**МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ  
РОССИЙСКОЙ ФЕДЕРАЦИИ**

**Федеральное государственное автономное образовательное учреждение   
высшего образования**

**«КРЫМСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ им. В. И. ВЕРНАДСКОГО»**

**ФИЗИКО-ТЕХНИЧЕСКИЙ ИНСТИТУТ**

**Кафедра компьютерной инженерии и моделирования**

**ОТЧЕТЫ ПО ЛАБОРАТОРНЫМ РАБОТАМ**

по дисциплине «БАЗЫ ДАННЫХ»

студента 2 курса группы ПИ-233(1)

Иващенко Дениса Олеговича

направления подготовки 09.03.04 «Программная инженерия»

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

Симферополь, 2025

**Часть 1. ОСНОВЫ ПОСТРОЕНИЯ БАЗ ДАННЫХ**

**Лабораторная работа № 1**

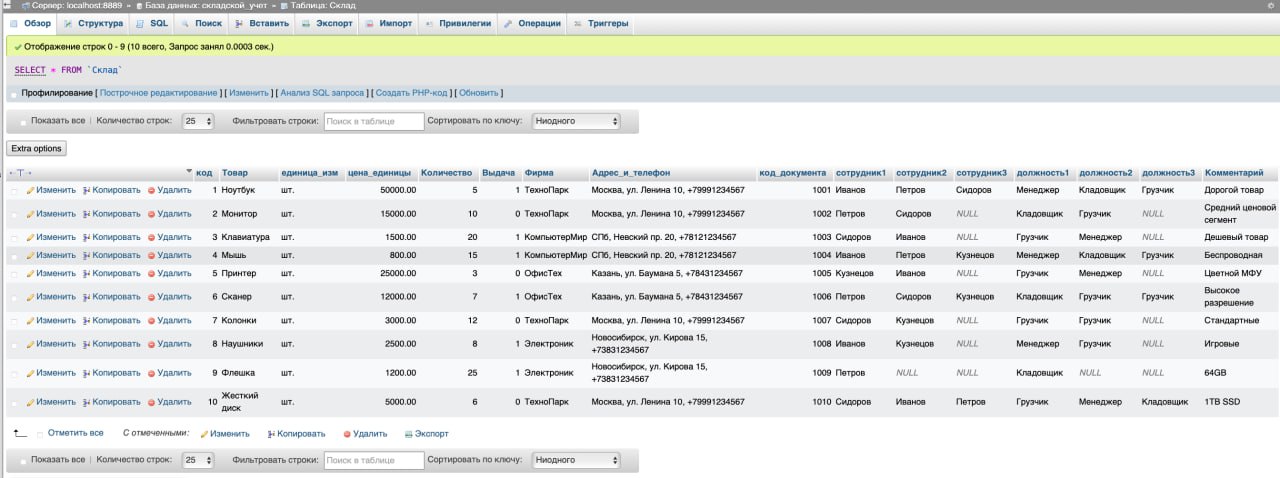
**СОЗДАНИЕ ПРОСТОЙ БАЗЫ ДАННЫХ**

**Цель работы:** ознакомиться с пользовательским интерфейсом СУБД. Научиться создавать таблицы реляционной СУБД в режиме конструктора.

1.1 Запустите программу для работы с базой данных. Войдите в режим конструктора таблиц.

1.2 Создайте таблицу для хранения данных, необходимых для учета продукции, хранящейся на складе. Каждая запись таблицы должна отражать факт приема или выдачи определенного количества товара определенной стоимости фирме-партнеру от определенных сотрудников склада (не более трех человек), по определенному документу. Сохраните созданную таблицу.

1.3 Заполните созданную таблицу данными (не менее 10 записей). Данные в столбцах с названиями товаров, фирм, и фамилий сотрудников должны повторяться два-три раза.



**Ответы на контрольные вопросы:**

1. Оптимальность хранения данных: Нет, хранение всех данных в одной таблице Складские\_Операции НЕ является оптимальным. Это связано с проблемами избыточности данных и аномалиями. Каждый раз, когда вы вводите информацию об одной и той же фирме или сотруднике, вы повторяете одни и те же данные.

• Избыточность данных: Информация о товарах, фирмах и сотрудниках будет повторяться в каждой записи, связанной с этими сущностями.

• Аномалии обновления: Изменение, например, названия фирмы, потребует обновления всех записей, где указана эта фирма.

• Аномалии вставки: Невозможно добавить информацию о фирме, если с ней нет ни одной операции на складе.

• Аномалии удаления: Удаление последней записи об операции с фирмой приведет к потере информации о самой фирме.

2. Типы данных в phpMyAdmin:

**1. Числовые типы данных**

* Целые числа:
  + TINYINT (1 байт, -128..127 или 0..255)
  + SMALLINT (2 байта, -32 768..32 767 или 0..65 535)
  + MEDIUMINT (3 байта, -8 388 608..8 388 607 или 0..16 777 215)
  + INT / INTEGER (4 байта, -2 147 483 648..2 147 483 647 или 0..4 294 967 295)
  + BIGINT (8 байта, очень большие числа)
* Числа с плавающей точкой:
  + FLOAT (4 байта, приблизительное значение)
  + DOUBLE (8 байт, более точное значение)
* Фиксированная точность:
  + DECIMAL(M, D) / NUMERIC(M, D) (точные числа, M — общее количество цифр, D — количество после запятой)

**2. Строковые типы данных**

* Строки фиксированной длины:
  + CHAR(N) (строка длиной ровно N символов, 0–255 байт)
* Строки переменной длины:
  + VARCHAR(N) (строка до N символов, 0–65 535 байт)
* Бинарные строки:
  + BINARY(N) (аналог CHAR, но для бинарных данных)
  + VARBINARY(N) (аналог VARCHAR, но для бинарных данных)
* Текст и BLOB (большие данные):
  + TINYTEXT, TEXT, MEDIUMTEXT, LONGTEXT (текстовые строки разного размера)
  + TINYBLOB, BLOB, MEDIUMBLOB, LONGBLOB (бинарные данные, например, изображения)

**3. Дата и время**

* DATE (дата в формате YYYY-MM-DD)
* TIME (время в формате HH:MM:SS)
* DATETIME (дата и время, YYYY-MM-DD HH:MM:SS)
* TIMESTAMP (метка времени, хранится как Unix-время, автоматически обновляется)
* YEAR (год, 2 или 4 цифры)

**4. JSON-данные**

* JSON (хранение и работа с JSON-структурами, начиная с MySQL 5.7.8)

**5. Геопространственные данные**

* GEOMETRY (общий тип для геоданных)
* POINT, LINESTRING, POLYGON и др. (специфические геометрические объекты)

**6. Перечисления и множества**

* ENUM('val1', 'val2', ...) (выбор одного значения из списка)
* SET('val1', 'val2', ...) (выбор нескольких значений из списка)

3. Уникальность ключевых полей: Набор данных в ключевых полях (первичных ключах) таблицы должен быть индивидуальным (уникальным) для каждой записи, потому что:

• Идентификация: Первичный ключ однозначно идентифицирует каждую запись в таблице.

• Целостность данных: Предотвращает дублирование записей.

• Связи между таблицами: Используется для установления связей между таблицами. Без уникального ключа невозможно однозначно связать записи из разных таблиц.

• Индексирование: Первичный ключ автоматически индексируется, что ускоряет поиск записей.

**Лабораторная работа № 2**

**НОРМАЛИЗАЦИЯ БАЗЫ ДАННЫХ**

**Цель работы:** Создать нормализованную базу данных, используя БД,

созданную в ходе выполнения лабораторной работы №1.

**2.1** Определите, какое количество разных сущностей содержит таблица,

созданная на предыдущей лабораторной работе. Создайте в той же базе

данных набор таблиц, позволяющих хранить ту же информацию.

**2.2** Проверьте, нет ли в получившихся таблицах полей, не входящих в

первичный ключ, но при этом содержащих одинаковую по смыслу

информацию. Проверьте, нет в таблицах полей, содержащих (в отдельном

поле) несколько разных свойств данного объекта. Внесите необходимые

исправления.

**2.3** Убедитесь, что данные в не ключевых полях таблиц зависят от всего

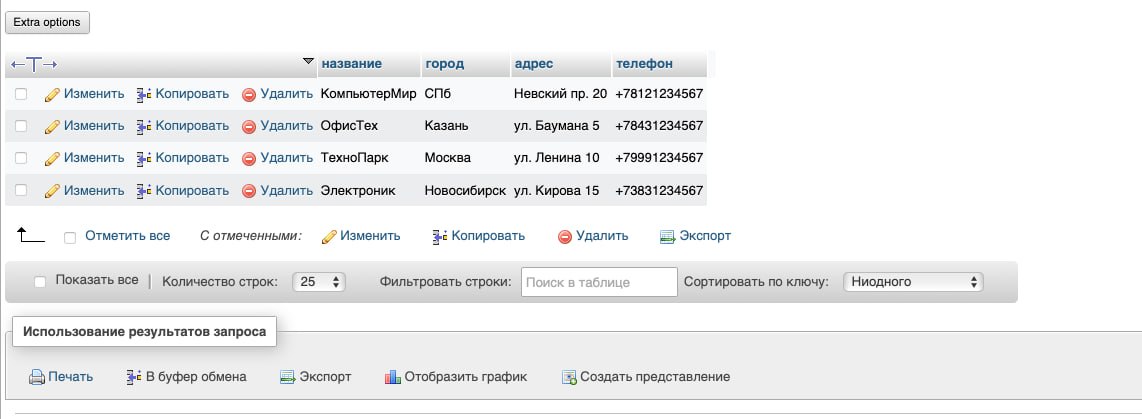
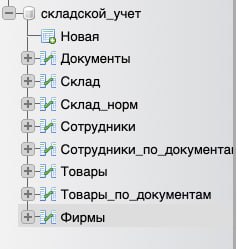
первичного ключа, а не от его части. Если это не так, внесите

необходимые исправления.

**2.4** Убедитесь, что новая база данных не содержит информации, которая

может быть легко вычислена. Если это не так, внесите необходимые

исправления.



Ответы на контрольные вопросы:

1. К каким результатам привела нормализация базы данных?

• Уменьшение избыточности данных: Исключаются повторяющиеся данные, которые занимают место и могут привести к несогласованности.

• Улучшение целостности данных: Снижается вероятность ошибок при вводе, обновлении и удалении данных. Связи между таблицами помогают поддерживать согласованность данных.

• Облегчение модификации структуры базы данных: При изменении требований к данным проще вносить изменения в нормализованную структуру, не затрагивая другие части базы.

• Оптимизация запросов: Более четкая структура позволяет создавать более эффективные запросы.

• Уменьшение места на диске: За счет устранения избыточности уменьшается объем хранимых данных.

• Упрощение логической структуры: База данных становится более понятной и логичной, что облегчает ее разработку и поддержку.

2. Какие существуют практические приемы нормализации? Какие из них можно реализовать в данной СУБД?

Существует несколько практических приемов нормализации, которые заключаются в последовательном приведении базы данных к различным нормальным формам:

• 1НФ (Первая нормальная форма):

• Прием: Устранение повторяющихся групп полей. Каждое поле должно содержать атомарные (неделимые) значения. Не допускаются многозначные атрибуты.

• Реализация в СУБД: Реализуется путем создания отдельных столбцов для каждого атрибута и/или создания связанных таблиц. В большинстве современных СУБД (включая Access, MySQL, PostgreSQL, SQL Server и др.) этот принцип является базовым и поддерживается по умолчанию.

• 2НФ (Вторая нормальная форма):

• Прием: Соблюдение 1НФ и устранение функциональной зависимости не ключевых атрибутов от части составного первичного ключа. Т.е. не ключевые атрибуты должны зависеть от всего первичного ключа, а не только от его части.

• Реализация в СУБД: Требует разделения таблицы на две или более таблицы. Создается новая таблица для атрибутов, зависящих только от части ключа, и связывается с исходной таблицей через внешний ключ. Практически все современные СУБД поддерживают создание связей между таблицами с использованием внешних ключей, что позволяет реализовать 2НФ.

• 3НФ (Третья нормальная форма):

• Прием: Соблюдение 2НФ и устранение транзитивной зависимости неключевых атрибутов от первичного ключа. Т.е. не ключевые атрибуты должны зависеть непосредственно от первичного ключа, а не через другой не ключевой атрибут.

• Реализация в СУБД: Разделение таблицы на две или более таблицы. Создается новая таблица для транзитивно зависимого атрибута и связывается с исходной таблицей через внешний ключ. Поддерживается во всех современных СУБД через механизмы создания таблиц и внешних ключей.

• НФБК (Нормальная форма Бойса-Кодда):

• Прием: Каждая детерминанта (атрибут, определяющий другие атрибуты) должна быть супер-ключом. Это более строгая версия 3НФ, касающаяся случаев с перекрывающимися составными ключами.

• Реализация в СУБД: Реализация может потребовать сложной реструктуризации таблиц и связей. Поддерживается косвенно через возможности СУБД по созданию таблиц, связей и ограничений.

• 4НФ (Четвертая нормальная форма):

• Прием: Устранение многозначной зависимости неключевых атрибутов от первичного ключа.

• Реализация в СУБД: Требует создания отдельной таблицы для каждой многозначной зависимости. Поддерживается косвенно через возможности СУБД по созданию таблиц и связей.

• 5НФ (Пятая нормальная форма):

• Прием: Устранение циклических зависимостей между атрибутами.

• Реализация в СУБД: Обычно требует специализированных методов проектирования и может быть сложна в реализации. Поддерживается косвенно через возможности СУБД по созданию таблиц и связей.

Реализация в данной СУБД

MySQL поддерживает все основные приемы нормализации (1НФ, 2НФ, 3НФ, а также НФБК, 4НФ и 5НФ .Вы можете создавать таблицы, определять первичные ключи, устанавливать связи с использованием внешних ключей, а также использовать ограничения для обеспечения целостности данных.

3. Какие нормальные формы (НФ) вы применили в ходе выполнения данной работы?

1. Создание таблицы Товары:

• Была создана таблица Товары с атрибутами Наименование (первичный ключ), Ед. изм, Цена единицы. Это позволило устранить избыточность информации о товарах.

2. Создание таблицы Фирмы:

• Была создана таблица Фирмы с атрибутами Название (первичный ключ), Город, Адрес, Телефон. Это устранило дублирование информации о фирмах и разрешило транзитивную зависимость.

3. Создание таблицы Сотрудники:

• Была создана таблица Сотрудники с атрибутами Фамилия (первичный ключ), Должность.

4. Создание таблицы Документы:

• Содержит Код Документа, дату документа, фирму, и логический столбец выдачи.

5. Соединительные таблицы:

• Товары по докум. (название товара, код документа, количество) - где Название товара - внешний ключ из Товары, а Код документа – внешний ключ из Документы.

• Сотрудники по доку (код документа, фамилия сотрудника) - где Фамилия сотрудника - внешний ключ из Сотрудники, а код документа – внешний ключ из Документы.

• Избыточность данных устранена. Информация о товарах и фирмах хранится только в соответствующих таблицах.

• Структура базы данных стала более гибкой и удобной для дальнейшего развития. Например, добавление нового товара требует изменения только таблицы Товары, а не всех записей в таблице Склад.

В целом, проведенная нормализация до 3НФ позволила создать более эффективную, надежную и масштабируемую базу данных.

Лабораторная работа № 3

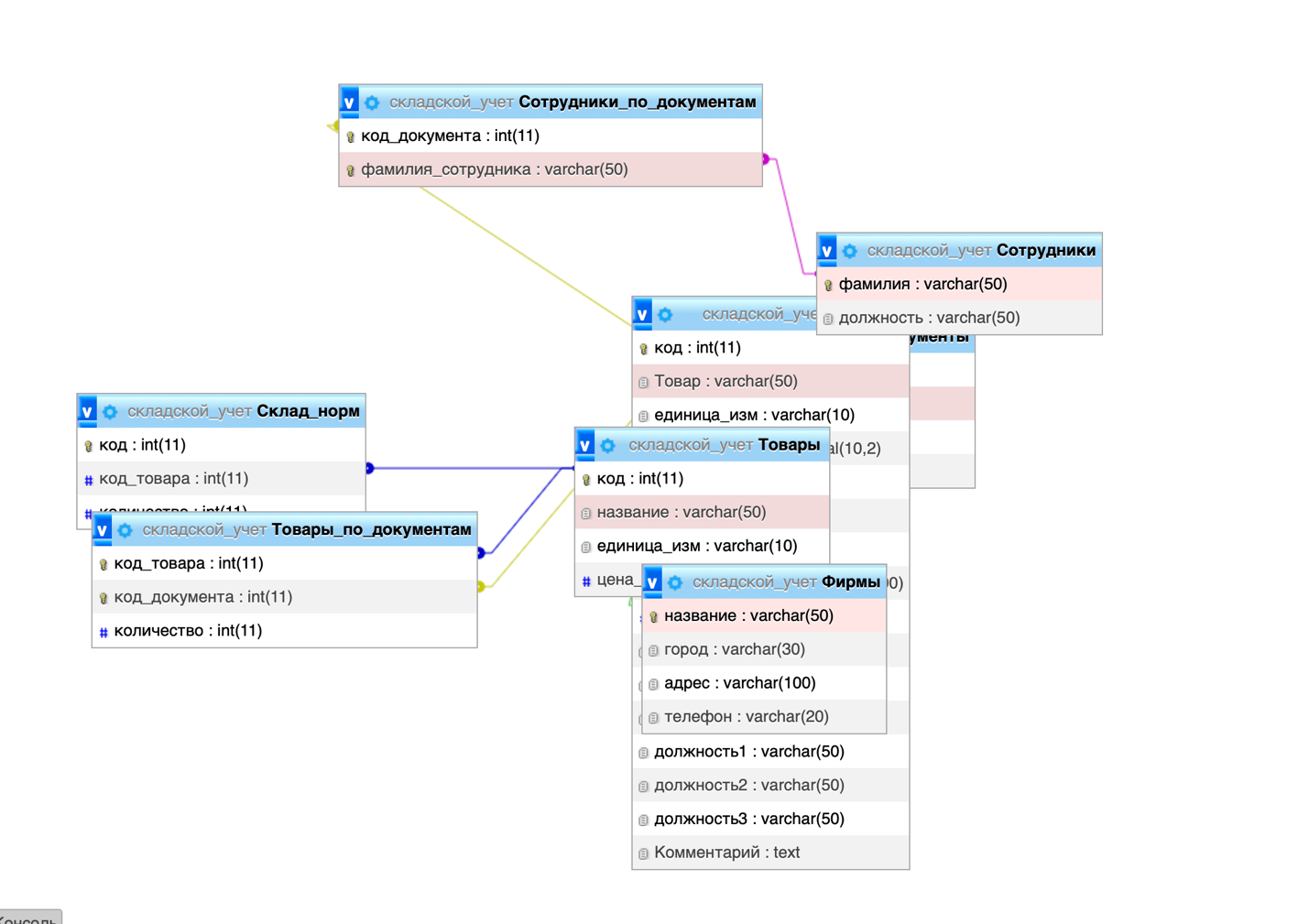
**УСТАНОВКА ФОРМАЛЬНЫХ РЕЛЯЦИОННЫХ ОТНОШЕНИЙ МЕЖДУ ТАБЛИЦАМИ. ЗАПОЛНЕНИЕ БАЗЫ ДАННЫХ**

Цель работы: Настроить в базе данных, созданной в ходе выполнения лабораторной работы №2, формальные связи между таблицами и соответствующие каскадные изменения данных.

3.1. Откройте окно, предназначенное для создания связей между таблицами. Добавьте в это окно образы таблиц.

3.2. Установите связи между таблицами (определите внешние ключи) согласно логике работы базы данных. Расположите образы таблиц так, чтобы линии связей не перекрывали друг друга.

3.3. Задайте правила обработки данных при изменении, добавлении и удалении информации в связанных таблицах. Проверьте корректность работы БД с учетом заданных правил.



Ответы на контрольные вопросы:

1. Какие реляционные отношения между таблицами существуют в вашей базе данных? Сколько отношений каждого вида имеется?

В базе данных существуют следующие реляционные отношения:

• Отношения "один-ко-многим" (one-to-many): Это наиболее распространенный тип отношений в данной базе данных.

