**BC/NW 2017 № 2 (31):8.1**

**ЛАБОРАТОРНЫЙ ПРАКТИКУМ «БАЗЫ ДАННЫХ»**

**Борисова С.В.**

**ОГЛАВЛЕНИЕ**

Предисловие…………………………………………..3

Лабораторная работа 1……………………………..4

Методические указания к выполнению лабораторной работы №1. ……….5

Лабораторная работа 2………………………………. 12

Методические указания к выполнению лабораторной работы №2. …………13

Лабораторная работа 3 ……………………………….18

Методические указания к выполнению лабораторной работы №3. …………19

Лабораторная работа 4 ……………………………….22

Методические указания к выполнению лабораторной работы №4. ……………….22

СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ ……….25

# Предисловие

Лабораторные работы по курсу «Базы данных» выполняются в среде My SQL и MS SQL SERVER. Задачей курса "Базы данных" является освоение технологий работы с базами данных. В курсе 8 лабораторных работ.

В первой части пособия рассмотрены лабораторные работы №1-4 в среде My SQL. В первых 4 работах студенты осваивают различные технологии работы с базами данных на примере СУБД MySQL**.** Работы выполняются в интерактивном режиме и знакомят студентов с основными операциями по работе с базами данных (создание базы данных и таблиц, занесение данных, выполнение простейших операций над данными, формирование запросов на языке SQL)

В работах №5-6 студенты знакомятся с особенностями языка Transact-SQL. На примере MS SQL SERVER создают процедуры и функции. Изучают и создают индексы, представления и триггеры

Лабораторные работы №7-8 посвящены проектированию базы данных. В лабораторных работах выполняются шаги по проектированию базы данных, а также разработка собственно базы данных (таблиц, процедур, функций).

# Лабораторная работа 1

**СОЗДАНИЕ И ИЗМЕНЕНИЕ БАЗЫ ДАННЫХ И ТАБЛИЦ**

**Цель работы**

Ознакомиться с возможностями клиентской программы **MySQL**, которая представляет собой утилиту командной строки. Создать с ее помощью базу данных, набор таблиц в ней, заполнить таблицы данными для последующей работы, провести модификацию таблиц.

**Содержание работы.**

1. Для работы с базой данных вначале необходимо запустить сервер MySQL. После того как сервер MySQL запущен, к нему можно подключиться. Открыть окно командной строки. Вывести список имеющихся баз данных. Команда **SHOW DATABASES**.

2. Изучить набор команд языка SQL, связанный с созданием базы данных, созданием, модификацией структуры таблиц и их удалением, вставкой, модификацией и удалением записей таблиц.

**SHOW DATABASES, CREATE DATABASE, SELECT DATABASE(), CREATE TABLE, SHOW TABLES, DESCRIBE, ALTER TABLE, DROP TABLE, DROP DATABASE, INSERT INTO, UPDATE, DELETE**

3. Создание базы данных.

*Создать базу данных для хранения сведений о животных для ветеринарной клиники с именем* ***A\_ХХ\_ГГ\_YY*** *, (где XX-номер группы, ГГ-год группы, YY-номер в журнале).*

Убедитесь, что она создана (**SHOW DATABASES**). Сделайте ее текущей.

4. Создание таблиц в базе данных.

*Создать таблицу Животные с именем* ***A\_ХХ\_ГГ\_YY\_PetName***

*( XX-номер группы, ГГ-год группы, YY-номер в журнале) со структурой:*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| *Pet\_id (int) unsigned* *not null primary key* | *Pet\_name (varchar20)*  | *Owner (varchar15)* | *Ves (int)* | *PYear (date)* |

5. Сделать активной созданную БД ***A\_ХХ\_ГГ\_YY*** и посмотреть структуру созданной таблицы ***A\_ХХ\_ГГ\_YY\_PetName***.

6. Изменить структуру таблицы.

*Добавить столбец Vid (varchar10) (вид – собака, кошка, птица, змея и т.п.) cлева от PYear.*

*Переименовать (изменить) столбец PYear в столбец PetBirth.*

*Добавить столбец ID (int) слева от Pet\_id (первый).*

*Посмотреть структуру таблицы.*

*Удалить столбец ID.*

7. Внести данные в таблицу (7-9 записей).

*Используйте не менее 3-х различных видов и владельцев (owner) животных. Вводить дату в разном формате: как строка "2012-10-01" или как последовательность чисел 20121001.*

*Пример:*

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| *Pet\_id (int)* | *Pet\_name (varchar20)*  | *Owner (varchar15)* | *Ves**(int)* | *Vid**(varchar10)* | *PYear (date)* |
| *1* | *БобикФИО****YY*** | *ИвановФИОYY* | *250* | *собакаФИОYY* | *1980* |
| *2* | *БарбосФИО****YY*** | *ПетровФИОYY* | *230* | *собакаФИОYY* | *2005* |

8. Проверить результат заполнения таблиц, написав и выполнив простейший запрос: ***SELECT \* FROM имя\_таблицы***

9. Выполнить ряд изменений в таблице.

*Удалить данные из строки 2.*

*Добавить столбец Town (varchar5).*

*Изменить имя и тип столбца Town(alter table).*

*Переименовать столбец Ves.*

*Изменить значение Вес в строке 1*.

10. Проверить результаты выполнения изменений*. (SELECT)*

## Методические указания к выполнению лабораторной работы №1.

**База данных** –набор сведений, хранящихся упорядоченным образом. Мы рассматриваем реляционные базы данных, которые состоят из таблиц.

Для работы с базой данных необходимаСУБД (система управления базами данных), т.е. программа, которая берет на себя все заботы, связанные с доступом к данным. Она содержит команды, позволяющие создавать таблицы, вставлять в них записи, искать и даже удалять записи*.*

Взаимодействие с базой данных осуществляется на **языке**, называемом **SQL** (Structured Query Language — язык структурированных запросов).

MySQL - это открыто распространяемая СУБД, функционирует по модели "клиент/сервер", доступна для загрузки на сайте MySQL.com.

**В MySQL для каждой базы данных создается отдельный каталог, а каждой таблице соответствуют три файла.**

Клиентская программа MySQL представляет собой утилиту командной строки. Эта программа подключается к серверу по сети.

