M-2.
- Квадратные скобки (`[' и `]') указывают для типа данных группы необязательных признаков.

TINYINT[(M)] [UNSIGNED] [ZEROFILL] - Очень малое целое число. Диапазон со знаком от -128 до 127. Диапазон без знака от 0 до 255.

BIT, BOOL - Являются синонимами для TINYINT(1).

SMALLINT[(M)] [UNSIGNED] [ZEROFILL] - Малое целое число. Диапазон со знаком от -32768 до 32767. Диапазон без знака от 0 до 65535.

MEDIUMINT[(M)] [UNSIGNED] [ZEROFILL] - Целое число среднего размера. Диапазон со знаком от -8388608 до 8388607. Диапазон без знака от 0 до 16777215.

INT[(M)] [UNSIGNED] [ZEROFILL] - Целое число нормального размера. Диапазон со знаком от -2147483648 до 2147483647. Диапазон без знака от 0 до 4294967295.

INTEGER[(M)] [UNSIGNED] [ZEROFILL] - Синоним для INT.

BIGINT[(M)] [UNSIGNED] [ZEROFILL] - Большое целое число. Диапазон со знаком от -9223372036854775808 до 9223372036854775807. Диапазон без знака от 0 до 18446744073709551615.

FLOAT(точность) [UNSIGNED] [ZEROFILL] - Число с плавающей точкой. Атрибут точности может иметь значение <=24 для числа с плавающей точкой обычной (одинарной) точности и между 25 и 53 - для числа с плавающей точкой удвоенной точности. Эти типы данных сходны с типами FLOAT и DOUBLE, описанными ниже. FLOAT(X) относится к тому же интервалу, что и соответствующие типы FLOAT и DOUBLE, но диапазон значений и количество десятичных знаков не определены.

FLOAT[(M,D)] [UNSIGNED] [ZEROFILL] - Малое число с плавающей точкой обычной точности. Допустимые значения: от -3,402823466E+38 до -1,175494351E-38, 0, и от 1,175494351E-38 до 3,402823466E+38. Если указан атрибут UNSIGNED, отрицательные значения недопустимы. Атрибут M указывает количество выводимых пользователю знаков, а атрибут D - количество разрядов, следующих за десятичной точкой. Обозначение FLOAT без указания аргументов или запись вида FLOAT(X), где X <=24 справедливы для числа с плавающей точкой обычной точности.

DOUBLE[(M,D)] [UNSIGNED] [ZEROFILL] - Число с плавающей точкой удвоенной точности нормального размера. Допустимые значения: от -1,7976931348623157E+308 до -2,2250738585072014E-308, 0, и от 2,2250738585072014E-308 до 1,7976931348623157E+308. Если указан атрибут UNSIGNED,

отрицательные значения недопустимы. Атрибут M указывает количество выводимых пользователю знаков, а атрибут D - количество разрядов, следующих за десятичной точкой. Обозначение DOUBLE без указания аргументов или запись вида FLOAT(X), где 25 <= X <= 53 справедливы для числа с плавающей точкой двойной точности.

DOUBLE PRECISION[(M,D)] [UNSIGNED] [ZEROFILL], REAL[(M,D)] [UNSIGNED] [ZEROFILL] - Данные обозначения являются синонимами для DOUBLE.

DECIMAL[(M[,D])] [UNSIGNED] [ZEROFILL] - Ведет себя подобно столбцу CHAR, содержащему цифровое значение (число хранится в виде строки).

DEC[(M[,D])] [UNSIGNED] [ZEROFILL], NUMERIC[(M[,D])] [UNSIGNED] [ZEROFILL] - Данные обозначения являются синонимами для DECIMAL.

DATE - Дата. Поддерживается интервал от '1000-01-01' до '9999-12-31'. MySQL выводит значения DATE в формате 'YYYY-MM-DD', но можно установить значения в столбец DATE, используя как строки, так и числа.

DATETIME - Комбинация даты и времени. Поддерживается интервал от '1000-01-01 00:00:00' до '9999-12-31 23:59:59'. MySQL выводит значения DATETIME в формате 'YYYY-MM-DD HH:MM:SS'.

TIMESTAMP[(M)] - Временная метка. Интервал от '1970-01-01 00:00:00' до некоторого значения времени в 2037 году. MySQL выводит значения TIMESTAMP в форматах YYYYMMDDHHMMSS, YYMMDDHHMMSS, YYYYMMDD или YYMMDD в зависимости от значений M: 14 (или отсутствующее), 12, 8, или 6; но можно также устанавливать значения в столбце TIMESTAMP, используя как строки, так и числа..

TIME - Время. Интервал от '-838:59:59' до '838:59:59'. MySQL выводит значения TIME в формате 'HH:MM:SS'.

YEAR[(2|4)] - Год в двухзначном или четырехзначном форматах (по умолчанию формат четырехзначный). Допустимы следующие значения: с 1901 по 2155, 0000 для четырехзначного формата года и 1970-2069 при использовании двухзначного формата (70-69).

[NATIONAL] CHAR(M) [BINARY] - Строка фиксированной длины, при хранении всегда дополняется пробелами в конце строки до заданного размера. Диапазон аргумента M составляет от 0 до 255 символов (от 1 до 255 в версиях, предшествующих MySQL 3.23). Концевые пробелы удаляются при выводе значения. Если не задан атрибут чувствительности к регистру BINARY, то величины CHAR сортируются и сравниваются как независимые от регистра в соответствии с установленным по умолчанию алфавитом. Атрибут NATIONAL CHAR (или его эквивалентная краткая форма NCHAR) представляет собой принятый в ANSI SQL способ указания, что в столбце CHAR должен использоваться установленный по умолчанию набор символов (CHARACTER). В MySQL это принято по умолчанию. CHAR является сокращением от CHARACTER.

CHAR - Это синоним для CHAR(1).

[NATIONAL] VARCHAR(M) [BINARY] - Строка переменной длины. Диапазон аргумента M составляет от 0 до 255 символов (от 1 до 255 в версиях, предшествующих MySQL Version 4.0.2).

TINYBLOB, TINYTEXT - Столбец типа BLOB или TEXT с максимальной длиной 255 (2^8 - 1) символов.

BLOB, TEXT - Столбец типа BLOB или TEXT с максимальной длиной 65535 (2^16 - 1) символов..

MEDIUMBLOB, MEDIUMTEXT - Столбец типа BLOB или TEXT с максимальной длиной 16777215 (2^24 - 1) символов.

LONGBLOB, LONGTEXT - Столбец типа BLOB или TEXT с максимальной длиной 4294967295 (2^32 - 1) символов.

ENUM('значение1','значение2',...) - Перечисление. Перечисляемый тип данных. Объект строки может иметь только одно значение, выбранное из заданного списка величин 'значение1',

'значение2', ..., NULL или специальная величина ошибки "". Список ENUM может содержать максимум 65535 различных величин.

SET('значение1','значение2',...) - Набор. Объект строки может иметь ноль или более значений, каждое из которых должно быть выбрано из заданного списка величин 'значение1', 'значение2', ... Список SET может содержать максимум 64 элемента.

## 6. Пояснения к выполнению работы

6.1. Исходные данные (предметная область): подсистема учета персонала на предприятии.

6.2. Основные сущности (таблицы с указанием полей):

СОТРУДНИК (Табельный номер, Имя сотрудника);

ОТДЕЛ (Номер отдела, Имя отдела);

ЗАКАЗЧИК (Имя заказчика, Адрес, ИНН, Валюта, Язык, Форма собственности);

КОНТРАКТ (Номер, Срок начала, Срок окончания, Сумма);

РАБОЧАЯ ГРУППА (Процент вознаграждения)

6.3. Исходные таблицы

6.3.1. Исходная таблица «Сотрудник» с примером заполнения данными

| Табельный номер | Имя сотрудника |
|---|---|
| 12 | Иванов Иван Владимирович |
| 32 | Петров Сергей Павлович |
| 34 | Костюков Петр Алексеевич |
| 45 | Данилова Ольга Сергеевна |

6.3.2. Исходная таблица «Отдел» с примером заполнения данными

| Номер отдела | Имя отдела |
|---|---|
| 1 | Бухгалтерия |
| 2 | Финансовый отдел |
| 3 | Отдел маркетинговых исследований |
| 4 | Дирекция |
| 5 | Канцелярия |

6.3.3. Исходная таблица «Заказчик» с примером заполнения данными

| Имя Заказчика | Адрес | ИНН | Валюта | Язык | Форма собственности |
|---|---|---|---|---|---|
| ОАО «Агат» | Пр. Мира, 30 | 5323454321 | Руб. | Русский | Коллективная |
| ЧП «Петров» | Ул. Ленина, 5 | 5222454545 | Руб. | Русский | Частная |
| НПО «Мир» | ул. Ленина, 12 | 4525221525 | Руб., евро | Русский, английский | Государственная |

6.3.4. Исходная таблица «Контракт» с примером заполнения данными

| Номер | Срок начала | Срок окончания | Сумма |
|---|---|---|---|
| 1 | 12.01.2003 | 28.02.2004 | 1 245 125 |
| 2 | 01.12.2003 | 30.05.2004 | 2 000 500 |
| 3 | 01.02.2004 | 30.06.2004 | 1 500 500 |

6.3.5. Исходная таблица «Рабочая группа» с примером заполнения данными

| Процент вознаграждения |
|---|
| 12 |
| 20 |
| 45 |
| 10 |
| 5 |
| 11 |

6.4. После нормализации[7] (оптимизации) получаем таблицы следующего содержания.

6.4.1. Измененная таблица «Сотрудник» с примером заполнения данными

| Табельный номер | Имя сотрудника | Номер отдела |
|---|---|---|
| 12 | Иванов Иван Владимирович | 1 |
| 32 | Петров Сергей Павлович | 2 |
| 34 | Костюков Петр Алексеевич | 2 |
| 45 | Данилова Ольга Сергеевна | 3 |

6.4.2. Измененная таблица «Состав рабочей группы» с примером заполнения данными

[7] Нормализация не является частью данной работы, поэтому далее приводится только результат. При реализации самостоятельных заданий, качество нормализации (оптимизации размещения данных в таблицах, связывание таблиц) не оценивается.

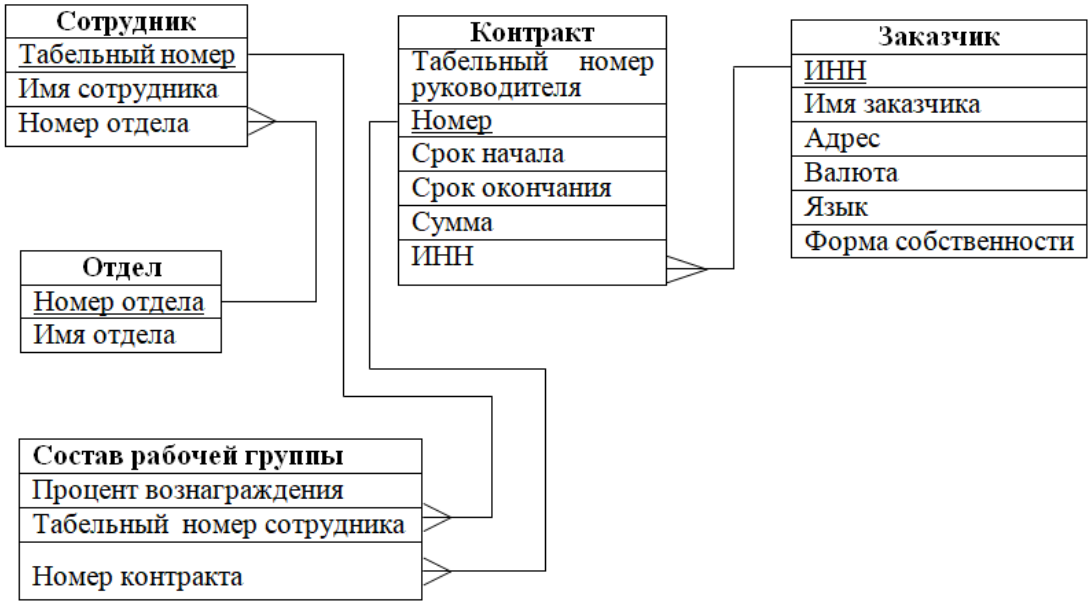
| Процент вознаграждения | Номер контракта | Табельный номер сотрудника |
|---|---|---|
| 12 | 1 | 12 |
| 20 | 1 | 32 |
| 45 | 2 | 34 |
| 10 | 2 | 45 |
| 5 | 2 | 32 |
| 11 | 3 | 12 |

6.4.3. Измененная таблица «Контракт» с примером заполнения данными

| Номер | Срок начала | Срок окончания | Сумма | ИНН | Табельный номер руководителя |
|---|---|---|---|---|---|
| 1 | 12.01.2003 | 28.02.2004 | 1 245 125 | 5323454321 | 32 |
| 2 | 01.12.2003 | 30.05.2004 | 2 000 500 | 5323454321 | 45 |
| 3 | 01.02.2004 | 30.06.2004 | 1 500 500 | 4525221525 | 12 |

3.4.4. Таблицы «Отдел» и «Заказчик» - без изменений.

6.5. Связи между таблицами и их состав приведены на рисунке.




6.6. Спецификации атрибутов для таблиц, типов данных, которые будут приписаны атрибутам.

6.6.1. Типы данных полей таблицы «Сотрудник»

| Имя поля | Тип поля | Свойства |
|---|---|---|
| Табельный номер | Счетчик | Индексированное поле – Да (Совпадения не допускаются) |
| Имя сотрудника | Текстовый | Размер поля – 50. Обязательное поле – Да |
| Номер отдела | Числовой | Размер поля – Целое. Индексированное поле – Да (Совпадения допускаются) |

6.6.2. Типы данных полей таблицы «Контракт»

| Имя поля | Тип поля | Свойства |
|---|---|---|
| Номер | Счетчик | Индексированное поле – Да. (Совпадения не допускаются) |
| Срок начала | Дата | |

| | | |
|---|---|---|
| Срок окончания | Дата | |
| Сумма | Денежный | |
| ИНН | Текстовый | Размер поля – 12 |
| Табельный номер руководителя | Числовой | Размер поля – Целое. Индексированное поле – Да (Совпадения допускаются) |

6.6.3. Типы данных полей таблицы «Отдел»

| Имя поля | Тип поля | Свойства |
|---|---|---|
| Номер отдела | Счетчик | Индексированное поле – Да. (Совпадения не допускаются) |
| Имя отдела | Текстовый | Размер поля – 50. Обязательное поле – Да |

6.6.4. Типы данных полей таблицы «Состав рабочей группы»

| Имя поля | Тип поля | Свойства |
|---|---|---|
| Процент вознаграждения | Числовой | Размер поля – Одинарное с плавающей точкой<br>Формат поля – Фиксированный<br>Число десятичных знаков – 2 |
| Номер контракта | Числовой | Размер поля – Длинное целое.<br>Индексированное поле – Да (Совпадения допускаются) |
| Табельный номер сотрудника | Числовой | Размер поля – Целое<br>Индексированное поле – Да (Совпадения допускаются) |

6.6.5. Типы данных полей таблицы «Заказчик»

| Имя поля | Тип поля | Свойства |
|---|---|---|
| ИНН | Текстовый | Индексированное поле – Да. (Совпадения не допускаются) |
| Имя заказчика | Текстовый | Размер поля – 50. Обязательное поле – Да |
| Адрес | Текстовый | Размер поля – 200. Обязательное поле – Да |
| Валюта | Текстовый | Размер поля – 6 |
| Язык | Текстовый | Размер поля – 15 |
| Форма собственности | Текстовый | Размер поля – 20. Обязательное поле – Да |

## 7. Варианты заданий к лабораторной работе

7.1. Предметная область: «БД службы проката бытовых приборов». Функция: «Выдача бытовых приборов в прокат». Таблицы:

Клиент: фамилия имя отчество, адрес с почтовым, индексом и названием города, номер паспорта, номер домашнего телефона и сотового.

Сотрудник: фамилия имя отчество, должность, домашний адрес.

Товар: название товара, тип товара, дата ввода в аренду, состояние, правила эксплуатации.

Документы на выдачу: (определить самостоятельно).

7.2. Предметная область: «БД службы проката бытовых приборов». Функция: «Возврат бытовых приборов в прокат». Таблицы:

Клиент: фамилия имя отчество, адрес с почтовым, индексом и названием города, номер паспорта, номер домашнего телефона и сотового.

Сотрудник: фамилия имя отчество, должность, домашний адрес.

Товар: название товара, тип товара, дата ввода в аренду, состояние, правила эксплуатации.

Документы на возврат: (определить самостоятельно).

7.3. Предметная область: «БД службы проката бытовых приборов». Функция: «Заказ нового товара для службы проката». Таблицы:

Сотрудник: фамилия имя отчество, должность, домашний адрес.

Товар: название товара, тип товара, дата ввода в аренду, состояние, правила эксплуатации.

Документ на заказ нового товара: код заказа, номер документа, дата документа, цена, номер товара, номер сотрудника, количество.

Документ на поставку нового товара: код поставки, номер документа, дата документа, номер сотрудника.

7.4. Предметная область: «БД службы проката бытовых приборов». Функция: «Списание устаревшего товара в службе проката». Таблицы:

Сотрудник: фамилия имя отчество, должность, домашний адрес.

Товар: название товара, тип товара, дата ввода в аренду, состояние, правила эксплуатации.

Документ на списание: номер документа, дата документа, товар, сотрудник, причина.

7.5. Предметная область: «БД службы проката бытовых приборов». Функция: «Оплата товаров, взятых в аренду». Таблицы:

Клиент: фамилия имя отчество, адрес с почтовым, индексом и названием города, номер паспорта, номер домашнего телефона и сотового.

Сотрудник: фамилия имя отчество, должность, домашний адрес.

Товар: название товара, тип товара, дата ввода в аренду, состояние, правила эксплуатации.

Оплата: период оплаты, дата оплаты, товар, сумма.

7.6. Предметная область: «БД службы проката бытовых приборов». Функция: «Оформление штрафа за товар, взятый в аренду с просроченным сроком оплаты». Таблицы:

Клиент: фамилия имя отчество, адрес с почтовым, индексом и названием города, номер паспорта, номер домашнего телефона и сотового.

Сотрудник: фамилия имя отчество, должность, домашний адрес.

Товар: название товара, тип товара, дата ввода в аренду, состояние, правила эксплуатации.

Оплата: период оплаты, дата оплаты, товар, сумма, клиент.

Штраф: дата выписки штрафа, причина, товар, клиент, сумма штрафа.

7.7. Предметная область: «БД Документооборота организации». Функция: «Заключение договора на выполнение работ». Таблицы:

Заказчики: название организации заказчика, контактное лицо (фамилия имя отчество), адрес организации с почтовым индексом и названием города, телефон городской, телефон сотовый, ИНН.

Сотрудники организации: фамилия, имя, отчество, домашний адрес, телефон городской, телефон сотовый, дата рождения.

Договор: номер договора, дата заключения договора, заказчик, объект, фамилия ответственного исполнителя, предмет договора, перечень работ, сроки выполнения, сумма договора, перечень отчетной документации.

## 8. Отчетность по лабораторной работе

8.1. В качестве приложения к отчету представить дамп базы данных из примера и из выбранного варианта заданий.

# ЛАБОРАТОРНАЯ РАБОТА №2. ВЗАИМОДЕЙСТВИЕ С БАЗОЙ ДАННЫХ (КОННЕКТОРЫ)

## 1. Цель работы

1.1. Изучение процесса взаимодействия с базой данных, выполнения запросов
(использование коннекторов).

Работа реализуется на языке программирования php, взаимодействие осуществляется
с СУБД MySQL.

## 2. Содержание работы

2.1. Установить и настроить LAMP или WAMP сервер (могли быть установлены в ходе выполнения лабораторной работы №1).

2.2. Используя расширение mysqli языка программирования php реализовать:

1) подключение к базе данных (созданной в ходе выполнения лабораторной работы №1);
2) выбрать значения из всех таблиц базы данных (данные должны быть представлены в виде нескольких таблиц на одной web-странице);
3) вывести значения одного столбца таблицы (таблицу и столбец выбрать произвольно) в порядке возрастания (ASC) и убывания (DESK);
4) вывести значения таблицы, удовлетворяющие определенному условию (таблицу и условие выбрать произвольно);
5) вывод диагностических сообщений об ошибках (как минимум: сообщение об отсутствии подключения к БД, об ошибке при выполнении запроса).
6) при отсутствии данных в таблице, выводить сообщение «В таблице … данные отсутствуют».

2.3. Реализовать задание п.2.2. используя модуль PDO_MYSQL.

### 3. Работе с расширением mysqli

Mysqli (MySQL Improved) - это расширение PHP, которое добавляет в язык полную поддержку баз данных MySQL.

Перед началом работы с данными внутри MySQL, нужно открыть соединение с сервером СУБД. В PHP это делается с помощью стандартной функции mysqli_connect(). Функция возвращает результат - ресурс соединения. Данный ресурс используется для всех следующих операций с MySQL.

Но чтобы выполнить соединение с сервером, необходимо знать как минимум три параметра:

- Адрес сервера СУБД;
- Логин;
- Пароль.

Базовый синтаксис функции **mysqli_connect():**

```
mysqli_connect(<адрес сервера>, <имя пользователя>, <пароль>, <имя базы данных>);
```

После соединения с СУБД необходимо выполнить проверку, что соединение было установлено. Соединение с MySQL устанавливается один раз в сценарии, а затем используется при всех запросах к БД.

Если подключение к MySQL не удалось, то функция **mysqli_connect()** вместо ресурса вернет логическое значение типа «ложь» - false.

Соединение с MySQL и проверка на ошибки:

```php
<?php
$link = mysqli_connect("localhost", "root", "");

if ($link == false){
    print("Ошибка: Невозможно подключиться к MySQL " . mysqli_connect_error());
}
else {
    print("Соединение установлено успешно");
}
```

Функция **mysqli_connect_error()** возвращает текстовое описание последней ошибки MySQL.

После установки соединения крайне желательно явно задать кодировку, которая будет использоваться при обмене данными с MySQL. Для этого используется команда:

```
mysqli_set_charset($link, «utf8»);
```

Все SQL-запросы можно разделить на две группы (запросы передаются через функцию **mysqli_query()**):

- Чтение информации (SELECT).
- Модификация (UPDATE, INSERT, DELETE).

При выполнении запросов из среды PHP, запросы из

- Первой группы возвращают результат запроса – ресурс результата. Может быть преобразован в ассоциативный массив или в массив.
- Второй группы возвращают результат их исполнения: успех или ошибку.

Выражение INSERT INTO используется для добавления новых записей в таблицу базы данных.

```php
<?php
$link = mysqli_connect("localhost", "root", "");

$sql = 'INSERT INTO cities SET name = "Санкт-Петербург"';
$result = mysqli_query($link, $sql);

if ($result == false) {
    print("Произошла ошибка при выполнении запроса");
}
```

Первым параметром для функциии **mysqli_query()** передаётся ресурс подключения, полученный от функции **mysqli_connect()**, вторым параметром следует строка с SQL-запросом.

Функция **mysqli_insert_id()** используется для получения идентификатора добавленной записи. В качестве аргумента - ресурс соединения.

Комплексный пример с подключением к MySQL и добавлением двух новых записей:

```php
<?php
$link = mysqli_connect("localhost", "root", "");

if ($link == false){
    print("Ошибка: Невозможно подключиться к MySQL " . mysqli_connect_error());
}
else {
    $sql = 'INSERT INTO cities SET name = "Санкт-Петербург"';
    $result = mysqli_query($link, $sql);

    if ($result == false) {
        print("Произошла ошибка при выполнении запроса");
    }
    else {
        $city_id = mysqli_insert_id($link);

        $sql = 'INSERT INTO weather_log SET city_id = ' . $city_id . ', day = "2017-09-03",
temperature = 10, cloud = 1';

        $result = mysqli_query($link, $sql);

        if ($result == false) {
            print("Произошла ошибка при выполнении запроса");
        }
    }
}
```

Частая операция при работе с базами данных - это получение записей из таблиц (запросы типа SELECT).

В примере показано, как вывести все существующие города из таблицы cities:

```php
<?php

$sql = 'SELECT id, name FROM cities';

$result = mysqli_query($link, $sql);

while ($row = mysqli_fetch_array($result)) {
    print("Город: " . $row['name'] . "; Идентификатор: . " . $row['id'] . "<br>");
}
```

В примере результат выполнения функции **mysqli_query()** сохранён в переменной $result.

В переменной результата находятся не данные из таблицы, а специальный тип данных - ссылка на результаты запроса. Чтобы получить действительные данные, то есть записи из таблицы, следует использовать другую функцию - **mysqli_fetch_array()** и передать ей ресурс результата. Каждый вызов функции **mysqli_fetch_array()** возвращает следующую запись из всего результирующего набора записей. Будет возращен ассоциативный массив.

Функция **mysqli_fetch_row()** используется для получения данных в виде массива.

Функция **mysqli_fetch_all($res, MYSQLI_ASSOC)** вернёт двумерный массив со всеми записями из результата последнего запроса.

```php
<?php

$sql = 'SELECT id, name FROM cities';
$result = mysqli_query($link, $sql);

$rows = mysqli_fetch_all($result, MYSQLI_ASSOC)

foreach ($rows as $row) {
    print("Город: " . $row['name'] . "; Идентификатор: . " . $row['id'] . "<br>");
}
```

Число записей результата можно получить с помощью функции **mysqli_num_rows()**.

Так как обычно подключения к БД используются множеством скриптов, то нередко настройки подключения выносятся в отдельный файл, благодаря чему легче их оперативно изменять.

Итак, создадим файл connection.php и добавим в его следующие строки:

```php
<?php
$host = 'localhost'; // адрес сервера
$database = 'compstore'; // имя базы данных
$user = 'root'; // имя пользователя
$password = '1234567'; // пароль
?>
```

Теперь мы можем подключиться к базе данных:

```php
<?php
require_once 'connection.php'; // подключаем скрипт

// подключаемся к серверу
$link = mysqli_connect($host, $user, $password, $database)
    or die("Ошибка " . mysqli_error($link));

// выполняем операции с базой данных

// закрываем подключение
mysqli_close($link);
?>
```

Первым делом подключаем скрипт с настройками с помощью инструкции **require_once.**

Для открытия подключения применяем функцию **mysqli_connect().** После окончания работы подключение нужно закрыть. Для этого применяется функция **mysqli_close(),** которая в качестве параметра принимает объект подключения.

Рассмотрим пример добавления данных в БД. Создадим файл create.php со следующим содержимым:

```
<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8">
</head>
<body>
<?php
```

```
    require_once 'connection.php'; // подключаем скрипт

    if(isset($_POST['name']) && isset($_POST['company'])){

        // подключаемся к серверу
        $link = mysqli_connect($host, $user, $password, $data-
base)
            or die("Ошибка " . mysqli_error($link));

        // экранирования символов для mysql
        $name = htmlentities(mysqli_real_escape_string($link,
$_POST['name']));
        $company = htmlenti-
ties(mysqli_real_escape_string($link, $_POST['company']));

        // создание строки запроса
        $query ="INSERT INTO tovars VALUES(NULL,
'$name','$company')";

        // выполняем запрос
        $result = mysqli_query($link, $query) or die("Ошибка "
. mysqli_error($link));
        if($result)
        {
            echo "<span style='color:blue;'>Данные
добавлены</span>";
        }
        // закрываем подключение
        mysqli_close($link);
    }
    ?>
    <h2>Добавить новую модель</h2>
    <form method="POST">
    <p>Введите модель:<br>
    <input type="text" name="name" /></p>
    <p>Производитель: <br>
    <input type="text" name="company" /></p>
    <input type="submit" value="Добавить">
    </form>
    </body>
    </html>
```

Здесь код взаимодействия с базой данных объединен с функциональностью форм: с помощью формы мы вводим данные для добавления в БД.

Чтобы исключить возможность использования XSS-атак и SQL-инъекций в примере используются функции **mysqli_real_escape_string()**[8] и **htmlentities()**[9].

Рассмотрим пример получения данных из БД. Создадим страницу index.php, которая будет выводить значения из таблицы tovars.

```
<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8">
</head>
<body>
<?php
require_once 'connection.php'; // подключаем скрипт

$link = mysqli_connect($host, $user, $password, $database)
    or die("Ошибка " . mysqli_error($link));

$query ="SELECT * FROM tovars";

$result = mysqli_query($link, $query) or die("Ошибка " .
mysqli_error($link));
if($result)
{
    $rows = mysqli_num_rows($result); // количество полу-
ченных строк

    echo "<ta-
"<ta-
ble><tr><th>Id</th><th>Модель</th><th>Производитель</th></tr>";
    for ($i = 0 ; $i < $rows ; ++$i)
    {
```

[8] Эта функция используется для создания допустимых в SQL строк, которые можно использовать в SQL выражениях.

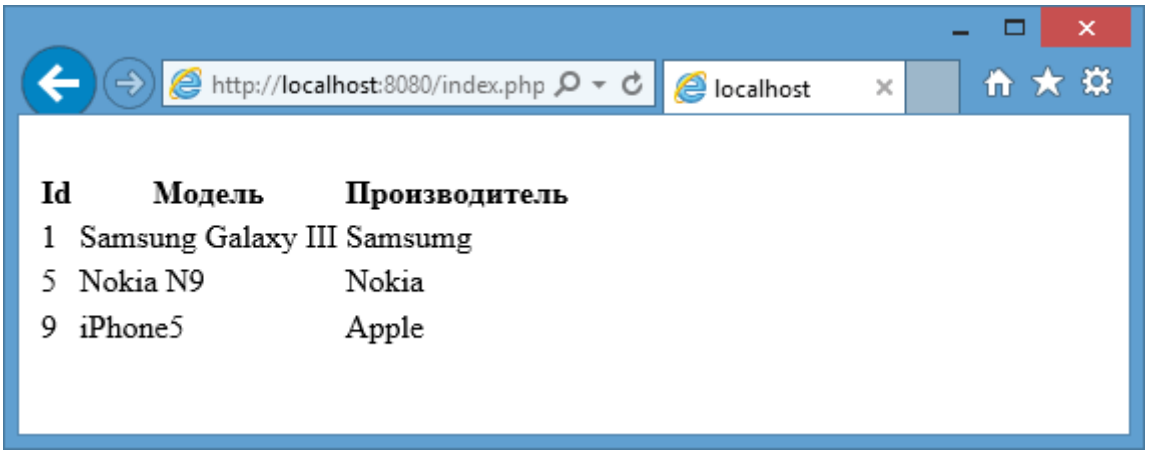
[9] Преобразует все символы в соответствующие HTML-сущности (для тех символов, для которых HTML-сущности существуют).

```
            $row = mysqli_fetch_row($result);
            echo "<tr>";
                for ($j = 0 ; $j < 3 ; ++$j) echo
"<td>$row[$j]</td>";
            echo "</tr>";
        }
        echo "</table>";

        // очищаем результат
        mysqli_free_result($result);
    }

    mysqli_close($link);
    ?>
    </body>
    </html>
```

Пример результата выполнения запроса.




Переменная $result будет хранить данные после выполнения запроса Для очистки памяти от ненужных данных можно использовать функцию **mysqli_free_result().**

Редактирование уже имеющихся данных осуществляется обычно в два этапа: получение редактируемых данных и обновление базы данных новыми значениями. Для обновления применяется выражение SQL "UPDATE".

Пример обновления данных. Создадим новый файл edit.php:

```
<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8">
</head>
<body>
<?php
require_once 'connection.php'; // подключаем скрипт
// подключаемся к серверу
$link = mysqli_connect($host, $user, $password, $database)
        or die("Ошибка " . mysqli_error($link));

// если запрос POST
if(isset($_POST['name']) && isset($_POST['company']) &&
isset($_POST['id'])){

        $id = htmlentities(mysqli_real_escape_string($link,
$_POST['id']));
        $name = htmlentities(mysqli_real_escape_string($link,
$_POST['name']));
        $company = htmlenti-
ties(mysqli_real_escape_string($link, $_POST['company']));

        $query ="UPDATE tovars SET name='$name', compa-
ny='$company' WHERE id='$id'";
        $result = mysqli_query($link, $query) or die("Ошибка "
. mysqli_error($link));

        if($result)
            echo "<span style='color:blue;'>Данные
обновлены</span>";
}

// если запрос GET
if(isset($_GET['id']))
{
        $id = htmlentities(mysqli_real_escape_string($link,
$_GET['id']));

        // создание строки запроса
        $query ="SELECT * FROM tovars WHERE id = '$id'";
```

```
        // выполняем запрос
        $result = mysqli_query($link, $query) or die("Ошибка "
. mysqli_error($link));
        //если в запросе более нуля строк
        if($result && mysqli_num_rows($result)>0)
        {
            $row = mysqli_fetch_row($result); // получаем
первую строку
            $name = $row[1];
            $company = $row[2];

            echo "<h2>Изменить модель</h2>
                <form method='POST'>
                <input type='hidden' name='id' value='$id' />
                <p>Введите модель:<br>
                <input type='text' name='name' value='$name'
/></p>
                <p>Производитель: <br>
                <input type='text' name='company' val-
ue='$company' /></p>
                <input type='submit' value='Сохранить'>
                </form>";

            mysqli_free_result($result);
        }
    }
    // закрываем подключение
    mysqli_close($link);
    ?>
    </body>
    </html>
```

Данные передаются через GET запрос. При обработке запроса GET получаем id записи и по ней получаем данные нужной записи.

Чтобы отредактировать первую запись, нужно обратиться в браузере к скрипту по адресу http://localhost:8080/edit.php?id=1.[10]

При удалении данных используется выражение SQL "DELETE".

```
$query ="DELETE FROM tovars WHERE id = '5'";
```

[10] Localhost – адрес компьютера на котором установлен web-сервер.

В данном случае удаляется строка, у которой id=5.