• Фирмы - Склад: Одна фирма может иметь много записей в таблице Склад (много товаров на складе). (1 отношение)

• Товары - Склад: Один товар может быть представлен много раз в таблице Склад (разное количество товара на складе). (1 отношение)

• Документы - Склад: Один документ может иметь много записей в таблице Склад. (1 отношение)

• Документы - Товары по докум.: Один документ может содержать много различных товаров. (1 отношение)

• Документы - Сотрудники по доку: С одним документом может быть связано несколько сотрудников. (1 отношение)

• Товары - Товары по докум.: Один товар может быть в разных документах. (1 отношение)

• Сотрудники - Сотрудники по доку: Один сотрудник может быть связан с разными документами. (1 отношение)

Итого: 7 отношений "один-ко-многим"

• Отношения "многие-ко-многим" (many-to-many):

• Такое отношение реализуется через связующие таблицы.

• Товары - Документы: (через таблицу Товары по докум.) Многие товары могут быть в разных документах, и один документ может содержать много товаров. (1 отношение)

• Сотрудники - Документы: (через таблицу Сотрудники по доку) Многие сотрудники могут быть связаны с разными документами, и один документ может быть связан со многими сотрудниками. (1 отношение)

Итого 2 отношения "многие-ко-многим"

Итого: 9 отношений

2. В каких случаях при попытке изменить данные в дочерней таблице возникнет сообщение об ошибке?

Сообщение об ошибке при изменении данных в дочерней таблице (таблице, содержащей внешний ключ) возникнет в следующих случаях:

• Нарушение ссылочной целостности при вставке:

• Вы пытаетесь вставить запись в дочернюю таблицу (например, Склад), указав значение внешнего ключа (Наименование\_товара, Название\_фирмы, Код документа), которого не существует в родительской таблице (Товары, Фирмы, Документы). Например, если вы пытаетесь добавить запись в Склад с Наименование\_товара = 'Несуществующий товар', а такого товара нет в таблице Товары, возникнет ошибка.

• Нарушение ссылочной целостности при обновлении:

• Вы пытаетесь обновить значение внешнего ключа в дочерней таблице на значение, которого нет в родительской таблице.

• Нарушение ссылочной целостности при удалении записи из родительской таблицы (если не настроено каскадное удаление):

• Вы пытаетесь удалить запись из родительской таблицы (например, Товары), на которую ссылаются записи в дочерней таблице (Склад, Товары по докум.). Например, если вы пытаетесь удалить товар из таблицы Товары, который используется в записях таблицы Склад, возникнет ошибка, если в базе данных не настроено каскадное удаление.

• Нарушение ограничений NOT NULL:

• Если столбец в таблице определен как NOT NULL, и вы пытаетесь вставить или обновить запись, оставив этот столбец пустым (NULL), возникнет ошибка. Особенно это актуально для первичных и внешних ключей.

• Нарушение ограничений UNIQUE:

• Если столбец имеет ограничение UNIQUE (уникальное значение), и вы пытаетесь вставить или обновить запись, дублируя существующее значение в этом столбце, возникнет ошибка.

• Нарушение других ограничений (CHECK constraints):

• Если на столбец наложено какое-либо другое ограничение (например, проверка диапазона значений), и вы пытаетесь вставить или обновить запись, нарушающую это ограничение, возникнет ошибка.

Лабораторная работа № 4

**СОЗДАНИЕ ПРОСТЫХ SQL-ЗАПРОСОВ К БАЗЕ ДАННЫХ. СОРТИРОВКА ЗАПИСЕЙ**

Цель работы: Изучить среду для создания запросов на языке SQL. Научиться создавать элементарные запросы к таблице, созданной в ходе выполнения лабораторной работы №1.

4.1 Откройте окно для создания запроса в режиме SQL. Изучите назначения кнопок, имеющихся на панели инструментов.

4.2 Создайте простейшей запрос для вывода всей таблицы «Склад», созданной в ходе выполнения лабораторной работы №1. Создайте 2-3 запроса для вывода отдельных полей данной таблицы в разном порядке.

4.3 Создайте 4 запроса с сортировкой данных по возрастанию, по убыванию, вложенной сортировкой по возрастанию и смешанной вложенной сортировкой (убыванию и возрастанию).  
  
  
4.1

• Перейдите на вкладку Создание ("Create").

• В группе Запросы ("Queries") нажмите Конструктор запросов ("Query Design").

• В появившемся окне Добавление таблицы ("Show Table") закройте его, нажав кнопку "Закрыть" ("Close") или кнопку "Отмена" ("Cancel").

3. Переключитесь в режим SQL:

• На вкладке Конструктор ("Design") (которая появится после создания запроса) в группе Тип запроса ("Query Type") нажмите кнопку SQL

• Режим (View): Переключение между представлением SQL (для редактирования кода) и представлением таблицы (для просмотра результатов).

• Выполнить (Run): Выполняет SQL-запрос и отображает результаты в виде таблицы.

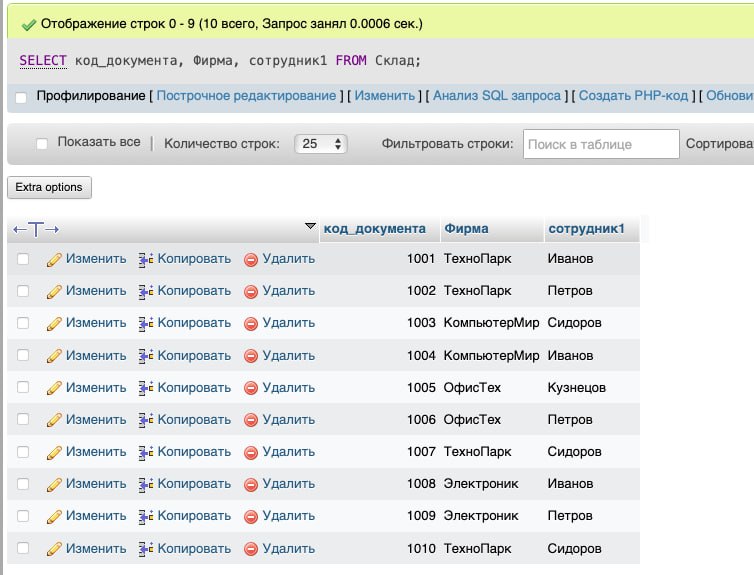
• Сохранить (Save): Сохраняет текущий SQL-запрос.

• Тип запроса (Query Type): Позволяет изменить тип запроса, например, на "Выборка", "Обновление", "Удаление" и т.д.

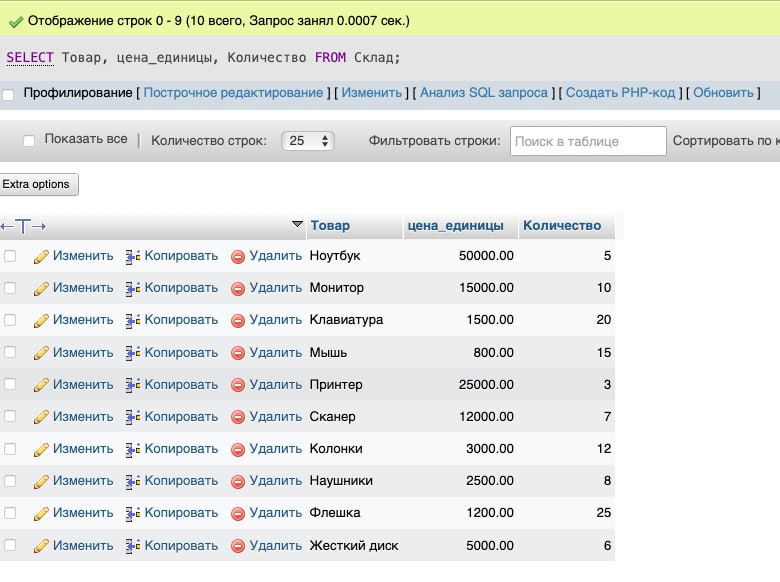
• Добавить таблицы (Show Table): Открывает окно добавления таблиц для использования в запросе (обычно не требуется для простых запросов SELECT).

• Построитель выражений (Expression Builder): Запускает инструмент для создания сложных вычислений и выражений внутри запроса.

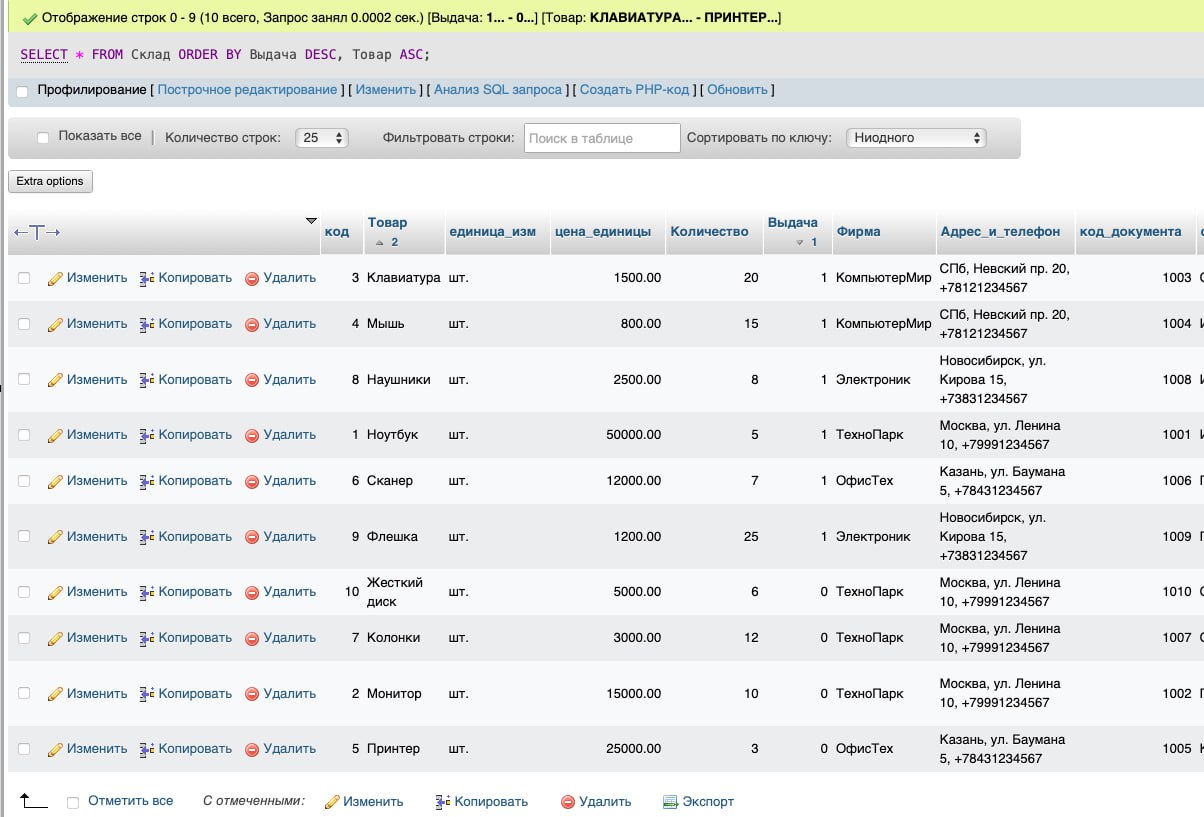
4.2  
Вывод всей таблицы Склад  
  
2-3 запроса для вывода разных полей таблицы

1)выводит код\_документа,Фрима, сотрудник1 из таблицы Склад  


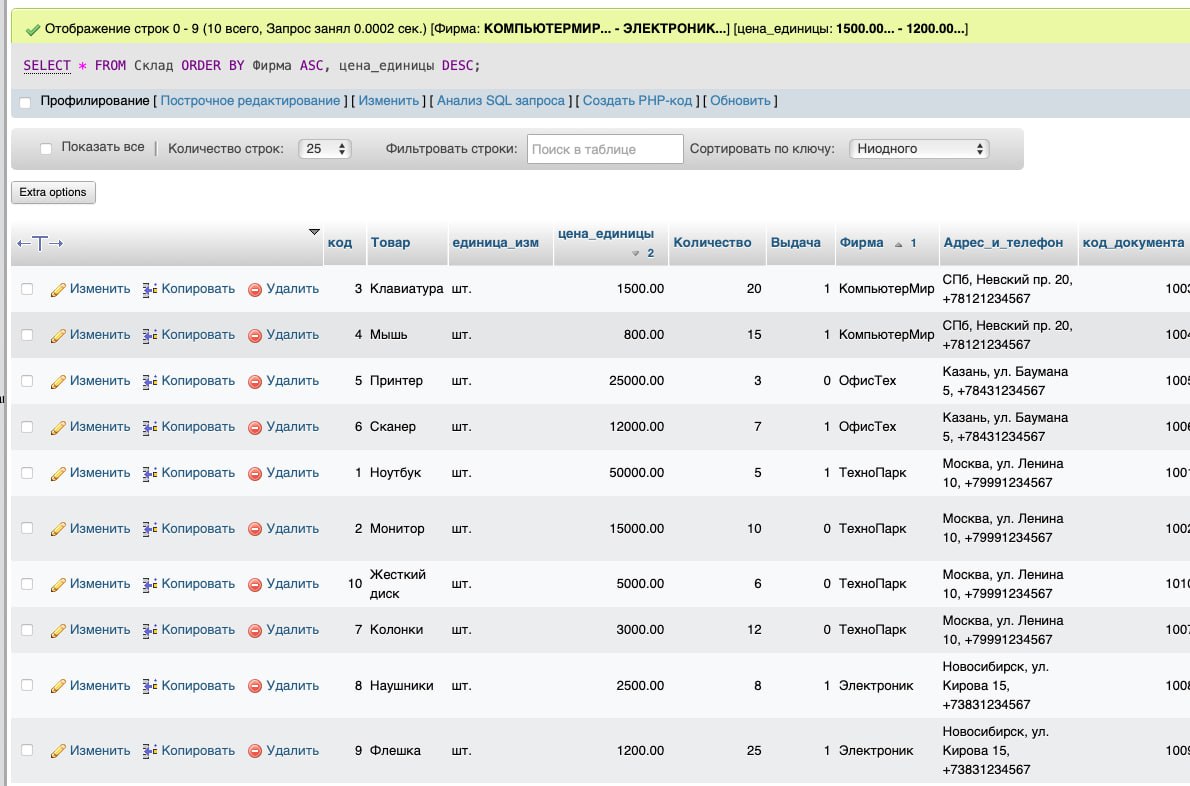
2) Товар,цена\_единицы, Количество из таблицы Склад



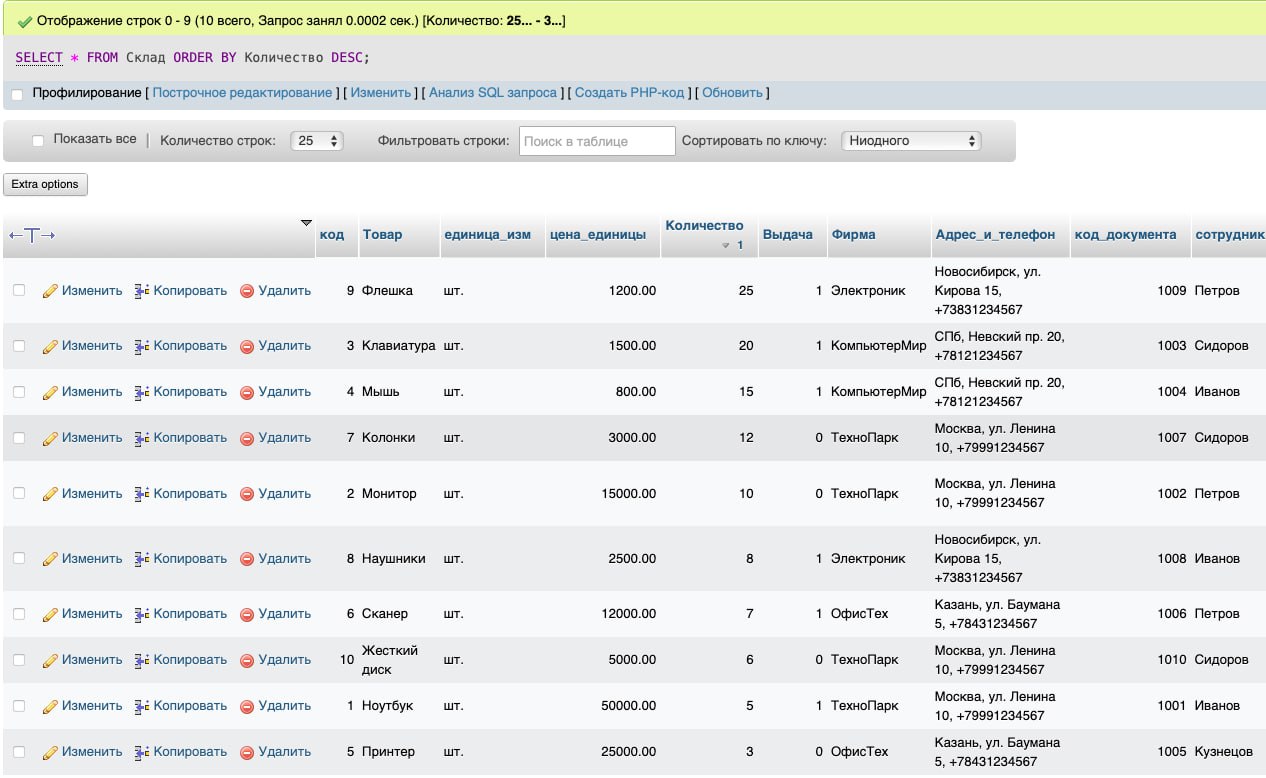
4 запроса с сортировкой данных(1-по возр. кол-ва, 2- по убыв. цены, 3- вложенная по возр., 4- смешанная) ASC – по возрастанию (по умолчанию)  
 1)сортировка по таблице “Выдача”, если будут Одинаковы значение у столбца Выдача, то будет сортировка по столбцу Товар по возрастанию



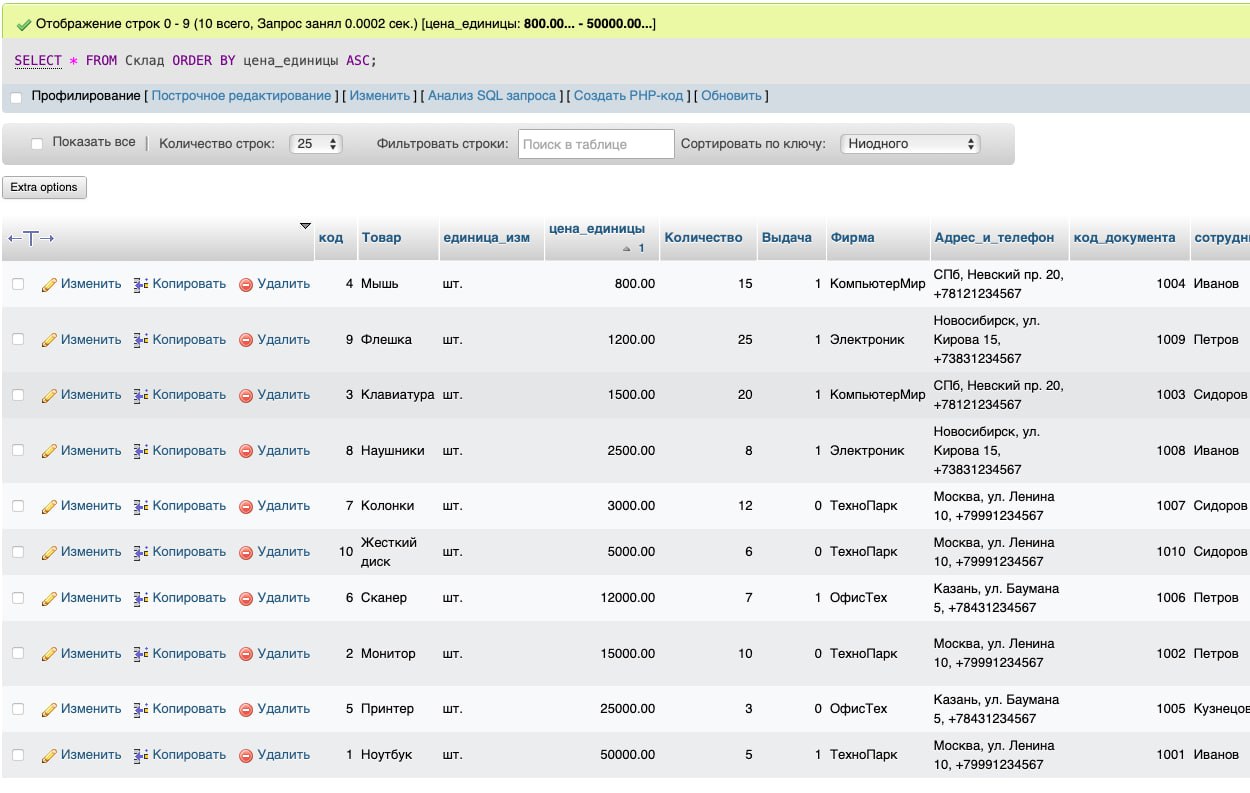
2)Берутся столбцы Фирма, которые сортируются по возрастанию и если будут одинаковые Фирма, то он будет сортировать по цене\_единице по убыванию.



