А. И. Новиков

АЛГОРИТМИЗАЦИЯ И ПРОГРАММИРОВАНИЕ SQL, Apache, PHP, HTML

Часть 1

Учебно-методическое пособие

Санкт-Петербург
2025

Министерство науки и высшего образования Российской Федерации

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«Санкт-Петербургский государственный университет промышленных технологий и дизайна»
Высшая школа технологии и энергетики

А. И. Новиков

АЛГОРИТМИЗАЦИЯ И ПРОГРАММИРОВАНИЕ SQL, Apache, PHP, HTML

Часть 1

Учебно-методическое пособие

Утверждено Редакционно-издательским советом ВШТЭ СПбГУПТД

Санкт-Петербург
2025

УДК 004.432
ББК 32.973
Н 731

Рецензенты:
кандидат технических наук, доцент, заведующий кафедрой автоматизации технологических процессов и производств Высшей школы технологии и энергетики Санкт-Петербургского государственного университета промышленных технологий и дизайна
Д. А. Ковалёв;
кандидат технических наук, доцент, Санкт-Петербургского государственного технологического института (технического университета)
В. В. Куркина

Новиков, А. И.
Н 731 Алгоритмизация и программирование. SQL, Apache, PHP, HTML. Часть 1: учебно-методическое пособие / А. И. Новиков. — СПб.: ВШТЭ СПбГУПТД, 2025. — 107 с.

Учебно-методическое пособие соответствует программам и учебным планам дисциплины «Алгоритмизация и программирование» для студентов, обучающихся по направлению подготовки 09.03.03 «Прикладная информатика». В учебно-методическом пособии изложены основы работы с различными СУБД и синтаксис языка SQL, а также приведены примеры выполнения SQL-запросов, в том числе с использованием языков PHP, HTML и Delphi (Pascal). Кроме того, учебно-методическое пособие содержит разделы для самостоятельного углубленного изучения.

Пособие предназначено для подготовки бакалавров очной и заочной форм обучения. Отдельные разделы пособия могут быть полезны магистрантам и аспирантам.

УДК 004.432
ББК 32.973

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

Данное пособие состоит из двух частей. В первой части изложен базовый материал по работе с СУБД и языком SQL.

Система управления базами данных (СУБД, DBMS) – это комплекс программ, позволяющих создать базу данных (БД) и манипулировать данными (вставлять, обновлять, выбирать и удалять). В большинстве случаев речь идет о Реляционных базах данных – т.е. о таких БД, которые представляются в виде двумерных таблиц. В качестве примера другой часто встречающейся модели данных стоит упомянуть Иерархическую, т.е. такую, в которой данные представлены в виде «дерева» (как например структура каталогов на диске ПК).

Для «общения» с БД используется язык структурированных запросов SQL. Язык SQL описан как международный стандарт ISO/IEC 9075. SQL не является конкретным программным обеспечением от одного производителя, это стандарт, имеющий различные реализации от различных производителей. Поэтому в данном пособии рассматривается работа с двумя СУБД: Microsoft Access и MySQL.

На самом деле, SQL не предназначен для постоянного ручного ввода запросов к базе данных. Он предназначен для автоматизации процесса проведения запросов и используется в составе какого-либо из языков программирования. В данном случае для этого применяется язык PHP, который, в свою очередь, работает как препроцессор для языка HTML.

* Пособие содержит разделы для самостоятельного углубленного изучения, их названия отмечены звездочкой.

1. ЯЗЫК SQL

СУБД – **С**истема **у**правления **б**азами **д**анных (**DBMS**, **D**ata**b**ase **M**anagement **S**ystem) – комплекс программ, позволяющих создать базу данных (БД) и манипулировать данными (вставлять, обновлять, выбирать и удалять). Система обеспечивает безопасность, надежность хранения и целостность данных, а также предоставляет средства для администрирования БД.

В большинстве случаев, речь идет о **Реляционных базах данных** (и, соответственно, о реляционных СУБД, РСУБД) – т.е. о таких БД, которые представляются в виде двумерных таблиц. Название происходит от англ. **relation** (отношение, зависимость, связь). С реляционными БД тесно связаны понятие **нормализация** – т.е. процесса устранения недостатков структуры БД, приводящих к ее *избыточности*, а также теория (нормализации) для осуществления данного процесса.

В качестве примера другой часто встречающейся модели данных стоит упомянуть **Иерархическую**, т.е. такую, в которой данные представлены в виде «**дерева**» (как, например, структура каталогов на диске ПК).

SQL (**S**tructured **Q**uery **L**anguage, язык структурированных запросов) – декларативный язык программирования (т.е. описывающий требуемый результат, а не способ его получения), применяемый для управления данными в реляционной БД (управляемой в свою очередь соответствующей СУБД). Читается как «Эс-Ку-Эль» или «Эс-Кью-Эль». Язык SQL описан как международный стандарт **ISO/IEC 9075** (изначально был принят в качестве американского стандарта **ANSI**). Является языком 4-го поколения (**4GL**). *В то время как C, C++, C#, Java, Python, PHP, Pascal, Fortran и многие др. являются языками 3-го поколения (3GL).* Технически, как язык программирования, SQL является так называемым языком программирования «неполным по Тьюрингу».

Операторы SQL делятся на:

— операторы определения данных, например, **CREATE** и **DROP** – для создания и удаления таблицы (или другого объекта);

— операторы манипуляции данными, например, **SELECT** – для выбора данных, удовлетворяющих заданным условиям, или **INSERT**, **UPDATE** и **DELETE** – для добавления, изменения и удаления данных;

— и др.

Операторы языка SQL (и другие ключевые слова SQL) принято писать прописными (т.е. БОЛЬШИМИ) буквами.

!!! При выполнении студенческих работ в рамках данного курса, эта рекомендация является обязательной к применению**!**

В конце SQL-запроса ставится точка с запятой.

SQL не является конкретным программным обеспечением от одного производителя. Это стандарт, имеющий различные реализации от различных производителей, например, Microsoft **SQL Server**, или **MySQL**. В **Microsoft Access** также имеется ограниченная поддержка SQL.

1.1. Система управления базами данных Microsoft Access

Microsoft Access (или полностью **Microsoft Office Access**) – реляционная система управления базами данных (СУБД, РСУБД) корпорации **Microsoft**. Входит в состав пакета **Microsoft Office** и является проприетарным (т.е. платным) программным обеспечением. Благодаря встроенному языку **VBA** позволяет разрабатывать экранные формы и целые приложения для работы с базами данных. Язык **SQL** в Microsoft Access не соответствует стандартам **ANSI** или **ISO**.

Microsoft Access предназначен для работы с базами данных на локальном компьютере (один пользователь) или в качестве файл-серверной СУБД (небольшое число пользователей, работающих с базой данных одновременно) и не рассчитан на многопользовательский (клиент-серверный) режим работы.

У корпорации Microsoft есть и другая СУБД – это **Microsoft SQL Server**, которая соответствует стандартам **ISO** и **ANSI** и позволяет работать в многопользовательском режиме.

Стоит обратить внимание на особенности сохранения в **Microsoft Access**, которое сильно отличается от сохранения в других программах **Microsoft Office** (таких как **Word** или **Excel**). В **Access** вносимые изменения сохраняются постоянно без нажатия кнопки «Сохранить». Кроме того, имя базы данных и папка для ее сохранения задаются сразу в момент создания базы данных (рис. 1).

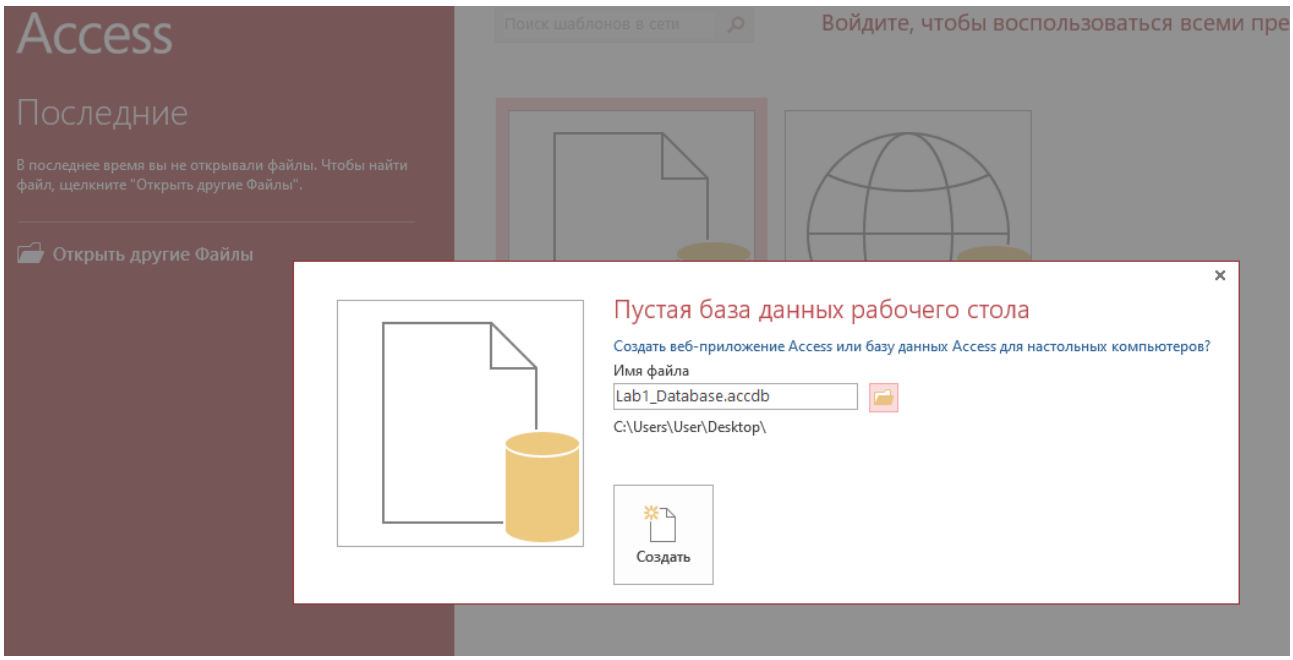

Рисунок 1 – Создание базы данных в Microsoft Access

!!! В процессе работы, даже если в базу не вносятся новые данные, размер файла **Access** постоянно увеличивается. Для сокращения объема памяти, занимаемого базой данных, необходимо периодически выполнять команду «**Сжать или восстановить базу данных**» (рис. 2) из меню «Работа с базами данных».

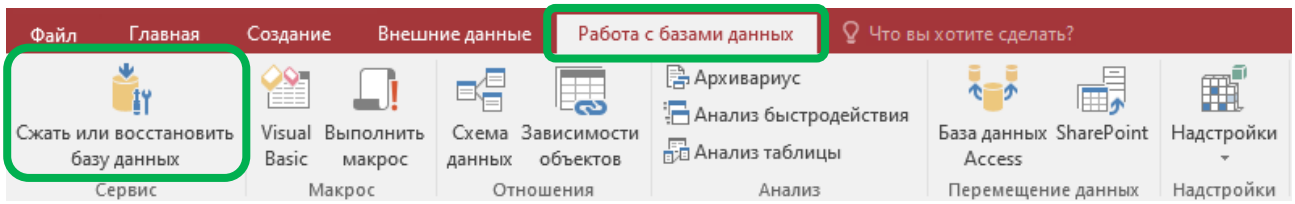

Рисунок 2 – Сжатие базы данных Microsoft Access

1.2. Создание таблиц в Access

Создание таблицы в **Access** происходит из меню «Создание» (рис. 3).
!!! В связи с тем, что предметом изучения является **SQL**, а не **Access**, здесь рассматривается только вариант создания простейших таблиц и не рассматривается работа с «Конструктором таблиц».

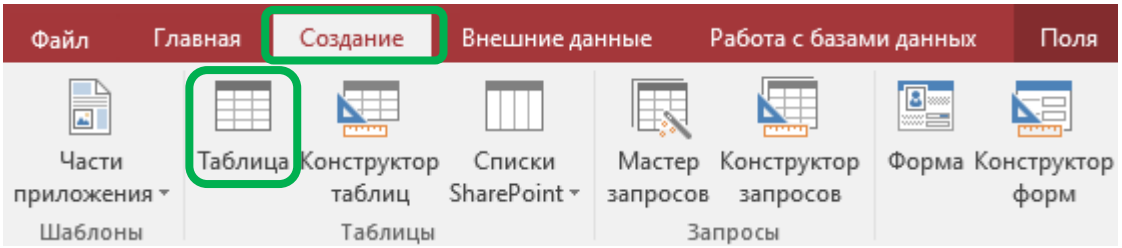

Рисунок 3 – Создание таблицы в Microsoft Access

После появления «пустой» таблицы в нее необходимо добавить требуемые столбцы (колонки). Это производится из выпадающего списка (рис. 4) в конце таблицы. В выпадающем списке требуется выбрать тип данных для добавляемого столбца (текст, число, логический и т.д.).

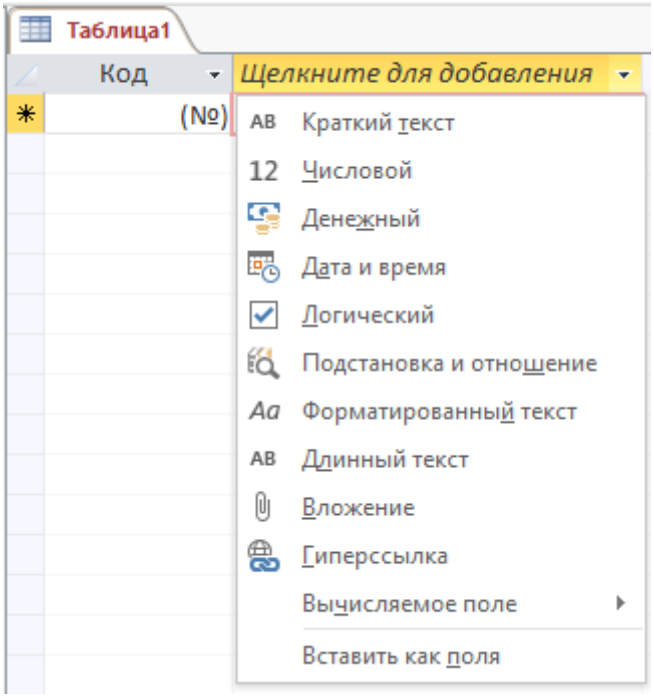

Рисунок 4 – Добавление столбцов в таблицу Microsoft Access

Создадим для примера таблицу (рис. 5), содержащую три текстовых столбца: «Фамилия», «Имя» и «Отчество». Также в таблице должен присутствовать числовой столбец «Код», который является первичным ключом (обычно создается автоматически вместе с созданием новой таблицы).

Созданную таблицу необходимо заполнить не менее чем 10-ю строками с фамилиями, именами и отчествами (ФИО) студентов. Для этого можно использовать ФИО ваших реальных одногруппников (если их меньше, то оставшиеся ФИО нужно придумать). В этой таблице фамилии располагаются не по алфавиту.

Студенты

Код	Фамилия	Имя	Отчество	Щелкните для добавления
1	Иванов	Иван	Иванович	
2	Петров	Петр	Петрович	
3	Сидоров	Сидор	Сидорович	
4	Николаев	Николай	Николаевич	
5	Александрова	Александра	Александровна	
6	Новиков	Александр	Игоревич	
(№)				

Рисунок 5 – Пример таблицы «Студенты»

Далее создадим таблицу «Журнал_по_АиП» (рис. 6), содержащую числовой столбец «Студент» (с кодом студента из прошлой таблицы), а также 5 логических столбцов («Лаб_1» .. «Лаб_5»), в которых будет отмечаться выполнение студентами лабораторных работ по дисциплине «СУБД». В данной таблице должно быть такое же количество строк, как и в предыдущей.

Журнал_по_АиП

Студент	Лаб_1	Лаб_2	Лаб_3	Лаб_4	Лаб_5
1	☐	☐	☐	☐	☐
2	☑	☑	☑	☑	☑
3	☑	☑	☑	☐	☐
4	☑	☐	☐	☐	☐
5	☑	☐	☐	☐	☐
6	☑	☑	☑	☑	☑
-1	☐	☐	☐	☐	☐

Рисунок 6 – Пример таблицы «Журнал_по_АиП»

Аналогично создадим таблицу «Журнал_по_СУБД» (рис. 7), но содержащую **4**, а не 5 лабораторных работ.

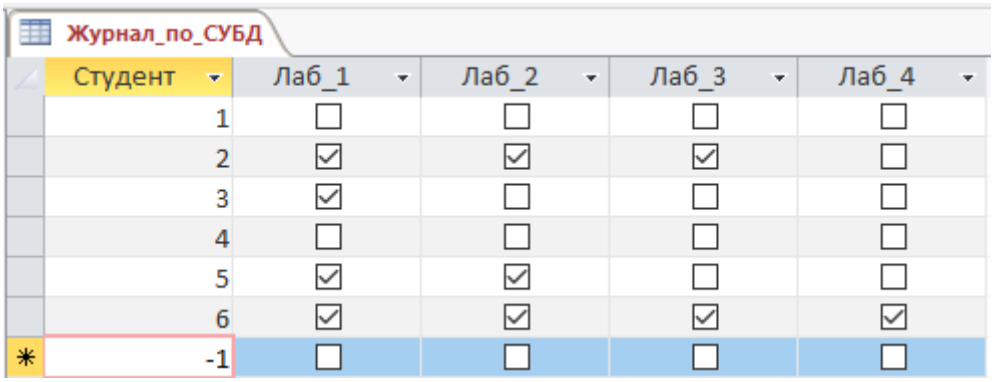

Рисунок 7 – Пример таблицы «Журнал_по_СУБД»

В итоге в **Microsoft Access**, мы получили 3 таблицы (рис. 8).

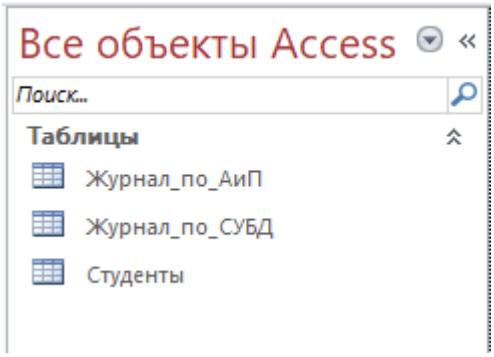

Рисунок 8 – Список таблиц в Microsoft Access

1.3. SQL-запросы в Access

Создание **SQL**-запросов в **Access** происходит через «**Конструктор запросов**» из меню «Создание» (рис. 9).

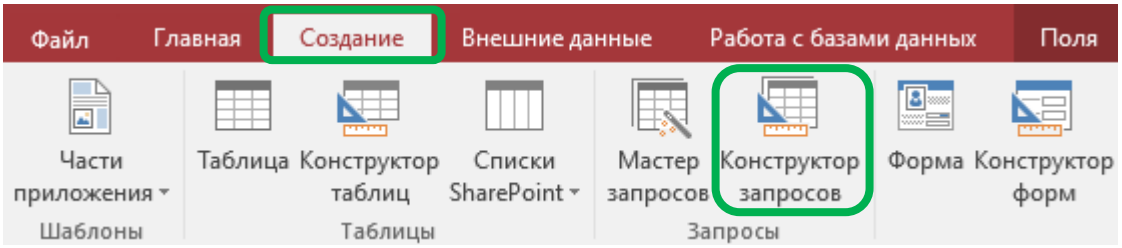

Рисунок 9 – Создание запроса в Microsoft Access

В появившемся далее окне необходимо нажать кнопку «Закрыть» (рис. 10).

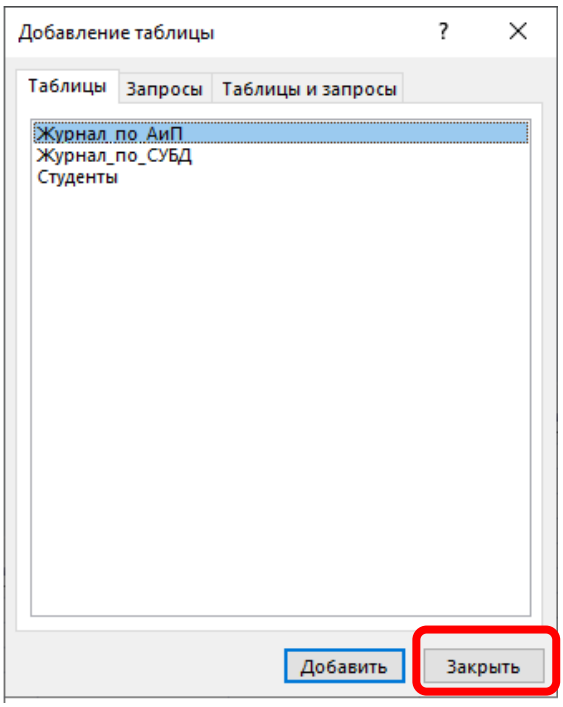

Рисунок 10 – Добавление таблиц к запросу

Далее перейдем в «**режим SQL**» (рис. 11).

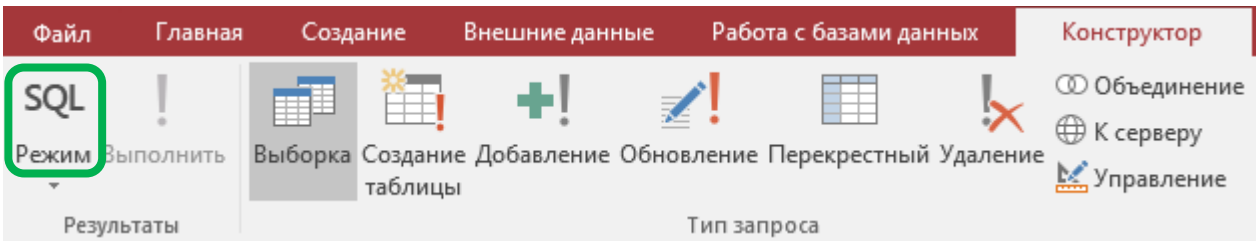

Рисунок 11 – Переход в режим SQL

Здесь введем **SQL**-запрос, представленный на рис. 12. Данный запрос позволяет нам вывести (рис. 13) содержание сразу двух таблиц. Этот запрос мы назовем «1) Журнал по АиП (с фамилиями)».

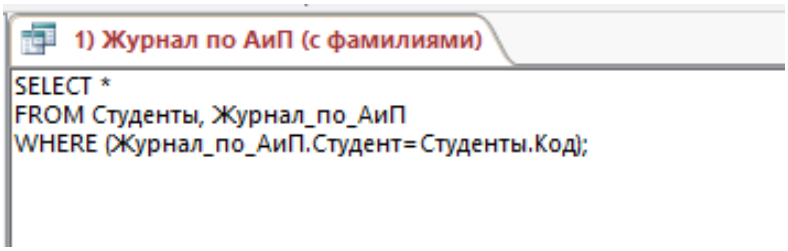

Рисунок 12 – Запрос «1) Журнал по АиП (с фамилиями)»

1) Журнал по АиП (с фамилиями)

Код	Фамилия	Имя	Отчество	Студент	Лаб_1	Лаб_2	Лаб_3	Лаб_4	Лаб_5
1	Иванов	Иван	Иванович	1	☐	☐	☐	☐	☐
2	Петров	Петр	Петрович	2	☑	☑	☑	☑	☑
3	Сидоров	Сидор	Сидорович	3	☑	☑	☑	☐	☐
4	Николаев	Николай	Николаевич	4	☑	☐	☐	☐	☐
5	Александрова	Александра	Александровна	5	☑	☐	☐	☐	☐
6	Новиков	Александр	Игоревич	6	☑	☑	☑	☑	☑

Рисунок 13 – Пример запроса «1) Журнал по АиП (с фамилиями)»

Недостатком выполненного ранее запроса является то, что были выведены все возможные колонки, в том числе повторяющиеся значения «Код» и «Студент». Создадим аналогичный запрос «2) Журнал по СУБД (с фамилиями)» (рис. 14), но без этих двух лишних колонок. Для этого в запросе вместо звездочки (*) необходимо перечислить имена всех отображаемых столбцов.

2) Журнал по СУБД (с фамилиями)

Фамилия	Имя	Отчество	Лаб_1	Лаб_2	Лаб_3	Лаб_4
Иванов	Иван	Иванович	☐	☐	☐	☐
Петров	Петр	Петрович	☑	☑	☑	☐
Сидоров	Сидор	Сидорович	☑	☐	☐	☐
Николаев	Николай	Николаевич	☐	☐	☐	☐
Александрова	Александра	Александровна	☑	☑	☐	☐
Новиков	Александр	Игоревич	☑	☑	☑	☑

Рисунок 14 – Пример запроса «2) Журнал по СУБД (с фамилиями)»

На данном этапе в **Microsoft Access**, мы получили 3 таблицы и 2 запроса (рис. 15).

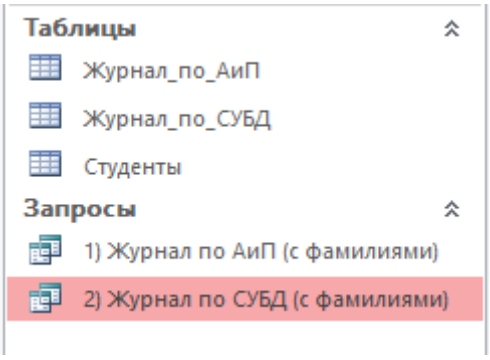

Рисунок 15 – Список объектов в Microsoft Access

1.4. РАБОТА № 1
«SQL-запросы в Access»

Необходимо выполнить работу в **Microsoft Access**, состоящую из трех таблиц и семи **SQL**-запросов (рис. 16). Создание таблиц описано в разделе **1.2**, создание первых двух запросов рассматривалось в разделе **1.3**. Подробнее синтаксис языка **SQL** рассмотрен в разделе **1.5**.

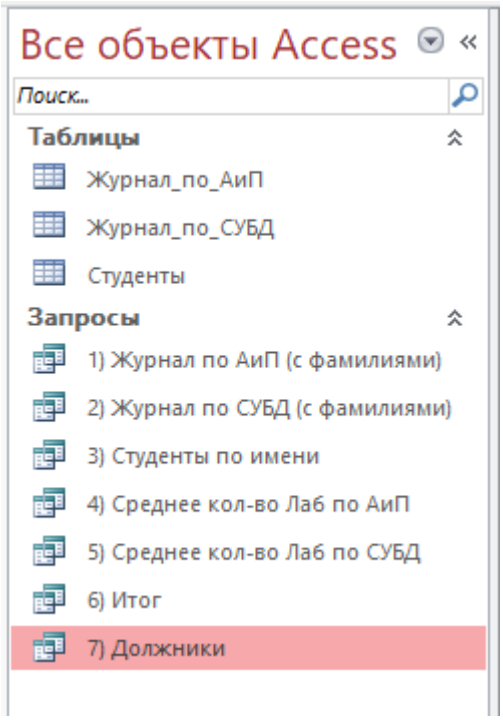

Рисунок 16 – Итоговый список объектов в Microsoft Access

Запрос «3) Студенты по имени» (рис. 17) выводит список всех студентов, чье имя начинается с заданных символов (в примере это все имена, начинающиеся с «Алекс»). Для этого необходимо выполнить SQL-запрос, представленный на рисунке 18.

3) Студенты по имени

Код	Фамилия	Имя	Отчество
5	Александрова	Александра	Александровна
6	Новиков	Александр	Игоревич
(№)			

Рисунок 17 – Пример запроса «3) Студенты по имени»

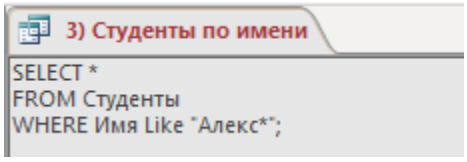

Рисунок 18 – Запрос «3) Студенты по имени»

Запрос «4) Среднее кол-во Лаб по АиП» (рис. 19) подсчитывает среднее количество лабораторных работ, выполненных студентами по дисциплине АиП, а также вычисляет процент выполненных работ (от общего числа работ, которое должна выполнить вся группа студентов).

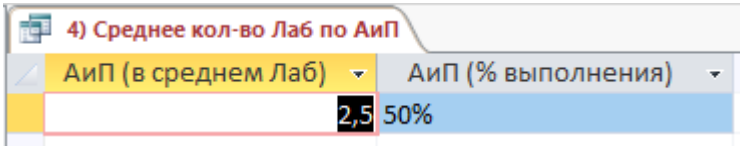

Рисунок 19 – Пример запроса «4) Среднее кол-во Лаб по АиП»

Здесь применяются агрегатные функции. На рисунке 20 представлена часть требуемого SQL-запроса, но его нужно доработать самостоятельно, чтобы получить результат, аналогичный изображенному на рисунке 19.
!!! Логические значения, как и числа, можно суммировать, используя операцию сложение «+». Стоит обратить внимание, что в **Access** установленная галочка ☑ дает значение «-1», а не «+1», как может показаться на первый взгляд**!**

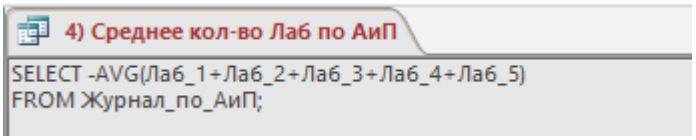

Рисунок 20 – Запрос «4) Среднее кол-во Лаб по АиП»

Аналогично запрос «5) Среднее кол-во Лаб по СУБД» (рис. 21) подсчитывает среднее количество лабораторных работ, выполненных студентами по дисциплине СУБД. Стоит обратить внимание, что в одном случае лабораторных было 4, а в другом 5.

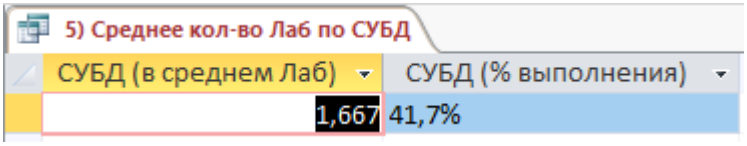

Рисунок 21 – Пример запроса «5) Среднее кол-во Лаб по СУБД»

Запрос «6) Итог» (рис. 22) подсчитывает количество лабораторных работ, выполненных каждым студентом. Здесь для получения результата используются сразу все 3 таблицы. Заготовка SQL-запроса представлена на рисунке 23.

6) Итог

Фамилия	Имя	Отчество	СУБД (сдано Лаб)	АиП (сдано Лаб)
Иванов	Иван	Иванович	0	0
Петров	Петр	Петрович	3	5
Сидоров	Сидор	Сидорович	1	3
Николаев	Николай	Николаевич	0	1
Александрова	Александра	Александровна	2	1
Новиков	Александр	Игоревич	4	5

Рисунок 22 – Пример запроса «6) Итог»

6) Итог

```
SELECT Фамилия, Имя, Отчество,
       -(Журнал_по_СУБД.Лаб_1+Журнал_по_СУБД.Лаб_2+Журнал_по_СУБД.Лаб_3+Журнал_по_СУБД.Лаб_4)
FROM Студенты, Журнал_по_СУБД, Журнал_по_АиП
WHERE (Журнал_по_СУБД.Студент=Студенты.Код) AND (Журнал_по_АиП.Студент=Студенты.Код);
```

Рисунок 23 – Запрос «6) Итог»

Запрос «7) Должники» (рис. 24), наоборот, выводит количество несданных лабораторных работ. Студенты, сдавшие все лабораторные работы, исключаются из таблицы. Список сортируется по фамилиям.

7) Должники

Фамилия	Имя	Отчество	СУБД (осталось сдать Лаб)	АиП (осталось сдать Лаб)
Александрова	Александра	Александровна	2	4
Иванов	Иван	Иванович	4	5
Николаев	Николай	Николаевич	4	4
Петров	Петр	Петрович	1	0
Сидоров	Сидор	Сидорович	3	2

Рисунок 24 – Пример запроса «7) Должники»

Отчет должен содержать:

— цель и задачи;
— описание хода работы (Что, как и в какой программе делалось? Со скриншотами);
— скриншоты каждой из созданных таблиц (минимум 10 строк в каждой таблице);
— текст SQL-запросов (7 шт.) и скриншоты результатов;
— описание использованных команд SQL;
— титульный лист, номера страниц, оглавление, список использованной литературы (включая Интернет-ресурсы и данную методичку), выводы/заключение и т.п.

1.5. Синтаксис языка SQL

!!! При выполнении студенческих работ запросы должны оформляться в несколько строк (как показано в примерах ниже) так, чтобы основные ключевые слова начинались с новой строки**!**

1. Комментарий

В SQL существует несколько вариантов записи комментариев:

```
/* Текст комментария
   (может занимать несколько строк) */

# Однострочный комментарий
```

-- Другой вариант однострочного комментария

!!! В последнем из вариантов, «пробел» после символов «--» является обязательным.

!!! Microsoft Access не позволяет писать комментарии в SQL-запросах. Но далее в **MySQL** их применение в студенческих работах обязательно**!**

2. Создание таблицы

Например, список в котором отмечено сколько лабораторных работ выполнил каждый из студентов, может быть создан следующим образом:

```
-- Создаем таблицу
CREATE TABLE Students (No INTEGER PRIMARY KEY,
                       FIO VARCHAR(50),
                       Lab_Count INTEGER);
```

Фактически эта команда создает «шапку» для таблицы (рис. 25) и предписывает, данные какого типа будут храниться в каждом столбце.

No	FIO	Lab_Count

Рисунок 25 – Шапка таблицы

В общем виде записывается как:

```
CREATE TABLE имя_табл (столбец_1 тип_данных_1 [PRIMARY KEY]
                       [NOT NULL] [UNIQUE]
                       [AUTO_INCREMENT] [DEFAULT …],
                       столбец_2 тип_данных_2 … );
```

Имена столбцов и таблиц не могут содержать пробелов (но можно использовать символ подчеркивания «_»). Также в них не рекомендуется использовать русские буквы (т.к. это может поддерживается не всеми версиями SQL). *Пробелы и др. символы можно использовать, если заключить имя в «правильные» кавычки (зависящие от версии SQL).*

PRIMARY KEY – «Первичный ключ», однозначно идентифицирующий каждую запись в таблице. Также означает, что данная колонка должна содержать **уникальные** значения, которые **не могут повторяться**. Не может принимать значение **NULL**. Таблица может иметь только один первичный ключ *(хотя он и может состоять из нескольких столбцов)*.

NOT NULL – означает, что значение не может быть пустым (содержать **NULL**). По существу, NOT NULL делает соответствующее поле **обязательным для заполнения** командой **INSERT**.

UNIQUE – ограничение, гарантирующее что все значения в столбце различны (уникальны)[1]. **PRIMARY KEY** включает в себя **UNIQUE**, но в таблице может быть только один первичный ключ, а уникальных полей может быть несколько.

AUTO_INCREMENT – позволяет автоматически генерировать уникальный номер при вставке новой записи в таблицу[2]. Часто применяется совместно с первичным ключом, для его автоматического заполнения. Начальное значение AUTO_INCREMENT = 1 и увеличивается на 1 при каждой новой записи.