При удалении, как и при редактировании, нам надо передать скрипту id удаляемой записи и выполнить запрос. Создадим скрипт delete.php со следующим содержимым:

```php
<?php
require_once 'connection.php';
if(isset($_GET['id']))
{
    $link = mysqli_connect($host, $user, $password, $database)
            or die("Ошибка " . mysqli_error($link));
    $id = mysqli_real_escape_string($link, $_GET['id']);

    $query ="DELETE FROM tovars WHERE id = '$id'";

    $result = mysqli_query($link, $query) or die("Ошибка " . mysqli_error($link));
    mysqli_close($link);
}
?>
```

Обратимся к скрипту с запросом http://localhost:8080/delete.php?id=1, и скрипт удалит первую строку в таблице.

### 4. Работе с расширением php pdo

PDO (PHP Data Objects) - расширение PHP, которое реализует взаимодействие с базами данных при помощи объектов. В данном случае отсутствует привязка к конкретной системе управления базами данных. PDO поддерживает СУБД: MySQL, PostgreSQL, SQLite, Oracle, Microsoft SQL Server и другие.

Подключение к серверу БД через конструктор PDO.

```php
$dbh = new PDO('mysql:dbname=db_name;host=localhost', 'логин', 'пароль');
```

Параметры конструктора PDO - параметры подключения. Например, SQL запрос, который будет выполнен сразу после подключения:

```
$dbh = new PDO('
   mysql:dbname=db_name;host=localhost',
   'логин',
   'пароль',
   array(PDO::MYSQL_ATTR_INIT_COMMAND => "SET NAMES 'utf8'")
);
```

Чтобы отследить ошибку подключения к БД используется исключение:

```
try {
   $dbh = new PDO('mysql:dbname=db_name;host=localhost', 'логин', 'пароль');
} catch (PDOException $e) {
   die($e->getMessage());
}
```

```
SQLSTATE[HY000] [1045] Access denied for user 'логин'@'localhost' (using password: YES)
```

PDO позволяет использовать в запросах псевдопеременные чтобы исключить SQL инъекции - вместо данных указывается «?» или «:id», а в методе execute() указываются реальные значения этих переменных.

```
$sth = $dbh->prepare("SELECT * FROM `category` WHERE `id` = ?");
$sth->execute(array('21'));
$array = $sth->fetch(PDO::FETCH_ASSOC);
print_r($array);
```

```
$sth = $dbh->prepare("SELECT * FROM `category` WHERE `id` = :id");
$sth->execute(array('id' => '21'));
$array = $sth->fetch(PDO::FETCH_ASSOC);
print_r($array);
```

Для выборки множества записей можно использовать ассоциативный массив.

```
$sth = $dbh->prepare("SELECT * FROM `category` ORDER BY `name`");
$sth->execute();
$array = $sth->fetchAll(PDO::FETCH_ASSOC);
print_r($array);
```

Можно получить данные о структуре таблицы.

```
$sth = $dbh->prepare("SHOW COLUMNS FROM `category`");
$sth->execute();
$array = $sth->fetchAll(PDO::FETCH_ASSOC);
print_r($array);
```

Добавление записей в БД:

```
$sth = $dbh->prepare("INSERT INTO `category` SET `parent` = :parent, `name` = :name");
$sth->execute(array('parent' => 1, 'name' => 'Виноград'));

// Получаем id вставленной записи
$insert_id = $dbh->lastInsertId();
```

Изменение записей:

```
$sth = $dbh->prepare("UPDATE `category` SET `name` = :name WHERE `id` = :id");
$sth->execute(array('name' => 'Виноград', 'id' => 22));
```

Удаление из БД:

```
$count = $dbh->exec("DELETE FROM `category` WHERE `parent` = 1");
echo 'Удалено ' . $count . ' строк.';
```

Или

```php
$sth = $dbh->prepare("DELETE FROM `category` WHERE `parent` = :parent");
$sth->execute(array('parent' => 1));
```

В PDO есть метод **errorInfo()** который возвращает сведенья об ошибке последнего запроса.

```php
// Таблицы `category_new` нет в БД.
$sth = $dbh->prepare("INSERT INTO `category_new` SET `parent` = :parent, `name` = :name");
$sth->execute(array('parent' => 1, 'name' => 'Виноград'));

$info = $sth->errorInfo();
print_r($info);
```

## 5. Операторы SQL

Назначение оператора SELECT состоит в выборке и отображении данных одной или более таблиц базы данных.

Общий формат оператора SELECT имеет следующий вид:

```
SELECT [DISTINCT | ALL] {* | [column_expression [AS
new_name]] [, . . . ]}
FROM table_name [alias] [, . . . ]
[WHERE condition]
[GROUP BY column_list] [HAVING condition]
[ORDER BY column_list]
```

Здесь:

- Параметр column_expression представляет собой имя столбца или выражение из нескольких имен.
- Параметр table_name является именем существующих в базе данных таблицы или представления, к которым необходимо получить доступ.
- Необязательный параметр alias — это сокращение, устанавливаемое для имени таблицы table_name.

Обработка элементов оператора выполняется в следующей последовательности:

- FROM - Определяются имена используемой таблицы или нескольких таблиц.
- WHERE - Выполняется фильтрация строк объекта в соответствии с заданными условиями.
- GROUP BY - Образуются группы строк, имеющих одно и то же значение в указанном столбце.
- HAVING - Фильтруются группы строк объекта в соответствии с указанным условием.
- SELECT - Устанавливается, какие столбцы должны присутствовать в выходных данных.
- ORDER BY - Определяется упорядоченность результатов выполнения оператора.

Порядок предложений и фраз в операторе SELECT не может быть изменен. Только два предложения оператора - SELECT и FROM являются обязательными, все остальные предложения и фразы могут быть опущены. Операция SELECT является закрытой: результат запроса к таблице представляет собой другую таблицу.

Поскольку выборка всех имеющихся в таблице столбцов выполнятся достаточно часто, в языке SQL определен упрощенный вариант записи значения “все столбы” - вместо имен столбцов указывается символ звездочки (*). Таким образом в качестве столбцов могут быть переданы названия столбцов таблиц, символ *, вычисляемые выражения на базе столбцов таблиц.

Например, отчет о ежемесячной зарплате всего персонала с указанием табельного номера, имени, фамилии и суммы зарплаты.

```
SELECT firstname, lastname, salary/12
FROM staff;
```

Результат выполнения запроса может содержать дублирующиеся значения, поскольку, в отличие от операции реляционной алгебры оператор SELECT не исключает дублирующихся значений при выполнении проекции одного или нескольких столбцов. Для удаления из

результирующей таблицы дублирующихся строк используется ключевое слово DISTINCT. Например, запрос выглядит следующим образом:

```
SELECT DISTINCT lastname
FROM staff;
```

Обычно столбцам результирующей таблицы присваиваются имена соответствующих им столбцов исходных таблиц базы данных. Однако в данном случае с вычисляемыми полями правило неприменимо. В одних диалектах языка SQL имена таким столбцам присваивают в соответствии с порядком их расположения в таблице (например, col4), в других диалектах у подобного столбца имя может вовсе отсутствовать или вместо него может использоваться выражение, помещенное в список SELECT.

Стандарт ISO позволяет явным образом задавать имена столбцов результирующей таблицы, для чего применяется фраза AS:

```
SELECT firstname, lastname, salary/12 AS monthly_salary
FROM staff;
```

В этих выражениях могут применяться операции сложения, вычитания, умножения и деления. При построении сложных выражений могут использоваться скобки. Для получения значения вычисляемого поля могут использоваться значения из нескольких столбцов таблицы, однако тип данных тех столбцов, которые входят в арифметические выражения, обязательно должен быть цифровым.

Часто требуется тем или иным образом ограничить набор строк, помещаемых в результирующую таблицу запроса. Это достигается с помощью указания в запросе предложения WHERE. Оно состоит из ключевого слова WHERE, за которым следует перечень условий поиска, определяющих те строки, которые должны быть выбраны при выполнении запроса.

Существует пять основных типов условий поиска (или предикатов, если пользоваться терминологией ISO):

- Сравнение. Сравниваются результаты вычисления одного выражения с результатами вычисления другого выражения.
- Диапазон. Проверяется, попадает ли результат вычисления выражения в заданный диапазон значений.
- Принадлежность к множеству. Проверяется, принадлежит ли результат вычисления выражения к заданному множеству значений.
- Соответствие шаблону. Проверяется, отвечает ли некоторое строковое значение заданному шаблону.
- Значение NULL. Проверяется, содержит ли данный столбец определитель NULL (неизвестное значение).

Рассмотрим примеры использования всех указанных типов условий поиска.

ПРИМЕР

Сравнение условий поиска. Перечислите весь персонал с размером заработной платы больше определенного значения.

```
SELECT firstname, lastname, salary
FROM staff
WHERE salary > 1000;
```

В языке SQL можно использовать следующие операторы сравнения:

- = равенство;
- < меньше;
- > больше;
- <= меньше или равно;
- >= больше или равно;
- <> не равно (стандарт ISO);
- != не равно (используется в некоторых диалектах).

Более сложные предикаты могут быть построены с помощью логических операторов AND, OR или NOT, а также с помощью скобок,

используемых для определения порядка вычисления выражения (если это необходимо или желательно).

Вычисление выражений в условиях выполняется по следующим правилам:

- Выражение вычисляется слева направо.
- Первыми вычисляются подвыражения в скобках.
- Операторы NOT выполняются до выполнения операторов AND и OR.
- Операторы AND выполняются до выполнения операторов OR.

Для устранения любой возможной неоднозначности рекомендуется использовать скобки.

Для указания диапазонов используется ключевое слово BETWEEN/NOT BETWEEN (аналогичная операция может быть реализована двумя операциями сравнения):

```
SELECT firstname, lastname, salary
FROM staff
WHERE salary BETWEEN 1000 AND 3000;[11]
```

Наличие ключевого слова BETWEEN требует задания границ диапазона значений. Имеется и негативная версия проверки диапазона значений (NOT BETWEEN). Используется в случае, когда проверяемое значение должно лежать вне границ заданного диапазона.

Условия поиска с проверкой вхождения во множество (IN/NOT IN):

```
SELECT firstname, lastname
FROM staff
WHERE lastname IN ('Ivanov', 'Petrov');
```

Существует и отрицательная версия этой проверки - NOT IN.

[11] `WHERE salary > = 1000 AND salary < 3000`

BETWEEN и IN в основном используются для повышения выразительности записи (позволяет сократить запись команды).

Условия поиска по шаблоны определяется через слово - LIKE/NOT LIKE.

В языке SQL существует два специальных символа шаблона, используемых при проверке символьных значений:

- Символ процента (%) представляет любую последовательность из нуля или более символов.
- Символ подчеркивания (_) представляет любой одиночный символ.

Все остальные символы в шаблоне представляют сами себя.

- Address LIKE ‘H%’ - этот шаблон означает, что первый символ значения обязательно должен быть символом H, а все остальные символы не представляют интереса и не проверяются;
- Address LIKE ’H____’ - этот шаблон означает, что значение должно иметь длину равную строго четырем символам (после H, 4-е символа подчеркивания), причем первым символом обязательно должен быть символ ‘H’;
- Address LIKE ‘%e’ – этот шаблон определяет любую последовательность символов длинной не менее одного символа, причем последним символом обязательно должен быть символ ‘e’;
- Address LIKE ’%Moscow%’ - этот шаблон означает, что нас интересует любая последовательность символов, включающая подстроку Moscow;
- Address NOT LIKE ‘H%’ - этот шаблон указывает, что требуются любые строки, которые не начинаются с символа H.

Пример запроса:

```
SELECT firstname, lastname, address, salary
FROM staff
```

```
WHERE address LIKE '% Moscow%';
```

В запросах можно использовать проверку на NULL (неизвестное, незаданное значение) - IS NULL. Отрицательная версия данного условия поиска (IS NOT NULL) может использоваться для проверки присутствия значения в столбце.

В общем случае, строки в результирующей таблице SQL-запроса не упорядочены каким-либо определенным образом.

Их можно отсортировать с помощью ORDER BY. Фраза ORDER BY включает список разделенных запятыми идентификаторов столбцов, по которым требуется упорядочить результирующую таблицу запроса. Идентификатор столбца представляет собой его имя (номера столбцов могут использоваться в тех случаях, когда столбцы, по которым следует упорядочить результат, являются вычисляемыми, а фраза AS с указанием имени вычисляемого столбца в операторе SELECT отсутствует).

Фраза ORDER BY позволяет упорядочить выбранные записи в порядке возрастания (ASC) или убывания (DESK) значений любого столбца или комбинации столбцов, независимо от того, присутствуют эти столбцы в таблице результатов или нет. Однако в некоторых диалектах требуется, чтобы фраза ORDER BY обязательно присутствовала в списке выборки оператора SELECT. В любом случае фраза ORDER BY всегда должна быть последним элементом в операторе SELECT.

Пример сортировки по фамилии:

```
SELECT firstname, lastname, salary
FROM staff
ORDER BY lastname DESC;
```

В качестве имени столбца можно указывать его порядковый номер.

Во фразе ORDER BY может быть указано и больше одного элемента. Если во фразе ORDER BY присутствуют второй и последую-

щие элементы, то такие элементы называют младшими ключами сортировки.

## 6. Задания к лабораторной работе

6.1 В ходе работы должно быть создано 2 web-страниц: с данными 2-х разных баз данных (можно использовать результаты из лабораторной работы №1). Данные должны упорядочиваться по убыванию дат и возрастанию ФИО (при наличии соответствующих полей).

6.2. Выборка данных из таблиц осуществляется через SQL запрос «SELECT». Правильность работы SELECT запросов можно проверить через phpmyadmin (выбрать базу данных, закладка SQL).

6.3. Дополнительно должны быть созданы web-страницы:

- удаления выбранной записи таблицы;
- добавления записи в таблицу.

## 7. Отчетность по лабораторной работе

7.1. В качестве приложения к отчету представить полный листинг php файлов, в которых реализуется взаимодействие с базой данных.

7.2. В качестве приложения к отчету представить список SQL запросов.

# ЛАБОРАТОРНАЯ РАБОТА №3. ВЗАИМОДЕЙСТВИЕ С БАЗОЙ ДАННЫХ (ФРЕЙМВОРКИ)

## 1. Цель работы

1.1. Изучение процесса взаимодействия с базой данных, выполнения запросов (использование фреймворков[12]).

Работа реализуется на языке программирования php, взаимодействие осуществляется с СУБД MySQL, фреймворк - CodeIgniter[13].

## 2. Содержание работы

2.1. Установить и настроить LAMP или WAMP сервер (могли быть установлены в ходе выполнения лабораторной работы №1).

2.2. Используя фреймворк, реализовать интерфейс для получения (добавления) данных из базы данных, включая перечисленные функции:

1) подключение к базе данных;
2) выбрать значения из всех таблиц базы данных (данные должны быть представлены в виде нескольких таблиц на одной web-странице);
3) вывести значения таблицы, удовлетворяющие определенному условию (таблицу и условие выбрать произвольно);
4) добавление данных в таблицы при открытии определенной web-страницы (таблицу и данные выбрать произвольно);
5) вывод диагностических сообщений об ошибках (как минимум: сообщение об отсутствии подключения к БД, об ошибке при выполнении запроса);
6) вывод информации о базе данных (количество таблиц, строк в таблицах, используемой СУБД, версии);

[12] Фре́ймворк, иногда фреймво́рк (англицизм, неологизм от framework «остов, каркас, структура») — заготовки, шаблоны для программной платформы, определяющие структуру программной системы; программное обеспечение, облегчающее разработку и объединение разных модулей программного проекта (Википедия).

[13] https://codeigniter.com/

7) при отсутствии данных в таблице, выводить сообщение «В таблице … данные отсутствуют».

## 3. Концепция MVC

MVC - это паттерн проектирования веб-приложений, который включает в себя несколько более мелких шаблонов.

При использовании MVC модель данных приложения, пользовательский интерфейс и взаимодействие с пользователем разделены на три отдельных компонента, благодаря чему модификация одного из них оказывает минимальное воздействие на остальные или не оказывает его вовсе.

Основная цель применения MVC состоит в разделении данных и бизнес-логики от визуализации (внешнего вида).

Концепция MVC разделяет данные, представление и обработку действий пользователя на **компоненты**:

- *Модель (Model)* - предоставляет собой объектную модель некой предметной области, включает в себя данные и методы работы с этими данными, реагирует на запросы из контроллера, возвращая данные и/или изменяя своё состояние, при этом модель не содержит в себе информации, как данные можно визуализировать, а также не «общается» с пользователем напрямую.
- *Представление (View)* - отвечает за отображение информации (визуализацию), одни и те же данные могут представляться различными способами, например, коллекцию объектов при помощи разных «вьюх» можно представить как в табличном виде, так и списком.
- *Контроллер (Controller)* - обеспечивает связь между пользователем и системой, использует модель и представление для реализации необходимой реакции на действия пользователя, как правило, на уровне контроллера осуществляется фильтрация полученных данных и авторизация (проверяются права пользователя на выполнение действий или получение информации).

## 4. Общие сведения по работе с фреймворком CodeIgniter[14]

CodeIgniter содержит конфигурационный файл, который хранит настройки подключения к базе данных (имя пользователя, пароль, имя бд и т.д). Файл config находится в application/config/database.php.

Также можно установить параметры подключения к базе данных для конкретных режимов разработки поместив database.php в соответствующей среде папку файл config.

Параметры хранятся в многомерном массиве:

```
$db['default'] = array(
    'dsn'   => '',
    'hostname' => 'localhost',
    'username' => 'root',
    'password' => '',
    'database' => 'database_name',
    'dbdriver' => 'mysqli',
    'dbprefix' => '',
    'pconnect' => TRUE,
    'db_debug' => TRUE,
    'cache_on' => FALSE,
    'cachedir' => '',
    'char_set' => 'utf8',
    'dbcollat' => 'utf8_general_ci',
    'swap_pre' => '',
    'encrypt' => FALSE,
    'compress' => FALSE,
    'stricton' => FALSE,
    'failover' => array()
);
```

Существует два способа подключения к базе данных:

1. Автоматическое подключение - свойство “auto connect”. Фрейворк будет загружать и инициализировать класс для работы с базой данных при каждой загрузке страницы. Для активации “auto connecting”, добавьте слово database в массив библиотеки в файле: application/config/autoload.php

[14] Пояснения приведены для версии 3

2. Подключение «вручную». Подключение к базе данных только на нужных страницах. Необходимо добавить следующую строку кода в любой функции, где это необходимо или в классе конструктора, чтобы сделать базу данных доступной во всем классе:

```
$this->load->database();
```

При отсутствии параметров в конструкторе будет осуществляться подключение к базе данных конфигурационного файла.

Закрытие подключения:

```
$this->db->close();
```

### 5. Запросы в CodeIgniter

Используя фреймворк можно работать с запросами «двух видов»: в виде запросов SQL и используя функции фреймоворка.

Функция query() возвращает объект результата запроса к базе данных.

Когда тип запросов “чтение” (select) – возвращается результат.

Когда тип запросов “запись” (insert, update) - возвращает TRUE или FALSE в зависимости от успеха или неудачи. При получении данных обычно записывают запрос в отдельную переменную:

```
$query = $this->db->query('ЗАПРОС');
```

Стандартный запрос с несколькими результатами (результат - объект):

```
$query = $this->db->query('SELECT name, title, email FROM my_table');

foreach ($query->result() as $row)
{
     echo $row->title;
     echo $row->name;
     echo $row->email;
```

```
}

echo 'Общие Результаты: ' . $query->num_rows();
```

Стандартный запрос с несколькими результатами (результат - массив):

```
$query = $this->db->query('SELECT name, title, email FROM my_table');

foreach ($query->result_array() as $row)
{
     echo $row['title'];
     echo $row['name'];
     echo $row['email'];
}
```

Стандартный запрос с одним результатом (результат - объект):

```
$query = $this->db->query('SELECT name FROM my_table LIMIT 1');
$row = $query->row();
echo $row->name;
```

Стандартный запрос с одним результатом (результат - массив):

```
$query = $this->db->query('SELECT name FROM my_table LIMIT 1');
$row = $query->row_array();
echo $row['name'];
```

Запрос через функции фреймоврка (построитель запросов):

```
$query = $this->db->get('table_name');
foreach ($query->result() as $row)
{
     echo $row->title;
}
```

Добавление данных:

```
$data = array(
        'title' => $title,
        'name' => $name,
        'date' => $date
);

$this->db->insert('mytable', $data);
// аналогично: INSERT INTO mytable (title, name, date)
VALUES ('{$title}', '{$name}', '{$date}')
```

Метод error() возвращает массив, содержащий её код и сообщение.

```
if (! $this->db->simple_query('SELECT `example_field` FROM
`example_table`'))
{
        $error = $this->db->error(); // Имеет ключи 'code' и
'message'
}
```

## 6. Контроллеры

Контроллер – это файл с описанием класса, методы которого доступны через URI.

Очень часто в MVC, шаблон URL имеет вид:

```
http://example.com/[controller-class]/[controller-
            method]/[arguments]
```

Пример: Создадим контроллер – индексная страница, возвращающая текст «Hello World!»

Файл с именем Blog.php в каталоге application/controllers/:

```
<?php
class Blog extends CI_Controller {

    public function index()
    {
    echo 'Hello World!';
    }
```

```
}
```

Имя файла ‘Blog.php’, должно начинаться с заглавной ‘B’. Имя класса должно начинаться с заглавной буквы.

Пример: Контроллер БД оплаты товаров, содержащий два метода: чтения данных из БД и записи в БД.

```
<?php
defined('BASEPATH') OR exit('No direct script access allowed');

class Db_controller extends CI_Controller {

     function db()
     {
          $this->load->library('table');
          $tmpl = array ( 'table_open' => '<table border="1" cellpadding="2" cellspacing="1" class="mytable">' );
          $this->table->set_template($tmpl);
          $this->load->model('db_model');

          $data['client'] = $this->db_model->get_client();
          $data['oplata'] = $this->db_model->get_oplata();
          $data['sotrudnik'] = $this->db_model->get_sotrudnik();
          $data['tovar'] = $this->db_model->get_tovar();
          $data['where'] = $this->db_model->where();

          $this->load->view('db_view',$data);
     }

     function add_sotrudnik()
     {
          $data['fio'] = "Панайотова Ольга Алексеевна";
          $data['dolzhnost'] = "Консультант";
          $data['homeadress '] = "Владимир, ул.Нижняя Дуброва, 45, 14";
          $this->load->model('db_model');
          $this->db_model->add_sotrudnik($data);
     }
}
```

## 7. Модели

Модели - это PHP классы, которые предназначены для манипулирования с данными. Например, это может быть класс модели, который содержит функции для вставки, обновления и извлечения данных из базы данных.

Пример того, как может выглядеть подобная модель класса (блога):

```
class Blog_model extends CI_Model {

     public $title;
     public $content;
     public $date;

     public function __construct()
     {
          // Call the CI_Model constructor
          parent::__construct();
     }

     public function get_last_ten_entries()
     {
          $query = $this->db->get('entries', 10);
          return $query->result();
     }

     public function insert_entry()
     {
          $this->title    = $_POST['title'];
          $this->content  = $_POST['content'];
          $this->date     = time();
          $this->db->insert('entries', $this);
     }

     public function update_entry()
     {
          $this->title    = $_POST['title'];
          $this->content  = $_POST['content'];
          $this->date     = time();
```

```
            $this->db->update('entries', $this, array('id'
=> $_POST['id']));
        }

    }
```

Классы моделей хранятся в application/models/. Они могут быть вложенными в подкаталоги.

Прототип для модели класса:

```
class Model_name extends CI_Model {
    public function __construct()
    {
    parent::__construct();
    }

}
```

В данном случае Model_name - имя класса. Первая буква имени класса должна быть заглавное, а остальные в нижнем регистре. Создаваемый класс должен расширять основной класс модели.

Имя файла должно совпадать с именем класса.

Например, если класс:

```
class User_model extends CI_Model {

    public function __construct()
    {
      parent::__construct();
    }

}
```

Ваш файл должен будеть таким: application/models/User_model.php

Модели, как правило, загружаются (вызываются) из контроллера. Чтобы загрузить модель, используется следующий метод:

```
$this->load->model('model_name');
```

Пример описания модели для контроллера БД оплаты товаров.

```
<?php
defined('BASEPATH') OR exit('No direct script access al-
lowed');

class Db_model extends CI_Model {

     function get_client()
     {
          $query = $this->db->get('client');
          if($query !== FALSE && $query->num_rows() > 0){
               return $query->result_array();
          }
          else {echo "ОШИБКА: Данные в таблице 'КЛИЕНТ'
отсутствуют!";}
     }

   function get_oplata()
     {
          $query = $this->db->get('oplata');
          if($query !== FALSE && $query->num_rows() > 0){
               return $query->result_array();
          }
          else {echo "ОШИБКА: Данные в таблице 'ОПЛАТА'
отсутствуют!";}
     }

   function get_sotrudnik()
     {
          $query = $this->db->get('sotrudnik');
          if($query !== FALSE && $query->num_rows() > 0){
               return $query->result_array();
          }
          else {echo "ОШИБКА: Данные в таблице 'СОТРУДНИК'
отсутствуют!";}
     }

     function get_tovar()
     {
```

```
            $query = $this->db->get('tovar');
            if($query !== FALSE && $query->num_rows() > 0){
                return $query->result_array();
            }
            else {echo "ОШИБКА: Данные в таблице 'ТОВАР' от-
сутствуют!";}
        }

        function where()
        {
            $this->db->where('summa', '5000');
            $query = $this->db->get('oplata');
            return $query->result_array();
        }

        function add_sotrudnik($data)
        {
            $this->db->insert('sotrudnik', $data);
            echo "Добавлена новая запись в таблицу 'СОТРУД-
НИК'!";
        }
    }
```

### 8. Представления (виды)

Представление (view) - это веб-страница, или фрагмент страницы. Представления могут встраиваться в другие представления.

Представления никогда не вызываются напрямую - они должны быть загружены контроллером. Загрузка представления осуществляется следующим методом (name - имя файла вида):

```
$this->load->view('name');
```

Представления хранятся в каталоге application/views/. Могут использоваться вложенные каталоги (в данном случае имя файла представления необходимо указывать относительно каталога views: $this->load->view('directory_name/file_name')).

Пример использования представления:

*Контроллер:*

```
<?php
class Blog extends CI_Controller {

     public function index()
     {
          $this->load->view('blogview');
     }
}
```

*Представление*:

```
<html>
<head>
     <title>My Blog</title>
</head>
<body>
     <h1>Welcome to my Blog!</h1>
</body>
</html>
```

В представление можно передать данные из контроллера через второй параметр метода view(). Данные передаются из контроллера в вид в виде массива или объекта при его загрузке.

```
$data = array(
     'title' => 'My Title',
     'heading' => 'My Heading',
     'message' => 'My Message'
);

$this->load->view('blogview', $data);
```

Переданные данные доступны в представлении как переменные php:

```
<title><?php echo $title;?></title>
```

Технически можно внедрять код php в представления для дополнительных манипуляций с данными. Таким образом можно создавать адаптируемые под данные шаблоны.

Третьим параметром метода view() является флаг возвращения представления как данных.

По умолчанию равен False, и результат загрузки представления сразу отправляется браузеру для отображения.

При установке значения в True представление будет возвращено как строка текста (и контроллер может ее передать в другое представление для отображения). Целесообразно использовать в ситуациях, когда представление является блоком более общего представления (например – меню).

## 9. Рекомендованная последовательность действий по установке и настройке

- Скачайте и установите фреймворк.
- Разархивируйте файлы, добавив в предварительно созданную папку на веб-сервере.
- Изучите папки controllers, models и views в папке applications.
- Внесите изменения в файл  изменить файл маршрутов «routes.php»[15].

Пример routes.php для БД оплаты товаров:

```
<?php
defined('BASEPATH') OR exit('No direct script access allowed');

$route['default_controller'] = 'db_controller/db';
$route['404_override'] = '';
$route['translate_uri_dashes'] = FALSE;
```

[15] Правила маршрутизации определяются в файле *application/config/routes.php*. Массив `$route` позволяет задать критерии маршрутизации. Маршруты могут быть заданы подстановкой или регулярным выражением.

Здесь «db_controller/db» соответствует конструкции «контроллер, к которому необходимо обратиться/функция».

## 10. Пример реализации новостной ленты

### 10.1. Схема базы данных

```
--
-- Структура таблицы `news`
--

CREATE TABLE IF NOT EXISTS `news` (
  `id` int(11) NOT NULL COMMENT 'первичный ключ таблицы',
  `caption` varchar(255) NOT NULL COMMENT 'заголовок публикуемой новости',
  `created` date NOT NULL COMMENT 'дата публикации новости',
  `expired` date NOT NULL COMMENT 'дата снятия новости с публикации',
  `preview_text` varchar(255) NOT NULL COMMENT 'краткий текст новости (анонс)',
  `preview_image` varchar(255) NOT NULL COMMENT 'изображение для анонса новости (ссылка на файл)',
  `text` text NOT NULL COMMENT 'полный текст новости',
  `author` varchar(255) NOT NULL COMMENT 'автор новости (сотрудник или внешняя организация)',
  `category` varchar(255) NOT NULL COMMENT 'перечень категорий новости (тегов), через запятую',
  `pushpin` int(11) NOT NULL COMMENT 'новость закреплена (подбираются в начало списка)',
  `file` varchar(255) NOT NULL COMMENT 'ссылка на файл - приложение',
  `visible` int(11) NOT NULL COMMENT 'режим публикации (черновик/опубликована)'
) ENGINE=InnoDB AUTO_INCREMENT=5 DEFAULT CHARSET=utf8mb4;
```

### 10.2. Фрагмент файла маршрутов - routes.php

```
// Новости
$route['news/latest/(:num)'] = 'news/latest/$1';
```

```
    $route['news/archive/(:num)/(:num)/(:num)'] =
'news/archive/$1/$2/$3';
    $route['news/view/(:num)'] = 'news/view/$1';
```

## 10.3. Контроллер - News.php

```
    <?php
    defined('BASEPATH') OR exit('No direct script access al-
lowed');

    class News extends CI_Controller {
        /*
            Контроллер работы с новостями
        */

        public function __construct() {
            parent::__construct();
            $this->load->model('news_model');

        }

        /*
            Вывод архива новостей с паджинатором на базе AJAX
        */

         public function index()  {
            $news_limit = 10;
            $data['news_total'] = $this->db-
>count_all('news');;
            $data['pagination'] = "<a href=\"#\" class=\"btn
btn-primary btn-sm btn-block\" role=\"button\"><span
class=\"glyphicon glyphicon-list\"></span> Перейти в архив
новостей</a>";

             $data['title'] = "Новости";
             $data['news'] = $this->news_model-
>getNews($news_limit, 0);
            $this->load->view('news/latest_view', $data);
         }

         /*
             Вывод определенной новости
```

```
            Параметры:
                $news_id - id новости
        */
        public function view($news_id = 0) {
            if ($news_id > 0) {
               $this->load->helper('url');

            $data['title'] = "Новость";
               $data['pagination'] = "<a
href=\"".base_url()."news/archive/\" class=\"btn btn-primary
btn-sm btn-block\" role=\"button\"><span class=\"glyphicon
glyphicon-list\"></span> Перейти в архив новостей</a>";
            $data['id'] = $news_id;

            $data['news'] = $this->news_model->getNews(0, 0,
$news_id);
               $this->load->view('news/news_view', $data);
            }
        } // public function view(

        /*
            Вывод последних новостей
            Параметры:
                $news_limit - количество выводимых новостей
        */
        public function latest($news_limit = 5) {
           $this->load->helper('url');

           $data['title'] = "Новостная лента";
           $data['news_total'] = $this->db-
>count_all('news');
           $data['news_limit'] = $news_limit;
           $data['pagination'] = "<a
href=\"".base_url()."news/archive/\" class=\"btn btn-primary
btn-sm btn-block\" role=\"button\"><span class=\"glyphicon
glyphicon-list\"></span> Перейти в архив новостей</a>";

            $data['news'] = $this->news_model-
>getNews($news_limit, 0);
           $this->load->view('news/latest_view', $data);
        } // public function latest

       /*
```

```
            Вывод архива новостей с паджинатором на базе AJAX
            Параметры:
                $news_page - номер страницы
                $news_perpage - количество новостей на страни-
це
                $items - выдается вся страница (>0) или только
элементы списка новостей (==0, нужно при запросе через AJAX,
чтобы убрать лишнее форматирование)
         */
          public function archive($news_page = 0, $news_perpage
= 10, $items = 1)   {
            $this->load->helper('url');

            $data['baseurl'] = base_url()."news/archive/";
            $data['news_total'] = $this->db-
>count_all('news');;

             $data['title'] = "Новости";
             $data['items'] = $items;
             $data['news'] = $this->news_model-
>getNews($news_perpage, $news_page * $news_perpage);

             if (($news_page + 1) * $news_perpage < $da-
ta['news_total']) {
                 // есть еще новости для отображения
                $data['pagination'] = "
                    <a id=\"loadmore\" newsba-
seurl=\"".$data['baseurl']."\" newsnextpage=\"".($news_page +
1)."\" newsperpage=\"".$news_perpage."\"
href=\"".$data['baseurl'].($news_page +
1)."/".$news_perpage."\" class=\"btn btn-primary btn-sm btn-
block\" role=\"button\">
                        <span
id=\"ajaxerror_msg\"></span><span class=\"glyphicon glyphicon-
list\"></span> Загрузить еще
                    </a>";
             } else {
                $data['pagination'] = "<a nohref=\"#\"
class=\"btn btn-primary btn-sm btn-block\"
role=\"button\"><span class=\"glyphicon glyphicon-
list\"></span> Загружены все новости</a>";
             }
```

```
            $this->load->view('news/archive_view', $data);
        } // public function archive
    }
```

## 10.4. Модель - News_model.php

```
    <?php

    class News_model extends CI_Model {

        public function __construct() {
            $this->load->database();
        }

        public function getNews($rowperpage = 15, $offset = 0,
$id = FALSE) {
            if ($id === FALSE) {
                // Идентификатор не задан, выводим список все
документов
                $this->db->order_by('id', 'DESC');
                $this->db->limit($rowperpage, $offset);
                $query = $this->db->get('news');
                return $query->result_array();
            }

            // Идентификатор задан, устанавливаем фильтр
            $query = $this->db->get_where('news', array('id'
=> $id));
            return $query->row_array();
        }
    }
```