Для работы с базой данных, вначале необходимо **запустить сервер MySQL**. Если сервер MySQL был сконфигурирован как сервис Windows то запускать и останавливать его можно с помощью компонента Службы панели управления. После того как сервер MySQL запущен**, к нему можно подключиться**.

Откройте  **окно командной строки** Windows, введите команду **cmd**. Измените текущий каталог:

**cd C:\Program Files\MySQL\MySQL Server 5.1\** и запустите программу клиента mysq: **Mysql -u root –p,** введите пароль (root). Появится приглашение mysql для работы.

**Просмотр Баз данных.**

Для просмотра имеющихся баз данных существует команда **SHOW DATABASES.** Введите ее в строке приглашения и сервер выведет список баз данных, которые были созданы MySQL во время установки.

Примечание. Изначально в MySQL присутствуют только две базы данных: mysql и test. В базах данных information\_schema и mysql хранится информация об учетных записях, системный каталог привилегий пользователей СУБД MySQL, региональные настройки и т.д. База данных test является пустой и создается при установке MySQL вместе с системными базами данных information\_schema и mysql. База данных mysql является реальной базой данных, а information\_schema — виртуальной, именно поэтому в каталоге данных нет подкаталога с соответствующим именем.

**Выход из системы.**

Для выхода введите "**QUIT"** в приглашении mysql, снова появится приглашение MS-DOS. По окончании работы остановите сервер MySQL.

**Создание базы данных в Windows. Команда CREATE DATABASE**

Синтаксис команды **CREATE DATABASE** имеет вид:

***CREATE DATABASE [IF NOT EXISTS] имя\_базы\_данных***

***[спецификация\_create[,спецификация\_create]...]***

Команда CREATE DATABASE создает базу данных с указанным именем. Для использования команды необходимо иметь привилегию CREATE для базы данных. Если база данных с таким именем существует, генерируется ошибка.

Спецификация\_create:

 [DEFAULT] CHARACTER SET имя\_набора\_символов

 [DEFAULT] COLLATE имя\_порядка\_сопоставления

Опция спецификация\_сrеаtе может указываться для определения характеристик базы данных. Характеристики базы данных сохраняются в файле db.opt, расположенном в каталоге данных. Конструкция CHARACTER SET определяет набор символов для базы данных по умолчанию. Конструкция COLLATION задает порядок сопоставления по умолчанию.

Поскольку изначально в базе нет никаких таблиц, оператор **CREATE DATABASE** только создает подкаталог в каталоге данных MySQL.

(Примечание: Команда заканчивается символом точки с запятой).

Создадим базу данных с именем DB1.

**CREATE DATABASE DB1;**

Сервер ответит, что запрос обработан, изменилась 1 строка (0.00 сек). База данных была успешно создана.

**Определение текущей базы данных.**

База данных DB1 уже создана. Для работы с ней, необходимо её "активировать" или "выбрать". Всякий раз при работе с клиентом MySQL необходимо определять, **какая база данных будет использоваться**.

**Определить текущую базу** данных можно несколькими способами:

* при запуске: mysql> **MySQL DB1;**
* с помощью оператора **USE**: mysql> **USE DB1**;
* с помощью **\u**: mysql**> \u DB1**;

После определения базы данных вводим команду **SELECT DATABASE();** и видим нашу БД.

**Таблицы. Создание. Команда CREATE TABLE**

Базы данных хранят данные в таблицах. Проще всего таблицы можно представлять себе, как состоящие из строк и столбцов. Каждый столбец определяет данные определенного типа. Строки содержат отдельные записи.

Синтаксис команды **CREATE TABLE** таков:

***CREATE [TEMPORARY] TABLE [IF NOT EXISTS] имя***

***[(спецификация, ...)] [опция, ...]***

***[ [IGNORE | REPLACE] запрос]***

Флаг TEMPORARY задает создание временной таблицы, существующей в течение текущего сеанса. По завершении сеанса таблица удаляется. Временным таблицам можно присваивать имена других таблиц, делая последние временно недоступными. Спецификатор **IF NOT EXIST** подавляет вывод сообщений об ошибках в случае, если таблица с указанным именем уже существует. Имени таблицы может предшествовать имя базы данных, отделенное точкой. Если это не сделано, таблица будет создана в базе данных, которая установлена по умолчанию.

Разрешается создавать таблицы без столбцов, однако в большинстве случаев спецификация хотя бы одного столбца все же присутствует. Спецификации столбцов и индексов приводятся в круглых скобках и разделяются запятыми.

**Формат спецификации** следующий:

***имя тип[NOT NULL | NULL][DEFAULT значение]***

***[AUTO\_INCREMENT][KEY][ссылка]***

Спецификация типа включает название типа и его размерность. По умолчанию столбцы принимают значения NULL. Спецификатор NOT NULL запрещает подобное поведение.

У любого столбца есть значение по умолчанию. Если оно не указано, программа MySQL выберет его самостоятельно. Для столбцов, принимающих значения NULL, значением по умолчанию будет NULL, для строковых столбцов — пустая строка, для численных столбцов — нуль. Изменить эту установку позволяет предложение **DEFAULT**.

Поля-счетчики, создаваемые с помощью флага AUTO\_INCREMENT, игнорируют значения по умолчанию, так как в них записываются порядковые номера. Тип счетчика должен быть беззнаковым целым. В таблице может присутствовать лишь одно поле-счетчик. Им не обязательно является первичный ключ. Когда MySQL встречается со столбцом с атрибутом auto\_increment, то генерируется новое значение, которое на единицу больше чем наибольшее значение в столбце. Поэтому мы не должны задавать для этого столбца значения, MySQL генерирует их самостоятельно. Из этого также следует, что каждое значение в этом столбце будет уникальным.

Примечание: в MySQL команды и имена столбцов не различают регистр символов, однако имена таблиц и баз данных могут зависеть от регистра в связи с используемой платформой (как в Linux).