1. По количеству, по убыванию



1. Сортировка по цена\_единица по возрастанию



Ответы на контрольные вопросы:  
1. Изменится ли результат запроса типа «SELECT \* FROM Таблица», если в таблице при помощи конструктора изменить порядок следования полей?

Нет, результат запроса SELECT \* FROM Таблица не изменится в плане \*данных\*. Этот запрос выводит все \*данные\* из таблицы, независимо от порядка столбцов в ее определении. Однако, изменится \*порядок отображения\* столбцов в результатах запроса.

\* SELECT \* означает "выбрать все столбцы".

\* Порядок столбцов в результате запроса SELECT \* будет соответствовать порядку, в котором столбцы определены в структуре таблицы.

Так, если вы в конструкторе таблицы измените порядок столбцов, то при выполнении SELECT \* столбцы в результате будут отображаться в новом порядке, но сами \*данные\* останутся теми же.

2. Можно ли вывести при помощи запроса одно и то же поле исходной таблицы в двух разных столбцах?

Да, это возможно. Вы можете вывести одно и то же поле несколько раз, используя \*псевдонимы\* (aliases) для столбцов.

Пример:

SELECT Товар AS "Название товара", Товар AS "Идентификатор товара"

FROM Склад;

В этом примере поле Товар из таблицы Склад будет выведено дважды: один раз под заголовком "Название товара", а второй раз под заголовком "Идентификатор товара".

3. Как работает вложенная сортировка в SQL-запросах?

Вложенная сортировка (или многоуровневая сортировка) в SQL-запросах (с использованием ORDER BY) позволяет упорядочить данные по нескольким столбцам, создавая иерархию сортировки.

Принцип работы:  
1. Первичная сортировка: Сначала данные сортируются по первому столбцу, указанному в предложении ORDER BY.

2. Вторичная сортировка: Внутри каждой группы строк, имеющих одинаковое значение в первом столбце (первичная сортировка), строки сортируются по второму столбцу, указанному в ORDER BY.

3. Третичная и последующие сортировки: Этот процесс продолжается для всех остальных столбцов, указанных в ORDER BY. Каждая последующая сортировка применяется только внутри групп, которые уже отсортированы по предыдущим столбцам.

Пример:

SELECT \*

FROM Склад

ORDER BY Категория ASC, Цена DESC;

В этом примере:

\* Сначала все записи таблицы Склад сортируются по столбцу Категория по возрастанию (ASC).

\* Затем, \*внутри каждой категории\* (например, "Фрукты", "Овощи"), записи сортируются по столбцу Цена по убыванию (DESC).

Результат:

Результатом будет таблица, в которой записи сгруппированы по категориям (в алфавитном порядке), а внутри каждой категории записи отсортированы по цене (от самой высокой к самой низкой).

Важность порядка указания столбцов:

Порядок указания столбцов в предложении ORDER BY \*критически важен\*. Сначала сортировка применяется к первому указанному столбцу, и только потом к последующим, внутри групп, образованных предыдущей сортировкой.

В общем, вложенная сортировка позволяет получить более детальное и организованное представление данных, особенно когда нужно упорядочить информацию по нескольким критериям.

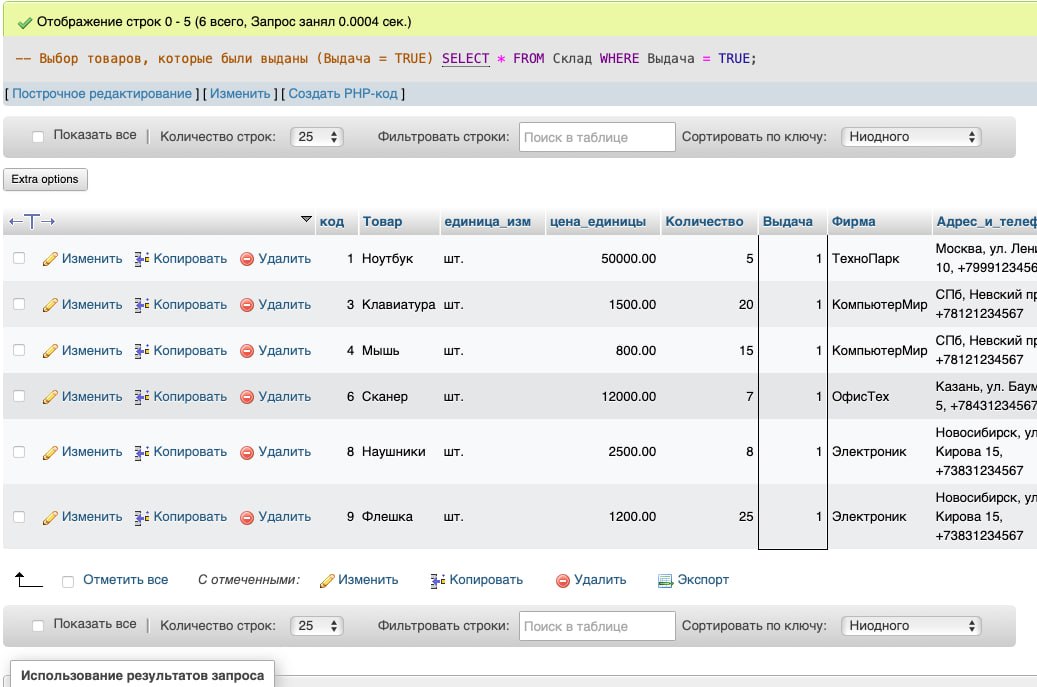
Лабораторная работа № 5

ИСПОЛЬЗОВАНИЕ ПРЕДЛОЖЕНИЯ WHERE, АРИФМЕТИЧЕСКИХ И ЛОГИЧЕСКИХ ОПЕРАТОРОВ

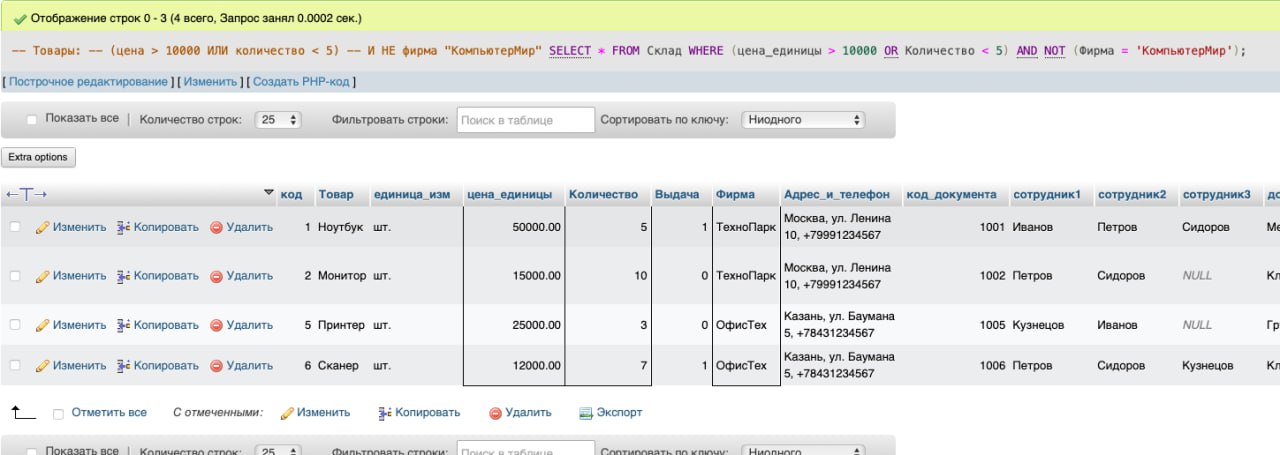
Цель работы: Освоить основные способы фильтрования записей, содержащихся в таблице БД.

5.1 Создайте простейший запрос, содержащий предложение WHERE.   
5.2 Создайте не менее двух запросов, содержащих логическое «И», логическое «ИЛИ» и операторы сравнения: =,>, =, <=, <>. Запросы должны включать в себя хотя бы одно сравнение значения поля с константой и, хотя бы одно сравнение значения поля со значением другого поля.

5.3 Создайте запрос, содержащий одновременно логическое «И», логическое «ИЛИ» и логическое «НЕ».

Выбираем все столбцы с таблицы и выводим там где “Выдача” = 1

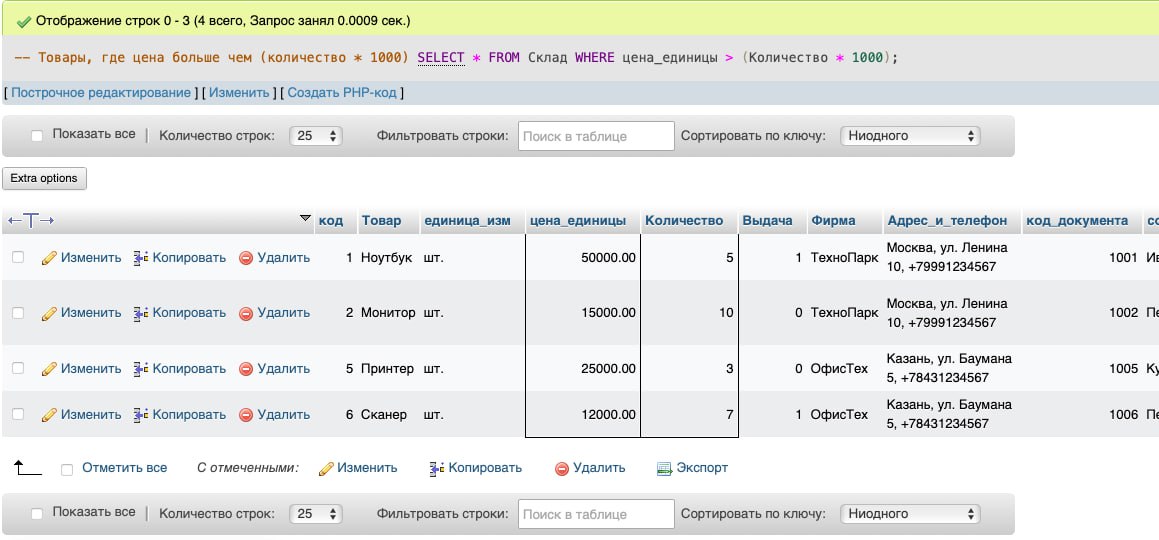
Выбираем все столбцы где “цена\_единицы” > 1000 или “Количество” < 5 и фирма не равняется “КомпьютерМир”

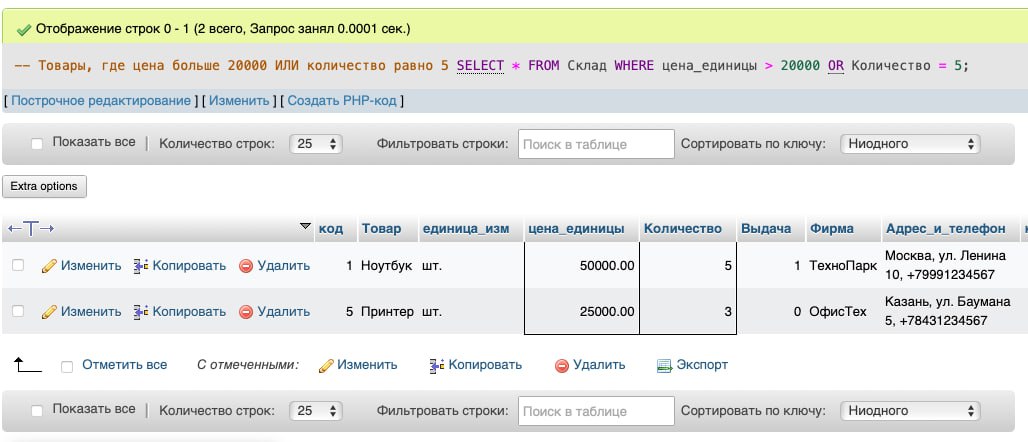


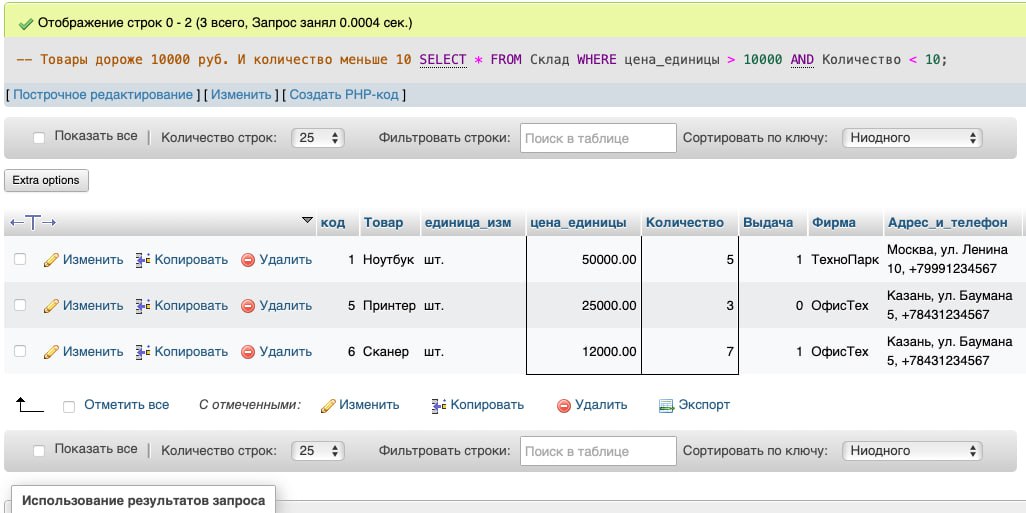
1)выбираем все стобцы таблицы где цена\_единицы > (Количество \* 1000).

2)Выбираем все столбцы где цена\_единицы больше 20000 или Кол-во = 5.

3)Выбираем все столбцы,где цена\_единицы больше 10000 и кол-во меньше 10.





Ответы на контрольные вопросы:

1. Какие типы операторов используются в предложении WHERE?

• Операторы сравнения:

\* =: Равно.

\* >: Больше.

\* <: Меньше.

\* >=: Больше или равно.

\* <=: Меньше или равно.

\* <> или !=: Не равно.

• Логические операторы:

\* AND: Логическое "И" (оба условия должны быть истинными).

\* OR: Логическое "ИЛИ" (хотя бы одно условие должно быть истинным).

\* NOT: Логическое "НЕ" (инвертирует условие).

• Операторы диапазона:

\* BETWEEN: Проверяет, находится ли значение в указанном диапазоне (включая границы).

\* NOT BETWEEN: Проверяет, находится ли значение за пределами указанного диапазона.

• Операторы проверки на NULL:

\* IS NULL: Проверяет, является ли значение NULL.

\* IS NOT NULL: Проверяет, не является ли значение NULL.

• Операторы соответствия шаблону:

\* LIKE: Проверяет, соответствует ли строка указанному шаблону (с использованием подстановочных символов).

\* NOT LIKE: Проверяет, не соответствует ли строка указанному шаблону.

• Операторы принадлежности множеству:

\* IN: Проверяет, входит ли значение в указанный список значений.

\* NOT IN: Проверяет, не входит ли значение в указанный список значений.

• Операторы существования:

\* EXISTS: Проверяет существование строк, удовлетворяющих условию подзапроса.

\* NOT EXISTS: Проверяет отсутствие строк, удовлетворяющих условию подзапроса.

2. Что можно сравнивать в предложении WHERE? Можно ли сравнивать значения разных типов данных?

В предложении WHERE можно сравнивать:

• Значения столбцов с константами (например, WHERE Количество > 10).

• Значения столбцов с другими значениями столбцов (например, WHERE Цена\_единицы > Себестоимость).

• Значения столбцов с результатами подзапросов (например, WHERE Количество > (SELECT AVG(Количество) FROM Склад)).

• Выражения (например, WHERE (Цена\_единицы \* Количество) > 1000).

• Результаты функций (например, WHERE Len(Товар) > 5).

Сравнение разных типов данных:

В общем случае, \*не рекомендуется\* сравнивать значения разных типов данных без явного приведения типов.

• Если вы пытается выполнить сравнение разных типов данных, он может попытаться автоматически преобразовать один из типов к другому. Например, при сравнении строки и числа может попытаться преобразовать строку в число.

• Неявное приведение типов может привести к непредсказуемым результатам, поэтому лучше избегать таких сравнений и использовать явное приведение типов, если это необходимо. Функции для явного приведения типов зависят от конкретной СУБД.

3. Можно ли использовать предложение ORDER BY в запросах с WHERE?

Да, можно и это очень распространено. Предложение ORDER BY используется для сортировки результатов запроса, а WHERE - для фильтрации.

4. Функционируют ли в вашем интерпретаторе языка SQL операторы !=, !<, !>. Если нет, то какое сообщение выводится на экран в случае попытки выполнения запроса?

В MySQL:

• Оператор != (не равно) функционирует.

• Операторы !< (не меньше), !> (не больше) не функционируют. Это нестандартные операторы, и MySQL их не поддерживает.

Сообщение об ошибке:

Если вы попытаетесь использовать операторы !< или !>, MySQL выдаст сообщение об ошибке синтаксиса, обычно что-то вроде:

"Синтаксическая ошибка (пропущен оператор) в выражении запроса"

или

"Недопустимый синтаксис."