DEFAULT – используется для установки значения по умолчанию для столбца[3]. Без него значением по умолчанию был бы **NULL**.

Основные типы данных[4]:

!!! В разных СУБД типы данных могут иметь разные названия. Даже если названия совпадают, могут отличаться их размер и другие детали. Всегда стоит проверять документацию на конкретную СУБД**!**

— **INT** или **INTEGER**– целые числа;

— **FLOAT** или **REAL** – числа с плавающей точкой (запятой);

— **BOOL** или **BOOLEAN** – логические значения;

!!! Стоит обратить внимание, что в отличие от других языков, в SQL тип **BOOL** имеет не два, а три варианта значений: «Да», «Нет» и «Не задано» (TRUE, FALSE, **NULL**), конечно, если для столбца не включен **NOT NULL**;

— **DATETIME** – дата и/или время;

— **VARCHAR**(n) – строка (текст), заданной предельной длинны **n** символов;

— **TEXT** – текст произвольной длинны, обычно применяется для многострочного текста;

!!! Строковые значения (текст) вводятся в одинарных кавычках.

— и др.

3. Добавление строк в таблицу

INSERT INTO имя_таблицы (столбец_1, столбец_2, …)
VALUES (значение_1, значение_2, …);

В скобках, после имени таблицы, могут быть указаны не все имена столбцов, которые использовались при создании таблицы (**CREATE TABLE**). Незадействованные поля будут заполнены значениями **NULL**. Но при этом столбцы, помеченные как **NOT NULL** обязательны для заполнения. Также здесь можно изменить порядок следования столбцов.

Например:

```
#Добавляем строку
INSERT INTO Students (No, FIO, Lab_Count)
VALUES (1, 'Иванов Иван Иванович', 2);
/* Причем, если имена столбцов идут в том же порядке, в каком они созданы командой CREATE, то их можно вообще не указывать */
INSERT INTO Students
VALUES (2, 'Петров Петр Петрович', 0);
```

```
/* Если нужно добавить сразу несколько строк, то INSERT можно написать только один раз */
INSERT INTO Students
VALUES (3, 'Сидоров Сидор Сидорович', 1),
       (4, 'Николаев Николай Николаевич', 0),
       (5, 'Синицын Синиц Синицевич', 3),
       (6, 'Новиков Александр Игоревич', 4);
```

Результат добавления строк представлен на рис. 26.

No	FIO	Lab_Count
1	Иванов Иван Иванович	2
2	Петров Петр Петрович	0
3	Сидоров Сидор Сидорович	1
4	Николаев Николай Николаевич	0
5	Синицын Синиц Синицевич	3
6	Новиков Александр Игоревич	4

Рисунок 26 – Таблица с добавленными строками

4. Вывод всего содержимого таблицы

```
SELECT * FROM имя_таблицы;
```

Например (см. рис. 26):

```
SELECT * FROM Students;
```

!!! В виде исключения, запрос на вывод всего содержимого таблицы можно (и даже нужно!) писать в одну строку.

5. Вывод части столбцов таблицы

```
SELECT выводимый_столбец_1, выводимый_столбец_2, …
FROM имя_таблицы;
```

Например:

```
SELECT Lab_Count, FIO
FROM Students;
```

Данная функция позволяет вывести не все столбцы (как при использовании звездочки *****), а только требуемые из них. Причем можно поменять их порядок. Результат выполнения запроса из примера представлен на рисунке 27.

Lab_Count	FIO
2	Иванов Иван Иванович
0	Петров Петр Петрович
1	Сидоров Сидор Сидорович
0	Николаев Николай Николаевич
3	Синицын Синиц Синицевич
4	Новиков Александр Игоревич

Рисунок 27 – Вывод части столбцов

6. Изменение имени столбца

выводимый_столбец **AS** "Новое имя столбца"

Например:

SELECT No, FIO **AS** "Ф.И.О.", Lab_Count **AS** "Сдано работ"
FROM Students;

В новом имени столбца можно использовать пробелы и русские буквы (рис. 28).

No	Ф.И.О.	Сдано работ
1	Иванов Иван Иванович	2
2	Петров Петр Петрович	0
3	Сидоров Сидор Сидорович	1
4	Николаев Николай Николаевич	0
5	Синицын Синиц Синицевич	3
6	Новиков Александр Игоревич	4

Рисунок 28 – Новые имена столбцов

7. Вывод части строк таблицы

Например, выведем список студентов, у которых не сдано две или более лабораторные работы из четырех, т.е. сдано 0, 1 или 2 работы (рис. 29):

SELECT *
FROM Students
WHERE Lab_Count < 3;

No	FIO	Lab_Count
1	Иванов Иван Иванович	2
2	Петров Петр Петрович	0
3	Сидоров Сидор Сидорович	1
4	Николаев Николай Николаевич	0

Рисунок 29 – Вывод части строк

В общем виде записывается как:

SELECT выводимый_столбец_1, выводимый_столбец_2, …
FROM имя_таблицы
WHERE условие_отбора;

Где «**условие_отбора**» – запись, содержащая **оператор сравнения** (или несколько операторов сравнения и логические операторы).

Логические операторы, используемые для объединения условий, представлены в таблице 1.

Операторы сравнения представлены в таблице 2. Сравниваемые значения могут быть числами, текстом (строковыми значениями) или датами и временем.

Таблица 1 – Логические операторы

Оператор	Описание
NOT	«НЕ», логическое отрицание, выводит противоположные значения
AND	«И», логическое умножение
OR	«ИЛИ», логическое сложение

Таблица 2 – Операторы сравнения

Оператор	Описание
Простые	
=	Равно
>	Больше
<	Меньше
< > *или* **!=**	Не равно
>=	Больше или равно (не меньше)
<=	Меньше или равно (не больше)
Сложные	
BETWEEN	«Между», в заданном диапазоне
IN	«В» списке (позволяет перечислить несколько значений для сравнения)
LIKE	«Подобный» («похожий»), поиск подстроки по шаблону

8. Оператор BETWEEN

WHERE сравниваемый_столбец **BETWEEN** от **AND** до;

Например, выведем список всех студентов (рис. 30), которые уже выполнили хотя бы одну лабораторную работу (>= 1), но еще не закончили выполнение всех работ (< 4):

```
SELECT *
FROM Students
WHERE Lab_Count BETWEEN 1 AND 3;
```

No	FIO	Lab_Count
1	Иванов Иван Иванович	2
3	Сидоров Сидор Сидорович	1
5	Синицын Синиц Синицевич	3

Рисунок 30 – Диапазон значений

Данное условие эквивалентно следующей записи через простые операторы сравнения:

```
WHERE (Lab_Count >= 1) AND (Lab_Count <= 3);
```

Если необходимо вывести значения, находящиеся наоборот за пределами диапазона, то применяется **NOT BETWEEN**. Например (рис. 31):

```
SELECT *
FROM Students
WHERE Lab_Count NOT BETWEEN 1 AND 3;
```

No	FIO	Lab_Count
2	Петров Петр Петрович	0
4	Николаев Николай Николаевич	0
6	Новиков Александр Игоревич	4

Рисунок 31 – За пределами диапазона

Данное условие с **NOT BETWEEN** эквивалентно следующей записи через простые операторы сравнения:

```
WHERE (Lab_Count < 1) OR (Lab_Count > 3);
```

9. Оператор IN

WHERE сравниваемый_столбец **IN** (Значение_1, Значение_2, …);

Например, узнаем количество выполненных лабораторных работ для заданного списка студентов (рис. 32):

```
SELECT *
FROM Students
WHERE FIO IN ('Иванов Иван Иванович',
              'Петров Петр Петрович',
              'Николаев Николай Николаевич');
```

No	FIO	Lab_Count
1	Иванов Иван Иванович	2
2	Петров Петр Петрович	0
4	Николаев Николай Николаевич	0

Рисунок 32 – Поиск по списку

Данное условие эквивалентно следующей записи через простые операторы сравнения и логический оператор **OR**:

```
WHERE  FIO = 'Иванов Иван Иванович' OR
       FIO = 'Петров Петр Петрович' OR
       FIO = 'Николаев Николай Николаевич';
```

Если необходимо вывести значения, наоборот, не входящие в список, то применяется **NOT IN**. Например (рис. 33):

```
SELECT *
FROM Students
WHERE FIO NOT IN  ('Иванов Иван Иванович',
                   'Петров Петр Петрович',
                   'Николаев Николай Николаевич');
```

No	FIO	Lab_Count
3	Сидоров Сидор Сидорович	1
5	Синицын Синиц Синицевич	3
6	Новиков Александр Игоревич	4

Рисунок 33 – Поиск вне списка

Данное условие с **NOT IN** эквивалентно следующей записи через простые операторы сравнения:

```
WHERE  FIO <> 'Иванов Иван Иванович' AND
       FIO <> 'Петров Петр Петрович' AND
       FIO <> 'Николаев Николай Николаевич';
```

10. Условие с указанием части строки (шаблона)

```
WHERE имя_столбца Like '%часть_значения%';
```

Позволяет проводить поиск подстроки по шаблону. Для замены в тексте некоторых символов используются следующие знаки:

— «**_**» *(нижняя черта)* – для замены ровно 1-го символа;

— «**%**» – для замены любого количества символов (от 0 до ∞ символов).

Можно также комбинировать данные знаки, например:

— «**_%**» – для замены от 1 до ∞ символов;

— «**__%**» – для замены от 2 до ∞ символов;

— и т.д…

— «**__**» – для замены ровно 2-ух символов;

— «**___**» – для замены ровно 3-ех символов;

— и т.д…

В сравниваемом с шаблоном тексте на месте этих знаков может располагаться указанное количество **любых** символов.

!!! В СУБД **Microsoft Access** вместо знаков «**_**» и «**%**» применяются, соответственно, знаки «**?**» (для одного символа) и «*****» (для любого количества символов).

Например, выведем список всех студентов, чьи фамилии начинаются на букву «Н» (рис. 34):

```
SELECT *
FROM Students
WHERE FIO Like 'Н%';
```

No	FIO	Lab_Count
4	Николаев Николай Николаевич	0
6	Новиков Александр Игоревич	4

Рисунок 34 – Поиск строки по шаблону

11. Сортировка

```
SELECT выводимый_столбец_1, выводимый_столбец_2, …
FROM имя_таблицы
WHERE условие_отбора
ORDER BY столбец_для_сортировки [ASC | DESC];
```

Для сортировки по **возрастанию**, к **ORDER BY** добавляется ключевое слово **ASC**. Для сортировки по **убыванию** к **ORDER BY** добавляется ключевое слово **DESC**. *По умолчанию (если не указанно ни **ASC**, ни **DESC**), сортировка производится по возрастанию.*

Например, выведем в начало списка студентов, сдавших большее количество работ (рис. 35):

```
SELECT *
FROM Students
ORDER BY Lab_Count DESC;
```

No	FIO	Lab_Count ▼ 1
6	Новиков Александр Игоревич	4
5	Синицын Синиц Синицевич	3
1	Иванов Иван Иванович	2
3	Сидоров Сидор Сидорович	1
2	Петров Петр Петрович	0
4	Николаев Николай Николаевич	0

Рисунок 35 – Сортировка по убыванию

Допускается (но не рекомендуется**!**) использовать вместо имени столбца для сортировки, его номер. Тогда равносильна примеру, рассмотренному выше, будет запись:

```
ORDER BY 3 DESC;
```

Кроме того, для сортировки можно применить сразу несколько столбцов, например:

```
ORDER BY Lab_Count DESC, FIO ASC;
```

В этом случае, вначале таблица будет отсортирована по убыванию Lab_Count, после чего строки с одинаковым значением Lab_Count будут выстроены по возрастанию FIO.

12. Агрегатные функции

Агрегатные функции *(групповые функции)* применяются сразу ко всем значениям в указанном столбце *(т.е. ко всей группе)*. В **SQL** существует 5 основных агрегатных функций:

— **SUM**(имя_столбца) – функция, возвращающая сумму значений указанного столбца. Может применяться только для **числовых** столбцов;

— **MIN**(имя_столбца) – функция, возвращающая минимальное значение в указанном столбце. Может применяться **не** только для числовых столбцов, но и для строк и дат;

— **MAX**(имя_столбца) – функция, возвращающая максимальное значение в указанном столбце. Может применяться **не** только для числовых столбцов, но и для строк и дат;
— **AVG**(имя_столбца) – функция, возвращающая среднее значение для указанного столбца. Может применяться только для **числовых** столбцов;
— **COUNT**(имя_столбца) – функция, возвращающая количество записей в указанном столбце, не содержащих **NULL**. Может применяться для **любых** столбцов (чисел, строк, дат и др.);
— **COUNT**(*****) – вариант записи функции, возвращающий количество строк в таблице (в группе). При такой записи на эту функцию никак не влияют значения **NULL**, т.к. для нее не указывается столбец.

Например, определим среднее, минимальное и максимальное количество выполненных студентами лабораторных работ, а также общее количество сданных в группе отчетов (рис. 36):

```
SELECT AVG(Lab_Count), SUM(Lab_Count), MIN(Lab_Count),
       MAX(Lab_Count)
FROM Students;
```

AVG(Lab_Count)	SUM(Lab_Count)	MIN(Lab_Count)	MAX(Lab_Count)
1.6667	10	0	4

Рисунок 36 – Агрегатные функции

Стоит различать два варианта применения агрегатных функций:

1) подсчет агрегатных функций ведется по всей таблице (как в примере выше), в запросе не применяется ключевое слово **GROUP BY**, результирующая таблица всегда состоит из **одной строки**. *При использовании данного вариант, все столбцы (перечисленные в **SELECT**) должны состоять только из агрегатных функций**!** Невозможно вывести столбцы без агрегатных функций (демонстрация чего будет представлена в раздел **5.2**)*;
2) в запросе применена группировка через **GROUP BY**, подсчет агрегатных функций ведется для каждой из групп, количество строк в результирующей таблице равно количеству групп. *Можно вывести столбцы **не** только с агрегатными функциями (но также и те столбцы, по которым ведется группировка).*

Подробнее про использование агрегатных функций см. раздел **5.2** «Агрегатные функции и группировка» в **Главе 5** «СЛОЖНЫЕ SQL-ЗАПРОСЫ».

13. Группировка

```
SELECT выводимый_столбец_1,
       АГРЕГАТ_ФУНКЦИЯ(выводимый_столбец_2), …
FROM имя_таблицы
WHERE условие_отбора
```

GROUP BY столбец_для_группировки
HAVING условие_группировки;

В качестве выводимых столбцов здесь можно использовать только столбцы, участвующие в группировке и агрегатные функции**!** Например, сгруппируем по количеству сданных работ, и найдем количество человек в каждой группе (рис. 37):

SELECT Lab_Count AS "Сдано работ", COUNT(*) AS "Кол-во человек"
FROM Students
GROUP BY Lab_Count;

Сдано работ	Кол-во человек
0	2
1	1
2	1
3	1
4	1

Рисунок 37 – Группировка

!!! В данном случае, нет разницы написать «COUNT(*)», или «COUNT(Lab_Count)» (или даже «COUNT(No)», или «COUNT(FIO)»), т.к. все поля исходной таблицы «Students» содержат значения (не содержат **NULL**). Но в случае наличия в таблице значения **NULL** разные способы могут вернуть разные результаты**!**

Для группировки можно применить сразу несколько столбцов, например:

GROUP BY столбец_для_группировки_1, столбец_для_группировки_2, …

При группировке практически всегда используются агрегатные функции (**SUM**, **COUNT**, **MIN**, **MAX**, **AVG**).

Ключевое слово **HAVING** является необязательным. Подробнее про **GROUP BY** и **HAVING** см. раздел **5.2** «Агрегатные функции и группировка» в **Главе 5** «СЛОЖНЫЕ SQL-ЗАПРОСЫ».

14. Объединение таблиц

До этого момента все манипуляции мы проводили только с одной таблицей. Для использования в запросе сразу нескольких таблиц необходимо обратить внимание на следующие отличия:

— во **FROM** нужно **перечислить несколько таблиц**:

FROM имя_таблицы_1, имя_таблицы_2, …

— используются **длинные имена столбцов**. При использовании в запросе нескольких таблиц, все имена столбцов должны записываться через **точку** в виде:

имя_таблицы.имя_столбца_в_таблице

В запросе, состоящем всего из одной таблицы, можно использовать только имя столбца без указания имени самой таблицы.

!!! *На самом деле, короткие имена можно использовать и в запросах, состоящих из нескольких таблиц, при условии, что эти имена уникальны (т.е. присутствуют только в одной таблице). Но в студенческих работах использование длинных имен является обязательным для всех запросов, состоящих из нескольких таблиц!*

— в **WHERE** прописываются **связь таблиц**. Для объединения таблиц одним из условий в **WHERE** должна выступать запись следующего вида:

таблица_1.имя_столбца = таблица_2.имя_столбца

Как правило, данное «имя_столбца» в обеих таблицах имеет одно и тоже название. Но это не является обязательным.

Для объединения условий используются логические операторы, в данном случае логическое умножение **AND**.

В общем виде запрос к нескольким таблицам можно записать следующим образом:

SELECT имя_таблицы.столбец_1, имя_таблицы.столбец_2, …
FROM имя_таблицы_1, имя_таблицы_2, …
WHERE связи_таблиц **AND** условие_отбора;

Также связь таблиц может быть реализована не через **WHERE**, а через оператор **JOIN** (имеющий варианты **INNER JOIN**, **LEFT JOIN** и **RIGHT JOIN**). **JOIN** используется совместно с оператором **ON**, в котором и указывается связь столбцов таблиц (в том же виде, как это делается в **WHERE**):

ON таблица1.имя_столбца = таблица2.имя_столбца

Подробнее про **INNER JOIN**, **LEFT JOIN** и **RIGHT JOIN** см. раздел **5.3** «Объединение таблиц (JOIN)» в **Главе 5** «СЛОЖНЫЕ SQL-ЗАПРОСЫ».

15. Структура запроса SELECT

Соберем воедино все ключевые слова языка **SQL**, перечисленные выше, тогда получаем:

```
SELECT выводимый_столбец1 AS "Новое имя 1",      /* SUM, COUNT, MIN, MAX, */
       выводимый_столбец2 AS "Новое имя 2", …  /* AVG, COALESCE, … */
FROM имена_таблиц     /* INNER JOIN, LEFT JOIN, RIGHT JOIN */
ON связи_таблиц_через_join
WHERE связи_таблиц_без_join AND условие_отбора  /* =, >, <, <>, >=, <=, BETWEEN,
                                                   IN, LIKE, NOT, AND, OR */
GROUP BY столбец_группировки
HAVING условие_группировки  /* Тоже, что для WHERE, плюс
                               SUM, COUNT, MIN, MAX, AVG */
```

```
ORDER BY столбец_сортировки [ASC|DESC]  /* SUM, COUNT, MIN, MAX, AVG */
LIMIT 20 OFFSET 0;
```

!!! В данном примере все основные ключевые слова являются началом новой строки. Именно этого стиля оформления стоит придерживаться.

Использованные выше ключевые слова **LIMIT** и **OFFSET**, а также функция **COALESCE**, ранее еще не упоминались, о их применении речь пойдет в **Главе 5** «СЛОЖНЫЕ SQL-ЗАПРОСЫ».

Справа от некоторых строк в виде комментария /* */ перечислены ключевые слова, функции и операторы, которые могут применяться в этой строке.

Большинство из перечисленных строк не являются обязательным, и, следовательно, могут не использоваться в тех или иных запросах. Обязательными здесь являются только ключевые слова **SELECT** и **FROM**.

*Отладка запросов SELECT

В случае, если запрос **SELECT** возвращает не тот результат, который ожидал пользователь, стоит временно отключить часть команд этого запроса. Обычно (для целей отладки) строки, записанные выше, можно считать выполняемыми раньше. Тогда последовательность будет следующей:

1. Оставляем от запроса только «обязательную» часть, т.е. только строки **SELECT** и **FROM**, а также объединение таблиц (если их несколько).

Кроме того, считаем, что **агрегатные функции** относятся к строке **GROUP BY** (которая будет использована дальше, но сейчас ее еще нет). Тогда, если в списке столбцов *(в первой строке, после ключевого слова **SELECT**)* имеются агрегатные функции, то необходимо их **удалить**, оставив только имена столбцов. Например, вместо **SUM**(Lab_Count) оставим только Lab_Count.

Таким образом, на данном этапе от запроса останется следующее:

```
SELECT выводимый_столбец1 AS "Новое имя 1",      /* Без агрегатных */
       выводимый_столбец2 AS "Новое имя 2", …  /* функций!!!       */
FROM имена_таблиц     /* INNER JOIN, LEFT JOIN, RIGHT JOIN */
ON связи_таблиц_через_join
WHERE связи_таблиц_без_join;
```

!!! На этом этапе количество выводимых строк будет максимальным. Все дальнейшие действия могут только **уменьшать** их количество, но не добавлять новые строки.

!!! К следующему этапу переходим только после того, как этот этап возвращает верный результат.

2. Добавляем условия **WHERE**, отсекая ненужные строки; причем, если таких условий несколько, то их лучше добавлять постепенно.
3. Добавляем группировку и возвращаем агрегатные функции (если таковые имеются) в список столбцов.
4. Добавляем условие **HAVING**.
5. Применяем сортировку.
6. Применяем **LIMIT** и **OFFSET**.

16. Объединение строк

Для конкатенации (объединения строковых значений) в SQL используется функция CONCAT, например:

SELECT **CONCAT**(F, ' ', I, ' ', O) FROM …

Данная функция работает с различными типами входных данных, автоматически преобразуя их в строковые значения.

Кроме того, имеется также функция CONCAT_WS, которой в качестве первого параметра передается разделитель. Тогда тот же пример можно переписать как:

SELECT **CONCAT_WS**(' ', F, I, O) FROM …

Некоторые СУБД позволяют дополнительно использовать операторы конкатенации, например: «+», «&», «||». Но такой способ не универсален и может не работать в других СУБД.

!!! Microsoft **Access** (в связи с его неполной поддержкой SQL) не содержит функцию CONCAT, в нем для объединения строк используется оператор «**&**». *В некоторых случаях также возможно применение оператора «+», но он работает только если оба операнда строки, а не числа.*

!!! В **MySQL** наоборот, символы «+» или «&» применяются для иных целей, а для объединения строк необходимо использовать CONCAT.

MySQL наряду с функцией CONCAT может использовать двойной символ «||» для объединения строк, но для него нужно вначале правильно изменить режим работы MySQL (т.к. по умолчанию «||» используется как аналог оператора OR), например:

SET sql_mode = 'PIPES_AS_CONCAT';
SELECT F || ' ' || I || ' ' || O **FROM** …

17. Арифметические операции

Язык **SQL** поддерживает арифметические операции, представленные в табл. 3.

Таблица 3 – Арифметические операции

Операция	Описание
+	Сложение
-	Вычитание
*****	Умножение
/	Деление
%	Остаток от деления (деление по модулю), например: 18 % 5 = 3

На самом деле, для записи запроса **SELECT** обязательным не является даже оператор **FROM**. Такой запрос будет выполняться без обращения к какой-

либо таблице. Хотя подобные («истинно однострочные») запросы годятся разве что для проверки работы операторов и функций в учебных целях и не применяются в реальной работе. Проверим работу арифметических операций (рис. 38):

```
SELECT 18+5, 18-5, 18*5, 18/5, 18%5;
```

18+5	18-5	18*5	18/5	18%5
23	13	90	3.6000	3

Рисунок 38 – Арифметические операции

Арифметические операции, как ясно из названия, работают с числами и не работают со строковыми значениями.

В отличие от операторов сравнения или логических операторов, которые используются только в условиях **WHERE** (или **HAVING**), арифметические операции могут применяться практически в любом месте запроса (любом, где вообще имеет смысл написать число, например, там, где корректно будет написать числовую константу **5**).

Арифметические операции могут применяться в следующих местах запросов:

— в условиях **WHERE** (или **HAVING**), например:

```
WHERE a > 2*b + 1;
```

где a и b – имена сравниваемых полей (столбцов) или константные значения;

— в списке выводимых столбцов (в первой строке, после оператора **SELECT**). Например, рассчитаем сколько лабораторных работ (из **4**-ех) осталось выполнить каждому студенту (рис. 39):

```
SELECT FIO AS "Ф.И.О.", 4-Lab_Count AS "Осталось сдать работ"
FROM Students;
```

Ф.И.О.	Осталось сдать работ
Иванов Иван Иванович	2
Петров Петр Петрович	4
Сидоров Сидор Сидорович	3
Николаев Николай Николаевич	4
Синицын Синиц Синицевич	1
Новиков Александр Игоревич	0

Рисунок 39 – Расчетный столбец

Здесь мы как бы создали «виртуальный» столбец, называющийся «Осталось сдать работ», который был рассчитан из реального столбца Lab_Count («Сдано работ») и константы **4**.

Далее имеет смысл не выводить в списке должников тех, кому осталось сдать 0 работ, а также вывести самых злостных должников в начало списка.

Доработаем, соответственно, предыдущий запрос (рис. 40):

```
SELECT FIO AS "Ф.И.О.", 4-Lab_Count AS "Осталось сдать работ"
FROM Students
WHERE 4-Lab_Count > 0
ORDER BY 4-Lab_Count DESC;
```

Ф.И.О.	Осталось сдать работ
Петров Петр Петрович	4
Николаев Николай Николаевич	4
Сидоров Сидор Сидорович	3
Иванов Иван Иванович	2
Синицын Синиц Синицевич	1

Рисунок 40 – Список должников

Здесь мы задействовали арифметические операции не только в **SELECT**, но и в **WHERE** и **ORDER BY**.

Операции производятся не обязательно со столбцом и константой, можно, например, сложить значения из нескольких столбцов:

```
SELECT a+b+c AS "Сумма"
```

где a, b и c – имена полей (столбцов);

— также арифметические операции могут применяться в строке **SET** запроса на обновление (**UPDATE**);

— и др.

18. Обновление (изменение) данных в таблице

```
UPDATE имя_таблицы
SET обновляемый_столбец = новое_значение
WHERE условие_отбора;
```

Обычно, имя обновляемого столбца присутствует в запросах на обновление **дважды**:

— **оба раза в SET** (до и после равенства). Например, студент Иванов сдал еще одну лабораторную работу (рис. 41):

```
UPDATE students
SET Lab_Count = Lab_Count + 1   /* Увеличить на единицу */
WHERE FIO LIKE 'Иванов %';
```

No	FIO	Lab_Count
1	Иванов Иван Иванович	3
2	Петров Петр Петрович	0
3	Сидоров Сидор Сидорович	1
4	Николаев Николай Николаевич	0
5	Синицын Синиц Синицевич	3
6	Новиков Александр Игоревич	4

Рисунок 41 – Увеличение значения

— один раз в **SET** (до равенства), а второй раз в условии **WHERE**:

```
UPDATE имя_таблицы
SET обновляемый_столбец = новое_значение
WHERE обновляемый_столбец = старое_значение;
```

Например, требуется заменить в некоторой таблице «markets» все записи «СПб» на «Санкт-Петербург»:

```
UPDATE markets
SET City = 'Санкт-Петербург'
WHERE City = 'СПб';
```

19. Удаление части записей (строк) из таблицы

```
DELETE
FROM имя_таблицы
WHERE условие_отбора;
```

Если необходимо удалить часть строк из таблицы1, связанной при этом с таблицей2 (из которой ничего не удаляется), то:

```
DELETE имя_таблицы_1
FROM имя_таблицы_1, имя_таблицы_2
WHERE связи_таблиц AND условие_отбора;
```

20. Удаление таблицы

```
DROP TABLE имя_таблицы;
```

!!! В **MySQL** и **Microsoft SQL Server** очистка содержимого таблицы перед ее удалением не требуется. Хотя в некоторых других СУБД может понадобиться выполнить вначале **DELETE** и только потом **DROP**.

1.6. Функции в SQL

Функции не являются основным инструментом SQL, и в большинстве случаев можно обойтись без них *(кроме агрегатных функций **SUM**, **COUNT**, **MIN**, **MAX** и **AVG**)*. Тем не менее, функции поддерживаются SQL, и в некоторых случаях могут быть полезны.

!!! Разные СУБД могут включать разные комплекты функций (с похожими или различными названиями). Здесь рассматриваются функции, поддерживаемые СУБД **MySQL**[5].

Кроме того, SQL позволяет создавать собственные функции (подробнее об этом можно прочитать самостоятельно в разделах **1.7** и **2.9**).

Строковые функции

Строковые функции[6] представлены в таблице 4.

Таблица 4 – Строковые функции

Функция	Описание
CONCAT(s1, s2, ...), **CONCAT_WS**(separator, s1, s2, ...)	Объединяет строковые значения
UPPER(s) *или* **UCASE**(s), **LOWER**(s) *или* **LCASE**(s)	Преобразовывает строку к Верхнему/Нижнему регистру
TRIM(s), **LTRIM**(s), **RTRIM**(s)	Удаляет начальные и/или конечные пробелы из строки
REPLACE(s, old, new)	Заменяет в строке значение old на new. Если значений несколько, то заменяются все
FORMAT(val, n)	Форматирует число к формату '#, ###. ##', округляя его до n знаков после запятой (точки)
HEX(n)	Переводит число в его шестнадцатеричное представление
LEFT(s, n), **RIGHT**(s, n)	Извлекает n символов из строки, начиная Слева/Справа
CHAR_LENGTH(s) *или* **CHARACTER_LENGTH**(s)	Возвращает длину строки (в символах). *Не стоит путать с функцией **LENGTH**(s), которая возвращает длину строки в байтах*
POSITION(substr **IN** s) *или* **LOCATE**(substr, s, start=1)	Возвращает позицию первого вхождения подстроки в строку. Если подстрока не найдена, возвращает 0. Поиск выполняется без учета регистра. **!!!** *Стоит обратить внимание, что в некоторых функциях в SQL вместо запятых могут применяться ключевые слова*

Окончание табл. 4

Функция	Описание
SUBSTRING(s, start, length) *или* **SUBSTR**(s, start, length) *либо:* **SUBSTRING**(s **FROM** start **FOR** length) *или* **SUBSTR**(s **FROM** start **FOR** length)	Извлекает подстроку из строки, начиная с позиции start и длинной до length символов. Нумерация символов в строке начинается с **1**
REPEAT(s, n), **SPACE**(n)	Повторяет строку/пробел указанное количество раз
REVERSE(s)	Переворачивает строку

Математические функции

Математические функции представлены в таблице 5.

Таблица 5 – Математические функции

Функция	Описание
ABS(f)	Возвращает абсолютное значение числа (модуль)
SIGN(f)	Возвращает знак числа. Если число > 0, возвращается 1. Если число = 0, возвращается 0. Если число < 0, возвращается -1
SQRT(f), **POW**(base, exp) *или* **POWER**(base, exp)	Корень и степень
PI(), **SIN**(f), **COS**(f), **TAN**(f), **ASIN**(f), **ACOS**(f), **ATAN**(f)	Число Пи, тригонометрические функции и обратные тригонометрические функции
RADIANS(f), **DEGREES**(f)	Преобразует значение из градусов в радианы и наоборот
EXP(f), **LN**(f), **LOG**(base, f), **LOG10**(f), **LOG2**(f)	Экспонента в указанной степени и логарифмы
f1 **DIV** f2, f1 **MOD** f2 *или* **MOD**(f1, f2) *или* f1 **%** f2	Целочисленное деление и остаток от целочисленного деления. *Здесь представлены не только функции, но и операторы*
ROUND(f, n), **CEIL**(f), **FLOOR**(f), **TRUNCATE**(f, n)	Округление чисел

Окончание табл. 5

Функция	Описание
RAND()	Случайное «нецелое» число от 0 (включительно) до 1 (исключая), т.е. $0 \le R < 1$. Например, генерация целого числа от 0 до 100 (включительно): SELECT **TRUNCATE**(RAND()***101**, 0); *или* SELECT **FLOOR**(RAND()***101**); Кроме того, можно применять RAND() для случайной сортировки строк результата, например: SELECT * FROM Students **ORDER BY** RAND();

Условия

Функции для работы с условиями представлены в таблице 6.

Таблица 6 – Условия

Функция	Описание
IF(condition, value_if_true, value_if_false)	Возвращает первое из значений, если условие ИСТИНА, или второе значение, если условие ЛОЖЬ. Например: SELECT **IF**(500<1000, 'YES', 'NO'); *В SQL условие IF соответствует конструкции **IF-ELSE** (с двумя активными ветвями) из других языков программирования. Здесь ее нельзя использовать всего с двумя аргументами (с одной активной ветвью)**!***
ELT(n, val1, val2, val3, ...)	Выбор значения по номеру. Позволяет выбирать более чем из двух вариантов. Например: SELECT **ELT**(2, 'Собака', 'Кошка', 'Попугай'); вернет «Кошка»
ISNULL(expression)	Если выражение равно NULL, эта функция возвращает 1. В противном случае она возвращает 0. Эквивалентно записи: (expression=**NULL**)
IFNULL(value, alt_value)	Возвращает value, если он не равен NULL. Но в случае, если value =NULL, возвращает второе значение (alt_value). Эквивалентно записи: **IF**(value <>**NULL**, value, alt_value) или **IF**(**ISNULL**(value), alt_value, value)
COALESCE(val1, val2, val3, ...)	Возвращает первое ненулевое (≠NULL) значение в списке. Эквивалентно: IFNULL(val1, IFNULL(val2, IFNULL(val3,...))) При двух аргументах совпадает с: IFNULL(val1, val2)

Задача

Имеется список имен файлов (рис. 42). В случае если имя длиннее 12 символов, в итоговой таблице (рис. 43) оно усекается таким образом, чтобы отображались первые 6 символов и последние 4 символа, между которыми располагаются 2 точки.

No	FileName
1	Курсовик по АИП 1
2	Курсовик по АИП 2
3	Курсовик по АИП 3
4	Мой диплом
5	Мой диплом (финал)
6	Мой диплом (самая последняя версия)
7	Презентация (диплом)
8	Презентация (диплом) 2

Рисунок 42 – Исходная таблица

FileName
Курсов..ИП 1
Курсов..ИП 2
Курсов..ИП 3
Мой диплом
Мой ди..нал)
Мой ди..сия)
Презен..лом)
Презен..м) 2

Рисунок 43 – Итоговая таблица

***Дата и время**

Некоторые из функций для работы с датой и временем представлены в таблице 7.

