***Лабораторная работа №1 – Основы SQL (MySQL)***

В реляционных СУБД для выполнения операций над отношениями используются две группы языков, в качестве математической базы для которых используются теоретические языки запросов, предложенные Эдгаром Коддом:

* *реляционная алгебра*;
* *реляционное исчисление*.

В первом случае (реляционная алгебра) операнды и результаты всех действий являются отношениями. Такие языки — процедурные, поскольку отношение, которое является результатом запроса к базе данных, вычисляется при последовательном выполнении операторов, применяемым к отношениям. Сами операторы состоят из операндов (отношений) и реляционных операций (их результатом тоже является отношение).

Языки реляционного исчисления — непроцедурные, их называют *описательными* или *декларативными*. Они позволяют составлять запросы с помощью предиката первого порядка, которому должны удовлетворять кортежи или домены отношений. Таким образом, запрос к базе данных на таком языке содержит только информацию о желаемом результате. К языкам этой группы относится и SQL.

SQL (Structured Query Language, "язык структурированных запросов") — универсальный язык, применяемый для создания, модификации и управления данными в реляционных базах данных. В нем создается линейная последовательность операторов языка, которые выполняются СУБД. Операторы состоят из:

* имен операций и функций;
* имен таблиц и их столбцов;
* зарезервированных ключевых слов и специальных символов;
* логических, арифметических и строковых выражений.

Общий вид простого оператора в SQL:

ОПЕРАТОР параметры;

Если параметр не один, а несколько, то они перечисляются через запятую.

Все предложения на языке SQL оканчиваются точкой с запятой. Например:

**USE** `first\_lab\_data\_base`;

**SELECT** `id`, `field1` **FROM** `mytable`;

Выражения в SQL не зависят от регистра, не требуют обязательного наличия кавычек при обозначении названий, дополнительные разделители (пробел, табуляция, переход на новую строку) игнорируются. Для обозначения названий баз данных, таблиц, атрибутов таблиц, то есть названий, связанных с объектами СУБД, могут использоваться кавычки типа «тупое ударение» («`»). Например:

`элемент\_бд`

Для текстовых данных, вводимых пользователем в базу и не связанных с элементами СУБД (например, обычных строковых значений), используются обычные или двойные кавычки: 'текст', "большой текст". Чтобы с помощью двойных кавычек указать строку, которая уже содержит двойные кавычки в конце и начале выражения, нужно записывать ее так (экранирование):

"**\"**заголовок**\"**"

Существует общепринятый стиль «правильного» оформления выражений. Оно заключается в том, что при написании каких-либо выражений:

* после естественных разделителей выражений (например, запятых) ставится пробел;
* дополнительные разделители (пробелы, табы) не используются, если нет необходимости записать многостроковое выражение в удобном для чтения виде;
* системные обозначения (названия операторов, функций, ключевых слов и т.п.) пишутся заглавными буквами;
* при указании названий, связанных с объектами СУБД, обязательно используются кавычки в виде «тупого ударения».

Например:

**INSERT** **INTO** `news` (`id`, `post\_date`) **VALUES** (42, '2015-02-02 04:13:15');

Далее **будут использоваться квадратные скобки для обозначения дополнительных (но необязательных) параметров**. Выражения, с параметром в квадратных скобках и без него, являются синтаксически правильными.

Например:

**SELECT** \* **FROM** `table` [ **WHERE** условие ];

Такое выражение можно интерпретировать не только так:

**SELECT** \* **FROM** `table` **WHERE** условие;

Но и как аналогичное выражение без необязательных параметров, т.е.:

**SELECT** \* **FROM** `table`;

**Типы данных в MySQL**

В MySQL представлено множество типов данных, в соответствии с которыми хранятся и обрабатываются все данные в таблицах. Перечислим некоторые из них.