Вместо !< и !> вам следует использовать комбинацию операторов NOT и <= или >=, соответственно:

• !< a эквивалентно NOT (поле <= a) или поле > a

• !> a эквивалентно NOT (поле >= a) или поле < a

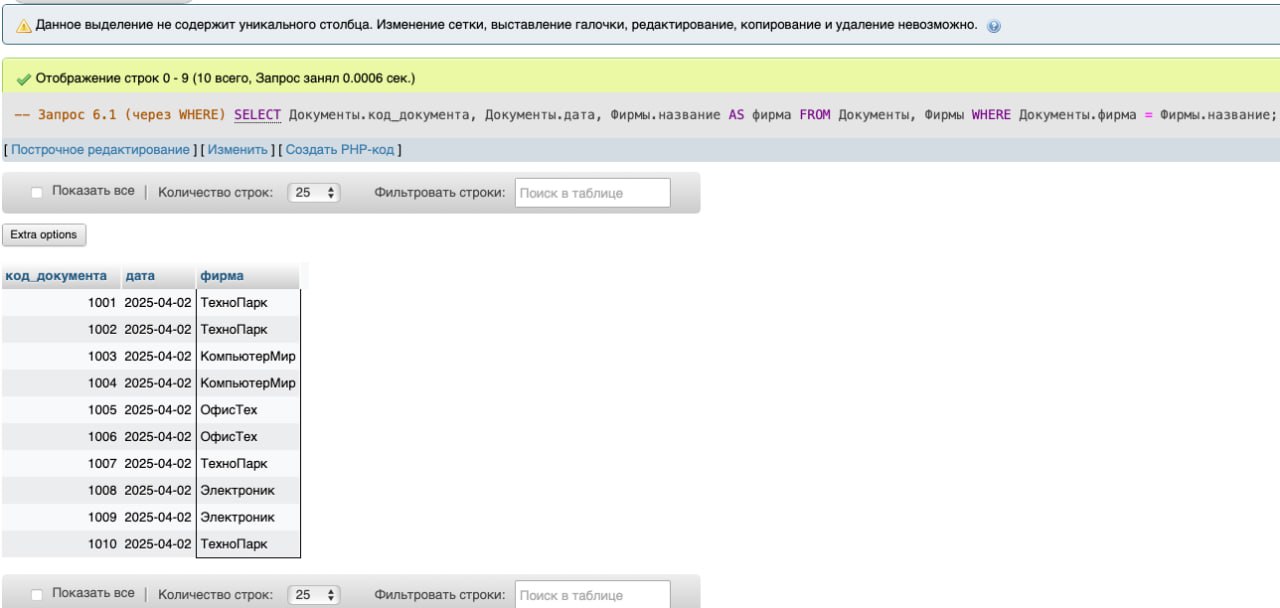
Лабораторная работа № 6

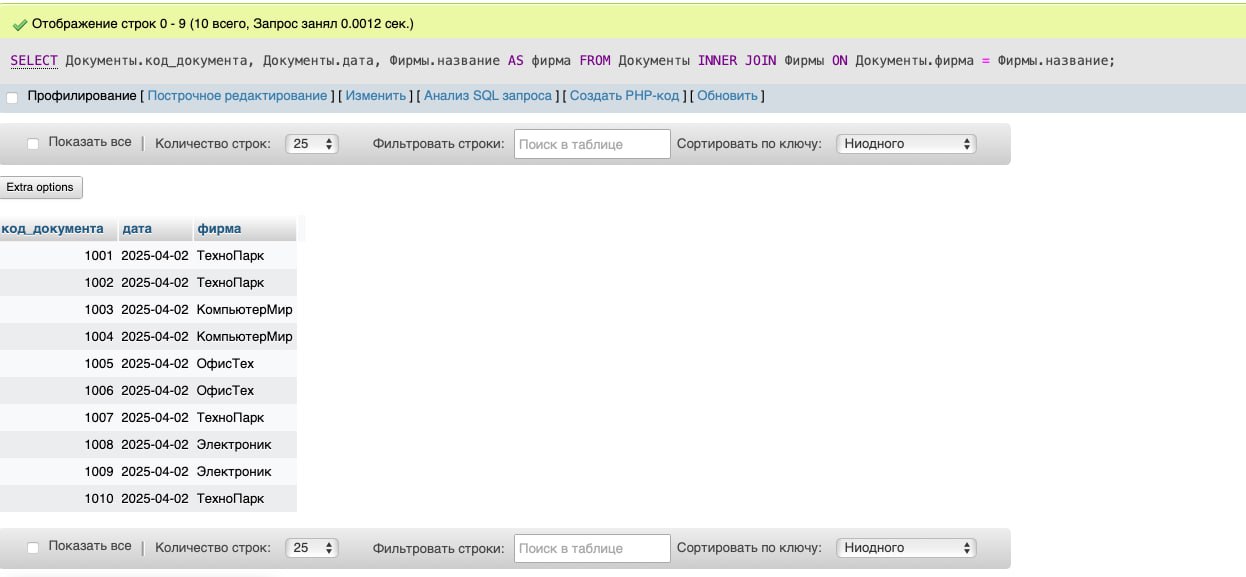
**ВНУТРЕННИЕ СОЕДИНЕНИЯ ТАБЛИЦ**

Цель работы: Изучить способы и приемы формирования SQLзапросов, получающих данные одновременно из нескольких таблиц.

6.1 Построить запрос к двум таблицам, соединенным при помощи предложения WHERE.

6.2 Построить запрос, аналогичный предыдущему, соединяющей таблицы при помощи оператора INNER JOIN.  
1)выбираем три поля из таблицы Документы строку код\_документа, где Фирмы.название просто переименовываем в фирма и соединяем в случае если совпадает Документы.фирма = Фирмы.название.Добавить адресс – фирмы.

2)  
  




Ответы на контрольные вопросы:

1. Как будет осуществляться внутреннее соединение таблиц, если одной записи таблицы А соответствует, согласно условию отбора, несколько записей таблица Б, а одной записи таблицы Б – несколько записей таблицы А?

• Для каждой записи в таблице A, которая соответствует нескольким записям в таблице B, будет создано несколько строк в результате соединения. Каждая строка в результате будет содержать данные из одной записи таблицы A и одной соответствующей ей записи таблицы B.

• Аналогично, для каждой записи в таблице B, которая соответствует нескольким записям в таблице A, также будет создано несколько строк в результате соединения.

• В результате получается комбинация всех возможных соответствий между записями в таблице A и записями в таблице B, которые удовлетворяют условию соединения. Это похоже на декартово произведение, но только для соответствующих строк.

2. Как будет осуществляться внутреннее соединение таблиц, если одной записи таблицы А, согласно условию отбора, не соответствует ни одной записи таблицы Б?

Если для записи в таблице A не найдено соответствий в таблице B на основании условия соединения, то эта запись из таблицы A не будет включена в результат внутреннего соединения (INNER JOIN). Другими словами, INNER JOIN требует, чтобы условие соединения было истинным для обеих таблиц, чтобы строка попала в результат.

Лабораторная работа № 7

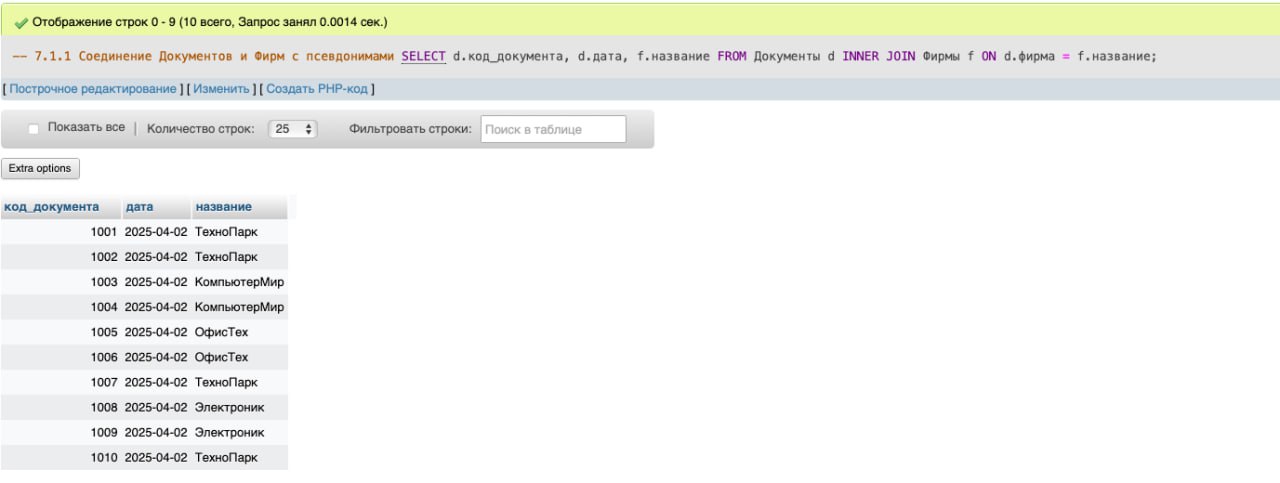
**ИСПОЛЬЗОВАНИЕ ПСЕВДОНИМОВ ПОЛЕЙ И ТАБЛИЦ**

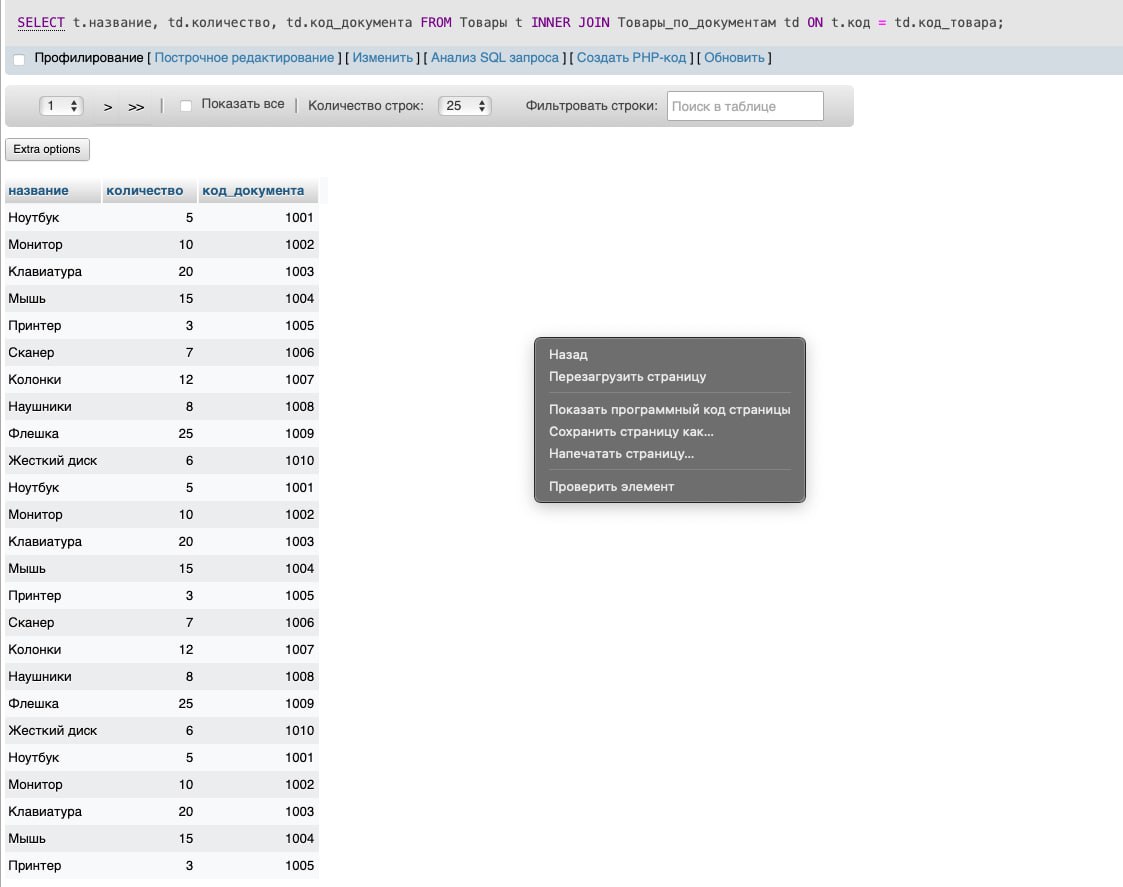
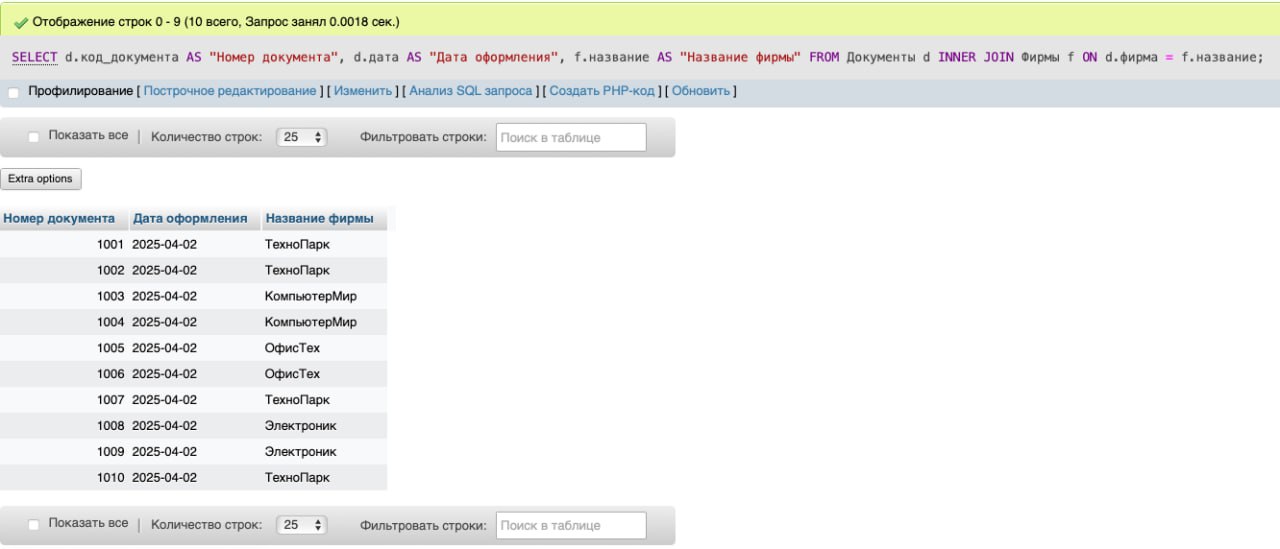
Цель работы: Научиться использовать на практике псевдонимы полей и таблиц в языке SQL.

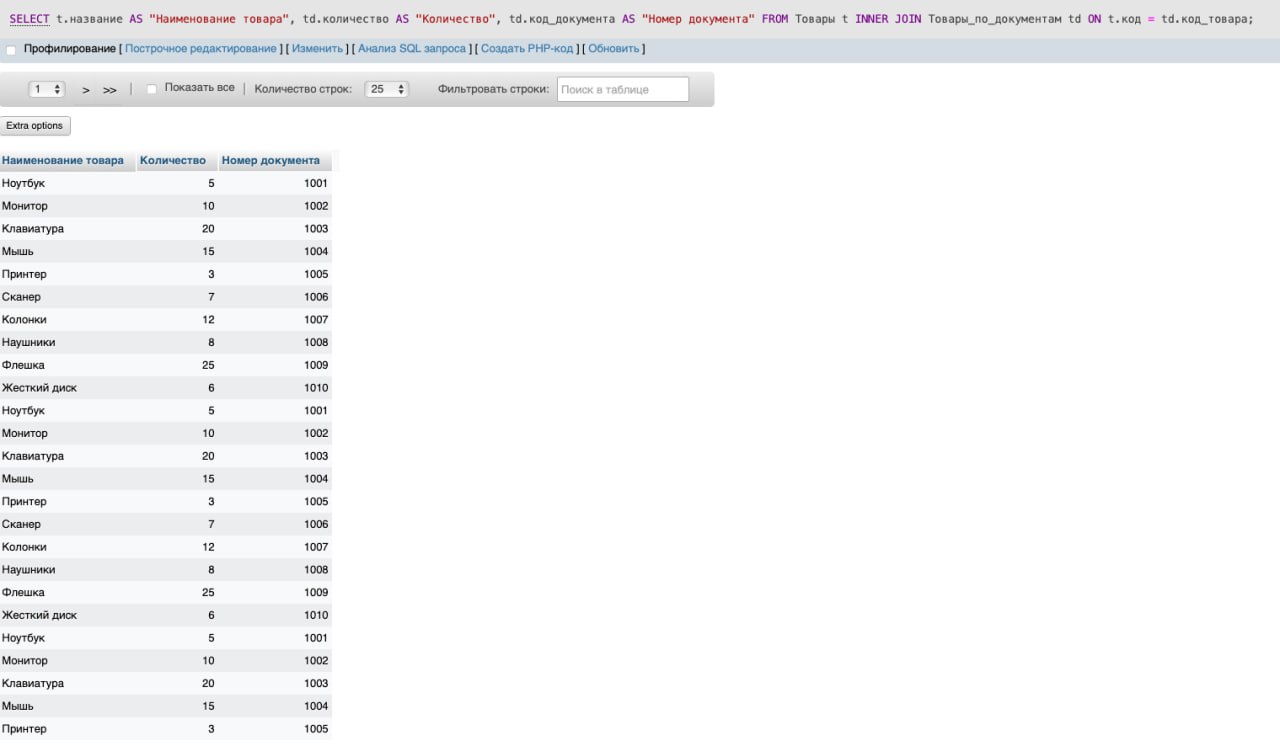
7.1 Создайте два запроса с внутренним соединением таблиц, аналогичные запросам из предыдущей лабораторной работы, но использующие вместо имен таблиц их псевдонимы.

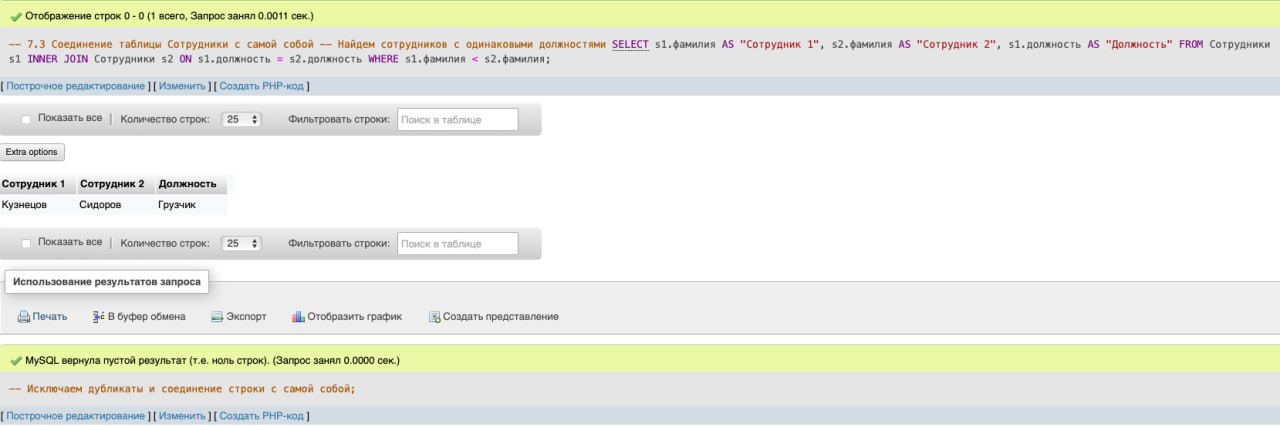
7.2 Модифицируйте запросы так, чтобы вместо названий полей в выводимых наборах данных содержались их псевдонимы.

7.3 Напишите запрос, который производит внутреннее соединение таблицы с этой же таблицей.







Ответы на контрольные вопросы:  
1. Какие преимущества дают псевдонимы полей и таблиц?

• Улучшение читаемости: Псевдонимы делают SQL-запросы более понятными и легкими для чтения. Это особенно важно для сложных запросов, включающих несколько соединений или подзапросы. Вместо длинных и потенциально запутанных имен таблиц и полей, можно использовать короткие и понятные псевдонимы.

• Упрощение синтаксиса: При работе с таблицами, имеющими длинные имена, псевдонимы позволяют сократить длину SQL-кода, что делает его менее громоздким и более удобным в написании.

• Разрешение неоднозначности: Если в нескольких таблицах, участвующих в запросе, есть поля с одинаковыми именами, псевдонимы таблиц необходимы для однозначного определения, к какой таблице относится каждое поле.

• Необходимость при самосоединениях: в запросах, где таблица соединяется сама с собой (самосоединение), псевдонимы таблиц абсолютно необходимы для различения между разными экземплярами одной и той же таблицы. Без псевдонимов невозможно указать, к какому экземпляру таблицы относится то или иное поле.

• Создание более описательных заголовков столбцов в результате: Псевдонимы полей позволяют изменить имена столбцов в результирующем наборе данных. Это полезно для предоставления более понятных и информативных заголовков для пользователей или других приложений, использующих данные.

• Упрощение рефакторинга: При изменении структуры базы данных (например, переименовании таблиц или столбцов) использование псевдонимов может упростить процесс обновления SQL-запросов, поскольку достаточно изменить псевдонимы, а не все упоминания старых имен.

2. Возможны ли SQL-запросы, в которых нельзя обойтись без использования псевдонима таблицы?

Да, существуют SQL-запросы, в которых нельзя обойтись без использования псевдонима таблицы. Главный пример - это самосоединение (self-join).

Как было указано выше, самосоединение - это соединение таблицы самой с собой. Для этого необходимо использовать псевдонимы, чтобы различать экземпляры таблицы. Без псевдонимов невозможно будет определить, к какой "копии" таблицы относится каждое поле в условии соединения или в выборе данных.