Таблица 7 – Дата и время

Функция	Описание
NOW(), **CURRENT_TIMESTAMP**(), **CURRENT_DATE**(), **CURRENT_TIME**()	Возвращает текущие дату и время, или только текущую дату, или только текущее время

Окончание табл. 7

Функция	Описание
YEAR(datetime), **MONTH**(datetime), **DAY**(datetime), **HOUR**(datetime), **MINUTE**(datetime), **SECOND**(datetime), **DAYOFWEEK**(datetime)	Извлекает из указанной даты: год, месяц, день (в месяце), часы, минуты, секунды или день недели (начиная с воскресенья=1, понедельник=2, …, суббота=7). Например, для вывода текущего года используется: **YEAR**(**NOW**()) Для вывода текущего дня недели необходимо выполнить: **DAYOFWEEK**(**NOW**())

***Агрегатные функции**

Агрегатные функции [7] представлены в таблице 8. Эти функции применяются ко всему столбцу, выдавая единственное значение с результатом (в случае если **не** используется группировка), либо по одному результату для каждой из групп (в случае если используется **GROUP BY**). Агрегатные функции игнорируют строки, содержащие значение **NULL**. *Если необходимо при агрегировании использовать значения **NULL** как «ноль», а не как «пустое место», то можно применить функции **IFNULL** или **COALESCE** (подробнее см. раздел **5.5**).*

Таблица 8 – Агрегатные функции

Функция	Описание
MIN(expr), **MAX**(expr), **SUM**(expr), **AVG**(expr), **COUNT**(expr), **COUNT**(*)	Основные агрегатные функции. Вычисляют минимальное, максимальное, суммарное или среднее значение, либо подсчитывают количество строк
GROUP_CONCAT(expr), **GROUP_CONCAT**(expr **SEPARATOR** ', ')	Объединяет значения из нескольких строк таблицы в одну строку. Строки объединяются через запятую, либо через указанный **SEPARATOR**. *Результат усекается до максимальной длины, заданной системной переменной **group_concat_max_len**, которая имеет значение по умолчанию **1024***

Окончание табл. 8

Функция	Описание
BIT_AND(expr), **BIT_OR**(expr), **BIT_XOR**(expr)	Вычисляют побитовое И, ИЛИ, Ислк. ИЛИ. *В случае если применяются к столбцу, имеющему тип Boolean, являются не «побитовыми», а обычными «логическими» И, ИЛИ, Искл. ИЛИ*
STDDEV_POP(expr), **VAR_POP**(expr)	Статистические функции. Среднеквадратичное отклонение (оно же Стандартное отклонение, **Standard deviation**, или σ) и дисперсия (**Variance**, σ^2, D) для генеральной совокупности (по англ. «**population**» – «популяция»). $$\sigma = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(x_i - \bar{x})^2},$$ где σ – среднеквадратичное отклонение; N – количество значений (количество строк в таблице); x_i – текущее значение; $\bar{x}$ – среднее значение для всех строк таблицы. ***STDDEV_POP()** – это квадратный корень из **VAR_POP()**.* Вычисление функции **VAR_POP**(x), равносильно следующей записи через основные агрегатные функции и подзапросы: SELECT **SUM**((x-(SELECT **AVG**(x) FROM tbl)) *(x-(SELECT **AVG**(x) FROM tbl))) /**COUNT**(x) FROM tbl;
STDDEV_SAMP(expr), **VAR_SAMP**(expr).	Статистические функции. Среднеквадратичное отклонение (Стандартное отклонение, s) и дисперсия (s^2) для **выборки** (от англ. «**sample**»). *Выборка является только частью генеральной совокупности («популяции»).* Формула отличается тем, что в знаменателе стоит N-1 вместо N. $$s = \sqrt{\frac{1}{N-1}\sum_{i=1}^{N}(x_i - \bar{x})^2},$$ При больших N значения s и σ сливаются в одно. Т.е. разница проявляется только на небольших выборках. ***STDDEV_SAMP()** – это квадратный корень из **VAR_SAMP()**.*

****Оконные функции**

Оконные функции используются только совместно с ключевым словом **OVER**(). Некоторые из оконных функций[8] представлены в таблице 9.

Таблица 9 – Оконные функции

Функция	Описание
ROW_NUMBER()	Номер текущей строки
RANK(), **DENSE_RANK**()	Ранг текущей строки (с пропусками номеров или без пропусков номеров). *Эти функции следует использовать с **ORDER BY** для сортировки строк раздела в желаемом порядке. Без **ORDER BY** все строки являются одноранговыми!*
LAG(), **LEAD**()	Значения из предыдущей/следующей строки
FIRST_VALUE(), **LAST_VALUE**()	Значение из первой/последней строки

Например, если отсортировать строки, то их нумерация будет не по порядку (рис. 44):

```
SELECT No, FIO, Lab_Count
FROM Students
ORDER BY Lab_Count;
```

No	FIO	Lab_Count ▲ 1
2	Петров Петр Петрович	0
4	Николаев Николай Николаевич	0
3	Сидоров Сидор Сидорович	1
1	Иванов Иван Иванович	2
5	Синицын Синиц Синицевич	3
6	Новиков Александр Игоревич	4

Рисунок 44 – Неверная нумерация строк

Но возможно «восстановить» правильную нумерацию (рис. 45), используя оконную функцию **ROW_NUMBER**:

```
SELECT ROW_NUMBER() OVER() AS "Номер" , FIO, Lab_Count
FROM Students
ORDER BY Lab_Count;
```

Номер	FIO	Lab_Count ▲ 1
1	Петров Петр Петрович	0
2	Николаев Николай Николаевич	0
3	Сидоров Сидор Сидорович	1
4	Иванов Иван Иванович	2
5	Синицын Синиц Синицевич	3
6	Новиков Александр Игоревич	4

Рисунок 45 – Правильная нумерация строк

При сортировке некоторые из строк имеют одинаковый **ранг** (см. первые две строки на рис. 44), т.е. одинаковое право занять строку с определенным номером. Тем не менее, лишь одна из них окажется в этой строке, а другие будут расположены ниже в таблице. При этом невозможно гарантировать какая из одноранговых строк окажется выше.

Следующие пример демонстрирует вывод ранга (рис. 46):

```
SELECT No, FIO, Lab_Count,
    RANK() OVER(ORDER BY Lab_Count) AS "RANK",
    DENSE_RANK() OVER(ORDER BY Lab_Count) AS "DENSE_RANK"
FROM Students;
```

No	FIO	Lab_Count ▲ 1	RANK	DENSE_RANK
2	Петров Петр Петрович	0	1	1
4	Николаев Николай Николаевич	0	1	1
3	Сидоров Сидор Сидорович	1	3	2
1	Иванов Иван Иванович	2	4	3
5	Синицын Синиц Синицевич	3	5	4
6	Новиков Александр Игоревич	4	6	5

Рисунок 46 – Ранг с пропусками и без

Здесь функция **RANK**() ставит две первые строки на первое место, третье место достается третьей строке. Но при использовании данной функции возможны **пропуски** в нумерации. В случае если ранг необходимо присваивать **без пропусков** значений, используется функция **DENSE_RANK**(), т.к. «dense» по англ. «плотный». В этом случае первое место также займут две первые строки, а второе место достанется третьей строке.

Также рассмотрим пример (рис. 47), выводящий предыдущее, следующее, первое и последнее значения:

```
SELECT No, FIO, Lab_Count,
        LAG(Lab_Count) OVER() AS "LAG",
        LEAD(Lab_Count) OVER() AS "LEAD",
```

```
        FIRST_VALUE(Lab_Count) OVER() AS "FIRST_VALUE",
        LAST_VALUE(Lab_Count) OVER() AS "LAST_VALUE"
FROM Students
ORDER BY FIO;
```

No	FIO	Lab_Count	LAG	LEAD	FIRST_VALUE	LAST_VALUE
1	Иванов Иван Иванович	2	*NULL*	0	2	3
4	Николаев Николай Николаевич	0	2	4	2	3
6	Новиков Александр Игоревич	4	0	0	2	3
2	Петров Петр Петрович	0	4	1	2	3
3	Сидоров Сидор Сидорович	1	0	3	2	3
5	Синицын Синиц Синицевич	3	1	*NULL*	2	3

Рисунок 47 – Значения из других строк

Стоит обратить внимание, что попытка получить значение перед первым или после последнего, выдаст **NULL**.

****Агрегатные функции в качестве Оконных функций**

Большинство **агрегатных** функций, представленных в таблице 8 (кроме **GROUP_CONCAT**), можно также использовать как оконные функции. Для этого к ним необходимо применить ключевое слово **OVER**().

Оконные функции выполняют агрегатную операцию над набором строк запроса. Но, в отличие от агрегатных функций, они не «сворачивают» строки группы в одну. Одинаковый результат будет выдан для каждой строки запроса. Количество строк не изменится (не уменьшится).

Рассмотрим пример использования агрегатных функций как оконных (рис. 48):

```
SELECT No, FIO, Lab_Count,
        MIN(Lab_Count) OVER() AS "MIN",
        MAX(Lab_Count) OVER() AS "MAX",
        SUM(Lab_Count) OVER() AS "SUM",
        AVG(Lab_Count) OVER() AS "AVG",
        COUNT(Lab_Count) OVER() AS "COUNT"
FROM Students
ORDER BY FIO;
```

No	FIO ▲ 1	Lab_Count	MIN	MAX	SUM	AVG	COUNT
1	Иванов Иван Иванович	2	0	4	10	1.6667	6
4	Николаев Николай Николаевич	0	0	4	10	1.6667	6
6	Новиков Александр Игоревич	4	0	4	10	1.6667	6
2	Петров Петр Петрович	0	0	4	10	1.6667	6
3	Сидоров Сидор Сидорович	1	0	4	10	1.6667	6
5	Синицын Синиц Синицевич	3	0	4	10	1.6667	6

Рисунок 48 – Агрегатные оконные функции

Аналогичный запрос, но без **OVER**, вернет лишь одну единственную строку (рис. 49):

```
SELECT No, FIO, Lab_Count,
        MIN(Lab_Count) AS "MIN",
        MAX(Lab_Count) AS "MAX",
        SUM(Lab_Count) AS "SUM",
        AVG(Lab_Count) AS "AVG",
        COUNT(Lab_Count) AS "COUNT"
FROM Students
ORDER BY FIO;
```

No	FIO	Lab_Count	MIN	MAX	SUM	AVG	COUNT
1	Иванов Иван Иванович	2	0	4	10	1.6667	6

Рисунок 49 – Обычные агрегатные функции

В этом случае строки были «свернуты» в одну. Кроме того, для этого варианта (см. рис. 49) выдаются несоответствующие действительности значения первых трех колонок (No, FIO, Lab_Count). Это случилось потому, что вместе с агрегатными функциями нельзя использовать «свободные» колонки (без агрегатных функций). При этом в предыдущем варианте (см. рис. 48) использование таких «свободных» колонок допустимо.

В приведенных выше примерах использовался **OVER**() с **пустыми** скобками (без указания окна), т.е. расчет велся по всей таблице. Хотя расчет можно вести по нескольким независимым группам (окнам) при помощи ключевого слова **PARTITION BY**, указываемого в скобках.

!!! Подробнее про оконные функции можно узнать в [9].

1.7. **Создание собственных функций

Функция в SQL – это хранимая программа, в которую можно передавать параметры и получать результат. Имеется возможность создавать собственные функции.

Создание функции

Функция создается командой CREATE FUNCTION, которая в общем виде записывается как:

```
CREATE FUNCTION имя_функции (параметр_1 тип_1,
                             параметр_2 тип_2, ...)
RETURNS тип_результата
BEGIN
   -- Объявление переменных
   DECLARE ... [DEFAULT …]

   -- Основная часть программы
   ...

   -- Вывод результата
   RETURN ...
END;
```

Стоит обратить внимание на следующие моменты:

— объявление переменных (**DECLARE**) происходит вначале программы;
— переменным можно сразу задать начальные значения (**DEFAULT**);
— для присвоения значения переменной (в основной части программы), используется ключевое слово **SET** *(будет показано далее)*;
— в такой программе можно использовать условия (**IF**) или циклы (**WHILE**, **REPEAT** и **LOOP**);
— программа обязательно должна вернуть результат, командой **RETURN**.

Например, создадим функцию, вычисляющую факториал переданного числа N:

```
CREATE FUNCTION Factorial (N INT) RETURNS INT
BEGIN
   -- Объявление переменных
   DECLARE P INT DEFAULT 1;
   DECLARE i INT DEFAULT 1;

   -- Основная часть программы (в данном случае цикл)
   WHILE i <= N DO
      SET P = P*i;
      SET i = i + 1;
   END WHILE;

   -- Вывод результата
   RETURN P;
END;
```

!!! Пример создания функции в MySQL через PhpMyAdmin приведен далее в разделе **2.9**.

***DELIMITER**

Описываемая далее проблема при работе через **PhpMyAdmin** может решаться гораздо проще, через графический интерфейс (см. раздел **2.9**).

Имеется проблема, связанная с тем, что в качестве разделителя (DELIMITER-а) команд в SQL выступает точка с запятой, и она же используется при написании кода функции. Поэтому необходимо временно сменить имеющийся разделитель на любой другой, после чего вернуть его обратно, например:

DELIMITER //
CREATE FUNCTION ...
BEGIN
...
END //
DELIMITER ;

При этом после END должен использоваться именно этот, новый, разделитель.

Использование функции

Для проверки работы созданной функции (рис. 50) достаточно ввести запрос SELECT, например:

SELECT Factorial(4);

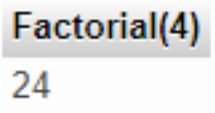

Рисунок 50 – Проверка работы функции

Функции можно использовать и в более сложных запросах, в тех же местах, где возможно использовать операторы (такие как, например, сложение, вычитание, умножение, деление, или объединения строк).

Удаление функции

Удаление созданной ранее функции производится командой DROP FUNCTION, например:

DROP FUNCTION Factorial;

****Агрегатные функции**

Кроме обычных функций, можно также создавать и собственные агрегатные функции, для этого используется команда **CREATE AGGREGATE**

FUNCTION, но такие функции должны загружаться из внешних библиотек через ключевое слово **SONAME**.

1.8. Подзапросы SQL

Подзапрос – это запрос, результат которого используется в другом (вышестоящем) запросе. При записи SQL-кода подзапросы **всегда** заключаются в круглые скобки.

Возьмем в качестве входных данных пример, рассмотренный ранее на рисунк 26. Необходимо вывести список всех студентов, которые учатся ниже среднего. Но при этом, нельзя писать:

```
SELECT * FROM students
WHERE Lab_Count < AVG(Lab_Count);
```

т.к. это выдаст ошибку.

Но можно решить (рис. 51) эту задачу, выполнив следующий SQL-запрос с подзапросом:

```
SELECT * FROM students
WHERE Lab_Count < (SELECT AVG(Lab_Count)
                   FROM students);
```

No	FIO	Lab_Count
2	Петров Петр Петрович	0
3	Сидоров Сидор Сидорович	1
4	Николаев Николай Николаевич	0

Рисунок 51 – Студенты, учащиеся ниже среднего

Здесь подзапрос возвращает значение **1.6667** (см. рис. 36). Далее это значение подставляется в вышестоящий запрос, где и производится сравнение. Основной запрос в данном случае равносилен выполнению:

```
SELECT * FROM students
WHERE Lab_Count < 1.6667;
```

Можно использовать подзапросы в различных видах запросов (SELECT, INSERT, UPDATE, DELETE), но сам подзапрос всегда должен быть SELECT. Подзапросы могут содержаться не только в WHERE, но и в строках HAVING, FROM, VALUES, SET, или в строке SELECT *(как один из столбцов, перечисленных в первой строке)*.

Оператор IN в подзапросах

Подзапросы могут возвращать не только одиночное значение (скалярное значение), но и целый столбец. В таком случае их можно использовать совместно с оператором **IN**, например:

```
SELECT * FROM students
```

```
WHERE FIO IN (SELECT FIO
              FROM students
              WHERE Lab_Count > 0);
```

Здесь вначале подзапрос вернет список фамилий студентов (рис. 52), которые выполнили хотя бы одну работу, после чего по этому списку будет выполнен основной запрос (рис. 53).

FIO
Иванов Иван Иванович
Сидоров Сидор Сидорович
Синицын Синиц Синицевич
Новиков Александр Игоревич

Рисунок 52 – Список фамилий

No	FIO	Lab_Count
1	Иванов Иван Иванович	2
3	Сидоров Сидор Сидорович	1
5	Синицын Синиц Синицевич	3
6	Новиков Александр Игоревич	4

Рисунок 53 – Результат выполнения основного запроса

Впрочем, конкретно этот запрос может быть переписан и без подзапроса:

```
SELECT *
FROM students
WHERE Lab_Count > 0;
```

Простые и сложные подзапросы

Простые подзапросы – это такие подзапросы, для которых можно вначале отдельно выполнить подзапрос, после чего подставить его значение в основной запрос. Простые подзапросы выполняются ровно один раз.

Сложные подзапросы (коррелированные подзапросы) – это подзапросы, которые выполняются множество раз, повторяясь для каждой строки основного запроса. Здесь невозможно заранее вычислить единственное значение для подзапроса, а потом использовать его в основном запросе, т.к. основной запрос и подзапрос переплетены (коррелированы).

!!! В студенческих работах простые подзапросы не рекомендуются. Автор должен обосновать необходимость их использования. Как говорится: «Первое правило подзапросов – не использовать подзапросы!». *В рассмотренном выше примере применение подзапроса обосновано тем, что в WHERE используется агрегатная функция. Но точно не стоит пытаться применять подзапросы*

вместо объединения таблиц (например, вместо Join) и не стоит использовать подзапросы, вложенные в другие подзапросы.

!!! Сложные подзапросы не рекомендуются в принципе, так как они являются, как правило, ошибкой проектирования. В студенческих работах они запрещены.

*Операторы ALL и ANY

Столбец из нескольких значений может использоваться не только с оператором **IN**, но и с операторами сравнения. Хотя при этом невозможно будет написать:

WHERE Value > (SELECT …)

т.к. это выдаст ошибку из-за того, что слева у нас единственное значение, а справа набор значений.

В таком случае можно писать:

WHERE Value > **ALL** (SELECT …)

или:

WHERE Value > **ANY** (SELECT …)

ALL – оператор возвращающий true, если все значения подзапроса удовлетворяют условию[10]. Это равносильно:

WHERE (Value > Value1) **AND** (Value > Value2) **AND** ...

ANY (он же **SOME**) – оператор возвращающий true, если любое (хотя бы одно) из значений подзапроса удовлетворяет условию[11]. Это равносильно:

WHERE (Value > Value1) **OR** (Value > Value2) **OR** ...

!!! Примеры были рассмотрены со знаком «>», но на его месте может использоваться любой другой оператор сравнения (>, <, = и т.д.).

**Ключевое слово DISTINCT

Добавление ключевого слова **DISTINCT** позволяет исключить повторы в возвращаемых значениях[12]. Например, если выполнить запрос:

SELECT Lab_Count FROM students;

То мы получим результат, представленный на рисунке 54.

Lab_Count
2
0
1
0
3
4

Рисунок 54 – Количество выполненных работ

Если теперь добавить перед именем выводимого столбца приставку **DISTINCT**, то повторы будут исключены (рис. 55):

SELECT **DISTINCT** Lab_Count FROM students;

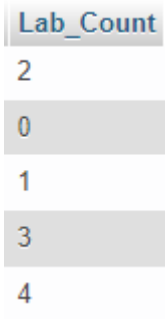

Рисунок 55 – Количество без повторов

Следующим образом можно, посчитать количество стран, задействованных в некой таблице Table1:

SELECT COUNT(**DISTINCT** Country) FROM Table1;

Ключевое слово **DISTINCT** часто используется с подзапросами, т.к. повторы в них обычно не нужны, например:

WHERE Value **IN** (SELECT **DISTINCT** …)

или:

WHERE Value = **ANY** (SELECT **DISTINCT** …)

1.9. *Виртуальные таблицы (Представления, View)

Представление (**View**)[13] – это виртуальная таблица, основанная на результате SQL-запроса. Представление отображает строки и столбцы, как и настоящая таблица.

В отличие от обычных таблиц, представление не хранит собственных данных. Содержимое представления динамически вычисляется на основании данных, находящихся в реальных таблицах. Изменение данных в реальной таблице немедленно отражается в содержимом всех представлений, построенных на основании этой таблицы.

Представление позволяет сохранить постоянную связь нескольких таблиц, и не описывать связь повторно в каждом запросе, а также при желании сменить названия столбцов. *Кроме того, представление дает возможность гибкой настройки прав доступа к данным за счет того, что права даются не на таблицу, а на представление. Это удобно в случае, если пользователю нужно дать права на отдельные строки таблицы или возможность получения не самих данных, а результата каких-то действий над ними (например, подсчет количества, или вычисление среднего).*

Представления являются чем-то средним между Запросами и обычными Таблицами, т.к. строят новую таблицу на основе запроса SELECT.

Создание представления

Для создания представления используется команда CREATE VIEW, после которой следует обычный запрос SELECT, например:

CREATE VIEW good_students **AS**
SELECT *
FROM students
WHERE Lab_Count > 0;

Представление – это исключительно способ сохранить SQL-запрос (на сервере). Представления не ускоряют выполнение запроса, а только упрощают работу программиста, структурируя данные.

При выполнении запроса в **PhpMyAdmin** весь введенный код SQL теряется после перехода к написанию следующего запроса или открытии содержимого таблицы. Чтобы не потерять код запроса и не вводить его повторно – можно сохранить его в Представление (см. раздел **2.8**).

В **MS Access** то, что названо «Запросами» (см. рис. 16), фактически является «сохраненными запросами» – т.е. Представлениями. При двойном щелчке по такому «Запросу», он откроется как таблица («Виртуальная таблица»).

Вывод содержимого

Содержимое представления выводится также, как и для обычной таблицы, командой SELECT:

SELECT * **FROM** good_students;

Представления можно использовать и в более сложных запросах, таким же образом и в том же месте, где и любую другую таблицу.

Запрос из Представления обрабатывается СУБД точно так же, как если бы на месте имени Представления находится Подзапрос. При этом СУБД перед выполнением запроса из представления (подзапроса) могут проводить совместную оптимизацию запроса верхнего уровня и запроса из представления (подзапроса).

Удаление представления

Для удаления представления используется команда:

DROP VIEW good_students;

2. СИСТЕМА УПРАВЛЕНИЯ БАЗАМИ ДАННЫХ MySQL

MySQL – свободная реляционная система управления базами данных (СУБД, РСУБД), одна из самых популярных реализаций языка и стандарта SQL. В настоящее время разработку и поддержку MySQL осуществляет корпорация **Oracle**. В 2008 году Sun Microsystems приобрела компанию MySQL AB (изначального разработчик MySQL) за 1 млрд долларов, в 2010 году Oracle приобрела Sun Microsystems за 7,4 млрд долларов. Продукт распространяется как под лицензией GNU GPL, так и под собственной коммерческой лицензией.

MySQL поддерживает множество платформ, в том числе работает на операционных системах Windows, Linux и macOS. MySQL (как впрочем и SQL вообще) поддерживается многими языками программирования, например такими как Delphi, Си, Си++, Java, PHP. *Примеры работы с MySQL на языке **PHP** будут продемонстрированы далее в **Главе 4**. Пример работы с MySQL в **Delphi** приведен далее в **Главе 6**.*

MariaDB – ответвление от системы управления базами данных MySQL, разрабатываемое сообществом под лицензией GNU GPL. Разработку и поддержку MariaDB осуществляет компания MariaDB Corporation Ab и фонд MariaDB Foundation. Создана в противовес политике лицензирования MySQL компанией Oracle. Ведущий разработчик MariaDB – Микаэль Видениус, автор оригинальной версии MySQL *(MySQL назван в честь его старшей дочери «Мю», а MariaDB в честь младшей – Марии)*.

MariaDB поддерживает высокую совместимость с MySQL. Несмотря на то, что у **MySQL** и **MariaDB** существует ряд серьезных отличий, в рамках данного курса мы будем считать их полностью идентичными, и говоря про MySQL, в равной мере будем подразумевать также и MariaDB. Но работать мы будем с MySQL.

2.1. Администрирование MySQL

Администрирование MySQL осуществляется при помощи программы PhpMyAdmin.

PhpMyAdmin – веб-приложение с открытым кодом, написанное на языке PHP и представляющее собой веб-интерфейс для администрирования СУБД MySQL. PhpMyAdmin позволяет через браузер осуществлять администрирование сервера MySQL, запускать команды SQL и просматривать содержимое таблиц и баз данных. Проект локализован на более чем 62 языках.

Для запуска **PhpMyAdmin** необходимо открыть любой браузер и ввести адрес **localhost** (или, что равносильно, адрес **127.0.0.1**). Данный адрес означает, что подключение производится к этому же компьютеру, т.е. серверная часть должна быть установлена на том же ПК. *Про установку серверной части подробнее рассказано в **Главе 4**, здесь же предполагается, что работа выполняется в компьютерном классе, где требуемое ПО уже установлено.*

После ввода адреса, в браузере отобразится главная страница WAMP-сервера (рис. 56).

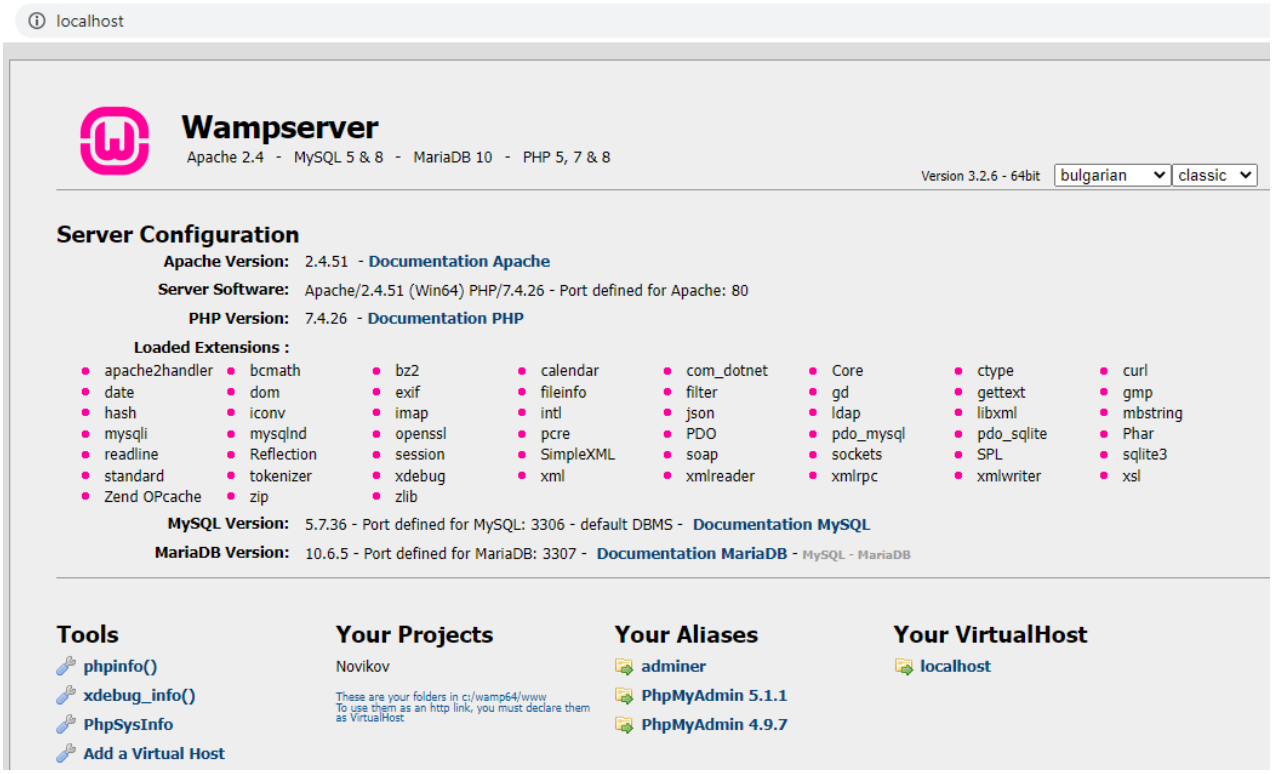

Рисунок 56 – Главная страница WAMP-сервера

Далее нужно перейти по ссылке **PhpMyAdmin** внизу страницы (рис. 57), если имеется несколько версий PhpMyAdmin, то лучше выбрать самую новую.

Рисунок 57 – Ссылки на администрирование СУБД

После чего мы попадаем в окно авторизации PhpMyAdmin (рис. 58), в котором нужно:

— указать имя пользователя **root**;
— поле для ввода пароля **оставить пустым**;
— выбрать базу данных (в нашем случае это **MySQL**, но здесь же можно было выбрать и **MariaDB**);
— нажать кнопку «**Вперёд**».

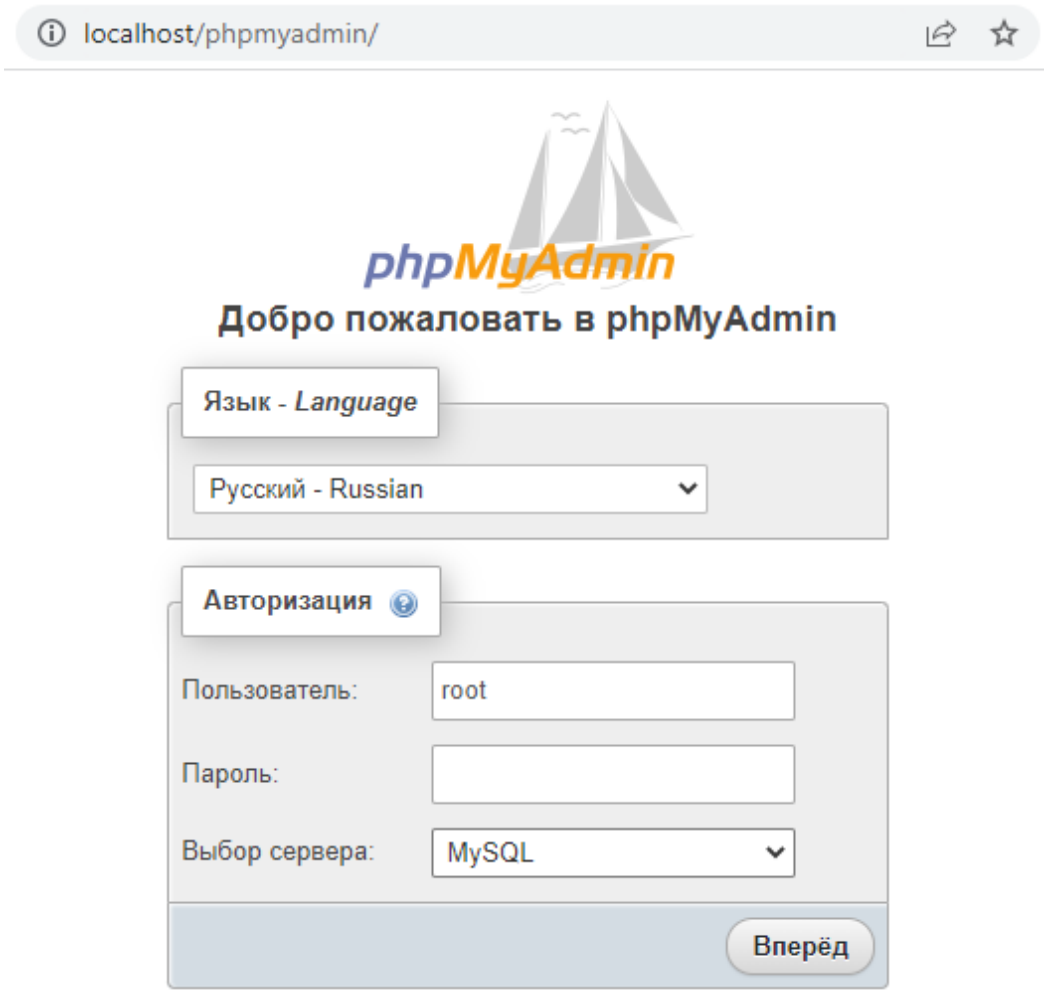

Рисунок 58 – Авторизации PhpMyAdmin

В результате мы попадаем на главную страницу PhpMyAdmin (рис. 59).

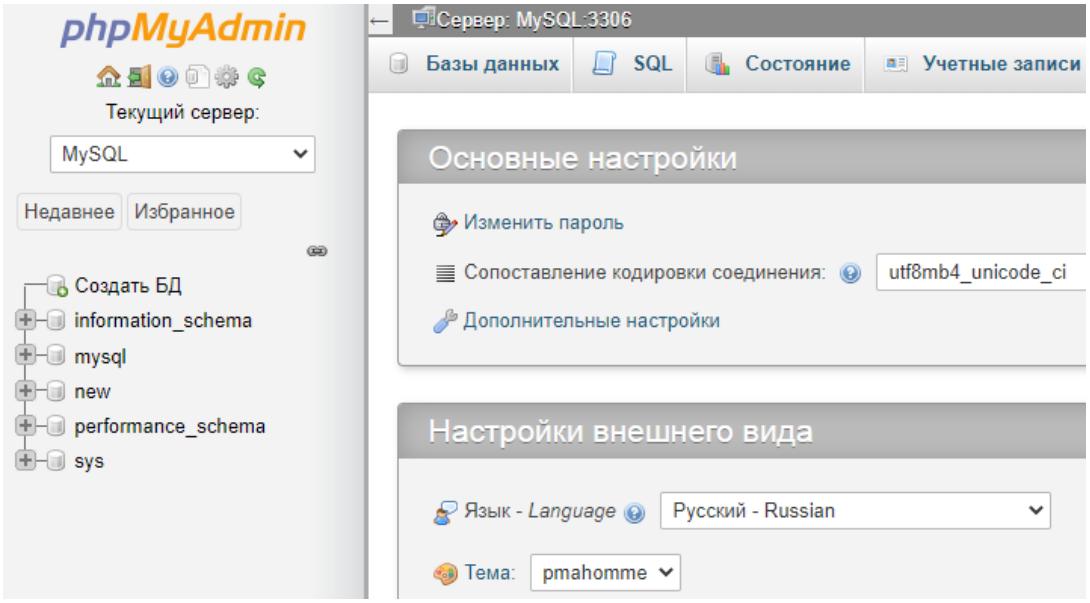

Рисунок 59 – Главная страница PhpMyAdmin

2.2. Создание базы данных

Выберем пункт «Создать БД» из меню слева (рис. 60).

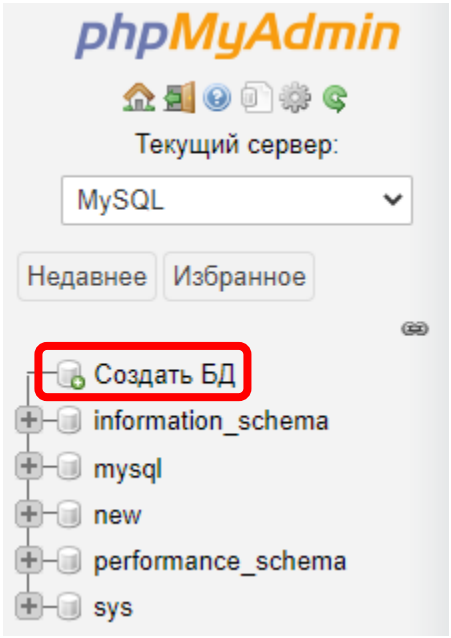

Рисунок 60 – Создание базы данных

Далее необходимо указать параметры (рис. 61) создаваемой базы данных: имя базы данных и кодировку символов.
!!! В качестве имени базы данных студенты указывают свою фамилию и текущий год! *В моих примерах это будет только фамилия «**Novikov**»*. Далее работа осуществляется только в этой базе, другие таблицы менять запрещено!
!!! В качестве кодировки, следует выбрать «**cp1251**», которая поддерживает русские символы (символы Кириллицы).