Ключевой столбец (primary key )необходим для того, чтобы исключить возможность совпадения данных. Если имеется столбец с уникальными значениями, то можно легко различить две записи. Лучше поручить присваивание уникальных значений самой системе MySQL

После создания таблицы убедимся, что она существует командой

**SHOW TABLES;**

**Просмотр структуры таблицы. DESCRIBE.**

Оператор **DESCRIBE ИмяТаблицы**;» позволяет посмотреть структуру таблицы:

**Изменение структуры таблицы. ALTER TABLE.**

Эта команда позволяет добавлять и удалять столбцы, создавать и уничтожать индексы, переименовывать столбцы и саму таблицу.

***ALTER TABLE table\_name alter\_spec;***

Основные преобразования, выполняемые оператором ALTER TABLE

* **ADD create definition [ FIRST | AFTER column\_name]** Добавление нового столбца. Параметр **create\_definition** представляет собой название нового столбца и его тип. Конструкция FIRST добавляет новый столбец перед столбцом column\_name. Конструкция AFTER добавляет новый столбец после столбца column\_name. По умолчанию столбец добавляется в конец таблицы.
* **ADD INDEX [index\_name] (index\_col\_name, ...)** Добавление индекса index\_name для столбца index\_col\_name. Если имя индекса index\_name не указывается, ему присваивается имя, совпадающее с именем столбца index\_col\_name.
* **ADD PRIMARY KEY (index\_col\_name, ...)** Делает столбец index\_col\_name или группу столбцов первичным ключом таблицы.
* **CHANGE old\_col\_name new\_col\_name type** Изменение столбца с именем **old\_col\_name** на столбец с именем **new\_col\_name** и типом **type**.
* **DROP col\_name** Удаление столбца с именем col\_name.
* **DROP PRIMARY KEY** Удаление первичного ключа таблицы.
* **DROP INDEX index\_name** Удаление индекса index\_name.

1. Добавление в таблицу new\_table нового столбца test с размещением его после столбца name можно выполнить следующим SQL-запросом

mysql> ***ALTER TABLE new\_table ADD test INT(10) AFTER name***;

2. Переименование столбца test в текстовый столбец new\_test можно осуществить следующим образом.

mysql> ***ALTER TABLE new\_table CHANGE test new\_test text;***

**Удаление таблиц. Команда DROP TABLE**

Для того, чтобы удалить таблицу, убедитесь сперва, что она существует. Это можно проверить с помощью команды SHOW TABLES.

Инструкция DROP TABLE имеет следующий синтаксис:

***DROP TABLE [IF EXISTS] таблица [RESTRICT | CASCADE***]

Спецификация IF EXISTS подавляет вывод сообщения об ошибке, выдаваемого в случае, если заданная таблица не существует. Можно указывать несколько имен таблиц, разделяя их запятыми.

Флаги RESTRICT и CASCADE предназначены для выполнения сценариев, созданных в других СУБД.

**Удаление баз данных. DROP DATABASE**

Команда удаляет базу данных со всеми таблицами, входящими в ее состав. Синтаксис команды:

**DROP DATABASE database\_name;**

**Запись данных в таблицу. Оператор INSERT**

Оператор **INSERT** заполняет таблицу данными.

***INSERT into table\_name (column1, column2, ...)values (value1, value2...);***

где table\_name является именем таблицы, в которую надо внести данные; column1, column2 … являются именами столбцов, а value1, value2 … являются значениями для соответствующих столбцов.

Несколько важных моментов:

* Значениями столбцов, которые являются текстовыми строками записываются в кавычках.
* Значение для столбца со свойством auto\_increment задает система MySQL, которая находит в столбце наибольшее значение, увеличивает его на единицу, и вставляет новое значение.

Если приведенная выше команда правильно введена в приглашении клиента mysql, то программа выведет сообщение об успешном выполнении. Создание дополнительных записей требует использования отдельных операторов INSERT. Чтобы облегчить эту работу можно поместить все операторы INSERT в файл. Это должен быть обычный текстовый файл с оператором INSERT в каждой строке.

**Ввод данных из файла. LOAD DATA LOAD DATA.**

Команда **LOAD DATA INFILE** читает строки из текстового файла и вставляет их в таблицу с очень высокой скоростью.

**LOAD DATA [LOW\_PRIORITY | CONCURRENT] [LOCAL] INFILE 'file\_name.txt'**

 **[REPLACE | IGNORE] INTO TABLE tbl\_name**

 **[FIELDS [TERMINATED BY '\t']**

 **[[OPTIONALLY] ENCLOSED BY ''] [ESCAPED BY '\\' ] ]**

 **[LINES TERMINATED BY '\n'] [IGNORE number LINES]**

 **[(col\_name,...)]**

Если задано ключевое слово LOCAL, то файл читается с клиентского хоста. Если же LOCAL не указывается, то файл должен находиться на сервере. Если текстовые файлы, которые нужно прочитать, находятся на сервере, то из соображений безопасности эти файлы должны либо размещаться в директории базы данных, либо быть доступными для чтения всем пользователям. Кроме того, для применения команды LOAD DATA INFILE к серверным файлам необходимо обладать привилегиями FILE для серверного хоста.

**Изменение записей. UPDATE**

Команда **UPDATE** выполняет **изменение данных** в таблицах. Она имеет очень простой формат.

***UPDATE имя\_таблицы SET имя\_столбца\_1 = значение\_1, имя\_столбца\_2 = значение\_2, имя\_столбца\_3 = значение\_3,***

***...[WHERE условия];***

Как и все другие команды SQL можно вводить ее на одной строке или на нескольких строках.

Предположим, в таблице Sotrudniki рабочим увеличили зарплату (поле Salary) на 1000 и надбавки (Premiya) на 500. Их предыдущая зарплата была 20000, а надбавки были 500.

***UPDATE Sotrudniki SET Salary=220000, Premiya=55000 WHERE title='рабочий';***

Можно воспользоваться арифметическими операторами. Следующий оператор сделает то же самое, при этом исходные данные знать заранее не требуется.

***UPDATE*** ***Sotrudniki SET Salary= Salary + 1000, Premiya= Premiya +500 WHERE title='рабочий'***;

В качестве другого примера можно попробовать изменить название должности "прораб" на "мастер".

***UPDATE Sotrudniki SET title='мастер' WHERE title='прораб';***

**Оператор UPDATE без условий изменит все данные столбца во всех строках**. Надо быть очень осторожным при внесении изменений.