Лабораторная работа № 8

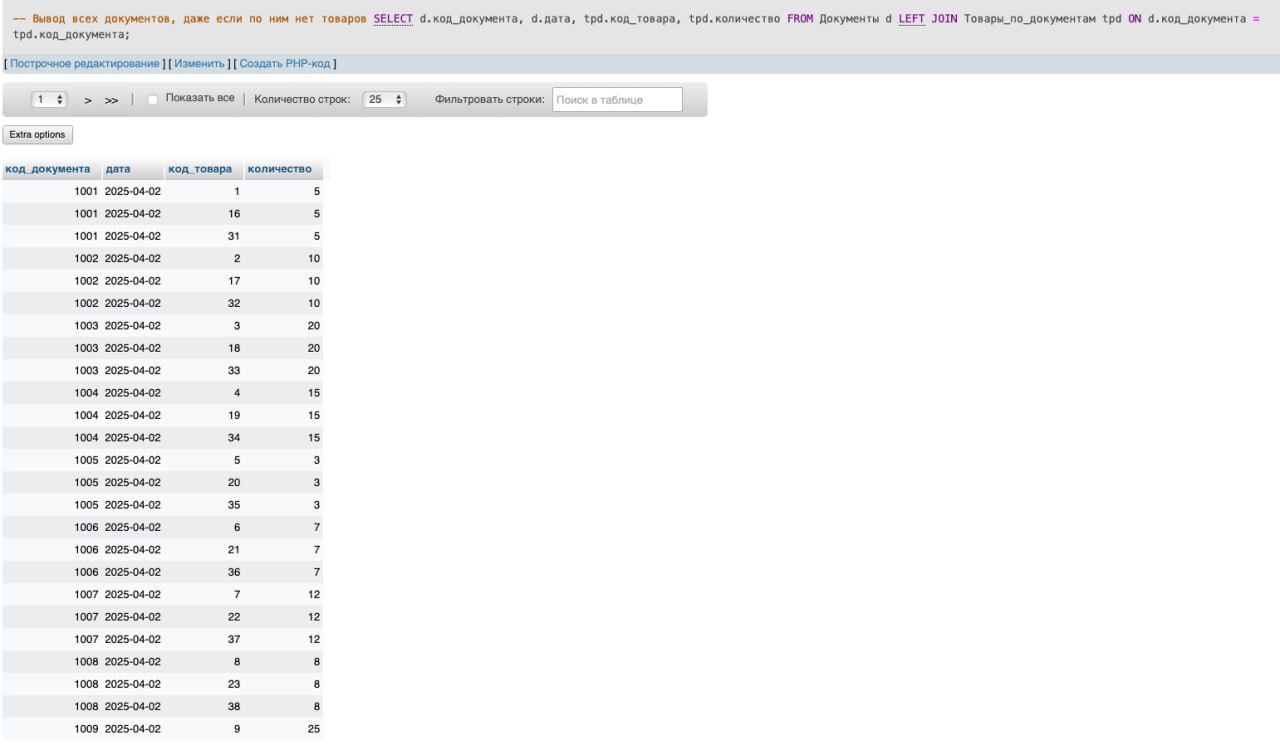
**ВНЕШНИЕ СОЕДИНЕНИЯ ТАБЛИЦ**

Цель работы: Освоить способы внешнего соединения таблиц. Улучшить понимание принципов внутреннего и внешнего соединения.

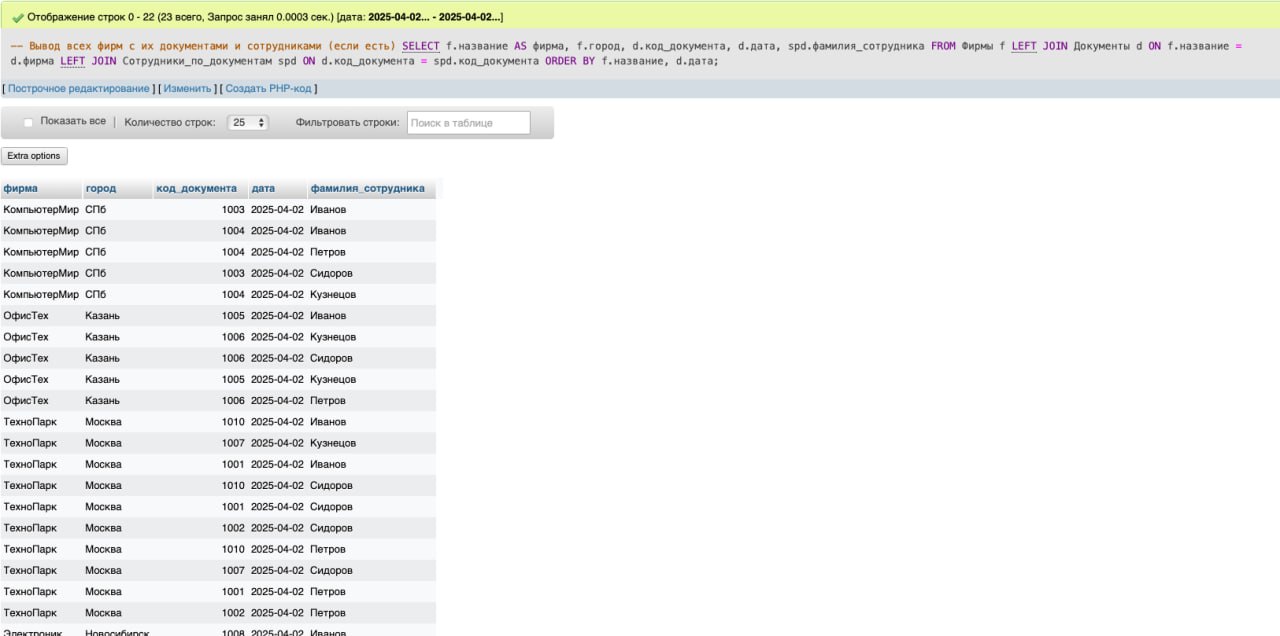
8.1 Создайте запросы к вашей базе данных, использующие левое (LEFT JOIN) и правое (RIGHT JOIN) соединения таблиц.

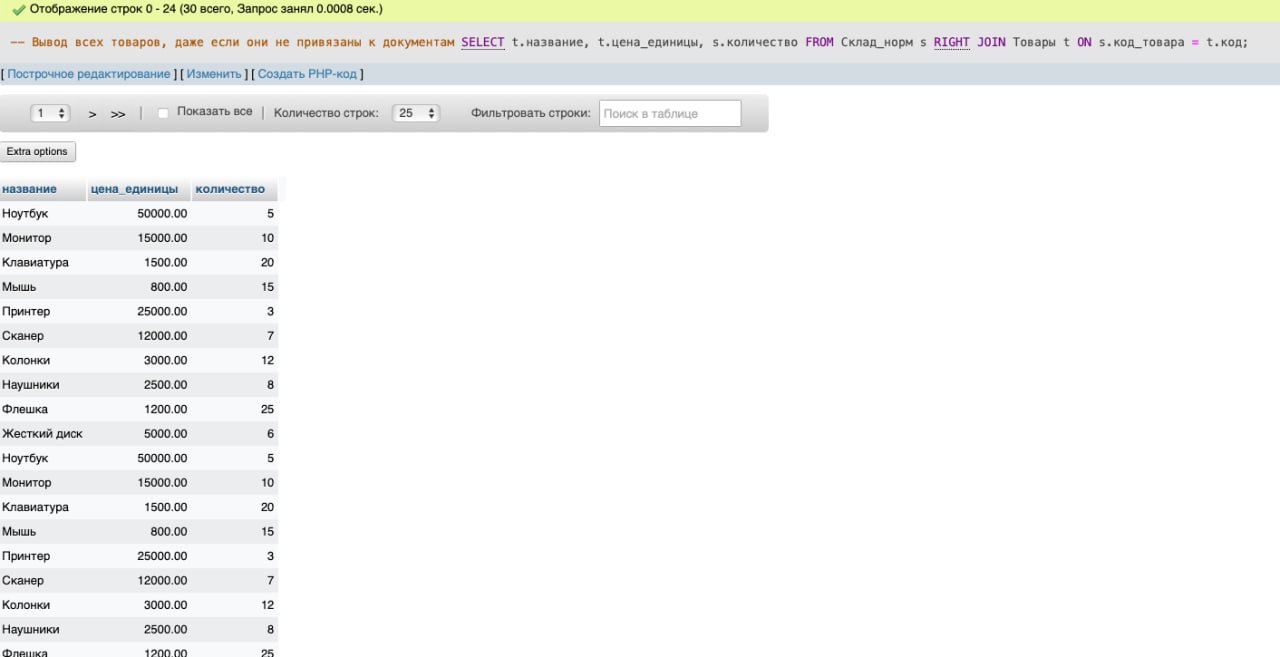
8.2 Создайте запрос, использующий внешнее соединение трех разных таблиц. Все таблицы должны быть заданы псевдонимами.

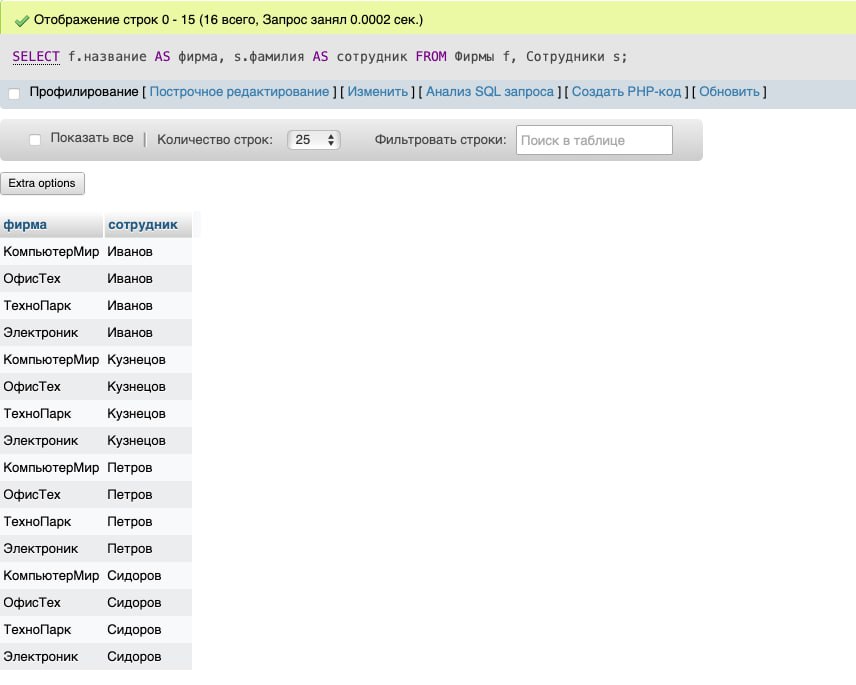
8.3 Создайте запрос к двум таблицам без указания способа соединения.



Получить все фирмы и информацию о документах, которые они создали. Если для какой-то фирмы нет соответствующих документов, показать NULL для полей документов.







Ответы на контрольные вопросы:  
1. Предположим, что мы создали запрос: SELECT A.\*, B.\* FROM A LEFT JOIN В ON... В таблице A – сто записей. Количество записей в таблице B – неизвестно. Может ли наборе данных, полученном в результате выполнения запроса быть: а) 0 записей; б) 90 записей; с) 100 записей; д) 120 записей.

а) 0 записей: Да, если таблица B пуста и ни одна запись из таблицы A не соответствует условию соединения.

б) 90 записей: Да, если только 90 записей из таблицы A имеют соответствующие записи в таблице B.

с) 100 записей: Да, если каждая запись из таблицы A имеет хотя бы одну соответствующую запись в таблице B.

д) 120 записей: Да, если некоторые записи из таблицы A имеют более одной соответствующей записи в таблице B, что приводит к увеличению общего количества записей в результате.

2. По какому принципу происходит объединение данных в таблицах, если способ соединения не задан?

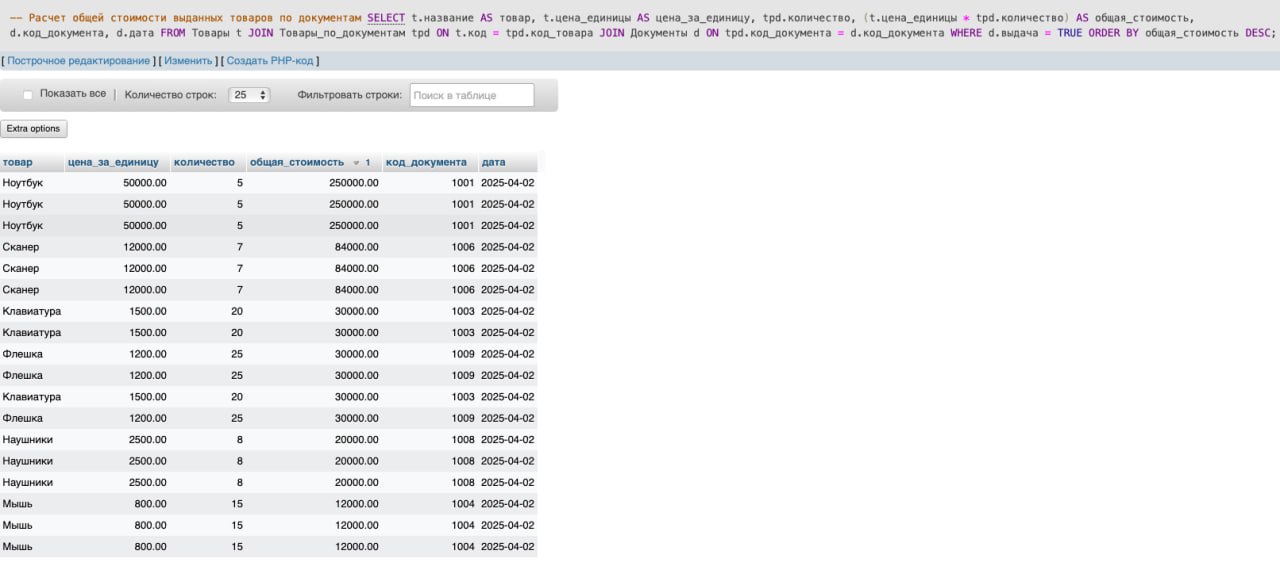
Если способ соединения не задан явно, то по умолчанию используется INNER JOIN. Это означает, что в результирующий набор данных попадут только те записи, для которых условие соединения выполняется в обеих таблицах. Если условие соединения не указано, то результатом будет декартово произведение (CROSS JOIN), то есть каждая запись из таблицы A будет соединена с каждой записью из таблицы B, что может привести к очень большому количеству записей в результате.

**Лабораторная работа № 9**

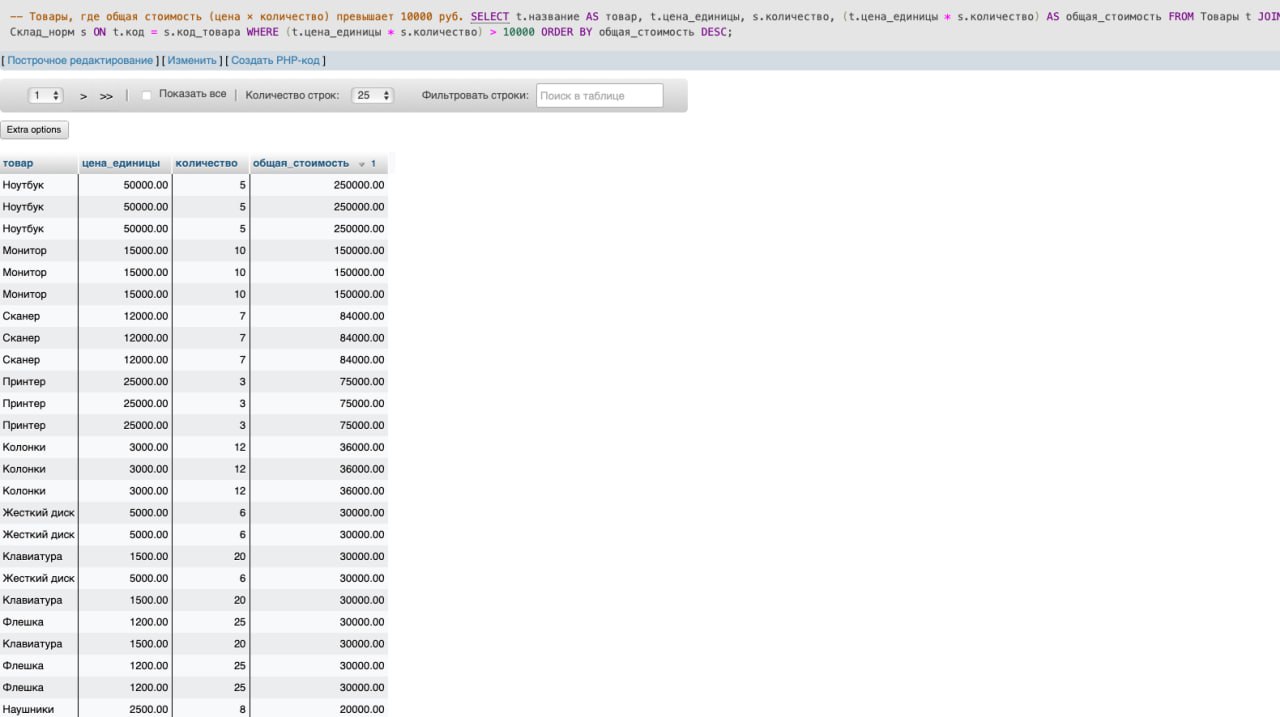
ВЫЧИСЛЕНИЯ В SQL-ЗАПРОСАХ. ПРЕОБРАЗОВАНИЕ ТИПОВ ДАННЫХ

***Цель работы****:* Освоить приемы вычислений и преобразование данных в SQL-запросах.

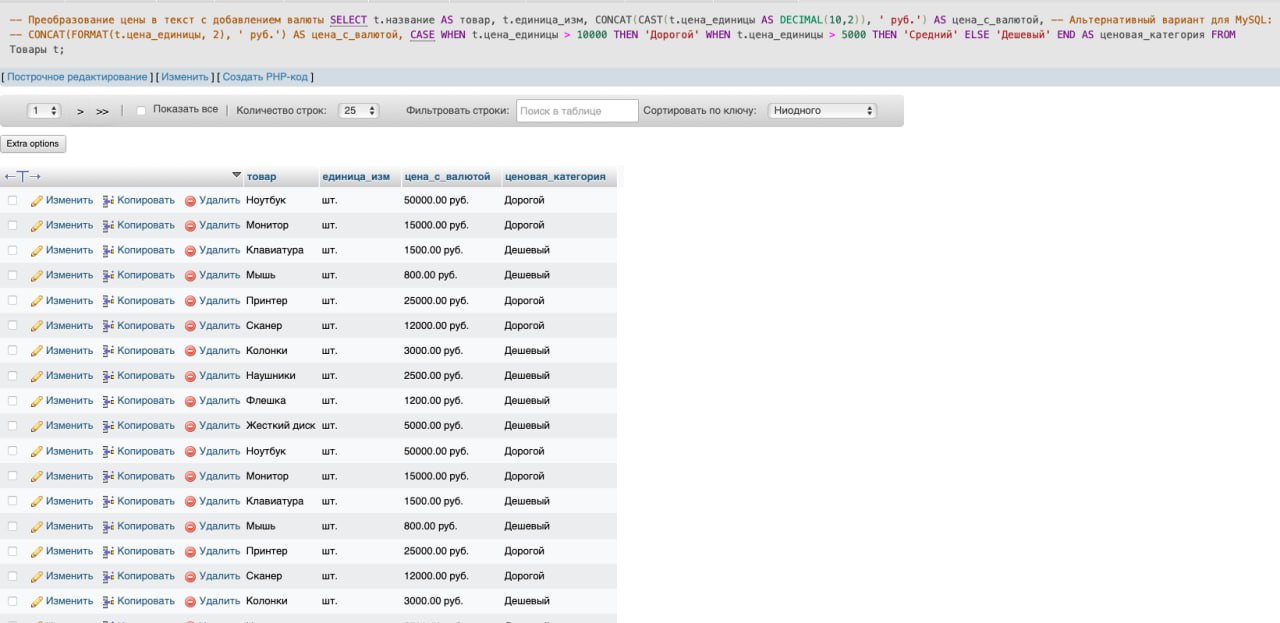
**9.1** Напишите запрос, вычисляющий стоимость выданных товаров по количеству выданных товаров и стоимости единицы товара. Количество и стоимость должны быть в разных исходных таблицах.



**9.2** Напишите запрос с вычислениями в предложении WHERE.



**9.3** Напишите запрос, вычисляющий значение столбца с преобразованием типа данных.



1. Как выглядит название вычисляемого поля в результате выполнения запроса? Как изменить это название?

Название вычисляемого поля в результате выполнения запроса задаётся с помощью ключевого слова AS. Если название не задано явно, СУБД может автоматически присвоить ему имя, например, Expr1000 или другое системное имя.