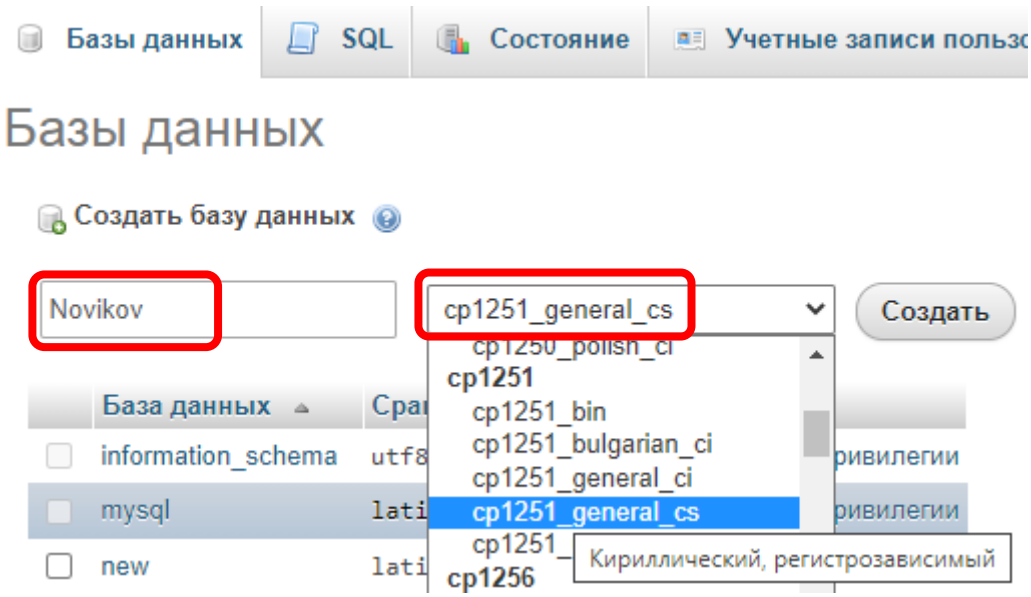

Рисунок 61 – Параметры создаваемой базы данных

После создания база данных появится страница с предложением создать таблицу в новой базе данных (рис. 62). Но нас больше интересует создание таблиц с помощью запросов на SQL.

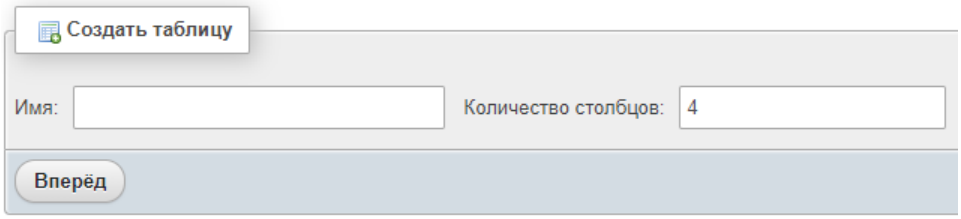

Рисунок 62 – Предложение создать таблицу

2.3. SQL-запросы в PhpMyAdmin

Для выполнения запросов на языке SQL требуется перейти на вкладку «SQL» (рис. 63).

1. Создание таблицы

Для создания таблицы необходимо написать соответствующий SQL-запрос **CREATE** (рис. 63).

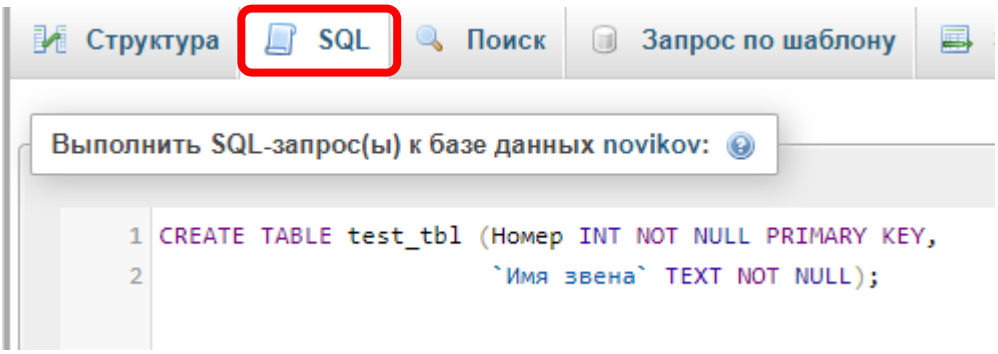

Рисунок 63 – Создание таблицы через SQL

В данном примере создается таблица с именем «**test_tbl**», имеющая два столбца с именами «**Номер**» и «**Имя звена**» (рис. 64).

Т.к. имя второго столбца содержит пробел, его обязательно нужно писать в кавычках. При этом для имен используются «косые» кавычки, которые вводятся с клавиатуры клавишей «ё» в английской раскладке. Стоит обратить внимание, что для ввода текстовых значений используются обычные одинарные кавычки, вводимые клавишей «э» в английской раскладке. Имя первого столбца «Номер» и имя таблицы «test_tbl» также могли быть взяты в косые кавычки, но в данном случае это было не обязательно.

Первый столбец имеет тип «**INT**», т.е. в нем будут храниться целые числа (номера). Второй столбец имеет тип «**TEXT**». Для обоих столбцов указано ключевое слово «**NOT NULL**», которое говорит, что соответствующие поля обязательны для заполнения и не могут быть оставлены пустыми. Кроме того,

для первого столбца также указано ключевое слово «**PRIMARY KEY**» (первичный ключ), говорящее, что значения в этом поле должны быть уникальными, т.е. не могут повторяться.

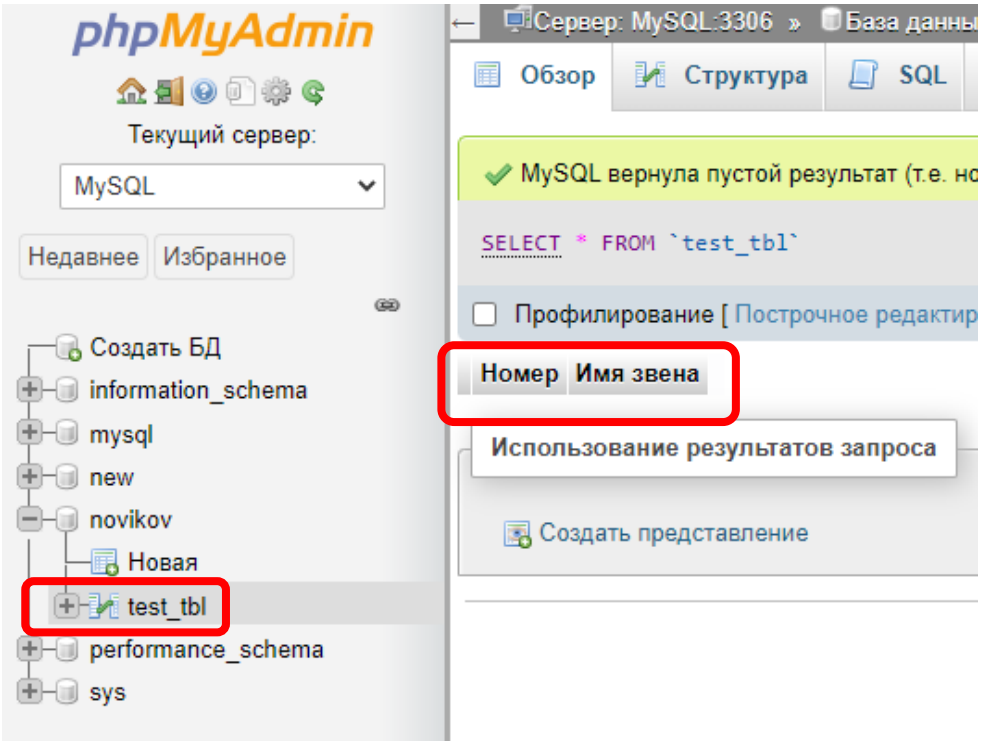

Рисунок 64 – Созданная таблица

2. Заполнение таблицы

Для заполнения таблицы строками используется ключевое слово «**INSERT**» (рис. 65).

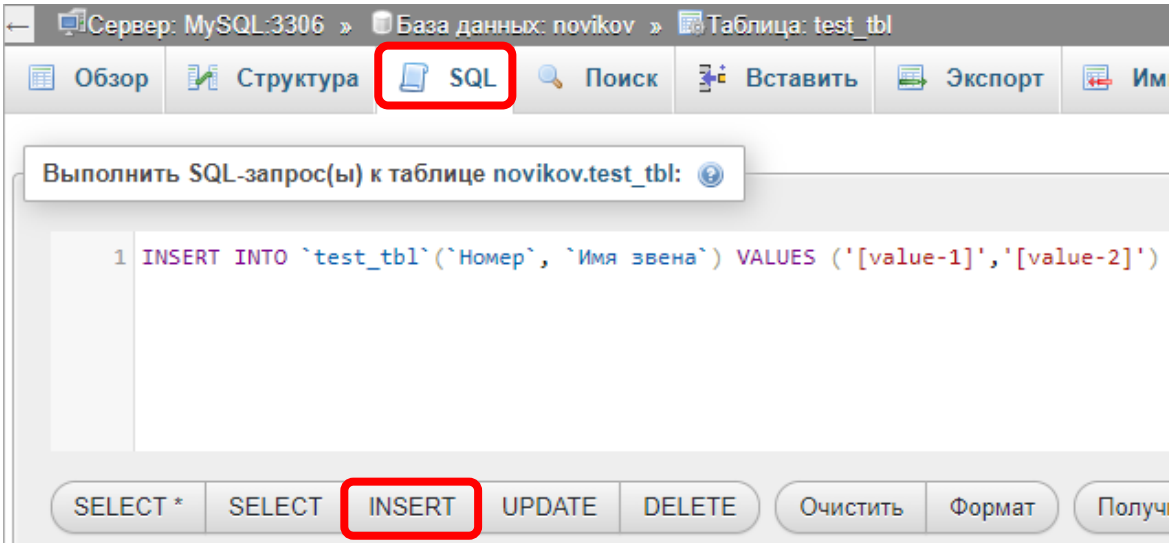

Рисунок 65 – Шаблон запроса

При этом **PhpMyAdmin** уже содержит удобные шаблоны для разных запросов, вызываемые нажатием соответствующей кнопки, в нашем случае кнопки «**INSERT**» (см. рис. 65), расположенной под полем ввода запроса.

Шаблон запроса уже содержит все имена столбцов из созданной ранее таблицы *(в нашем случае имена столбцов не понадобятся, т.к. мы заполняем все значения по порядку)*. Также в шаблон уже добавлены черновые значения в формате «**[value-1]**», вместо которых необходимо подставить свои значения.

Стоит отметить, что шаблон не содержит точки с запятой в конце запроса. **MySQL** допускает отсутствие точки с запятой в конце запроса, а если пишется сразу несколько запросов, то только в конце последнего из них.

!!! В студенческих работах использование точки с запятой обязательно в конце каждого запроса!

Исправим шаблон и скопируем его несколько раз для заполнения таблицы 5-ю строками (рис. 66).

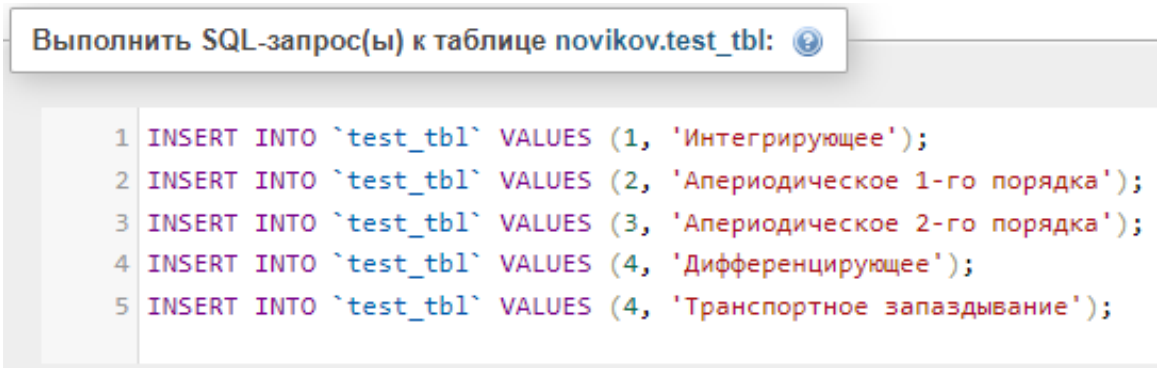

Рисунок 66 – Добавление строк в таблицу

При этом, если цифры были введены в точности как на рис. 66, то мы получим сообщение об ошибке (рис. 67).

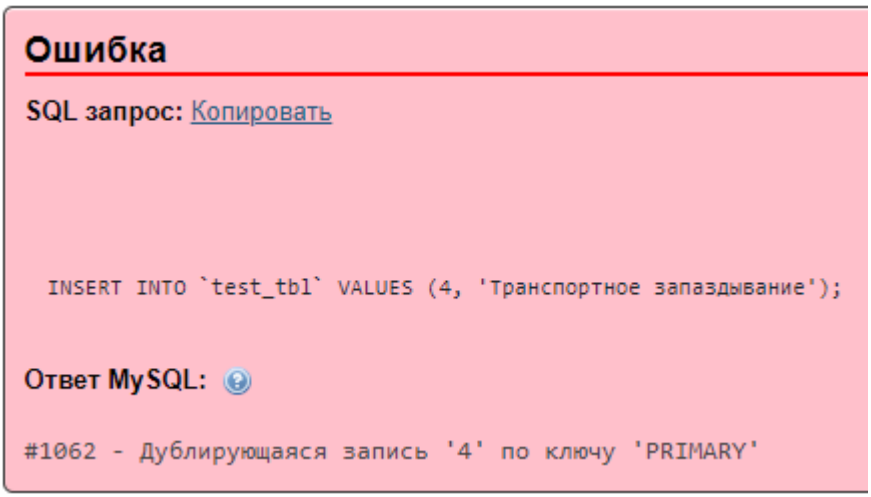

Рисунок 67 – Ошибка при добавлении строк в таблицу

Это связано с тем, что в уникальное поле («**PRIMARY KEY**») мы пытаемся записать два одинаковых номера 4. Перейдя на вкладку «Обзор» мы увидим, что в таблицу были добавлены только первые 4 строки (рис. 68).

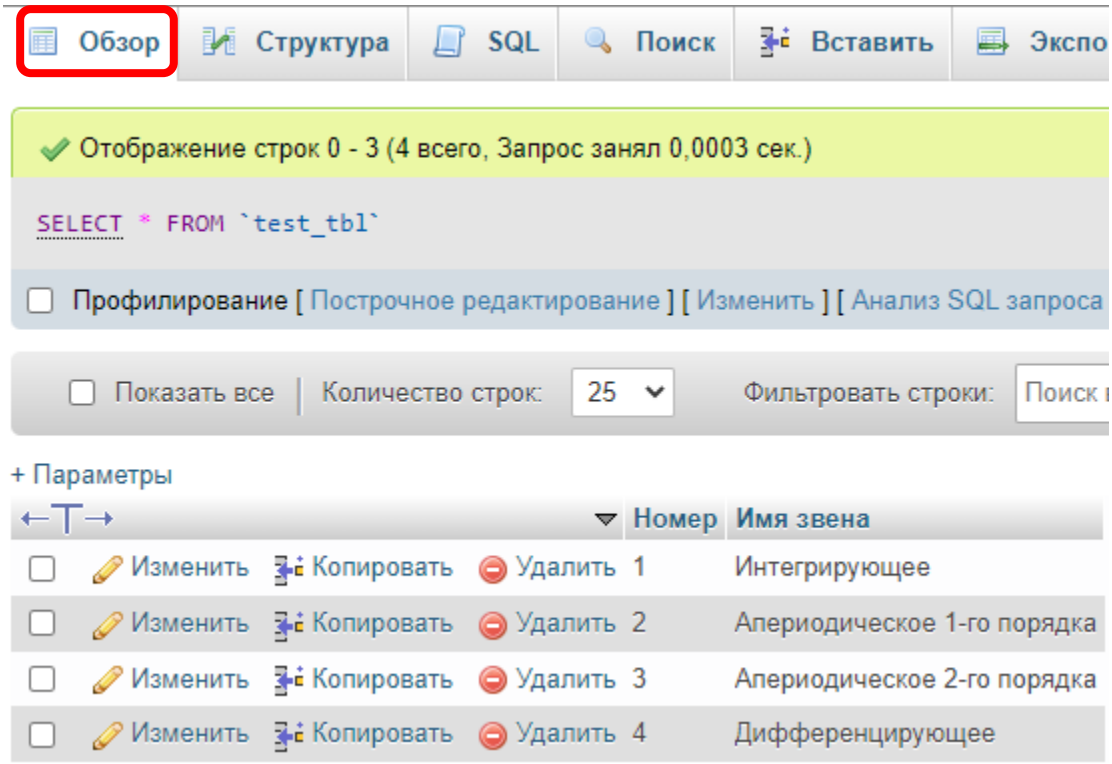

Рисунок 68 – Таблица с добавленными строками

3. Вывод таблицы

Как видно из рис. 68, вызов вкладки «Обзор» автоматически формирует запрос «**SELECT**» на вывод всего содержимого таблицы, в данном случае это запрос:

SELECT * **FROM** `test_tbl`

Если мы хотим выбрать только часть строк, то необходимо вновь перейти на вкладку «SQL» и ввести соответствующий запрос с **WHERE** (рис. 69).

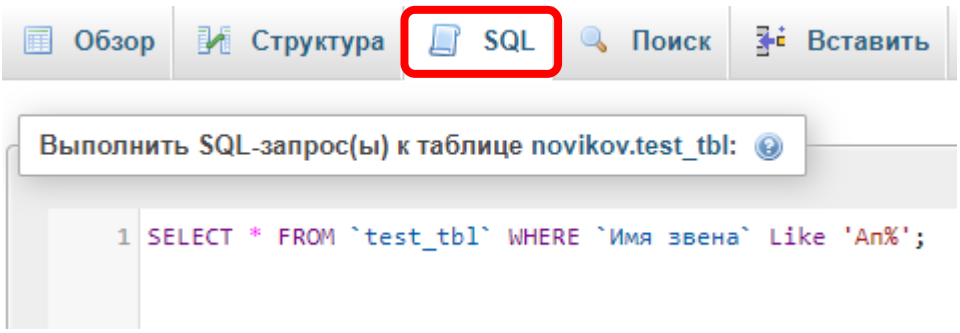

Рисунок 69 – Запрос SELECT с условием

В результате мы найдем 2 подходящие строки (рис. 70), из 4 исходных.

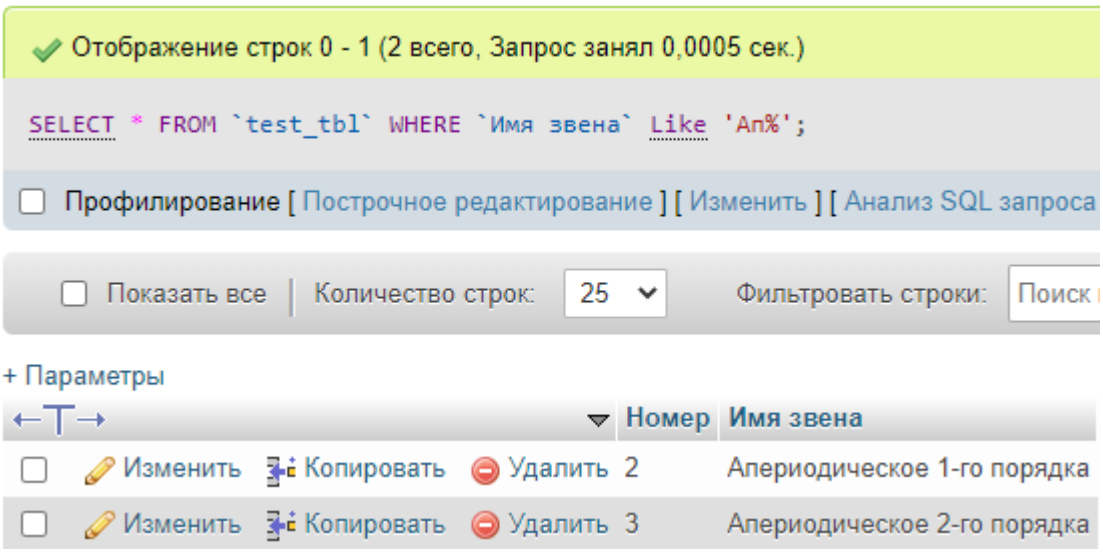

Рисунок 70 – Результат запроса SELECT

4. Удаление таблицы

Для удаления таблицы необходимо выполнить запрос «**DROP**» (рис. 71).

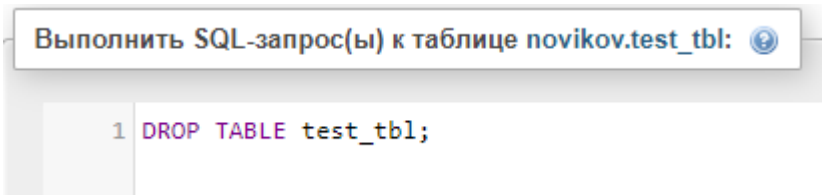

Рисунок 71 – Запрос на удаление таблицы

2.4. РАБОТА № 2
«Базы данных MySQL в PhpMyAdmin»

В данной работе, студенту предстоит выбрать **10** устройств, объединенных общей мыслью, и внести их в таблицу, разработанную в **MySQL**.

Выполнение данной работы рассматривается на примере датчиков температуры. Студенту необходимо выбрать свой тип датчиков (или др. устройств), которыми он будет заполнять таблицы. При этом нельзя выбирать датчики температуры (т.к. они уже есть в примере), а также необходимо, чтобы у студентов в группе были выбраны разные типы устройств.

Оборудование можно найти, например, посетив сайты ведущих российских производителей оборудования для промышленной автоматизации, таких как: *https://owen.ru/*, *https://segnetics.com/*, *https://www.mzta.ru/*. Не обязательно брать все устройства с одного сайта и не обязательно пользоваться именно этими сайтами.

!!! Необходимо выбрать оборудование, для которого имеются фотографии! Они еще понадобятся нам в дальнейшем *(в Главах **3** и **4**)*.

Далее приводится далеко не полный перечень промышленного оборудования, которое можно выбрать: датчики давления, датчики уровня, датчики влажности, программируемые контроллеры, модули для контроллеров, панели оператора (HMI), измерители-регуляторы (например, такие как Овен ТРМ1), реле и контакторы электромагнитные, твердотельные реле, программируемые реле, автоматические выключатели (автоматы), счетчики (на воду, на газ, на электричество – предпочтительно электронные, а не механические), частотные преобразователи, электроприводы заслонок и клапанов, насосы, блоки питания 24В, а также различное программное обеспечение для автоматизации (например, SCADA-системы).

Шапка таблицы с устройствами должна быть примерно следующей: наименование устройства, тип устройства, производитель, нижний предел измерения, верхний предел измерения, погрешность, выходной сигнал, описание, гарантия, цена, рейтинг *(последний можно придумать самостоятельно)*. Пример содержимого таблицы для выбранных датчиков температуры, представлен в табл. 10 и на рис. 72. *Если выбраны не датчики, а другие устройства, то некоторые из перечисленных параметров могут быть к ним не применимы, но у них появятся свои параметры, например, количество и типы входных/выходных сигналов для контроллеров или модулей к контроллерам.*

Кроме того, необходимо придумать вторую таблицу (**8-10** строк), в которой содержится информация о складах, на которых хранятся данные датчики, или магазины, в которых их можно приобрести. В этой таблице должны быть отражены: название магазина, адрес магазина, телефон магазина, время работы магазина и т.п. Пример содержимого таблицы со складами/магазинами, представлен на рис. 73.

Связь между этими двумя таблицами организована через третью таблицу, отрывок которой представлен на рис. 74. В этой таблице также содержится количество датчиков, имеющихся в наличии в указанном магазине. Необходимо предусмотреть варианты, при которых некоторые датчики имеются сразу в нескольких магазинах, другие только в одном магазине, а нескольких датчиков нет в наличие ни в одном магазине. В этой таблице должно быть не менее **15** строк.

Отчет должен содержать:

— цель и задачи;
— описание хода работы (Что, как и в какой программе делалось? Со скриншотами);
— текст SQL-запросов на создание и заполнение таблиц (10 устройств) и скриншоты каждой из созданных таблиц (достаточно только части каждой таблицы);

— пример запроса Select, выводящего содержимое всех таблиц в одну (текст SQL-запроса и скриншот результата);
— расшифровку терминов, которые встречаются в отчете (такие как SQL, PhpMyAdmin и т.п.);
— описание использованных команд SQL;
— титульный лист, номера страниц, оглавление, список использованной литературы (включая Интернет-ресурсы и данную методичку), выводы/заключение и т.п.

Таблица 10 – Пример таблицы с датчиками

Номер	1	2
Наименование	ДТС015-PT1000.B2.200	ДТС035М-50М.1,0.120.МГ.И [3]
Тип	Термометр сопротивления	Термометр сопротивления
Изображение	dts015.png	dts035m.png
Рейтинг	4,7	4,9
Производитель	Овен	Овен
Страна	Россия	Россия
Гарантия, месяцев	24	24
Цена, руб	2490,00	7998,00
Описание	ДТСхх5 с коммутационной головкой позволяют измерят...	Датчики изготавливаются на базе термометров сопрот...
Диапазон измеряемых температур	–60…+500 °C	0…+150 °C
Погрешность	0,5%	1%
Выходной сигнал	Сопротивление (Pt1000)	4...20 мА
Схема подключения	Двухпроводная	Двухпроводная
Напряжение питания (номинальное)	Не требуется	24 В
Диапазон допустимых напряжений питания	Не требуется	12...36 В
Ссылка на документацию	https://owen.ru/uploads/292/kratkoe_rukovodstvo_dtshh5.pdf	https://owen.ru/uploads/324/re_oven_dts-i_1-ru-18399-1.10.pdf
Температура окружающей среды	–60…+85 °C	–40…+85 °C

Окончание табл. 10

Количество чувствительных элементов	1	1
Длина погружаемой части L, мм	200	120
Материал коммутационной головки	Пластмасса	Металл
Класс защиты	IP54	IP65
Среда измерения	Твердые, жидкие и газообразные среды (неагрессивны...	Твердые, жидкие, газообразные и сыпучие среды
Сопротивление изоляции	не менее 100 МОм	
Средний срок службы	не менее 10 лет	не менее 10 лет
...	...	...

Номер	Наименование	Тип	Изображение	Рейтинг	Производитель	Страна	Гарантия	Цена
1	ДТС015-PT1000.B2.200	Термометр сопротивления	dts015.png	4.7	Овен	Россия	24	2490
2	ДТС035М-50М.1,0.120.МГ.И [3]	Термометр сопротивления	dts035m.png	4.9	Овен	Россия	24	7998
3	ДТС035М-50М.1,0.120.И [3]	Термометр сопротивления	dts035m.png	4.9	Овен	Россия	24	7398

Рисунок 72 – Пример таблицы с датчиками

Номер	Город	Адрес	Телефон	E-mail
1	г. Санкт-Петербург	ул. Александра Матросова, д. 4, корп. 2, литер Д	+7 (812) 677-56-..	sales@овенэюя.рф
2	г. Екатеринбург	ул. Малышева, 164	+7 (343) 228-18-..	sale@uralenergoxyz.ru
3	г. Москва	ул. Кусковская, дом 20А, офис В706	+7 (495) 777-83-..	zakaz@e-xyz.ru
4	г. Санкт-Петербург	пр. Стачек, д. 41	+7(812) 628-28-..	info@owen-xyz.ru
5	г. Псков	улица Советская, дом 49, помещение 4	+7 (911) 157-32-..	info@owen-xyz.ru

Рисунок 73 – Пример таблицы со складами

Номер записи	Номер датчика	Количество	Номер магазина
1	1	400	1
2	1	123	2
3	2	2	2
4	2	3	3
5	2	7	4

Рисунок 74 – Пример связи таблиц

2.5. *Работа с таблицами средствами PhpMyAdmin

PhpMyAdmin позволяет создавать и удалять таблицы, а также изменять их содержимое, не прибегая к написанию запросов на языке SQL.

1. Создание таблицы

Для создания таблицы необходимо в базе данных **со своей фамилией** нажать кнопку «Новая» (рис. 75).

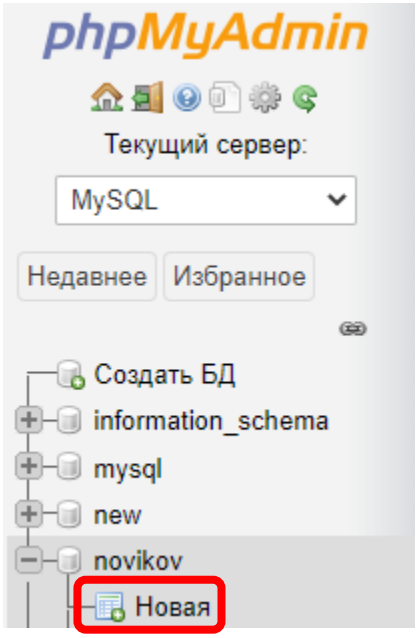

Рисунок 75 – Создание таблицы средствами PhpMyAdmin

В появившемся окне (рис. 76) требуется ввести имя создаваемой таблицы («test2_tbl») и указать имена столбцов (в нашем случае их три, «Номер», «Имя звена» и «Комментарий»), а также указать типы данных для этих столбцов.

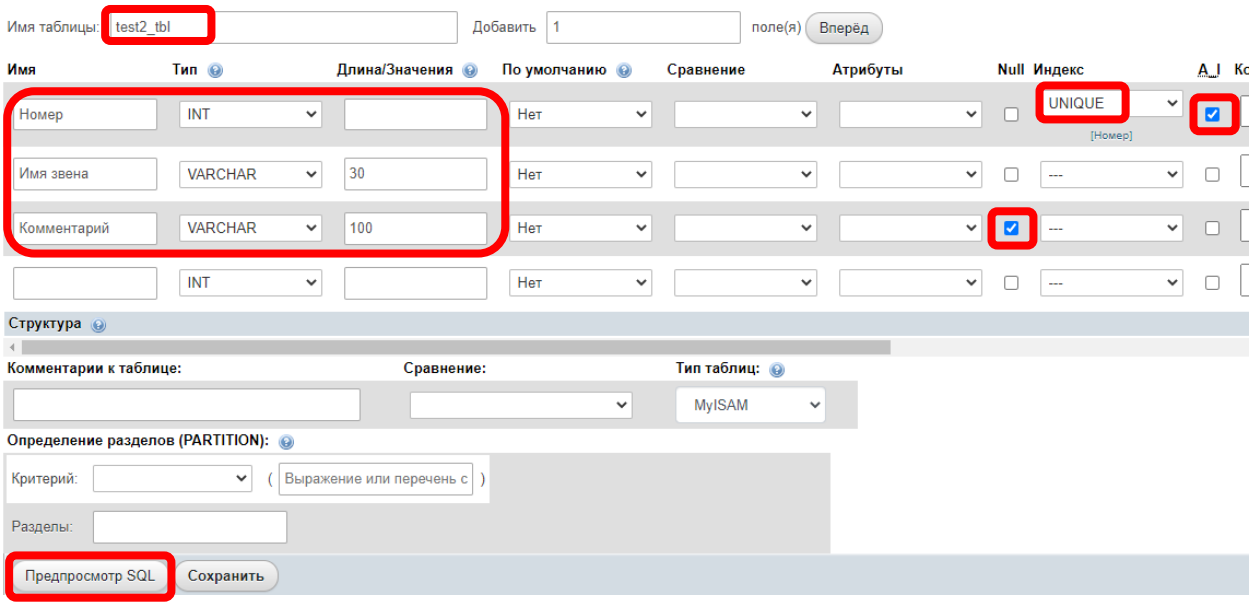

Рисунок 76 – Параметры создаваемой таблицы

Как и в прошлом примере, столбец «**Номер**» имеет тип «**Int**», т.е. будет целым числом. Для «**Имя звена**» и «**Комментарий**» укажем тип «**VarChar**», а не «**Text**» (как в прошлый раз). *Это связанно с тем, что при добавлении строк в таблицу через PhpMyAdmin (о котором будет рассказано далее), тип Text будет восприниматься как многострочный текст и поля для ввода будут занимать весь экран, а для VarChar мы получим компактные поля для ввода.* При этом VarChar также хранит текстовые значения. Для него необходимо задать максимальное количество символов, которое можно ввести в это поле (в нашем случае это будет «**30**» и «**100**»).

Для столбца «**Номер**» необходимо выбрать «**Unique**» («Уникальный») и поставить галочку «**Auto_Increment**» (**A_I**, «Автоматическое увеличение на единицу», которое в дальнейшем позволит не заполнять поле номера вручную). Для столбца «**Комментарий**» установим галочку «**Null**».

Перед тем как сохранить таблицу, нажмем кнопку «**Предпросмотр SQL**» и увидим автоматически сгенерированное содержимое SQL-запроса (рис. 77).
!!! В отчетах не принимаются запросы, сгенерированные автоматически, а соответствующая работа считается **невыполненной**!

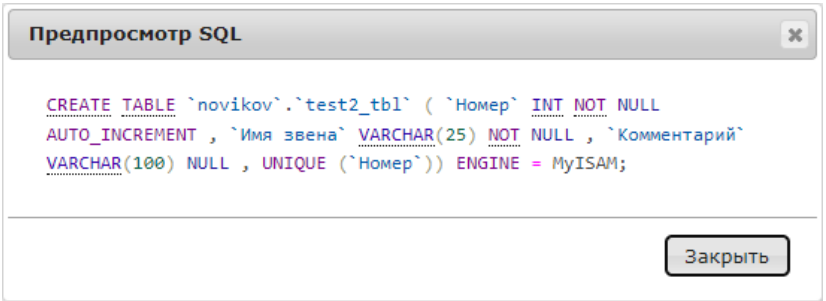

Рисунок 77 – Автоматически сгенерированный код SQL

2. Добавление строк в таблицу

Для добавления строк в таблицу необходимо перейти на вкладку «**Вставить**» (рис. 78). Здесь требуется заполнить «**Имя звена**» (в нашем случае это «Интегрирующее»). Т.к. для поля «**Номер**» мы ранее указали «**Auto_Increment**», то теперь его можно оставить пустым, оно будет заполнено автоматически. Можно заполнить сразу и вторую строку («Апериодическое 1-го порядка»), после чего необходимо нажать на любую из кнопок «**Вперед**».