**Числа**

Числа разделяются на целые и дробные. Целые представлены следующими типами данных:

* TINYINT — 1 байт, т.е. принимает значения от -128 до 127 (в случае использования TINYINT UNSIGNED, т.е. без учета знака перед числом — 0..255);
* SMALLINT — 2 байта, -32768..32767 (0..65535);
* MEDIUMINT — 3 байта, -8388608..8388607 (0..2^24-1);
* INT — 4 байта, -2147483648..2147483647 (0..2^32-1);
* BIGINT — 8 байт, -2^32..2^32-1 (0..2^64).

Дробные числа представлены следующими типами данных:

* FLOAT (4 байта);
* DOUBLE (8 байт) — вдвое большая точность после запятой.

**Строки**

* CHAR — дополняет до заданной «ширины» (например, в случае использования CHAR(6) строка из пяти символов «hello» будет храниться как «hello », т.е. с пробелом на конце);
* VARCHAR — использует необходимый минимум памяти (в случае использования VARCHAR(6) строка из пяти символов «hello» будет храниться как «hello»);
* BLOB, TINYBLOB, MEDIUMBLOB, LONGBLOB — бинарные данные разных размеров;
* TEXT, TINYTEXT, MEDIUMTEXT, LONGTEXT — текстовые данные разных размеров;
* ENUM — одно из заданных значений (например, в ячейке типа ENUM(0,1,2) может храниться ноль, единица или двойка);
* SET — ноль или более заданных значений (в ячейке типа SET(0,1,2) может храниться любая комбинация из ноля, единицы и двойки — в том числе и пустое значение).

**Другие типы**

* булев: BOOL, BOOLEAN;
* уникальный автоматически увеличившийся идентификатор: SERIAL (== BIGINT UNSIGNED NOT NULL AUTO\_INCREMENT UNIQUE);
* дата и время: DATETIME, DATE, TIMESTAMP, TIME, YEAR.

**Работа с базами данных и таблицами**

**1. Команды для манипулирования базами данных**

1. Создание базы данных:

**CREATE** **DATABASE** `db\_name`;

Параметр команды создания баз данных — имя, выдаваемое создаваемой базе данных. Например, для создания базы данных под названием "my\_database" нужно ввести команду:

**CREATE** **DATABASE** `my\_database`;

Несмотря на то, что современная версия MySQL позволяет создавать БД с кириллическими и специальными символами в названии, принято использовать латинские буквы, цифры и знаки подчеркивания («\_»).

1. При одновременной работе в нескольких базах данных в командах нужно уточнять, с данными какой БД вы работаете. Для этого используется разделитель точка — «.». Так, чтобы обратиться к атрибуту "attribute" таблицы "table", находящейся в базе данных "database1", нужно использовать запись:

`database1`.`table`.`attribute`

Если же вам понадобится обратиться к аналогичному атрибуту такой же таблицы, находящейся в БД "database2", запись станет такой:

`database2`.`table`.`attribute`

Для того, чтобы вводимые команды применялись к конкретной базе данных по умолчанию, можно воспользоваться командой USE и ввести название базы данных, с которой мы будем в дальнейшем работать:

**USE** `my\_database`;

После выполнения команды USE следующие записи будут эквивалентны:

`my\_database`.`table1`

`table1`

1. Удаление существующей базы данных выполняется командой DROP DATABASE, которая в качестве единственного аргумента принимает название удаляемой базы данных. Например, чтобы удалить созданную вами в начале работы БД "my\_database", нужно выполнить:

**DROP** **DATABASE** `my\_database`;

После успешного удаления вы можете заново создать ее. **Учтите, что если бы в вашей базе данных были таблицы и данные, вся эта информация была бы утеряна навсегда.**

1. Для просмотра информации о базах данных, их таблиц, а также привилегий текущего пользователя, используется команда SHOW.
   1. Увидеть список всех доступных пользователю баз данных можно с помощью команды:

**SHOW** **DATABASES**;

* 1. Увидеть список всех таблиц в используемой базе данных можно с помощью команды:

**SHOW** **TABLES**;

* 1. Для вывода информации о таблице "table\_name" используйте команду:

**SHOW** **CREATE** **TABLE** `table\_name`;

Обратите внимание, что результатом выполнения команды будет команда создания таблицы с учетом всех изменений, произведенных над таблицей в процессе работы с базой данных.

