

ЦИФРОВАЯ
КУЛЬТУРА
УНИВЕРСИТЕТ ИТМО

Redis

Высшая Школа Цифровой Культуры
Университет ИТМО

dc@itmo.ru

Содержание

1	Основы Redis	2
2	Структуры данных. Строки и хеши	9
3	Структуры данных. Списки и множества	16
4	За пределами структур данных	24

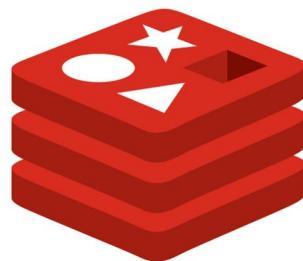
1 Основы Redis

В этой лекции мы познакомимся с основами Redis. Почему именно Redis? Да просто потому, что именно его считают наиболее ярким представителем группы “ключ-значение”. Часто Redis описывают как хранилище данных *in*

Почему именно Redis?

Redis - наиболее яркий представитель группы “ключ-значение”.

<https://redis.io/> - официальный сайт Redis.



memory (то есть в оперативной памяти). Да, действительно, Redis, держит все данные в оперативной памяти, однако периодически сохраняет эти данные на диске. Кроме того, Redis не просто хранит данные типа ключ-значение, а содержит данные гораздо более сложных структур.

Данные в Redis могут быть представлены в виде пяти разных структур, и только одна из них, собственно, и есть в чистом виде структура типа ключ-значение. Понимание этих пяти структур, как они работают, какие методы

Основные характеристики Redis

- Хранит все данные в оперативной памяти
- Периодически сохраняет данные на диске
- Представляет данные в виде структур 5 типов



предоставляют для взаимодействия с пользователем, и что можно сделать с их помощью, как раз и являются ключом к пониманию Redis. Но сначала разберемся с тем, какие именно структуры данных возможны в Redis.

Если обратиться к реляционным базам, то можно сказать, что базы данных предоставляют один универсальный тип структур данных – таблицы. Таблицы одновременно сложные и в то же время гибкие. Их универсальность заключается в том, что любые структуры данных можно смоделировать с помощью таблиц. Тем не менее, они не идеальны. А именно: их не

Сравнение структур данных реляционных баз и Redis

Реляционная база	Redis
Таблицы	Строки Списки Хеши Множества Упорядоченные множества



всегда достаточно просто приспособить для представления данных, и они не всегда достаточно быстро позволяют организовать доступ к требуемым данным. Именно поэтому и появилась идея вместо универсальной структуры использовать специализированные структуры, ориентированные на конкретные прикладные задачи. Конечно, в этом случае, наверное, найдутся данные, которые мы не сможем представить в виде узкоспециализированных структур, однако для каких-то данных, мы выиграем в простоте и скорости доступа. Использование специфичных структур данных для специфичных задач? Да, именно в этом и состоит подход Redis. Если в прикладной задаче вам необходимы не таблицы, а скаляры, списки, хеши или множества, то почему бы с самого начала не пытаться представлять их не в виде таблиц, а хранить как скаляры, списки, хеши и множества?

Как и где можно познакомиться со структурами данных и приемами работы в Redis чтобы получить минимальные практические навыки? Можно скачать дистрибутив Redis с официального сайта <https://redis.io/download>, установить его на своем компьютере и получить базу Redis в свое полное распоряжение. Redis использует знакомую всем концепцию базы дан-

Как получить доступ к Redis?

Загрузка Redis
с официального сайта

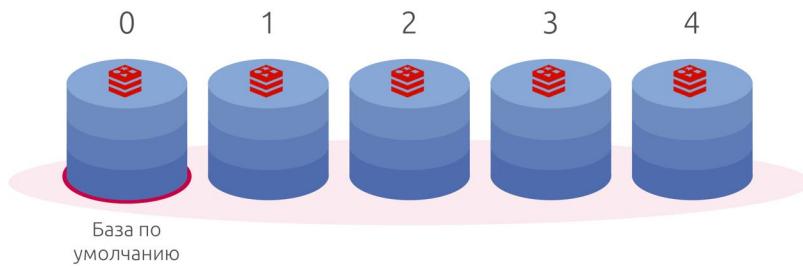


ных. База – это набор данных. Типичное предназначение базы данных Redis –

это группировка всей информации определенного приложения в одном месте и изоляция ее от других приложений.

