

Инструментальные средства информационных систем

Лабораторная работа 4 Hello MySQL, MongoDB, Redis

Цель:

познакомиться с основами построения запросов в БД на примере MySQL, MongoDB, Redis.

Задачи:

- 1) Научиться получать информацию о составе и структуре БД
- 2) Научиться импортировать данные в БД
- 3) Научиться составлять простые SELECTы в MYSQL
- 4) Познакомиться с основами работы с MongoDB и с типами данных
- 5) Познакомиться с основами работы с Redis и типами данных

Содержание

1 Введение в MySQL	3
1.1 Команды сбора информации о БД в MySQL	3
1.2 Импорт данных	5
Задание 1	6
1.3 Простые SELECTы MySQL	6
Задание 2	8
1.4 Простые функции	8
Задание 3	11
1.5 Агрегирующие функции	12
Задание 4	14
1.6 Сортировка результатов Order by и ограничение результатов LIMIT	14
Задание 5	15
1.7 Подзапросы	16
Задание 6	17
1.8 Объединение результатов нескольких SELECTов Union	17
Задание 7	18
1.9 Группировка Group by	18
Задание 8	19
2 Введение в MongoDB	20
2.1 Импорт данных	20
2.2 Основные команды	20
Задание 9	20
2.3 Запросы, обновление, добавление и удаление данных	21
Задание 10	23
2.4 Типы данных и организация хранения	23
Задание 11	25
2.5 Агрегация	25
Задание 12	26
3 Введение в Redis	26
3.1 Команды	27
Задание 13	29
3.2 Типы данных и организация хранения	29
Задание 14	33
3.3 Pub/Sub	33
Задание 15	34

В предыдущей лабораторной работе мы познакомились, как развернуть у себя на компьютере контейнеры с Базами данных. Теперь познакомься поближе с самим Базами данных.

1 Введение в MySQL

Запустите в Docker контейнер `my-mysql` и войдите в консоль MySQL.

Правило первое: все команды в консоли MySQL должны заканчиваться “;”.

1.1 Команды сбора информации о БД в MySQL

Команды по сбору информации нужно чтобы понять структуру БД, связь между таблицами или если нужно вспомнить какие поля есть в той или иной таблице.

На одном сервере мы можем иметь несколько баз данных, чтобы посмотреть все существующие базы данных на сервере используется команда:

`show databases;`

```
mysql> show databases;
+-----+
| Database           |
+-----+
| information_schema |
| mysql              |
| performance_schema |
| sys                |
+-----+
4 rows in set (0.03 sec)
```

Перейти в указанную БД и выполнять следующие команды для неё. (Эта команда не для сервера, а для клиента, поэтому в конце нет “;”.)

`use <имя базы данных>`

```
mysql> use mysql
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

Database changed
```

Показать какие существуют таблицы в текущей БД:

```
show tables;
```

```
mysql> show tables;
```

```
+-----+
| Tables_in_mysql |
+-----+
| columns_priv    |
| component       |
| db              |
| default_roles   |
| ...             |
```

Посмотреть описание какой-то конкретной таблицы:

```
describe <имя таблицы>;
```

```
mysql> desc help_category;
```

```
+-----+-----+-----+-----+-----+-----+
| Field          | Type          | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+-----+
| help_category_id | smallint unsigned | NO   | PRI | NULL    |       |
| name            | char(64)       | NO   | UNI | NULL    |       |
| parent_category_id | smallint unsigned | YES  |     | NULL    |       |
| url             | text           | NO   |     | NULL    |       |
+-----+-----+-----+-----+-----+-----+
4 rows in set (0.01 sec)
```

Рассмотрим, что входит в описание таблицы:

Field – названия полей таблицы,

Type – тип значений полей таблицы,

Null – может ли поле принимать значение Null,

Key – обозначение является ли поле ключевым или уникальным,

Default – значение по умолчанию,

Extra – дополнительные параметры поля, например, что поле является авто-инкрементируемым или заполняется автоматически.

Таблицы создаются в MySQL командой create table, чтобы посмотреть каким образом таблица была создана используется команда:

```
show create table <имя таблицы>;
```

```
mysql> show create table default_roles;

+-----+-----+
| Table      | Create Table |
+-----+-----+
| default_roles | CREATE TABLE `default_roles` (
  `HOST` char(255) CHARACTER SET ascii COLLATE ascii_general_ci NOT NULL DEFAULT '',
  `USER` char(32) COLLATE utf8_bin NOT NULL DEFAULT '',
  `DEFAULT_ROLE_HOST` char(255) CHARACTER SET ascii COLLATE ascii_general_ci NOT NULL DEFAULT '%',
  `DEFAULT_ROLE_USER` char(32) COLLATE utf8_bin NOT NULL DEFAULT '',
  PRIMARY KEY (`HOST`,`USER`,`DEFAULT_ROLE_HOST`,`DEFAULT_ROLE_USER`)
) /*!50100 TABLESPACE `mysql` */ ENGINE=InnoDB DEFAULT CHARSET=utf8mb3
  COLLATE=utf8_bin STATS_PERSISTENT=0 ROW_FORMAT=DYNAMIC COMMENT='Default roles' |
+-----+-----+
1 row in set (0.00 sec)
```

1.2 Импорт данных

Для того чтобы начать эксперименты с БД нам нужно создать свою БД, свой набор данных. Мы, конечно, это можем сделать вручную, можем создать свою базу данных `create database`, можем создать таблицы `create table`, можем вставить в них данные `insert into table`. Сэкономим время на этом, сделаем импорт готовых данных. Перед импортом данных следует выйти из консоли MySQL.