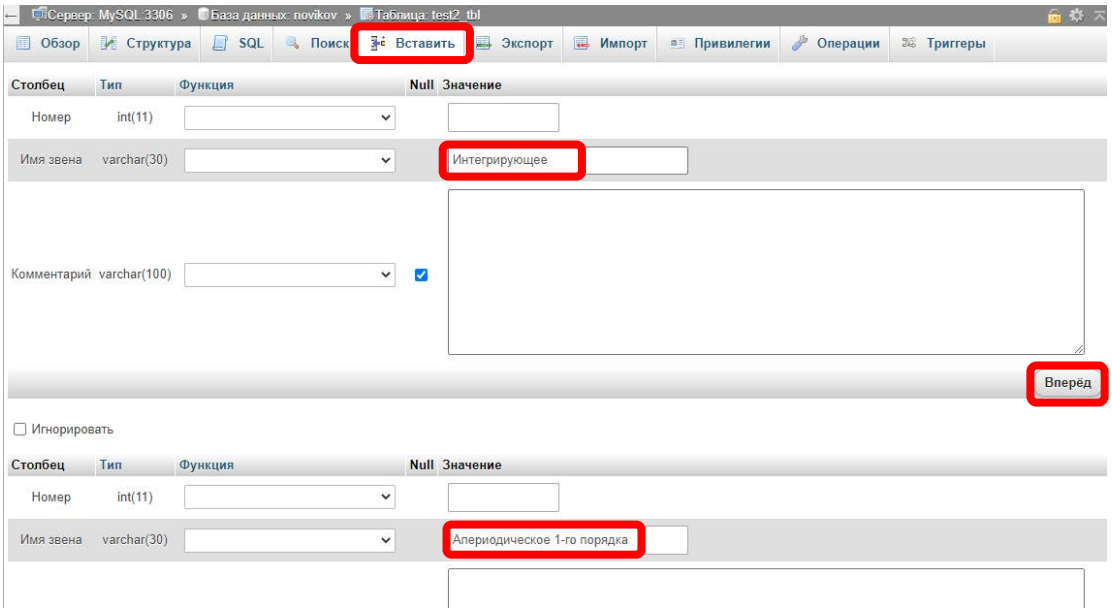

Рисунок 78 – Добавление строк в таблицу

3. Изменение, копирование и удаление строк таблицы

Для изменения, копирования или удаления строки таблицы, необходимо перейти на вкладку «**Обзор**» и нажать соответствующую кнопку в выбранной строке (рис. 79).

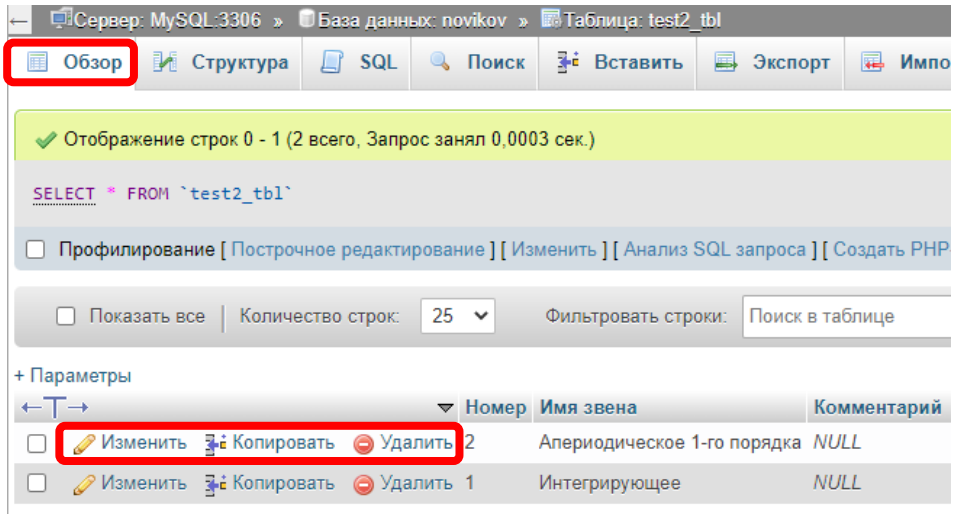

Рисунок 79 – Изменение, копирование или удаление строки таблицы

При выборе «**Изменить**» или «**Копировать**» будет открыто окно, аналогичное окну добавления строк, рассмотренному ранее (см. рис. 78).

Можно работать и с несколькими строками таблицы одновременно, для этого необходимо проставить галочки в требуемых строках и нажать одну из кнопок **под таблицей** (рис. 80).

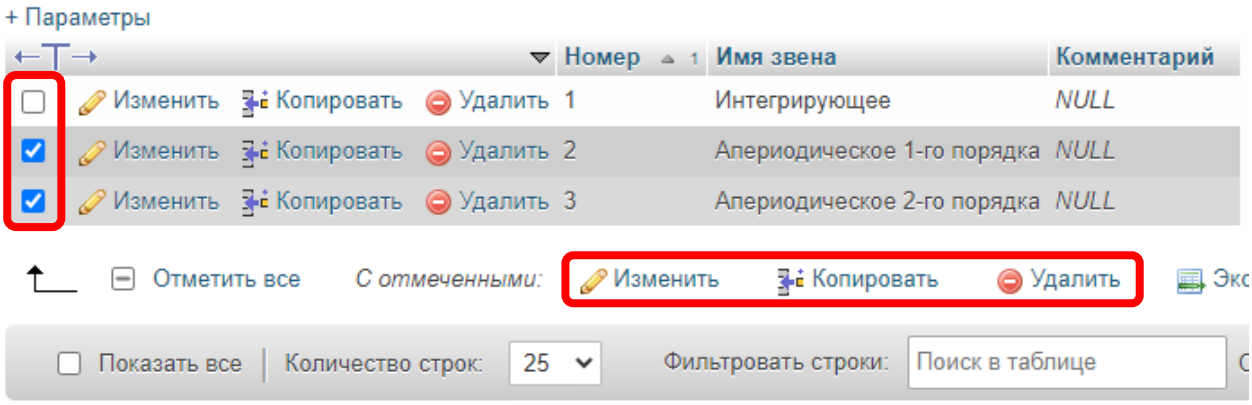

Рисунок 80 – Изменение, копирование или удаление нескольких строк таблицы

Кроме того, можно изменять значения отдельных полей прямо в режиме «**Обзор**». Для этого достаточно дважды кликнуть по соответствующему полю (рис. 81).

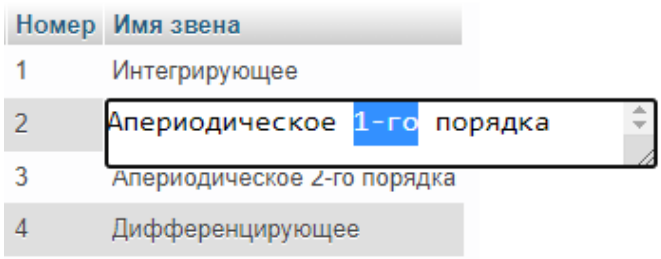

Рисунок 81 – Редактирование значения отдельного поля

4. Редактирование структуры таблицы

Помимо редактирования строк таблицы возможно редактирование и столбцов для уже созданной таблицы. Для этого необходимо перейти на вкладку «**Структура**» (рис. 82). Здесь нам будут доступны такие функции как: добавление и удаление столбцов, изменение имени и типа столбца, изменение порядка (перемещение) столбцов.

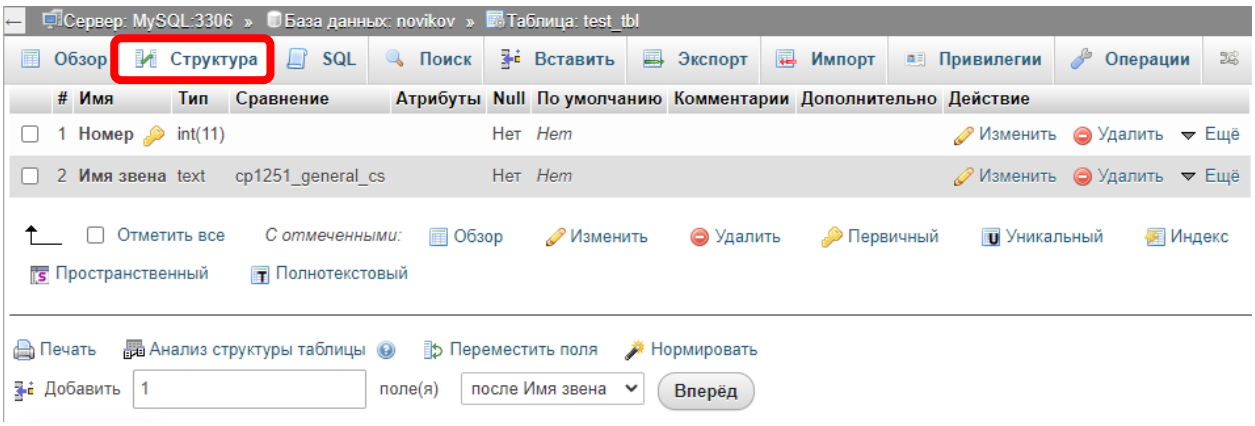

Рисунок 82 – Редактирование структуры таблицы

5. Копирование, перемещение и переименование таблицы

Копирование, перемещение и переименование таблиц осуществляется на вкладке «**Операции**». Переименование таблицы производится в разделе «**Move table**» (рис. 83), при этом нужно ввести новое имя таблицы, а имя базы данных оставить без изменения.

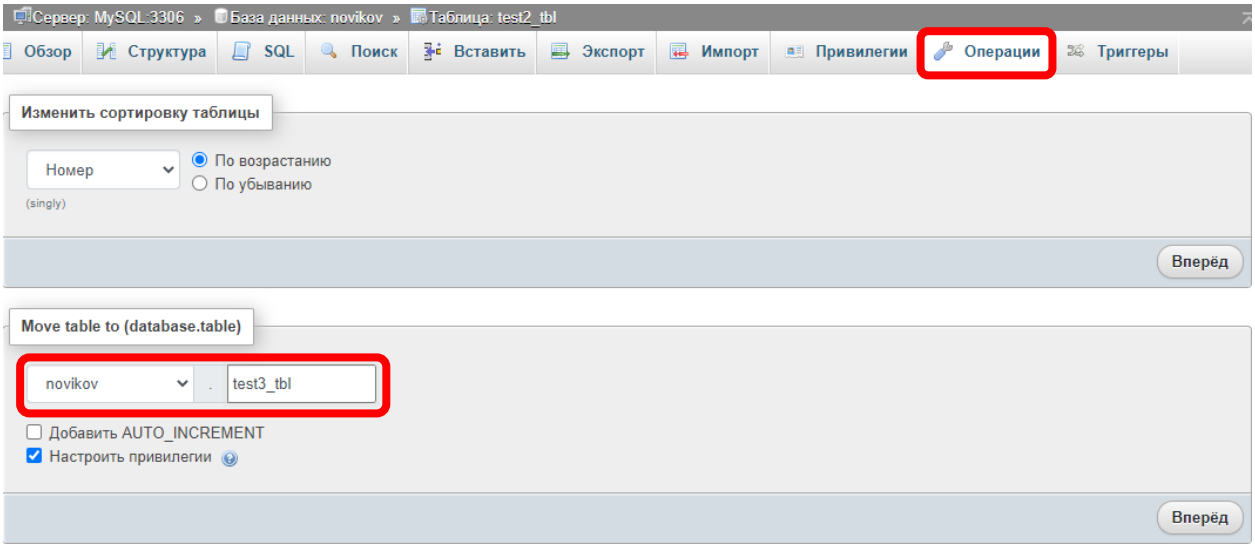

Рисунок 83 – Переименование таблицы

Перемещение в другую базу данных производится аналогично (в разделе «**Move table**»), только нужно выбрать из выпадающего списка имя базы данных, в которую производится перемещение. При это можно как задать новое имя для перемещаемой таблицы, так и оставить его прежним.

Копирование таблицы осуществляется из раздела «**Copy table**» (рис. 84). Копирование можно осуществлять как внутри этой же базы данных, так и выбрать из выпадающего списка другую базу данных. Также возможно копирование таблицы как вместе с данными (заполненными строками), так и только структуры таблицы («шапки», без заполненных строк).

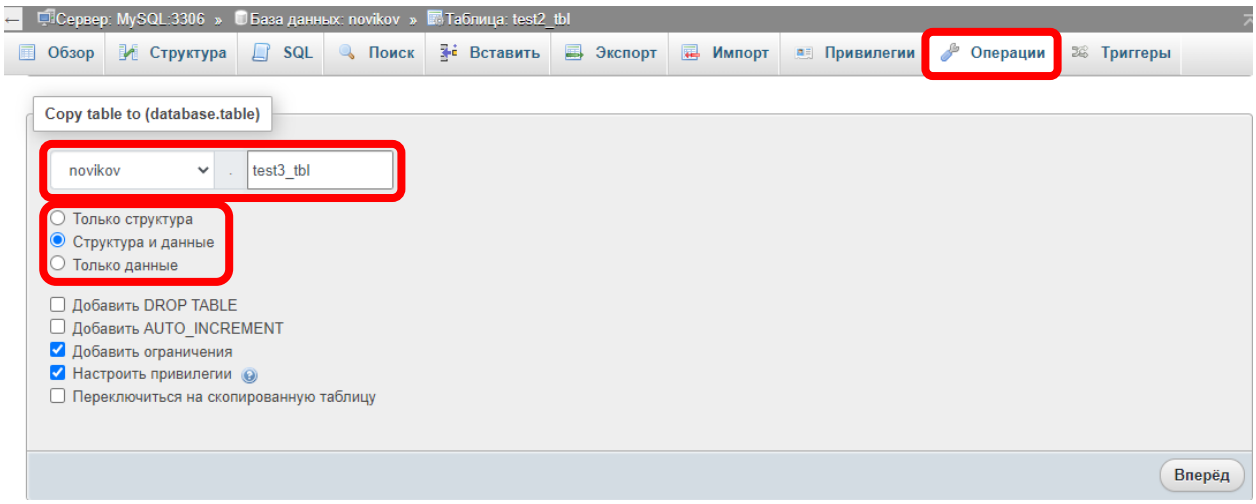

Рисунок 84 – Копирование таблицы

6. Очистка и удаление таблицы

Очистка и удаление таблицы также производится на вкладке «**Операции**». Для этого необходимо перейти в раздел «**Удалить данные и таблицу**» (рис. 85),

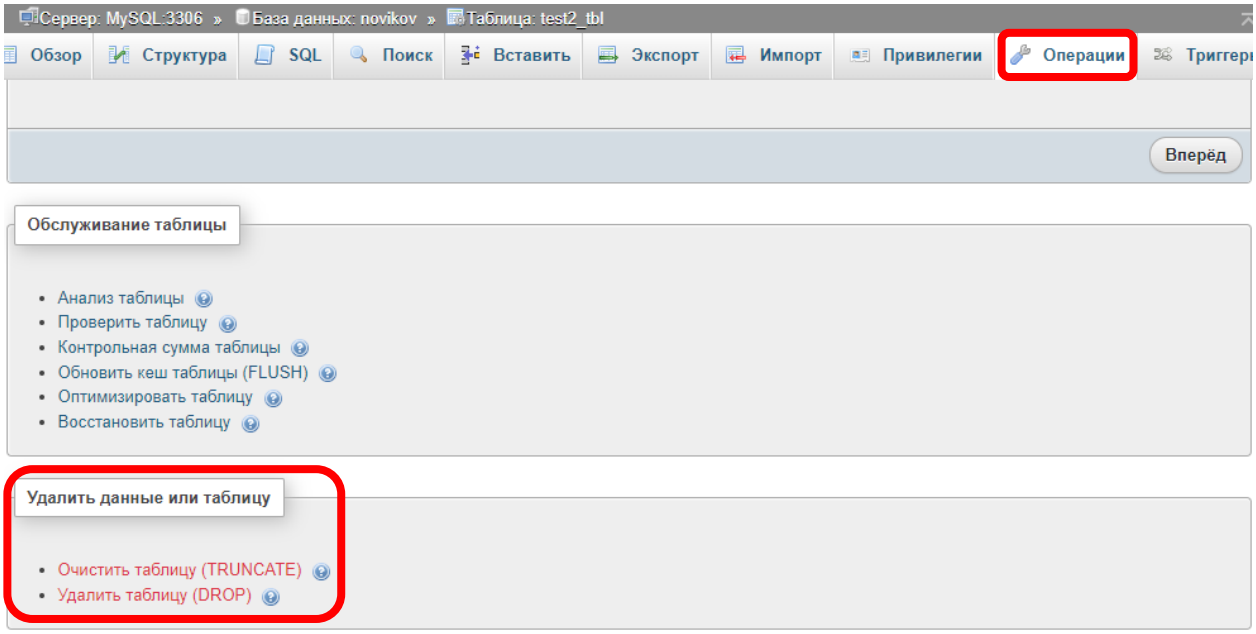

Рисунок 85 – Очистка и удаление таблицы

2.6. *Экспорт таблиц из PhpMyAdmin в Excel и «*.sql»

Бывает необходимо перенести созданные в **MySQL** базы данных на другой компьютер (сервер). Для этого необходимо экспортировать (рис. 86) базу данных в файл «***.sql**», а потом импортировать его на другом компьютере. Файлы ***.sql** – это текстовые файлы, содержащие **SQL**-запросы на создание и заполнение соответствующих таблиц. Их можно просмотреть через блокнот (рис. 87).

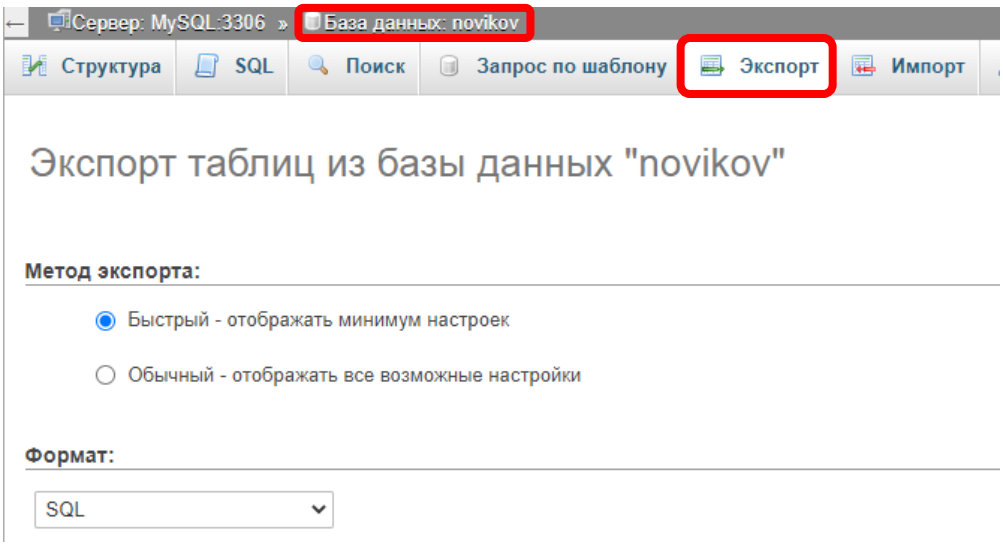

Рисунок 86 – Экспорт базы данных

```
SET SQL_MODE = "NO_AUTO_VALUE_ON_ZERO";
START TRANSACTION;
SET time_zone = "+00:00";


/*!40101 SET @OLD_CHARACTER_SET_CLIENT=@@CHARACTER_SET_CLIENT */;
/*!40101 SET @OLD_CHARACTER_SET_RESULTS=@@CHARACTER_SET_RESULTS */;
/*!40101 SET @OLD_COLLATION_CONNECTION=@@COLLATION_CONNECTION */;
/*!40101 SET NAMES utf8mb4 */;

--
-- База данных: `novikov`
--

-- --------------------------------------------------------

--
-- Структура таблицы `lab2`
--

DROP TABLE IF EXISTS `lab2`;
CREATE TABLE IF NOT EXISTS `lab2` (
  `Номер` int(11) NOT NULL AUTO_INCREMENT,
  `Наименование` varchar(100) COLLATE cp1251_general_cs NOT NULL,
  `Тип` varchar(100) COLLATE cp1251_general_cs NOT NULL,
  `Изображение` varchar(200) COLLATE cp1251_general_cs NOT NULL,
```

Рисунок 87 – Пример файла *.sql

Для переноса содержимого **SQL**-таблицы в **Excel** требуется:

1) через вкладку «**Обзор**» поместить содержимое выбранной таблицы в **буфер обмена** (рис. 88);

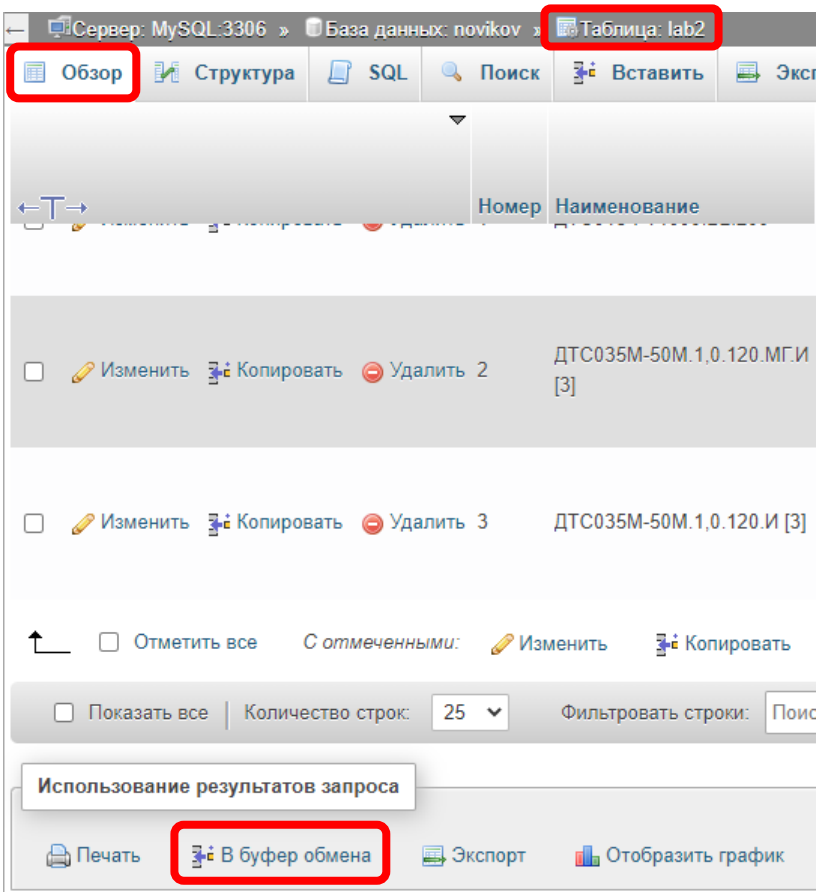

Рисунок 88 – Копирование таблицы в буфер обмена

2) на новом **Excel**-листе выделить все ячейки (Ctrl+A) и изменить их формат на «Текстовый» (рис. 89);

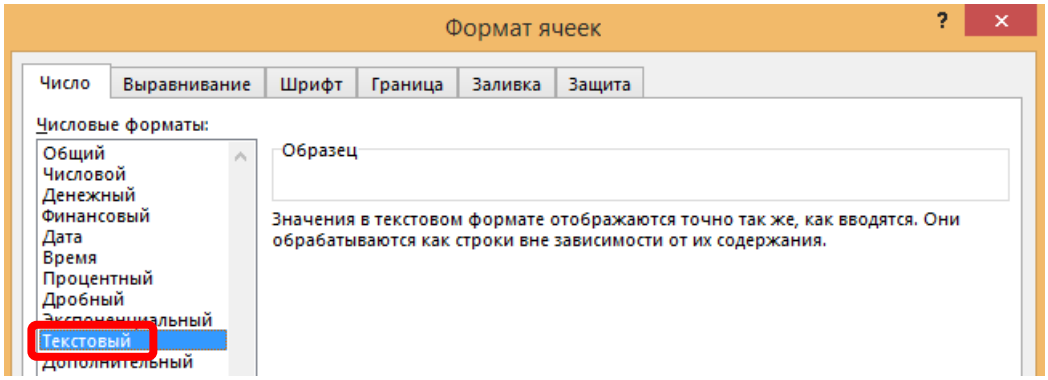

Рисунок 89 – Формат ячеек в Excel

3) вставить на лист содержимое буфера обмена (Ctrl+V);
4) *изменение формата ячеек требовалось для правильной вставки числовых значений, например, значений «рейтинга», которые записаны как числа через точку, но для* ***Excel*** *требуется* ***запятая****, поэтому необходимо* выделить ячейки с числовыми значениями и нажать Ctrl+F, где перейдя на вкладку «**Заменить**» (рис. 90), произвести замену точек на запятые;

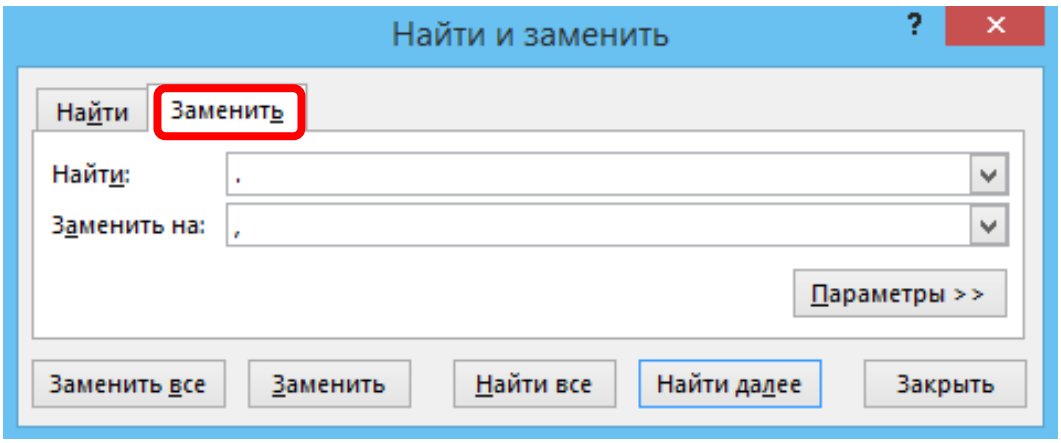

Рисунок 90 – Замена точек на запятые

5) далее вновь выделить все ячейки (Ctrl+A) и изменить их формат обратно на «Общий».

Пример итоговой таблицы **Excel**, представлен на рис. 91.

	A	B	C	D	E	F	G	H	I
1	MySQL:3306/novikov		http://localhost/phpmyadmin/index.php?route=/sql&db=novikov&tab						
2									
3	Отображение строк 0 - 2 (3 всего, Запрос занял 0,0004 сек.)								
4									
5									
6	SELECT * FROM `lab2`								
7									
8									
9	Номер	Наименов	Тип	Изображе	Рейтинг	Производ	Страна	Гарантия	Цена
10	1	ДТС015-Р	Термоме	dts015.pn	4,7	Овен	Россия	24	2490
11	2	ДТС035М-	Термоме	dts035m.p	4,9	Овен	Россия	24	7998
12	3	ДТС035М-	Термоме	dts035m.p	4,9	Овен	Россия	24	7398

Рисунок 91 – Итоговая таблица в Excel

Таким способом можно перенести в **Excel** не только таблицу целиком, но и результаты выполнения отдельных запросов.

2.7. *Просмотр истории запросов

Для того чтобы увидеть историю выполненных ранее запросов, необходимо нажать кнопку «Консоль» внизу экрана (рис. 92). Далее можно выбрать требуемый запрос из списка и повторить его.

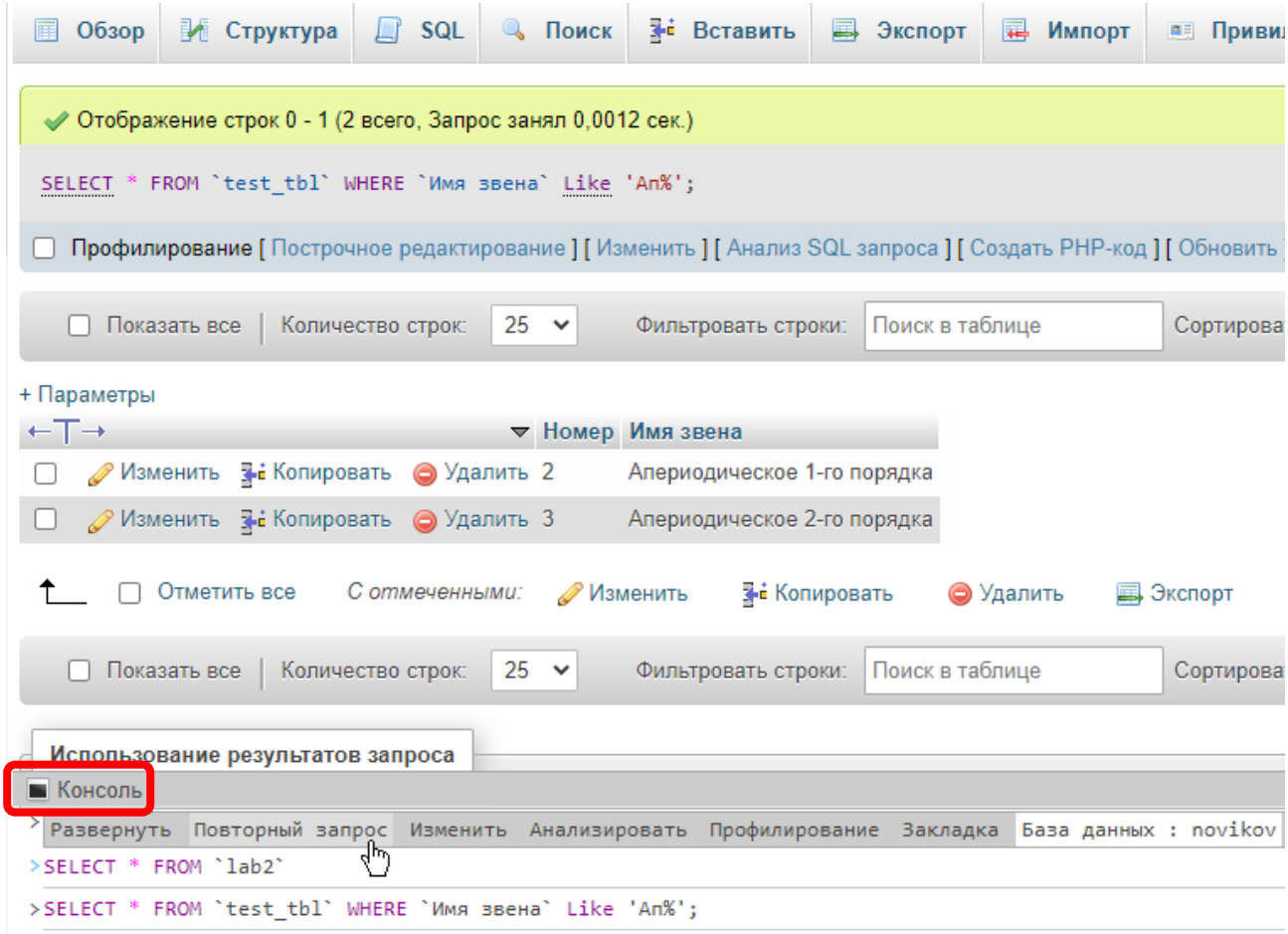

Рисунок 92 – Консоль в PhpMyAdmin

2.8. *Представления (View) в PhpMyAdmin

Подробнее про Представления рассказывалось в разделе **1.9**. Здесь же лишь демонстрируется работа с ними в MySQL через PhpMyAdmin.

Создание представления

Повторим запрос, который ранее уже рассматривался в разделе **2.3**. Теперь для создания Представления по этому запросу, необходимо нажать «Создать представление» (рис. 93).

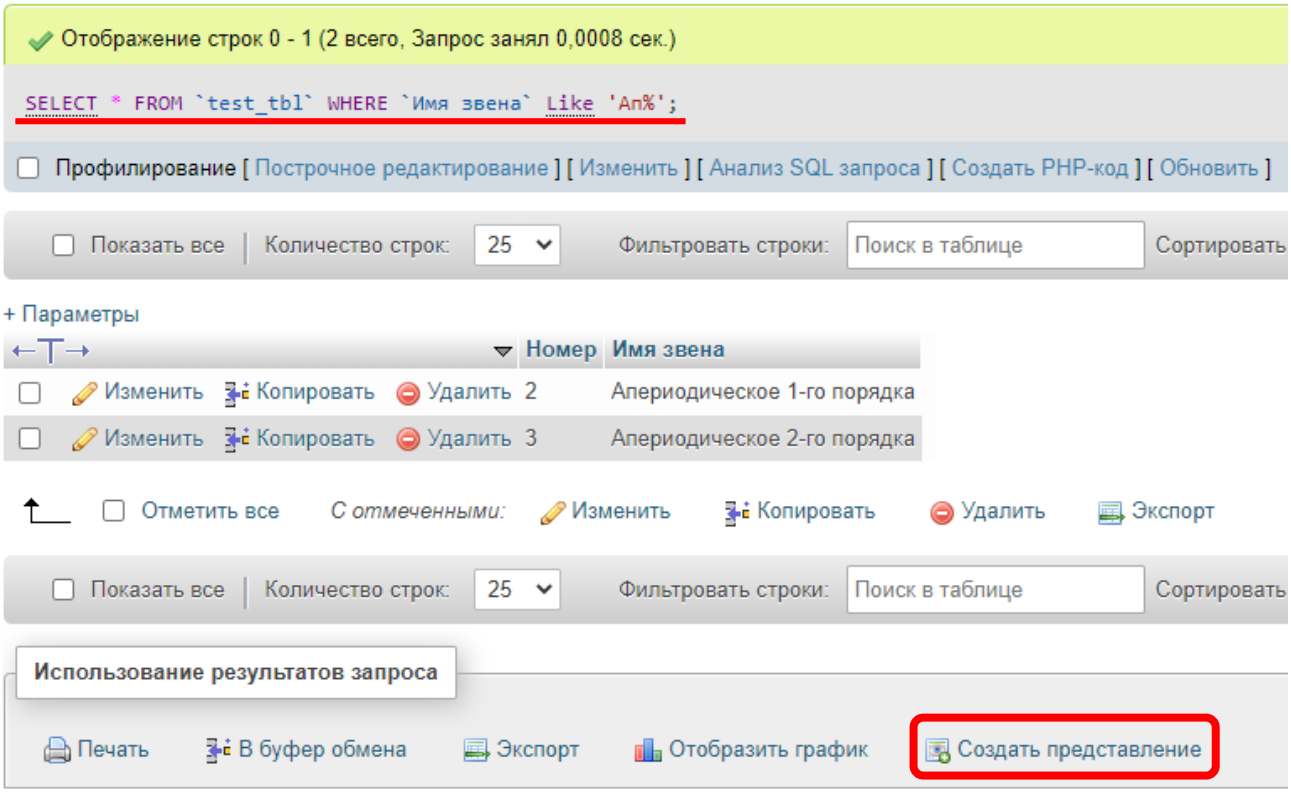

Рисунок 93 – Создание представления из запроса

Далее достаточно написать только имя для нашей «виртуальной таблицы», а SQL-запрос будет подставлен автоматически (рис. 94).