В Redis база данных идентифицируется целым числом, которое по умолчанию равняется 0 (а соответствующая база называется базой по умолчанию). Если необходимо переключиться на другую базу данных, то можно это

Идентификация баз в Redis



сделать командой `SELECT`. Например, для переключения на базу с идентификатором 1 достаточно в командной строке ввести команду: `SELECT 1`

Redis должен ответить сообщением `OK`, а в терминале вы должны увидеть что-то типа того, что видите сейчас на слайде. Если требуется переключиться



обратно на базу по умолчанию, просто введите в командной строке: `SELECT 0`.

Одним из основополагающих понятий Redis является понятие ключа. Несмотря на то, что Redis больше, чем просто хранилище типа ключ-значение, в его основе каждая из пяти используемых структур обязательно имеет ключ и значение. Поэтому очень важно разобраться в том, что такое ключи и что такое значения, перед тем как двигаться дальше.

Итак, что собой представляет ключ в паре “ключ-значение”? Прежде всего, он должен быть уникальным в базе. Это идентификатор, который позволяет получить доступ к ассоциированному с ним значению. Теоретически он

может быть любым. Однако в практических реализациях всегда есть ограничения, связанные, как минимум, с его размером. В качестве ключа можно использовать любую последовательность символов – от короткой строки текста до содержимого файла изображения. Даже пустая строка является допустимым ключом. Однако из соображений производительности следует избегать слишком длинного ключа. В хранилищах типа "ключ-значение" большое внимание уделяется выбору структуры самого ключа. Ключи можно генерировать с помощью специальных алгоритмов и задавать явно. В качестве ключа может выступать идентификатор пользователя, электронный адрес, номер телефона и т.п. Ключ можно формировать из даты, времени и иден-

Примеры ключей

- 0126148
- abc
- agent:007:name
- anna@mail.ru
- (921)572800
- 17/04/2019:15:40



тификаторов приложений, работающих с базой. Еще одна идея – формировать ключи в виде следующей структуры: <тип объекта:идентификатор объекта:атрибут>. Таким образом можно сохранять в хранилищах “ключ-значение” всю информацию об объектах и получать доступ к значениям атрибутов объектов. С помощью специальных методов можно задавать интервал времени, по истечении которого ключ станет недействительным. Последнее свойство особенно удобно для работы с сессионными объектами и корзинами покупателей. На слайде Вы можете видеть примеры ключей. Они достаточно разнообразны. Это и просто последовательность символов, и имя агента 007, и электронный адрес, и номер телефона и даже простые дата и время. Любая из этих структур может быть ключом.

Значением в хранилище “ключ-значение” может быть что угодно. Например, длинный или короткий текст, число, программный код, изображение и т. п. Значение также может быть списком, множеством или даже другой парой ключ-значение, инкапсулированной в объект.

Некоторые хранилища позволяют указывать тип данных для значения. Например, можно указать, что значение должно быть целым числом. Другие хранилища не предоставляют эту функциональность и, следовательно,

значение может быть любого типа. Однако не все команды применимы к значениям любого типа. Например, команда, которая увеличивает значение на 1, подразумевает, что ее операндом может быть только численное значение. Рассмотрим некоторые примеры ассоциированных ключей и значений. Очень хорошо известный нам пример – справочник телефонов (или как сей-

Справочник телефонов

Ключ	Значение
Catarina	(987) 446-7890
Annet	(921) 567-8301
Adelaida	(945) 678-9012
Elena	(856) 759-0123



час говорят – контактов). Ключ-имя, а значение – номер телефона.

Еще один возможный пример – коллекция мультфильмов. Здесь ключ организован по принципу <тип объекта:идентификатор:атрибут> и в приведенном примере легко разобраться где хранится название мультфильма, где год выпуска, а где – количество просмотров.

Мультфильмы

Ключ	Значение
cartoon:1:name	The Lion King
cartoon:1:year	1994
cartoon:1:views	5495742
cartoon:2:name	The Little Mermaid
cartoon:2:year	1989
cartoon:2:views	3295423



Для хранилищ типа “ключ-значение” характерно использование языков запросов низкого уровня (не декларативных). Запросы в хранилищах типа “ключ-значение” могут выполнять команды по ключу и ничего более. Результатом запроса является все значение. Часть значения вытащить с помощью запроса невозможно. Даже, если значением является слабоструктурированный документ JSON-формата, вытащить отдельные атрибуты документа на

уровне запроса не удастся. Анализ выбранного документа может быть произведен только на уровне приложения.