Скопируйте себе файлы из репозитория GitHub:

[NataliaSafiullina/Information-Systems-Tools/Лабораторные/DatabaseForMySQL/schema.sql](https://github.com/NataliaSafiullina/Information-Systems-Tools/blob/master/Лабораторные/DatabaseForMySQL/schema.sql)

[NataliaSafiullina/Information-Systems-Tools/Лабораторные/DatabaseForMySQL/data.sql](https://github.com/NataliaSafiullina/Information-Systems-Tools/blob/master/Лабораторные/DatabaseForMySQL/data.sql)

В файле `schema.sql` команды SQL по созданию базы данных и таблиц, можете посмотреть его. В файле `data.sql` команды SQL, которые заполняют наши таблицы данными.

Положите файлы в какую-нибудь папку, в своей командной строке перейдите в эту папку.

Теперь передадим эти файлы на вход MySQL. Делается это так и в такой последовательности, сначала схема, потом данные:

```
D:\SQL>docker exec -i my-mysql mysql -uroot -ppass <schema.sql

D:\SQL>docker exec -i my-mysql mysql -uroot -ppass <data.sql
```

Проверяем появилась ли новая БД, которая называется my_db:

```
mysql> show databases;
+-----+
| Database          |
+-----+
| information_schema |
| my_db              |
| mysql              |
| performance_schema |
| sys                |
+-----+
5 rows in set (0.00 sec)
```

Задание 1

Напишите команды, сохраните их в файл, чтобы потом отправить на проверку.

- 1) Смените текущую БД на my_db.
- 2) Посмотрите, какие таблицы есть в my_db?
- 3) Посмотрите, какие типы полей есть в таблице user_private_message?

1.3 Простые SELECTы MySQL

По традиции начинаем с Hello world! Посмотрите на конструкцию в код-блоке:

```
mysql> select "Hello world!" first;
+-----+
| first      |
+-----+
| Hello world! |
+-----+
1 row in set (0.01 sec)
```

`select` – ключевое слово, с которого обычно начинается запрос, за ним следует описание, что мы хотим получить.

`"Hello world!"` – константа, а могло быть выражение, какие-то математические операции, функции и т.д.

`first` – просто название столбца, которым мы подписали свой результат, этот параметр не обязателен. Сравните:

```
mysql> select "Hello world!";
+-----+
```

```
| Hello world! |  
+-----+  
| Hello world! |  
+-----+  
1 row in set (0.00 sec)
```

Выполните все примеры в консоли MySQL.

Пример 1

Давайте выберем из таблицы user имена и фамилии пользователей.

```
mysql> select first_name Name, last_name "Family name" from user;  
+-----+-----+  
| Name      | Family name |  
+-----+-----+  
| Carla     | Mariet      |  
| Sanjeet   | Iwona       |  
| Nannanne  | Anand       |  
...  
+-----+-----+
```

Мы взяли два поля, указали их через запятую в select.

Подписали колонки, чтобы было понятно что в них.

Затем указали из какой таблицы взять в from.

Наш запрос был без каких либо условий, поэтому он выдал нам все записи из таблицы. Если посмотреть самый конец вывода результатов, то там будет написано 200 rows – у нас 200 результатов.

Пример 2

Давайте отберем из таблицы user имена и фамилии только активных пользователей, т.е. тех, у кого в поле is_active стоит "1" (true).

```
select first_name Name, last_name "Family name"  
from user  
where is_active = true;
```

Список должен уменьшиться. Мы добавили условие, сравнили поле is_active с true (что значит 1, а false = 0), сравнение делается одним знаком равно. Можно также попробовать другие знаки сравнения: !=, <>, <, >, <=, >=.

Задание 2

Напишите запрос, который вернет названия дискуссионных групп, которые требуют подтверждение регистрации, т.е. таблица – discussion_group, поле approve_required равно 1 или true.

1.4 Простые функции

Подобно другим языкам, функции в SQL вычисляют какие-либо результаты и возвращают их. Но рассмотрим сначала несколько нужных операторов.

1) Оператор **LIKE** – применяется для поиска похожего значения, например:

```
mysql> select first_name Name, last_name "Family name"
->      from user
->      where first_name like "A%Y";

+-----+-----+
| Name   | Family name |
+-----+-----+
| Amandy | Makam       |
| Aubrey | La          |
+-----+-----+
2 rows in set (0.00 sec)
```

Запрос отбирает записи из таблицы user, у которых имя пользователя начинается на А и заканчивается на Y,

% – представляет ноль, один или несколько символов,

_ – представляет собой один символ.

2) Оператор **BETWEEN** – выбирает значения в заданном диапазоне. Значения могут быть числами, текстом или датами. Например:

```
mysql> select first_name, date_of_birth
from user
where date_of_birth between '1980-01-01' and '1989-12-31';

+-----+-----+
| first_name | date_of_birth |
+-----+-----+
| Fedora    | 1982-06-15   |
| Blair     | 1981-08-14   |
| Zach      | 1981-05-15   |
| ...       |               |
```


Выбрали пользователей родившихся в период с 1980 по 1989.