**Удаление записей из таблицы. DELETE**

Оператор DELETE требует задания имени таблицы и необязательных условий. Если никакие условия не заданы, то удаляются все данные в таблице.

***DELETE from имя\_таблицы [WHERE условия];***

Предположим, сотрудник Иванов уволился из компании. Надо удалить его запись из таблицы Sotrudniki.

***DELETE from Sotrudniki WHERE firstname = 'Иванов';***

**Контрольные вопросы**

1. Как просмотреть список имеющихся баз данных?
2. Принципы физического хранения данных в СУБД MySQL?
3. Какова архитектура СУБД MySQL?
4. Для чего предназначен язык SQL?
5. Назовите команды для создания БД и таблиц.
6. Можно ли создать таблицу до создания базы данных?
7. Как изменить значение по умолчанию столбца?
8. Сколько полей -счетчиков, создаваемые с помощью флага AUTO\_INCREMENT, может быть в таблице?
9. Может ли поле счетчик не быть первичным ключом?
10. Какие типы данных допустимы при создании таблицы?
11. Как вставить строки данных в таблицу средствами SQL?
12. Как изменить строки таблицы средствами SQL?
13. Каким образом выполнить просмотр таблицы?
14. Как получить информацию о структуре таблицы?
15. Какими способами можно заполнить таблицу БД?

# Лабораторная работа 2

**ВЫБОР И МОДИФИКАЦИЯ ДАННЫХ ТАБЛИЦ**

**Цель работы**

Используя данные базы данных из первой лабораторной работы, подготовить и реализовать серию запросов, связанных с выборкой информации и модификацией данных таблиц.

**Содержание работы**

1. Изучить состав, правила и порядок использования ключевых фраз оператора SELECT:

**SELECT, FROM, WHERE, GROUP BY, HAVING, ORDER BY**

Порядок следования фраз в команде **SELECT** должен соответствовать приведенной выше последовательности.

3. Подготовить и выполнить запросы по выборке информации из таблицы базы данных для решения нижеприведенных задач.

1. *Выбрать любого хозяина и отобразить всех его питомцев.*
2. *Выбрать любой год и отобразить животных с годом рождения позже выбранного.*
3. *Вывести количество животных любого одного вида.*
4. *Вывести клички животных, имена хозяев которых содержат букву «А».*
5. *Вывести имена животных с весом, например, от 200 до 300г. (2 варианта).*
6. *Выбрать любого хозяина, найти у него животное с наименьшим весом и вывести запись об этом животном*
7. *Вывести имена животных, относящиеся к двум разным видам (использовать IN и Or).*
8. *Вывести записи, отсортированные по году рождения.*
9. *Подсчитать количество уникальных видов (distinct) животных.*
10. *Вывести имена животных, отсортированные в убывающем порядке.*
11. *Вывести название вида и количество животных каждого вида.*
12. *Вывести название вида и количество животных этого вида с весом, например, от 200 до 300г.*
13. *Вывести средний вес всех животных с использованием псевдо имени столбца «Средний вес.».*
14. *Удалить строку 5.*

Показать результаты работы преподавателю.

## Методические указания к выполнению лабораторной работы №2.

**Запрос данных из таблицы MySQL SELECT**

Запрос данных выполняется с помощью команды MySQL **SELECT**.

В общем виде синтаксис оператора SELECT имеет следующий вид:

***SELECT [ALL/DISTINCT] <список атрибутов>/\****

***FROM <список таблиц> [WHERE <условие выборки>]***

 ***[GROUP BY <список атрибутов>] [HAVING <условие>]***

***[ORDER BY <список атрибутов>]***

***[UNION<выражение с оператором SELECT>]***

В квадратных скобках указываются необязательные элементы.

Ключевое слово **ALL** означает, что результатом будут все строки, удовлетворяющие условию запроса, в том числе и одинаковые строки. **DISTINCT** означает, что в результирующий набор не включаются одинаковые строки. Далее идет список атрибутов исходной таблицы, которые будут включены в таблицу-результат. Символ **\*** означает, что в таблицу-результат включаются все атрибуты исходной таблицы.

Обязательным ключевым словом является слово **FROM**, за ним следуют имена таблиц, к которым осуществляется запрос.

В предложении с ключевым словом **WHERE** задаются условия выборки строк таблицы. В таблицу-результат включаются только те строки, для которых условие, указанное в предложении WHERE, принимает значение истина.

В предложении с ключевым словом **GROUP BY** задается список атрибутов группировки (разъяснение этого и последующего ключевого слова будет представлено немного позднее).

В предложении **HAVING** задаются условия, накладываемые на каждую группу.

Ключевое слово **ORDER BY** задает операцию упорядочения строк таблицы-результата по указанному списку атрибутов.

Короче можно записать:

***SELECT имена\_столбцов FROM имя\_таблицы [WHERE ...условия];***

Часть оператора с условиями является необязательной (мы рассмотрим ее позже). Оператор SELECT без условий выводит все данные из указанных столбцов.

Извлечем имена (столбец 1\_name) и фамилии (столбец 2\_name) всех сотрудников из таблицы Sotrudniki.

***SELECT 1\_name, 2\_name from Sotrudniki;***

Данные представлены в том порядке, в котором они были введены. Более того, последняя строка указывает число строк в таблице. Чтобы вывести всю таблицу, можно воспользоваться упрощенной формой оператора SELECT: ***SELECT \* from Sotrudniki;***

Символ \* в этом выражении означает 'ВСЕ столбцы'.

**Выборка данных с помощью условий. SELECT имена\_столбцов FROM имя\_таблицы [WHERE условия];**

**Операторы сравнения = и != , больше и меньше, <= и >=**

В условии могут использоваться операторы сравнения =, !=, **<= >=**

Выборка столбцов с условием для поля "имя".

***SELECT name, 2\_name FROM Sotrudniki WHERE name = 'Иван';***

Этот оператор выводит имена и фамилии всех сотрудников, которые имеют имя Иван. Отметим, что слово Иван в условии заключено в одиночные кавычки. Можно использовать также двойные кавычки. Кавычки являются обязательными. Кроме того, сравнения MySQL не различают регистр символов, что означает, что с равным успехом можно использовать "Иван", "иван" и даже "ИвАн".