* 1. Увидеть список всех прав текущего пользователя СУБД можно с помощью команды:

**SHOW** GRANTS;

**2. Команды для работы с таблицами БД**

1. Для **создания таблицы** используется команда CREATE TABLE. В качестве аргумента ей передается название новой таблиц и перечисление всех атрибутов таблицы и их описаний (типы данных, значения по умолчанию, ограничения на применяемые значения и т.п.) — все перечисление берется в круглые скобки, а разделителем между атрибутами служит запятая. В общем виде команда выглядит так:

**CREATE** **TABLE** `table\_name` (

`название\_первого\_поля` его\_тип [параметры],

`название\_второго\_поля` его\_тип [параметры],

...

`название\_последнего\_поля` его\_тип [параметры]

);

Обратите внимание, что после последнего атрибута запятая не требуется. Пример создания таблицы "news" с тремя атрибутами (идентификатор новости, время публикации, текст новости):

**CREATE** **TABLE** `news` (

`id` MEDIUMINT(8) **UNSIGNED** **PRIMARY** **KEY** **NOT** **NULL** **AUTO\_INCREMENT**,

`posted` TIMESTAMP **NOT** **NULL**,

`content` TEXT

);

2. Для **удаления таблицы** используется команда DROP TABLE с названием таблицы в качестве единственного аргумента. Например, для удаления созданной таблицы "news" команда будет выглядеть так:

**DROP** **TABLE** `news`;

3. Для **изменения таблиц** используется команда ALTER TABLE. Вид производимого изменения определятся последующими дополнительными командами:

3.1. **Переименование** таблицы осуществляется с помощью подкоманды RENAME. Например, чтобы переименовать таблицу "news" в "news\_new", нужно выполнить следующую команду:

**ALTER** **TABLE** `news` **RENAME** **TO** `news\_new`;

3.2. Для **добавления нового атрибута в таблицу** потребуется подкоманда ADD COLUMN, для которой нужно ввести название нового атрибута и указать его тип. Например, добавление к таблице "news" нового атрибута "author" (имя автора) будет выглядеть следующим образом:

**ALTER** **TABLE** `news` **ADD** **COLUMN** `author` VARCHAR(42);

Кроме того, можно задать положение добавляемого поля. Для этого в конец команды добавляется инструкция, указывающая, после какого столбца будет добавлено новое поле. Например, команда добавления атрибута "author" в таблицу "news" с тем, чтобы "author" стал вторым полем, выглядит следующим образом:

**ALTER** **TABLE** `news` **ADD** **COLUMN** `author` VARCHAR(42) AFTER `id`;

Чтобы поле стало первым в таблице, нужно заменить конструкцию с "AFTER ..." на ключевое слово "FIRST". Команда добавления атрибута "author" в таблицу "news" в качестве первого поля будет выглядеть так:

**ALTER** **TABLE** `news` **ADD** **COLUMN** `author` VARCHAR(42) FIRST;

3.3. Для **изменения типа атрибута таблицы** служит подкоманда MODIFY, для которой нужно указать таблицу, название атрибута и заново перечислить все требуемые для него параметры. Например, чтобы изменить тип атрибута "author" таблицы "news" на CHAR(42), нужно выполнить следующую команду:

**ALTER** **TABLE** `news` **MODIFY** **COLUMN** `author` CHAR(42);

3.4. Для **удаления атрибута из таблицы** служит подкоманда DROP COLUMN. Например, команда удаления атрибута "author" из таблицы "news" выглядит так:

**ALTER** **TABLE** `news` **DROP** **COLUMN** `author`;

**3. Команды для работы с данными таблиц баз данных**

Все команды из двух предыдущих подразделов («Команды для манипулирования базами данных» и «Команды для работы с таблицами БД») относятся к языку DDL (Data Definition Language), являющемуся частью языка SQL, которая обеспечивает работу со схемой БД. Другая, не менее важная, часть SQL — язык DML (Data Manipulation Language), позволяющий манипулировать самими данными в таблицах баз данных. К нему относятся команды, речь о которых пойдет уже в этом подразделе.