- 3) Функция **IF()** – возвращает выражение 1, если условие истина, иначе возвращает выражение 2.

```
select if(условие, выражение 1 если истина,  
          выражение 2 если ложь)
```

Посмотрите интересную особенность в код-блоке, Null не равен Null (любое сравнение с Null будет ложным):

```
mysql> select if(Null=Null, 'Null = Null', 'Null <> Null') "IF  
NULL=NULL?";
```

```
+-----+  
| IF NULL=NULL? |  
+-----+  
| Null <> Null   |  
+-----+  
1 row in set (0.00 sec)
```

- 4) Функция **Coalesce()** – возвращает первое не NULL значение, проверят поля в порядке их указания. Выберем записи из таблицы, где любое из двух полей дат позже 01.11.2020:

```
mysql> select message_id, read_time, send_time  
from user_private_message  
where coalesce(read_time, send_time) > '2020-11-01' limit 3;
```

```
+-----+-----+-----+  
| message_id | read_time          | send_time          |  
+-----+-----+-----+  
| 6521181 | 2020-11-21 00:39:04 | 2020-10-16 00:39:04 |  
| 6521186 | 2020-11-17 00:39:04 | 2020-10-19 00:39:04 |  
| 6521190 | NULL                | 2020-11-01 00:39:04 |  
+-----+-----+-----+  
3 rows in set (0.00 sec)
```

- 5) Функция **Greatest()** – возвращает наибольшее значение из списка значений, рассмотрим не сильно логичный запрос:

```
mysql> select greatest(read_time, send_time) res,
        read_time,
        send_time
        from user_private_message limit 3;
```

```
+-----+-----+-----+
| res          | read_time      | send_time      |
+-----+-----+-----+
| NULL         | NULL           | 2020-10-29 00:39:04 |
| 2020-11-21 00:39:04 | 2020-11-21 00:39:04 | 2020-10-16 00:39:04 |
| 2020-11-17 00:39:04 | 2020-11-17 00:39:04 | 2020-10-19 00:39:04 |
+-----+-----+-----+
3 rows in set (0.00 sec)
```

6) Функция **Concat()** – склеивание строк.

```
mysql> select concat(first_name, ' ', last_name) Name
        from user limit 3;
```

```
+-----+
| Name          |
+-----+
| Carla Mariet  |
| Sanjeet Iwona |
| Nannanne Anand |
+-----+
3 rows in set (0.00 sec)
```

7) Функция **Curdate()** – возвращает текущую дату.

```
mysql> select curdate();
+-----+
| curdate() |
+-----+
| 2022-09-29 |
+-----+
1 row in set (0.01 sec)
```

8) Функция **Adddate()** – возвращает скорректированную дату.

```
mysql> select adddate(curdate(), interval -365 day);
+-----+
| adddate(curdate(), interval -365 day) |
+-----+
| 2021-09-29                             |
+-----+
1 row in set (0.01 sec)
```

9) Функции YEAR(), DATE(), TIME() – возвращают от указанной даты соответственно год, дату и время.

```
mysql> select date(curdate()) TODAY;
+-----+
| TODAY      |
+-----+
| 2022-09-30 |
+-----+
1 row in set (0.00 sec)
```

Задание 3

Напишите запрос, который из таблицы user_private_message отберет записи:

- отправленные в ноябре 2020 года (поле send_time),
- текст сообщения начинается на 'A' (поле message_text),
- прочитанные не позже 10 дней от даты отправки (поле read_time),

Что должен показать вывод:

- выбрать максимальное значение из трех ID: message_id, user_from_id, user_to_id,
- вывести значения полей read_time и send_time только в виде даты и подписать их 'READ' и 'SEND'.

Ответ должен быть такой:

```
+-----+-----+-----+-----+
| ID      | READ      | SEND      | TEXT                                     |
+-----+-----+-----+-----+
| 6521460 | 2020-11-11 | 2020-11-02 | assets_c AdClick ddp ... |
| 6521318 | 2020-11-12 | 2020-11-04 | admindemo dealtime ...   |
+-----+-----+-----+-----+
2 rows in set (0.00 sec)
```

1.5 Агрегирующие функции

Агрегирующие функции позволяют вычислять для групп данных итоговые значения такие как количество записей, сумма, минимальное и максимальное значения, среднее значение и т.п.

1) Функция **Count()** – подсчет количества записей.

Пример 1

Подсчитаем количество записей в таблице user:

Способ1:

```
mysql> select count(*) from user;
+-----+
| count(*) |
+-----+
|      201 |
+-----+
1 row in set (0.10 sec)
```

Способ2:

```
mysql> select count(1) from user;
+-----+
| count(1) |
+-----+
|      201 |
+-----+
1 row in set (0.01 sec)
```

Существует два способа, если подходит второй count(1), то лучше всего использовать его, так как он не задействует память для хранения промежуточных результатов.

Пример 2.

Посчитаем количество ненулевых записей:

Способ 1:

```
mysql> select count(read_time) from user_private_message;
+-----+
| count(read_time) |
+-----+
|              302 |
+-----+
1 row in set (0.00 sec)
```

Способ 2:

```
mysql> select count(*) from user_private_message where read_time
is not null;
+-----+
| count(*) |
+-----+
|      302 |
+-----+
1 row in set (0.00 sec)
```

В первом случае загружаем в память все значения этого поля из таблицы и передаем их в функцию итерационно.

Во втором случае сначала выбираем строчку, потом фильтруем строку в запросе и результат отправляем в функцию.

Разница по быстродействию между двумя этими запросами будет в том, насколько будет сложнее отфильтровать строку.