Выборка столбцов с условием для поля "возраст"

***SELECT name, last\_name FROM Sotrudniki WHERE age=40;***

Это список имен и фамилий всех сотрудников с возрастом 40 лет. Тип столбца age задан как int, поэтому кавычки вокруг 40 не требуются.

Оператор != означает 'не равно' и является противоположным оператору равенства.

Используемые в основном с целочисленными данными операторы меньше или равно (<=) и больше или равно (>=) обеспечивают дополнительные возможности.

**Поиск текстовых данных по шаблону с помощью предложения WHERE и оператора LIKE.**

Как быть, если надо вывести данные о сотрудниках, имя которых начинается с буквы В? Язык SQL позволяет выполнить поиск строковых данных по шаблону. Для этого в предложении **WHERE** используется оператор **LIKE**. В условии вместо знака равенства используется LIKE и знак процент в шаблоне. **Знак % действует как символ-заместитель** (аналогично \* в системе DOS). Он заменяет собой любую последовательность символов. Таким образом "В%" обозначает все строки, которые начинаются с буквы В. Аналогично "%В" выбирает строки, которые заканчиваются символом В, а "%В%" строки, которые содержат букву В.

Найдем сотрудников, имя которых начинается с буквы В

***SELECT \* FROM Sotrudniki WHERE name LIKE "В%"***

Выведем сотрудников, которые имеют в названии должности строку "про".

***SELECT \* FROM Sotrudniki WHERE*** ***title LIKE '%про%';***

**Логические операторы AND, OR, NOT**

Можно выбирать данные на основе условий SQL, представленных с помощью булевых (логических) операторов**AND, OR, NOT**

Показан оператор SELECT, который выводит имена сотрудников, которые получают более 7000, но меньше 9000.

***SELECT name, last\_name FROM Sotrudniki WHERE salary > 7000 AND salary < 9000;***

Вывести имена и возраст сотрудников, имена которых начинаются с К или Л, и которые младше 30 лет.

***SELECT name, age FROM Sotrudniki WHERE (name like 'К%' OR name like 'Л%') AND age <30;***

Скобки предназначены для выделения различных логических условий и удаления двусмысленностей.

Оператор NOT поможет при поиске всех сотрудников, которые не являются программистами (поле title).

***SELECT name, last\_name FROM Sotrudniki WHERE title NOT LIKE "%программист%";***

**Операторы IN и BETWEEN**

Оператор IN (в множестве) позволяет указать список значений, либо введенный явным образом, либо при помощи подзапроса и сравнить некое значение с этим списком в предложении WHERE или HAVING.

Выведем имена сотрудников, являющихся рабочими или кладовщиками.

***SELECT name FROM Sotrudniki WHERE title IN ('рабочий', 'кладовщик');***

Использование NOT перед IN позволяет вывести данные, которые не входят в множество, определяемое условием IN.

Оператор **BETWEEN** используется для определения **целочисленных границ**.

Выведем сотрудников, возраст которых от 32-х до 40 лет.

***SELECT \* FROM Sotrudniki WHERE age BETWEEN 32 AND 40***

Вместо age >= 32 AND age <= 40 мы использовали age BETWEEN 32 AND 40. NOT также можно использовать вместе с BETWEEN.

**Ключевое слово DISTINCT**

Ключевое слово **DISTINCT** (РАЗЛИЧНЫЙ) исключает появление повторяющихся данных.

Выведем все уникальные должности из таблицы.

***SELECT DISTINCT title FROM Sotrudniki***

Если не использовать DISTINCT, то в списке вывода будут повторяющиеся должности.

**Агрегатные функции. Поиск максимального, минимального и среднего значений, нахождение суммы и количества записей.**

В MySQL есть 5 агрегатных функций для вычисления минимального, максимального и среднего значений, нахождение суммы и количества записей. **Агрегатные функции выполняют вычисление на наборе значений и возвращают одиночное значение**. Аргументами агрегатных функций могут быть как столбцы таблиц, так и результаты выражений над ними. Агрегатные функции и сами могут включаться в другие арифметические выражения.

1. **MIN(): минимальное значение**
2. **MAX(): максимальное значение**
3. **SUM(): сумма значений**
4. **AVG(): среднее значений**
5. **COUNT(): подсчитывает число записей**

Рассмотрим, например, поиск минимальной зарплаты

***SELECT MIN(salary) from Sotrudniki;***

Агрегатная функция **COUNT()** подсчитывает и выводит общее число записей.

Функция COUNT имеет два формата. В первом случае возвращается количество строк входной таблицы, во втором случае — количество значений аргумента во входной таблице:

**COUNT(\*), COUNT([DISTINCT | ALL] выражение)**

Простейший способ использования этой функции — подсчет количества строк в таблице (всех или удовлетворяющих указанному условию). Для этого используется первый вариант синтаксиса.

Во втором варианте синтаксиса функции COUNT в качестве аргумента может быть использовано имя отдельного столбца. В этом случае подсчитывается количество либо всех значений в этом столбце входной таблицы, либо только неповторяющихся (при использовании ключевого слова DISTINCT).

Например, чтобы подсчитать общее число записей в таблице, нужно выполнить следующую команду. Знак \* означает "все данные".

SELECT COUNT(\*) FROM Sotrudniki;

**Именование столбцов. AS**

MySQL позволяет задавать имена для выводимых столбцов с помощью оператора **AS**.

Вывод средней зарплаты с использованием псевдо-имен столбцов

***SELECT AVG(salary) AS 'Средняя зарплата' FROM Sotrudniki;***

**Предложения GROUP BY и HAVING**

Предложение GROUP BY позволяет группировать аналогичные данные и используется для определения групп выходных строк, к которым могут применяться агрегатные функции (**COUNT**, **MIN**, **MAX, AVG** и **SUM**). Все выходные строки запроса разбиваются на группы, характеризуемые одинаковыми комбинациями значений в столбцах группировки. После чего к каждой группе применяются агрегатные функции. Следует иметь в виду, что для **GROUP BY** все значения **NULL** трактуются как равные, то есть при группировке по полю, содержащему **NULL**-значения, все такие строки попадут в одну группу. Если при наличии предложения **GROUP BY**, в предложении **SELECT** отсутствуют агрегатные функции, то запрос просто вернет по одной строке из каждой группы.