1. Для **добавления строк в таблицу** используется команда INSERT. В качестве аргументов ей передается название таблицы и набор всех значений для одной из строк или набор из названий атрибутов и соответствующих им значений, подразумевая, что значения остальных атрибутов будут заполнены автоматически.

Пример добавления строки в таблицу "news" с 4 атрибутами ("id", "posted", "content", "author") с указанием всех значений:

**INSERT** **INTO** `news` **VALUES** (1, NOW(), "Текст новости", "Редактор");

Использованное в запросе выражение "NOW()" — это обращение к встроенной в MySQL функции, которая возвращает текущее время, Таким образом, в качестве значения "posted" для добавляемой строки запишется текущее время.

Пример добавления строки в таблицу "news" с 4 атрибутами ("id", "posted", "content", "author") с указанием только некоторых значений:

**INSERT** **INTO** `news` (`posted`, `content`) **VALUES** (NOW(), "Текст новости");

В данном случае для добавляемой строки мы задали только значения атрибутов "posted" и "content". В поле "id" автоматически запишется число, которое будет на единицу больше последнего значения "id" в таблице (или 1, если добавляемая строка — первая), а в поле "author" для этой строки запишется значение NULL — отметка об отсутствии значения (**не путайте ее с нулевым значением**).

2. Для **изменения данных в строке таблицы** используется команда UPDATE. В качестве аргументов ей передаются название таблицы, названия атрибутов, значения которых будут изменены, и новые значения этих атрибутов.

Например, для того, чтобы изменить значение поля "author" во всех строках таблицы "news" на "Пользователь", а значение поля "content" — на "Пустой текст", нужно выполнить следующую команду:

**UPDATE** `news` **SET** `author` = "Пользователь", `content` = "Пустой текст";

В данном случае будут изменены все строки таблицы, но зачастую нужно изменять значения только в определенных строках. Для этого предусмотрена возможность установления условия отбора тех строк (WHERE), для которых будут проведены изменения.

Например, чтобы теперь изменить значение поля "content" на "Новый текст" только для строки с "id", равным 1, в таблице "news", нужно выполнить следующую команду:

**UPDATE** `news` **SET** `content` = "Новый текст" **WHERE** id = 1;

Правила формирования условий запросов подробно описаны в следующем подразделе — «Команда выборки».

3. Для **удаления данных из таблицы** используется команда DELETE. В качестве аргумента ей передается только название таблицы. Кроме того, можно ограничить список удаляемых строк, задав условие с помощью WHERE (по аналогии с тем, как это делалось в команде UPDATE) — тогда СУБД выберет строки, соответствующие условию, и удалит из таблицы только их. Без указания условия будут удалены все строки из таблицы.

Например, чтобы удалить из таблицы "news" строку с "id", равным 1, нужно выполнить следующую команду:

**DELETE** **FROM** `news` **WHERE** id = 1;

**Команда выборки**

**1. О выборке**

Для получения значений атрибутов таблиц БД используется команда SELECT. Общий вид команды представляется следующим образом:

**SELECT** `my\_field1`, `my\_field2`, ..., `my\_fieldN`

**FROM** `my\_table`

**WHERE** условие;

Здесь:

* my\_field1, my\_field2 и т.д. — это перечисление названий атрибутов, значения которых мы "выбираем", т.е. в результирующей таблице будут выведены только значения указанных атрибутов. В случае, если требуется вывод значений всех атрибутов результирующей таблицы, для упрощения записи запроса используется символ звездочки («\*»).
* my\_table — это название таблицы, из которой будет сделана выборка.
* условие WHERE может иметь сложную структуру или отсутствовать.

Например, для выборки всех значений всех атрибутов из таблицы "news" достаточно ввести следующую команду:

**SELECT** \* **FROM** `news`;

При выполнении запроса в оперативной памяти создается виртуальная таблица, состоящая из всех данных, удовлетворяющих условию отбора, а исходная таблица при этом никак не изменяется.