2. Можно ли использовать функции преобразования типов данных в предложении WHERE?

Да, функции преобразования типов данных можно использовать в предложении WHERE. Это полезно, если данные хранятся в одном типе (например, текстовом), а для сравнения или вычислений требуется другой тип (например, числовой).

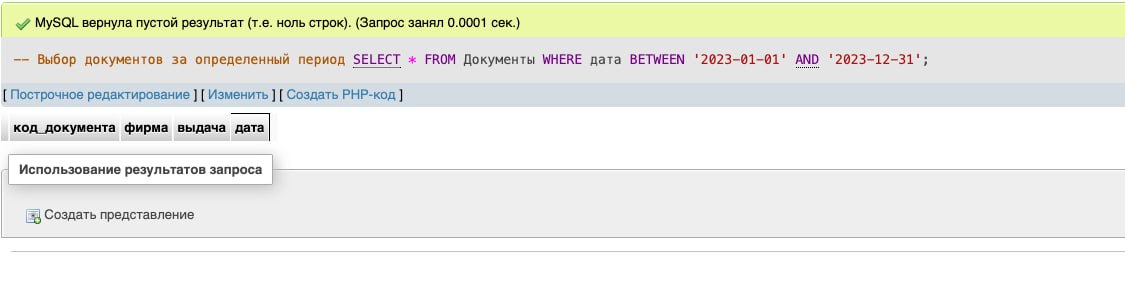
СУБД (например, MySQL, PostgreSQL) используются функции CAST или CONVERT

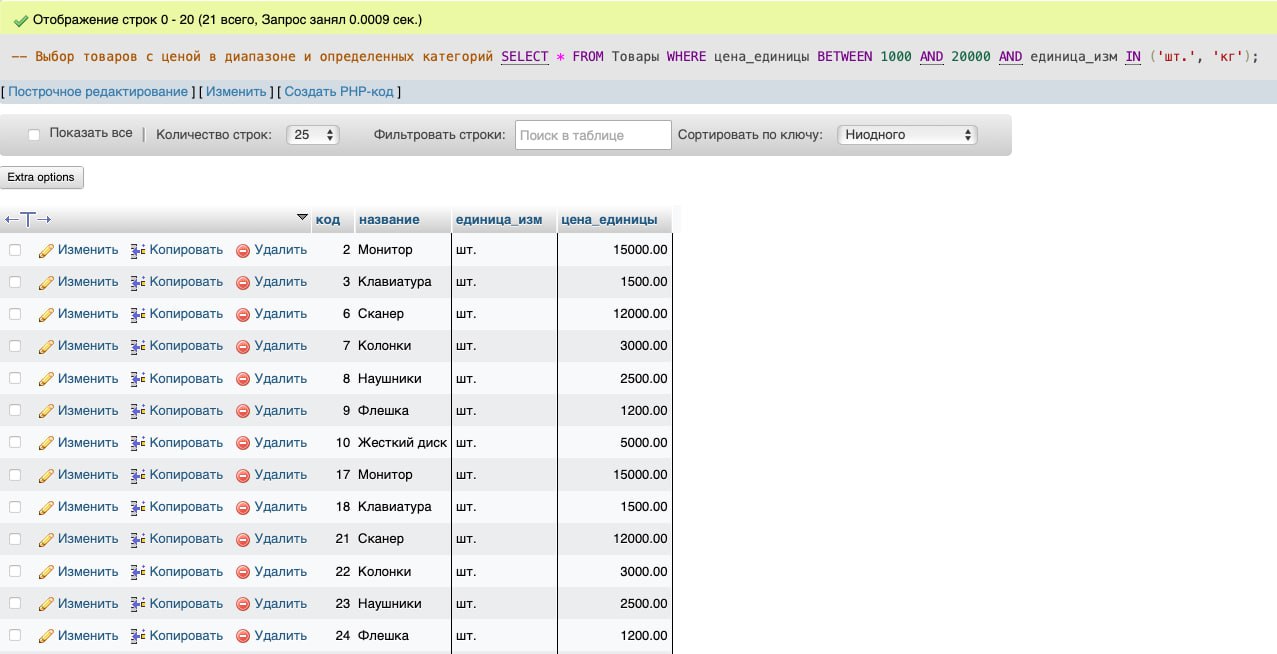
**Лабораторная работа № 10**

## ОПЕРАТОРЫ IN, BETWEEN. ИСПОЛЬЗОВАНИЕ КЛЮЧЕВОГО СЛОВА DISTINCT. АГРЕГАТНЫЕ ФУНКЦИИ

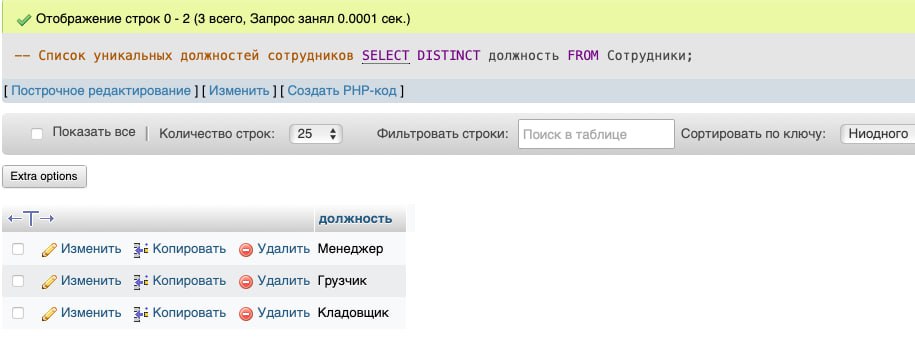
***Цель работы****:* Освоить создание простых запросов с использованием интервалов и списков значений, устранением повторяющихся значений и статистическими функциями.

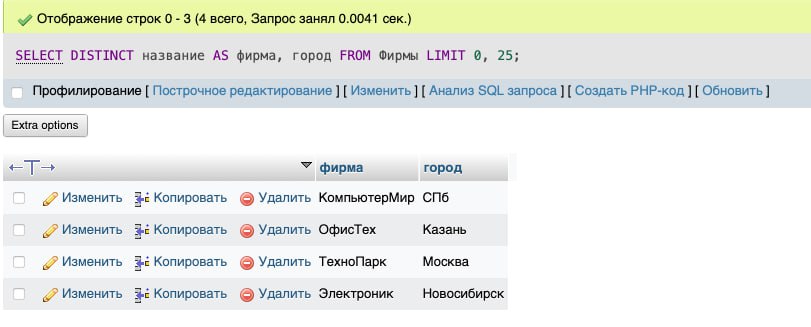
**10.1** Создайте запросы с использование операторов IN и BETWEEN.



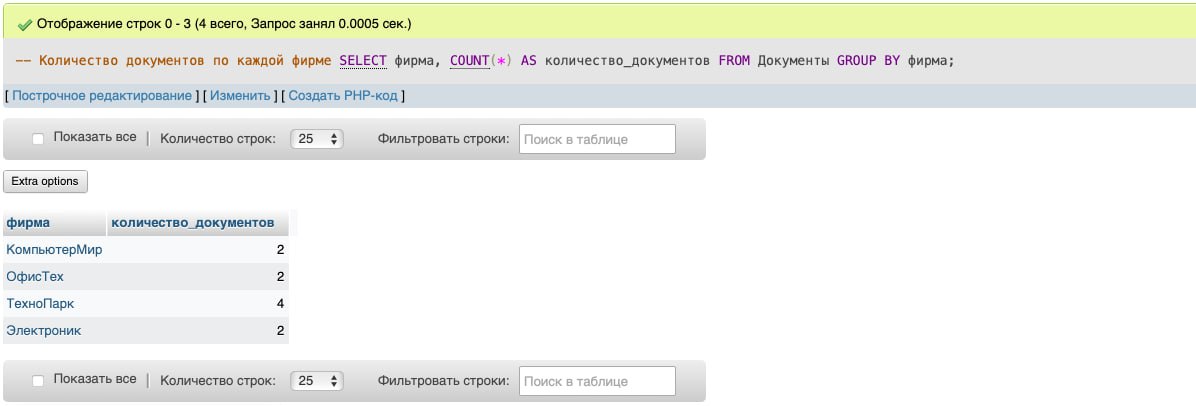


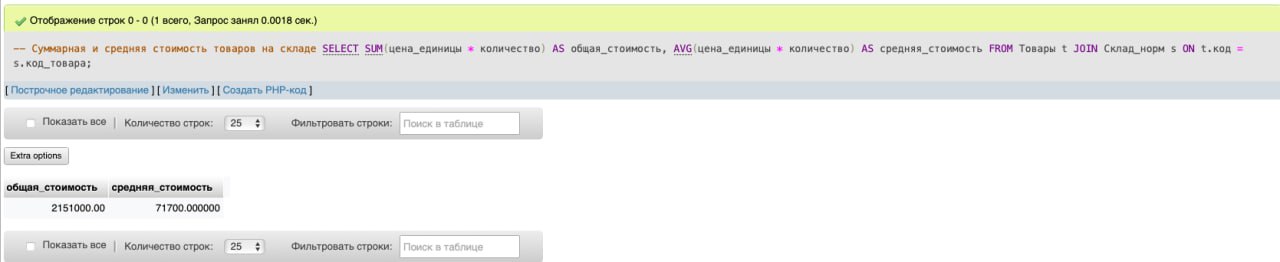
**10.2** Создайте запрос с ключевым словом DISTINCT.

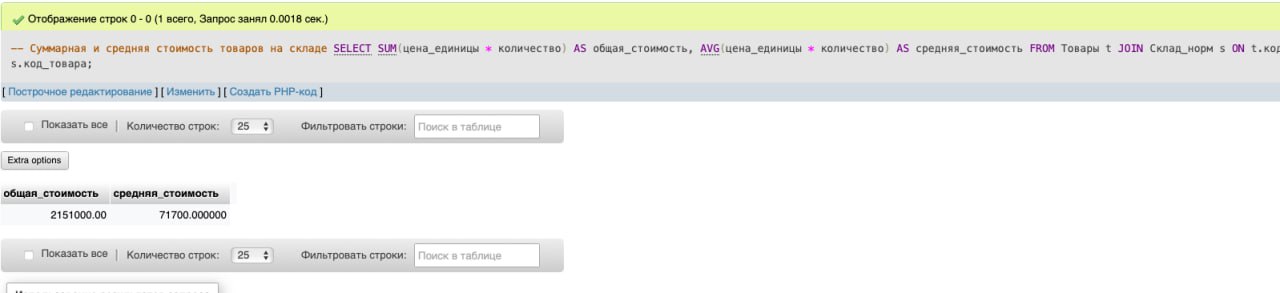


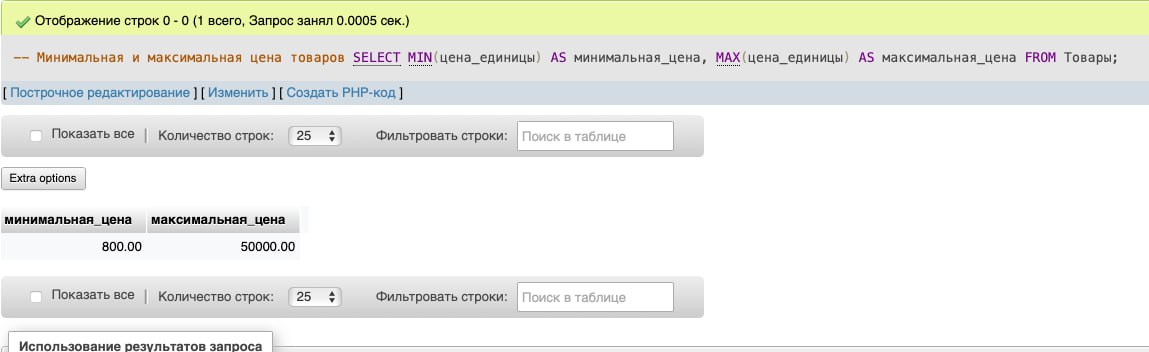


**10.3** Создайте запросы с использованием функций MIN, MAX, SUM, COUNT и AVG.









### Контрольные вопросы

1. В каких практических случаях целесообразно использовать агрегатные функции?   
   Агрегатные функции используются для выполнения вычислений над набором данных и возврата одного результата. Они полезны в следующих случаях:

Анализ данных: Получение общей суммы, среднего значения, минимального или максимального значения.

Пример: Вычисление общей суммы продаж за месяц.

Статистика: Подсчёт количества записей, удовлетворяющих определённым условиям.

Пример: Подсчёт количества товаров в категории.

Отчёты: Формирование сводных данных для отчётов.

Пример: Средняя цена товаров в каждой категории.

Оптимизация запросов: Упрощение сложных вычислений путём агрегации данных.

Пример: Получение общего количества заказов для каждого клиента.

1. Какие агрегатные функции применимы к полям текстового типа?

COUNT: Подсчёт количества строк.

Пример: SELECT COUNT(НазваниеТовара) FROM Товары;

MIN: Возвращает минимальное значение (по алфавитному порядку).

Пример: SELECT MIN(НазваниеТовара) FROM Товары;

MAX: Возвращает максимальное значение (по алфавитному порядку).

Пример: SELECT MAX(НазваниеТовара) FROM Товары;

1. Какие названия назначаются полям набора данных, полученным при использовании статистических функций? Как изменить эти названия?

Названия по умолчанию:

Если название поля не задано явно, СУБД присваивает автоматическое имя, например:

Expr1000

COUNT(\*)

SUM(ЦенаЕдиницы)

Чтобы задать понятное имя для поля, используйте ключевое слово AS.

**Лабораторная работа № 11**

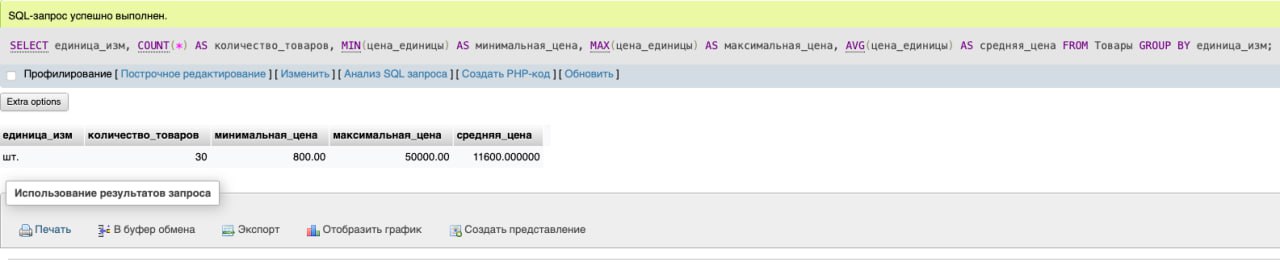
## ГРУППИРОВКА ЗАПИСЕЙ В SQL-ЗАПРОСЕ

***Цель работы****:* Освоить способ группирования записей набора данных в SQL-запросе.

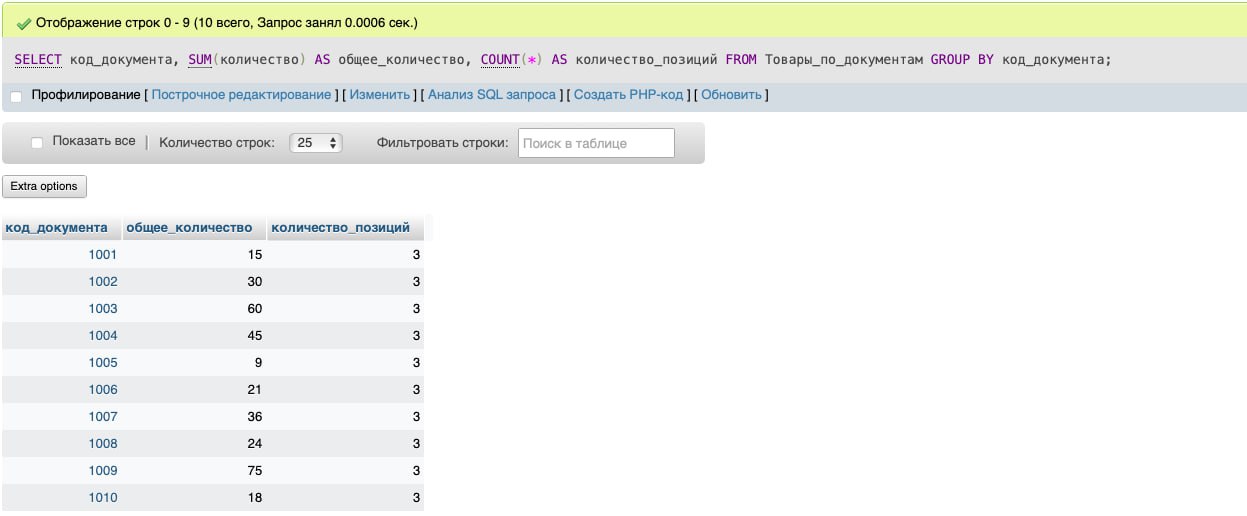
**11.1** Создайте собственные аналоги запросов с GROUP BY, приведенных в методических указаниях к данной работе.

1)группирует документы по **фирмам**, подсчитывая для каждой:

**общее количество документов**, число документов с признаком *"выдача = TRUE"* (выдано) и *"выдача = FALSE"* (получено), используя агрегатные функции COUNT и условные суммы через CASE.

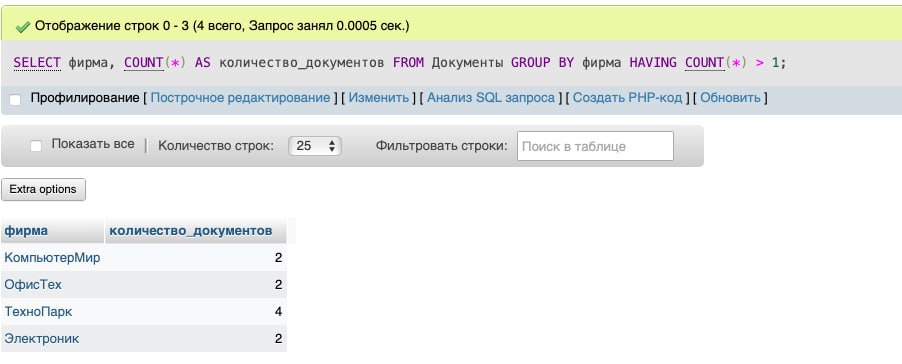


2) группирует товары по **единицам измерения**, выводя для каждой: количество товаров, минимальную, максимальную и среднюю цену, используя агрегатные функции COUNT, MIN, MAX и AVG.

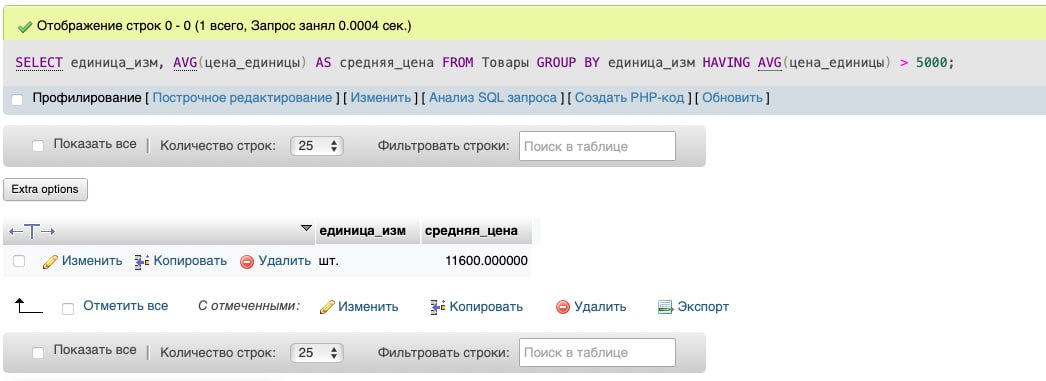


**11.2** Создайте собственные аналоги запросов с HAVING, приведенных в методических указаниях к данной работе.

3) Этот SQL-запрос группирует данные по **коду документа**, выводя для каждого: **общее количество товаров** (сумма кол-ва), **число позиций в документе** (COUNT строк), используя агрегатные функции SUM и COUNT..



4) Этот SQL-запрос группирует товары по **единицам измерения**, выводит их среднюю цену и фильтрует группы, оставляя только те, где средняя цена превышает 5000, используя GROUP BY и HAVING.