## 10.5. Представление (приведено только одно из представлений) – news\archive_view.php

```
    <?php if ($items): // убираем оформление, если нужно отоб-
ражить только элемент списка новостей ?>

    <script src="http://code.jquery.com/jquery-
1.11.1.min.js"></script>
```

```
    <link
href="http://maxcdn.bootstrapcdn.com/bootstrap/3.4.0/css/bootst
rap.min.css" rel="stylesheet" id="bootstrap-css">
    <script
src="http://maxcdn.bootstrapcdn.com/bootstrap/3.4.0/js/bootstra
p.min.js"></script>

    <!-- Блок новостей: Архив новостей -->
    <style>

        a:hover {text-decoration:none;}
        .btn a {color: #fff;}

        .widget .panel-body { padding:0px; }
        .widget .list-group { margin-bottom: 0; }
        .widget .panel-title { display:inline }
        .widget .label-info { float: right; }
        .widget li.list-group-item {border-radius: 0;border:
0;border-top: 1px solid #ddd;}
        .widget li.list-group-item:hover { background-color:
rgba(86,61,124,.1); }
        .widget .mic-info { color: #666666;font-size: 11px; }
        .widget .action { margin-top:5px; }
        .widget .comment-text { font-size: 12px; }
        .widget .btn-block { border-top-left-
radius:0px;border-top-right-radius:0px; }

    </style>

    <div class="container">
        <div class="row">

            <div class="panel panel-default widget">
                <div class="panel-heading">
                    <span class="glyphicon glyphicon-
th"></span>
                    <h3 class="panel-title"><?php echo $title;
?></h3>
                    <span class="label label-info">Всего:
<?php echo $news_total; ?></span>
                </div>

            <div class="panel-body">
```

```
                <ul class="list-group" id="news_list">

    <?php endif; //if ($items): ?>
                    <?php foreach ($news as $key => $value):
?>
                    <li class="list-group-item" id="news_<?php
echo $value['id']; ?>">
                        <div class="row">
                            <div class="col-xs-2 col-md-1">
                                <?php
                                    if
(strlen($value['preview_image']) > 0) {}
                                    else {
                                        $val-
ue['preview_image'] = "/images/news_defaulticon3.png";
                                    }
                                ?>
                                <img src="<?php echo $val-
ue['preview_image']; ?>" class="img-circle img-responsive"
alt="" />
                            </div>

                            <div class="col-xs-10 col-md-11">
                                <div>
                                    <a href="/<?php echo $val-
ue['id']; ?>"><?php echo $value['caption']; ?></a>
                                    <div class="mic-info">
                                        Источник: <a
href="#/<?php echo $value['author']; ?>"><?php  echo $val-
ue['author']; ?></a>
                                        от <?php echo date("j
F Y", strtotime($value['created'])); ?>
                                    </div>
                                </div>

                                <div class="comment-
text"><?php echo $value['preview_text']?></div>
                                <div class="action">
                                    <button type="button"
class="btn btn-success btn-xs" title="Дополнительная
информация">
                                         <span
class="glyphicon glyphicon-exclamation-sign"></span> 
```

```
                                        </button>

                                        <?php if ($val-
ue['pushpin']): ?>
                                        <button type="button"
class="btn btn-primary btn-xs" title="Новость прилеплена">
                                             <span
class="glyphicon glyphicon-pushpin"></span> 
                                        </button>
                                        <?php endif; ?>

                                        <?php if
(strlen($value['file']) > 0): ?>
                                        <a href="<?php echo $val-
ue['file']; ?>">
                                            <button type="button"
class="btn btn-success btn-xs" title="Вложение (файл)">
                                                 <span
class="glyphicon glyphicon-paperclip" download-
alt></span> 
                                            </button>
                                        </a>
                                        <?php endif; ?>

                                        <?php foreach (ex-
plode(",", $value['category']) as $cat_key => $cat_value):?>
                                        <a href="<?php echo
$cat_value; ?>">
                                            <button type="button"
class="btn btn-danger btn-xs" title="<?php echo $cat_value;
?>">
                                                <span
class="glyphicon glyphicon-tag"></span> <?php echo $cat_value;
?>
                                            </button>
                                        </a>
                                        <?php endforeach;?>
                                    </div>
                                </div>
                            </div>
                        </li>
                        <?php endforeach; ?>
```

```
    <?php if ($items): // убираем оформление, если нужно отоб-
ражить только элемент списка новостей ?>

                    </ul>
                    <div class="text-center"
id="pagination"><?php echo $pagination; ?></div>
                </div>
            </div>
         </div>
    </div>

    <script type='text/javascript'>

        function getNewsByAjax(newshref) {
            // Запрос новостей с сайта через AJAX
            $.ajax({
                url: newshref + "/0",
                async: true,
                timeout: 15000, // 15 секунд
                type: "GET", // запрос GET или POST

                beforeSend: function(jqXHR, settings) {
                    // что-то делаем ... - подготовка перед
отправкой
                },

                success: function(data) {
                    // Запрос обработан, данные получены
                    if (data.length > 300) {
                        // пришло значимое количество данных
                        $('#news_list').append(data);
                    } else {
                        var pag = docu-
ment.getElementById("loadmore");
                        pag.setAttribute('nohref', "#");
                        pag.removeAttribute('newsnextpage');
                        pag.removeAttribute('newsperpage');
                        pag.removeAttribute('newsbaseurl');
                        pag.innerHTML = "Загружены все ново-
сти";
                    }
                },
```

```
                error:  function (jqXHR, exception) {
                    var msg = '';
                    if (jqXHR.status === 0) {
                        msg = 'Нет подключения к интернету.\n
Проверьте параметры сети.';
                    } else if (jqXHR.status == 404) {
                        msg = 'Запрошенная страница не найдена
[404].';
                    } else if (jqXHR.status == 500) {
                        msg = 'Внутренняя ошибка сервера
[500].';
                    } else if (exception === 'parsererror') {
                        msg = 'Запрос JSON не разобран.';
                    } else if (exception === 'timeout') {
                        msg = 'Истекло время ожидания
ответа.';
                    } else if (exception === 'abort') {
                        msg = 'Запрос был отменен.';
                    } else {
                        msg = 'Неизвестная ошибка.\n' +
jqXHR.responseText;
                    }

                    var res = docu-
ment.getElementById("ajaxerror_msg");
                    res.innerHTML = msg + "<br />";
                    //alert(msg);
                }
            }); // $.ajax
        } // function getNewsByAjax

        $('#pagination').on('click','a',function(e){
            // реагируем на нажатие блока pagination
            e.preventDefault();
            var newsnextpage = $(this).attr('newsnextpage');
            var newsperpage  = $(this).attr('newsperpage');
            var newsbaseurl  = $(this).attr('newsbaseurl');
            var newshref = $(this).attr('href');

            if  ((newsnextpage != undefined) && (newsperpage
!= undefined) && (newsbaseurl != undefined)) {
                // переформатируем ссылку на новую страницу
```

```
                var newpaginationhref = String(newsbaseurl +
(parseInt(newsnextpage) + 1) + "/" + newsperpage);
                var pag = document.getElementById("loadmore");
                pag.setAttribute('href', newpaginationhref);
                pag.setAttribute('newsnextpage', par-
seInt(newsnextpage) + 1);

                // запрос новостей
                getNewsByAjax(newshref);
            }

        }); // $('#pagination').on('click

    </script>

    <!-- Блок новостей: конец блока -->
    <?php endif; //if ($items): ?>
```

## 11. Пояснения к выполнению работы

11.1. Все манипуляции с базой данных должны выполняться с использованием команд фреймворка (без использования SQL запросов).

11.2. В ходе работы должно быть создано несколько web-страниц: с данными базы данных (можно использовать результаты работы по лр №1 и 2).

11.3. Руководство пользователя CodeIgniter (перевод) можно прочитать на http://codeigniter3.info/. Несколько базовых видеоуроков можно посмотреть на https://fructcode.com/ru/courses/codeigniter-mvc/[16]
.

11.4. Выполнение задания не предполагает реализацию форм для ввода данных, но и не исключает их.

[16] В открытом доступе 5 коротких фрагментов, которые дают общее представление о фреймворке, показан процесс работы с БД.

## 12. Отчетность по лабораторной работе

13.1. В качестве приложения к отчету представить полный листинг контроллеров, моделей и видов, используемых для реализации функций, перечисленных в работе и содержимое файла config/routes.php.

# ЛАБОРАТОРНАЯ РАБОТА №4. ВЗАИМОДЕЙСТВИЕ С БАЗОЙ ДАННЫХ (ФРЕЙМВОРКИ)

## 1. Цель работы

1.1. Изучение процесса взаимодействия с базой данных, выполнения запросов (использование фреймворков).

Работа реализуется на языке программирования python, взаимодействие осуществляется с СУБД SQLite, фреймворк - Django[17].

## 2. Содержание работы

2.1. Установить и настроить LAMP или WAMP сервер (могли быть установлены в ходе выполнения лабораторной работы №1).

2.2. Используя фреймворк, реализовать интерфейс для получения (добавления) данных из базы данных, включая перечисленные функции:

1) выбрать значения из всех таблиц базы данных (данные должны быть представлены в виде нескольких таблиц на одной web-странице);
2) вывести значения таблицы, удовлетворяющие определенному условию (таблицу и условие выбрать произвольно);
3) добавление данных в таблицы при открытии определенной web-страницы (таблицу и данные выбрать произвольно);
4) при отсутствии данных в таблице, выводить сообщение «В таблице … данные отсутствуют».

## 3. Установка и настройка среды

Далее приведены краткие сведения по установке и настройке среды Django. Сервер работает под управлением ОС Linux (разработка производилась на базе ОС Xubuntu 16.04).

Создание каталога проектов осуществляется командой: *mkdir djangoprojects*.

[17] https://www.djangoproject.com/

Целесообразно изолировать зависимости в рамках проекта. Virtualenv позволяет изолировать зависимости Python/Django для каждого отдельного проекта. Для некоторых дистрибутивов Linux необходимо установить дополнительные пакеты командой: *sudo apt-get install python-virtualenv*. Создание окружение для проектов осуществляется командой: *python -m virtualenv ENV*. В дальнейшем запуск виртуального окружения осуществляется командой: *source ENV/bin/activate.*

Установка django может быть осуществлена через pip: *pip install django*.

На Рисунок 1 и Рисунок 2 представлены результаты выполнения отмеченных ранее команд.

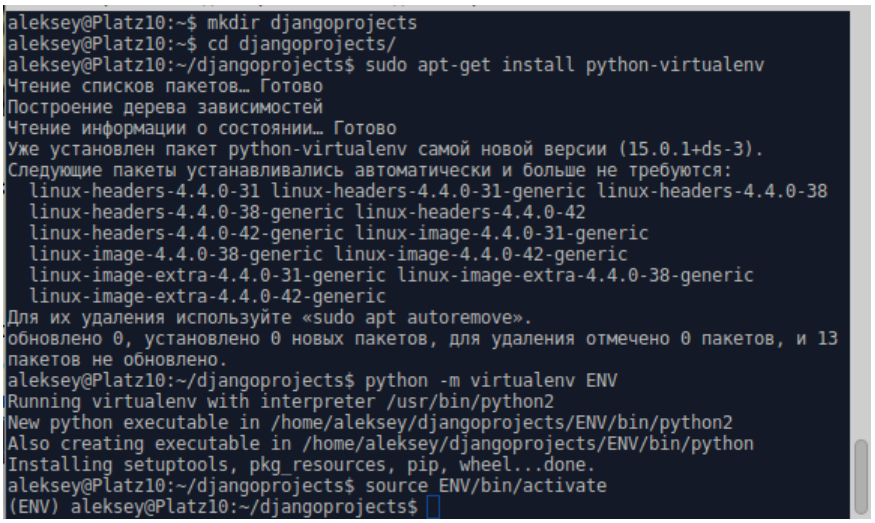



Рисунок 1 – Создание виртуального окружения

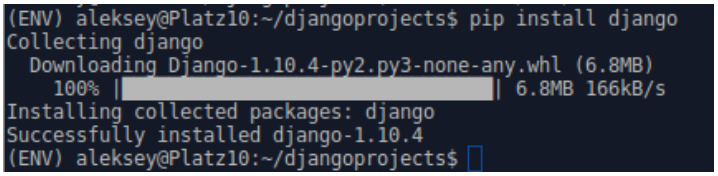


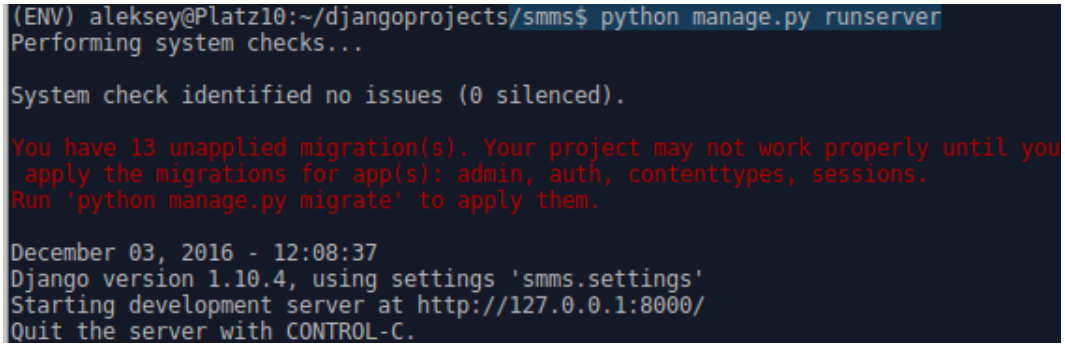

Рисунок 2 – Установка Django

Далее необходимо создать проект командой (проект называется SMMS):

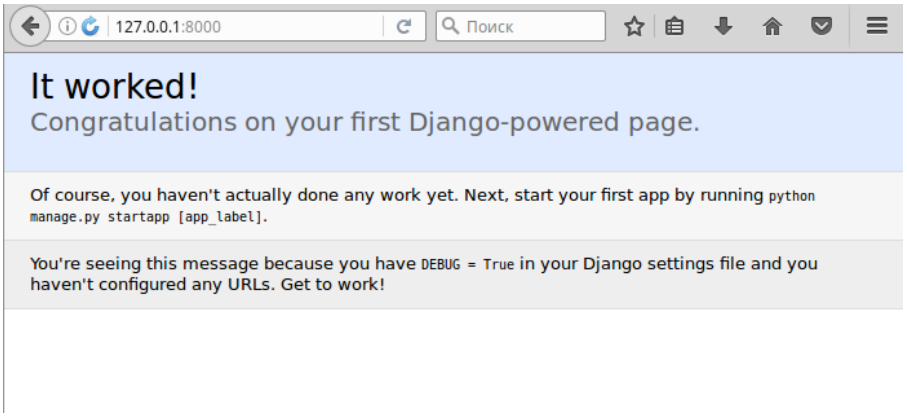
```
django-admin startproject smms
```

Запуск сервера осуществляется одной из следующих команд, в зависимости от требования к доступности сервера (Рисунок 3):

- *python manage.py runserver* – локальный запуск без указания порта (используется порт 8000).
- *python manage.py runserver 8080* - локальный запуск с указанием порта.
- *python manage.py runserver 0.0.0.0:8080* - с указанием доступности для всех IP и порта (сервер становится доступен с других компьютеров в сети).




а




б

Рисунок 3 – Запуск сервера (а) и проверка его работоспособности (б)

Настройка параметров работы сервера (файл settings.py):

- LANGUAGE_CODE = 'ru-ru' – параметр определяет язык интерфейса.

- TIME_ZONE = 'Europe/Moscow' – параметр определяет часовую зону сервера.
- STATIC_URL = '/static/' – параметр определяет расположение статических файлов.
- STATIC_ROOT = os.path.join(BASE_DIR, 'static') - параметр определяет полный путь к статическим файлам.

В качестве основной БД системы в тестовой версии используется sqlite (используется по умолчанию после установки django). Перенос данных на другую СУБД осуществляется изменением настроек. Django может работать с СУБД: PostgreSQL, MySQL, SQLite, Microsoft SQL Server, DB2, Firebird, SQL Anywhere и Oracle.

Впоследствии необходимо изменить параметры ALLOWED_HOST, чтобы разрешить доступ к серверу. Список может содержать полное имя домена (например, 'www.example.com'), тогда значение заголовка Host будет сравниваться с этим значением (проверка без учета регистра и порта). Значение для проверки и поддоменов: '.example.com', удовлетворяет example.com, www.example.com и всем поддоменам example.com. Значение '*' принимает все значения.

Первичное заполнение таблиц БД осуществляется командой (Рисунок 4):

```
python manage.py migrate
```

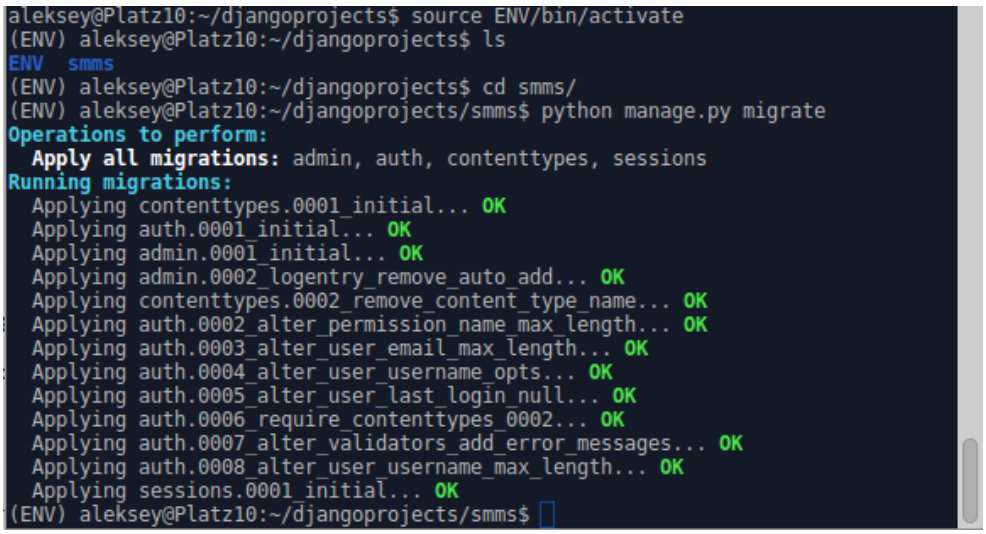
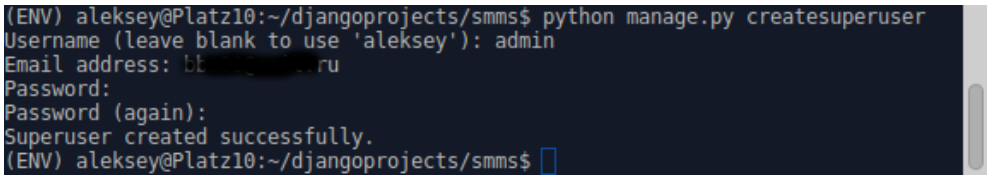


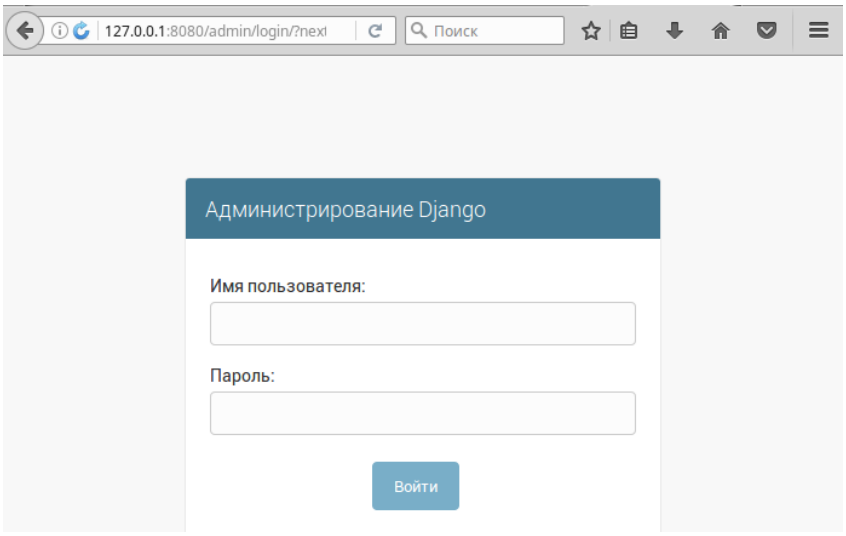

Рисунок 4 – Первичное заполнение таблиц БД

Для доступа к административному интерфейсу необходимо создать суперпользователя - пользователя, который имеет полный доступ к управлению сайтом. Используется команда: *python manage.py createsuperuser*.

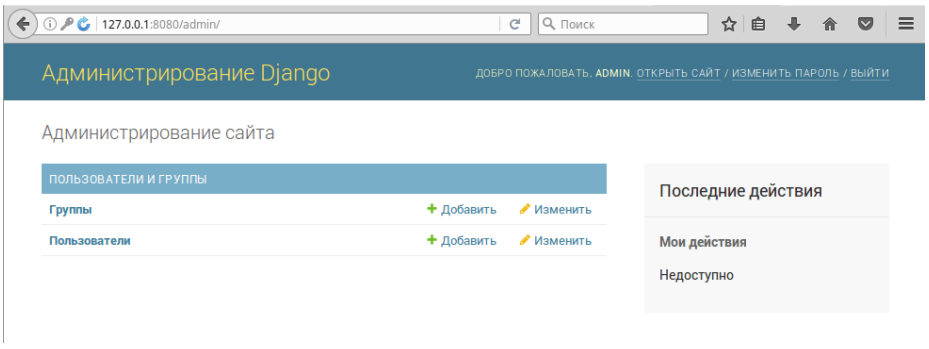
После этого появляется доступ в административный раздел сайта (Рисунок 5).




а




б




в

Рисунок 5 – Создание суперпользователя (а) и административный интерфейс (б, в)

Согласно концепции системы – каждый модуль должен представлять собой приложение проекта, а сам проект – серверную часть. Т.к. управлять доступом к приложениям в рамках проекта можно средствами django, то часть функций модулей управления соединениями и обработки запросов необходимо реализовывать в рамках одного приложения. Таким образом, функционал регистрации и управления шлюзами мобильных платформ (gates) реализован в виде отдельного приложения (используя функции обоих указанных ранее модулей).

Создание приложения gates осуществляется командой: *python manage.py startapp gates.*

Далее необходимо добавить приложение в список приложений проекта: добавляется запись в INSTALLED_APPS в файле smms/settings.py имя приложения 'gates.apps.GatesConfig'. В данном случае параметры настройки и работы приложения будет считываться из файла apps.py приложения (gates/apps.py).

Далее необходимо добавить приложение в шаблоны url проекта (файл smms/urls.py). Таким образом изменения шаблонов ссылок для данного приложения можно будет осуществлять в файле gates/urls.py (независимо от других приложений проекта). Шаблоны ссылок рекомендуется строить на основе регулярных выражений.

Соответствующие настройки представлены на Рисунок 6.


Рисунок 6 – Добавление приложения в проект

Первичное заполнение файла шаблонов url (urls.py):

```
from django.conf.urls import url
from . import views
```

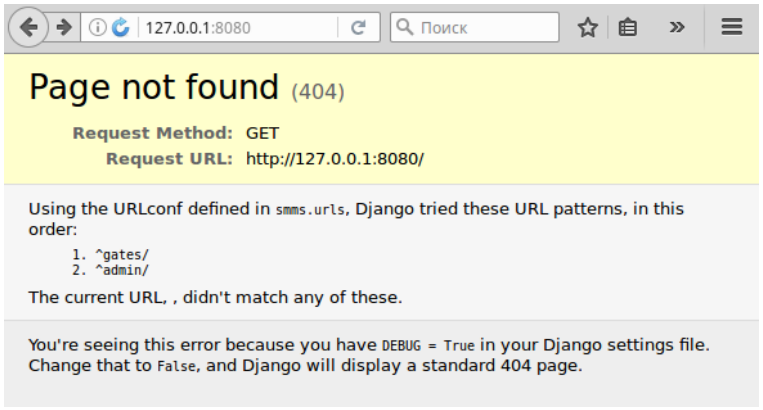
```
urlpatterns = [
    url(r'^$', views.index, name='index'),
]
```

Определение индексного файла (в views.py):

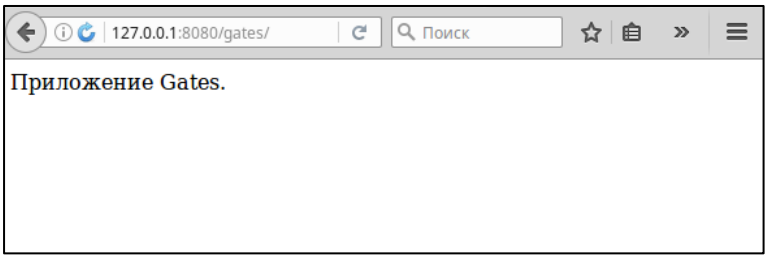
```
# -*- coding: utf-8 -*-
from django.http import HttpResponse
def index(request):
    return HttpResponse("Приложение Gates.")
```

Для корректного отображения букв русского языка необходимо использовать кодировку utf-8 и указывать ее в первой строке файлов .py: # -*- coding: utf-8 -*-

Результаты тестовых запросов к страницам сайта приведены на Рисунок 7.




а




б

Рисунок 7 – Тестовые запросы к страницам сайта (а – страница не найдена, б – индексная страница приложения)

## 4. Краткие сведения по работе с объектами Django[18]

Django использует систему: класс модели представляет таблицу, а экземпляр модели - запись в этой таблице.

Модели отображают информацию о данных, с которыми вы работаете. Они содержат поля и поведение ваших данных. Обычно одна модель представляет одну таблицу в базе данных.

Основы:

- Каждая модель это класс унаследованный от django.db.models.Model.
- Атрибут модели представляет поле в базе данных.
- Django предоставляет автоматически созданное API для доступа к данным.

Примеры созданных моделей:

```
from django.db import models

class Musician(models.Model):
    first_name = models.CharField(max_length=50)
    last_name = models.CharField(max_length=50)
    instrument = models.CharField(max_length=100)

class Album(models.Model):
    artist = models.ForeignKey(Musician,
on_delete=models.CASCADE)
    name = models.CharField(max_length=100)
    release_date = models.DateField()
    num_stars = models.IntegerField()
```

Для определения параметров элементов моделей используются следующие атрибуты (не полный перечень):

- max_digits – определяет максимальное количество цифр;
- decimal_places – определяет количество знаков после запятой;

[18] Фрагменты из официальной документации к программному продукту. Русскую версии документации можно посмотреть на сайте https://djbook.ru/

- max_length – определяет максимальное количество знаков (длина строки);
- blank – определяет, что поле необязательное для заполнения поле (поле может не содержать значение);
- null – определяет, что поле может иметь значение - не установлено (или не устанавливалось);
- verbose_name – определяет метку поля для использования в формах редактирования в административном интерфейсе (при отсутствии указанного атрибута, метки будут автоматически сгенерированы на основании имен полей моделей).

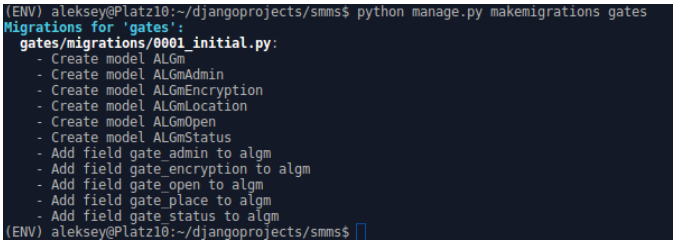
Для организации визуального отображения записей используются строковые представления. Например, для вывода данных поля gate_status[19]:

```
def __unicode__(self):
    return u'%s' % (self.gate_status)
```

После определения моделей (их изменения), их нужно активировать. Активация моделей осуществляется следующими командами (Рисунок 8):

- уведомление об изменение модели: *python manage.py makemigrations gates.*
- проверка системы перед изменениями (при необходимости): *python manage.py check.*
- внесение изменений в БД: *python manage.py migrate.*

При обновлении моделей нужно обязательно выполнять *makemigrations* и *migrate*.




[19] В некоторых версиях используется __str__()

а

```
(ENV) aleksey@Platz10:~/djangoprojects/smms$ python manage.py migrate
Operations to perform:
  Apply all migrations: admin, auth, contenttypes, gates, sessions
Running migrations:
  Applying gates.0001_initial... OK
(ENV) aleksey@Platz10:~/djangoprojects/smms$
```

б

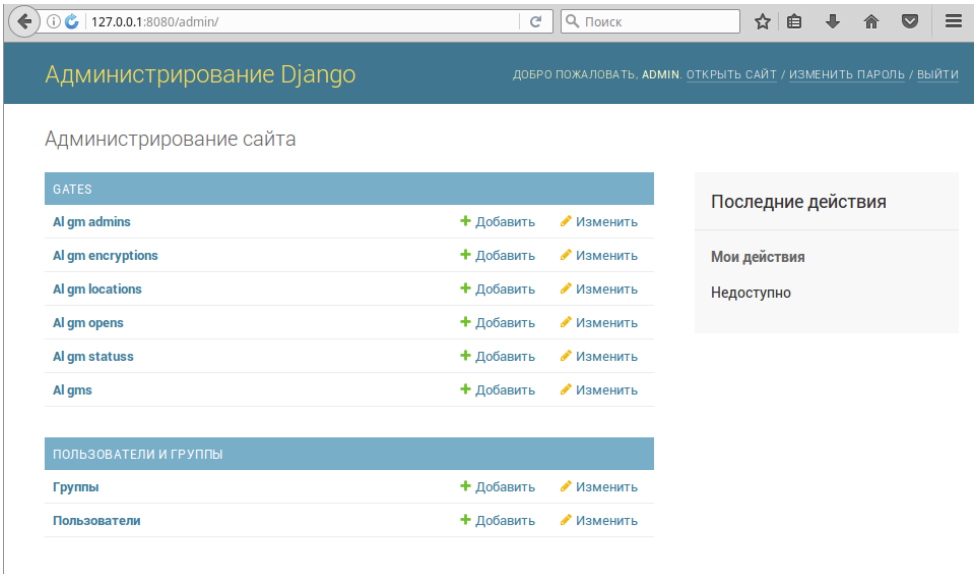
Рисунок 8 – Активация моделей (пример)

Для работы с моделями (и данными) в административном интерфейсе, их нужно зарегистрировать в интерфейсе, добавив записи в файл admin.py приложения (Рисунок 9).

```
/home/aleksey/django~/smms/gates/admin.py   [----] 87 L:[  1+ 4   5/ 12] *(174 / 364b) 00
# -*- coding: utf-8 -*-
from django.contrib import admin

# Register your models here.
from .models import ALGm, ALGmStatus, ALGmOpen, ALGmEncryption, ALGmAdmin, ALGmLocation
admin.site.register(ALGm)
admin.site.register(ALGmStatus)
admin.site.register(ALGmOpen)
admin.site.register(ALGmEncryption)
admin.site.register(ALGmAdmin)
admin.site.register(ALGmLocation)
```

а




б

Рисунок 9 – Регистрация моделей в административном интерфейса (а) и их отображение (б)

Для управления форматом вывода полей (и работы с ними) используются следующие основные атрибуты:

- list_display – параметры отображения и использования элементов интерфейса (поля, отображаемые в списке для изменения и их порядок);
- list_filter – добавление фильтров (поля, по которым возможна установка фильтров);
- search_fields – панель поиска (поля, по которым возможно осуществлять поиск);
- ordering – определение параметров сортировки (поля, по которым возможна сортировка и порядок сортировки);
- fields – определение полей, доступных на форме редактирования;
- raw_id_fields – использование строки ввода (с дополнительной формой поиска при необходимости).

Пример определения формата вывода полей в административном интерфейсе:

```
    class ALGmAdminAdmin(admin.ModelAdmin):
        # параметры отображения и использования элементов ин-
терфейса
        # поля, отображаемые в списке для изменения и их поря-
док
        list_display = ('edit', 'gate_id', 'name', 'phone',
'email', 'set_date', 'user', )
        # добавление фильтров (поля, по которым возможна уста-
новка фильтров)
        list_filter = ('name', 'set_date', )
        # панель поиска (поля, по которым возможно осуществ-
лять поиск)
        search_fields = ('name', 'gate_id', )
        # определение фильтра по дате (можно использовать
только поле даты, в ед.экземпляре)
        # необходима установка pytz
        #date_hierarchy = 'set_date'
        # определение параметров сортировки (поля, по которым
возможна сортировка и порядок сортировки)
        ordering = ('gate_id', 'name', 'user', '-set_date', )
        # определение полей, доступных на форме редактирования
        #fields = ('', '', )
```

```
        # формат представления элементов, типа select - по
умолчанию выбор из списка
        # изменение на использование панелей выбора (указыва-
ются поля, для которых их использовать)
        # только для полей многие-ко-многим
        #filter_horizontal = ('', )
        # использование вместо select строки ввода (с
доп.формой поиска при необходимости)
        raw_id_fields = ('user', 'gate_id')
        # поле используемое как ссылка для редактирования ('',
...) или None
        #list_display_links = None
        # список полей, доступных для редактирования
        #list_editanle = ('', )
        # список полей, доступных только для чтения
        #readonly_fields = ('gate_id', )
        pass
```

В модели также можно добавлять поля, которые генерируются на основе других полей.

Чтобы создать объект, необходимо создать экземпляр класса модели, указав необходимые поля в аргументах и вызвать метод save() для его сохранения в базе данных.

Предположим, что модель находится в mysite/blog/models.py:

```
>>> from blog.models import Blog
>>> b = Blog(name='Beatles Blog', tagline='All the latest
Beatles news.')
>>> b.save()
```

В результате выполнения этого кода будет создан INSERT SQL-запрос. Django не выполняет запросов к базе данных, пока не будет вызван метод save(). Метод save() ничего не возвращает.