- 2) Функции **Min()** и **Max()** – возвращают минимальное и максимальное значение поля таблицы.

Запрос, который отбирает последнее прочитанное сообщение:

```
mysql> select max(read_time) from user_private_message;
+-----+
| max(read_time) |
+-----+
| 2020-11-24 00:39:04 |
+-----+
1 row in set (0.01 sec)
```

Давайте создадим ошибку агрегации:

```
mysql> select max(send_time), send_time
        from user_private_message
        where send_time is null;
```

```
ERROR 1140 (42000): In aggregated query without GROUP BY,
expression #2 of SELECT list contains nonaggregated column
'my_db.user_private_message.send_time'; this is incompatible with
sql_mode=only_full_group_by
```

MySQL говорит, что в нашем агрегированном запросе есть колонка результаты которой не агрегированы, СУБД не сможет вывести такие результаты.

3) Функция **Avg()** – возвращает среднее значение.

4) Функция **Sum()** – возвращает сумму значений.

Агрегирующие функции можно использовать вместе:

```
mysql> select count(1) "TOTAL", max(read_time) "Last read",  
sum(is_read) "READ" from user_private_message;
```

```
+-----+-----+-----+  
| TOTAL | Last read          | READ |  
+-----+-----+-----+  
|    400 | 2020-11-24 00:39:04 |   302 |  
+-----+-----+-----+  
1 row in set (0.00 sec)
```

Задание 4

Напишите запрос, который выберет из таблицы `users_to_discussion_groups`:

- количество подтверждений присоединения к группам,
- наиболее раннюю дату присоединения пользователя к группе,
- дату наиболее позднего подтверждения участника в группе.

1.6 Сортировка результатов Order by и ограничение результатов LIMIT

ORDER BY – сортирует полученные результаты.

```
mysql> select last_name Name, first_name "Family name"  
from user  
order by 1 desc  
limit 3;
```

```
+-----+-----+  
| Name  | Family name |  
+-----+-----+  
| Zach  | Zach        |  
| Yung  | Damita      |  
| Yonik | Supriya     |  
+-----+-----+
```

```
3 rows in set (0.00 sec)
```

asc – значение по умолчанию, сортировка по возрастанию, NULL будут идти первыми.

desc – сортировка по убыванию.

В ORDER BY можно указывать несколько полей через запятую, и вместо имен полей можно указывать номер колонки.

Сортировка происходит в памяти СУБД. СУБД сначала получает результат, потом сортирует его.

В select необязательно указывать поля, по которым есть сортировка.

LIMIT – возвращает указанное количество записей.

```
mysql> select user_id, first_name
       from user
       order by first_name
       limit 10;
```

Получим первые 10 записей.

```
mysql> select user_id, first_name
       from user
       order by first_name
       limit 10,5;
```

```
+-----+-----+
| user_id | first_name |
+-----+-----+
|    7996 | Amandy    |
|    8261 | Amelia    |
|    7545 | Amit      |
|    7849 | Anabel    |
|    7757 | Anetta    |
+-----+-----+
5 rows in set (0.00 sec)
```

Получим 5 записей, с 11-ой записи по 15-ую.

Задание 5

Напишите SQL-запрос, который выбирает 20 последних зарегистрированных пользователей. Поля в результатах выборки: user_id, registration_time.

Чтобы выбрать последних зарегистрированных пользователей, достаточно отсортировать их и добавить ограничение на количество результатов.

1.7 Подзапросы

Подзапросы нужны чтобы не повторять код.

Подзапросы могут использоваться:

- 1) с помощью конструкции WITH,
- 2) в результатах запроса,
- 3) в FROM,
- 4) в условиях WHERE,
- 5) в ORDER BY.

Не будем рассматривать все варианты, они все аналогичные. Уделим внимание WITH (первый пункт) и второму пункту.

- 1) Подзапросы с помощью WITH в MySQL строятся так:

- ключевое слово with
- имя подзапроса + as
- сам подзапрос
- затем просто select где мы обращаемся к подзапросу по имени, как будто это таблица.

Пример 1

Отберем активных пользователей, которые зарегистрировались через google.

```
mysql> with
google_active_user as
(select * from user
 where is_active=1 and registration_type='google')

select first_name NAME, last_name FANME, registration_type REG
from google_active_user;
```

Мы можем основывать подзапросы на других подзапросах, разделяя их запятой.

Пример 2.

Тот же самый запрос, но с двумя подзапросами, мы разделили условия по двум подзапросам:


```
mysql> with
active_user as (select * from user where is_active=1),
google_active_user as (select * from active_user
                        where registration_type='google')

select first_name NAME, last_name FNAME, registration_type REG
from google_active_user;
```

2) В результатах запроса подзапрос должен возвращать один результат.

Запрос ниже возвращает ID и имя группы и имя с фамилией её администратора:

```
mysql> select
        group_id,
        name,
        (select concat(first_name, ' ', last_name)
         from user where user_id = admin_user_id) "User name"
from discussion_group;
```

В данном случае для каждой строки запроса выполняется подзапрос.

Задание 6

Напишите SQL-запрос, который удовлетворяет следующим критериям:

1) В запросе в секции WITH указаны два подзапроса:

- groups_with_approve — выбирает группы, в которых требуется подтверждение;
- new_groups — группы, созданные в 2020 году или позже, в которых требуется подтверждение.

2) Между подзапросами groups_with_approve и new_groups есть зависимость.