### Контрольные вопросы

1. Какие столбцы в запросе GROUP BY можно не задавать при помощи агрегатных функций?

В запросе с GROUP BY все столбцы, которые не используются в агрегатных функциях (таких как SUM, COUNT, AVG, MIN, MAX и т.д.), должны быть включены в предложение GROUP BY. Это связано с тем, что GROUP BY группирует строки по уникальным комбинациям значений указанных столбцов, и для каждой группы вычисляются агрегатные функции.  
Если вы используете только агрегатные функции в SELECT, то GROUP BY можно опустить.

1. Какие различия имеют условия в HAVING и WHERE?

Условия в HAVING и WHERE используются для фильтрации данных, но они применяются на разных этапах выполнения запроса и имеют разные назначения:

WHERE:

* Применяется до группировки (до GROUP BY).
* Используется для фильтрации строк на уровне отдельных записей.
* Может использоваться в любом запросе, не только в тех, где есть GROUP BY.
* Не может содержать агрегатные функции (например, SUM, COUNT и т.д.).

HAVING:

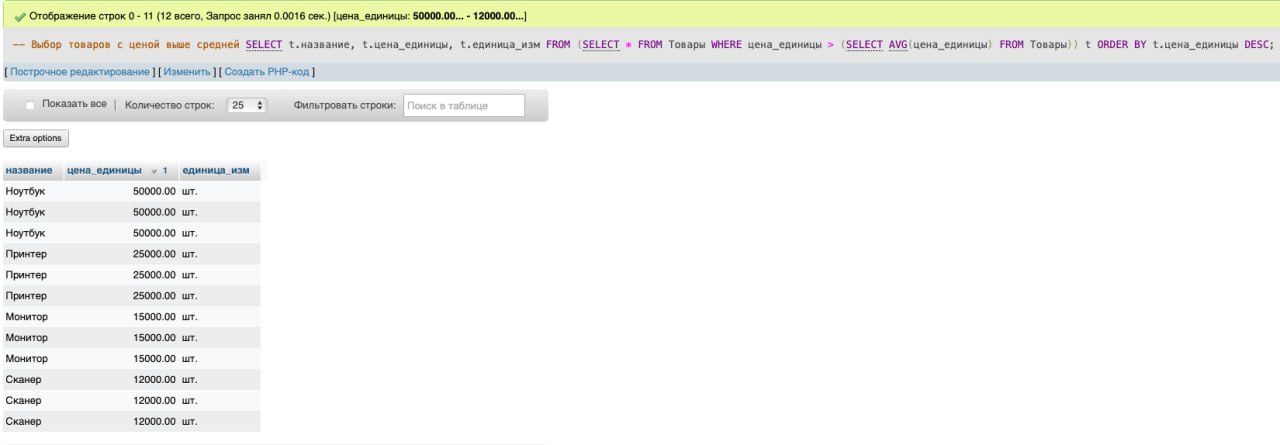
* Применяется после группировки (после GROUP BY).
* Используется для фильтрации групп данных, а не отдельных строк.
* Может содержать агрегатные функции, так как работает с результатами группировки.
* Используется только в запросах с GROUP BY или агрегатными функциями.

**Лабораторная работа № 12**

## ИСПОЛЬЗОВАНИЕ ПОДЗАПРОСОВ В ПРЕДЛОЖЕНИИ FROM

***Цель работы****:* Научиться создавать и применять подзапросы в качестве исходных наборов данных SQL-запроса.

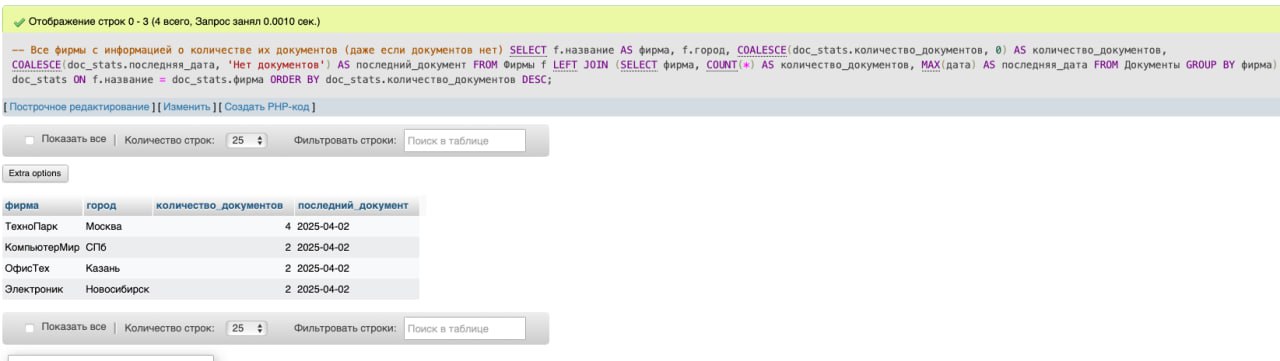
**12.1** Напишите простой запрос с подзапросом в предложении FROM.

запрос выводит названия, цены и единицы измерения товаров, чья цена выше средней по всей таблице, отсортированные от самой дорогой к более дешевой.

**12.2** Напишите запрос с внешним соединением таблицы с подзапросом в

предложении FROM.

* Агрегировать документы по фирмам (подсчитать количество и найти последнюю дату).
* Присоединить данные к таблице фирм через LEFT JOIN, заменив NULL на нули/текст, и отсортировать результат по убыванию количества документов.



Контрольные вопросы

1. Можно ли считать подзапрос в предложении FROM полноценным запросом?

Да, подзапрос в предложении FROM можно считать полноценным запросом. Он выполняется как самостоятельный запрос, и его результат используется как временная таблица (или представление) в основном запросе. Подзапрос в FROM может содержать все элементы обычного SQL-запроса, такие как SELECT, FROM, WHERE, GROUP BY, HAVING, и даже другие подзапросы.

1. Можно ли использовать в основном запросе поля таблицы подзапроса, не указанные в подзапросе после слова SELECT?  
   Нет, в основном запросе можно использовать только те поля, которые явно указаны в подзапросе после SELECT. Подзапрос в FROM создает временную таблицу, и только те столбцы, которые были выбраны в подзапросе, доступны в основном запросе.

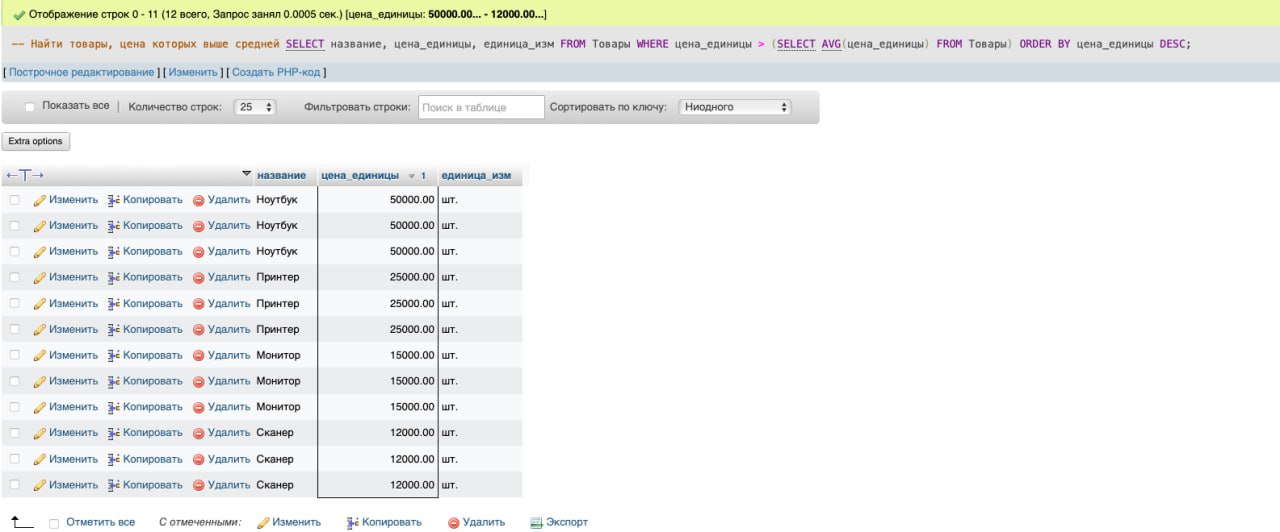
**Лабораторная работа № 13**

## ИСПОЛЬЗОВАНИЕ ПОДЗАПРОСОВ В ПРЕДЛОЖЕНИИ WHERE

***Цель работы****:* Научиться создавать и применять использовать подзапросы в условиях отбора (фильтрации) записей.

**13.1** Создайте простой запрос с подзапросом в предложении WHERE.

выводятся названия, цены и единицы измерения товаров, которые дороже среднего по таблице, в порядке убывания цены.



**13.2** Создайте запрос с подзапросом и внутренним соединением таблицы

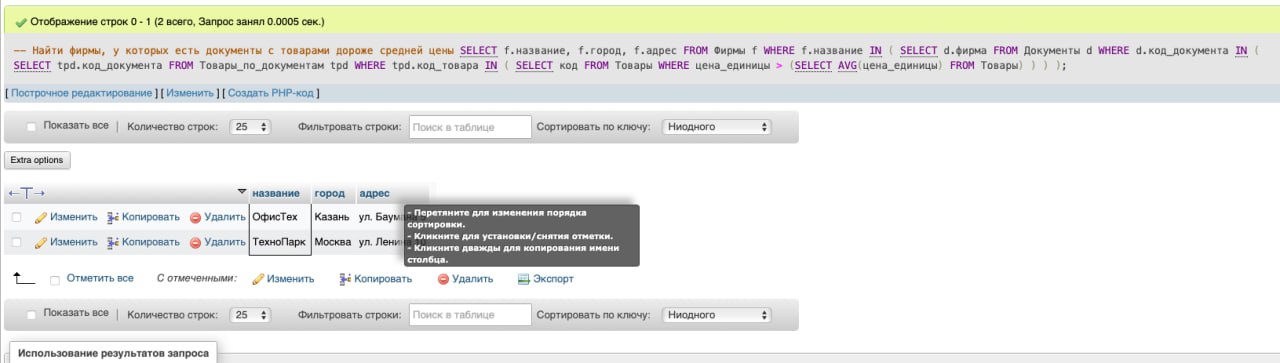
в предложении WHERE.

запрос возвращает код документа, дату и фирму для всех документов, содержащих хотя бы один товар дороже 10 000 рублей, отсортированные по убыванию даты. Это помогает быстро находить «дорогие» заказы или поставки.



**13.3** Создайте запрос с подзапросом, содержащим подзапрос.

* Возвращает **название, город и адрес фирм**, у которых есть хотя бы один документ с товарами дороже средней цены.
* Использует **вложенные подзапросы** для фильтрации данных:
  + От товаров → к документам → к фирмам.
* **Не учитывает дубликаты**: если у фирмы несколько документов с дорогими товарами, она будет выведена один раз.



### Контрольные вопросы

1. Возможно ли использования в подзапросе той же таблицы, которая используются в основном запросе.

Да, это возможно. Такой подзапрос называется коррелированным подзапросом. В коррелированном подзапросе таблица из основного запроса используется в подзапросе, и подзапрос выполняется для каждой строки основного запроса. Коррелированные подзапросы часто используются, когда нужно сравнить данные в текущей строке основного запроса с агрегированными или отфильтрованными данными из той же таблицы.

1. В каких случаях результат выполнения подзапроса в предложении WHERE можно сравнивать с другим значением при помощи операторов >, <, >=, <=?

Результат подзапроса в предложении WHERE можно сравнивать с другим значением при помощи операторов >, <, >=, <= в следующих случаях:

1. Подзапрос возвращает одно значение (скалярный подзапрос):

Подзапрос должен возвращать только одно значение (одну строку и один столбец).

Если подзапрос возвращает несколько строк или столбцов, это вызовет ошибку.  
Подзапрос SELECT AVG(ЦенаЕдиницы) FROM [2Товары] возвращает одно значение (среднюю цену), которое можно сравнить с ЦенаЕдиницы.

1. Подзапрос используется с агрегатными функциями:

Если подзапрос использует агрегатные функции (SUM, AVG, MIN, MAX, COUNT), он обычно возвращает одно значение, которое можно сравнивать.

1. Подзапрос используется с операторами ANY, ALL, SOME:

Если подзапрос возвращает несколько значений, его можно сравнивать с помощью операторов ANY, ALL или SOME. Эти операторы позволяют сравнивать значение с каждым элементом результата подзапроса.

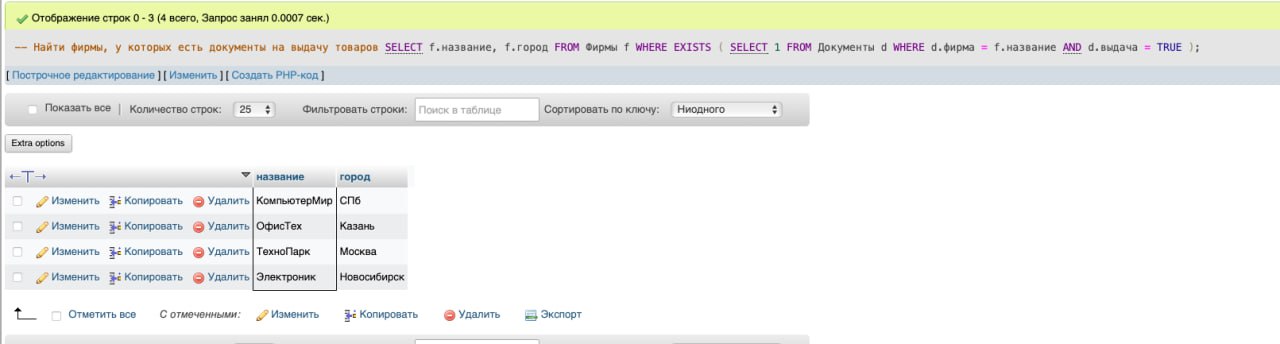
**Лабораторная работа № 14**

## ПРИМЕНЕНИЕ ОПЕРАТОРОВ EXISTS, ALL, SOME (ANY)

***Цель работы****:* Изучить дополнительные возможности подзапросов, возвращающих единичные и множественные значения.

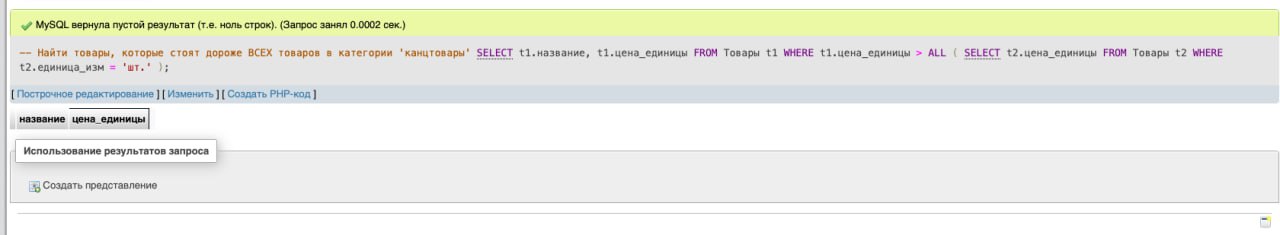
**14.1** Напишите запрос, использующий в условии отбора записей предложение EXISTS.

Этот SQL-запрос находит **фирмы, у которых есть документы на выдачу товаров**, проверяя через WHERE EXISTS наличие хотя бы одного документа с условием выдача = TRUE для каждой фирмы, что эффективно отсекает компании без таких документов.



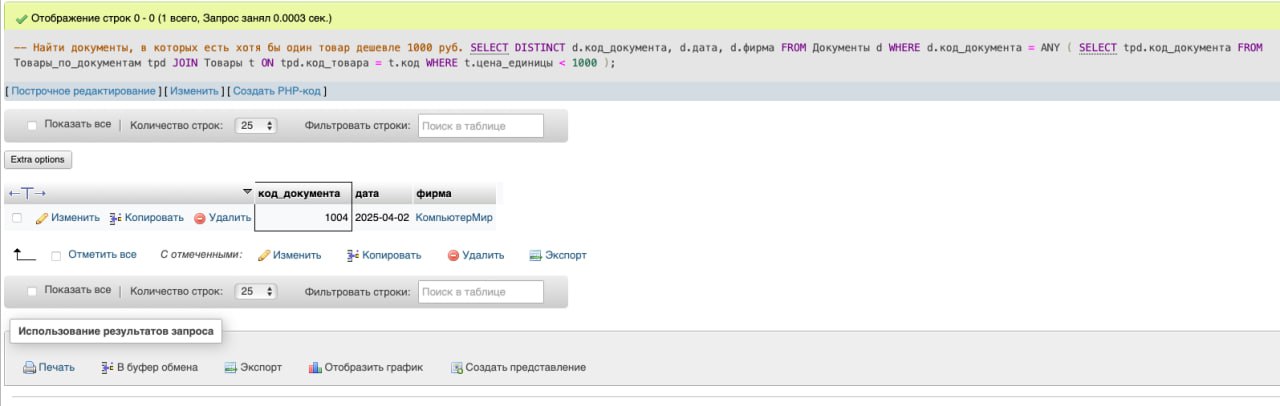
**14.2** Напишите запрос, использующий в условии отбора записей предложение ALL.

запрос выводит товары, которые дороже **всех товаров, измеряемых поштучно**,



**14.3** Напишите запрос, использующий в условии отбора записей предложение SOME (ANY).

Этот SQL-запрос находит документы, содержащие **хотя бы один товар дешевле 1000 рублей**



### Контрольные вопросы

1. Как создать запрос, отображающий только те записи, для которых подзапрос возвращает одно или более значений, не используя предложения EXISTS?

Для создания запроса, который отображает только те записи, для которых подзапрос возвращает одно или более значений, можно использовать операторы IN, SOME или ANY. Эти операторы позволяют проверить, соответствует ли значение в основном запросе хотя бы одному значению, возвращаемому подзапросом.

1. Можно ли написать работающий запрос, аналогичный запросу с ALL, но использующий SOME или ANY?

Да, можно написать запрос, аналогичный запросу с ALL, но использующий SOME или ANY. Однако, важно понимать, что ALL и SOME/ANY имеют разные семантики:

ALL требует, чтобы условие выполнялось для всех значений, возвращаемых подзапросом.

SOME/ANY требует, чтобы условие выполнялось хотя бы для одного значения, возвращаемого подзапросом.

1. Можно ли написать работающий запрос, аналогичный запросу с EXISTS, но использующий SOME или ANY?

Да, можно написать запрос, аналогичный запросу с EXISTS, но использующий SOME или ANY. Однако, важно понимать, что EXISTS и SOME/ANY имеют разные семантики:

EXISTS проверяет, возвращает ли подзапрос хотя бы одну строку.

SOME/ANY сравнивает значение с каждым значением, возвращаемым подзапросом.

Этот запрос вернет товары, цена которых равна хотя бы одной цене из подзапроса, что не эквивалентно запросу с EXISTS. Однако, если подзапрос возвращает хотя бы одно значение, то условие с SOME/ANY будет выполнено.

## Лабораторная работа № 15

**ИСПОЛЬЗОВАНИЕ HAVING И АГРЕГАТНЫХ ФУНКЦИЙ ДЛЯ ВЛОЖЕННЫХ ПОДЗАПРОСОВ. ОПЕРАТОР UNION.**

***Цель работы****:* Освоить способы использования подзапросов в предложении HAVING и оператора UNION.

**15.1** Создайте не менее двух запросов с использованием подзапроса в предложении HAVING.

Запрос находит документы, где количество товаров превышает среднее значение по всем документам. Это полезно для выявления "нагруженных" документов или анализа распределения товаров.