Для сохранения изменений в объект, который уже существует в базе данных, используйте save(). В данном примере изменяется название объекта b5 модели Blog и обновляется запись в базе данных:

```
>>> b5.name = 'New name'
>>> b5.save()
```

В результате выполнения этого кода будет создан UPDATE SQL запрос. Django не выполняет каких либо запросов к базе данных, пока не будет вызван метод save().

Обновление ForeignKey работает так же, как и сохранение обычных полей - просто назначьте полю объект необходимого типа и потом вызовите метод save().

Обновление ManyToManyField работает немного по-другому - используйте метод add() поля, чтобы добавить связанный объект.

Django вызовет исключение, если вы попытаетесь добавить объект неверного типа.

Для получения объектов из базы данных, создается QuerySet через Manager модели.

QuerySet представляет выборку объектов из базы данных. Он может не содержать, или содержать один или несколько *фильтров* – критерии для ограничения выборки по определенным параметрам. В терминах SQL, QuerySet - это оператор SELECT, а фильтры - условия такие, как WHERE или LIMIT.

QuerySet, возвращенный Manager, описывает все объекты в таблице базы данных. Обычно вам нужно выбрать только подмножество всех объектов. Для создания такого подмножества, вы можете изменить QuerySet, добавив условия фильтрации. Два самых простых метода изменить QuerySet - это:

- filter() - возвращает новый QuerySet, который содержит объекты удовлетворяющие параметрам фильтрации.
- exclude(**kwargs) - возвращает новый QuerySet содержащий объекты, которые *не* удовлетворяют параметрам фильтрации.

Например, для создания QuerySet чтобы получить записи с 2006, используйте filter() таким образом:

```
Entry.objects.filter(pub_date__year=2006)
```

Это аналогично:

```
Entry.objects.all().filter(pub_date__year=2006)
```

Результат изменения QuerySet - это новый QuerySet и можно использовать цепочки фильтров (добавлять фильтр в результатам обработки фильтров). Например:

```
>>> Entry.objects.filter(
...     headline__startswith='What'
... ).exclude(
...     pub_date__gte=datetime.date.today()
... ).filter(
...     pub_date__gte=datetime(2005, 1, 30)
... )
```

В этом примере к начальному QuerySet, который возвращает все объекты, добавляется фильтр, затем исключающий фильтр, и еще один фильтр. Полученный QuerySet содержит все объекты, у которых заголовок начинается с “What”, и которые были опубликованы между 30-го января 2005 и текущей датой.

filter() всегда возвращает QuerySet, даже если только один объект возвращен запросом - в этом случае, это будет QuerySet содержащий один объект. Если вы знаете, что только один объект возвращается запросом, вы можете использовать метод get() менеджера (Manager), который возвращает непосредственно объект:

```
>>> one_entry = Entry.objects.get(pk=1)
```

Вы можете использовать для get() аргументы, такие же, как и для filter().

Учтите, что есть разница между использованием get() и filter() с [0]. Если результат пустой, get() вызовет исключение DoesNotExist. Также будет исключение если запрос get() вернет не один объект - будет вызвано исключение MultipleObjectsReturned.

Для добавления функционала работы с экземпляром модели(«row-level» functionality), необходимо просто добавить метод в модель. В то время, как методы Manager работают с таблицей, методы модели работают с конкретной записью в таблице. Это хороший под-

ход для хранения бизнес логики работы с данными в одном месте – модели.

Для организации интерфейса пользователя используются представления (views).

Пример простого представления, которое отображает текущую дату и время, как документ HTML:

```
from django.http import HttpResponse
import datetime

def current_datetime(request):
    now = datetime.datetime.now()
    html = "<html><body>It is now %s.</body></html>" % now
    return HttpResponse(html)
```

Давайте проанализируем код строка за строкой:

- Сначала мы импортировали класс HttpResponse из модуля django.http и библиотеку Python datetime.
- Теперь определим функцию current_datetime. Это функция представления. Каждая функция представления принимает объект HttpRequest первым аргументом, который обычно называют request.
- Название функции может быть каким угодно, нет никаких конкретных правил для именования. Мы назвали функцию current_datetime.
- Представление возвращает объект HttpResponse, который содержит сгенерированный ответ. Каждая функция представления должна возвращать объект HttpResponse.

Чтобы привязать это представление к определенному URL-у, необходимо создать *URLconf* (прописать в файле путей (urls) формат и параметры доступа к представлению).

Вот пример простого URLconf:

```
from django.urls import path
from . import views
urlpatterns = [
```

```
        path('articles/2003/', views.special_case_2003),
        path('articles/<int:year>/', views.year_archive),
        path('articles/<int:year>/<int:month>/',
views.month_archive),
        path('articles/<int:year>/<int:month>/<slug:slug>/',
views.article_detail),
    ]
```

### 5. Пояснения к выполнению работы

5.1. Все манипуляции с базой данных должны выполняться с использованием команд фреймворка (без использования SQL запросов).

5.2. В ходе работы должно быть создано несколько web-страниц: с данными базы данных (можно использовать результаты работы по лр №1 и 2).

5.3. Руководство пользователя можно прочитать на https://djbook.ru/.

5.4. Выполнение задания не предполагает реализацию форм для ввода данных, но и не исключает их.

### 6. Отчетность по лабораторной работе

6.1. В качестве приложения к отчету представить полный листинг файлов моделей (models), представлений (views), URLconf (urls) и файл настроек.

# ЛАБОРАТОРНАЯ РАБОТА №5. ОПЕРАЦИИ НАД ОТНОШЕНИЯМИ В РЕЛЯЦИОННЫХ БД

## 1. Цель работы

1.1. Изучение операций построения отношений на основе других отношений, управление выводом результатов.

Работа реализуется в консоли СУБД через запросы к БД.

## 2. Содержание работы

2.1. Восстановить схемы БД по примерам и заполнить данными.

2.2. Воспроизвести все примеры и написать запросы заданий из текста описания лабораторной работы.

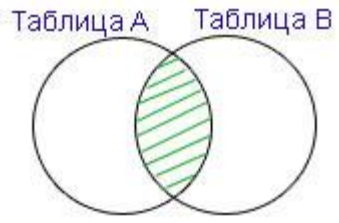
## 3. Запросы объединения (соединения)

Оператор языка SQL JOIN предназначен для соединения двух или более таблиц базы данных по совпадающему условию. Оператор SQL JOIN имеет следующий синтаксис:

```
SELECT имена_столбцов (1..N) FROM имя_таблицы_1 JOIN имя_таблицы_2 ON условие
```

Возможны несколько вариантов соединения: INNER JOIN, LEFT OUTER JOIN, RIGHT OUTER JOIN, FULL OUTER JOIN, CROSS JOIN.

Запрос с оператором INNER JOIN (**внутреннее соединение**) предназначен для соединения таблиц и вывода результирующей таблицы, в которой данные полностью пересекаются по условию, указанному после ON.



То же самое делает и просто JOIN. Таким образом, слово INNER - не обязательное.

**Пример.** Есть база данных портала объявлений. В ней есть таблица Categories (категории объявлений) и Parts (рубрики категориям). Например, части Квартиры, Дачи относятся к категории Недвижимость, а части Автомобили, Мотоциклы - к категории Транспорт.

Таблица Parts:

| Part_ID | Part | Cat |
|---|---|---|
| 1 | Квартиры | 505 |
| 2 | Автомашины | 205 |
| 3 | Доски | 10 |
| 4 | Шкафы | 30 |
| 5 | Книги | 160 |

Таблица Categories:

| Cat_ID | Cat_name | Price |
|---|---|---|
| 10 | Стройматериалы | 105,00 |
| 505 | Недвижимость | 210,00 |
| 205 | Транспорт | 160,00 |
| 30 | Мебель | 77,00 |
| 45 | Техника | 65,00 |

Требуется соединить данные этих двух таблиц так, чтобы в результирующей таблице были поля Part (Часть), Cat (Категория) и Price (Цена подачи объявления) и чтобы данные полностью пересекались по условию. Условие - совпадение идентификатора категории в таблице Categories и ссылки на категорию в таблице Parts.

Запрос:

```
SELECT PARTS.Part, CATEGORIES.Cat_ID AS Cat, CATEGO-
RIES.Price FROM PARTS INNER JOIN CATEGORIES ON PARTS.Cat = CAT-
EGORIES.Cat_ID
```

Результатом выполнения запроса будет следующая таблица:

| Part | Cat | Price |
|---|---|---|
| Квартиры | 505 | 210,00 |
| Автомашины | 205 | 160,00 |
| Доски | 10 | 105,00 |
| Шкафы | 30 | 77,00 |

В результирующей таблице нет Книг, так как эта запись ссылается на категорию, которой нет в таблице Categories, и Техники, так как эта запись имеет внешний ключ в таблице Categories, на который нет ссылки в таблице Parts.

Запрос с оператором LEFT OUTER JOIN **(левое внешнее соединение)** предназначен для соединения таблиц и вывода результирующей таблицы, в которой данные полностью пересекаются по условию, указанному после ON, и дополняются записями из первой по порядку (левой) таблицы, даже если они не соответствуют условию.

У записей левой таблицы, которые не соответствуют условию, значение столбца из правой таблицы будет NULL (неопределённым).


**Пример.** База данных из предыдущего примера. Запрос:

```
SELECT PARTS.Part, CATEGORIES.Cat_ID AS Cat, CATEGO-
RIES.Price FROM PARTS LEFT OUTER JOIN CATEGORIES ON PARTS.Cat =
CATEGORIES.Cat_ID
```

Результатом выполнения запроса будет следующая таблица:

| Part | Cat | Price |
|---|---|---|
| Квартиры | 505 | 210,00 |
| Автомашины | 205 | 160,00 |
| Доски | 10 | 105,00 |
| Шкафы | 30 | 77,00 |
| Книги | 160 | NULL |

В результирующей таблице, в отличие от таблицы из примера 1, есть Книги, но значение столбца Цены (Price) у них - NULL, так как эта запись имеет идентификатор категории, которой нет в таблице Categories.

Запрос с оператором RIGHT OUTER JOIN **(правое внешнее соединение)** предназначен для соединения таблиц и вывода результирующей таблицы, в которой данные полностью пересекаются по условию, указанному после ON, и дополняются записями из второй по порядку (правой) таблицы, даже если они не соответствуют условию.

У записей правой таблицы, которые не соответствуют условию, значение столбца из левой таблицы будет NULL (неопределённым).


**Пример.** База данных из предыдущего примера. Запрос:

```
    SELECT PARTS.Part, CATEGORIES.Cat_ID AS Cat, CATEGO-
RIES.Price FROM PARTS RIGHT OUTER JOIN CATEGORIES ON PARTS.Cat
= CATEGORIES.Cat_ID
```

Результатом выполнения запроса будет следующая таблица:

| Part | Cat | Price |
|---|---|---|

| | | |
|---|---|---|
| Квартиры | 505 | 210,00 |
| Автомашины | 205 | 160,00 |
| Доски | 10 | 105,00 |
| Шкафы | 30 | 77,00 |
| NULL | 45 | 65,00 |

В результирующей таблице, в отличие от таблицы из примера 1, есть запись с категорией 45 и ценой 65,00, но значение столбца Части (Part) у неё - NULL, так как эта запись имеет идентификатор категории, на которую нет ссылок в таблице Parts.

Запрос с оператором FULL OUTER JOIN **(полное внешнее соединение)** предназначен для соединения таблиц и вывода результирующей таблицы, в которой данные полностью пересекаются по условию, указанному после ON, и дополняются записями из первой (левой) и второй (правой) таблиц, даже если они не соответствуют условию.

У записей, которые не соответствуют условию, значение столбцов из другой таблицы будет NULL (неопределённым).


**Пример.** База данных из предыдущего примера. Запрос:

```
SELECT PARTS.Part, CATEGORIES.Cat_ID AS Cat, CATEGO-
RIES.Price FROM PARTS FULL OUTER JOIN CATEGORIES ON PARTS.Cat =
CATEGORIES.Cat_ID
```

Результатом выполнения запроса будет следующая таблица:

| Part | Cat | Price |
|---|---|---|
| Квартиры | 505 | 210,00 |

| | | |
|---|---|---|
| Автомашины | 205 | 160,00 |
| Доски | 10 | 105,00 |
| Шкафы | 30 | 77,00 |
| Книги | 160 | NULL |
| NULL | 45 | 65,00 |

В результирующей таблице есть записи Книги (из левой таблицы) и с категорией 45 (из правой таблицы), причём у первой из них неопределённая цена (столбец из правой таблицы), а у второй - неопределённая часть (столбец из левой таблицы).

Для именования извлекаемых столбцов можно использовать полные наименования таблиц или псевдонимы[20] таблиц (их сокращённые имена).

Общее правило: в секции запроса, определяющей соединение (вокруг слова JOIN) нужно указать полные имена таблиц за которыми следуют псевдонимы таблицы (PARTS P).

**Пример.** Использование псевдонимов соединяемых таблиц. Запрос:

```
SELECT P.Part, C.Cat_ID AS Cat, C.Price FROM PARTS P INNER
JOIN CATEGORIES C ON P.Cat = C.Cat_ID
```

Данные, необходимые в процессе выполнения операции могут содержаться в нескольких таблицах. В этих случаях используются цепочки соединённых таблиц (связь идет по ключам). При помощи оператора SQL JOIN в одном запросе можно соединить все таблицы цепочки.

Синтаксис SQL запроса для соединения более двух таблиц следующий:

```
SELECT имена_столбцов (1..N) FROM имя_таблицы_1 JOIN
имя_таблицы_2 ON условие JOIN имя_таблицы_3 ON условие ... JOIN
имя_таблицы_m ON условие
```

[20] Возможно любое количество букв в псевдониме (в т.ч. и одна буква).

Использование оператора SQL CROSS JOIN **(перекрестное соединение)** реализует операцию **декартова произведения в реляционной алгебре**. Результатом такого соединения будет сцепление каждой строки первой таблицы с каждой строкой второй таблицы.

Таблицы могут быть записаны в запросе либо через оператор CROSS JOIN, либо через запятую между ними.

**Пример.** База данных из предыдущего примера. Запрос:

```
SELECT (*) Categories CROSS JOIN Parts
```

Или без явного указания CROSS JOIN - через запятую:

```
SELECT (*) Categories, Parts
```

Результат - 25 строк, фрагмент которой приведён ниже:

| Cat_ID | Cat_name | Price | Part_ID | Part | Cat |
|---|---|---|---|---|---|
| 10 | Стройматериалы | 105,00 | 1 | Квартиры | 505 |
| 10 | Стройматериалы | 105,00 | 2 | Автомашины | 205 |
| 10 | Стройматериалы | 105,00 | 3 | Доски | 10 |
| 10 | Стройматериалы | 105,00 | 4 | Шкафы | 30 |
| 10 | Стройматериалы | 105,00 | 5 | Книги | 160 |
| ... | ... | ... | ... | ... | ... |
| 45 | Техника | 65,00 | 1 | Квартиры | 505 |
| 45 | Техника | 65,00 | 2 | Автомашины | 205 |
| 45 | Техника | 65,00 | 3 | Доски | 10 |
| 45 | Техника | 65,00 | 4 | Шкафы | 30 |
| 45 | Техника | 65,00 | 5 | Книги | 160 |

Для CROSS JOIN можно задать условие соединения. При использовании оператора "запятая" вместо явного указания CROSS JOIN условие соединения задаётся не словом ON, а словом WHERE. Запрос

с перекрестным соединением по условию соединения полностью аналогичен запросу с внутренним соединением - INNER JOIN.

**Пример.** База данных из предыдущего примера. Используя перекрестное соединение, соединить таблицы так, чтобы данные полностью пересекались по условию. Запрос:

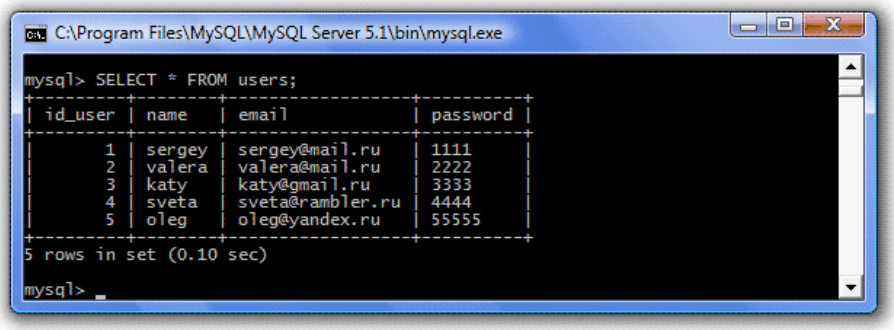
```
SELECT P.Part, C.Cat_ID AS Cat, C.Price FROM PARTS P, CATEGORIES C WHERE P.Cat = C.Cat_ID
```

### 4. Выборка данных

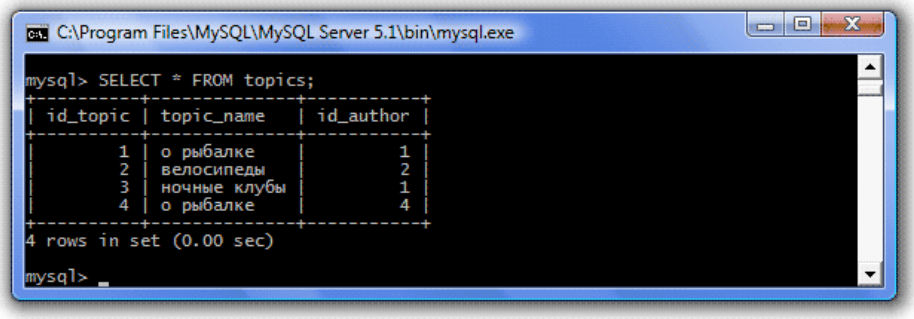
В БД forum есть три таблицы: users (пользователи), topics (темы) и posts (сообщения). И мы хотим посмотреть, какие данные в них содержатся.

Выбрать данные по всем столбцам из таблицы:

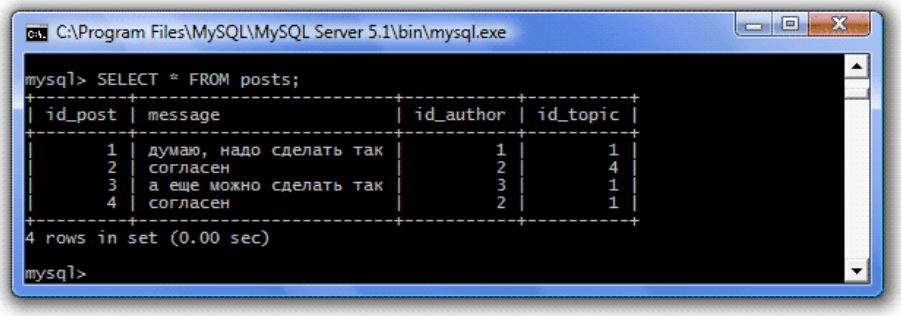
```
SELECT * FROM users;
```




```
SELECT * FROM topics;
```



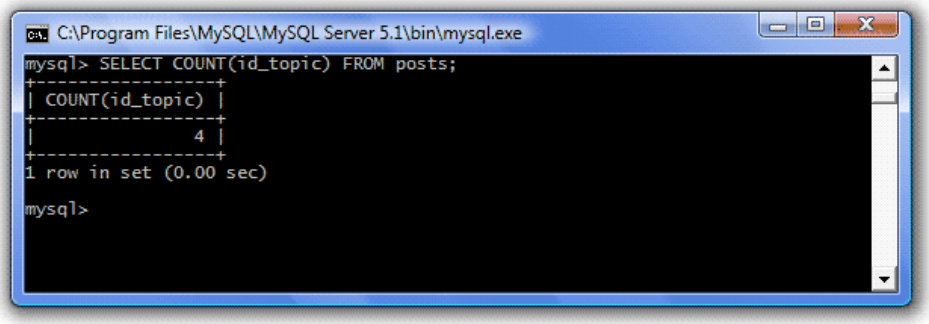

```
SELECT * FROM posts;
```




## 5. Функции

Запрос для подсчета количество строк по столбцу id_topic:

```
SELECT COUNT(id_topic) FROM posts;
```




Итоговые функции еще называют статистическими, агрегатными или суммирующими. Эти функции обрабатывают набор строк для подсчета и возвращения одного значения. Таких функций всего пять:

- AVG() - Функция возвращает среднее значение столбца.
- COUNT() - Функция возвращает число строк в столбце.
- MAX() - Функция возвращает самое большое значение в столбце.
- MIN() - Функция возвращает самое маленькое значение в столбце.
- SUM() - Функция возвращает сумму значений столбца.

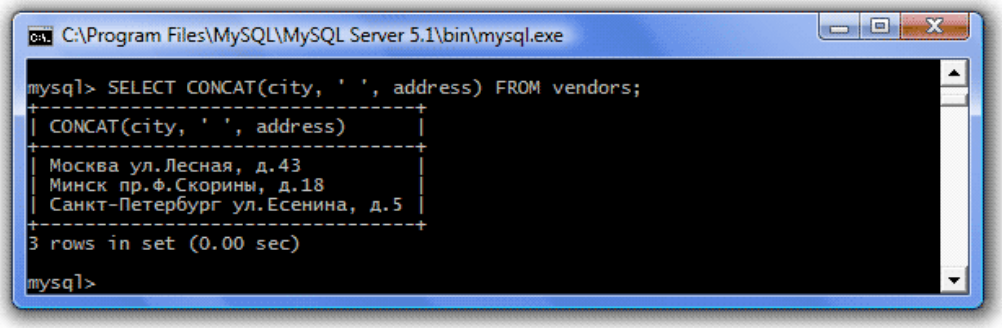
В запросах могут присутствовать вычисляемые поля. Ключевое слово AS может использоваться для задания имени вычисляемого столбца:

```
SELECT имя_столбца_1, имя_столбца_2,
имя_столбца_1*имя_столбца_2 AS имя_вычисляемого_столбца FROM
имя_таблицы;
```

Строковые функции (некоторые из них):

CONCAT(str1,str2...) - Возвращает строку, созданную путем объединения аргументов (аргументы указываются в скобках - str1,str2...). Например, в нашей таблице Поставщики (vendors) есть столбец Город (city) и столбец Адрес (address). Предположим, мы хотим, чтобы в результирующей таблице Адрес и Город указывались в одном столбце:

```
SELECT CONCAT(city, ' ', address) FROM vendors;
```

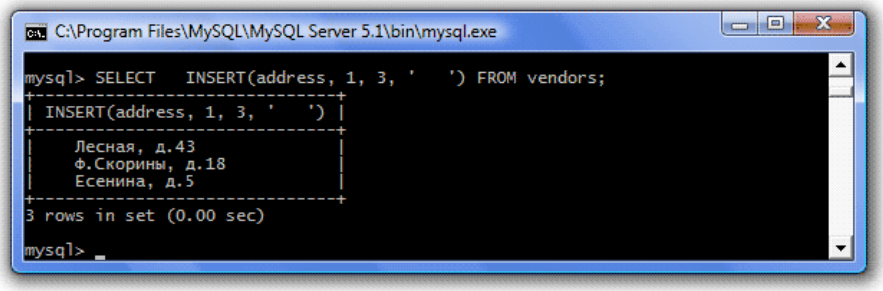



Как видите, пробел считается тоже аргументом и указывается через запятую.

Если объединяемых столбцов было бы больше, то указывать каждый раз пробелы было бы нерационально. В этом случае можно было бы использовать строковую функцию - CONCAT_WS(разделитель, str1,str2...).

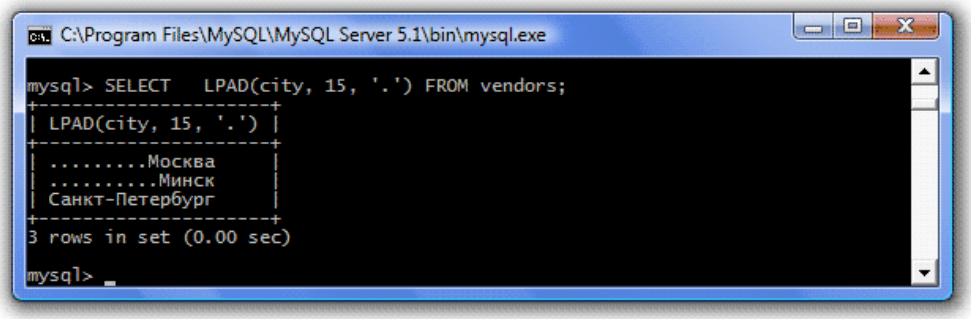
INSERT(str, pos, len, new_str) - Возвращает строку str, в которой подстрока, начинающаяся с позиции pos и имеющая длину len символов, заменена подстрокой new_str. Предположим, мы решили в столбце Адрес (address) не отображать первые 3 символа (сокращения ул., пр., и т.д.), заменив их на пробелы:

```
SELECT INSERT(address, 1, 3, '   ') FROM vendors;
```




LPAD(str, len, dop_str) - Возвращает строку str, дополненную слева строкой dop_str до длины len. Предположим, мы хотим, чтобы при выводе городов поставщиков они располагались бы справа, а пустое пространство заполнялось бы точками:

```
SELECT LPAD(city, 15, '.') FROM vendors;
```



Обратите внимание, значение len ограничивает количество выводимых символов, т.е. если название города будет длиннее 15 символов, то оно будет обрезано.

RPAD(str, len, dop_str) - Возвращает строку str, дополненную справа строкой dop_str до длины len.

LTRIM(str) - Возвращает строку str, в которой удалены все начальные пробелы.

RTRIM(str) - Возвращает строку str, в которой удалены все конечные пробелы.

TRIM(str) -Возвращает строку str, в которой удалены все начальные и конечные пробелы.

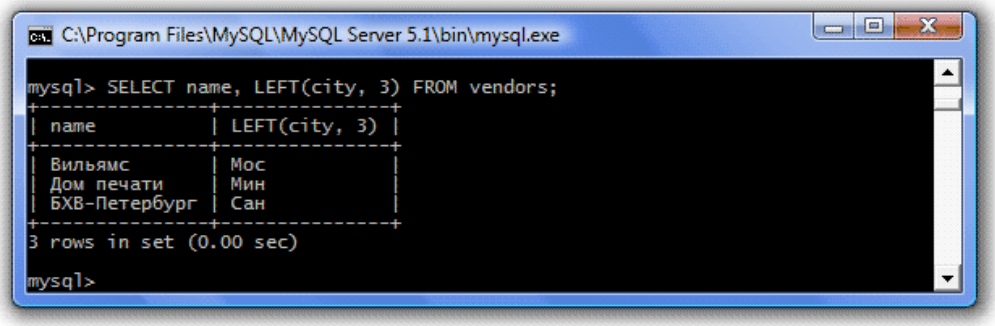
LOWER(str) - Возвращает строку str, в которой все символы переведены в нижний регистр.

UPPER(str) - Возвращает строку str, в которой все символы переведены в верхний регистр.

LENGTH(str) - Возвращает длину строки str.

LEFT(str, len) - Возвращает len левых символов строки str. Например, пусть в городах поставщиков выводится только первые три символа:

```
SELECT name, LEFT(city, 3) FROM vendors;
```



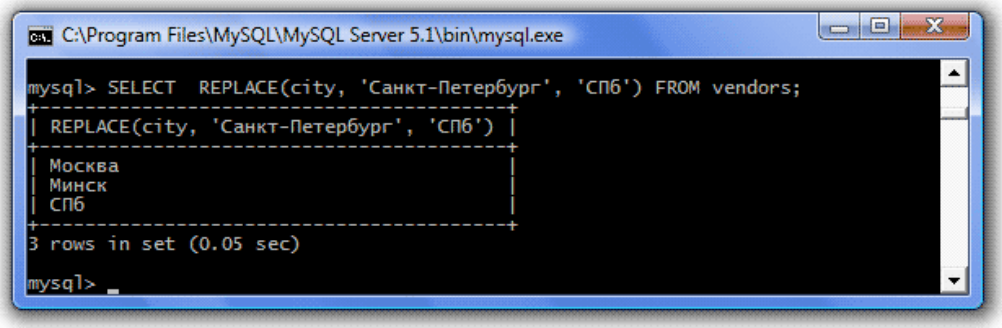

RIGHT(str, len) - Возвращает len правых символов строки str.

REPEAT(str, n) - Возвращает строку str n-количество раз.

REPLACE(str, pod_str1, pod_str2) - Возвращает строку str, в которой все подстроки pod_str1 заменены подстроками pod_str2. Напри-

мер, пусть мы хотим, чтобы в городах поставщиков вместо длинного 'Санкт-Петербург' выводилось короткое 'СПб':

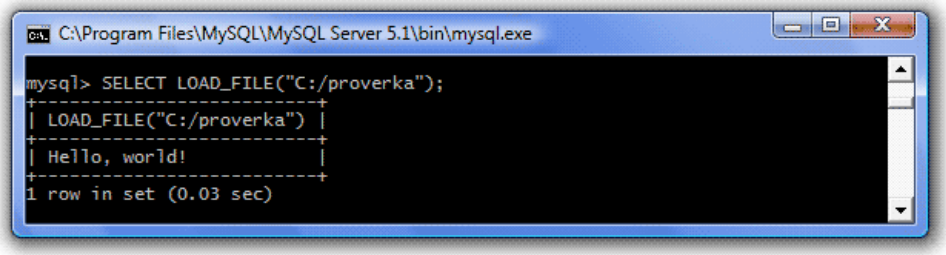
```
SELECT  REPLACE(city, 'Санкт-Петербург', 'СПб') FROM ven-
dors;
```




REVERSE(str) - Возвращает строку str, записанную в обратном порядке.

LOAD_FILE(file_name) - читает файл file_name и возвращает его содержимое в виде строки:

```
SELECT LOAD_FILE("C:/proverka");
```

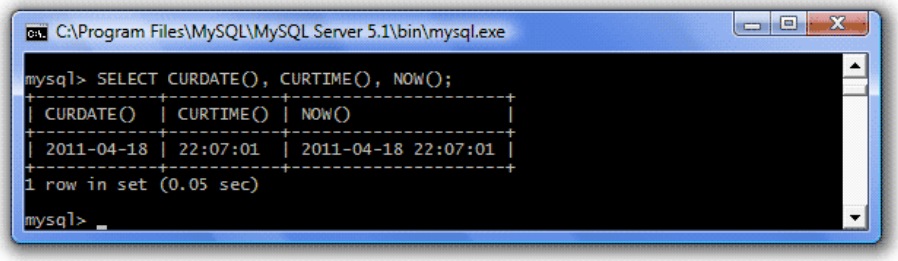



Обратите внимание, необходимо указывать абсолютный путь к файлу.

Функции, предназначенные для работы с календарными типами данных (наиболее применимые):

CURDATE(), CURTIME() и NOW() - функция возвращает текущую дату, текущее время,  текущую дату и время.

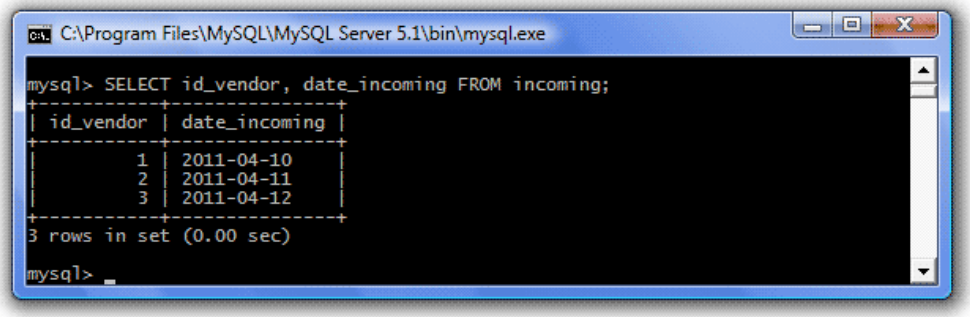
```
SELECT CURDATE(), CURTIME(), NOW();
```




Функции CURDATE() и NOW() удобно использовать для добавления в базу данных записей, использующих текущее время. Если бы мы хранили дату с типом datatime, то нам больше подошла бы функция NOW().

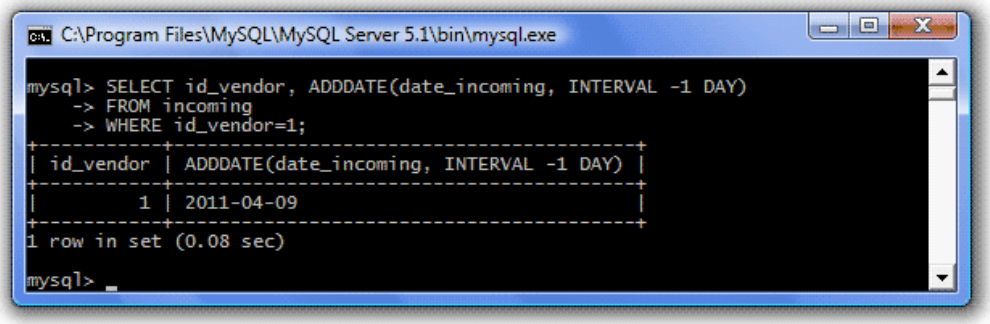
ADDDATE(date, INTERVAL value) - функция возвращает дату date, к которой прибавлено значение value. Значение value может быть отрицательным, тогда итоговая дата уменьшится.

```
SELECT id_vendor, date_incoming FROM incoming;
```




Уменьшим дату поставки на одни сутки:

```
SELECT id_vendor, ADDDATE(date_incoming, INTERVAL -1 DAY)
FROM incoming WHERE id_vendor=1;
```

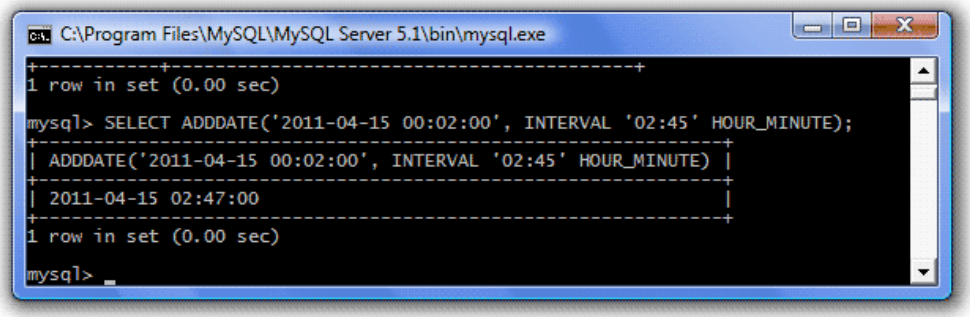



В качестве значения value могут выступать не только дни, но и недели (WEEK), месяцы (MONTH), кварталы (QUARTER) и годы (YEAR).