3) В основном запросе происходит выборка всего из new_groups.

1.8 Объединение результатов нескольких SELECTов Union

UNION и UNION ALL – вертикальное соединение результатов запросов, т.е. мы к результатам первой выборки присоединяем результаты последующих выборок.

UNION – выводит только уникальные результаты.

UNION ALL – выводит все результаты, включая дубли.

Записи объединяются в один столбец по псевдониму поля.

Пример 1

Выберем из двух таблиц поля дату регистрации пользователя и дату создания группы, подпишем из какой таблицы запись в поле TableName.

```
mysql> select registration_time TIME, "user" TableName
        from user
        union all
        select creation_time TIME, "group" TableName
        from discussion_group;
```

Задание 7

Напишите SQL-запрос, который выбирает уникальные идентификаторы пользователей среди администраторов групп и отправителей частных сообщений.

1.9 Группировка Group by

Оператор GROUP BY служит для распределения строк, полученных в результате отбора, по заданным группам.

Группировать можно:

- по полю,
- по результату функции,
- по нескольким полям одновременно.

Пример 1, по полю.

Подсчитаем количество сообщений, которые отправили пользователи в разрезе пользователей:

```
mysql> select user_from_id, count(1) as cnt
        from user_private_message
        group by user_from_id;
```

Пример 2, по результату функции.

Посчитаем количество пользователей, у которых имена начинаются на одну и ту же букву, в разрезе букв.

```
mysql> select substring(first_name, 1,1), count(1)
```

```
from user
group by substring(first_name, 1,1)
order by 1;
```

Пример 3, составной ключ группировки и агрегирующие функции.

Узнаем в каком период пользователи поддерживали переписку через личные сообщения, т.е. сгруппируем уникальные пары пользователей и найдём даты самого первого и последнего сообщения.

```
mysql> select user_from_id, user_to_id,
             min(send_time),
             max(send_time)
from user_private_message
group by user_from_id, user_to_id;
```

Оператор **HAVING** фильтрует результаты групп непосредственно перед тем как отправить эти результаты в качестве результатов текущего запроса.

Пример 4, ограничение на результаты групп.

Найдём группы, где количество сообщений более 7.

```
mysql> select group_id, count(1) as cnt
from users_to_discussion_groups
group by group_id having cnt > 7;
```

```
+-----+-----+
| group_id | cnt |
+-----+-----+
| 570775 | 9 |
| 570796 | 8 |
| 570823 | 8 |
| 570848 | 10 |
+-----+-----+
4 rows in set (0.00 sec)
```

Задание 8

Сложное задание.

Напишите запрос, который выберет все даты, в которые были отправлены какие-либо личные сообщения, и в которые любой из отправивших сообщения сделал это только один раз в этот день.

Используем таблицу user_private_message.

Не забываем про функцию `date(send_time)`.

2 Введение в MongoDB

Познакомимся с документо ориентированной СУБД MongoDB, не забывайте, что общую информацию о СУБД можно посмотреть в лекциях, тут будет только практика.

2.1 Импорт данных

Сразу сделаем импорт готовых данных.

Две коллекции `users` и `posts` находятся на GitHub:

[NataliaSafiullina/Information-Systems-Tools/tree/main/Лабораторные/DatabaseForMongoDB](https://github.com/NataliaSafiullina/Information-Systems-Tools/tree/main/Лабораторные/DatabaseForMongoDB)

Кладём будущие коллекции в какую нибудь папку, открываем командную строку, переходим в эту папку.

Копируем будущие коллекции во временный каталог в контейнере:

```
D:\JSON>docker cp users.json my-mongo:/tmp
D:\JSON>docker cp posts.json my-mongo:/tmp
```

Запускаем команду импорта для каждой коллекции:

```
D:\JSON>docker exec -it my-mongo mongoimport --db my_db
--collection users --legacy /tmp/users.json

D:\JSON>docker exec -it my-mongo mongoimport --db my_db
--collection posts --legacy /tmp/posts.json
```

2.2 Основные команды

show dbs – просмотр какие есть базы данных.

use skdb – переключиться на использование какой-то базы данных.

show collections – просмотр коллекций.

Задание 9

Проверьте существуют ли коллекции `users` и `posts` в БД `my_db`.

Напишите использованные команды и получившийся результат.

2.3 Запросы, обновление, добавление и удаление данных

База данных является объектом и на текущую базу данных ссылается переменная `db`. У объекта есть методы и объекты. У `db` объектами являются коллекции, наши коллекции называются `users` и `posts`.

Запросы

Запросы выполняются через метод `find()`.

```
> db.users.find()
```

Где `find` это уже метод объекта `users`.

```
> db.users.find({"_id": "vjaniya@example.ru"})
```

Где `{"_id": "vjaniya@example.ru"}` это значение параметра метода.

```
> db.users.find({"karma": {$lt: -10}, "first_name": /.an.*\/})
```

Где:

`$lt` – ключевое слово, less than, меньше чем,

`/.an.*\/` – регулярное выражение, вхождение подстроки “an”.

Метод `limit()` – ограничение количества записей в выборке.

```
> db.users.find({"karma": {$lt: -10},  
"first_name": /.an.*\/}).limit(1)
```

Обновление данных

– `update`

– `updateMany`

```
db.users.updateMany({"karma": {$lt: -10}, "first_name": /.an.*\/},  
{$set: {"karma": 0}})
```

Где первый параметр повторяет запрос, а второй параметр говорит что изменить. Мы поменяли карму с -10 на 0.