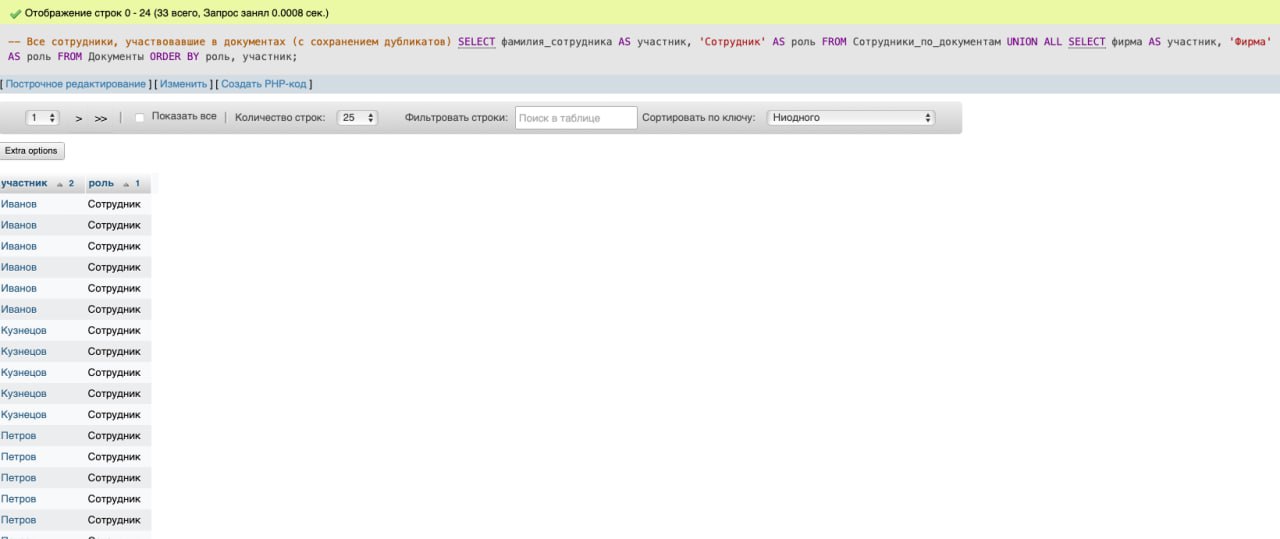
Соединяет таблицы **Документы**, **Товары\_по\_документам** и **Товары**, группирует данные по фирмам, вычисляя общую стоимость товаров для каждой фирмы.

Фильтрует результат через HAVING, оставляя только фирмы, чья общая стоимость превышает среднюю стоимость одного документа, рассчитанную через вложенный подзапрос.



**15.2** Создайте запрос с использованием оператора UNION.

Объединяет данные из таблиц **Товары** (с ценой > 1000) и **Фирмы** в единый список, добавляя метки 'Товар'/'Фирма' и NULL для цены фирм, после чего сортирует результат по типу (фирмы первыми) и названию. UNION автоматически удаляет полные дубликаты строк.



### Контрольные вопросы

1. Допустимо ли использование в подзапросах предложения GROUP BY без предложения HAVING?

Да, это допустимо. Предложение GROUP BY используется для группировки строк с одинаковыми значениями в одном или нескольких столбцах в одну строку. HAVING является дополнительным и используется для фильтрации этих сгруппированных строк на основе определенного условия. Вы можете использовать GROUP BY для группировки данных, а затем использовать результаты в подзапросе без необходимости фильтрации с помощью HAVING.

1. Допустимо ли использование в подзапросах предложения HAVING без предложения GROUP BY?

Нет, это недопустимо. Предложение HAVING всегда должно использоваться вместе с GROUP BY. HAVING предназначено для фильтрации результатов, которые уже были сгруппированы предложением GROUP BY. Использование HAVING без GROUP BY приведёт к синтаксической ошибке.

1. Можно ли использовать в предложении HAVING подзапрос к нескольким таблицам?

Да, это возможно. В предложении HAVING можно использовать подзапросы, которые обращаются к нескольким таблицам. Главное, чтобы подзапрос возвращал скалярное значение (одно значение), которое затем можно использовать для сравнения в условии HAVING. Подзапрос может использовать JOIN или другие операции для доступа к данным из нескольких таблиц.

**Лабораторная работа № 16**

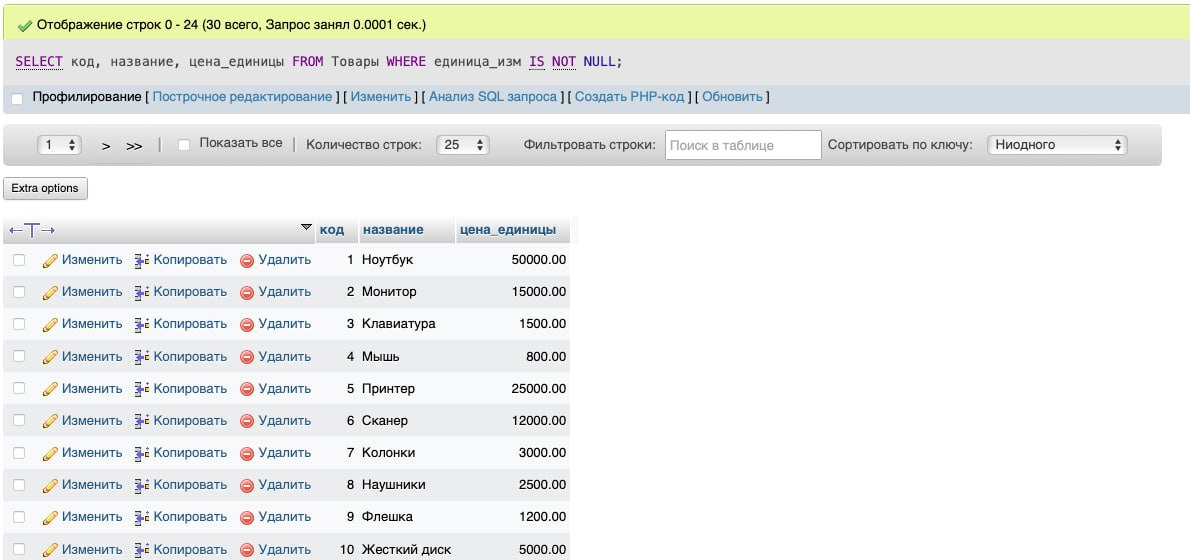
**IS NULL, IS NOT NULL.**

## РАБОТА С ТЕКСТОВЫМИ ДАННЫМИ В SQL-ЗАПРОСАХ

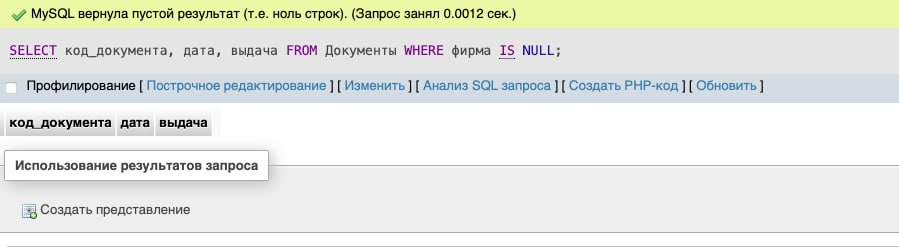
***Цель работы****:* Освоить составление запросов к таблицам, содержащим пустые ячейки. Освоить работу с текстовыми данными в запросах.

**16.1** Напишите запрос, использующий значения NULL и NOT NULL.

запрос возвращает список документов, не привязанных к какой-либо фирме, что помогает выявить незаполненные или ошибочные записи.

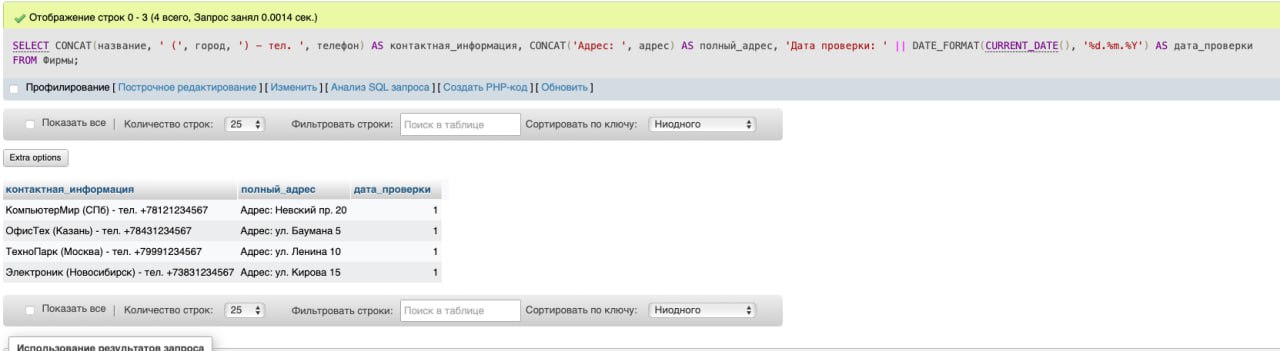


Запрос возвращает список товаров с заполненной единицей измерения (например, "шт.", "кг", "литр"), исключая записи с пропущенными данными в этом поле. Это помогает работать только с корректными и полными данными.



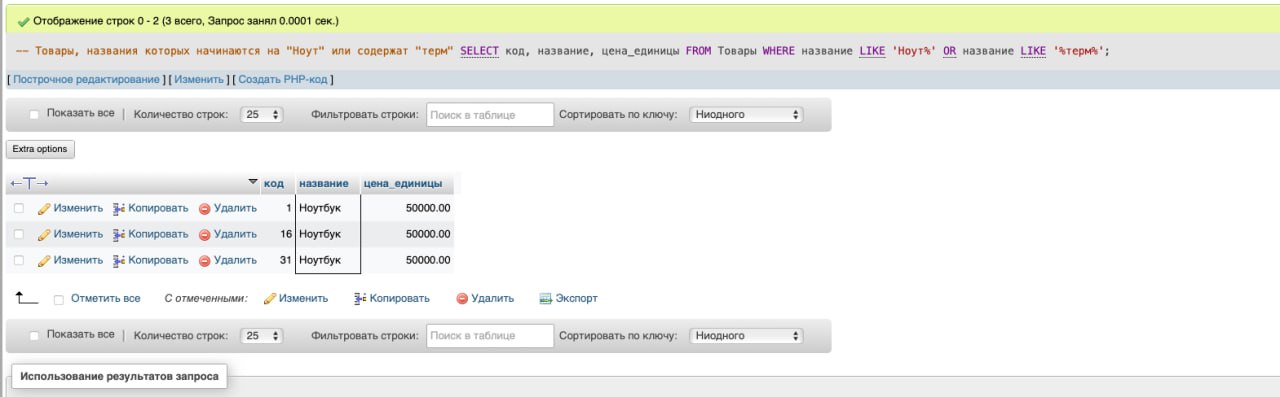
**16.2** Напишите запрос, использующий оператор или функцию сцепления строк. Данный запрос должен содержать хотя бы одну строковую константу.

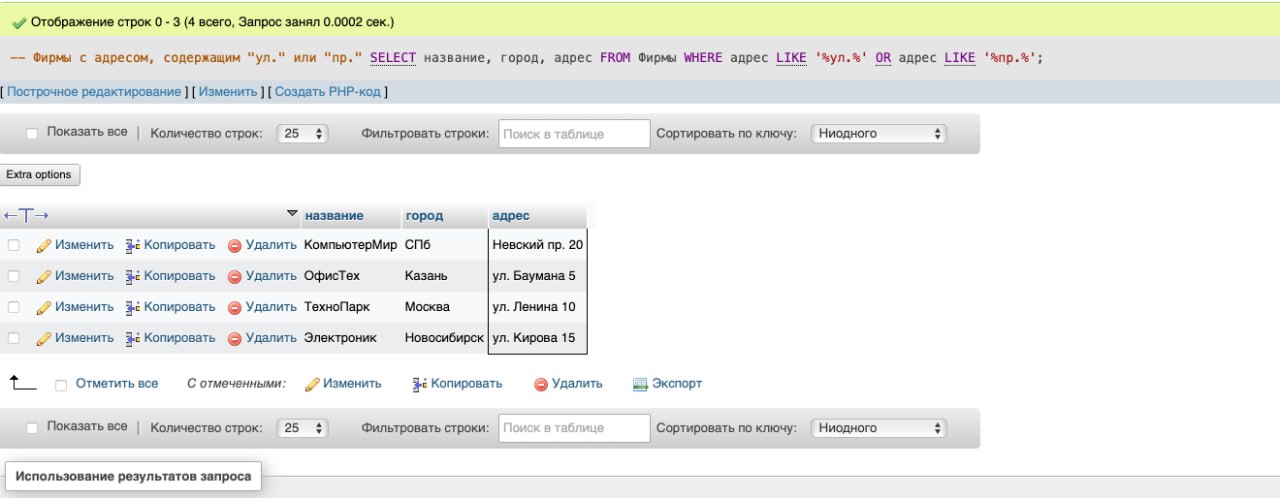
Объединяет данные таблицы **Фирмы** в форматированные строки: контактную информацию (название, город, телефон), полный адрес и текущую дату проверки, используя функции CONCAT и DATE\_FORMAT. Запрос выводит результат для всех записей без фильтрации.



Напишите 2 разных запроса, в которых используется предложение LIKE.

Выбирает товары, названия которых начинаются на «Ноут» **или** содержат «терм» в любой части, используя операторы LIKE с шаблонами 'Ноут%' и '%терм%', и выводит их код, название и цену.





### Контрольные вопросы

1. Как следует модернизировать запрос из методического указания к задаче 16.1, чтобы он выводил не только записи без значения в поле, но и записи со значениями «пустая строка»?

SELECT \*

FROM Документы

WHERE Фирма IS NULL

Модифицированный запрос  
SELECT \*

FROM Документы

WHERE Фирма Is Null OR Фирма = "";

1. Какие способы сцепления строк имеются в используемой вами СУБД?

Оператор &: самый распространенный способ. Просто объединяет две строки в одну.

Функция Str(): преобразует числовое значение в строковое, что может быть необходимо при объединении чисел со строками.

Использование оператора + (в некоторых случаях): в некоторых ситуациях + может работать для объединения строк, но поведение может быть непредсказуемым, особенно если операнды могут быть числовыми значениями. Поэтому рекомендуется использовать &.

1. Позволяет ли функция LIKE производить поиск строк без учета регистра символов?

По умолчанию функция LIKE в Microsoft Access выполняет поиск с учётом регистра. Однако можно выполнить поиск без учёта регистра следующими способами:

Использование функций UCase() или LCase(): позволяет преобразовать как столбец, так и шаблон поиска в верхний или нижний регистр перед сравнением.

Использование опции Database Compare Mode (требуется настройка базы данных): можно изменить параметры базы данных (на уровне БД) для выполнения сравнения без учета регистра.

Перейдите в меню "Файл" -> "Параметры".

Выберите "Общие".

Найдите параметр «Режим сравнения базы данных» (Database Compare Mode). Измените его на «Общий» (General).

Важно: изменение этого параметра может повлиять на другие запросы и сравнения в базе данных.

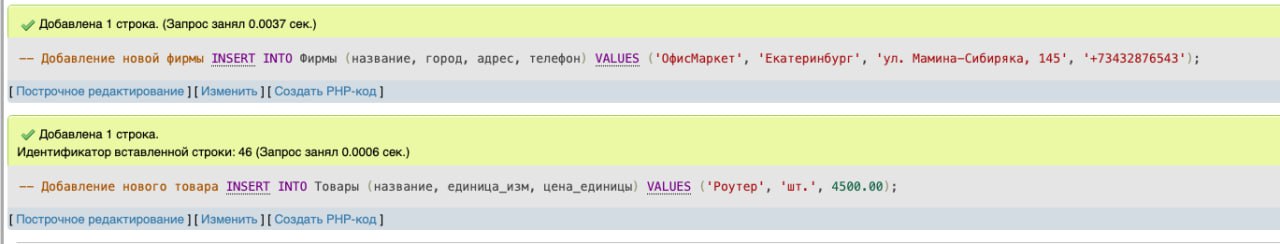
**Лабораторная работа № 17**

## ОПЕРАТОРЫ INSERT, UPDATE И DELETE

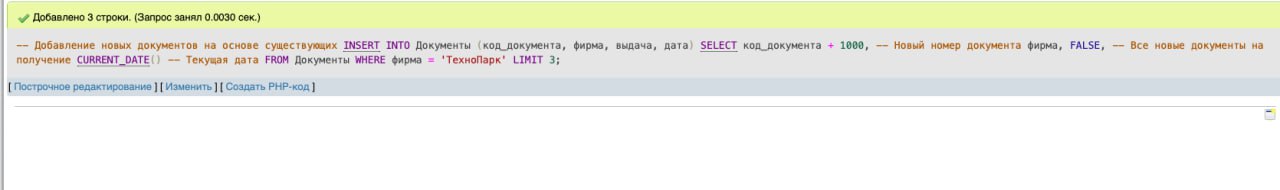
***Цель работы****:* Освоить способы изменения содержимого таблиц баз данных при помощи функций языка SQL.

**17.1** Создайте запрос для добавления данных в таблицу при помощи явного указания списка значений и при помощи оператора SELECT.

Оба INSERT добавляют новые записи: первый — фирму "ОфисМаркет" в таблицу **Фирмы** с указанием города, адреса и телефона, второй — товар "Роутер" в таблицу **Товары** с единицей измерения "шт." и ценой 4500.00.

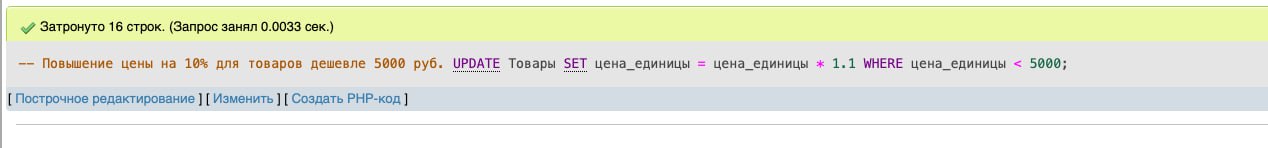


Вставляет 3 новых документа в таблицу **Документы**, используя данные существующих записей фирмы "ТехноПарк": увеличивает их код на 1000, устанавливает выдача = FALSE и текущую дату.



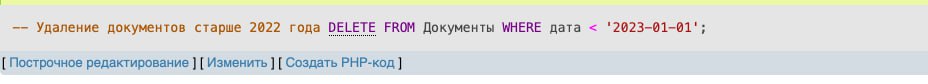
**17.2** Создайте запрос для изменения существующих в таблице данных.

Увеличивает цену на 10% для всех товаров, у которых текущая стоимость меньше 5000 руб., используя оператор UPDATE с условием WHERE цена\_единицы < 5000. Изменения применяются только к строкам, удовлетворяющим условию.



**17.3** Создайте запрос для удаления данных из таблицы.

Удаляет все записи из таблицы **Документы**, у которых дата раньше 1 января 2023 года (WHERE дата < '2023-01-01'), что соответствует документам за 2022 год и ранее.



### Контрольные вопросы

1. Возможно ли добавление записей при помощи оператора INSERT сразу из нескольких таблиц?

Да, возможно добавлять записи из нескольких таблиц с помощью оператора INSERT INTO, но для этого необходимо использовать оператор SELECT вместе с объединением таблиц (например, INNER JOIN, LEFT JOIN, RIGHT JOIN, FULL OUTER JOIN), чтобы сначала объединить данные из нескольких таблиц, а затем вставить их в целевую таблицу. Пример:

INSERT INTO Table3 (colA, colB)

SELECT t1.colX, t2.colY

FROM Table1 t1

INNER JOIN Table2 t2 ON t1.id = t2.t1\_id;

Этот запрос вставляет данные из Table1 и Table2 в Table3, объединяя их по связующим столбцам id16.

1. В каких случаях выполнение запроса UPDATE, при правильном указании имен таблиц и полей и при правильном синтаксисе может привести к ошибке?

Запрос UPDATE может привести к ошибке в следующих случаях:

Несовместимые типы данных: Если тип данных в поле, которое вы обновляете, не соответствует типу данных значения, которое вы пытаетесь вставить.

Нарушение ограничений целостности: Например, если вы пытаетесь обновить значение в поле с уникальным индексом на значение, которое уже существует в другой строке.

Отсутствие прав доступа: Если у пользователя нет разрешения на обновление таблицы или полей.

Неправильное указание имен таблиц или полей: Если в запросе указаны несуществующие таблицы или поля, даже при правильном синтаксисе.

1. Почему после слова DELETE не надо указывать имя полей таблицы?

После слова DELETE не нужно указывать имя полей таблицы, потому что оператор DELETE удаляет всю строку целиком, а не отдельные поля. Если вы хотите удалить только определенные поля, следует использовать оператор UPDATE и присвоить этим полям значение NULL, если это допустимо для этих полей