И, тем не менее, этих, казалось бы, весьма ограниченных, возможностей для многих приложений вполне хватает. На слайде приведены несколько запросов в СУБД Redis. Составным ключам мультфильм-идентификатор-

Основные команды Redis

SET<ключ><значение>

– задать значение по ключу

GET<ключ>

– выбрать значение по ключу

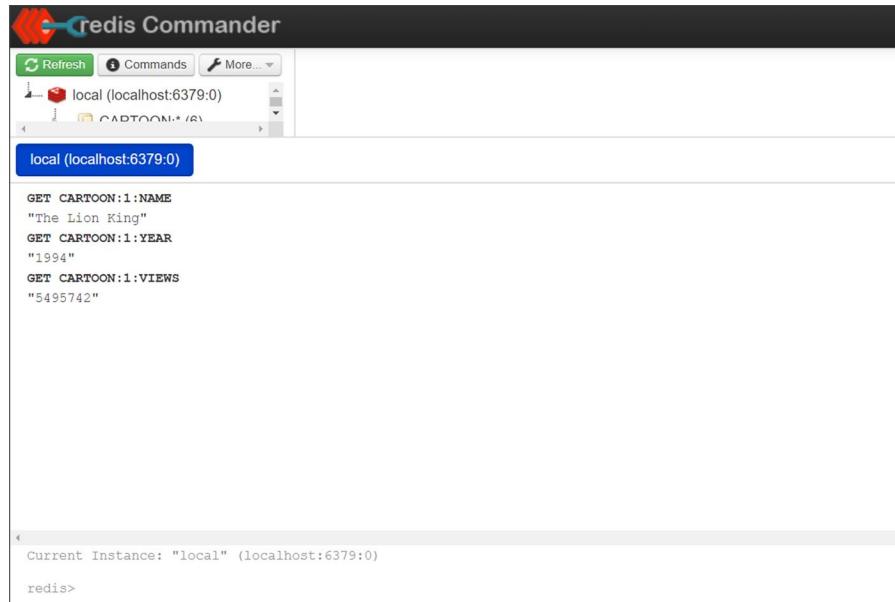


атрибут сопоставляются имена, годы выпуска и количество просмотров, соответствующих мультфильмам. Сопоставление производится с помощью команды SET.

```
redis> SET CARTOON:1:NAME "The Lion King"
redis> "OK"
redis> SET CARTOON:1:YEAR 1994
redis> "OK"
redis> SET CARTOON:1:VIEWS 5495742
redis> "OK"
redis> SET CARTOON:2:NAME "The Little Mermaid"
redis> "OK"
redis> SET CARTOON:2:YEAR 1989
redis> "OK"
redis> SET CARTOON:2:VIEWS 3295423
redis> "OK"

Current Instance: "local" (localhost:6379:0)
redis>
```

Есть и обратная операция. Ее действие также продемонстрировано на слайде. Это команда GET. Она позволяет достать из хранилища значение, соответствующее ключу. Можно достать имя мультфильма и любые другие значения.

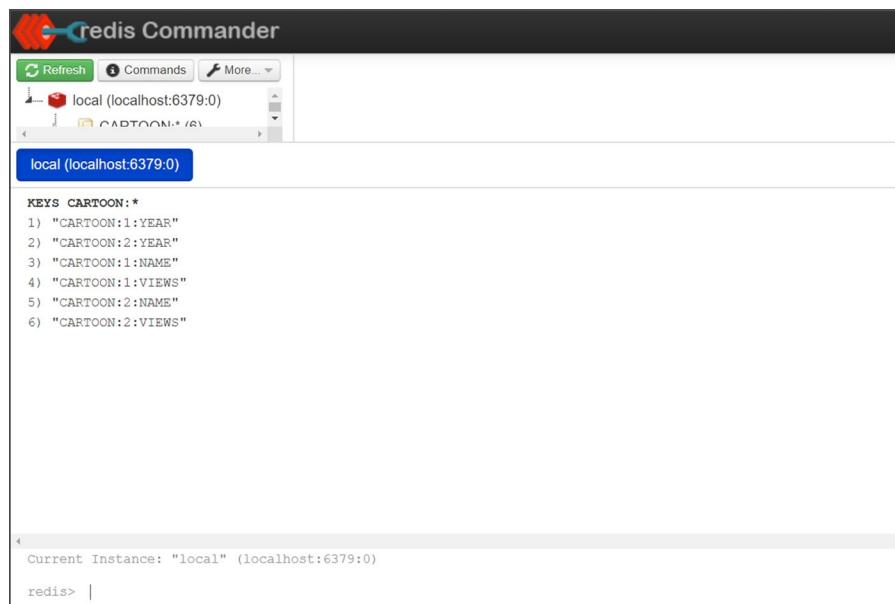


Redis Commander interface showing the results of a Redis command:

```
local (localhost:6379:0)
GET CARTOON:1:NAME
"The Lion King"
GET CARTOON:1:YEAR
"1994"
GET CARTOON:1:VIEWS
"5495742"

Current Instance: "local" (localhost:6379:0)
redis>
```