Добавление записей

- `insert()` – один документ,
- `insertMany()` – список документов.

Добавим одну запись:

```
> db.users.insert({"first_name" : "Test", "karma" : 100 , "admin" : true})

WriteResult({ "nInserted" : 1 })
```

Где ID не указан, MongoDB сама формирует `_id`, мы также добавили ключ которого нет ни у одной другой записи: `admin`.

Если найти эту запись, видим `_id`:

```
> db.users.find({"admin":true})

{ "_id" : ObjectId("62e624f3d693620fb821a9ba"), "first_name" : "Test", "karma" : 100, "admin" : true }
```

Добавим множество записей

```
> db.users.insertMany
([{"first_name" : "Test 2" , "karma" : 100 , "admin" : true} ,
 {"first_name" : "Test 3" , "karma" : 100 , "admin" : true}])
```

Найдём все не консистентные документы, пусть это будут документы без `last_name`:

```
db.users.find({"last_name" : {$exists : false}})
```

Удаление записей

- `remove()`

А теперь удалим все записи без `last_name`:

```
> db.users.remove({"last_name" : {$exists : false}})

WriteResult({ "nRemoved" : 1 })
```

Задание 10

Из коллекции постов выберите документы, в которых среди топигов встречается 'as', идентификатор автора содержит example.ru, а score больше 100.

2.4 Типа данных и организация хранения

Строка

Основной тип данных в MongoDB является строка. Строки в формате UTF-8

```
> db.misc.insert({"textValue" : "Simple text на любом языке с  
поддержкой utf-8 :) "})  
  
WriteResult({ "nInserted" : 1 })
```

При этом у нас появится новая коллекция.

Числа

Три типа чисел:

- double, с плавающей точкой,
- int, 32 бита, целочисленный,
- long, 64 бита, целочисленный.

```
> db.misc.insertOne({doubleValue : 9.99, integerValue : 9})
```

Дата и время

- date, указаны и дата и время
- timestamp

Чтобы работать с датой, мы должны преобразовать нашу строку в формат ISO 86.01:

```
> db.misc.insertOne({nowTs : new Timestamp(), created :  
ISODate('2022-01-11T00:00:00Z')})  
{  
  "acknowledged" : true,  
  "insertedId" : ObjectId("6336fbbf1b14b47a8acfce0")  
}
```

`ISODate()` – функция конструктор, которая возвращает объект типа `date`.

`new` – оператор, чтобы вставить текущее время.

`ObjectId` – это тоже тип, который является ключевым.

Массивы

Добавим две записи с массивами из двух элементов:

```
> db.misc.insertMany([ {tags: ["mongodb","mysql"]} , {tags:
["mongodb","postgresql"]} ])

{
  "acknowledged" : true,
  "insertedIds" : [
    ObjectId("6336fde91b14b47a8acfcef1"),
    ObjectId("6336fde91b14b47a8acfcef2")
  ]
}
```

Поиск можно осуществлять по одному элементу массива или по всему массиву, причем строго в той последовательности как указано в массиве в БД.

```
> db.misc.find({tags : "mysql" })

{ "_id" : ObjectId("633703481b14b47a8acfcef3"), "tags" : [
"mongodb", "mysql" ] }
```

Встроенные документы

В документ мы добавляем еще документы, которые в реляционной БД были бы связаны через внешний ключ. Т.е., например, если у нас есть какая-то новость в БД, то комментарии к ней будут в виде списка из объектов.

Например, одна новость зеленым шрифтом и два комментария к ней тоже выделены зеленым шрифтом:

```
> db.articles.insert( { "title" : "Good news everyone", "text" :
"Everything is ok" , "comments" :

[ { "_id" : new ObjectId() , "author" : "byuko@example.com",
"message" : "good, thanks!" } ,
  { "_id" : new ObjectId, "author" : "mmickey@example.com" ,
"message" : "thank you" } ] } )
```



```
WriteResult({ "nInserted" : 1 })
```

В результате, внутри документа (новость) формируются документы (комментарии) со своими идентификаторами.

Давайте найдем самую новость без комментариев:

```
> db.articles.find( { }, { "comments" : 0 } )

{ "_id" : ObjectId("633704be1b14b47a8acfcef7"), "title" : "Good news everyone", "text" : "Everything is ok" }
```

Теперь выберем какой-то комментарий:

```
> db.articles.find( { "comments" : { "$elemMatch" : { "_id" : ObjectId("633704be1b14b47a8acfcef6") } } } , { "comments.$" : 1 } )

{ "_id" : ObjectId("633704be1b14b47a8acfcef7"), "comments" : [ { "_id" : ObjectId("633704be1b14b47a8acfcef6"), "author" : "mmickey@example.com", "message" : "thank you" } ] }
```

Задание 11

Одним запросом добавьте два документа к коллекции posts:

- 1) creation_date — текущее время, автор — skbx@example.com, topics должен быть списком из одного элемента "mongodb";
- 2) creation_date — 31 декабря 2021 года, автор — skbx@example.ru.

2.5 Агрегация

В MongoDB существует понятие Aggregation Pipeline – агрегация происходит в некотором pipeline, некотором конвейере, где данные преобразовываются по шагам.

Все шаги, которые будут использованы, перечисляются в массиве.