При использовании типа datatime можно использовать функцию ADDDATE и для времени. В качестве значения value в этом случае могут выступать секунды (SECOND), минуты (MINUTE), часы (HOUR) и их комбинации: минуты и секунды (MINUTE_SECOND), часы, минуты и секунды (HOUR_SECOND), часы и минуты (HOUR_MINUTE), дни, часы, минуты и секунды (DAY_SECOND), дни, часы и минуты (DAY_MINUTE), дни и часы (DAY_HOUR), года и месяцы (YEAR_MONTH).

Например, давайте к дате 15 апреля 2011 года две минуты первого прибавим 2 часа 45 минут:

```
SELECT ADDDATE('2011-04-15 00:02:00', INTERVAL '02:45' HOUR_MINUTE);
```



SUBDATE(date, INTERVAL value) - функция идентична предыдущей, но производит операцию вычитания, а не сложения.

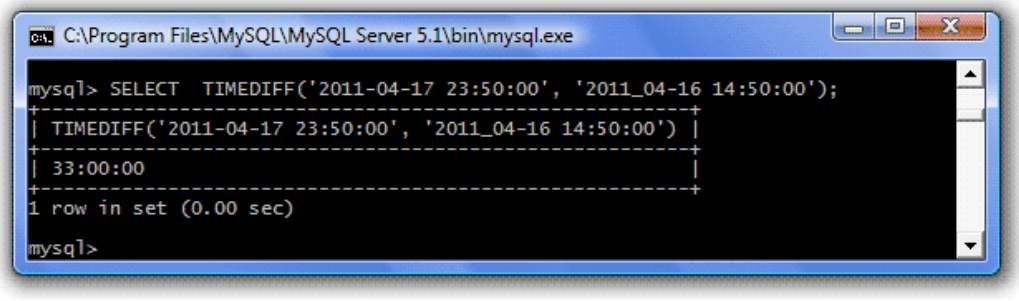
PERIOD_ADD(period, n) - функция добавляет n месяцев к значению даты period. Нюанс: значение даты должно быть представлено в формате YYYYMM.

TIMESTAMPADD(interval, n, date) - функция добавляет к дате date временной интервал n, значения которого задаются параметром interval. Возможные значения параметра interval:

- FRAC_SECOND – микросекунды
- SECOND - секунды
- MINUTE – минуты
- HOUR – часы
- DAY – дни
- WEEK – недели
- MONTH – месяцы
- QUARTER – кварталы
- YEAR – годы

TIMEDIFF(date1, date2) вычисляет разницу в часах, минутах и секундах между двумя датами.

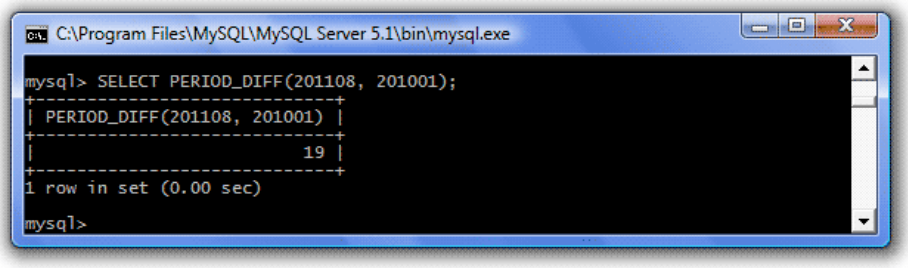
```
SELECT  TIMEDIFF('2011-04-17 23:50:00', '2011_04-16
14:50:00');
```




DATEDIFF(date1, date2) вычисляет разницу в днях между двумя датами.

PERIOD_DIFF(period1, period2) - функция вычисляет разницу в месяцах между двумя датами, представленными в формате YYYYMM. Давайте узнаем разницу между январем 2010 и августом 2011:

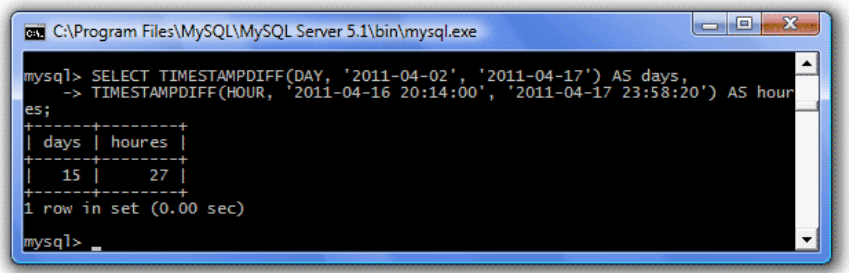
```
SELECT PERIOD_DIFF(201108, 201001);
```




TIMESTAMPDIFF(interval, date1, date2) функция вычисляет разницу между датами date2 и date1 в единицах, указанных в параметре interval. Возможные значения параметра interval:

- FRAC_SECOND – микросекунды
- SECOND – секунды
- MINUTE – минуты
- HOUR – часы
- DAY – дни
- WEEK – недели
- MONTH – месяцы
- QUARTER – кварталы
- YEAR – годы

```
SELECT TIMESTAMPDIFF(DAY, '2011-04-02', '2011-04-17') AS
days,
TIMESTAMPDIFF(HOUR, '2011-04-16 20:14:00', '2011-04-17
23:58:20') AS houres;
```




DATE(datetime) возвращает дату, отсекая время.

TIME(datetime) возвращает время, отсекая дату.

TIMESTAMP(date) функция принимает дату date и возвращает полный вариант со временем.

DAYNAME(date), DAYOFWEEK(date) и WEEKDAY(date) функции возвращают день недели, первая - его название, вторая - номер дня недели (отсчет от 1 - воскресенье до 7 - суббота), третья - номер дня недели (отсчет от 0 - понедельник, до 6 – воскресенье).

WEEK(date), WEEKOFYEAR(datetime) обе функции возвращают номер недели в году, первая для типа date, а вторая - для типа datetime, у первой неделя начинается с воскресенья, у второй - с понедельника

MONTH(date) и MONTHNAME(date) обе функции возвращают значения месяца. Первая - его числовое значение (от 1 до 12), вторая - название месяца.

QUARTER(date) функция возвращает значение квартала года (от 1 до 4).

YEAR(date) функция возвращает значение года (от 1000 до 9999).

DAYOFYEAR(date) возвращает порядковый номер дня в году (от 1 до 366).

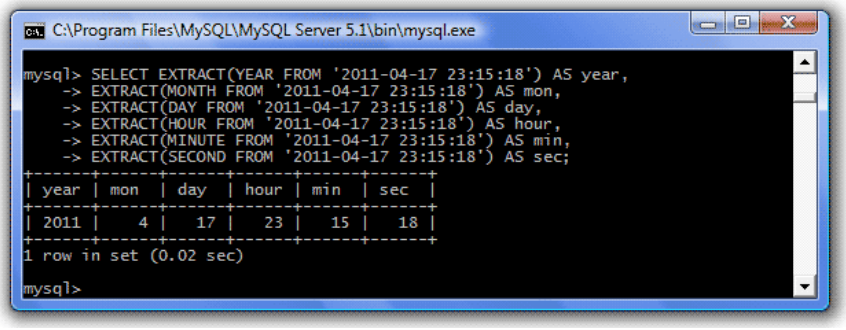
HOUR(datetime) возвращает значение часа для времени (от 0 до 23).

MINUTE(datetime) возвращает значение минут для времени (от 0 до 59).

SECOND(datetime) возвращает значение секунд для времени (от 0 до 59).

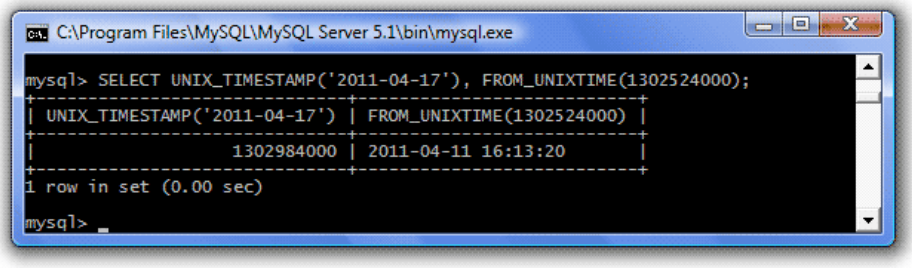
EXTRACT(type FROM date) возвращает часть date определяемую параметром type:

```
SELECT EXTRACT(YEAR FROM '2011-04-17 23:15:18') AS year,
EXTRACT(MONTH FROM '2011-04-17 23:15:18') AS mon, EXTRACT(DAY
FROM '2011-04-17 23:15:18') AS day, EXTRACT(HOUR FROM '2011-04-
17 23:15:18') AS hour, EXTRACT(MINUTE FROM '2011-04-17
23:15:18') AS min, EXTRACT(SECOND FROM '2011-04-17 23:15:18')
AS sec;
```




UNIX_TIMESTAMP(date) и FROM_UNIXTIME(n) взаимообратные функции. Первая преобразует дату в количество секунд, прошедших с 1 января 1970 года. Вторая, наоборот, принимает число секунд, с 1 января 1970 года и преобразует их в дату:

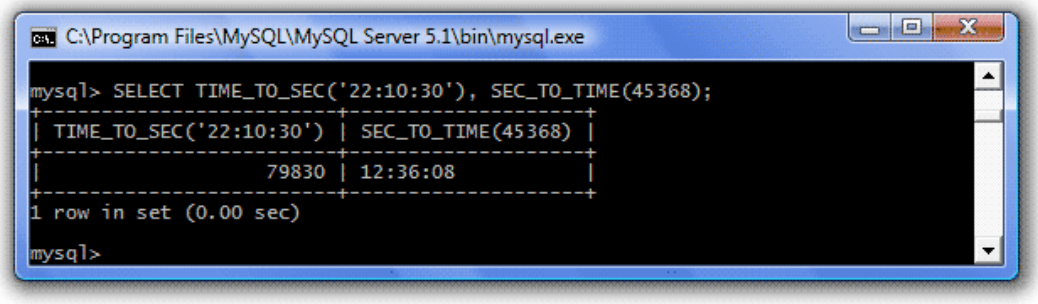
```
SELECT UNIX_TIMESTAMP('2011-04-17'),
FROM_UNIXTIME(1302524000);
```




TIME_TO_SEC(time) и SEC_TO_TIME(n) взаимообратные функции. Первая преобразует время в количество секунд, прошедших от

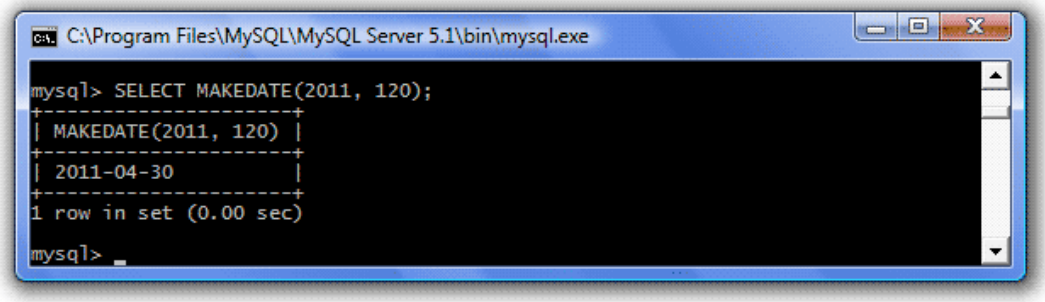
начала суток. Вторая, наоборот, принимает число секунд с начала суток и преобразует их во время:

```
SELECT TIME_TO_SEC('22:10:30'), SEC_TO_TIME(45368);
```




MAKEDATE(year, n) функция принимает год и номер дня в году и преобразует их в дату:

```
SELECT MAKEDATE(2011, 120);
```




## 6. Задания к лабораторной работе

6.1. Восстановить схемы БД по примерам и заполнить данными.

6.2. Воспроизвести все примеры и написать запросы:

6.2.1. Выполнить следующие запросы с представленными ранее данными:

- SELECT * FROM topics WHERE id_author=4;
- SELECT * FROM topics WHERE id_author>2;
- SELECT * FROM topics WHERE id_author IS NOT NULL;

- SELECT * FROM topics WHERE id_author IS NULL;
- SELECT * FROM topics WHERE id_author BETWEEN 1 AND 3;
- SELECT * FROM topics WHERE id_author NOT IN (1, 4);
- SELECT * FROM topics WHERE topic_name LIKE 'вел%';
- SELECT * FROM topics WHERE topic_name NOT LIKE 'вел%';
- SELECT * FROM topics WHERE topic_name LIKE %клуб%';
- SELECT * FROM topics WHERE topic_name LIKE _ рыб%';
- SELECT name FROM users WHERE id_user IN (SELECT id_author FROM topics WHERE topic_name='велосипеды');
- SELECT id_topic, COUNT(id_topic) FROM posts GROUP BY id_topic;
- SELECT id_topic, COUNT(id_topic) FROM posts GROUP BY id_topic HAVING COUNT(id_topic) > 2;

6.2.2. Написать запросы:

- выполнить внутреннее объединение таблиц topics и users;
- выполнить внутреннее объединение таблиц topics и users по полю id_author и id_user (значения поля id_author таблицы topics совпадают со значениями поля id_user таблицы users);
- выполнить левое внешнее объединение таблиц topics и users по полю id_author id_user;
- выполнить правое внешнее объединение таблиц topics и users по полю id_author id_user;
- найти самое большое, самое маленькое значение и количество записей в столбце id_author таблицы posts;
- дополнить строку адресов справа до 20 знаков точками;
- перевести имена всех авторов в верхний регистр;
- вычислить среднюю длину сообщения;

- вывести следующие параметры текущей даты: порядковый номер дня в году, день недели, номер недели в году, наименование месяца, номер квартала года и год;
- вычислить количество дней до нового года.

## 7. Отчетность по лабораторной работе

7.1. В качестве приложения к отчету представить дамп базы данных в формате SQL.

7.2. В качестве приложения к отчету представить список SQL запросов с результатами выполнения запросов.

# ЛАБОРАТОРНАЯ РАБОТА №6. ОПЕРАЦИИ НАД ОТНОШЕНИЯМИ В РЕЛЯЦИОННЫХ БД (SQL-ИНЪЕКЦИИ)

## 1. Цель работы

1.1. Рассмотрение возможных уязвимостей при построении запросов (SQL-инъекции).

## 2. Содержание работы

2.1. Установить и настроить среду выполнения лабораторной работы по SQL-инъекциям (sql-lab-master.zip[21]): развернуть из архива сайт, создать БД, импортировать данные, установить параметры доступа к БД.

2.2. Выполнить задания лабораторной работы по SQL-инъекциям (5 заданий).

## 3. Общие сведения

SQL-инъекции - это атака на базу данных, которая позволит выполнить некоторое действие, которое не планировалось создателем скрипта.

Атака с использованием SQL-инъекции состоит из вставки SQL-запроса в реализуемый запрос через входные данные, полученные от клиента через поля форм приложения или параметры запросов.

Успешное использование SQL-инъекции может обеспечить:

- доступ к конфиденциальным данным из базы данных (позволяют злоумышленникам подделывать идентификационные данные);
- возможность изменять данные (вставлять/обновлять/удалять);
- выполнять административные операции с базой данных
- вызывать команды операционной системы.

SQL-инъекции более критичны в приложениях PHP и ASP из-за преобладания старых функций обработки данных.

[21] Исходный код проекта и описание размещены на сайте http://toogle.github.io/sql-lab/

Пример SQL-инъекции (слепая инъекция SQL): при обращении к странице указывается ее идентификатор в качестве параметра. В данном случае злоумышленник может проверить наличие уязвимости: если встраивание инъекции в параметр приведет к невозможности отображения страницы, то значит, что этот параметр не контролируется.

Строка запроса (штатный режим работы, ответ – содержимое страницы):

```
https://example.com/items.php?id=2
```

При обработке запроса скрипт получает данные о странице из базы данных следующим запросом:

```
SELECT title, description, body FROM items WHERE ID = 2
```

В штатном (нормальном) режиме работы скрипт получит данные страницы из БД, осуществит ее форматирование и отдаст результат пользователю, если указанная страница существует.

Злоумышленник может отправить запрос:

```
https://example.com/items.php?id=2 and 1=2
```

Если скрипт не имеет механизмов защиты от инъекций, то в СУБД будет передан запрос (который никогда не вернет данные, т.к. 1=2 всегда ложно):

```
SELECT title, description, body FROM items WHERE ID = 2
and 1=2
```

Отсутствие вывода сигнализирует о том, что параметр `id` никак не проверяется и возможно внедрение SQL-инъекций в данный запрос.

Пример SQL-инъекции (получение скрытых данных из базы данных): получение дополнительных данных из таблиц.

Строка запроса (электронного магазина) может содержать наименование категории товара, доступного для заказа. Типовой запрос имеет следующий вид:

```
https://example.com/products?category=Gifts
```

Скрипт выполняет запрос к БД, дополняя его параметром наличия возможности заказа товара `instock = 1`:

```
SELECT * FROM products WHERE category = 'Gifts' AND in-
stock = 1
```

Злоумышленник может отправить запрос:

```
https://example.com/products?category=Gifts'--
```
или
```
https://example.com/products?category=Gifts'+OR+1=1--
```

Скрипт выполняет запрос к БД:

- в первом случае результатом будет список всех товаров из категории 'Gifts' (‘--’ закомментирует остаток строки, сделав запрос более общим[22]):

```
SELECT * FROM products WHERE category = 'Gifts'-- ' AND
instock = 1
```

- во втором случае результатом будет список всех товаров в БД (1=1 всегда истино, параметр категории будет проигнорирован):

```
SELECT * FROM products WHERE category = 'Gifts' OR 1=1-- '
AND instock = 1
```

[22] Сервер MySQL поддерживает следующие способы задания комментариев: с помощью символа `#', за которым следует текст комментария до конца строки; с помощью двух символов --, за которыми идет текст комментария до конца строки; и (для многострочных комментариев) с помощью символов /* (начало комментария) и */ (конец комментария). При использовании для комментирования способа с -- (двойное тире) требуется наличие хотя бы одного пробела после второго тире

Пример SQL-инъекции: изменение логики обработки результатов запроса скриптом.

Традиционный запрос проверки прав доступа пользователя реализуется запросом (пользователь – admin, пароль – 1234):

```
SELECT * FROM users WHERE username = 'admin' AND password
= '1234'
```

Результат запроса будет пустой, если пользователя не существует или пароль не подходит. Иначе, результат может содержать данные профиля пользователя (сам факт наличия хотя бы одной записи в результате сигнализирует о том, что пользователь ввел правильный логин и пароль).

Злоумышленник может переписать запрос следующим образом:

```
SELECT * FROM users WHERE username = 'admin'-- ' AND pass-
word = ''
```

Запрос будет выполнен и пользователь авторизован (если он существует) без проверки пароля. Таким образом, возможна авторизация под любым пользователем. Также возможен сценарий с подбором имен пользователей с необходимыми полномочиями.

Пример SQL-инъекции: получение данных о СУБД.

С помощью следующего запроса можно получить список всех таблиц СУБД:

```
SELECT * FROM information_schema.tables
```

К другим способам проведения атак относят:

- Использование объединение запросов (UNION) - у злоумышленника появляется возможность совмещать результаты одних запросов с результатми других запросов, которые отображаются пользователю (например, соединить данные таблиы

товаров и пользователей, чтобы при запросе списка товаров получить и информацию о пользователях).

- Формирование запросов, содержащих ошибки - на основе полученных сообщений об ошибках злоумышленник может сформировать представление об инфраструктуре базы данных.
- Формирование запросов, «замедляющих работу» СУБД - злоумышленник направляют SQL-запрос к базе данных, вынуждая ее сделать задержку, прежде чем она подтвердит или опровергнет полученный запрос.

Наиболее эффективным способом предотвражения инъекция является контроль пользовательского ввода: считается, что все входящие данные по-умолчанию содержат уязвимости до их проверки. При этом не важно кто отправляет запросы: внутренние или внешние пользователи.

Развернув сайт (архив) вы увидите 5 лабораторных работы, каждая из которых предполагает выполнение SQL-инъекции. Описание к лабораторной работе (содержится в архиве) содержит необходимый минимум информации для проведения SQL-инъекций.

**4. Задания к лабораторной работе**

4.1. Развернуть и настроить среду для выполнения заданий (архив). Загрузить данные в базу данных.

4.2. Составить запросы и провести SQL-инъекции над тестовой базой данных (дамп базы данных присутствует в архиве).

**5. Отчетность по лабораторной работе**

5.1. В качестве приложения к отчету представить запросы к сайту при выполнении SQL-инъекций и снимок экрана с результатами проведения SQL-инъекции.

# ЛАБОРАТОРНАЯ РАБОТА №7. СПЕЦИАЛИЗИРОВАННЫЕ СУБД

## 1. Цель работы

1.1. Рассмотрение реализаций СУБД.

## 2. Содержание работы

2.1. Установить и настроить среду выполнения лабораторной работы. В работе используется платформа 1С:Предприятие 8[23].

2.2. Дополнительно можно скачать консоль запросов.

2.2. Восстановить схему используемой в конфигурации базы данных.

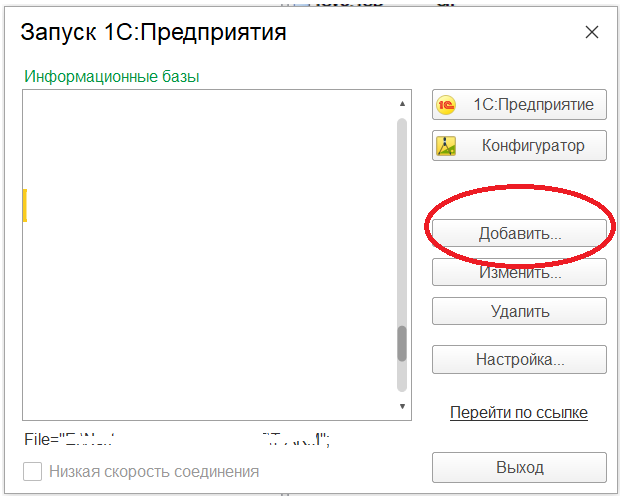
## 3. Общие сведения

Система 1С:Предприятие работает в двух основных режимах:

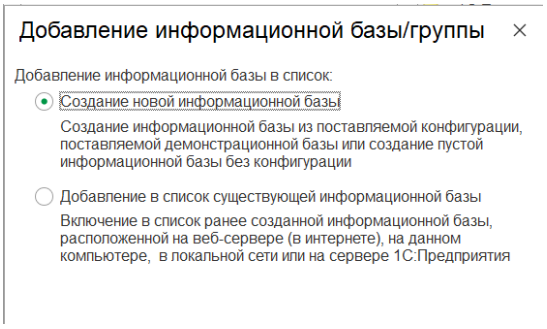
- Предприятие – В данном режиме осуществляется работа с данными в выбранной конфигурации.
- Конфигуратор – В данном режиме осуществляется создание конфигурация (разработка и программирование) и ее отладка.

Рассмотрим процесс создания конфигурации на основе уже существующей конфигурации.

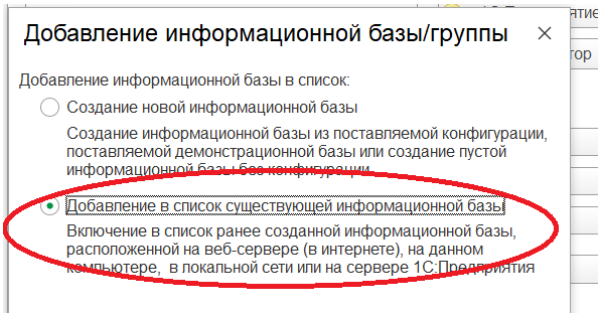
[23] Версия для обучения программированию доступна для скачивания по ссылке - https://online.1c.ru/catalog/free/learning.php



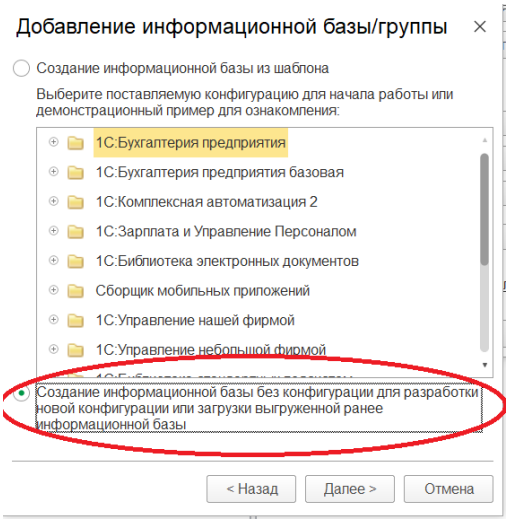

При создании базы данных выберите «Создание новой ИБ».



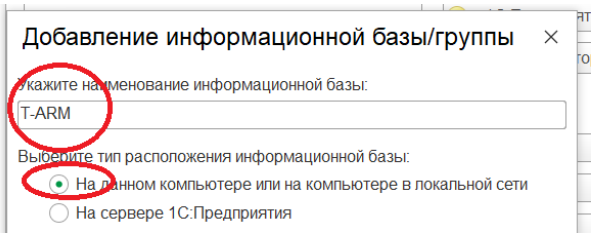

При добавлении уже существующей базы данных выберите соответствующий пункт.



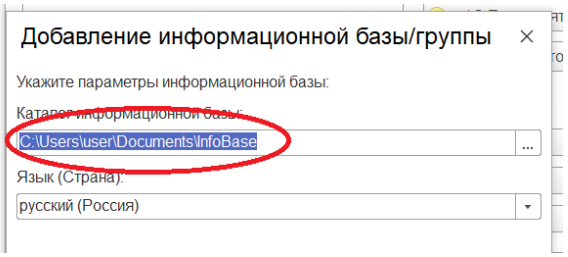

После этого нужно будет указать путь к каталогу БД.
Рассмотрим далее ситуацию создания БД.




Определяете имя базы данных и форму хранения данных (1С:Предприятие может работать в файловом варианте, когда все данные информационной базы располагаются в одном файле - в файловой СУБД или в клиент-серверном варианте, когда используются внешние СУБД).

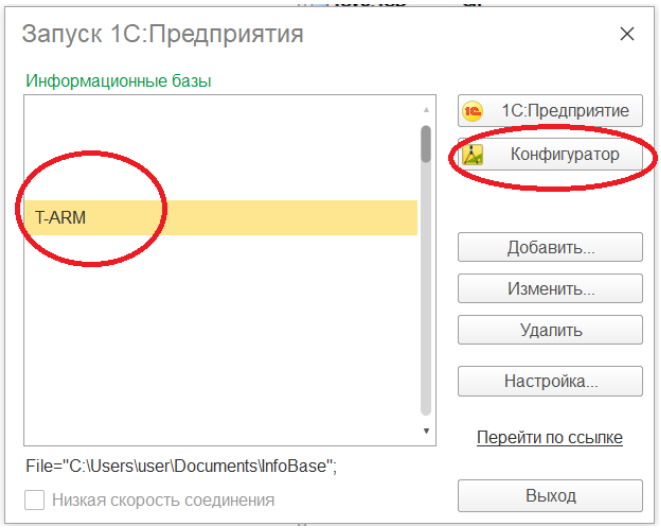



Задаете каталог хранения БД.

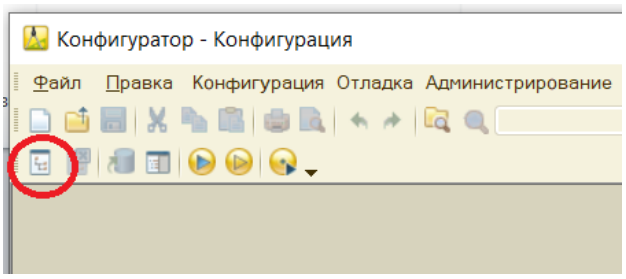



В последней форме можно указать параметры запуска 1С (в данном случае – по умолчанию).

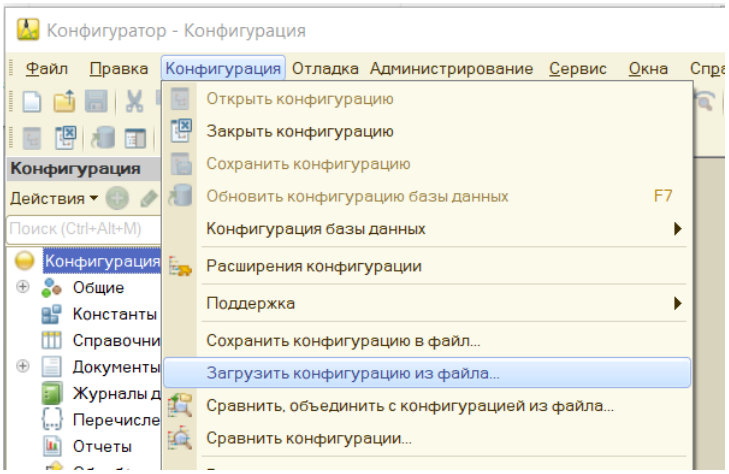
Выбираете в списке созданную БД, и открываете ее в режиме конфигуратора.



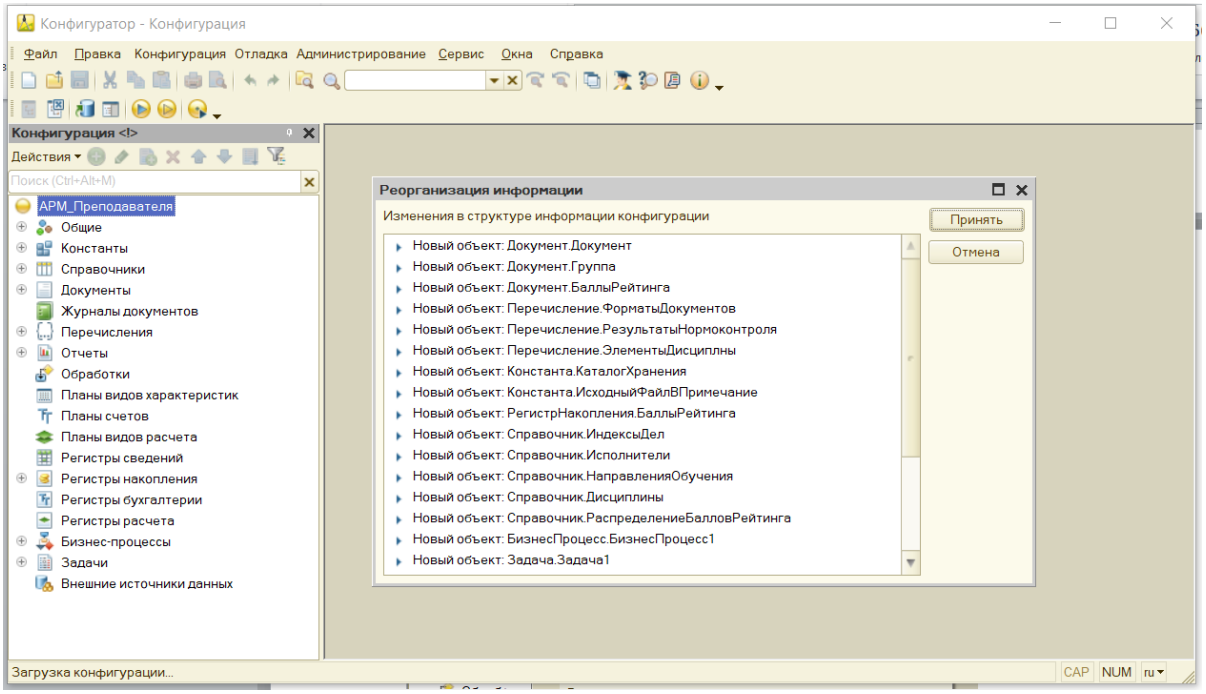

После запуска конфигуратора необходимо открыть конфигурации (кнопка на панели или через меню Конфигурация→Открыть конфигурацию.




Далее загрузить конфигурацию из файла.



В процессе загрузки будет предложено преобразовать существующую структуру БД к загружаемой структуре.




После загрузки конфигурации можно закрыть конфигуратор и запустить систему в режиме предприятия для ввода данных.

Данные информационной базы могут быть сохранены отдельно от конфигурации (с целью резервного копирования или для переноса).

Загрузка и выгрузка данных производится в режиме конфигуратора (меню Администрирование: «Выгрузить информационную базу» или «Загрузить информационную базу»).

## 4. Объекты конфигурации

Объекты конфигурации - это составные элементы, из которых складывается любое прикладное решение.

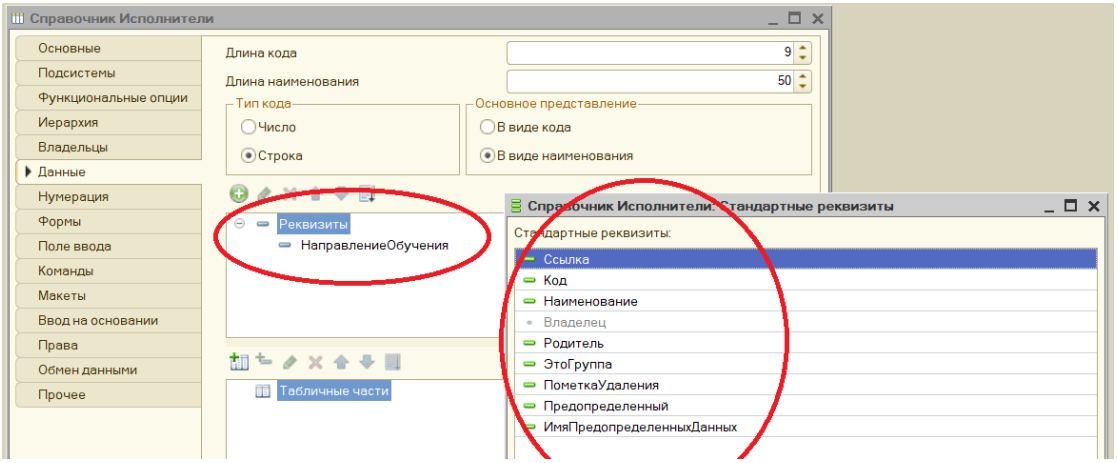
Они представляют собой проблемно-ориентированные объекты, поддерживаемые на уровне платформы. По большому счету задача разработчика заключается в том, чтобы собрать из этих объектов, как из конструктора, необходимую структуру прикладного решения и за-

тем описать специфические алгоритмы функционирования и взаимодействия этих объектов, отличающиеся от их типового поведения.