Кроме того, в списке основных команд есть команда **KEYS**. Она позволяет выбирать значения ключей в соответствии с заданным шаблоном. На слайде вы можете видеть, как выбираются все ключи с префиксом **CARTOON**.



Redis Commander interface showing the results of a Redis command:

```
local (localhost:6379:0)
KEYS CARTOON:*
1) "CARTOON:1:YEAR"
2) "CARTOON:2:YEAR"
3) "CARTOON:1:NAME"
4) "CARTOON:1:VIEWS"
5) "CARTOON:2:NAME"
6) "CARTOON:2:VIEWS"

Current Instance: "local" (localhost:6379:0)
redis> |
```

2 Структуры данных. Строки и хеши

В этом и следующем разделах мы познакомимся с основными структурами данных Redis и возможными способами их использования. Эти структуры, как уже говорилось ранее, принято делить на 5 типов:

- строки;
- хэши;
- списки;
- множества;
- упорядоченные множества.

В этом разделе мы познакомимся со строками и хэшами.

Итак, начнем со строк. Строки являются самой простой структурой данных в Redis. Когда вы задаете пару ключ-значение при помощи команды `SET`, вы задаете именно строку. Строки должны иметь уникальные имена (т.е. ключи), а значения строк могут быть какими угодно. В качестве значений, в частности, могут использоваться числа. Именно поэтому, эти значения еще

Примеры строк

- user1
- Anna
- 68049
- 20/03/17



иногда называют не просто строками, а «скалярными величинами».

Мы уже рассматривали типичный пример использования строк – хранение значений объектов с определенными ключами. Это именно то, с чем приходится сталкиваться чаще всего.

```

redis> SET CARTOON:1:NAME "The Lion King"
"OK"
redis> SET CARTOON:1:YEAR 1994
"OK"
redis> SET CARTOON:1:VIEWS 5495742
"OK"
redis> SET CARTOON:2:NAME "The Little Mermaid"
"OK"
redis> SET CARTOON:2:YEAR 1989
"OK"
redis> SET CARTOON:2:VIEWS 3295423
"OK"

Current Instance: "local" (localhost:6379:0)
redis>

```

Однако Redis не просто хранит и выдает строковые значения, но и позволяет выполнять более сложные манипуляции над строками. Например, команда **STRLEN** используется для вычисления длины строкового значения, связанного с ключом; команда **GETRANGE** – возвращает подстроку, **APPEND** добавляет значение к концу существующей строки, ассоциированной с указанным ключом (или создает новое значение, если указанный ключ не определен) и выдает в качестве результата длину строки.

Команды для работы со строками

STRLEN <key>

– вычисляет длину значения

GETRANGE <key> <begin> <end>

– возвращает подстроку

APPEND <key> <value>

– добавляет введенное значение

и выдает длину строки



Продемонстрируем эти команды в действии. Напомним, что ранее ключу **CARTOON:1:NAME** было сопоставлено значение **"The Lion King"**. Вычислим длину соответствующего значения с помощью команды **STRLEN**. Как видите, получилось ожидаемое значение – 13.

С помощью команды **GETRANGE** вырежем подстроку, которая начинается с 4 символа, а заканчивается 7. Увидим, что результат соответствует подстроке **"Lion"**.

```

STRLEN CARTOON:1:NAME
13
GETRANGE CARTOON:1:NAME 4 7
"Lion"
APPEND CARTOON:1:NAME "-THE BEST CARTOON!"
31
GET CARTOON:1:NAME
"The Lion King-THE BEST CARTOON!"

Current Instance: "local" (localhost:6379:0)
redis> |

```

С помощью команды APPEND добавим к строке, ассоциированной с ключом строку THE BEST CARTOON". Увидим, что после добавления длина строки стала равна 31 символу.

И, наконец, с помощью команды GET проверим, чему стала равна строка, ассоциированная с ключом CARTOON:1 : NAME, после всех этих манипуляций.