Например, по году рождения пользователей вычислять среднее количество кармы. При этом надо отфильтровать пользователей с определенным доменом в email.

aggregate() – метод агрегации.

Шаг 1 – вызываем метод отбираем записи по домену email.

```
> db.users.aggregate( { $match : { '_id' : /example.info/ } } )
```

\$match – ключевое слово сравнения.

Шаг 2 – добавим массив (квадратные скобки) и укажем что хотим получить:

```
> db.users.aggregate( [ { $match : { '_id' : /example.info/ } } , {  
$project: { karma: "$karma", year: { $year: "$birth_day" } } } ] )
```

\$project – ключевое слово, указываем, какие поля должны возвращаться по запросу.

Шаг 3 – агрегация, добавляем группировку по году и расчёт средней кармы:

```
> db.users.aggregate( [ { $match : { '_id' : /example.info/ } } , {  
$project: { karma: "$karma", year: { $year: "$birth_date" } } } , {  
$group: { _id: "$year" , avg_karma: { $avg: "$karma" } } } ] )
```

```
{ "_id" : 1919, "avg_karma" : 106 }  
{ "_id" : 2003, "avg_karma" : 97 }  
{ "_id" : 1990, "avg_karma" : -46 }  
{ "_id" : 1960, "avg_karma" : 80 }  
{ "_id" : 1968, "avg_karma" : -3 }  
{ "_id" : 1995, "avg_karma" : -33 }  
{ "_id" : 1975, "avg_karma" : 120 }
```

\$group – ключевое слово, за которым следует описание группировки.

\$avg – ключевое слово, вычисление среднего значения.

Задание 12

Посчитайте сумму кармы по первым буквам имён пользователей для тех пользователей, у которых больше 300 визитов.

3 Введение в Redis

Запустите в Docker контейнер с Redis, увидите следующее приглашение от redis:

```
C:\>docker exec -it myredis redis-cli  
127.0.0.1:6379>
```

3.1 Команды

Пробуйте вводить все нижеперечисленные команды.

ping – проверка соединения с сервером, если соединение есть сервер ответит PONG.

echo <слово> – Redis ответит указанным словом, это тоже своего рода проверка связи.

Обратите внимание, когда вы вводите команду Redis пишет вам подсказку какие параметры имеет команда.

set – записать значение по ключу, регистр букв имеет значение.

set hello world

где **hello** - ключ, **world** - значение.

get – получить значение.

get hello

```
127.0.0.1:6379> set Hello world
OK
127.0.0.1:6379> get hello
(nil)
127.0.0.1:6379> get Hello
"world"
127.0.0.1:6379>
```

Принято разделять части ключа знаками пунктуации, например так:

set user:10:name Ivan

Конечно, для самого Redis двоеточия ничего не значат, это для удобства программистов.

exists – проверка ключа на существование, если ключ существует, вернет 1, иначе вернет 0.

Проверьте: **exists user:10**

del – удаление ключа, при удачном удалении вернет 1, иначе 0.

Проверьте: `del user:10:name`

Можно создавать временные ключи.

`set hello world ex 60` – этот ключ исчезнет через 60 секунд, для задания времени в миллисекундах используется опция **px**.

ttl – проверка сколько осталось жить ключу.

persist – изменение времени жизни ключа на постоянный.

append – дозапись значения по ключу, возвращает общую длину значения в символах.

```
127.0.0.1:6379> append Hello "!!!"
(integer) 8
127.0.0.1:6379> get Hello
"world!!!"
```

incr/decr – инкремент/декремент значения.

```
127.0.0.1:6379> set index 0
OK
127.0.0.1:6379> incr index
(integer) 1
127.0.0.1:6379> incr index
(integer) 2
127.0.0.1:6379> get index
"2"
127.0.0.1:6379> decr index
(integer) 1
127.0.0.1:6379> get index
"1"
```

Redis будет следить чтобы операции инкремента и декремента выполнялись атомарно.

rename – переименование ключа.

keys * – посмотреть все существующие ключи.

Задание 13

Напишите последовательность команд для Redis:

1. Создайте ключ `index` со значением “index precalculated content”.
2. Проверьте, есть ли ключ `index` в БД.
3. Узнайте, сколько еще времени будет существовать ключ `index`.
4. Установите ключу время жизни 2 минуты.
5. Отмените запланированное удаление ключа `index`.

3.2 Типы данных и организация хранения

множества, упорядоченные множества

Списки (List)

– множество любых значений, могут быть дубли.

С помощью списков можно организовывать очереди, легко можно добавлять и убирать элементы слева и справа. Рассмотрим как.

LPush user:5:skills redis – добавить множество по ключу, где **user:5:skills** – имя ключ, **redis** – значение.

RPush user:5:skills mongodb – добавить значение в множество справа.

Сделайте как в примере ниже:

```
127.0.0.1:6379> lpush user:5:skills redis
(integer) 1
127.0.0.1:6379> rpush user:5:skills mysql
(integer) 2
127.0.0.1:6379> lpush user:5:skills mongodb postgresql
(integer) 4
```

LRange user:5:skills 0 5 – получить значения, начиная с нулевого элемента по пятый (0 5).

```
127.0.0.1:6379> lrange user:5:skills 0 5
1) "postgresql"
2) "mongodb"
3) "redis"
```

4) "mysql"

`lpop user:5:skills` – взять элемент слева.