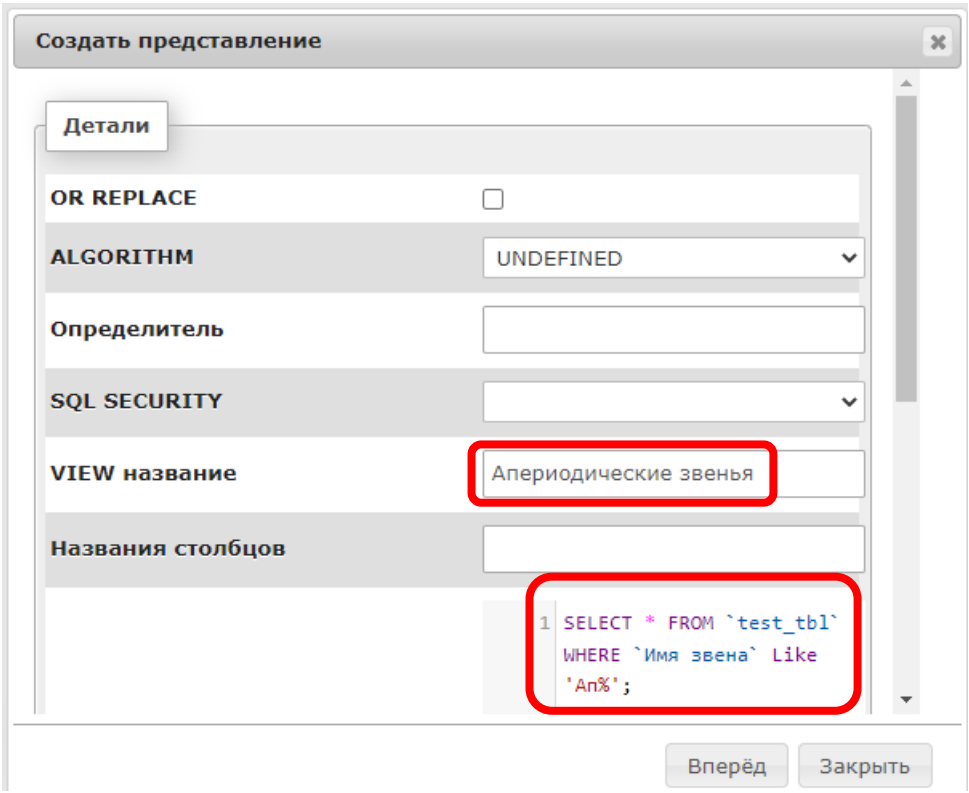

Рисунок 94 – Создание представления

Добавленные представления отображаются в списке над таблицами (рис. 95). *Если список не отображается, необходимо нажать кнопку обновления наверху.*

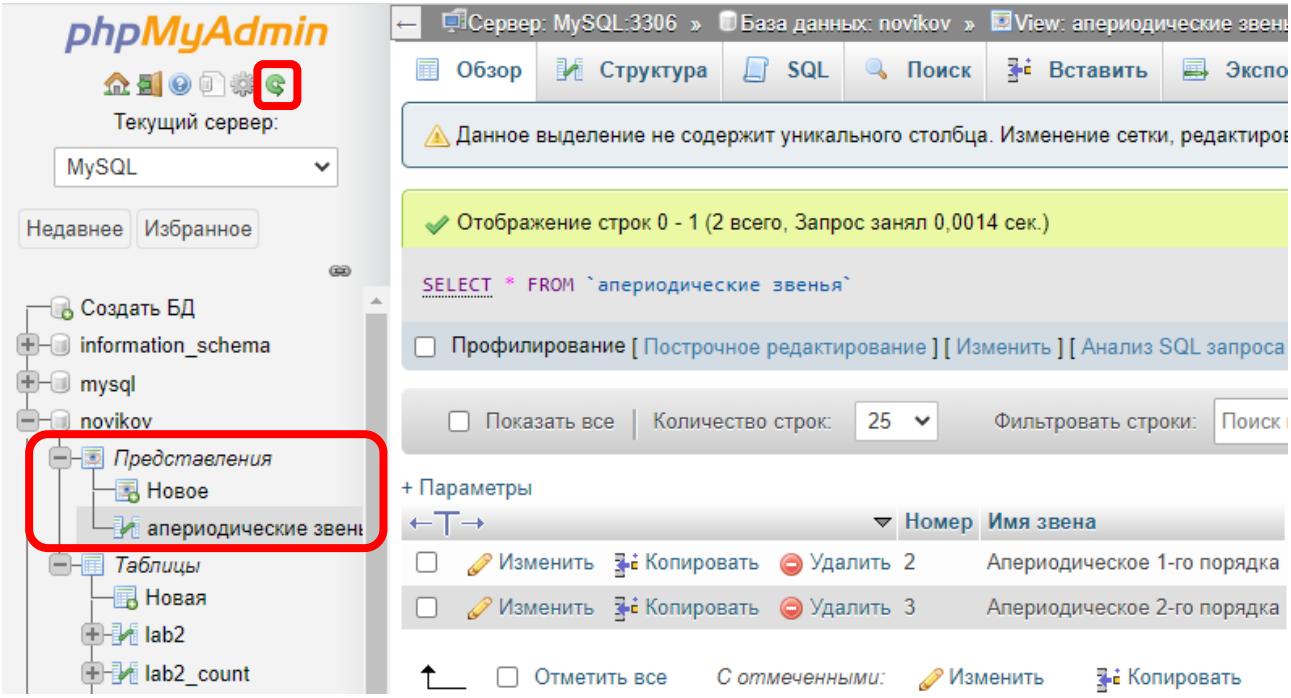

Рисунок 95 – Список представлений

Просмотр и редактирование представлений

Для того чтобы увидеть SQL-запрос, связанный с выбранным представлением, необходимо перейти на вкладку «Структура» и далее нажать «Редактировать представление» (рис. 96).

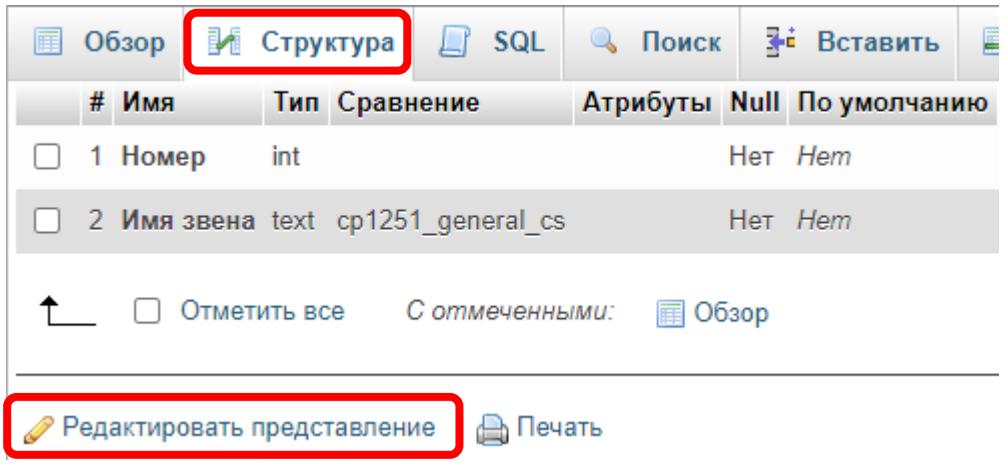

Рисунок 96 – Редактировать представление

При этом код запроса будет отображен не совсем в том же виде, как мы его вводили (рис. 97). Здесь же можно поменять текст запроса.

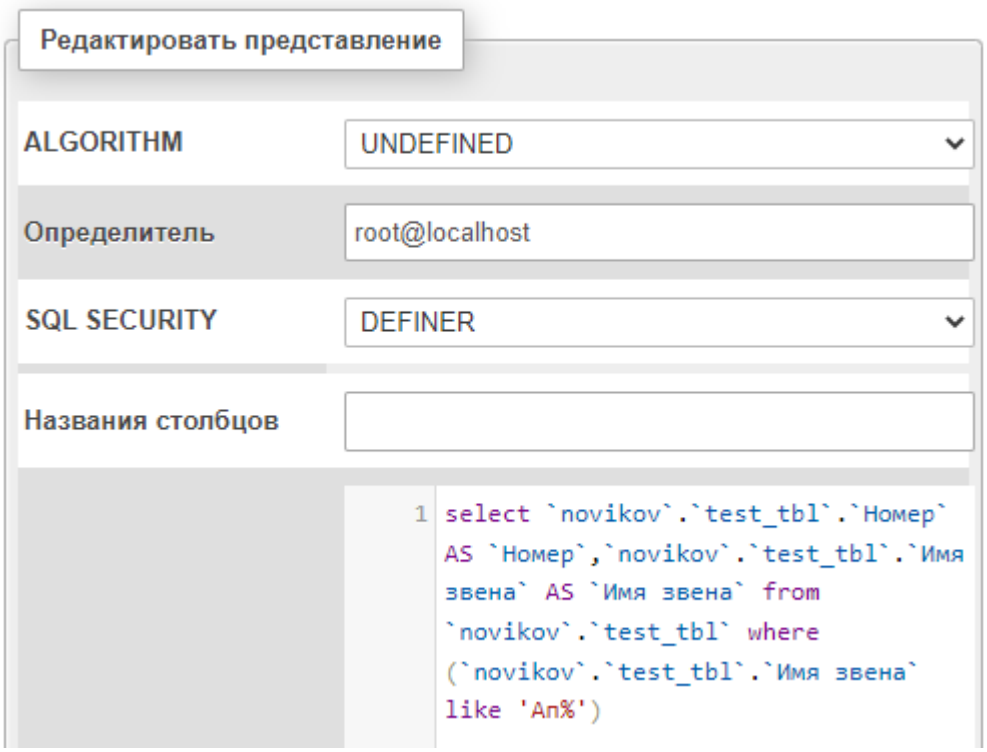

Рисунок 97 – Редактирование представления

Удаление представления

Для удаления представления необходимо выбрать заголовок раздела «Представления», после чего нажать кнопку «Удалить» для требуемого представления (рис. 98).

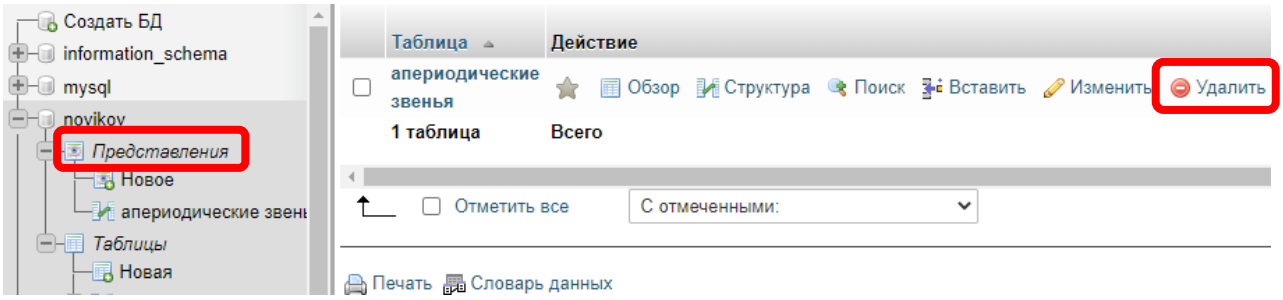

Рисунок 98 – Удаление представления

2.9. **Создание собственных функций в PhpMyAdmin

Подробнее про создание Функций рассказывалось в разделе **1.7**. Здесь же лишь демонстрируется работа с ними в MySQL через PhpMyAdmin.

Создание функции

Создание функции можно осуществить через обычный SQL-запрос, с той лишь разницей, что необходимо **стереть** разделитель внизу окна (рис. 99). *Разделитель можно и не стирать, но тогда необходимо использовать команду DELIMITER (см. раздел **1.7**).*

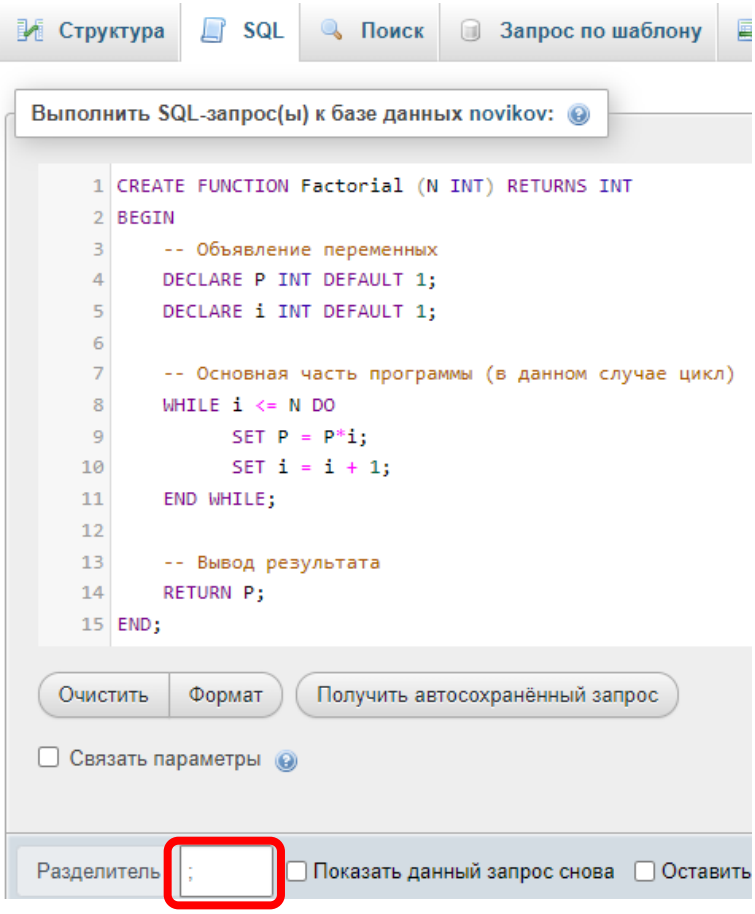

Рисунок 99 – Создание функции

Добавленная функция отобразится в списке под таблицами (рис. 100). *Если список не отображается, необходимо нажать кнопку обновления наверху.*

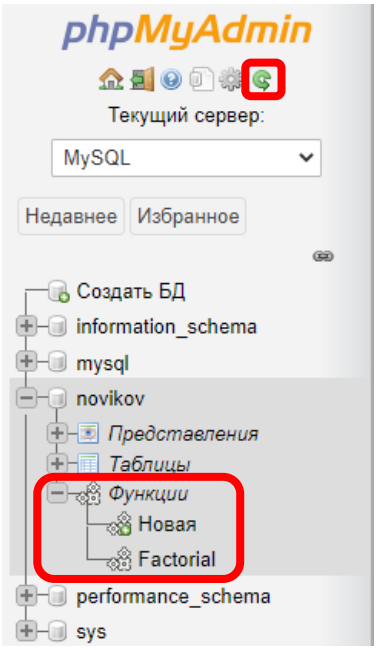

Рисунок 100 – Список функций

Просмотр и редактирование функции

Для просмотра и редактирования функции, достаточно выбрать ее имя в списке (см. рис. 100), после чего откроется окно редактирования (рис. 101).

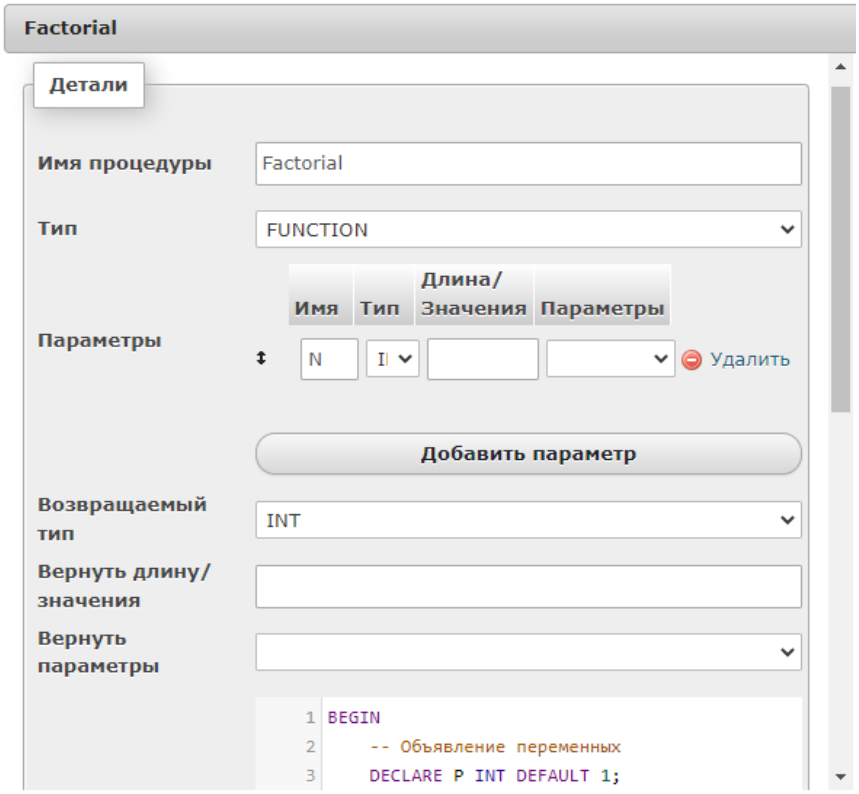

Рисунок 101 – Редактирование функции

Использование функции

Для проверки работы созданной функции достаточно ввести запрос SELECT (рис. 102). *Функции можно использовать и в более сложных запросах, в тех же местах, где возможно использовать операторы (такие как, например, сложение или объединения строк).*

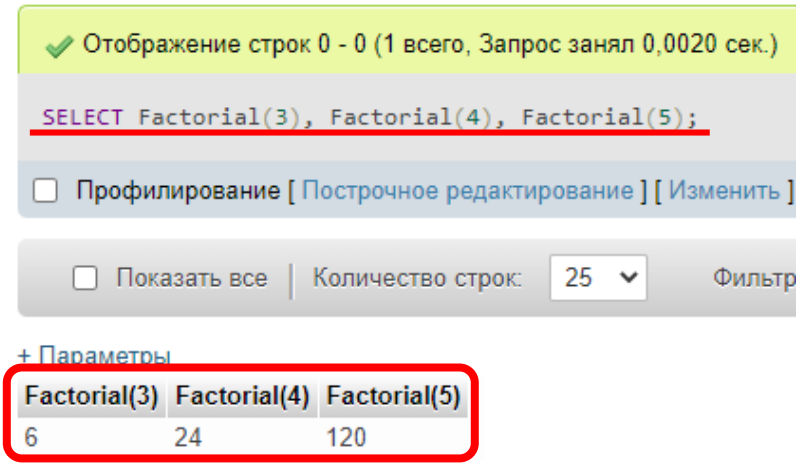

Рисунок 102 – Тестирование созданной функции

Удаление функции

Для удаления функции необходимо выбрать заголовок раздела «Функции», после чего нажать кнопку «Удалить» для требуемой функции (рис. 103).

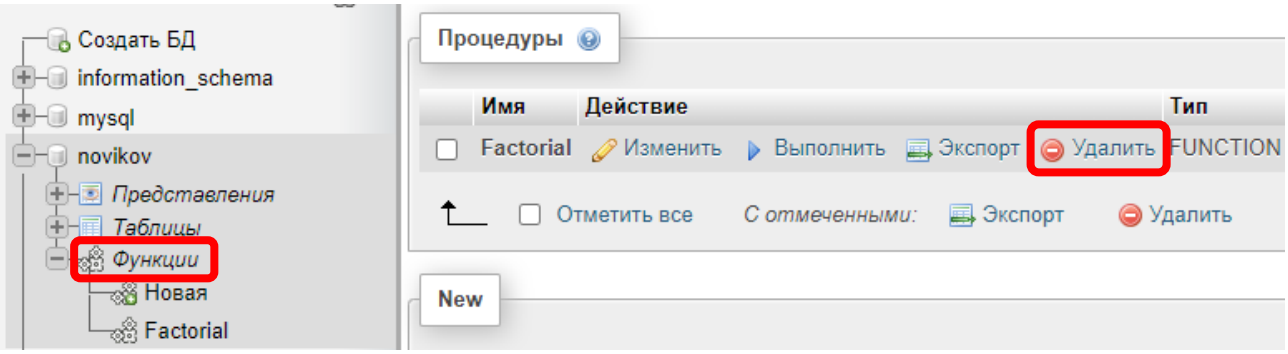

Рисунок 103 – Удаление функции

3. ОСНОВЫ ЯЗЫКА HTML

HTML (читается как «Эйчь-Ти-Эм-Эл» или «Эйчь-Ти-Эм-Эль») – язык разметки интернет-страниц (они же «Web-страницы», или «HTML-страницы»). Базовый пример кода на языке HTML выглядит следующим образом:

```
<html>
  <head> <title>СУБД - Новиков</title> </head>
  <body>
     <b>Жирный текст</b><br>
     <i>Курсив</i>
  </body>
</html>
```

Язык состоит из «тегов», которые могут быть парными, т.е. иметь закрывающийся тег (как например и), и непарными (как например
).

!!! В студенческих работах написание имен тегов маленькими[**14**] буквами обязательно!

Заголовок страницы, отображаемый как название соответствующей закладки в браузере, указывается между тегами **<title>** и **</title>**.

!!! В студенческих работах необходимо указывать в качестве заголовка название предмета, фамилию студента и текущий год!

Целью данного курса не является изучение всей структуры языка HTML, поэтому далее мы будем работать только с телом интернет-страницы, расположенным между тегами **<body>** и **</body>**.

Примеры основных тегов языка HTML:

1. выделение текста **жирным**, *курсивом* или подчеркиванием:
   ```
   <b>Жирный текст</b>, <i>Курсив </i>,
   <b><u>Жирный подчеркнутый</u></b>
   ```
2. переход на новую строку:
   ```
   <br>
   ```
3. гиперссылка:
   ```
   <a href="http://gturp.spb.ru">Ссылка на сайт</a>
   ```
4. вставка изображения:
   ```
   <img src="Img/logo_green.png" align="right">
   ```
5. размер и цвет шрифта:
   ```
   <font size="7" color="red">Большой красный текст</font>
   <font color="#FF00FF">Розовый текст</font>
   ```
   Размер шрифта задается цифрой от 1 до 7. Цвет можно задать по имени (на английском) или по цветовым координатам в формате RRGGBB, где RR – красный (red), GG – зеленый (green), BB – синий (blue). Каждый цвет задается в диапазоне от 00 до FF (в 16-ричной системе).
6. вставка кнопк.и:
   ```
   <button>Нажми меня!</button>
   ```

3.1. Создание Web-страницы

Для создания Web-страницы необходимо:

1. Создать (рис. 104) на рабочем столе текстовый файл («**Текстовый документ**»).

!!! После окончания работы необходимо удалять за собой файлы с рабочего стола!

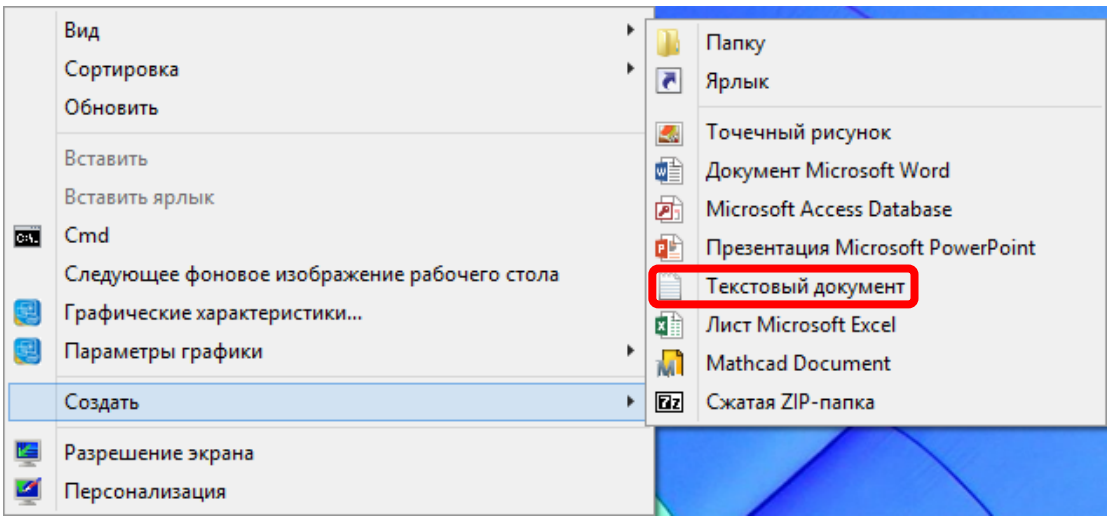

Рисунок 104 – Создание текстового документа

2. Открыть созданный документ в блокноте и напечатать в нем код HTML-страницы (базовый пример кода, рассмотренный ранее).
3. Сохранить напечатанный код и закрыть блокнот.
4. Переименовать данный файл в «**Demo.html**».
5. Открыть файл с помощью браузера (рис. 105).

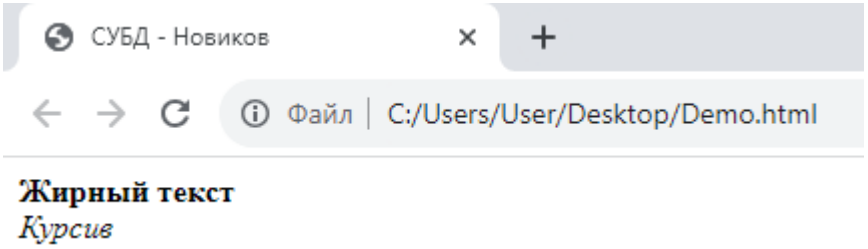

Рисунок 105 – Пример Demo.html, открытый в браузере

Операционная система Windows позволяет открывать файл двойным кликом мышкой. При этом файл открывается только в какой-либо одной программе. В зависимости от настроек системы, это могут быть разные программы. Если требуется открывать один файл из разных программ (как в нашем случае из блокнота, и из браузера), то необходимо использовать меню «Открыть с помощью» (рис. 106).

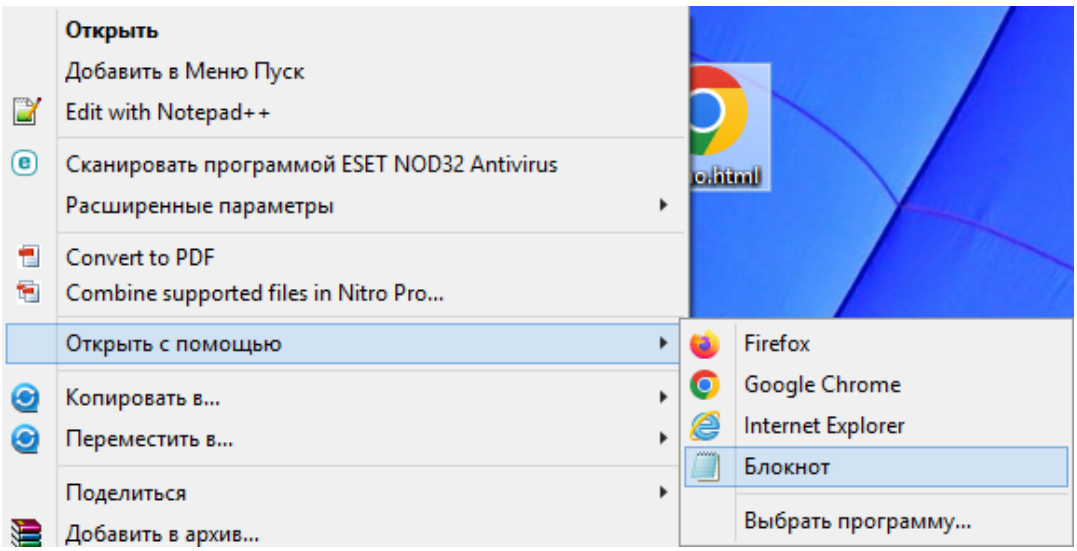

Рисунок 106 – Меню «Открыть с помощью»

В Windows может быть отключено отображение расширений файлов (рис. 107). В этом случае будет невозможно правильно переименовать файл, сменив расширение файла (с *.txt на *.html).

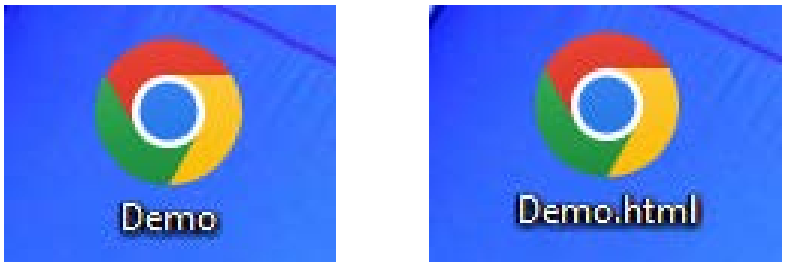

Рисунок 107 – Отображение файла без расширения и с расширением

Для решения этой проблемы нужно воспользоваться одним из двух способов:

1. Перенастроить Windows на отображение расширений файлов через «Параметры папок». Для этого в любой папке необходимо открыть меню «**Вид**» (рис. 108) и нажать кнопку «**Параметры**». Далее в появившемся окне «**Параметры папок**» (рис. 109), необходимо **снять** галочку «**Скрывать расширения для зарегистрированных типов файлов**».
2. Либо открыть требуемый файл в блокноте и, воспользовавшись функцией «Сохранить как», задать имя файла вместе с правильным расширением.

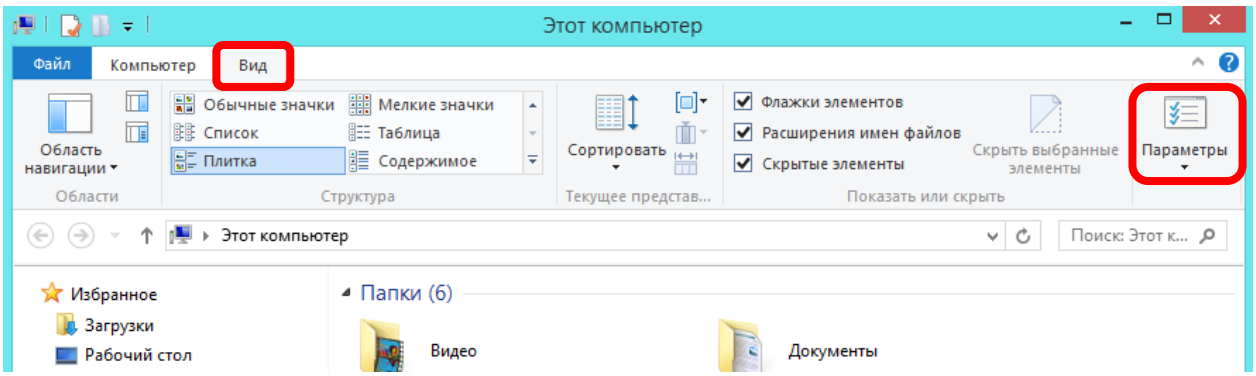

Рисунок 108 – Вызов «Параметры папок»

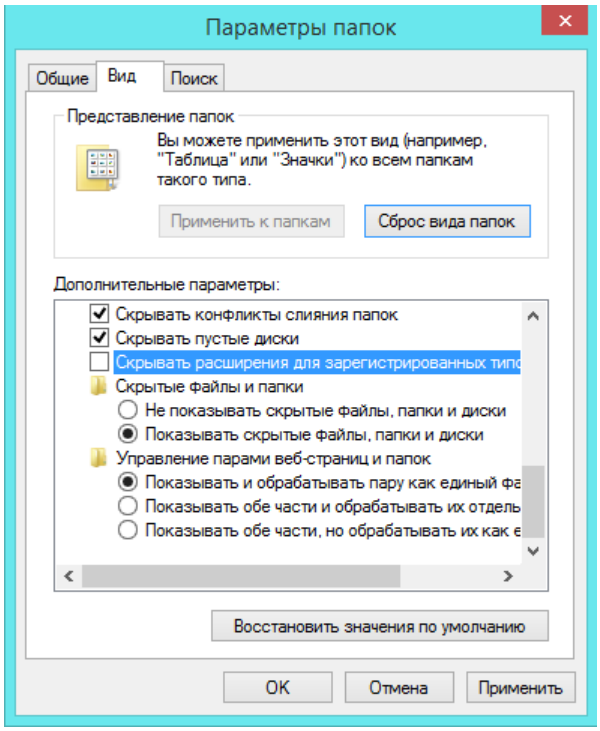

Рисунок 109 – Настройка отображения расширения файла

3.2. РАБОТА № 3 (часть 1 из 3)

Необходимо разработать **HTML**-страницу для одного из датчиков (устройств, приборов), который использовался в предыдущей Работе (см. раздел **2.4**). Страница должна содержать все параметры, использованные ранее в таблицах, в том числе: наименование датчика, его описание и список характеристик, рейтинг товара, информацию о цене, гарантии и наличие устройства, количество магазинов (складов) где это устройство имеется, список этих магазинов (складов) и их адреса. Кроме того, **HTML**-страница должна содержать изображение описываемого устройства и кнопку «Купить» (или «Заказать»), или несколько кнопок «Купить», по одной для каждого из магазинов.

Для достижения поставленной цели необходимо:

1. переименовать ранее созданный файл «Demo.html» (или его копию) в «**Lab3.html**»;
2. создать на рабочем столе папку «**Img**» для размещения в ней изображений;
3. найти в Интернете изображение требуемого датчика и поместить его в папку «**Img**»;
4. задать для этого изображения понятное имя, например, «**dts015.png**» (т.к. в моем случае используется датчик «Овен ДТС-015»). *В дальнейшем в теге <img>, адрес этого изображения будет «**src="Img/dts015.png"**»;*
5. открыть в блокноте файл «**Lab3.html**» и, редактируя его между тегами **<body>** и **</body>**, используя при этом текст и теги <b>, <i>, <u>,
, <a>, <img>, <font> и <button>, добиться требуемого результата.

!!! Можно использовать и другие теги, если автор работы в состоянии объяснить их назначение и принцип работы.

Для более удобного редактирования **HTML**-документов, необходимо использовать блокнот с подсветкой синтаксиса, например, **Notepad++**.

Пример итоговой **HTML**-страницы представлен на рис. 110.