Redis не придает смысла введенным значениям. Это действительно так в большинстве случаев. Тем не менее, некоторые команды предназначены лишь для строк, значения которых удовлетворяют определенным условиям. В качестве примера можно привести команды INCR, INCRBY, DECR и DECRBY. Они

Команды для работы со строками-числами

INCR <key>

– увеличить значение на 1

INCRBY <key> <step>

– увеличить значение на заданный шаг

DECR <key>

– уменьшить значение на 1

DECRBY <key> <step>

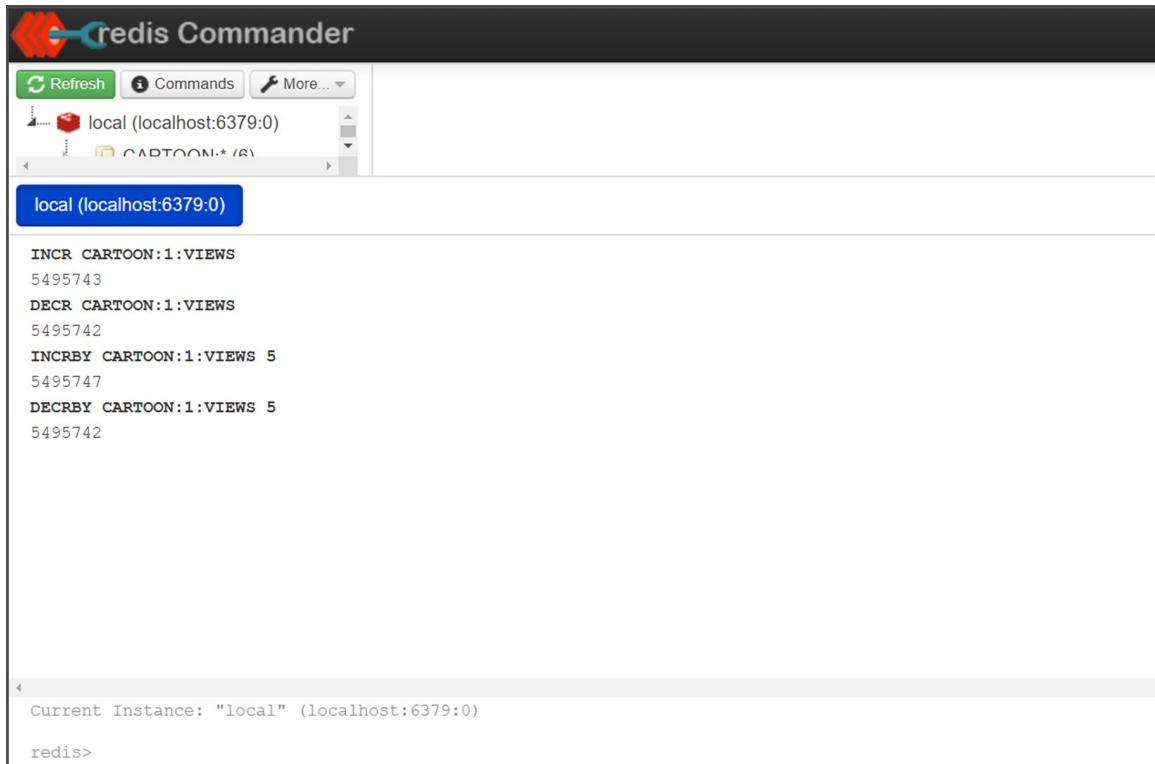
– уменьшить значение на заданный шаг



предназначены для выполнения увеличения и уменьшения значения значе-

ний строк только в том случае, если строковые значения являются числами. В случае неправильного использования этих команд Redis выдаст сообщение об ошибке.

Возьмем значение, ассоциированное с ключом `CARTOON:1:VIEWS`. Оно соответствует количеству просмотров мультфильма и является целочисленным. Увеличим это значение на 1 с помощью команды `INCR`. Результат увеличения виден на экране. Применим обратную команду `DECR` (которая соответствует



The screenshot shows the Redis Commander application window. At the top, there's a toolbar with 'Refresh', 'Commands', and 'More...'. Below it is a tree view showing a single node 'local (localhost:6379:0)' which contains a sub-node 'CARTOON:1 (6)'. The main area is a text console window with a dark background and white text. It displays the following command history:

```
INCR CARTOON:1:VIEWS
5495743
DECR CARTOON:1:VIEWS
5495742
INCRBY CARTOON:1:VIEWS 5
5495747
DECRBY CARTOON:1:VIEWS 5
5495742
```

At the bottom of the console, there's a status message 'Current Instance: "local" (localhost:6379:0)' and a prompt 'redis>'. The entire window has a black border.