`rpop user:5:skills 2` – взять два элемента справа.

```
127.0.0.1:6379> lpop user:5:skills
"postgresql"
127.0.0.1:6379> rpop user:5:skills 2
1) "mysql"
2) "redis"
```

Взять – это значит изъять его из множества, больше значение не будет состоять в множестве.

`llen user:5:skills` – проверить остались ли элементы в списке.

Хэш (Hash)

– структура данных, при которой внутри значения key-value хранилища мы можем хранить другой набор key-value пар.

`hset user:5 name James` – записать хэш-значение.

`hget user:5 name` – получить значение.

Где `user:5` – ключ одного хранилища, а `name` – ключ во внутреннем хранилище, `James` – значение из внутреннего хранилища.

```
127.0.0.1:6379> hset user:5 name James lastname Bond
(integer) 2
127.0.0.1:6379> hget user:5 name
"James"
127.0.0.1:6379> hget user:5 lastname
"Bond"
```

Если использовать просто `get`, то получим ошибку, что используем не тот тип данных:

```
127.0.0.1:6379> get user:5
```

```
(error) WRONGTYPE Operation against a key holding the wrong kind of value
```

Множества (Set)

– неупорядоченный набор уникальных значений.

sadd users one@example.com – добавить элемента ["one@example.com"](mailto:one@example.com) в множество users.

smembers users – посмотреть текущий set по данному ключу.

```
127.0.0.1:6379> sadd users one@example.com two@example.com
(integer) 2
127.0.0.1:6379> smembers users
1) "two@example.com"
2) "one@example.com"
```

Можно проводить операции над set. Например получим список пользователей исключая забаненных:

sdiff users ban – разность двух множеств.

```
127.0.0.1:6379> sadd ban three@example.com one@example.com
(integer) 1
127.0.0.1:6379> sdiff users ban
1) "two@example.com"
127.0.0.1:6379> sdiff ban users
1) "three@example.com"
```

sunion users ban – получить список всех уникальных объектов из двух и более множеств.

```
127.0.0.1:6379> sunion users ban
1) "three@example.com"
2) "one@example.com"
3) "two@example.com"
```

Упорядоченное множество (Sorted set)

– набор уникальных строк отсортированных согласно заданному весу, если вес одинаковый, то сортируется по строке.

zadd users 10 James – добавить элемент во множество, где:

users – имя множества

10 – вес элемента

James – имя элемента

zrange users 0 2 – получить элементы с 0 по 2.

```
127.0.0.1:6379> zadd users 10 James
(integer) 1
127.0.0.1:6379> zadd users 5 Anna
(integer) 1
127.0.0.1:6379> zadd users 12 Jhon
(integer) 1
127.0.0.1:6379> zadd users 12 Bob
(integer) 1
127.0.0.1:6379> zrange users 0 10
1) "Anna"
2) "James"
3) "Bob"
4) "Jhon"
```

Получим трёх users с максимальными весами:

```
127.0.0.1:6379> zrange users 0 10 rev withscores
1) "Jhon"
2) "12"
3) "Bob"
4) "12"
5) "James"
6) "10"
```

zpopmin users – выбор из множества элемента с минимальным весом.

zpopmax users – выбор из множества элемента с максимальным весом.

zrank users Bob – возвращает позицию элемента в множестве


```
127.0.0.1:6379> zrank users Bob
(integer) 2
```

Задание 14

Напишите последовательность команд для Redis:

1. Создайте в Redis структуру данных с ключом `ratings` для хранения следующих значений рейтингов технологий: `mysql` — 10, `postgresql` — 20, `mongodb` — 30, `redis` — 40.
2. По этому же ключу увеличьте значение рейтинга `mysql` на 15.
3. Удалите из структуры элемент с максимальным значением.
4. Выведите место в рейтинге для `mysql`.

3.3 Pub/Sub

В Redis реализована реализована поддержка паттерна Publish–subscribe. Познакомимся как использовать каналы pub/sub в Redis.

Запустите два клиента Redis в разных окнах.

`subscribe chat1` – подписаться на канал `chan1`, если такого канала нет, то канал создастся.

`publish chat1 Hello!` – опубликовать сообщение “Hello!” в канале `chat1`.

В первом окне подписываемся на два канала:

```
127.0.0.1:6379> subscribe chat1 chat2
Reading messages... (press Ctrl-C to quit)
1) "subscribe"
2) "chat1"
3) (integer) 1
1) "subscribe"
2) "chat2"
3) (integer) 2
```

Во втором окне, отправляем сообщения в каналы:

```
127.0.0.1:6379> publish chat1 Hello!
(integer) 1
127.0.0.1:6379> publish chat2 Hi!
(integer) 1
```

При этом в первом окне видим сообщения:

```
127.0.0.1:6379> subscribe chat1 chat2
Reading messages... (press Ctrl-C to quit)
1) "subscribe"
2) "chat1"
3) (integer) 1
1) "subscribe"
2) "chat2"
3) (integer) 2
1) "message"
2) "chat1"
3) "Hello!"
1) "message"
2) "chat2"
3) "Hi!"
```

psubscribe chat* – подписаться на все каналы по маске, по паттерну.

pubsub channels – проверить сколько сейчас есть каналов.

pubsub numpat – узнать количество уникальных паттернов.

pubsub numsub chat1 – узнать сколько подписчиков для канала.

Задание 15

Напишите две команды для СУБД Redis:

1. Подпишитесь на все события, опубликованные на каналах, начинающихся с `events`.
2. Опубликуйте сообщение на канале `events42` с текстом "Hello there".