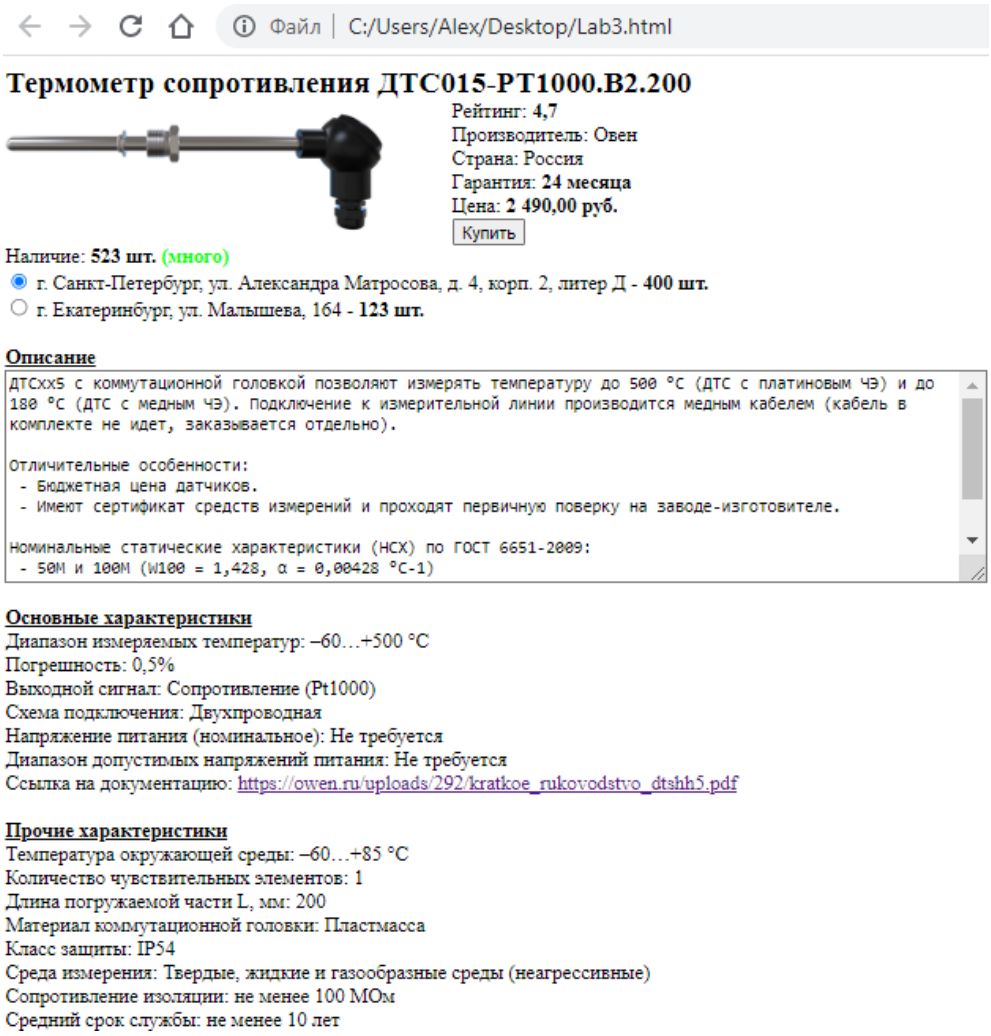

Рисунок 110 – Пример итоговой HTML-страницы

3.3. *Блочная структура HTML-документа

В приведенном выше примере (см. рис. 110) была рассмотрена **HTML**-страница, в которой рисунок и текст разбиты на две колонки. Добиться такого результата можно, окружив соответствующие часть **HTML**-документа тегами ** ** и ** **. Хотя рамки для этих элементов и не отображаются, продемонстрируем их пунктирными линиями на рис. 111.

Рисунок 111 – Блочная структура документа

Положение блоков задается через стиль элемента, также в нем можно указать и ширину блока. Например, можно написать следующий **HTML**-код:

```
<div style="display: inline-block; width: 350;">
  <img width="300" src="Img/dts015.png">
</div>
<div style="display: inline-block;">
  Рейтинг: <b>4,7</b>
  <br>Производитель: Овен
  <br>Страна: Россия
  <br>...
</div>
```

4. APACHE-СЕРВЕР И ЯЗЫК PHP

Apache HTTP-сервер (произносится «**апа́ч**») – программное обеспечение, позволяющее создать HTTP-сервер (он же Web-сервер) для запуска страниц на языке HTML. Клиентом для подключения к такому серверу, является Web-браузер.

Сервер Apache и язык HTML позволяют запускать статические страницы. Для запуска динамических страниц совместно с ними используется язык **PHP** (читается как «Пи-Аш-Пи» или «Пи-Эйчь-Пи»). Фактически **PHP** является «препроцессором» для **HTML**, т.е. осуществляет его обработку прежде, чем отправить его клиенту. Динамизация страниц бывает двух видов: на стороне сервера и на стороне клиента. **PHP** отвечает за динамизацию на сервере. Для динамизации на стороне клиента используются **JavaScript** и **JSON** *(о которых здесь рассказываться не будет)*. Возможен и смешанный вариант создания динамической страницы, которая вначале обрабатывается на стороне сервера, а потом на стороне клиента.

WampServer (**WAMP**-сервер) – сборка веб-сервера, содержащая Apache, MySQL, интерпретатор скриптов PHP, phpMyAdmin и другие дополнения, предназначенная для web-разработки под Windows. Имеет автоматический инсталлятор. Для управления сервером и его настройками WampServer создает иконку в трее (у часов). Позволяет выбирать различные версии Apache, MySQL, MariaDB и PHP. Название расшифровывается как:
WAMP = **W**indows + **A**pache + **M**ySQL + **P**HP.

4.1. Установка WAMP-сервера

Для установки **WAMP**-сервера необходимо вначале скачать дистрибутив программы с официального сайта[15]:
https://www.**wampserver**.com/

При написании данного пособия использовалась версия **Wampserver 3.2.6 x64**, включающая следующие версии программ:

— Apache 2.4.51;
— PHP 5.6.40/7.4.26/8.0.13/8.1.0;
— MySQL 5.7.36|8.0.27;
— MariaDB 10.5.13|10.6.5;
— PhpMyAdmin 4.9.7 & 5.1.1.

Также рекомендуется установить блокнот с подсветкой синтаксиса языков программирования, например **Notepad++**[16], который можно скачать по адресу:
https://**notepad-plus-plus**.org/downloads/

Установка **WAMP**-сервера в основном производится с настройками по умолчанию. При этом стоит обратить внимание, что установка должна производиться в папку «**C:\wamp64**» (рис. 112), т.е. только в корень диска (нельзя устанавливать данную программу в «C:\Program Files …»!). Путь не должен содержать пробелов и других специальных знаков, а также русских букв. Установка возможна только **от имени Администратора**.

!!! Нельзя устанавливать **WAMP**-сервер поверх имеющейся версии! Если ранее в данную папку уже был установлен **WAMP**-сервер, то необходимо вначале полностью удалить предыдущую версию при помощи стандартной функции Windows «Удалить или изменить программу», а при необходимости (если папка осталась) также удалить папку «C:\wamp64» вручную.

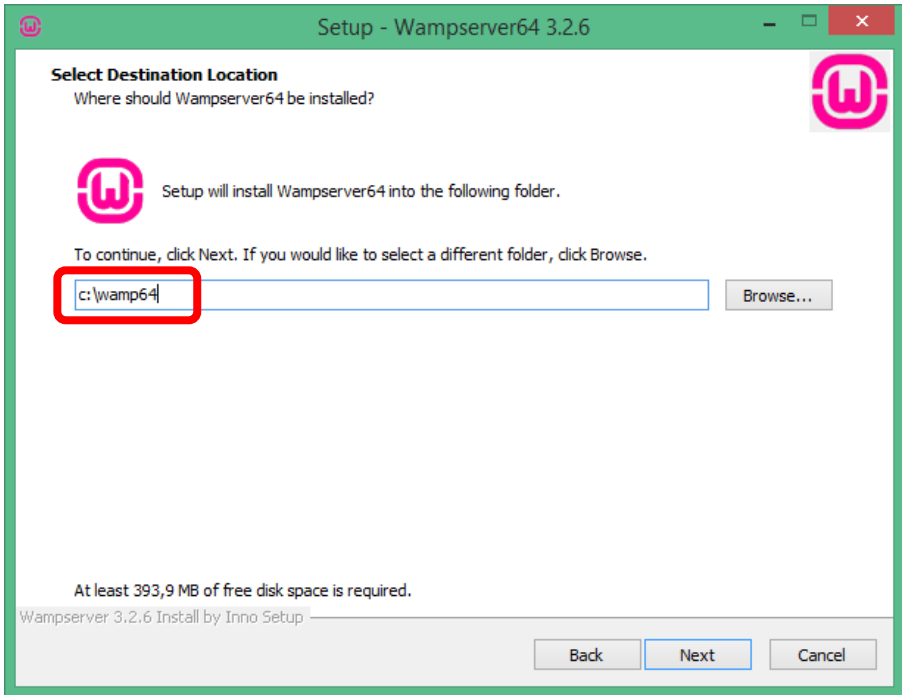

Рисунок 112 – Выбор папки для установки

При установке на **Windows 8** (и более ранние версии), необходимо выбрать для установки **MySQL версии 5**. При установке на **Windows 10** и более новые версии, необходимо выбрать для установки **MySQL версии 8** (рис. 113).

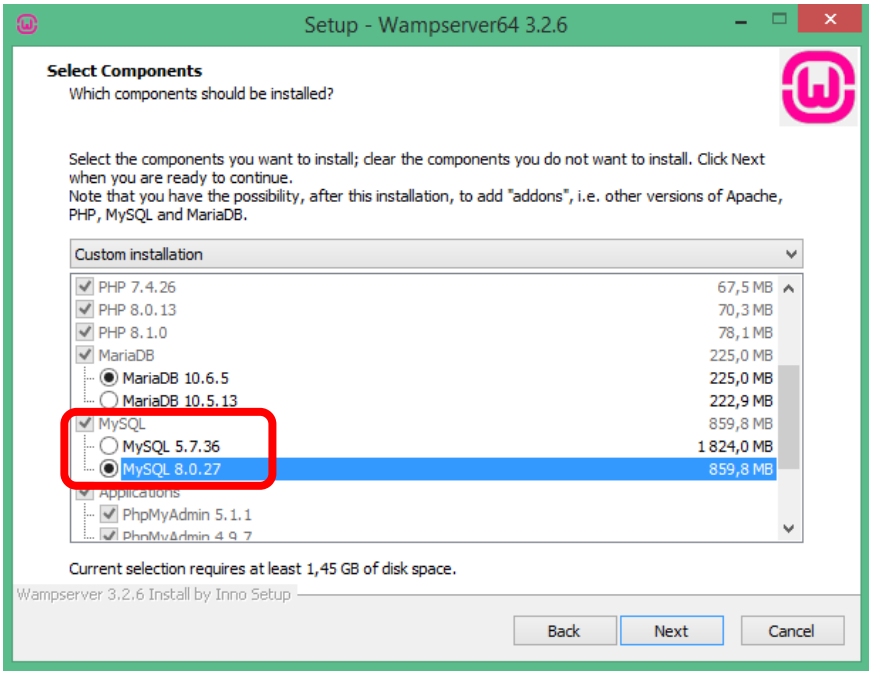

Рисунок 113 – Выбор версии MySQL для установки

После установки необходимо:

1. Запустить появившийся на рабочем столе ярлык «**Wampserver64**» (рис. 114).

Рисунок 114 – Ярлык WAMP-сервера на Рабочем столе

2. Щелкнуть **правой** кнопкой мыши по соответствующему значку у часов и переключить «Язык» («**Language**») на «**russian**» (рис. 115). *Сервер стартует не мгновенно, может потребоваться подождать загрузки сервера до 1 минуты.* После запуска значок у часов должен стать **зеленым**.

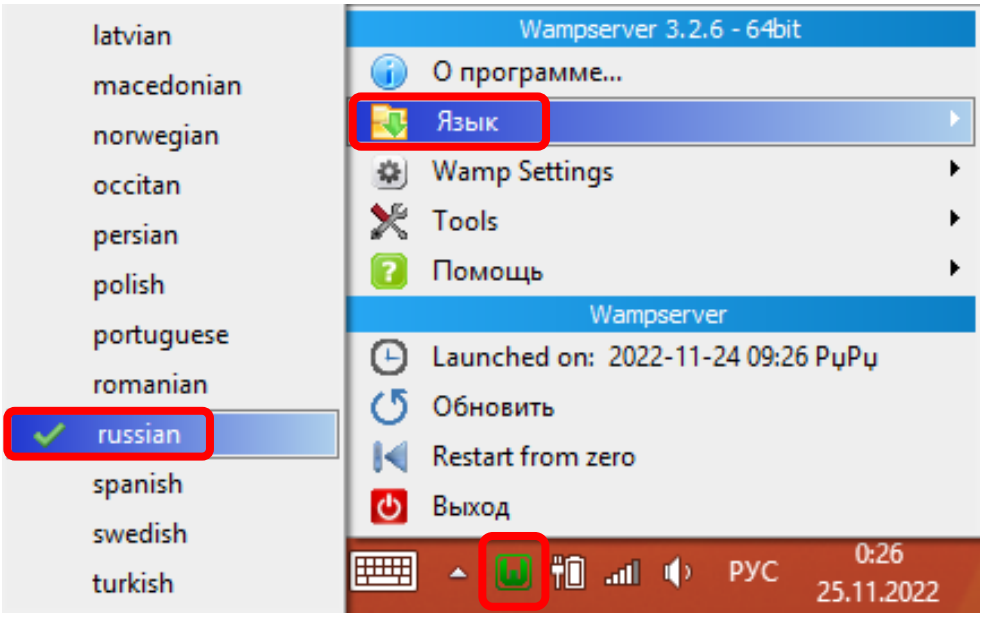

Рисунок 115 – Выбор языка

!!! Если установка прошла успешно, но WAMP-сервер не запускается (значок не становится зеленым), то см. раздел «4.2. *Возможные проблемы при установке».

3. Проверить работу сервера, запустив браузер и введя адрес **localhost** (или, что равносильно, адрес **127.0.0.1**). Далее выбрать «**PhpMyAdmin**» (рис. 116).

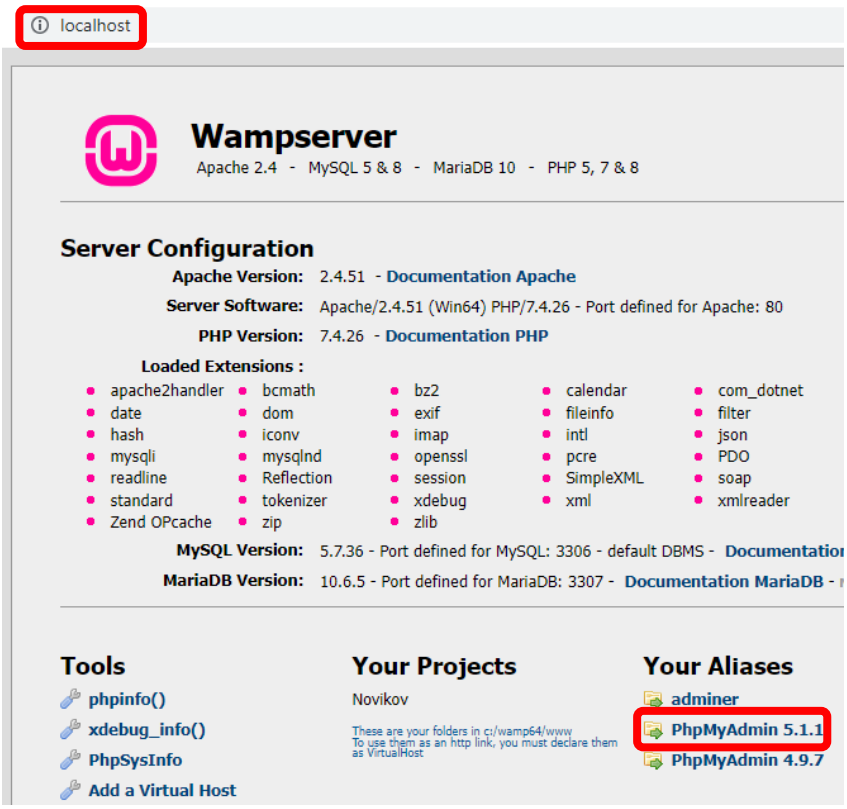

Рисунок 116 – Подключение к WAMP-серверу через браузер

4. Ввести имя пользователя **root** (без пароля) и подключиться к серверу **MySQL** (рис. 117).

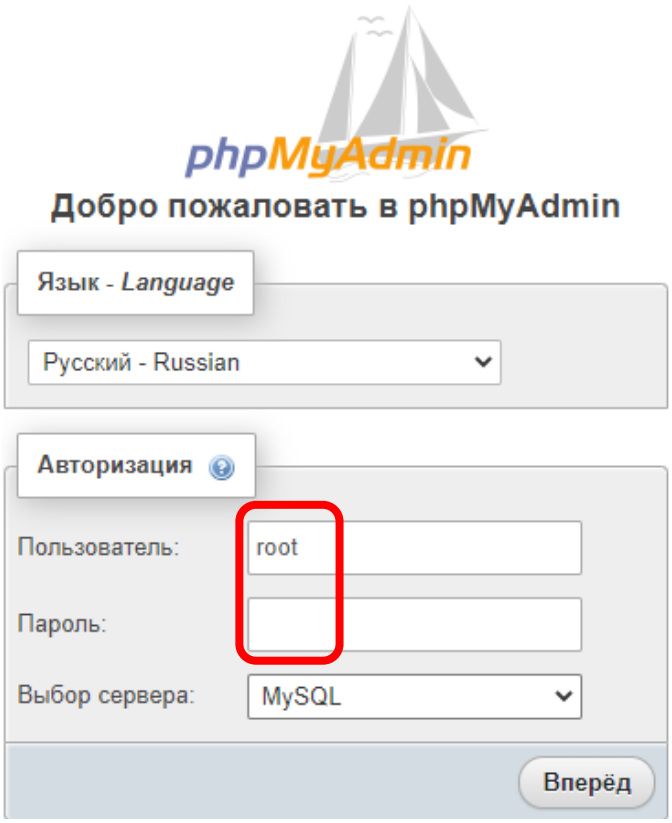

Рисунок 117 – Подключение к серверу MySQL

4.2. *Возможные проблемы при установке

Как правило проблемы запуска WAMP-сервера связаны с тем, что порт 80 уже занят.

Отключение IIS Windows

По умолчанию в Windows может быть включен «**IIS Windows**» (**I**nternet **I**nformation **S**ervices), тогда он будет перехватывать вход по адресу **localhost** (он же **127.0.0.1**). Тогда в браузере вместо страницы **WAMP**-сервера отобразится страница **IIS Windows** (рис. 118).

Для отключения **IIS Windows** необходимо:

1) зайти в «Установку и удаление программ» → «**Включение или отключение компонентов Windows**» (рис. 119);
2) отключить «**Службы IIS**» (рис. 120);
3) если в браузере все еще видна страница **IIS Windows**, нажать в нем **Ctrl+F5** (обновить).

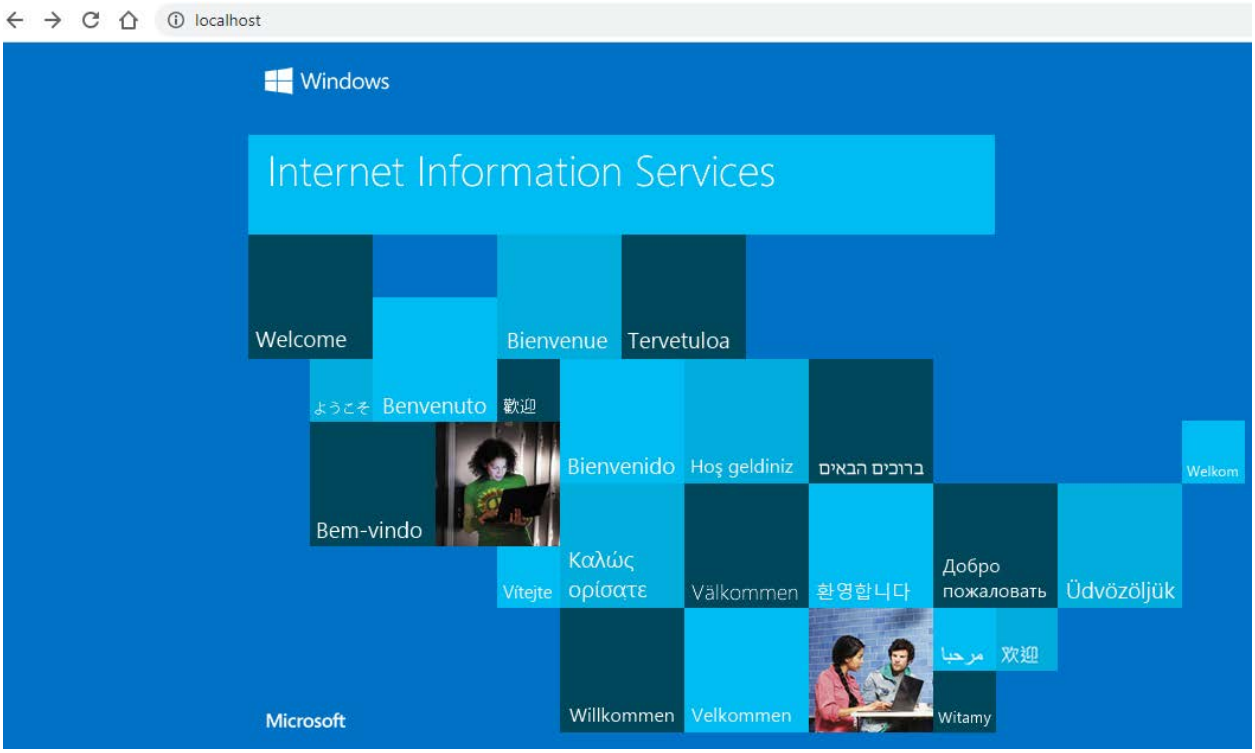

Рисунок 118 – Главная страница Windows IIS

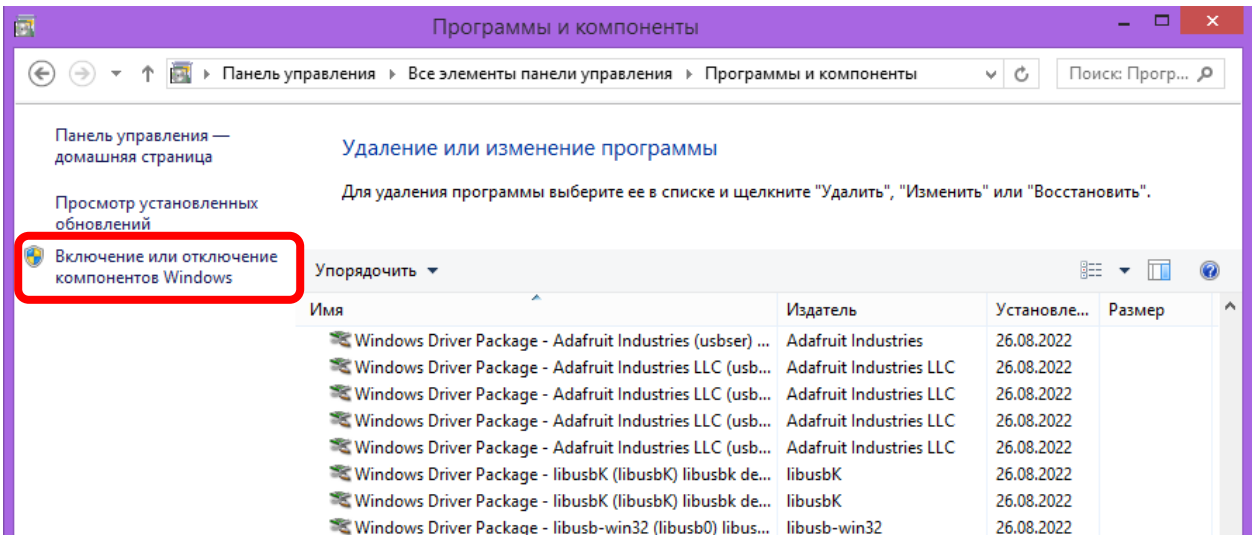

Рисунок 119 – Включение и отключение компонентов Windows

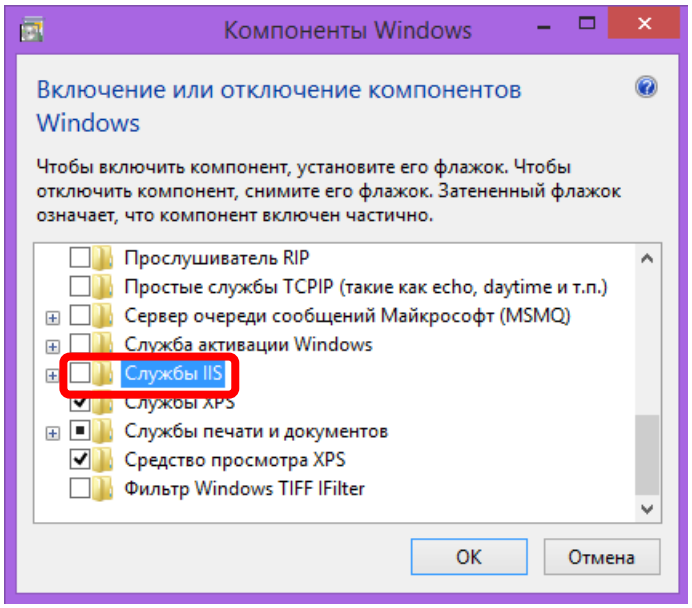

Рисунок 120 – Отключение служб IIC

Проверка занятости порта 80

Кроме **IIC**, перехватывать вход по адресу **localhost** могут и другие программы (например, **Skype**). Команда **netstat** позволяет проверить отсутствие других программ на порту **80**. Для ее запуска необходимо нажать комбинацию клавиш **Win+R** (рис. 121) и ввести «**cmd**». В появившемся окне ввести (рис. 122):

```
netstat -a
```

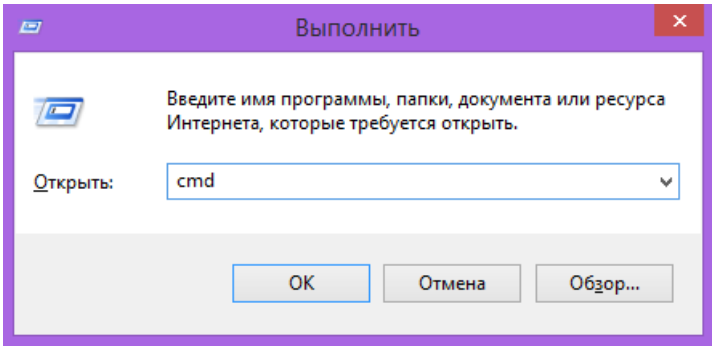

Рисунок 121 – Вызов «cmd»

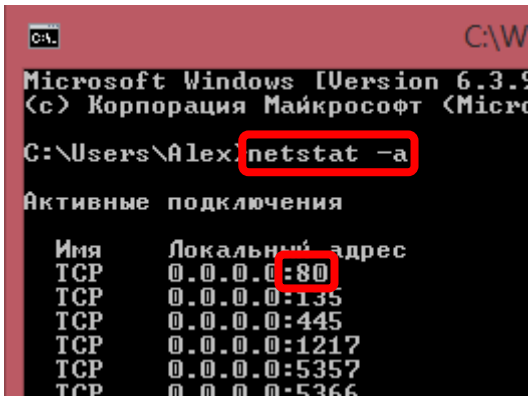

Рисунок 122 – Результат работы команды netstat

Поиск и отключение программ, занимающих порт 80

Если мы знаем, что какая-то из программ занимает порт 80, но не знаем какая именно, то необходимо ввести команду:

netstat -a -o

В этом случае мы сможем увидеть **PID** процесса (рис. 123), который занял порт.

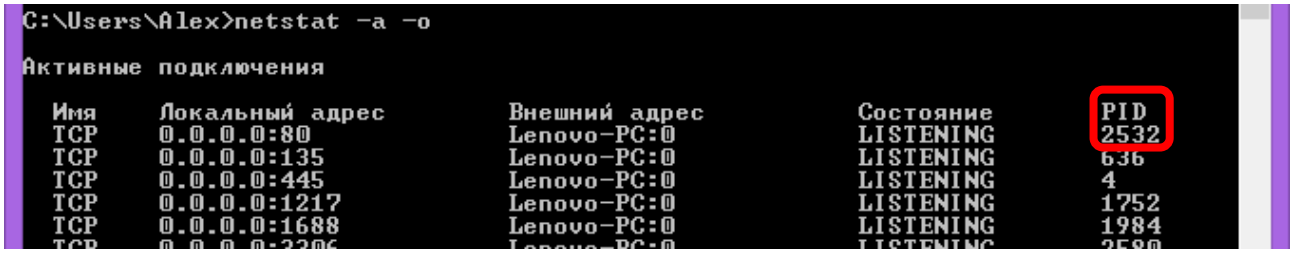

Рисунок 123 – PID процесса, занявшего порт 80

Далее запускаем **Диспетчер задач** (Ctrl+Shift+Esc) и находим процесс с данным идентификатором (рис. 124).

Далее у нас есть два варианта:

— завершить этот процесс, нажав кнопку «Снять задачу»;

— либо «Открыть расположение файла» (рис. 125), для того чтобы понять, что это за программа и можно ли ее удалить.

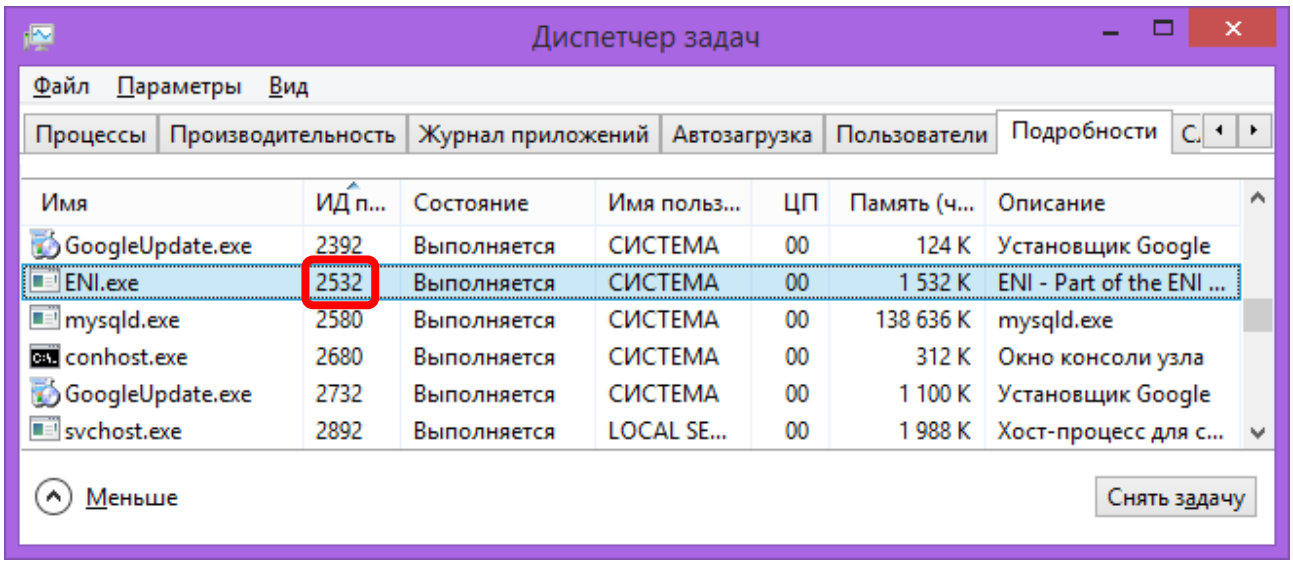

Рисунок 124 – Поиск процесса по идентификатору

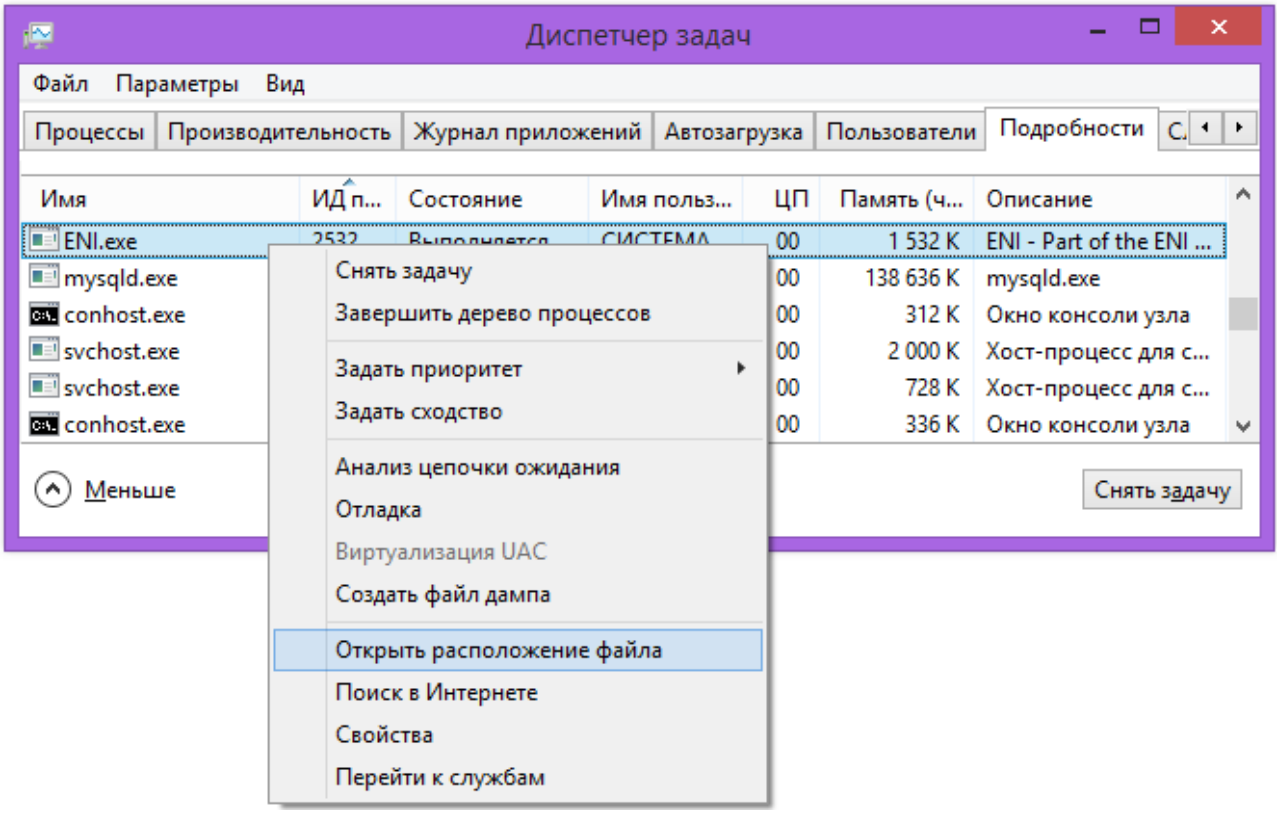

Рисунок 125 – Открыть расположение файла

****Смена порта**

Кроме того, если порт 80 занят, то можно изменить порт, который будет использовать WAMP-сервер (обычно в таком случае применяется порт 8080).

Для этого необходимо нажать **правую** кнопку мышки на значке WAMP-сервера и далее выбрать «**Tools**» → «**Use a port other than 80**» (рис. 126).