уменьшению значения на 1) и вернемся к старому значению количества просмотров. Следующий слайд демонстрирует увеличение количества просмотров на 5 с помощью команды `INCRBY` с параметром 5. И обратную операцию, `DECRBY` с параметром 5, которая возвращает количество просмотров к исходному значению.

Строки Redis отлично подходят для аналитики. Разумеется, есть и более сложные команды для работы со строками. Например, команды работы с битовыми масками `SETBIT` и `GETBIT`. Эти команды крайне эффективно могут быть использованы для выполнения некоторых узко специализированных запросов. В частности, можно использовать эти две команды для эффективного ответа на вопрос «сколько уникальных посетителей было у нас сайте сегодня?». Утверждается, что для 128 миллионов пользователей ноутбук генерирует ответ менее чем за 50 мс и использует всего лишь 16 МБ памяти. Не так уж важно понимание того, как работают битовые маски, но важно понять, что строки в Redis являются гораздо более мощным инструментом, чем кажется на первый взгляд. Тем не менее, наиболее частыми примерами

использования строк являются именно те, что мы видели: хранение объектов (простых или сложных) и разного рода счетчиков.

Следующая структура данных – **хеши**. Хеши – хороший пример того, почему называть Redis хранилищем пар ключ-значение не совсем корректно. Хэши во многом похожи на строки. Важным отличием является то, что они предоставляют дополнительный уровень адресации для данных сложных структур, так называемые – **хеш- поля**.

Рассмотрим этот вопрос подробнее. В предыдущем разделе мы сохранили информацию о мультфильмах, используя сложную систему формирования ключей (<тип объекта>:<номер объекта>:<имя поля>). Этот способ

CARTOON: 1 : NAME

<тип объекта> <# объекта> <имя поля>



идентификации атрибутов хранимых документов с помощью большого количества ключей специального вида не всегда представляется достаточно удобным (слишком много ключей).

Конечно, мы могли бы представить всю эту информацию и с помощью одного ключа, а значение, ему соответствующее, представить в JSON формате. Например, так, как это представлено на слайде. Однако в этом случае

{name: "The Lion King", year: 1994,
duration: 88, views: 5495742}



будет затруднен (а на уровне хранилища ключ-значение и просто невозможен) доступ к отдельным полям JSON-документа. Именно поэтому и были придуманы хеш-структуры.

Эквивалентами команд **SET** и **GET** для хеш-структур являются команды **HSET** и **HGET**. Кроме того, можно устанавливать значения сразу нескольких

полей, получать все поля со значениями, выводить список всех полей и удалять отдельные поля. На слайде приведен краткий список команд для хеш-структур.

Основные команды для хеш-структур

HSET <key> <field> <value>

– задает значение поля

HGET <key> <field>

– возвращает значение указанного поля

HMSET <key> <field> <value> <field> <value> ...

– задает значения для множества полей

HVALS <key>

– возвращает все значения хеш-полей

HKEYS <key>

– возвращает все имена хеш-полей

HDEL <key> <field>

– удаляет указанное хеш-поле



Рассмотрим несколько примеров использования этих команд. Ассоциируем с ключом **CARTOON:2** поле **NAME** и соответствующее ему значение "**The Lion King**". Добавим к этому же ключу еще одно поле – **YEAR** и сопоставим ему подходящее значение. Добавим к этому же ключу еще одно поле – количество просмотров и соответствующее значение.

The screenshot shows the Redis Commander interface. At the top, there's a toolbar with Refresh, Commands, and More... buttons. Below the toolbar, a tree view shows a local instance at localhost:6379:0 with a single key named CARTOON:2. The main pane displays the following Redis command history:

```
HSET CARTOON:2 NAME "The Lion King"
1
HSET CARTOON:2 YEAR 1994
1
HSET CARTOON:2 VIEWS 5495742
1
```

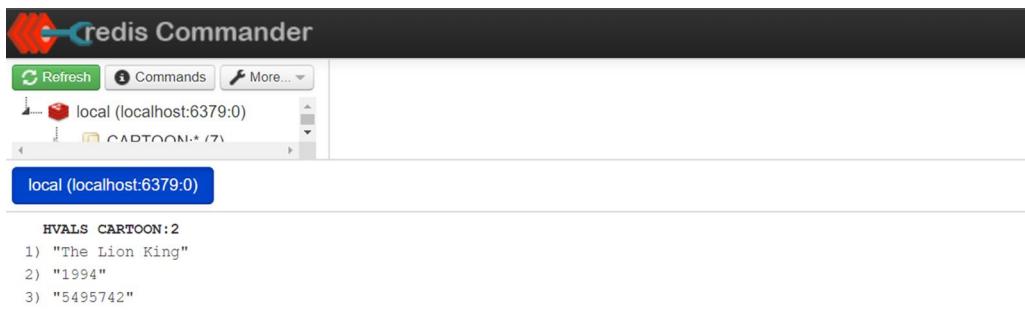
At the bottom of the interface, a status bar indicates the current instance is "local" (localhost:6379:0) and a redis> prompt is shown.