Для того чтобы стандартизировать и упростить процесс разработки и модификации прикладных решений, разработчику предоставляется графический интерфейс, с помощью которого он имеет возможность описать состав объектов.

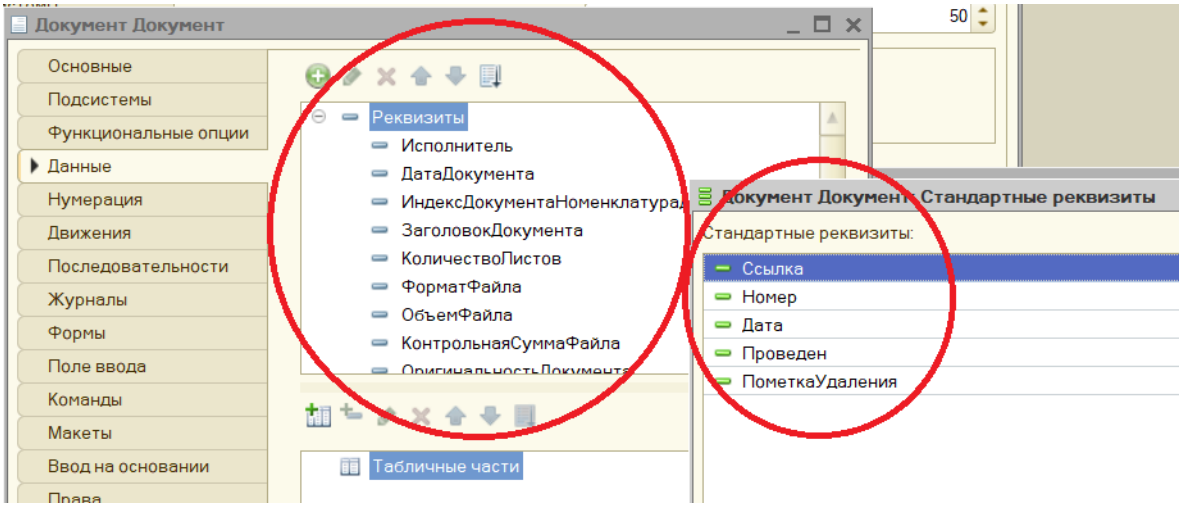
В предложенной для изучения конфигурации использованы следующие объекты конфигурации:

Справочники позволяют хранить в информационной базе данные, имеющие одинаковую структуру и списочный характер. Помимо кода и наименования, каждый элемент справочника, как правило, содержит некоторую дополнительную информацию, которая подробно описывает этот элемент. Набор такой информации является одинаковым для всех элементов конкретного справочника. Некоторые реквизиты могут быть таблицами (структура подобных данных определяется в табличной части).

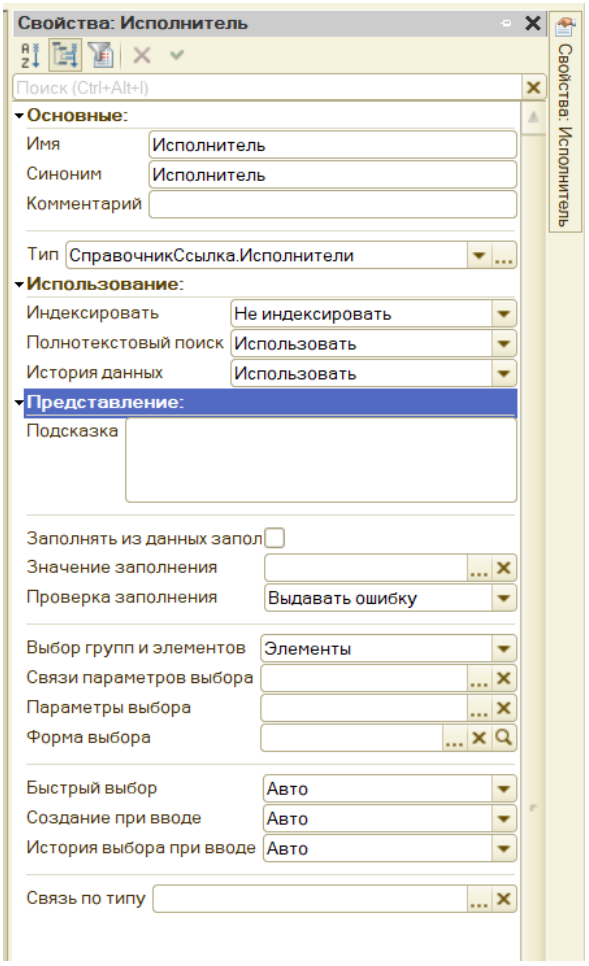



Документы позволяют хранить в прикладном решении информацию о совершенных операциях или о событиях.

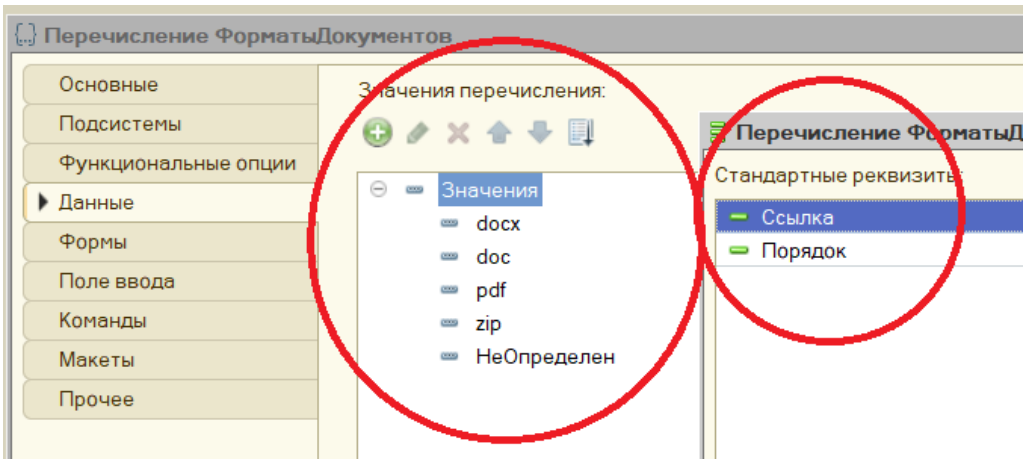
Каждый документ характеризуется номером, датой и временем. Также как и в случае со справочниками, помимо стандартных реквизитов, каждый документ, как правило, содержит некоторую дополнительную информацию, которая подробно описывает этот документ.




В свойствах реквизитов описаны параметры данного реквизита: параметры его использования и отображения.

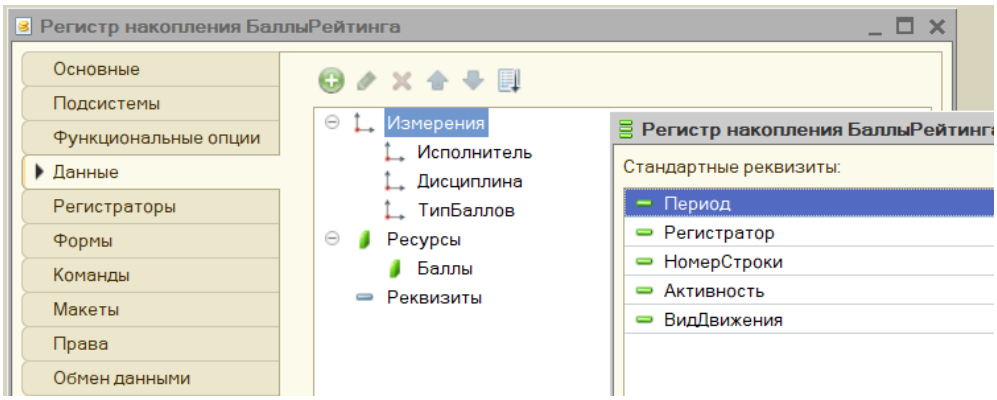


Перечисления позволяют хранить в информационной базе наборы значений, которые не изменяются в процессе работы прикладного решения.

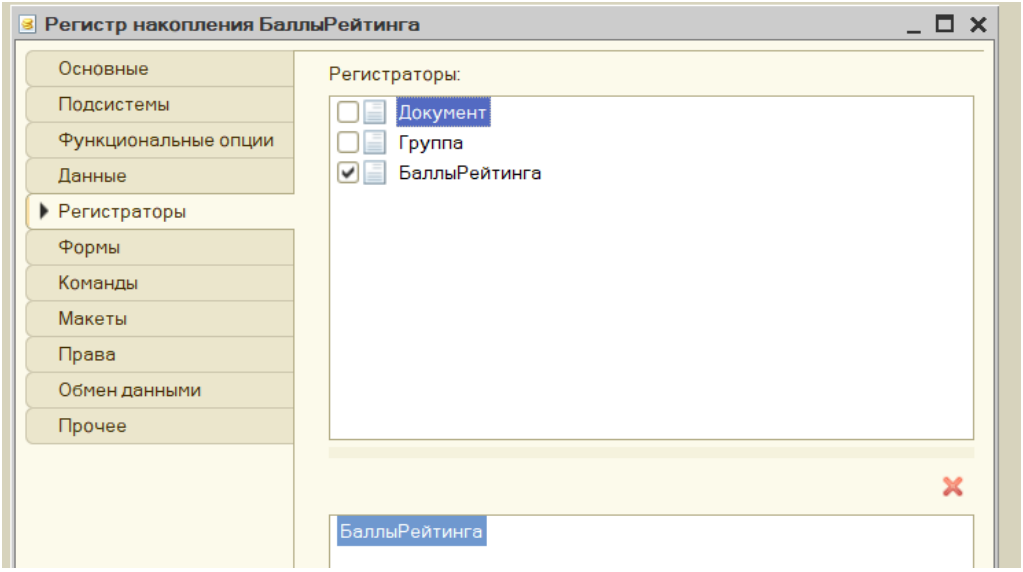


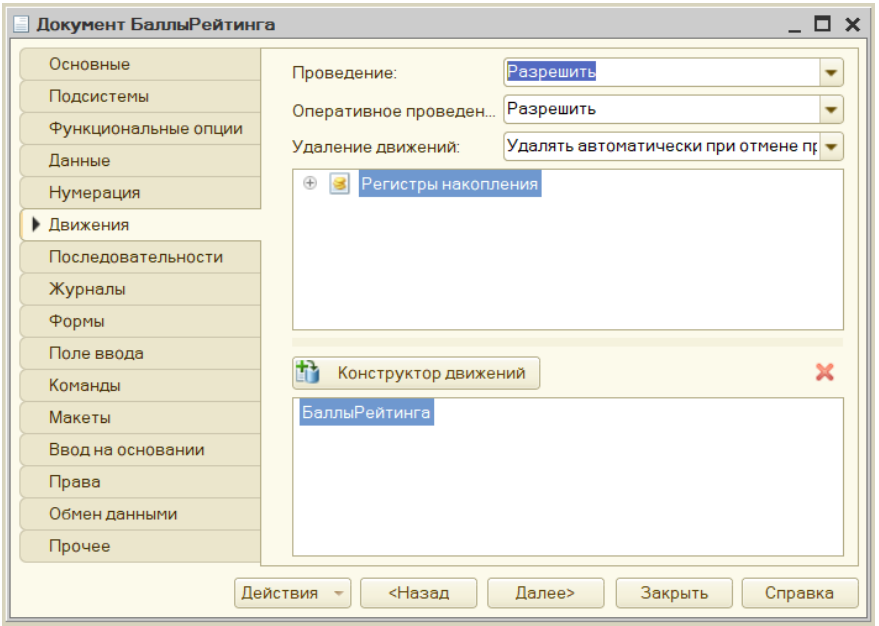

Регистры накопления составляют основу механизма учета движения данных (финансов, товаров, материалов и т. д.). Регистр накопления образует многомерную систему измерений и позволяет «накапливать» числовые данные в разрезе нескольких измерений.

Движения, в общем случае, могут либо добавлять некоторые приращения к хранимым ресурсам, либо отнимать их. Если должно выполняться увеличение хранимых ресурсов, - такое движение называется движением прихода («+»), если уменьшение хранимых ресурсов - движением расхода («-»).



Изменение состояния регистра накопления происходит, как правило, при проведении документа. Поэтому каждая запись регистра связана с определенным документом, называемым регистратором.

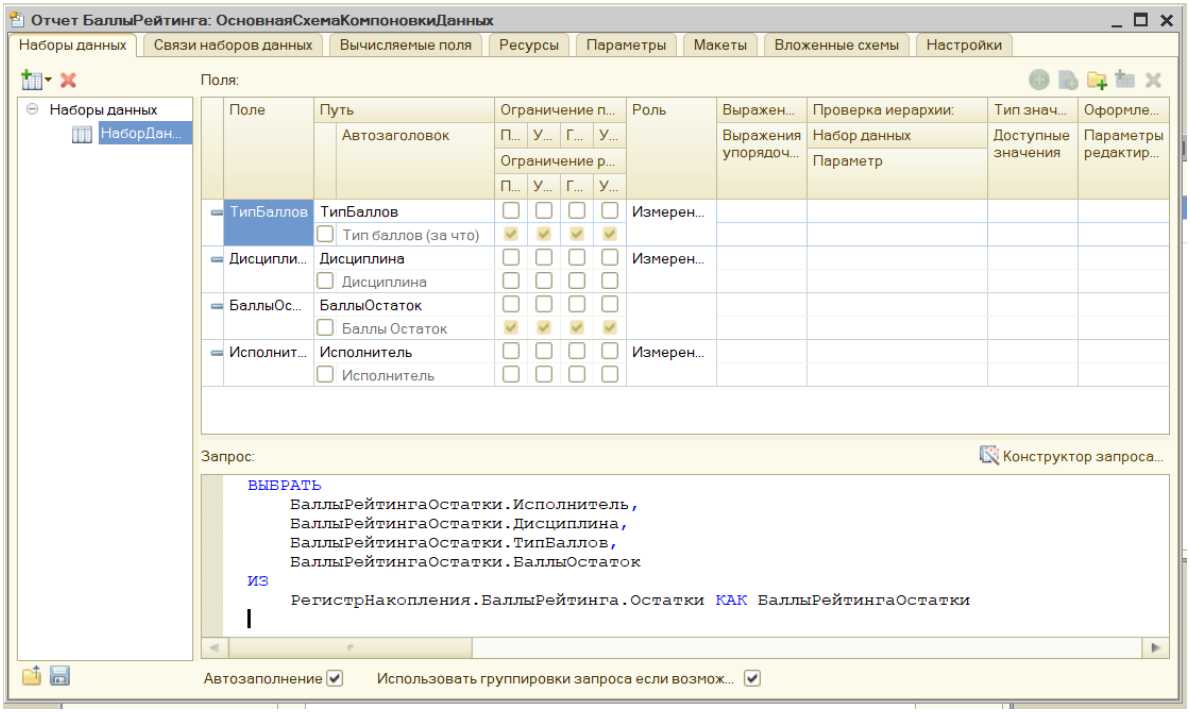



Отчеты предназначены для обработки накопленной информации и получения сводных данных в удобном для просмотра и анализа виде. Конфигуратор позволяет формировать набор различных отчетов, достаточных для удовлетворения потребности пользователей системы в достоверной и подробной выходной информации.

Как правило, для формирования выходных данных отчет использует систему компоновки данных.

Отчет может содержать одну или несколько форм, с помощью которых, при необходимости, можно организовать ввод каких-либо параметров, влияющих на ход алгоритма.




## 5. Задания к лабораторной работе

5.1. Развернуть и настроить среду для выполнения заданий. Загрузить конфигурацию базу данных.

5.2. Восстановить схему базы данных (связи реквизитов объектов конфигурации).

## 6. Отчетность по лабораторной работе

6.1. В качестве приложения к отчету представить схему базы данных.

# ЛАБОРАТОРНАЯ РАБОТА № 8. БАЗЫ ДАННЫХ ВРЕМЕННЫХ РЯДОВ (TIME SERIES DATABASE)

## 1. Цель работы

1.1. Рассмотрение реализаций СУБД.

## 2. Содержание работы

2.1. Установить систему сбора и хранения данных временных рядов (influxdb + Grafana).

2.2. Оптимально предоставить данные для заполнения через jMeter - характеристики доступа к сайту.

## 3. Общие сведения

Временные ряды – один из наиболее часто встречающихся в аналитической практике объектов. Временной ряд – это статистика серии наблюдений за одним и тем же явлением, параметром какого-либо процесса, на протяжении некоторого времени. Каждому результату наблюдения (измерению) соответствует время, когда это наблюдение было сделано, или его порядковый номер – опять же, по шкале времени. Таким образом, при анализе временных рядов учитываются не только базовые статистические закономерности, но и взаимосвязь измерений со временем.

Такого рода данные требуются и накапливаются при самых разных задачах и для самых разных нужд. Можно привести несколько примеров:

- статистика использования сайта (посещения сайта, аппаратных и сетевых ресурсов и т.д.);
- финансовая, биржевая, актуарная статистика;
- макроэкономическая статистика.

Популярность специализированных хранилищ связана с интенсивным ростом интеграции информационных технологий: Big Data, социальные сети, IoT, мониторинг highload-инфраструктуры.

Базы данных временных рядов (TSBD, time series database) позволяют пользователям создавать, считать, обновлять и удалять различные временные ряды и организовывать их неким образом. Сервер часто поддерживает ряд основных вычислений, которые работают на ряды в целом, например, умножая, складывая или иным образом комбинируя различные временные ряды в новый временной ряд.

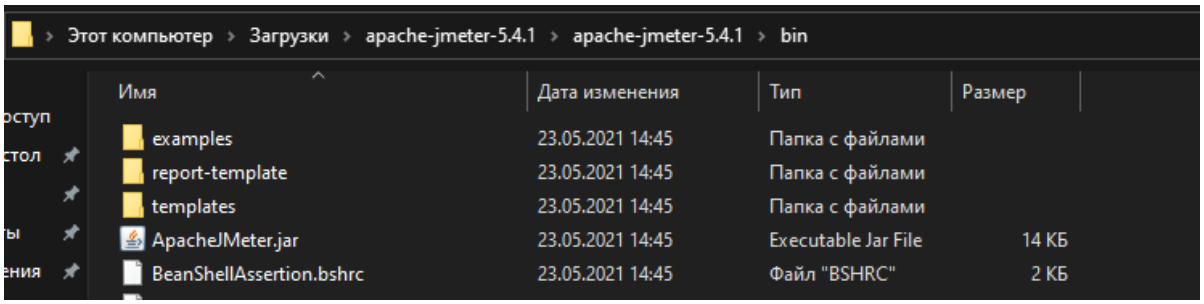
Среди специальных TSBD преобладают продукты, основанные на свободной лицензии: Open TSDB, InfluxDB, Geras, Druid и пр. Стабильно лидирует InfluxDB.

Реализация функционала базы данных временных рядов возможна в обычной реляционной БД на основе SQL при условии, что программное обеспечение базы данных поддерживает одновременно большие двоичные объекты (BLOB) и пользовательские функции. Но эффективность такой системы также будет невысока.

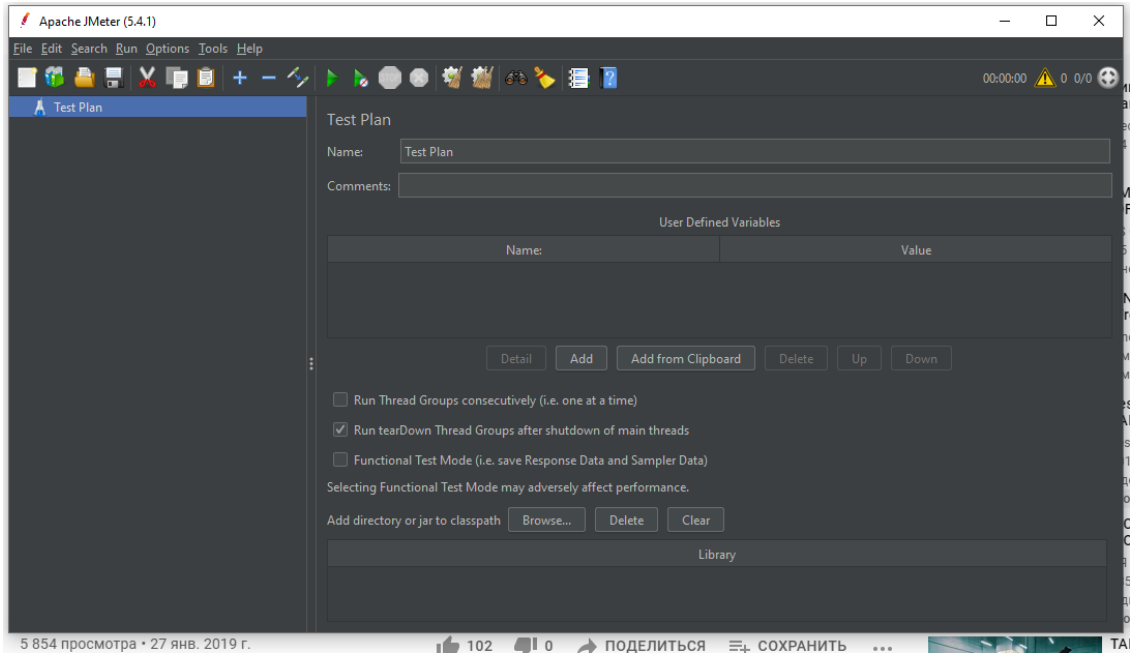
Почему это выгодно технически: В TSBD практически не наблюдается деградация (или она незначительна) при увеличении количества запросов (увеличении нагрузки на СУБД) в отличии от обычных СУБД.

## 4. Описание работы

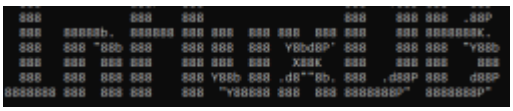
4.1. С официального сайта скачиваем apache JMeter.



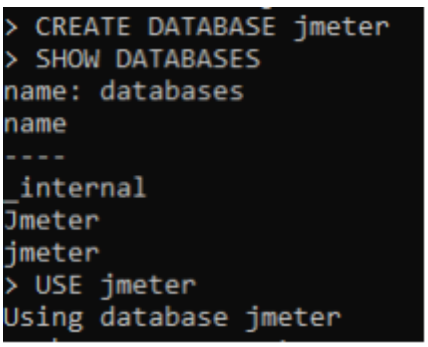

4.2. Открываем среду для написания теста.



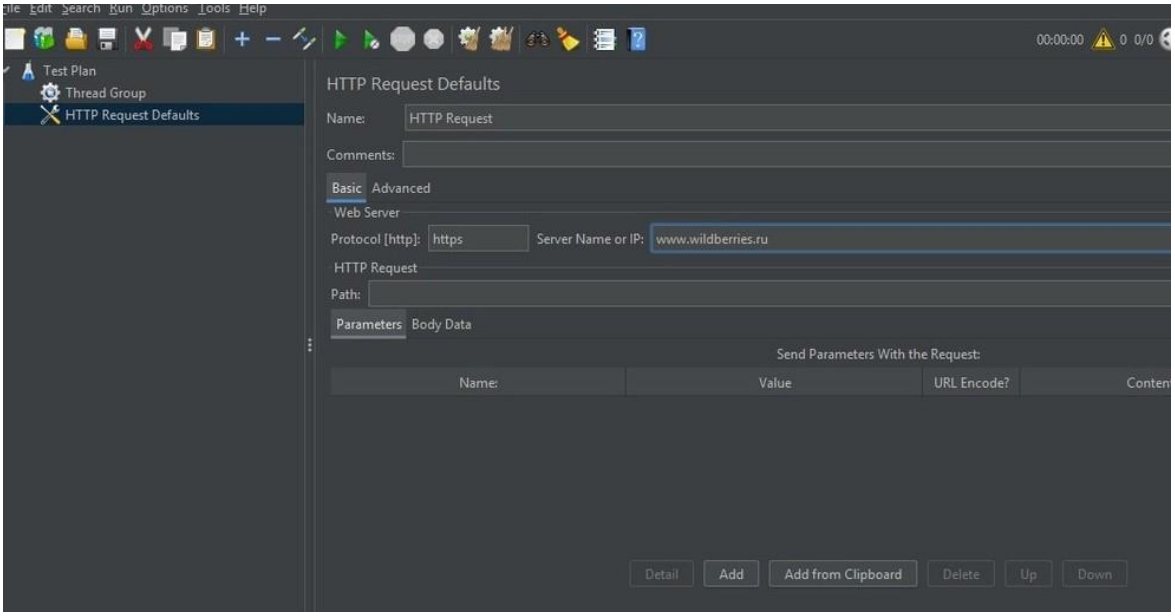

4.3. Скачиваем и настраиваем InfluxDB, Grafana. Запускается файл influxd.



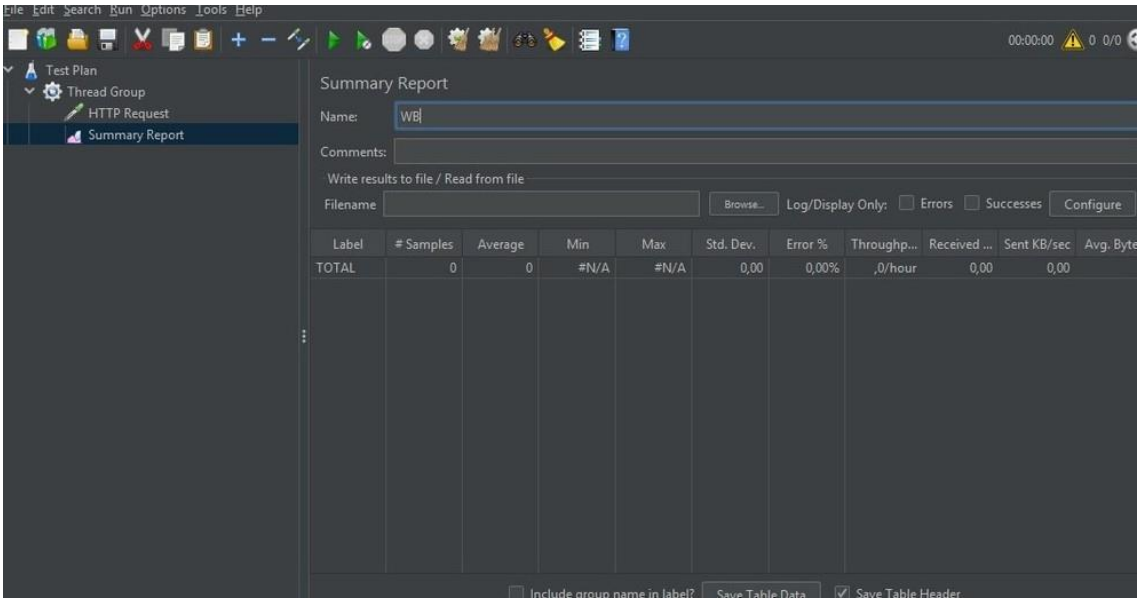
4.4. В первую очередь необходимо создать в Influxdb базу данных jmeter, в которую будет записываться полученная информация по результатам тестировании в jMeter. Для этого нужно открыть файл influx:



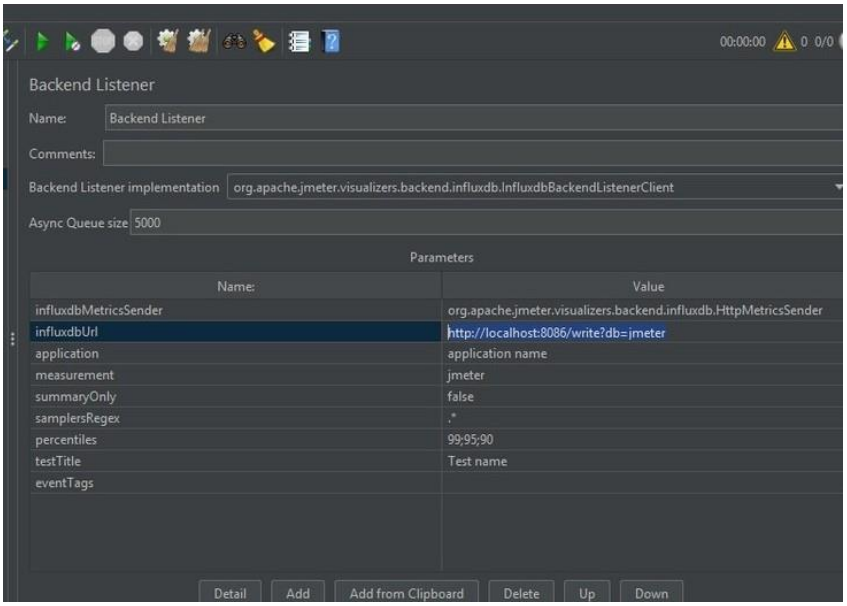
4.5. Далее в Jmeter необходимо создать тест для сайта, который будет тестироваться (в примере, сайт Wildberries). Для этого вводится URL сайта в сценарий теста, количество пользователей, период тестирования и количество запросов (заполняется HTTP Request Defaults).



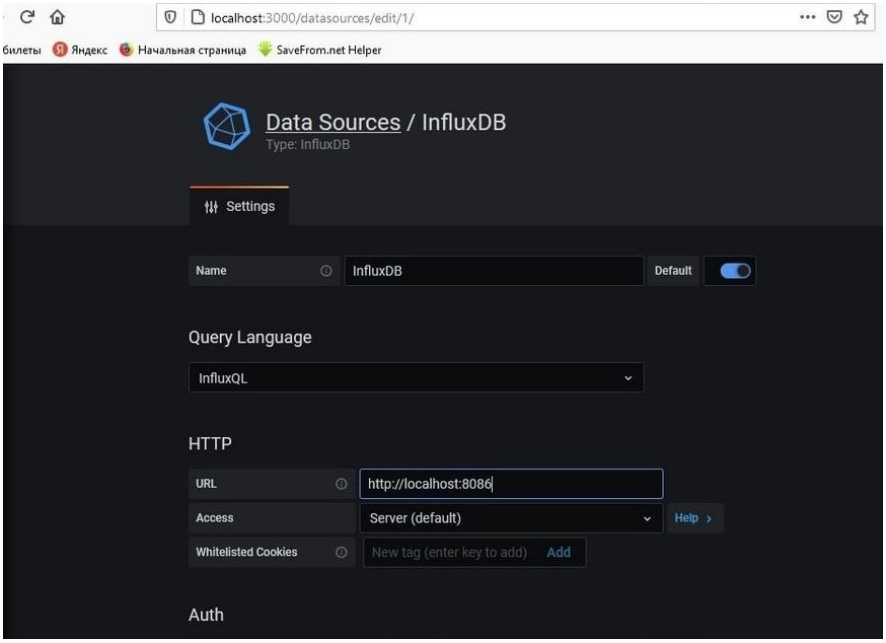

4.6. Для сбора статистики по тестированию создается SummaryReport и GraphResult.

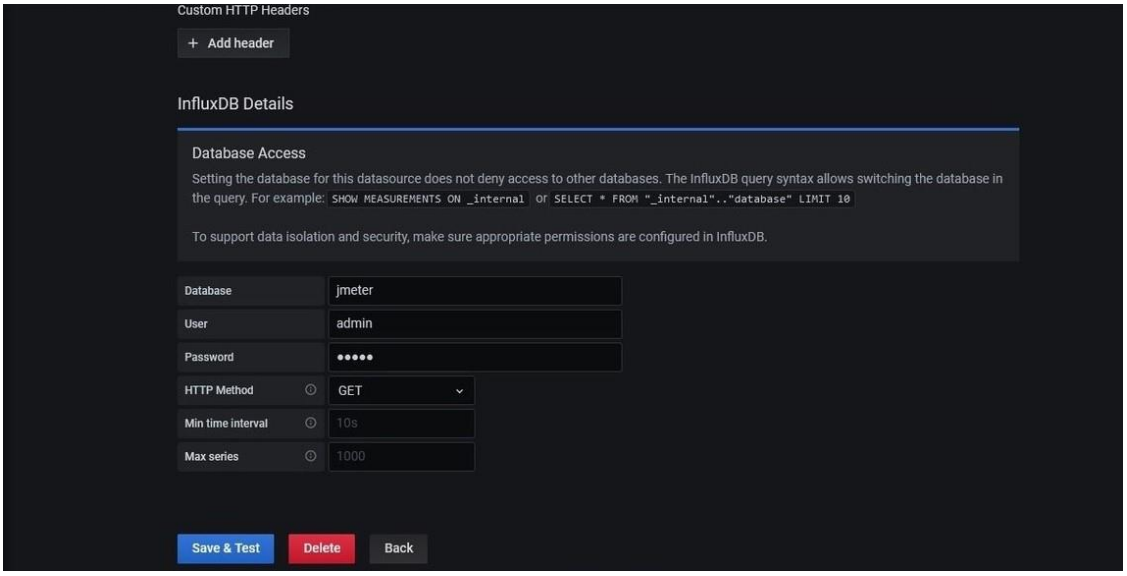


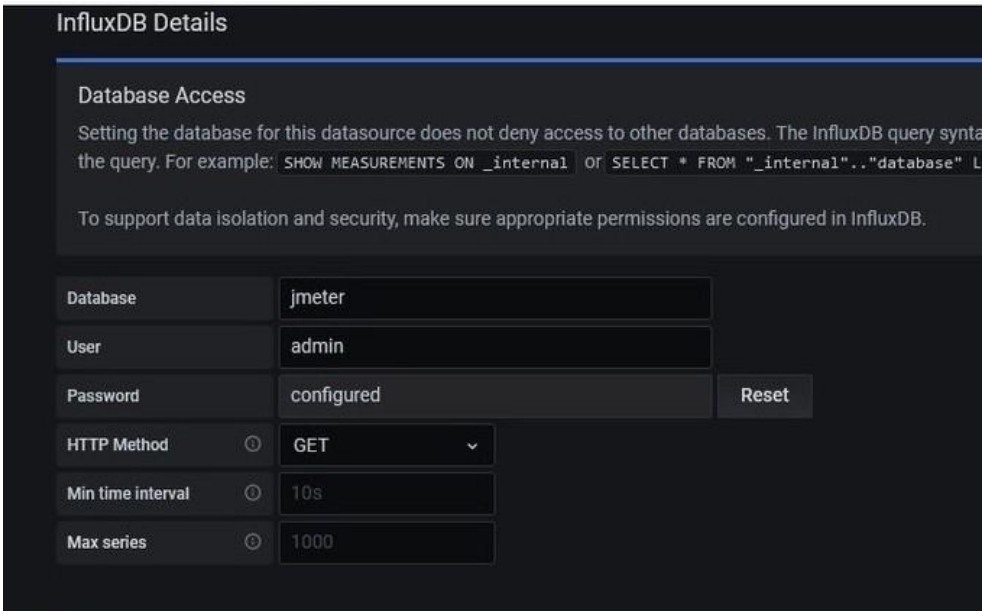
4.7. Далее необходимо осуществить активацию Backend Listener и связать его с созданной БД.