**2. Условия выборки**

Для решения некоторых задач условие отбора (WHERE) может иметь сложную структуру. Для построения таких условий используют логические операторы AND, OR, NOT и некоторые другие возможности языка SQL. Для группировки условий используются разделительные скобки («(» и «)»).

1. Для сравнения выражений предусмотрены: равенство («=»); больше или равно («>=»), меньше или равно («<=»), не равно («<>» или «!=»). Например, выборка значений атрибута "content" из таблицы "news", в строках которой значение атрибута "id" меньше 42, осуществляется следующим образом:

**SELECT** `content` **FROM** `news` **WHERE** `id` < 42;

1. Логическое умножение (И) записывается как AND и используется, когда требуется одновременное выполнение двух и более условий. Пример выборки значений атрибута "content" из таблицы "news", в строках которой значение атрибута "id" больше 21 и меньше 42:

**SELECT** `content` **FROM** `news` **WHERE** `id` > 21 **AND** `id` < 42;

1. Логическое сложение (ИЛИ) записывается как OR и используется, когда требуется, чтобы выполнялось хотя бы одно из нескольких условий. Пример выборки значений атрибута "content" из таблицы "news", в строках которой значение атрибута "id" больше 21 или меньше 10:

**SELECT** `content` **FROM** `news` **WHERE** `id` > 21 **OR** `id` < 10;

Логическое сложение имеет меньший приоритет чем логическое умножение, поэтому для корректной записи логических формул может потребоваться использование разделяющих скобок. Пример выборки значений атрибута "content" из таблицы "news", в строках которой значение атрибута "id" либо больше 21 и меньше 42, либо больше 84:

**SELECT** `content` **FROM** `news` **WHERE** ( `id` > 21 **AND** `id` < 42 ) **OR** `id` > 84;

1. Логическое отрицание (НЕ) записывается как NOT и используется для инвертирования последующего условия. Например, выборку значений атрибута "content" из таблицы "news", в строках которой значение атрибута "id" меньше 21 или больше 42, можно осуществить так:

**SELECT** `content` **FROM** `news` **WHERE** **NOT** ( `id` >= 21 **AND** `id` <= 42 );

1. Если у атрибута отсутствует значение, то оно записывается как NULL *(NULL — символ отсутствия значения, что не путать с пустым значением: пустое значение существует, а NULL указывает на его отсутствие)*. Для составления условий на выборку подобных отсутствующих значений предусмотрено специальное выражение IS NULL. Пример выборки значений атрибута "content" из таблицы "news", в строках которой значение атрибута "author" отсутствует:

**SELECT** `content` **FROM** `news` **WHERE** `author` **IS** **NULL**;

1. Для указания принадлежности значения атрибута какому-либо интервалу предусмотрено выражение BEETWEEN .. AND. Выражение "BETWEEN a AND b". Пример выборки значений атрибута "content" из таблицы "news", в строках которой значение атрибута "id" больше 21 и меньше 42:

**SELECT** `content` **FROM** `news` **WHERE** `id` **BETWEEN** 21 **AND** 42;

1. Для указания принадлежности значения атрибута какому-либо множеству предусмотрено выражение IN. Пример выборки значений атрибута "content" из таблицы "news", в строках которой значение атрибута "id" принимает значения 21, 42, 84 или 168:

**SELECT** `content` **FROM** `news` **WHERE** `id` **IN** (21, 42, 84, 168);

1. Для поиска строковых значений, содержащих заданную строку по шаблону, предусмотрена выражение LIKE. В качестве аргумента оператору LIKE передается шаблон в виде строки, в которой помимо текста могут содержаться метасимволы «\_» (обозначает любой одиночный символ) и «%» (набор любых символов любой длины). Пример выборки значений атрибута "content" из таблицы "news", в строках которой значение атрибута "author" соответствует шаблону «\_user%» (т.е. строка, которая строится как любой символ + user + любая последовательность символов):

**SELECT** `content` **FROM** `news` **WHERE** `author` **LIKE** "\_user%";

Такому шаблону будут удовлетворять значения атрибута вроде «1user», «1user222», «xuser42» и т.д.