вим ему подходящее значение. Добавим к этому же ключу еще одно поле – количество просмотров и соответствующее значение.

Все три последние команды можно было объединить в одну команду HMSET и сразу сопоставить указанному ключу поля и ассоциированные с ними значения так, как это показано на слайде.



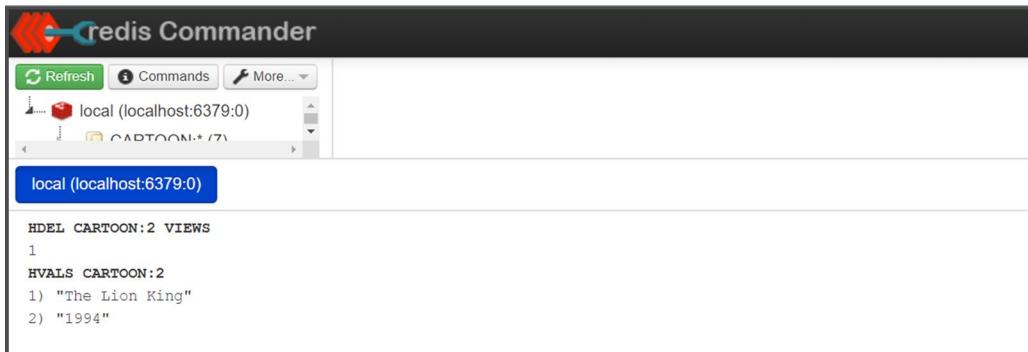
Увидеть названия хэш-полей можно с помощью команды HKEYS.



А, собственно, значения всех хэш-полей с помощью команды HVALS.



И, наконец, последняя из упомянутых команд – команда HDEL может использоваться для удаления хеш-полей и ассоциированных с ними значений. На слайде вы можете видеть удаление поля, содержащего сведения о количестве просмотров мультфильма. Убедитесь в том, что удаление, действительно, произошло можно с помощью команды HVALS.



Итак, вместо того чтобы хранить данные о мультфильме в виде одного неделимого значения или с использованием большого количества ключей, мы можем использовать хеш-структуры для более детального представления объектов. Преимуществом будет возможность извлечения, изменения и удаления отдельных частей объекта без необходимости читать и записывать все значения объекта целиком.

3 Структуры данных. Списки и множества

Рассмотрим очередную структуру данных – **справки**. Справки позволяют хранить и обрабатывать массивы последовательных значений для заданного ключа.

Можно добавлять значения в список, получать первое или последнее значение из списка, определять длину списка и манипулировать значениями с заданными индексами. Справки сохраняют порядок своих значений и