4.8. Связь Grafana с Influxdb осуществляется следующим образом. Перейти на http://localhost:3000. Далее в настройках DataSources указывается БД, в которую записываются данные по проведенному тестированию.

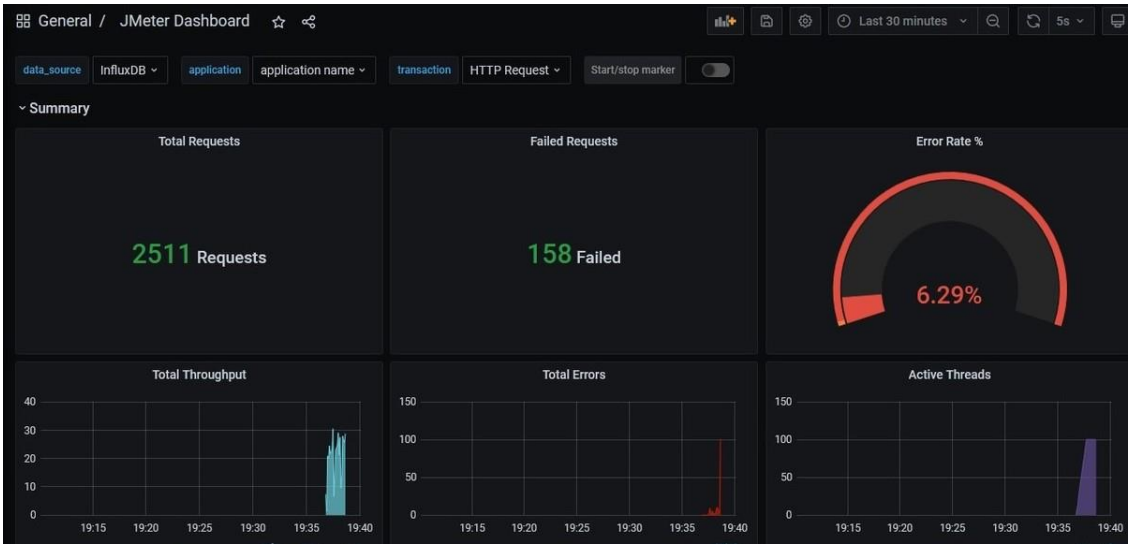


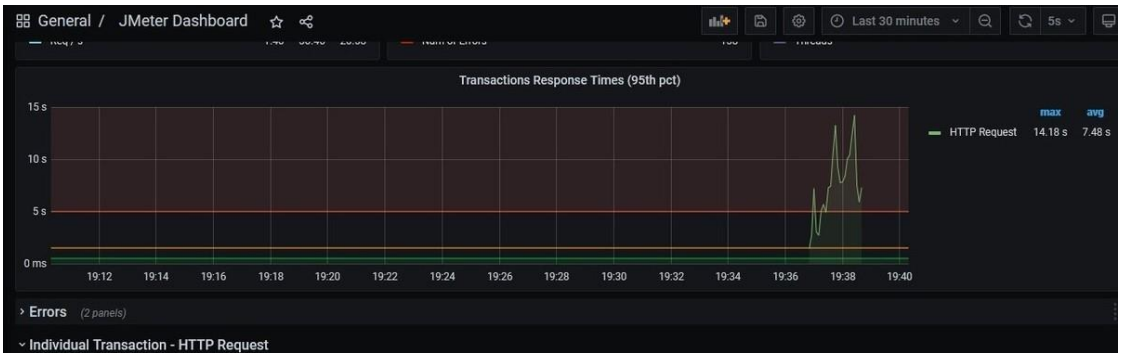



4.9. Далее устанавливается DashBoard для представления данных, полученных во время тестирования, и добавление его в Grafana.

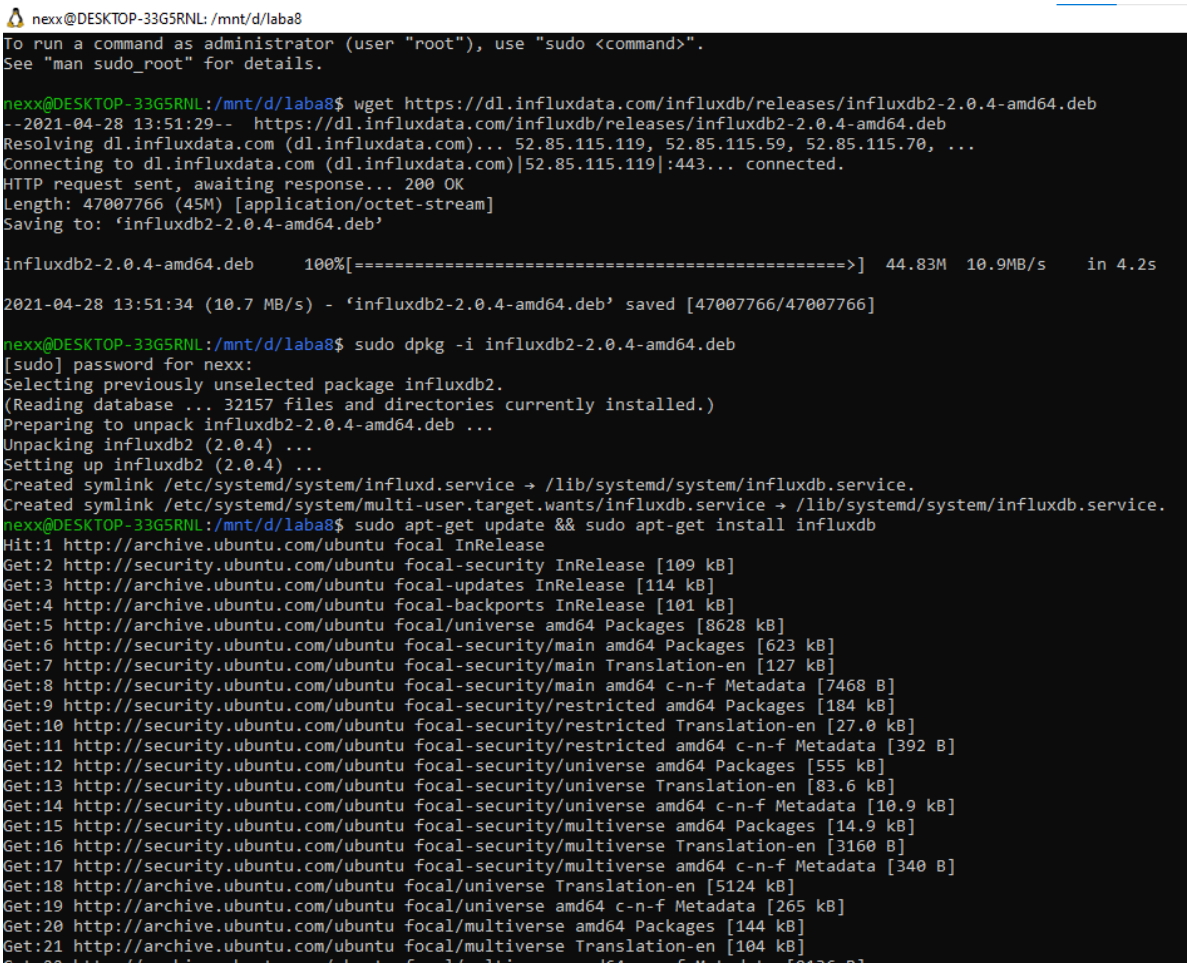
4.10. В результате, после запуска теста, получаем данные в jMeter и Grafana.


4.11. Установка в ОС Linux осуществляется аналогично.

В терминале Linux выполнить следующие команды для осуществления установки:




Запуск InfluxDB:

```
nexx@DESKTOP-33G5RNL:/mnt/d/laba8$ sudo service influxdb start
 * Starting database influxd
nexx@DESKTOP-33G5RNL:/mnt/d/laba8$
```

Запуск Influx:

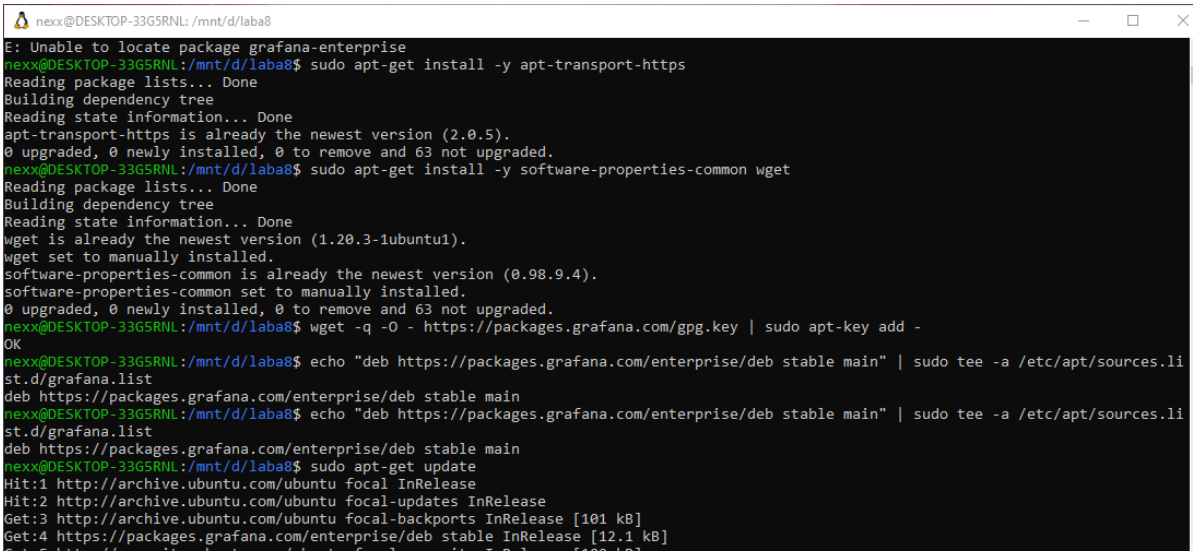
```
nexx@DESKTOP-33G5RNL:/mnt/d/laba8$ sudo apt install influxdb-client
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following NEW packages will be installed:
  influxdb-client
0 upgraded, 1 newly installed, 0 to remove and 63 not upgraded.
Need to get 1575 kB of archives.
After this operation, 5425 kB of additional disk space will be used.
```

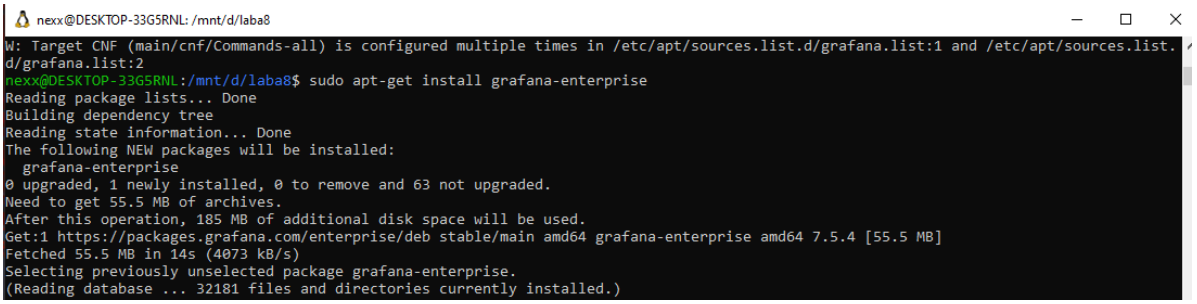
```
nexx@DESKTOP-33G5RNL:/mnt/d/laba8$ influx -precision rfc3339
Connected to http://localhost:8086 version 1.6.4
InfluxDB shell version: 1.6.4
> 
```

Создание БД и пользователя:

```
> CREATE DATABASE newbd
> SHOW DATABASES
name: databases
name
----
_internal
newbd
> CREATE USER nexx WITH PASSWORD '1111'
> SHOW USERS
user admin
---- -----
nexx false
> 
```

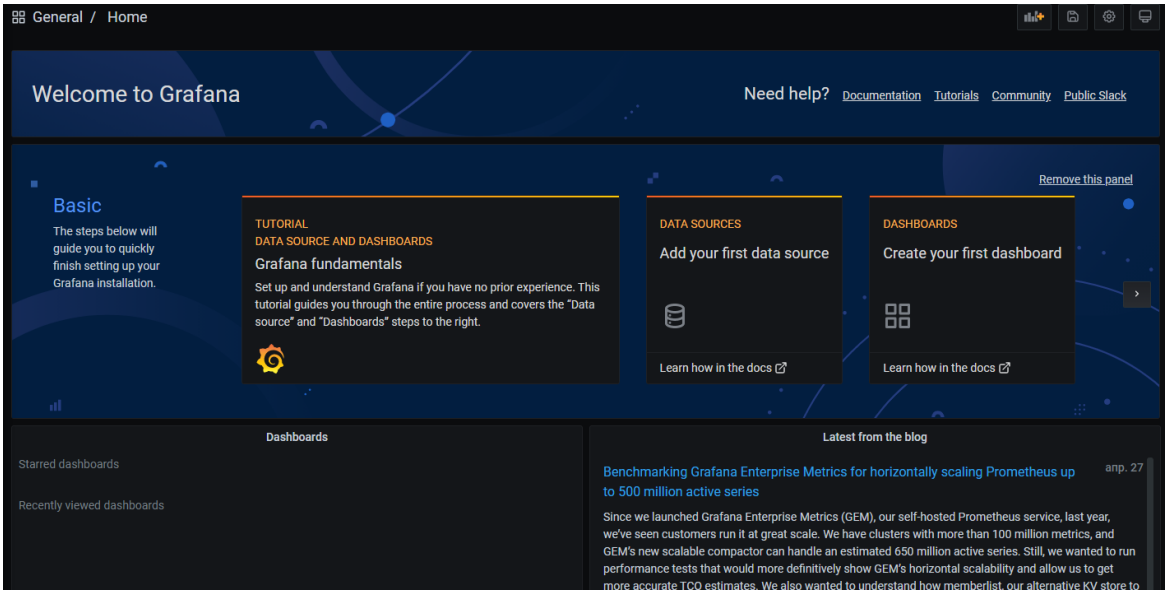
Установка Grafana:



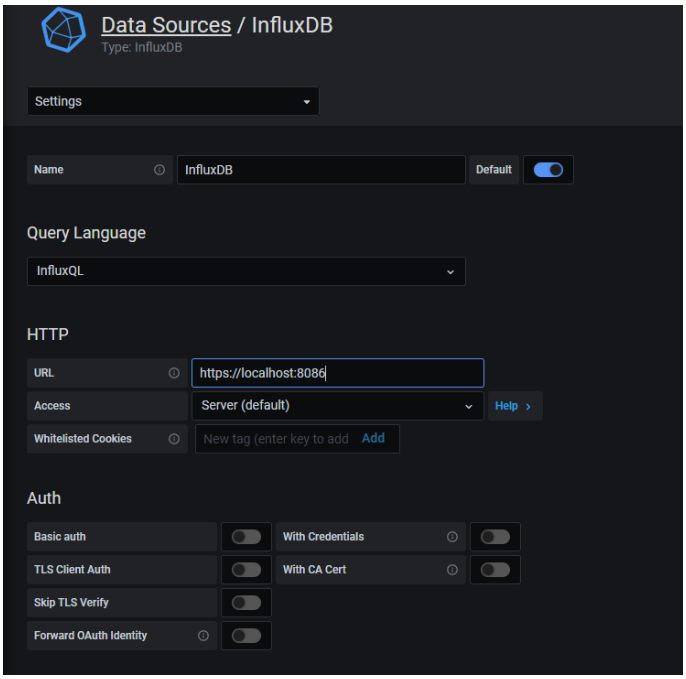



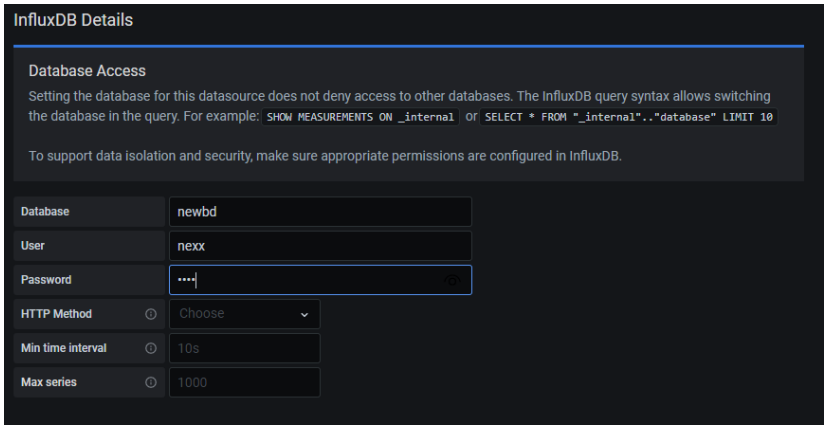
```
nexx@DESKTOP-33G5RNL:/mnt/d/laba8$ sudo service grafana-server start
 * Starting Grafana Server                                                    [ OK ]
nexx@DESKTOP-33G5RNL:/mnt/d/laba8$ sudo service grafana-server status
 * grafana is running
nexx@DESKTOP-33G5RNL:/mnt/d/laba8$ sudo systemctl enable grafana-server.service
Synchronizing state of grafana-server.service with SysV service script with /lib/systemd/systemd-sysv-install.
Executing: /lib/systemd/systemd-sysv-install enable grafana-server
Created symlink /etc/systemd/system/multi-user.target.wants/grafana-server.service → /lib/systemd/system/grafana-server.service.
nexx@DESKTOP-33G5RNL:/mnt/d/laba8$
```

Вводим адресную строку: http://localhost:3000/login:

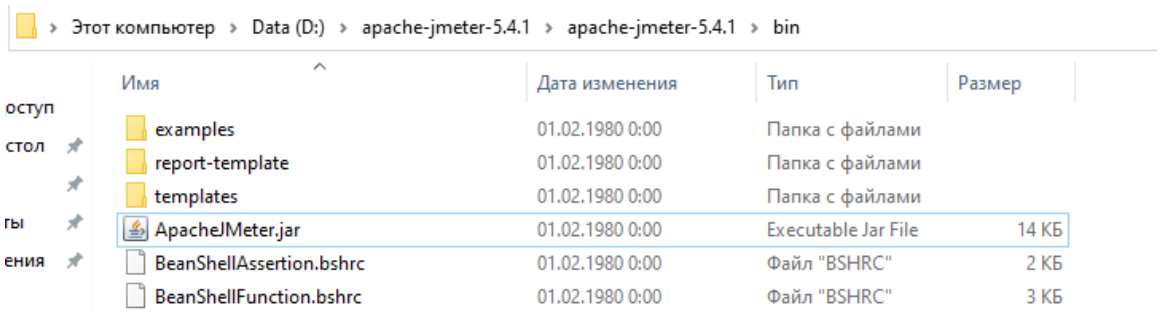



Добавление ранее созданной БД:





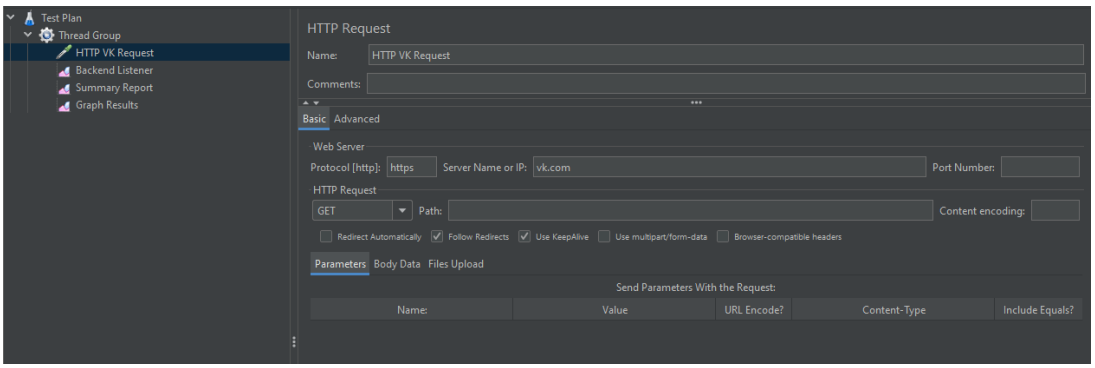

Установка jMeter. В скаченном архиве нужно зайти в папку bin и запустить файл ApacheJMeter.jar:



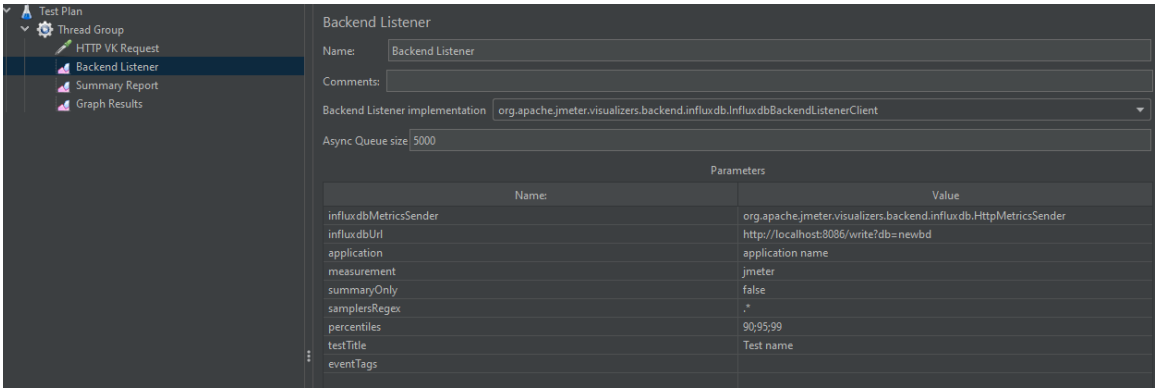

Создаём базовую структуру. Добавим в Thread Group по очереди элементы просмотра:

- Thread Group — Add — Sampler — HTTP Request.
- Thread Group — Add — Listener — Backend Listener, Summary Report и Graph Results.

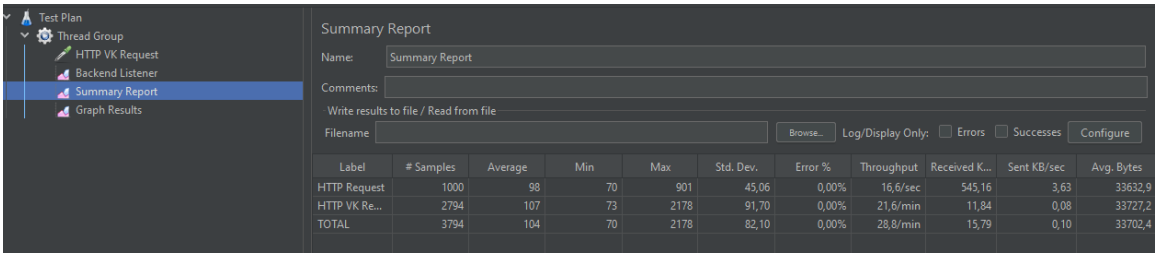
HTTP Request содержит запрос на сайт (в примере, vk.com):



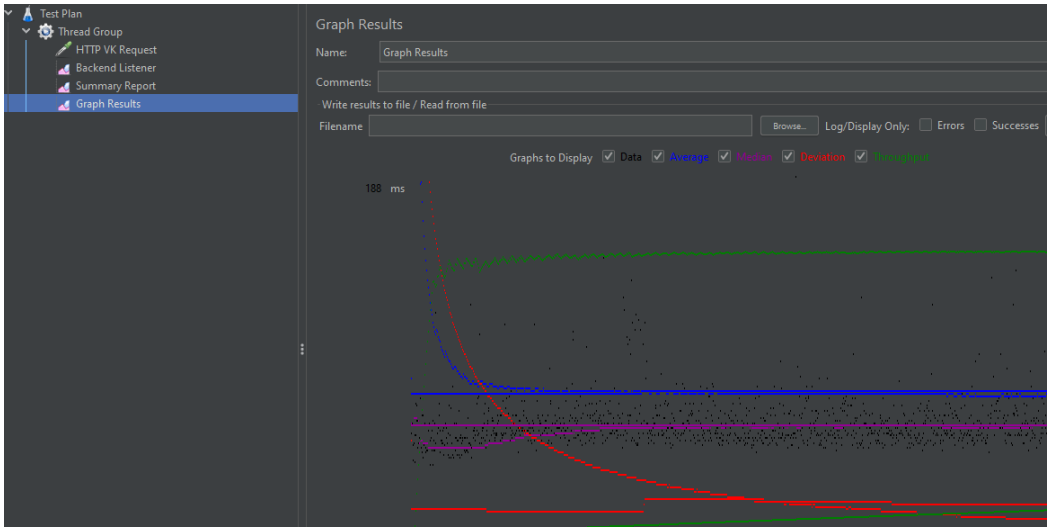
Backend Listener нужен для соединения с созданной БД.



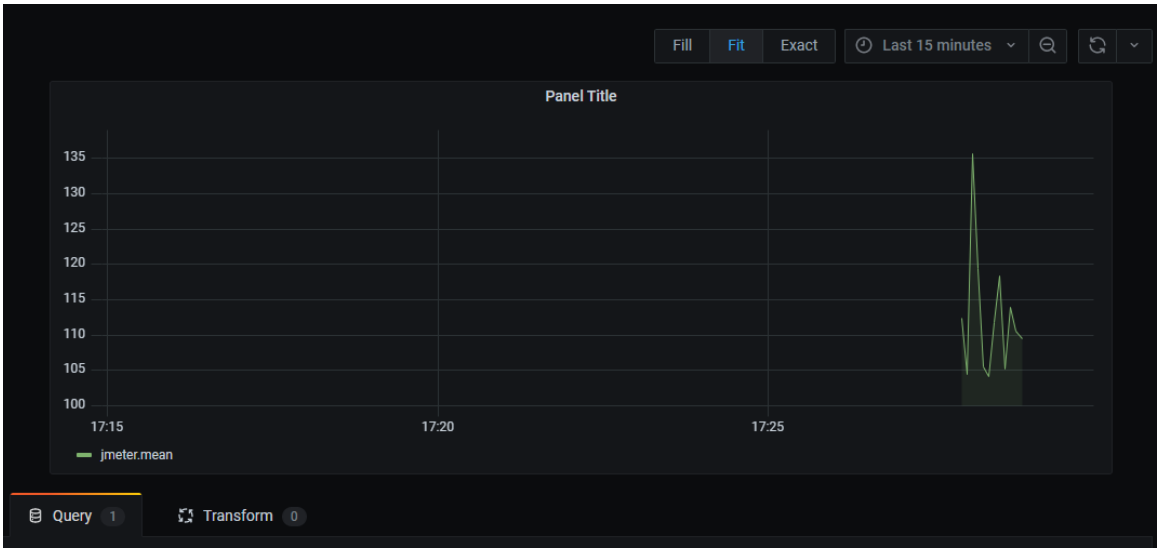

Summary Report.




Graph Results отображает результат в виде графика.




Показатели Dashboard Grafana:




## 5. Задания к лабораторной работе

5.1. Развернуть и настроить среду для проведения тестирования.

5.2. Протестировать доступность сайта (сайт выбрать произвольно).

## 6. Отчетность по лабораторной работе

6.1. В качестве приложения к отчету представить образы экранов (скриншоты) результатов тестирования.

# ЛАБОРАТОРНАЯ РАБОТА № 9. ОБОЛОЧКИ ЭКСПЕРТНЫХ СИСТЕМ

## 1. Цель работы

1.1. Изучение систем представления и обработки знаний.

## 2. Содержание работы

2.1. Установить систему CLIPS[24] (C Language Integrated Production System)

2.2. Воспроизвести примеры управления фактами.

2.3. Реализовать поставленные задания.

## 3. Общие сведения

**Оболочки экспертных систем** - программный продукт, обладающий средствами представления знаний. Задача пользователя заключается не в непосредственном программировании, а в формализации и вводе знаний с использованием предоставленных оболочкой возможностей.

CLIPS[25] (C Language Integrated Production System) является инструментальным средством разработки экспертных систем, база знаний которых представляет собой совокупность правил продукции, содержит встроенный редактор и средства отладки. CLIPS является оболочкой ЭС. CLIPS[26] является свободно распространяемым программным продуктом с доступными исходными кодами.

Основные элементы CLIPS: База знаний представлена в виде фактов, машина логического вывода, осуществляющая прямой вывод (forward chaining).

Основные правила:

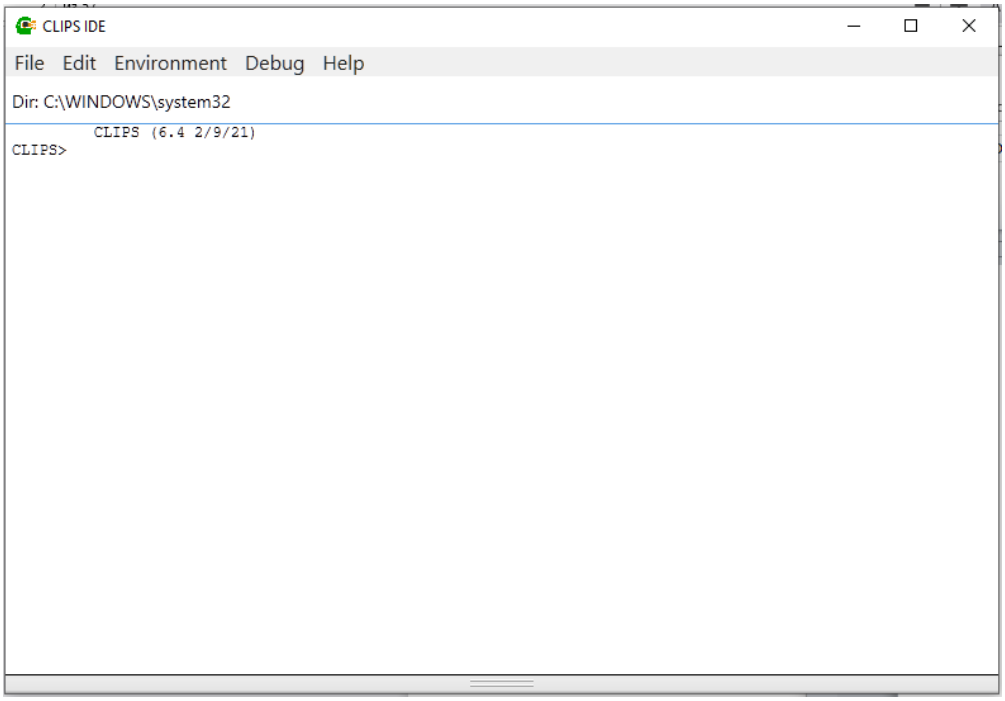
- Case-sensitive.

[24] Страница проекта в интернете - http://clipsrules.sourceforge.net/

[25] Страница проекта в интернете - http://clipsrules.sourceforge.net/

[26] Рекомендуется использовать версию CLIPS v. 6.4. В отличие от предыдущих эта версия поддерживает кириллические символы в текстовых строках, именах переменных, метках правил, описании фактов.

- Восемь типов данных – вещественное, целое, символ, строка, внешний адрес[27], адрес факта[28], имя экземпляра[29], адрес экземпляра.
- Команды - в круглых скобках, иначе это интерпретируется как символ.
- Строки - в двойных кавычках.
- Комментарий - ;.




Основным методом взаимодействия с CLIPS является использование командной строку CLIPS.

CLIPS был разработан таким образом, чтобы имелась возможность встраивания CLIPS в другие системы и вызов внешних функций в CLIPS.

[27] Внешний адрес - это адрес внешней структуры данных, возвращаемой функцией, интегрированной в CLIPS. Этот тип данных может быть создан только при вызове функции.

[28] На факты ссылаются по индексу или адресу. Печатный формат адреса факта следующий: <Fact-XXX>, где XXX - индекс факта.

[29] Определяемый пользователем класс может быть создан с помощью конструкция defclass. Имя экземпляра

формируется путем заключения символа квадратные скобки - [foo]. Адрес экземпляра может быть получен только путем связывания возвращаемого значения функции с именем экземпляра.

Вызовы функций в CLIPS имеют префиксную форму: аргументы функции всегда следуют после имени функции. Вызов функции начинается с открывающейся скобки, за которой следует имя функции, затем идут аргументы, каждый из которых отделен, по крайней мере, одним пробелом. Аргументами функций могут быть переменные примитивных типов, константы или вызовы других функций.