**3. Сортировка и ограничение результатов**

Для **ограничения количества результирующих строк**, удовлетворяющих условию отбора, используется выражение LIMIT, которое дописывается в конец запроса с максимальным числом строк для вывода в качестве простейшего аргумента.

Пример для выборки информации о 5 первых новостях из таблицы "news" со значениями атрибута "id" больше 5:

**SELECT** \* **FROM** `news` **WHERE** `id` > 5 **LIMIT** 5;

Для **сортировки результата отбора** предусмотрен оператор ORDER BY. Он позволяет сортировать строки в результирующей таблице в соответствии со значениями выбранных атрибутов (если атрибутов несколько, то приоритет сортировки убывает при перечислении атрибутов слева направо). Поддерживается два типа сортировки: по возрастанию (ASC, этот режим используется по умолчанию) и по убыванию (DESC).

Пример для выборки информации о 10 первых новостях из таблицы "news" со значениями атрибута "id" больше 5, отсортированной сначала по авторам ("author") в алфавитном порядке, а затем — по идентификаторам ("id") в обратном порядке:

**SELECT** \* **FROM** `news` **WHERE** `id` > 5 **ORDER** **BY** `author` **ASC**, `id` **DESC** **LIMIT** 10;

**4. Числовые операции (агрегирующие функции)**

1. Для получения среднего арифметического значения атрибута по всем результирующим строкам предусмотрена функция AVG(). В качестве единственного аргумента ей передается название атрибута, значения которого будут учитываться. Пример выборки для нахождения среднего значения идентификатора ("id") новости:

**SELECT** AVG(`id`) **FROM** `news`;

1. Для поиска минимального и максимального значений атрибута среди всех результирующих строк есть функции MIN() и MAX() соответственно. В качестве единственного аргумента им передается название атрибута, значения которого будут учитываться. Пример выборки для нахождения минимального и максимального значений идентификатора ("id") новости:

**SELECT** MIN(`id`), MAX(`id`) **FROM** `news`;

1. Для подсчета суммы числовых значений атрибута среди всех результирующих строк предусмотрена функция SUM(). В качестве единственного аргумента ей передается название атрибута, значения которого будут учитываться. Пример выборки для нахождения суммы значений идентификаторов ("id") всех новостей:

**SELECT** SUM(`id`) **FROM** `news`;

1. Для подсчета количества строк выборки предусмотрена функция COUNT(). В качестве единственного аргумента ей передается название атрибута, значения которого будут учитываться. Кроме того, поскольку обычно не важно, по какому атрибуту считать число результирующих строк, вместо названия атрибута в качестве аргумента может использоваться метасимвол звездочки («\*»). Пример выборки для нахождения количества строк в таблице "news":

**SELECT** COUNT(\*) **FROM** `news`;

Дополнительный параметр DISTINCT, указанный перед названием атрибута, позволяет вычесть из результата все вхождения по повторяющимся значениям выбранного поля. Пример выборки для нахождения количества новостей, написанных разными авторами:

**SELECT** COUNT(**DISTINCT** `author`) **FROM** `news`;

**5. Группировка результатов**

Для разбиения результатов выборки по группам используется оператор GROUP BY. Разбиение строк по группам бывает необходимо для проведения каких-либо операций не над всеми строками по отдельности, а над группами, отобранными по какому-либо атрибуту и условию.

Группировка производится по указываемому атрибуту (или набору атрибутов), и строки распределяются по группам в соответствии со значением атрибута в этой строке (каждую группу образует набор строк с одним значением атрибута, по которому производится группировка).