1.Вывести все уникальные должности в таблице:

***SELECT title FROM Sotrudniki GROUP BY title;***

Аналогичный результат будет при использовании **DISTINCT.**

2.Подсчитать количество сотрудников на каждой должности.

***SELECT title, COUNT(\*) FROM Sotrudniki GROUP BY title;***

В этой команде MySQL сначала создает группы различных должностей, а затем выполняет подсчет в каждой группе.

3.Выведем среднюю зарплату сотрудников для каждой должности.

***SELECT title, AVG(salary) FROM Sotrudniki GROUP BY title;***

Предложение **HAVING** применяется после группировки для наложения ограничений на групповые значения. Оно необходимо для проверки значений, которые получены с помощью агрегатной функции не из отдельных строк источника записей, определенного в предложении **FROM**, а из групп таких строк. Поэтому такая проверка не может содержаться в предложении **WHERE**.

Выведем только те подразделения, где средняя зарплата более 1000. Выполним это с помощью предложения **HAVING.**

***SELECT title, AVG(salary) FROM Sotrudniki GROUP BY title HAVING AVG(salary) > 1000;***

**Упорядочивание данных. ORDER BY**

SQL позволяет сортировать извлеченные данные с помощью предложения ORDER BY на основе какого-либо столбца. Предложение ORDER BY может сортировать в возрастающем порядке (ASCENDING или ASC) или в убывающем порядке (DESCENDING или DESC) в зависимости от указанного аргумента.

Выведем имена сотрудников с упорядоченными по алфавиту фамилиями сотрудников (в возрастающем порядке).

***SELECT name, last\_name FROM Sotrudniki ORDER BY last\_name;***

Теперь найдем и выведем число сотрудников, имеющих различные должности, и отсортируем их с помощью ORDER BY, используя псевдо-имя для столбца количество.

***SELECT title, COUNT(\*)AS Number FROM Sotrudniki GROUP BY title ORDER BY Number;***

**Ограничение количества извлекаемых данных.**

**Ограничить число записей**, выводимых оператором SELECT. можно с помощью **предложения LIMIT.** Общая форма оператора LIMIT имеет следующий вид:

***SELECT (что-то) FROM имя таблицы LIMIT начальная строка, извлекаемое число записей;***

LIMIT можно использовать также для **извлечения подмножестваданных**, используя дополнительные аргументы.

Выведем три записи, начиная с шестой.

***SELECT name FROM Sotrudniki LIMIT 6,3;***

**Контрольные вопросы**

1. Как узнать число строк в таблице с помощью оператора SELECT
2. Что делает оператор SELECT без условий?
3. Какие служебные слова обязательно присутствуют в операторе SELECT?
4. Какой оператор задает имена для выводимых столбцов?
5. Для чего используется оператор BETWEEN?
6. Извлеките 5 записей из таблицы table, начиная с 10 строки?
7. Какой оператор группирует аналогичные данные?
8. Какой оператор позволяет наложить ограничение на группу?
9. Что делает оператор UPDATE без условий?
10. Зачем нужно ключевое слово DISTINCT?

# Лабораторная работа 3

**ВЫБОР ДАННЫХ ИЗ НЕСКОЛЬКИХ ТАБЛИЦ**

**Цель работы**

Создать базу данных и несколько таблиц. Подготовить и реализовать серию запросов, связанных с выборкой информации из созданных таблиц. Изучить функции для работы с датой и временем.

**Содержание работы.**

*Создать базу данных «Книги» (****B\_ХХ\_ГГ\_YY)*** *, (где XX-номер группы, ГГ –год группы, YY-номер в журнале)*

1. *Создание таблиц в базе данных и заполнение их записями.*

*Создать в БД* ***B\_ХХ\_ГГ\_YY*** *таблицу авторов* ***Avt\_ХХ\_ГГ\_YY****:*

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| *Аvt\_id* | *А\_firstname (varchar50)* | *А\_lastname (varchar40)* | *BirthD (date)* | *Sex**(enum)* | *Town**(int)* |
| *1* | *Иванов* | *Иван* | *15.09.1989* | *м* | *2* |

1. *Проверить, что таблица была создана (SHOW TABLES);*
2. *Заполнить ее записями - придумать штук 10 авторов разного пола, возраста от 30 до 80 лет и с разным количеством детей.*

*Вывести содержимое таблицы.*

1. *Создать таблицу книги* ***Book\_ХХ\_ГГ\_YY****,*

*Задать* ***значение по умолчанию*** *для поля* ***Student*** *-* ***ХХГГYY*** *(XX-номер группы, ГГ-год группы, YY-номер в журнале)*

*Добавить для столбца Avt\_id ограничение* ***Внешний ключ****:*

|  |  |  |  |
| --- | --- | --- | --- |
| *Book\_id* | *Book\_name* | *Avt\_id* | *Student* |
| *Int unsigned not null primary key**auto\_increment* | *varchar(10)* | *Int, unsigned* | *int* |

*Int, unsigned, FOREIGN KEY (Avt\_id) REFERENCES Avt (Avt\_id)*

*Заполнить ее любым способом. {Авторы берутся из предыдущей таблицы). Могут быть авторы, у которых нет книг.*

1. *Посмотреть структуру таблиц (DESCRIBE). Вывести содержимое таблицы.*

*Выполнение запросов.*

1. *Вывести авторов, родившихся в конкретном месяце (например, в феврале);*
2. *Вывести фамилии авторов, у которых есть книги;*

|  |  |
| --- | --- |
| ***F*** | ***m*** |
| *5* | *6* |

1. *Вывести количество авторов каждого пола (2 варианта вывода )*

|  |  |
| --- | --- |
| ***F*** | *5* |
| ***m*** | *6* |

1. *Вывести авторов, фамилия которых содержит букву «В».*
2. *Вывести авторов, которым меньше 50 лет.*
3. *Вывести 5 самых старых авторов.*
4. *Вывести общее количество книг у авторов каждого пола.*

Показать результаты работы преподавателю.