Примеры:

```
(printout t "Привет мир" crlf)
(+ 3 (* 8 9) 4)
(* 8 (+ 3 (* 2 3 4) 9) (* 3 4))
(max 2.4 -3)
(bind ?a 5) <= связать переменную со значением
(+ ?a ?a 3)
```

Среди **математических функций** следует выделить следующие группы[30]:

- Стандартные: +, -, *, /, max, min, div (целочисленное деление), abs (абсолютное значение), float (преобразование в тип float), integer (преобразование в тип integer )
- Расширенные: sqrt (извлечение корня), round (округление числа), mod (вычисление остатка от деления), Random n1 n2 (Выбор целого случайного числа из интервала [n1, n2])
- Тригонометрические: sin, sinh, cos, cosh, tan, tanh, acos, acosh, acot, acoth, acsc, acsch, asec, asech, asin, asinh, atan, atanh, cot, coth, csc, csch, sec, sech, deg-grad (преобразование из градусов в секторы), deg-rad (преобразование из градусов в радианы), grad-deg (преобразование из секторов в градусы), rad-deg (преобразование из радиан в градусы)
- Логарифмические: log, log10, exp, pi

[30] Существует несколько типов функций. Функции, определяемые пользователем, и функции, определяемые системой.

В CLIPS есть несколько определяющих конструкций: defmodule, defrule, deffacts, deftemplate, defglobal, deffunction, defclass, definstances, defmessage-handler, defgeneric и defmethod.

Все конструкции в CLIPS окружены круглыми скобками.

Конструкция открывается левой круглой скобкой и закрывается правой круглой скобкой.

Определение конструкции отличается от вызова функции в основном по действию[31]. В отличие от вызовов функций, конструкции никогда не имеют возвращаемого значения.

Базовые команды:

- (load <file-name>) – загрузка из файла. Функция, по умолчанию отображает процесс, TRUE/FALSE.
- (save <file-name>) – сохранение в файл текущих конструкций (environment).
- bsave/bload – аналогично, двоичный файл.
- (clear) – очистка окружения. Некоторые элементы не удаляются/не изменяются.
- (exit [<integer-expression>]) – выход.
- (reset) – Очищает список фактов, целей, объекты (instances), реинициализация, смена модуля на MAIN.
- (agenda) - текущие правила, готовые к активации

Для представления информации в CLIPS существует три основных формата:

- факты;
- объекты;
- глобальные переменные.

Факты помещаются в текущий список фактов.

Факты могут быть добавлены в список фактов (команда assert), удалены из списка фактов (команда retract), изменены (команда modify, только для неупорядоченных фактов) или продублированы

[31] Обычно вызов функции оставляет среду CLIPS без изменений (за некоторыми заметными исключениями, такими как сброс или очистка среды или открытие файла).

(команда duplicate, только для неупорядоченных фактов). Команды изменения и дублирования требуют указания факта, который указывается либо по индексу, либо по адресу.

**Упорядоченные факты** состоят из символа, за которым следует последовательность из нуля или более полей, разделенных пробелами и разграниченных открывающей скобкой справа и закрывающей скобкой справа[32].

```
(altitude is 10000 feet)
(assert (weather is fine))
(assert (totalcost (* 10 13)))
(assert (altitude is 10000 feet))
```

**Неупорядоченные (или deftemplate) факты** предоставляют пользователю возможность абстрагировать структуру факта путем присвоения имени каждому полю (слоту) факта. Порядок слотов не важен. Атрибут default может использоваться для задания значения по умолчанию.

Конструкция deffacts позволяет определить набор априорных или начальных знаний как набор фактов.

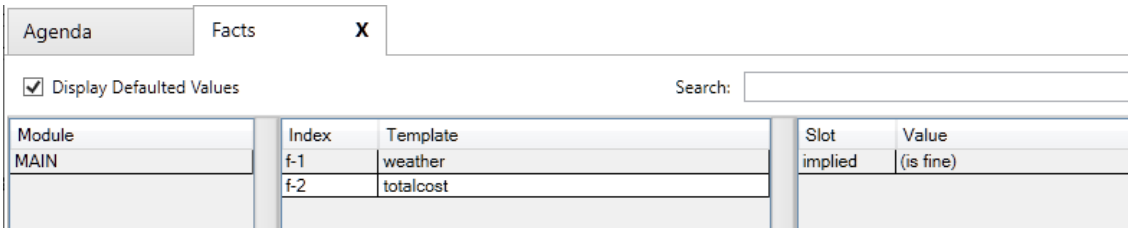
Когда среда CLIPS сбрасывается (с помощью команды reset), каждый факт, указанный в конструкции deffacts в базе знаний CLIPS добавляется в список фактов.

Конструкция defglobal позволяет определять переменные, которые являются глобальными по области применения во всей среде CLIPS.

```
(deftemplate client (slot name) (slot id))[33]
(assert (client (name "Joe Brown") (id X9345A)))
(deftemplate class …)
(assert (class (teacher "Martha Jones") (Room "37A")))
(modify 1 (id "123"))
```

(facts) – используется для просмотра фактов.

[32] Поля в неупорядоченном факте могут быть любого из примитивных типов данных.
[33] Оптимально выгрузить в файл




Аргументом retract может являться либо переменная, связанная с адресом факта, либо индексом факта без префикса (номер), либо выражением, вычисляющим этот индекс, либо * - все.

```
(retract 0 (+ 0 2) (+ 0 2 2))
```
[34]

С помощью конструкции deffacts можно определить список фактов, которые автоматически утверждаются каждый раз, когда выполняется команда сброса – (reset).

```
(deffacts startup "Refrigerator Status"
(refrigerator light on)
(refrigerator door open)
(refrigerator temp 50))
(reset)
```

Правило (defrule) - это набор условий и действий, которые должны быть предприняты при выполнении этих условий.

```
(defrule <rule-name> [<comment>] [<declaration>] ; Rule
Properties
<conditional-element>* ; Left-Hand Side (LHS)
=>
<action>*) ; Right-Hand Side (RHS)
```

По умолчанию условия объединяются через and.

Запустить проверку правил можно командой (run).

[34] Удаление из системы фактов с номерами 0, 2 и 4.

Среди **логических функций** (возвращающих значения true или false) следует выделить следующие группы:

- функции булевой логики: and, or, not
- функции сравнения чисел: =, ≠, ≥, ≤
- предикативные функции для проверки принадлежности проверяемому типу: integerp, floatp, stringp, symbolp, pointerp (относится ли аргумент к xternal-address ), numberp (относится ли аргумент к integer или float ), lexemerp (относится ли аргумент к string или symbol ), evenp (проверка целого на четность), oddp (проверка целого на нечетность), multifildp (является ли аргумент составным полем).
- функции сравнения по типу и по значению: eq, neq

## 4. Примеры работы с фактами

4.1. Ситуация 1 (упорядоченные факты):
Определяем начальное состояние холодильника:

```
(deffacts startup "Состояние холодильника"
(refrigerator Электричество Подключено)
(refrigerator Дверь Открыта)
(refrigerator Температура 50))
```

Сбрасываем систему к начальному состоянию:

```
(reset)
```

Добавляем правило:

```
(defrule Работает "Если открыта дверь, то еда испорчена"
(refrigerator Электричество Подключено)
(refrigerator Дверь Открыта)
=>
(assert (refrigerator Еда Испорчена)))
```

Проверяем состояние фактов:

```
(facts)
f-1     (refrigerator Электричество Подключено)
f-2     (refrigerator Дверь Открыта)
f-3     (refrigerator Температура 50)
For a total of 3 facts.
```

Запускаем обработку правил:

```
(run)
```

Проверяем факты:

```
(facts)
f-1     (refrigerator Электричество Подключено)
f-2     (refrigerator Дверь Открыта)
f-3     (refrigerator Температура 50)
f-4     (refrigerator Еда Испорчена)
For a total of 4 facts.
```

Сбрасываем систему к начальному состоянию:

```
(reset)
```

Фактов снова 3.

4.2. Ситуация 1 (упорядоченные факты):

Изменяем состояние холодильника (т.к. факты упорядоченные, то удаляем факт с дверью (по индексу 2) и добавляем новый – дверь закрыта):

```
(retract 2)
(assert (refrigerator Дверь Закрыта))
```

Запускаем обработку правил:

```
(run)
```

Проверяем факты:

```
(facts)
f-1     (refrigerator Электричество Подключено)
f-3     (refrigerator Температура 50)
f-4     (refrigerator Дверь Закрыта)
For a total of 3 facts.
```

Новый факт не был добавлен.

4.3. Аналогичный пример для неупорядоченных фактов:

Добавляем шаблон факта:

```
(deftemplate refrigerator (slot Электричество (default
Подключено)) (slot Дверь) (slot Температура))
```

Проверяем факты:

```
(facts)
```

Фактов нет.

Добавляем правило (Еда испорчена. Можно также добавить правило - когда еда нормальная):

```
(defrule Работает "Если открыта дверь, то еда испорчена"
(refrigerator (Электричество Подключено))
(refrigerator (Дверь Открыта))
=>
(assert (Еда Испорчена)))
```

Проверяем правило:

```
(run)
```

Вывод пустой.

Добавляем факт про холодильник:

```
(assert (refrigerator (Дверь Закрыта) (Температура 25)))
```

Проверяем факты:

```
(facts)
f-1 (refrigerator (Электричество Подключено) (Дверь Закры-
та) (Температура 25))
For a total of 1 fact.
```

Проверяем правило:

```
(run)
```

Вывод пустой.
Проверяем факты:

```
(facts)
f-1 (refrigerator (Электричество Подключено) (Дверь Закры-
та) (Температура 25))
For a total of 1 fact.
```

Изменяем состояние холодильника (1 факт):

```
(modify 1 (Дверь Открыта))
```

Проверяем факты:

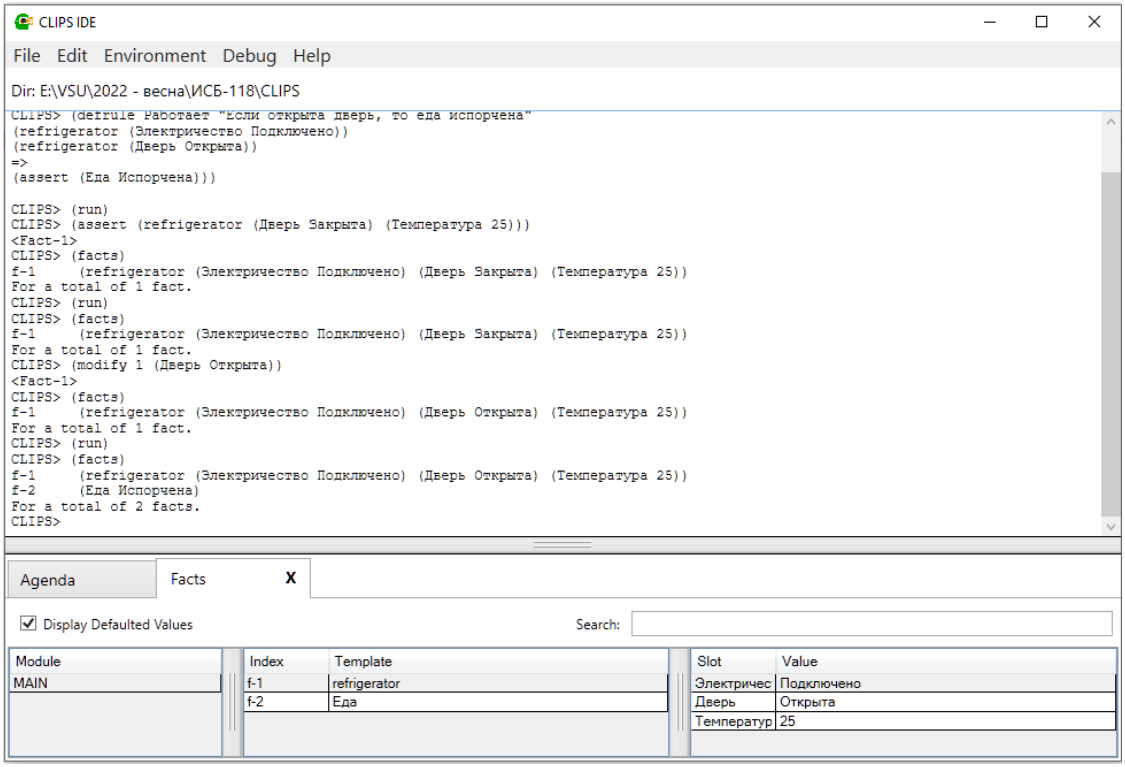
```
(facts)
f-1     (refrigerator (Электричество Подключено) (Дверь
Открыта) (Температура 25))
For a total of 1 fact.
```

Проверяем правило:

```
(run)
```

Проверяем факты:

```
(facts)
f-1 (refrigerator (Электричество Подключено) (Дверь Откры-
та) (Температура 25))
f-2 (Еда Испорчена)
For a total of 2 facts.
```




## 5. Использование правил для вывода результатов

Реализуется через вызов правила (пример варианта реализации правила подобного типа):

```
(defrule findstate
(refrigerator (Электричество ?x))
(refrigerator (Дверь ?y))
(Еда ?z)
=>
(printout t ?x " : " ?y " : " ?z crlf))
```

Запускаем (если нет второго правила, когда еда нормальная, то когда правило не сработает, то вывод будет пустой):

```
(run)
Подключено : Открыта : Испорчена
```

## 6. Задания к лабораторной работе

6.1. В режиме командной строки вычислить значения выражений:

- 9/5*8
- $(4^2 -5)*(3 + 4)$
- $4^2 + 28/(5+2)$
- $\max (2^3,3^2, 2^5)$

## 7. Отчетность по лабораторной работе

7.1. В качестве приложения к отчету представить образы экранов (скриншоты) или log файл с результатами выполнения команд вычисления выражений.

# ЛАБОРАТОРНАЯ РАБОТА № 10. ОБОЛОЧКИ ЭКСПЕРТНЫХ СИСТЕМ (ФУНКЦИИ)

## 1. Цель работы

1.1. Изучение методов использования функции в процессе представления и обработки знаний в экспертных системах.

## 2. Содержание работы

2.1. Воспроизвести примеры управления фактами.
2.2. Реализовать поставленные задания.

## 3. Общие сведения

В языке CLIPS функции конструируются примерно так же, как в языке LISP.

Существенное отличие состоит в том, что переменные должны иметь префикс ‘?’. Например, определение функции вычисления значения гипотенузы по теореме Пифагора:

```
(deffunction hypotenuse (?a ?b) (sqrt (+ (* ?a ?a) (* ?b ?b)) ))
```

Формат определения функции в CLIPS следующий:

```
(deffunction <имя функции (<аргумент> ... <аргумент>) <выражение> <выражение> )
```

Функция возвращает результат последнего выражения в списке. Иногда выполнение функции имеет побочные эффекты, как в приведенном ниже примере.

```
(deffunction init (?day)
(reset)
(assert (today is ?day)) )
```

В результате после запуска функции на выполнение командой

```
CLIPS> (init Sunday)
```

будет выполнена команда reset и, следовательно, очищена база фактов, а затем в нее будет включен новый факт (today is Sunday).

Базовые функции языка CLIPS представлены в таблице.

| Функция | Назначение |
|---|---|
| (bind ) | Связывает переменную с указанным значением |
| (if then + [else +]) | Вычисляет выражение, указанное в then, если TRUE, иначе вычисляет выражение, указанное в else |
| (while [do] *) | Вычисляет * до тех пор, пока не примет значение FALSE |
| (loop-for-count <диапазон> do <действие>*) | Выполняет действие некоторое количество раз согласно заданному диапазону |

**Строковые функции** языка CLIPS представлены в таблице.

| Функция | Назначение |
|---|---|
| (lowcase ) | Возвращает аргумент, приведенный к нижнему регистру |
| (str-cat *) | Возвращает все аргументы, объединенные в строку |
| (str-compare ) | Возвращает ноль, если оба аргумента равны, иначе отличное от нуля значение |
| (str-index ) | Возвращает позицию первого аргумента во втором аргументе, если первый аргумент подстрока второго, иначе FALSE |
| (str-length ) | Возвращает длину строки |
| (sub-string ) | Возвращает подстроку из string-expression^ начинающуюся в позиции и заканчивающуюся в позиции |
| (sym-cat *) | Возвращает все аргументы, соединенные в идентификатор |
| (upcase ) | Возвращает аргумент, приведенный к верхнему ре- |

| | |
|---|---|
| | гистру |

**Предикатные функции** языка CLIPS представлены в таблице.

| Функция | Назначение |
|---|---|
| (and +) | TRUE, если каждый из аргументов равен TRUE, иначе FALSE |
| (eq +) | TRUE, если первый аргумент равен по типу и по значению всем остальным аргументам, иначе FALSE |
| (evenp ) | TRUE, если четное целое, иначе FALSE |
| (floatp ) | TRUE, если типа float, иначе FALSE |
| (integerp ) | TRUE, если типа integer, иначе FALSE |
| (lexemep ) | TRUE, если типа string или symbol, иначе FALSE |
| (multifieldp ) | TRUE, если - multifield, иначе FALSE |
| (neq +) | TRUE, если его первый аргумент не равен по типу или по значению всем остальным, иначе FALSE |
| (not ) | TRUE, если значение аргумента FALSE, иначе TRUE. |
| (numberp ) | TRUE, если типа float или integer, иначе FALSE |
| (oddp ) | TRUE, если нечетное целое, иначе FALSE |
| (or +) | TRUE, если хотя бы один из аргументов равен TRUE, иначе FALSE |
| (stringp ) | TRUE, если является строкой, иначе FALSE |
| (symbolp ) | TRUE, если - идентификатор, иначе FALSE |
| (= +) | TRUE, если его первый аргумент равен по числовому значению всем остальным аргументам, иначе FALSE |
| (<> <numeric-expressions)< p=""> </numeric- | TRUE, если его первый аргумент не равен по числовому значению всем остальным аргументам, иначе FALSE |

| | |
|---|---|
| expressions)<> | |
| (> ) | TRUE, если для всех аргументов, аргумент п - 1 больше, чем аргумент п, иначе FALSE |
| (>= +) | TRUE, если для всех аргументов, аргумент п - 1 больше или равен аргументу п, иначе FALSE |
| (< +) | TRUE, если для всех аргументов, аргумент п - 1 меньше, чем аргумент п, иначе FALSE |
| (<= +) | TRUE, если для всех аргументов, аргумент п - 1 меньше или равен аргументу п, иначе FALSE |

## 4. Примеры создания пользовательских процедурных и рекурсивных функций

4.1.Определение функции

```
CLIPS> (deffunction hypotenuse-length (?a ?b)
(** (+ (* ?a ?a) (* ?b ?b)) 0.5))
CLIPS> (hypotenuse-length 10 15)
18.0277563773199
```

4.2. Использование функции if-then-else

Пусть требуется вычислить функцию func1 в зависимости от условия:

$a^2$-$b^2$, если a>b

$a^2$+$b^2$, если a<=b.

Определение функции func1:

```
(deffunction func1 (?a ?b)
(if (or (not (numberp ?a)) (not (numberp ?b))) then
(printout t "Ошибка типа данных a или b" crlf)
else
(if(> ?a ?b)
then (- (* ?a ?a) (* ?b ?b))
else (+ (* ?a ?a) (* ?b ?b))
)))
```

Вызов функции:

```
CLIPS> (func1 10 a12)
Ошибка типа данных a или b
CLIPS> (func1 10 12)
244
```

### 4.3. Пример функции while

Пусть требуется написать функция вычисления факториала числа:

```
CLIPS> (deffunction factorial(?limit)
(bind ?i 1)
(bind ?f 1)
(while (<= ?i ?limit) do
(bind ?f (* ?f ?i))
(bind ?i (+ ?i 1))
)
(printout t "f(" ?limit ") = " ?f crlf)
)
```

Вызов функции:

```
CLIPS> (factorial 3)
f(3) = 6
CLIPS> (factorial 5)
f(5) = 120
```

### 4.4. Пример использования функции loop-for-count

Вычисления факториала чисел в выбранном диапазоне.

```
CLIPS> (deffunction tablef(?a ?b)
(printout t "Расчет начат" crlf)
(bind ?i 0)
(loop-for-count (?i ?a ?b) do
    (bind ?y (factorial ?i))
    (printout t ?y)
)
(printout t "Расчет завершен" crlf)
)
```

Вызов функции:

```
CLIPS> (tablef 1 5)
Расчет начат
f(1) = 1
f(2) = 2
f(3) = 6
f(4) = 24
f(5) = 120
Расчет завершен
```

### 4.5. Рекурсивная функция

Пусть требуется написать функция вычисления факториала числа:

```
(deffunction factorialr(?a)
(if (or (not (integerp ?a )) (< ?a 0))
then
(printout t "Ошибка!" crlf)
else
(if (= ?a 1) then
1
else (* ?a (factorialr (- ?a 1))))
))
```

Вызов функции:

```
CLIPS> (factorialr 5)
120
CLIPS> (factorialr 3)
6
```

## 5. Работа с функциями

5.1. Функция может возвращать значение. Таким образом, она может использоваться в выржениях. Пример реализации функции, возвращающей значение:

```
(deffunction hypotenuse-length (?a ?b)
(bind ?temp (+ (* ?a ?a) (* ?b ?b)))
(bind ?c (** ?temp 0.5))
```

```
(return ?c))
```

5.2. Передача параметров  в функцю может быть реализована в интерактивном режиме (организация диалога):

```
(deffunction check-input (?question $?values)
(printout t ?question " " ?values " ")
(bind ?answer (read))
(while (not (member$ ?answer ?values))
(printout t ?question " " ?values " ")
(bind ?answer (read)))
(return ?answer))
```

Вызов функции:

```
CLIPS> (check-input "Вы тут?" да нет)
Вы тут? (да нет) y
Вы тут? (да нет) n
Вы тут? (да нет) да
Да
```

5.3. Команды работы с конструкторами deffunction

| Команда | Описание | |
|---|---|---|
| ppdeffunction | Вывод определения конструктора deffunction в диалоговое окно CLIPS | (ppdeffunction <имя-функции>) |
| list-deffunctions | Вывод в диалоговое окно системы списка всех определенных конструкторов deffunction | (list-def functions) |
| undeffunction | Удаление определенного конструктора def function. В качестве параметра <имя-функции> возможно использование символа *. В этом случае команда попытается удалить все определенные пользователем функции. | (undeffunction <имя-функции>) |

5.4. Отслеживание функций

Для отслеживания работы конструкций deffunctions используется команда watch. Каждый раз, когда начинается или заканчивается выполнение конструкции, выводится информационное сообщение.

```
CLIPS> (watch deffucntions)
```

Обозначение DFN, в начале сообщения, указывает, что оно относится к конструкции deffunctions.

Символ >> показывает, что происходит переход в конструкцию, а символ << говорит о том, что осуществляется выход из конструкции. Следующий символ представляет собой имя конструкции deffunctions, в которую входит или из которой выходит программа.

Символ ED обозначает «глубину вложенности» - показывает, как вкладываются друг в друга вызовы конструкций. Отсчет этого значения начинается с нуля, а затем увеличивается (или уменьшается) на единицу после перехода в (или выхода) очередную конструкцию deffunctions.

Последним фрагментом информации, отображаемым в строке, являются фактические параметры, передаваемые в конструкцию deffunctions.

Примеры использования отслеживания при вызове функций (всех функций):

```
CLIPS> (watch deffunctions)
CLIPS> (check-input "Вы тут?" да нет)
DFN >> check-input ED:1 ("Вы тут?" да нет)
Вы тут? (да нет) y
Вы тут? (да нет) n
Вы тут? (да нет) r
Вы тут? (да нет) lf
Вы тут? (да нет) да
DFN << check-input ED:1 ("Вы тут?" да нет)
Да

CLIPS> (factorialr 5)
DFN >> factorialr ED:1 (5)
DFN >> factorialr ED:2 (4)
DFN >> factorialr ED:3 (3)
```

```
DFN >> factorialr ED:4 (2)
DFN >> factorialr ED:5 (1)
DFN << factorialr ED:5 (1)
DFN << factorialr ED:4 (2)
DFN << factorialr ED:3 (3)
DFN << factorialr ED:2 (4)
DFN << factorialr ED:1 (5)
120
```

Можно отслеживать только определенные функции, указав имя функции:

```
CLIPS> (watch deffucntions hypotenuse-length)
```

Отменить отслеживание можно командой unwatch:

```
CLIPS> (unwatch deffunctions)
```

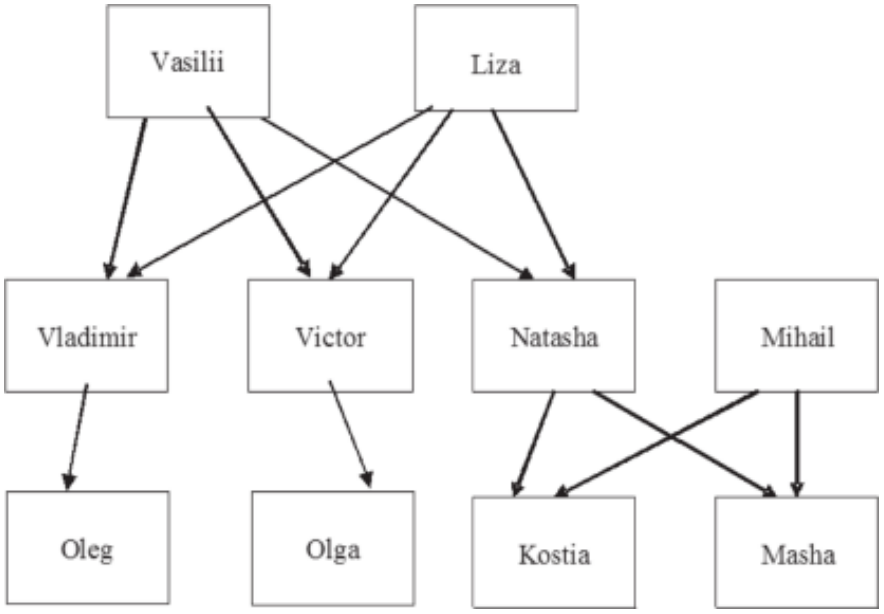
## 6. Задания к лабораторной работе

6.1. Для приведенной предметной области:

- определить, кто для кого является отцом (матерью), и, создать соответствующие факты;
- разработать функцию выбора пола родителя (в диалоговом режиме);
- разработать функцию (`parent`), которая будет сбрасывать факты и выбирать людей в соответствии с указанным полом родителя, создавая соответствующие факты.

6.2. Описание предметной области и базовых фактов: база знаний «Родственники» (генеалогическое дерево).

Родственные связи представлены на графе.




Шаблон для хранения информации о человеке:

```
(deftemplate person (slot name) (slot gender) (slot age))
```

Шаблон для хранения информации о родственной связи:

```
(deftemplate tparent (slot parentname) (slot childname))
```

Данные по людям:

```
(deffacts fperson
(person (name Vasilii) (gender male) (age 65))
(person (name Liza) (gender female) (age 62))
(person (name Vladimir) (gender male) (age 42))
(person (name Victor) (gender male) (age 40))
(person (name Natasha) (gender female) (age 37))
(person (name Oleg) (gender male) (age 18))
(person (name Olga) (gender female) (age 14))
(person (name Mihail) (gender male) (age 41))
(person (name Kostia) (gender male) (age 10))
(person (name Masha) (gender male) (age 12))
)
```

Данные по родственным связям:

```
(deffacts fparent
```

```
(tparent (parentname Vasilii) (childname Vladimir))
(tparent (parentname Vasilii) (childname Victor))
(tparent (parentname Vasilii) (childname Natasha))
(tparent (parentname Liza) (childname Vladimir))
(tparent (parentname Liza) (childname Victor))
(tparent (parentname Liza) (childname Natasha))
(tparent (parentname Vladimir) (childname Oleg))
(tparent (parentname Victor) (childname Olga))
(tparent (parentname Natasha) (childname Kostia))
(tparent (parentname Natasha) (childname Masha))
(tparent (parentname Mihail) (childname Kostia))
(tparent (parentname Mihail) (childname Masha))
)
```

Результат корректного заполнения исходных данных:

```
CLIPS> (reset)
CLIPS> (facts)
f-1     (person (name Vasilii) (gender male) (age 65))
f-2     (person (name Liza) (gender female) (age 62))
f-3     (person (name Vladimir) (gender male) (age 42))
f-4     (person (name Victor) (gender male) (age 40))
f-5     (person (name Natasha) (gender female) (age 37))
f-6     (person (name Oleg) (gender male) (age 18))
f-7     (person (name Olga) (gender female) (age 14))
f-8     (person (name Mihail) (gender male) (age 41))
f-9     (person (name Kostia) (gender male) (age 10))
f-10    (person (name Masha) (gender male) (age 12))
f-11    (tparent (parentname Vasilii) (childname Vladi-
mir))
f-12    (tparent (parentname Vasilii) (childname Victor))
f-13    (tparent (parentname Vasilii) (childname Natasha))
f-14    (tparent (parentname Liza) (childname Vladimir))
f-15    (tparent (parentname Liza) (childname Victor))
f-16    (tparent (parentname Liza) (childname Natasha))
f-17    (tparent (parentname Vladimir) (childname Oleg))
f-18    (tparent (parentname Victor) (childname Olga))
f-19    (tparent (parentname Natasha) (childname Kostia))
f-20    (tparent (parentname Natasha) (childname Masha))
f-21    (tparent (parentname Mihail) (childname Kostia))
f-22    (tparent (parentname Mihail) (childname Masha))
For a total of 22 facts.
```

## 7. Отчетность по лабораторной работе

7.1. В качестве приложения к отчету представить образы экранов (скриншоты) или log файл с результатами выполнения команд вычисления выражений.

## КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Какой порядок установки среды разработки (Apache, php, MySQL) в операционной системе MS Windows?
2. Какие параметры необходимо указать в скриптах php для подключения к СУБД?
3. Какие операции можно выполнить с использованием phpmyadmin?
4. Какая утилита используется для выполнения SQL запросов в командной строке?
5. Какие существуют типы данных атрибутов?
6. В чем заключается отличие модулей php: mysqli и pdo?
7. Какие команды php используются для выполнения SQL запросов?
8. Какими способами можно исключить возможность использования XSS-атак и SQL-инъекций при обработке данных в php?
9. Какой результат исполнения SQL запросов insert и delete?
10. Общий формат оператора SELECT.
11. Какими способами осуществляется фильтрация и сортировка данных в SQL запросах?
12. Концепция MVC. Какие требования накладываются на процесс разработки при использовании указанной концепции?
13. Какая модель доступа к данным используется в Django?
14. Доступ к каким СУБД может быть реализован в Django?
15. Какие команды SQL могут использоваться для соединения двух и более таблиц базы данных? Какие виды запросов соединения существуют?
16. Перечислите агрегатными фунции SQL.
17. Перечислите строковые фунции SQL.
18. Какие возможные виды SQL-инъекций существуют?
19. Каким образом можно проверить наличие возможности проведения SQL-инъекции?
20. Какие типы объектов конфигурации существуют в 1С:Предприятие 8?

21. Какие стандартные реквизиты существуют для объектов конфигурации 1С:Предприятие 8?
22. Назначение баз данных временных рядов? В каком виде хранятся данные с СУБД?
23. Какой общий порядок настройки jmeter для проведения нагрузочного тестирования?
24. Какой формат определения упорядоченных фактов в CLIPS?
25. Какой формат определения неупорядоченных фактов в CLIPS?

## БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1) MySQL.RU .:. Одобрено лучшими российскими программистами [Электронный ресурс] // URL: http://www.mysql.ru/docs/mysql-man-4.0-ru/tutorial.html (дата обращения 18.05.2023)

2) WampServer: как установить и настроить [Электронный ресурс] // URL: https://1cloud.ru/help/windows/ustanovka-nastroika-wampserver (дата обращения 18.05.2023)

3) Как установить LAMP-стек (Linux, Apache, MySqL, PHP) на сервере с Ubuntu // URL: https://1cloud.ru/help/linux/kak_ustanovit_lamp_stek_linux_apache_mysql_php_ubuntu (дата обращения 18.05.2023)

4) Жданов, С. А. Информационные системы : учебник / С. А. Жданов, М. Л. Соболева, А. С. Алфимова. — Москва : Прометей, 2015. — 302 с. — ISBN 978-5-9906-2644-7. — Текст : электронный // Лань : электронно-библиотечная система. — URL: https://e.lanbook.com/book/64787 (дата обращения: 18.05.2023).

5) Введение в СУБД MySQL : Курс лекций / — Москва : Интуит НОУ, 2016. — 228 с. — URL: https://book.ru/book/917666 (дата обращения: 18.05.2023). — Текст : электронный.

6) Php результат выполнения запроса [Электронный ресурс] // URL: https://cutvideo.ru/php-rezul-tat-vypolneniya-zaprosa/ (дата обращения 18.05.2023)

7) Жоау Э.Р., Матвеев Ю.Н., Михальцов Н.Г. ПРИМЕНЕНИЕ ШАБЛОНА ПРОЕКТИРОВАНИЯ MVC В РАЗРАБОТКЕ СОВРЕМЕННЫХ ВЕБ-СИСТЕМ // ПРОБЛЕМЫ ИНФОРМАТИКИ В ОБРАЗОВАНИИ, УПРАВЛЕНИИ, ЭКОНОМИКЕ И ТЕХНИКЕ. Сборник статей XIX Международной научно-технической конференции, Пензенский государственный университет, 2019. ISBN: 978-5-8356-1735-7

8) Оператор join – SQL JOIN - соединение таблиц базы данных [Электронный ресурс] // URL: https://djbook.ru/ (дата обращения 18.05.2023)

9) Уроки SQL и баз данных [Электронный ресурс] // URL: https://www.site-do.ru/db/db.php (дата обращения 18.05.2023)

*Учебное электронное издание*

ВОРОНИН Алексей Александрович

БАЗЫ ДАННЫХ И ЭКСПЕРТНЫЕ СИСТЕМЫ. ЧАСТЬ 1

Методические указания для студентов по выполнению лабораторных работ по дисциплине

*Издается в авторской редакции*

***Системные требования:*** Intel от 1,3 ГГц; Windows XP/7/8/10; Adobe Reader;
дисковод CD-ROM.

Институт информационных технологий и радиоэлектроники
Кафедра информатики и защиты информации
aleksey.voronin@vlsu.ru