Пример выборки значений всех атрибутов таблицы "news" с группировкой по атрибуту "author":

**SELECT** \* **FROM** `news` **GROUP** **BY** `author`;

С помощью группировки можно, например, посчитать количество новостей, добавленных каждым автором:

**SELECT** `author`, COUNT(\*) **FROM** `news` **GROUP** **BY** `author`;

Для добавления условий на результат группировки в конструкцию GROUP BY добавляют выражение HAVING, работающее по аналогии с WHERE, но с группами строк.

Например, к прошлому запросу с помощью HAVING можно добавить условие, по которому будут выводиться только те авторы, которые добавили более 3 новостей:

**SELECT** `author`, COUNT(\*) **AS** `cnt` **FROM** `news` **GROUP** **BY** `author` **HAVING** `cnt` > 3;

**Задания к лабораторной работе №1**

* Все запросы выполняются во вкладке SQL утилиты phpMyAdmin.
* Для сохранения запросов используйте закладки (метки):



* Выполните экспорт БД в формате sql:



* Подготовьте в электронном виде отчет, содержащий код (или принстскрины) всех запросов. Формат отчета: Назначение запроса – сам запрос.

***Пример варианта:***

*Создать БД с таблицей «Книги», в которой есть такие поля: Автор книги, название, год издания, цена, количество экземпляров, краткая аннотация.*

**В каждом варианте:**

* Создать таблицу согласно номеру варианта. При необходимости возможно добавление дополнительных полей.

***Номер варианта соответствует последней цифре в порядковом номере в группе. (например, 18 номер в группе – 8 вариант, 10 – 0 вариант);***

* Таблицу заполнить 10 записями (на SQL);
* Самостоятельно подготовить запросы:
  1. На изменение значения всех (или конкретного) полей (например, повышение цены на 99.99 руб)
  2. На удаление записи по заданному условию (например, удалить все книги 2008 года издания)
  3. **2 запроса** - Выборка данных с заданным условием с использованием операторов AND, OR, NOT, BETWEEN..AND, IN (например, выборка книг, стоимость которых не превышает 500 рублей, выборка книг Автора1, Автора2, Автора3)
  4. Поиск по строковому значению (например, найти все книги о магии)
  5. **2 запроса** - Выборка данных с использованием агрегирующих функций + с заданным условием (например, поиск самой дорогой книги Автора1)
  6. Группировка с использованием условия (например, группировка книг по издательству с указанием количества книг)

**Варианты**

|  |  |
| --- | --- |
| #варианта | Задание |
| 0 | Создать БД с таблицей «Оптовая база», в которой есть такие поля: Название товара, количество на складе, единица измерения, стоимость единицы товара, примечания - описание товара; |
| 1 | Создать БД с таблицей «Сеть магазинов», в которой есть такие поля: ФИО, адрес, телефон владельца магазина, номер регистрации, дата регистрации; |
| 2 | Создать БД с таблицей «Деканат», в которой есть такие поля: Название группы, курс, количество студентов, ФИО старосты; |
| 3 | Создать БД с таблицей «Поликлиника», в которой есть такие поля: Фамилия, имя, отчество, дата рождения пациента, лечащий врач; |
| 4 | Создать БД с таблицей «Спорт», в которой есть такие поля: ФИО спортсмена, год рождения, название команды, спортивный разряд; |
| 5 | Создать БД с таблицей «Сельскохозяйственные работы», в которой есть такие поля: Название сельскохозяйственного предприятия, дата регистрации, число работников, основной вид продукции, прибыль; |
| 6 | Создать БД с таблицей «Городской транспорт», в которой есть такие поля: Вид транспорта, номер маршрута, количество остановок в пути, количество машин на маршруте, количество пассажиров в день, стоимость проезда; |
| 7 | Создать БД с таблицей «География», в которой есть такие поля: Название страны, регион, столица, площадь территории, является ли страна развитой в экономическом отношении (да / нет); |
| 8 | Создать БД с таблицей «Домоуправление», в которой есть такие поля: Номер квартиры, номер дома, число жителей, площадь; |
| 9 | Создать БД с таблицей «Аэропорт» », в которой есть такие поля: Номер самолета, тип, число мест, скорость полета (число маха). |