## Методические указания к выполнению лабораторной работы №3.

**Выбор данных из значений типа Date.**

**DAY(date) и DAYOFMONTH(date**) функции-синонимы, возвращают из даты порядковый номер дня месяца.

SELECT DAY('2011-04-17'), DAYOFMONTH('2011-04-17');

**MONTH(date) и MONTHNAME(date)** обе функции возвращают значения месяца. Первая - его числовое значение (от 1 до 12), вторая - название месяца. Названия месяцев различают регистр символов. Поэтому January будет работать, а JANUARY не будет

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

SELECT YEAR('2011-04-17');

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

**ADDDATE(date, INTERVAL value)** Функция возвращает дату date, к которой прибавлено значение value. Значение value может быть отрицательным, тогда итоговая дата уменьшится. В качестве значения value могут выступать не только дни, но и недели (WEEK), месяцы (MONTH), кварталы (QUARTER) и годы (YEAR).

**PERIOD\_ADD(period, n)** функция добавляет n месяцев к значению даты period. Нюанс: значение даты должно быть представлено в формате YYYYMM. Давайте к февралю 2011 (201102) прибавим 2 месяца: SELECT PERIOD\_ADD(201102, 2);.

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

**PERIOD\_DIFF(period1, period2)** функция вычисляет разницу в месяцах между двумя датами, представленными в формате YYYYMM.

**SUBTIME(date, time)** функция вычитает из времени date время time:

**DATE(datetime)** возвращает дату, отсекая время.

**TIME(datetime)** возвращает время, отсекая дату.

**TO\_DAYS(date) и FROM\_DAYS(n)** взаимообратные функции. Первая преобразует дату в количество дней, прошедших с нулевого года. Вторая, наоборот, принимает число дней, прошедших с нулевого года и преобразует их в дату:

Текущую дату, месяц и год можно вывести с помощью аргумента **CURRENT\_DATE** предложений **DAYOFMONTH(), MONTH() и YEAR(),**

**CURDATE(), CURTIME() и NOW()** Первая функция возвращает текущую дату, вторая - текущее время, а третья - текущую дату и время. Функции CURDATE() и NOW() удобно использовать для добавления в базу данных записей, использующих текущее время

**Строковые функции**

**CHAR\_LENGTH(строка); CHARACTER\_LENGTH(строка)**

Возвращает длину строки строка, измеренную в символах. Многобайтные символы считаются как один

**CONCAT(строка1, строка2, ...)**

Возвращает строку, которая состоит из сцепленных аргументов. Возвращает NULL, если любой из аргументов равен NULL. Принимает один или более аргументов. Числовой аргумент преобразуется в эквивалентную строковую форму.

**FIELD(строка, строка1, строка2, строка3, ...)**

Возвращает позицию вхождения аргумента строка в список строка1, строка2, строка3, ... Возвращает 0, если вхождение не найдено.

**FIND\_IN\_SET(строка, список\_строк)**

Возвращает значение от 1 до N, если строка находится в списке строк список\_строк, состоящего из N подстрок. Список строк - это строка, состоящая из подстрок, разделенных символом ','. Возвращает 0, если строка не входит в список строк, или если список\_строк — пустая строка.

**INSERT(строка, позиция, длина, новая\_строка)**

Возвращает строку строка, в которой подстрока длиной длина, начинающаяся с позиции позиция, заменяется строкой «новая\_строка».

**LEFT(строка, длина)**

Возвращает первые длина символов строки строка.

**LENGTH(строка)**

Возвращает длину строки строка в байтах.

**LOWER(строка); (UPPER(строка);)**

Возвращает строку строка, в которой все символы приведены к нижнему (верхнему)регистру в соответствии с текущим набором символов.

**LTRIM(строка)**

Возвращает строку строка с удаленными ведущими пробелами.

**REPLACE(строка, строка\_2, строка\_3)**

Возвращает строку строка, в которой все вхождения строка\_2 заменены на строка\_3.

**Функции сравнения строк.**

Обычно, если любое выражение в сравнении строк чувствительно к регистру, то сравнение также чувствительно к регистру.

**выражение LIKE шаблон [ESCAPE 'символ-отмены']**

Проверка на соответствие шаблону, заданному простыми регулярными выражениями SQL. Возвращает 1 (TRUE) или 0 (FALSE). Если выражение или шаблон равны NULL, возвращает NULL.

В шаблонах LIKE можно использовать следующие два символа:

"%" – соответствие любому числу символов, включая нуль символов, "\_" – соответствие любому одному символу.

 -> 1

**Контрольные вопросы**

1. Какие функции выделяют из даты значение года или месяца?
2. Какие операторы используются для определения наличия значение NULL?
3. Результат SELECT CONCAT('My', NULL, 'QL');?
4. Какое значение вернет CHAR\_LENGTH('') для строки, состоящей из пяти двухбайтных символов?
5. Выведите первые пять символов строки «abcdefg»?
6. Какая функция приводит символы к нижнему регистру?
7. Выведите фамилию Иванов из строки ‘ФИО\_ИВАНОВ’
8. Чем отличаются функции CHAR\_LENGTH(строка) и LENGTH(строка)?
9. С помощью какой функции проверить идентичность строк?
10. Результат выполнения функции Select Month(‘2015-07-08’);
11. Как вывести сегодняшнюю дату?

# Лабораторная работа 4

**ВЫБОР ДАННЫХ из ТАБЛИЦ. ВЛОЖЕННЫЕ ЗАПРОСЫ. ОБЪЕДИНЕНИЕ ТАБЛИЦ.**

**Цель работы**

Используя данные базы данных из третьей лабораторной работы, подготовить и реализовать серию запросов, связанных с выборкой информации из таблиц.

**Содержание работы**

1. *Вычислить, сколько лет автору на сегодняшний день (использовать текущую дату), вывести имя автора, дату рождения и возраст.*
2. *Вывести все названия книг одного автора.*
3. *Вывести список авторов и количество книг, которых они написали.*
4. *Вывести автора, у которого наибольшее количество книг.*
5. *Используя объединение таблиц, вывести названия книги, имя автора, дату рождения автора сначала для авторов, родившихся в январе, затем в феврале.*
6. *Вывести название книг, фамилию автора и название города, для авторов из одного города (на ваш выбор), используя вложенный запрос.*

## Методические указания к выполнению лабораторной работы №4.

**Соединения.**

**Соединения предназначены для обеспечения выборки данных из нескольких таблиц и включения этих данных в один результирующий набор**. **Существует четыре вида соединений**: внутреннее, внешнее, полное, перекрестное.