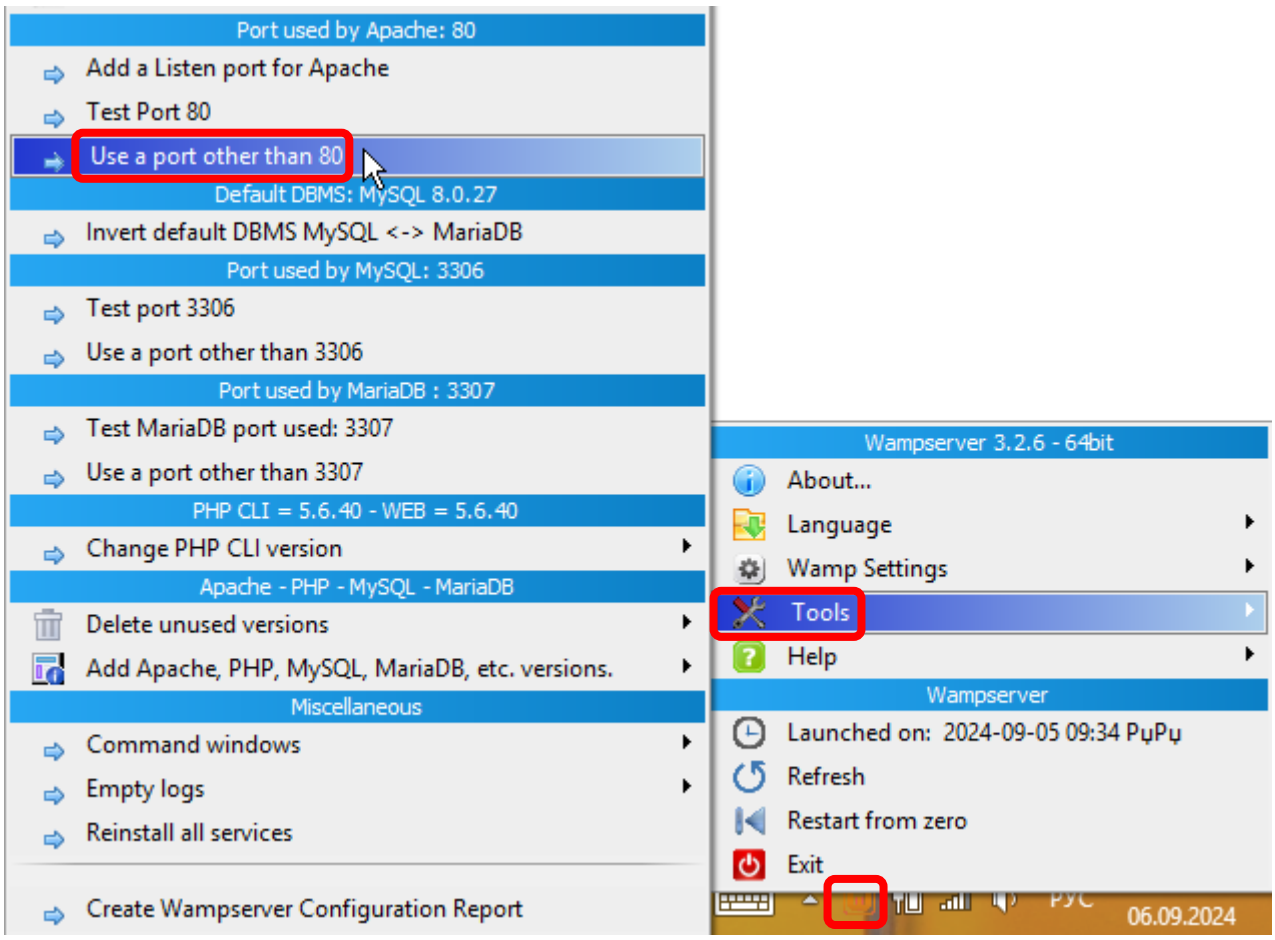

Рисунок 126 – Изменение номера порта

Вводим новый номер порта (рис. 127).

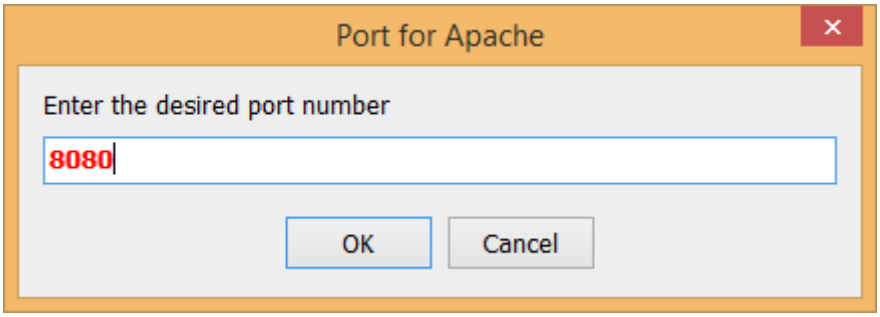

Рисунок 127 – Ввод нового номера порта

!!! Теперь для входа на сервер, вместо адреса «localhost» необходимо использовать адрес «**localhost:8080**» (рис. 128).

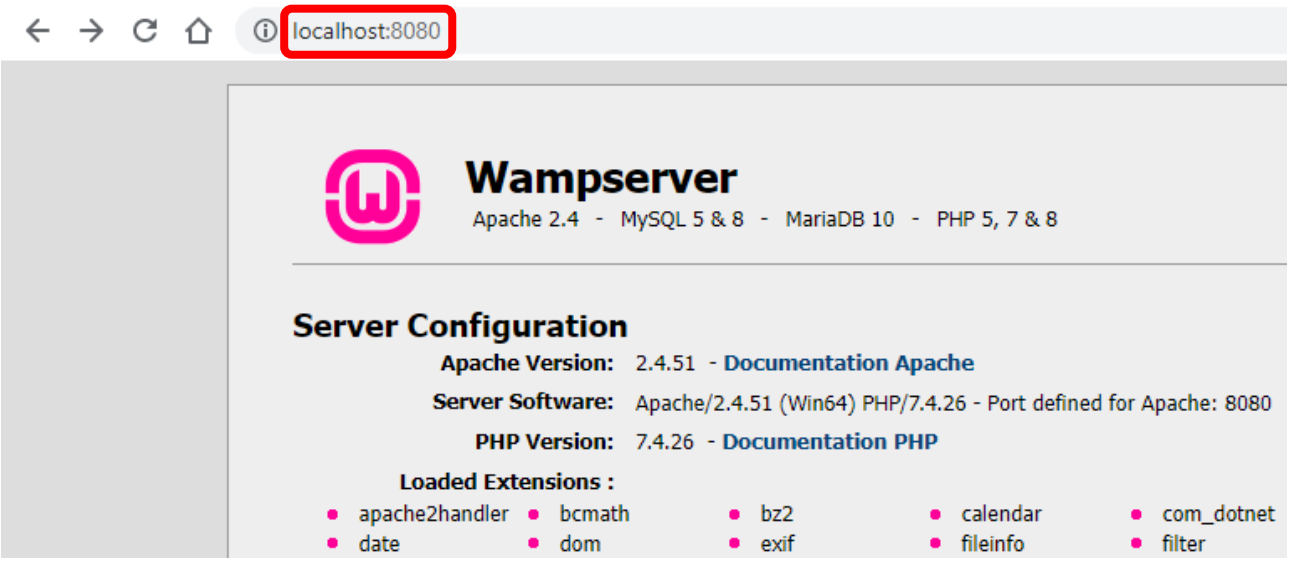

Рисунок 128 – Адрес с номером порта

4.3. Подготовка к работе

Перед началом работы необходимо проделать следующее:

1) зайти в папку «**C:\wamp64\www**» (это папка Web-сервера, отображаемая при вводе в браузере адреса «**localhost**»);
2) в папке «**www**» создать папку со своей фамилией и текущим годом (необходимо использовать только латинские буквы, а вместо пробела использовать знак подчеркивания «_»).
   *В моих примерах это будет только фамилия «**Novikov**». Адрес (для работы с этой папкой в браузере) будет, соответственно, «**localhost/Novikov**».*
   !!! Допускается изменять и удалять файлы только внутри этой своей папки!
3) внутри своей папки создать еще две папки «**Admin**» и «**Img**»;
4) переместить созданные ранее (в **Главе 3**) файлы «**Demo.html**» и/или «**Lab3.html**», а также папку «**Img**» с изображениями (если имеется), в папку со своей фамилией;
5) открыть скопированные **HTML**-файлы из браузера. Их адреса будут, соответственно, «**localhost/Novikov/Demo.html**» и «**localhost/Novikov/Lab3.html**».

!!! Если после открытия страницы в браузере вместо русских букв отображается нечитаемый текст («битые символы»), то необходимо **сменить кодировку** соответствующего HTML-документа. Для этого необходимо:

1) открыть требуемый файл в блокноте **Notepad++**;
2) выбрать меню «Кодировки» → «**Преобразовать в UTF-8 с BOM**» (рис. 129);
3) сохранить файл;
4) обновить страницу в браузере (**F5**).

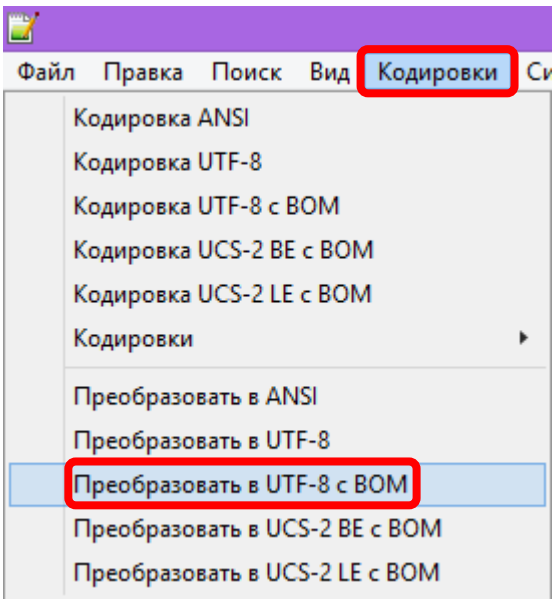

Рисунок 129 – Смена кодировки

4.4. Основы языка PHP

Основной задачей языка **PHP** является «подстановка значений» в код **HTML**, т.е. процесс частичной автоматической генерации кода (хотя возможна и полная генерация).

Рассмотрим простейший пример **HTML**-документ:

```
<html>
  <head> <title>СУБД - Новиков</title> </head>
  <body>
      Дата обращения: 17.10.2022<br>
      Время обращения: 13:50:43
  </body>
</html>
```

В данном случае, страница просто отображает (рис. 130) две заранее прописанные строчки текста. Но невозможно постоянно вручную менять содержимое страницы, поэтому необходимо, чтобы сервер сам подставлял текущие дату и время. Для этого и существует **PHP**.

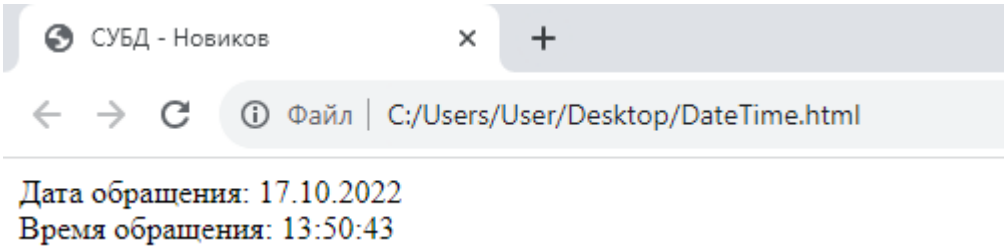

Рисунок 130 – Пример **HTML**-страницы, открытой в браузере

На месте всех подставляемых значений, требуется написать код:

```
<?php echo $Имя_переменной;?>
```

Этот код выводит значение переменной при помощи команды «эхо». В нашем случае таких переменных две, это:

```
<?php echo $StartDate;?>
```

и

```
<?php echo $StartTime;?>
```

Конечно же, чтобы что-то вывести, нужно вначале это «что-то» в переменную записать, поэтому выше в коде необходимо задать значение переменной:

```
$Имя_переменной = ...;
```

Стоит обратить внимание, что имена переменных в PHP всегда начинаются со знака доллара «**$**».

Во всех наших **PHP**-файлах мы будем придерживаться структуры из **двух частей**:

1. Вначале – **код на языке PHP**, содержащий запись значений всех переменных внутри тега «**<?php … ?>**». Здесь же будут производиться все необходимые вычисления для получения этих значений (в том числе запрос их из базы данных);
2. Во второй части – **код на языке HTML** с выводом в нужных местах значений командой «**echo**». В команде «**echo**» используются непосредственно имена переменных и не проводятся вычисления, максимум речь может идти лишь о выполнении здесь простейших операций, таких как взятие элемента массива, округление, указание формата отображения чисел с плавающей точкой (запятой), и т.п.

!!! Без веской на то причины стоит всегда придерживаться именно такой структуры **PHP**-документа. *Вескость причины доказывает автор работы***!**

Тогда рассмотренный вначале **HTML**-документ примет следующий вид **PHP**-документа:

```
<?php
  //Запоминаем текущие дату и время в переменные,
  //чтобы вывести их дальше
  $StartDate = date('d.m.Y');
  $StartTime = date('H:i:s');
?>

<html>
  <head> <title>СУБД - Новиков</title> </head>
  <body>
      Дата обращения: <?php echo $StartDate;?><br>
      Время обращения: <?php echo $StartTime;?>
  </body>
</html>
```

В языке **PHP** имеется несколько вариантов записи комментариев. Мы будем пользоваться комментарием, начинающимся с символов «//» и заканчивающимся в конце строки (или в конце PHP-тега, символами «**?>**»):

```
//Однострочный комментарий
```

Стоит отметить, что, несмотря на сказанное вначале (про «подстановку значений»), **PHP** является полноценным языком программирования общего назначения, поддерживающим все типовые средства для программирования, такие, например, как: условия, циклы, или массивы.

4.5. SQL-запросы на языке PHP

На самом деле, **SQL** не предназначен для постоянного ручного ввода запросов к базе данных. Он предназначен для автоматизации процесса проведения запросов, и используется в составе какого-либо из языков программирования (в нашем случае это будет язык **PHP**).

1. Подключение к серверу

Для выполнения запроса, необходимо вначале установить подключение к серверу, которое выполняется командой **mysqli_connect**(Адрес, Логин, Пароль, База_Данных), например:

```
//Подключиться к базе данных
$connect = mysqli_connect("localhost", "root", "", "Novikov");
```

Далее необходимо проверить наличие ошибок соединения:

```
//Завершить работу в случае ошибки
if ($connect == false) {
    print("Невозможно подключиться к MySQL");
    exit;
}
```

Кроме того, необходимо установить кодировку, для правильного отображения русских символов:

```
//Установить кодировку
mysqli_set_charset($connect, 'utf8');
```

Естественно, что в конце данное соединение необходимо закрыть командой:

```
//Завершить соединение с базой данных
mysqli_close($connect);
```

2. Выполнение запроса

Запросы **SQL** пишутся между команд на подключение и на отключение (после команды установки кодировки). В одном подключении можно выполнить и несколько **SQL**-запросов.

Рассмотрим простейший запрос к таблице «**Test_tbl**», созданной ранее (см. раздел **2.3**):

```
//Запросить данные из БД
$sql = "SELECT * FROM Test_tbl WHERE Номер=1";
$result = mysqli_query($connect, $sql);
//Завершить работу в случае ошибки
if ($result == false) {
    print("Ошибка при выполнении запроса<br>$sql");
    exit;
}
```

Обычно каждый запрос строится из трех частей: текст запроса, команда на выполнение этого запроса и проверка на ошибку.

Язык **PHP** позволяет объединять несколько строковых значений в одно, это делается через символ точка «**.**». Данная функция позволит правильно оформить **SQL**-запросы в несколько строк:

```
$sql = "SELECT * ".
       "FROM lab2 ".
       "WHERE Номер=1";
```

3. Получение и вывод результата запроса

Описанный выше запрос в случае успеха помещает результат в переменную **$result**, при этом результат, это **всегда таблица**, т.е. двумерный массив (даже если содержит всего одну строку). Данная таблица **всегда имеет шапку**. Таблица может быть и пустой (содержать ноль строк).

Например, мы можем получить результат запроса для «Номер=1» с одной строкой (рис. 131). Или результат с нулем строк для «Номер=99» (рис. 132). Если вовсе убрать условие WHERE, то получим всю таблицу с 5-ю строками (рис. 133). *Данные таблицы с результатами не выводятся на экран, они хранятся в памяти в переменной* ***$result****, которую и нужно использовать в дальнейшем для отображения результатов.*

Номер	Имя звена
1	Интегрирующее

Рисунок 131 – Результат запроса с одной строкой

Номер	Имя звена

Рисунок 132 – Результат запроса с нулем строк (пустая таблица)

Номер	Имя звена
1	Интегрирующее
2	Апериодическое 1-го порядка
3	Апериодическое 2-го порядка
4	Дифференцирующее
5	Транспортное запаздывание

Рисунок 133 – Результат запроса с несколькими строками

Для того, чтобы узнать количество полученных строк, в PHP используется команда:

```
$result->num_rows
```

Для того, чтобы получить только одну строку результата (первую), т.е. одномерный массив, необходимо выполнить команду:

```
//Получаем первую строку с результатом
$r = mysqli_fetch_array($result);
```

!!! Данную команду необходимо выполнять, даже если в таблице с результатом заведомо всего одна строка**!**

Теперь, во второй части документа (в коде **HTML**), мы сможем получить любое из значений этой строки, используя команду «**echo**», например:

```
<?php echo $r['Имя звена'];?>
```

Что касается имени переменной **$r**, то оно намеренно выбрано коротким, из одной буквы, чтобы не загромождать в дальнейшем код HTML многочисленным ее использованием. *Стоит обратить внимание, что если выполняются несколько запросов, то и имен переменных (**$result** и **$r**) для них может понадобиться несколько разных, иначе мы рискуем затереть предыдущие значения новыми. При этом имена таким переменным стоит давать более осмысленные, отражающие суть хранящихся в них данных, например, **$markets**, а переменную, содержащую одну строчку, по-прежнему стараться назвать коротким именем, например, просто **$m**.*

Запрос по **номеру** хорош тем, что мы уверены, что нам будет выдано не более одной строки (т.к. поле «**Номер**» помечено как уникальное). Но при этом нам может быть выдано и менее одной строки, т.е. ноль строк (когда строка с таким номером не существует). Попытка получить значения таких несуществующих полей приведет к множественным ошибкам (рис. 134). Поэтому необходимо проверять количество полученных строк:

```
//Проверяем, что найдена хотя бы одна строка
if ($result->num_rows < 1) {
    print("Указан неверный номер строки");
    exit;
}
```

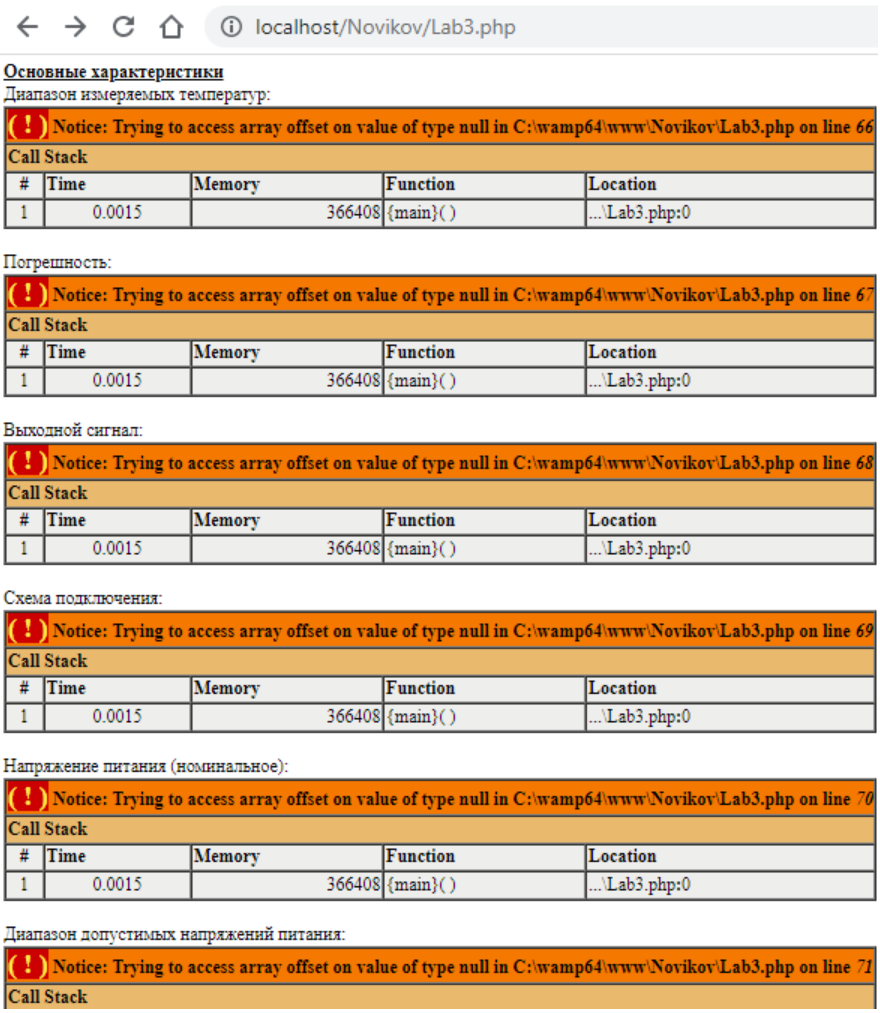

Рисунок 134 – Множественные ошибки при обращении к пустой таблице

4.6. РАБОТА № 3 (часть 2 из 3)
«Использование SQL в PHP»

!!! Для того, чтобы приступить к данной части, необходимо вначале завершить выполнение Работы № 2 и Работы № 3 (часть 1).

Создадим копию файла «**Lab3.html**», дав ей имя «**Lab3.php**» (данная операция проводится в папке со своей фамилией, например, «**C:\wamp64\www\Novikov**»).

Далее отредактируем файл «**Lab3.php**» таким образом, чтобы:

— он начал состоять из двух частей: **PHP** и **HTML**;

— в первой части файла организуем подключение к базе данных (со списком оборудования), созданной в прошлой Работе;

— разработаем **SQL**-запрос (или несколько запросов), получающий все характеристики для одного из устройств (например, первого в списке) и реализуем его в **PHP**;

— заменим (во второй части файла, в коде **HTML**) все статически выводимые на странице характеристики устройства, соответствующими командами «**echo**», реализовав таким образом

получение его характеристик из базы данных. *Данный пункт имеет смысл разбить на два этапа, вначале вывести все характеристики, касающиеся непосредственно устройства (такие как его название, изображение, производитель, погрешность, диапазон измерения, напряжение питания и т.п.), а потом вывести значения из связанных таблиц (такие как, например, магазины и наличие в этих магазинах)*;

— в завершении поменяем в запросе номер устройства (например, на второй), и убедимся в правильности работы и в этом случае;
— кроме того, поменяем номер устройства на заведомо несуществующий (например, 99), и убедимся, что правильно отрабатывается ошибка.

Для проверки работоспособности созданной страницы откроем ее из браузера по адресу «**localhost/Novikov/Lab3.php**». Необходимо выполнять проверку работоспособность страницы **на каждом из этапов** (и даже чаще!), для этого в браузере необходимо обновить страницу нажатием клавиши **F5**. Стоит напомнить, что перед обновлением страницы в браузере необходимо нажать кнопку **«Сохранить» в блокноте**.

Отчет должен содержать:

— цель и задачи;
— скриншоты таблиц (из Работы № 2, возможно немного доработанные) по которым осуществляются запросы;
— описание хода работы (Что, как и в какой программе делалось? Со скриншотами);
— код созданных страниц (HTML+PHP+SQL) с выделением синтаксиса разными цветами и скриншоты созданных страниц, в том числе страницы с ошибкой при неправильном вводе id;
— расшифровку терминов, которые встречаются в отчете (но которых не было в предыдущих отчетах, такие как HTML, PHP и т.п.);
— описание использованных команд SQL и PHP;
— титульный лист, номера страниц, оглавление, список использованной литературы (включая Интернет-ресурсы и данную методичку), выводы/заключение и т.п.

4.7. Повторение HTML-кода в PHP-цикле

При выполнении предыдущей части Работы многие могли столкнуться с трудностью: «Как вывести список магазинов, когда запрос выдает несколько строк?». Для этого необходимо обрабатывать строки в цикле.

Цикл в **PHP** имеет следующую структуру:

```
<?php while ($m = mysqli_fetch_array($markets)):?>
   <!-- Здесь размещается HTML-код, повторяемый в цикле -->
<?php endwhile;?>
```

!!! Данная конструкция используется **во второй части** нашего файла! Т.е. в коде на языке **HTML**, а не в коде на языке PHP.

Например, следующий код может вывести результат, представленный на рис. 135:

```
<?php while ($m = mysqli_fetch_array($markets)):?>
  <br><?php echo $m['Город'];?> -
  <b><?php echo $m['Количество'];?> шт.</b>
<?php endwhile;?>
```

г. Екатеринбург - **2 шт.**
г. Москва - **3 шт.**
г. Санкт-Петербург - **7 шт.**

Рисунок 135 – Вывод информации о трех магазинах

При этом в переменной **$markets** уже должен содержаться список магазинов, который необходимо получить **в первой части** файла (в коде на языке **PHP**). Список загружается при помощи **SQL**-запроса командой **mysqli_query** (см. раздел **4.5**).

4.8. *PHP-условия в HTML-коде

Аналогично циклам **while**, можно использовать и условия **if-then** или **if-then-else**. Условия имеют следующую структуру:

```
<?php if (/*Условие*/):?>
   <!-- HTML-код при True -->
<?php else:?>
   <!-- HTML-код при False -->
<?php endif;?>
```

Например, следующий код может вывести разные надписи (и разными цветами), при разном количестве имеющегося оборудования

```
<?php if ($count >= 50):?>
  <font color="#00FF00"><b>(много)</b></font>
<?php else:?>
  <font color="#FFC000"><b>(мало)</b></font>
<?php endif;?>
```

Использование «**else**» является необязательным.

4.9. Ввод данных на страницу

Для ввода данных на страницу необходимо вначале **PHP**-кода использовать следующую команду:

```
<?php
  //Номер отображаемого датчика
  //(например: localhost/Novikov/Lab3.php?id=1 )
  $id = !empty($_GET['id']) ? $_GET['id'] : 1
  …
```

Теперь в **SQL**-запросах вместо «Номер=1», «Номер=2», … «Номер=99», необходимо использовать код «**Номер=$id**».

Для вызова страницы с требуемым номером необходимо использовать адрес вида:

localhost/Novikov/Lab3.php**?id=1**

В итоге, изменяя **id** в адресе, мы сможем получить разное содержимое страницы (рис. 136-139).

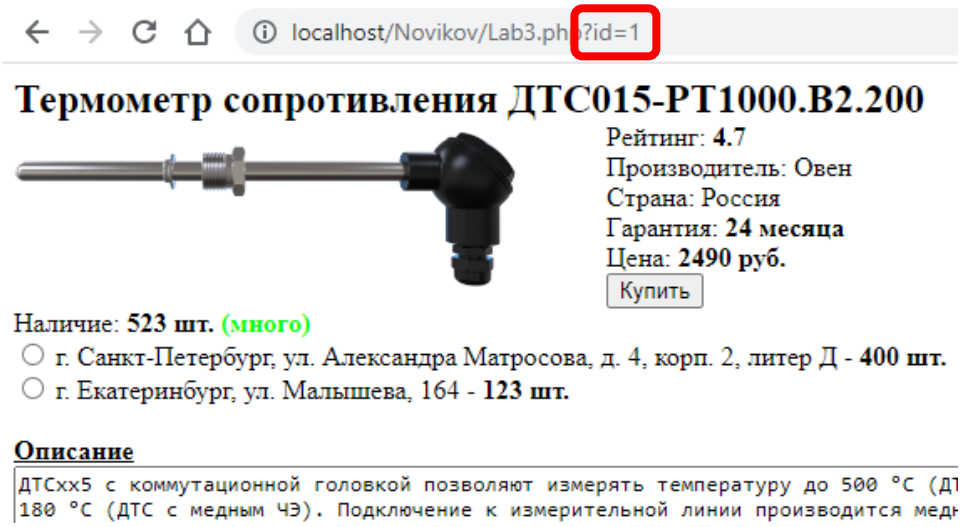

Рисунок 136 – Страница с id=1

Рисунок 137 – Страница с id=2

Рисунок 138 – Страница с id=3

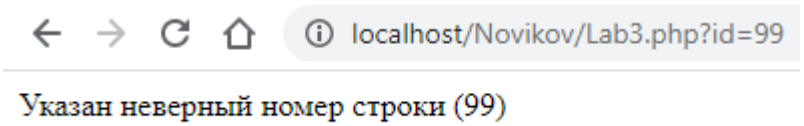

Рисунок 139 – Страница с id=99

4.10. РАБОТА № 3 (часть 3 из 3)
«Использование SQL в PHP»

В продолжение предыдущей части Работы необходимо выполнить следующее:

— доделать вывод списка магазинов (складов) при помощи цикла;
— дополнить страницу, разработанную в частях 1 и 2 данной Работы, вводом **id** в адресной строке браузера;
— проверить работоспособность страниц для всех **10**-ти возможных **id**, вводимых в адресную строку. А также проверить правильность отработки ошибки при вводе **id** для несуществующей страницы.

Содержание отчета описано в **части 2** данной Работы.

4.11. *РАБОТА № 3 (часть 4 из 3)
«Использование SQL в PHP»

!!! В настоящем разделе содержатся дополнения к Работе № 3, предназначенные для углубленного изучения, и, таким образом, не являющиеся обязательным для выполнения всеми студентами.

В проделанной ранее Работе необходимо модернизировать следующее:

— добавить вывод звезд рейтинга, в том числе нецелых (рис. 140). Для этого необходимо заранее подготовить несколько изображений (например, «Star0.png», «Star25.png», «Star50.png», «Star75.png» и «Star100.png») и разместить их в папке «**Img**»;

Рисунок 140 – Пример отображения звезд рейтинга

— добавить вывод фразы «Нет в наличии», и при этом скрывать кнопку «Купить»;
— добавить слова «Много» и «Мало» в зависимости от имеющегося в наличии количества оборудования (например, меньше 50 шт.);
— Добавить вывод текста разными цветами в зависимости от условия (например, «**Много**» – зеленым, «**Мало**» – желтым, «**Нет в наличии**» – красным);
— скрывать в списке характеристики, которые для этого устройства не заданы (поля таблицы пустые);
— уменьшать количество товара на единицу, после нажатия кнопки «Купить»;
— в папке «**Admin**» создать файл «**index.php**», позволяющий администратору (заходя по адресу «**localhost/Novikov/Admin**») увеличивать количество товара (имитировать поступление товара на склад).

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. SQL UNIQUE Constraint // W3Schools Online Web Tutorials : [сайт]. – 2024. – URL: https://www.w3schools.com/sql/sql_unique.asp (дата обращения: 02.12.2024). – Текст: электронный.
2. SQL AUTO INCREMENT Field // W3Schools Online Web Tutorials : [сайт]. – 2024. – URL: https://www.w3schools.com/sql/sql_autoincrement.asp (дата обращения: 02.12.2024). – Текст: электронный.
3. SQL DEFAULT Constraint // W3Schools Online Web Tutorials : [сайт]. – 2024. – URL: https://www.w3schools.com/sql/sql_default.asp (дата обращения: 02.12.2024). – Текст: электронный.
4. SQL Data Types for MySQL, SQL Server, and MS Access // W3Schools Online Web Tutorials : [сайт]. – 2024. – URL: https://www.w3schools.com/sql/sql_datatypes.asp (дата обращения: 02.12.2024). – Текст: электронный.
5. MySQL Functions // W3Schools Online Web Tutorials : [сайт]. – 2024. – URL: https://www.w3schools.com/sql/sql_ref_mysql.asp (дата обращения: 02.12.2024). – Текст: электронный.
6. String Functions and Operators // MySQL Developer Zone : [сайт]. – 2024. – https://dev.mysql.com/doc/refman/8.4/en/string-functions.html (дата обращения: 02.12.2024). – Текст: электронный.
7. Aggregate Function Descriptions // MySQL Developer Zone : [сайт]. – 2024. – https://dev.mysql.com/doc/refman/8.4/en/aggregate-functions.html (дата обращения: 02.12.2024). – Текст: электронный.
8. Window Function Descriptions // MySQL Developer Zone : [сайт]. – 2024. – https://dev.mysql.com/doc/refman/8.4/en/window-function-descriptions.html (дата обращения: 02.12.2024). – Текст: электронный.
9. Window Function Concepts and Syntax // MySQL Developer Zone : [сайт]. – 2024. – https://dev.mysql.com/doc/refman/8.4/en/window-functions-usage.html (дата обращения: 02.12.2024). – Текст: электронный.
10. SQL ALL Ключевое слово // Веб Учебники онлайн — schoolsw3.com : [сайт]. – 2024. – URL: https://www.schoolsw3.com/sql/sql_ref_all.php (дата обращения: 02.12.2024). – Текст: электронный.
11. SQL ANY Ключевое слово слово // Веб Учебники онлайн — schoolsw3.com : [сайт]. – 2024. – URL: https://www.schoolsw3.com/sql/sql_ref_any.php (дата обращения: 02.12.2024). – Текст: электронный.
12. SQL SELECT DISTINCT Statement // W3Schools Online Web Tutorials : [сайт]. – 2024. – URL: https://www.w3schools.com/sql/sql_distinct.asp (дата обращения: 02.12.2024). – Текст: электронный.
13. SQL Views // W3Schools Online Web Tutorials : [сайт]. – 2024. – URL: https://www.w3schools.com/sql/sql_view.asp (дата обращения: 02.12.2024). – Текст: электронный.

14. HTML Style Guide // W3Schools Online Web Tutorials : [сайт]. – 2024. – URL: https://www.w3schools.com/html/html5_syntax.asp (дата обращения: 02.12.2024). – Текст: электронный.
15. WampServer : [сайт]. – URL: https://www.wampserver.com/ (дата обращения: 05.09.2023). – Текст: электронный.
16. Notepad++ : [сайт]. – URL: https://notepad-plus-plus.org/ (дата обращения: 05.09.2023). – Текст: электронный.

Учебное издание

Новиков Александр Игоревич

Алгоритмизация и программирование SQL, Apache, PHP, HTML

Часть 1

Учебно-методическое пособие

Редактор и корректор Д. А. Романова
Техн. редактор Д. А. Романова Темплан 2025 г., поз. 5261

Подписано к печати 12.03.2025. Формат 60x84/16. Бумага тип № 1.
Печать офсетная. Печ.л. 6,7. Уч.-изд. л. 6,7.
Тираж 30 экз. Изд. № 17. Цена «С». Заказ №

Ризограф Высшей школы технологии и энергетики СПбГУПТД,
198095, Санкт-Петербург, ул. Ивана Черных, 4.