Для объединения трех и более таблиц можно применять последовательность соединений.

**Для соединения таблиц необходимо раздел FROM дополнить ключевыми словами JOIN, которое определяет соединяемые таблицы и метод соединения, и ON, указывающее общие для таблиц поля.**

**SELECT <select list> FROM <first table> [<alias>]**

**<JOIN TYPE> <second table> [<alias>] [ON <join condition>][;]**

**При внутреннем соединении** сравниваются значения общих полей двух таблиц. Конструкция **INNER JOIN** возвращает только строки, согласованные по всем полям, которые обозначены как используемые для соединения. **Записи, для которых не имеется пары в связанной таблице, в результат не включаются.**

По умолчанию применяются внутренние соединения, ключевое слово **INNER** является необязательным

**Внешние соединения LEFT OUTER JOIN, RIGHT OUTER JOIN**

При использовании конструкции **OUTER** существует возможность включить в результирующий набор строки, которые соответствуют **хотя бы одному из заданных критериев.** Такие соединения применяются для получения **полного набора записей** одной из таблиц.

В соединениях различаются стороны **– левая и правая**. **Левой** считается таблица, указанная **в первую очередь**, а **правой** – таблица, **указанная после нее. Ключевое слово OUTER необязательно.**

**Полные соединения FULL JOIN**

Если применяется соединение с ключевым словом **FULL,** то в результаты должны быть **включены все строки из таблиц**, находящихся по обе стороны от ключевого слова **JOIN**, без каких либо исключений

**Перекрестные соединения**

При таком соединении выводятся **все комбинации записей таблиц**, при этом не требуется указание совпадающих значений полей, поэтому **условие ON опускается**.

Происходит соединение каждой строки таблиц, находящихся с одной стороны от ключевого слова JOIN, с каждой строкой таблиц, находящихся с другой стороны от ключевого слова JOIN. В итоге формируется **декартово произведение всех строк**, заданных по обе стороны ключевого слова JOIN.

**Использование UNION**

Иногда бывает нужно получить два списка записей из таблиц в виде одного. Для этой цели может быть использовано ключевое слово **UNION, которое позволяет объединить результирующие наборы данных двух запросов в один набор данных.**

Технических ограничений на количество запросов в операторе UNION не существует. Общий синтаксис следующий.

< инструкция **SELECT 1**> **UNION** [ALL | DISTINCT]

< инструкция **SELECT 2**> **UNION** [ALL | DISTINCT]

**UNION -п**оказывает, что результирующие наборы будут объединены в один результирующий набор. Дубликаты строк по умолчанию удаляются.

**ALL -**объединяются и дубликаты строк из всех результирующих наборов.

**DISTINCT -**Из результирующего набора удаляются дубликаты строк. Столбцы, содержащие значения NULL, считаются дубликатами. По умолчанию принимается DISTINCT.

Существует лишь одно важное правило, о котором следует помнить при использовании оператора UNION: **порядок, количество и тип данных столбцов должны быть идентичны во всех запросах**.

**Типы данных не обязательно должны быть идентичны, но они должны быть совместимы**. Например, типы CHAR и VARCHAR являются совместимыми. По умолчанию результирующий набор использует размер наибольшего из совместимых типов.

**Имена полей итогового набора берутся только из первого запроса, поэтому создание псевдонимов полей выполняется в нем**

Для получения отсортированного набора данных раздел **ORDER BY** указывается после последнего оператора **SELECT. В отличие от соединения, записи в итоговый набор добавляются друг за другом.**

**Вложенные запросы**

В SQL предусмотрена возможность объединять запросы в один путем превращения одного из них в подзапрос (вложенный запрос).

В одном запросе может быть несколько подзапросов, синтаксис у такого запроса следующий:

**SELECT имя\_столбца FROM Табл. WHERE часть условия IN**

**(SELECT имя\_столбца FROM Табл WHERE часть условия IN**

**(SELECT имя\_столбца FROM имя\_таблицы WHERE условие))**

Подзапросы могут выбирать только один столбец, значения которого они будут возвращать внешнему запросу. Попытка выбрать несколько столбцов приведет к ошибке. Стоит отметить, что не рекомендуется создавать запросы со степенью вложения больше трех. Это приводит к увеличению времени выполнения и к сложности восприятия кода.

Приведенный синтаксис вложенных запросов, скорее наиболее употребительный, но вовсе не единственный. Мы можем использовать любые операторы, используемые с ключевым словом WHERE.

Контрольные вопросы.

1. Перечислите четыре вида соединений.
2. При каком соединении записи, для которых не имеется пары в связанной таблице, в результат не включаются?
3. При каком соединении условие ON опускается?
4. Какие соединения применяются для получения **полного набора записей** одной из таблиц?
5. С помощью какого соединения можно получить декартово произведение таблиц?
6. Чем отличается порядок записей в итоговом наборе, полученном с помощью соединения Join и объединения Union?
7. Из какого запроса берутся имена полей итогового набора при объединении (Union)?

# СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ

**Основной**

1. Мартин Дж. Организация баз данных в вычислительных системах: Пер. с англ. /Под ред. А.А. Стогния и А.Л. Щерса. – М.: Мир, 1980. – 664 с.
2. Конноли Т., Бэгг К., Страчан А. Базы данных: проектирование, реализация и сопровождение. Теория и практика. 2-е изд.: Пер. с англ. – М.: Издательский дом «Вильямс», 2000. – 1120 с.
3. Крёнке Д. Теория и практика построения баз данных. 8-е изд. – СПб.: Питер, 2003. – 800 с.
4. Дейт К. Дж. Введение в системы баз данных. – М.: Вильямс, 2008.

**Дополнительный**

1. Грофф Дж., Вайнберг П. Энциклопедия SQL. 3-е изд. СПб.: Питер, 2003.
2. Шумаков П.В. Delphi 3 и создание приложений баз данных. М.: Изд-во «Нолидж», 1998.
3. Мамаев Е. Microsoft SQL Server 2000 в подлиннике. СПб.: Изд-во BHV, 2001