Основные команды для работы со списками

RPUSH <key> <value>

– добавляет новое значение в список

LLEN <key>

– определяет длину списка

RPOP <key>

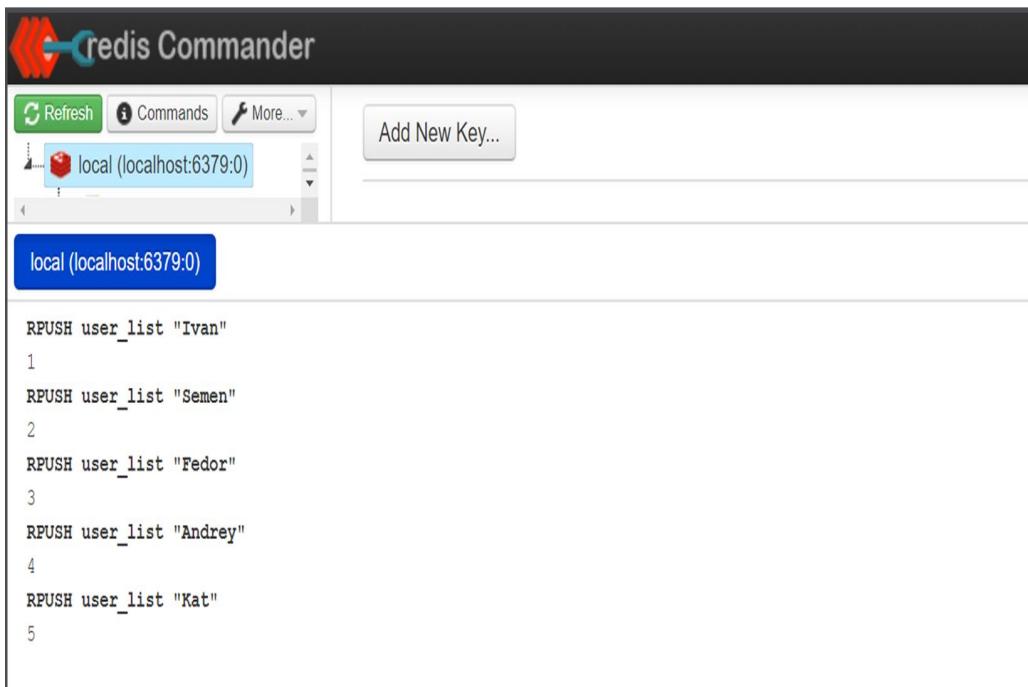
– возвращает последнее добавленное значение и удаляет его из списка



имеют эффективные операции с использованием индексов. Например, можно создать список `user_list`, содержащий зарегистрировавшихся на нашем

сайте пользователей. Продемонстрируем приемы работы со списками на конкретном примере.

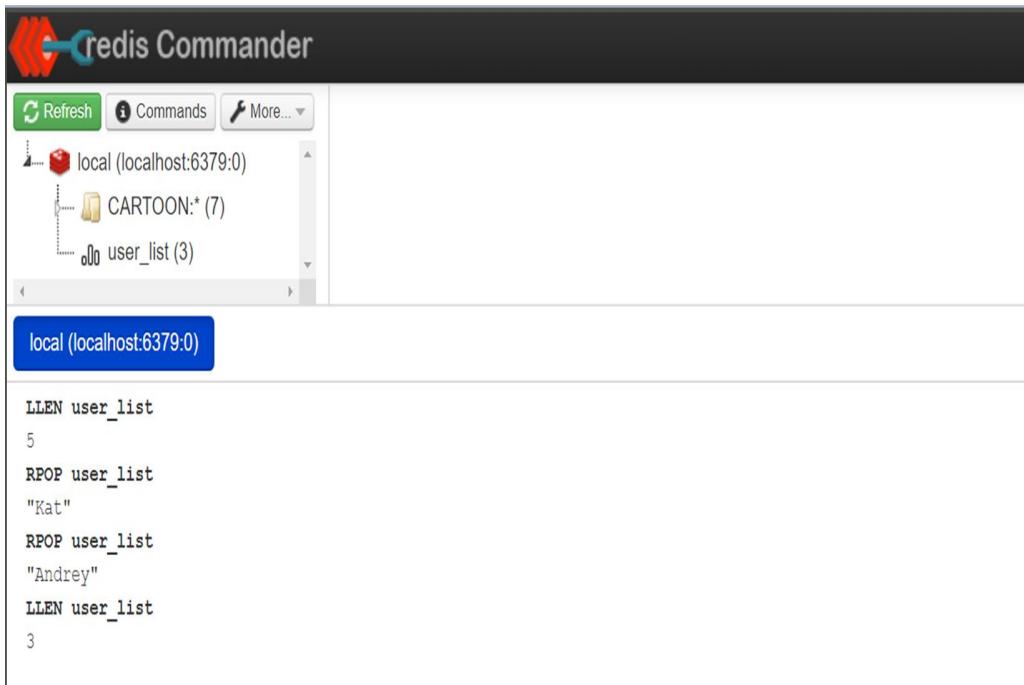
Воспользуемся командой RPUSH и добавим к списку user_list пользователя Иван. При этом, если к моменту выполнения этой команды, список еще не существует, то он будет создан автоматически. Добавим к списку еще одного пользователя – Семена. И еще Федора, Андрей, и Кэт.



Затем, с помощью команды LLEN выясним длину списка. Она равна пяти, так как к списку последовательно были добавлены 5 имен пользователей. С помощью команды RPOP выберем и удалим из списка верхнее (то есть последнее добавленное значение). Этим значением окажется пользователь по имени Кэт. Выберем и удалим из списка следующее значение (оказавшееся на вершине списка). Им окажется пользователь Андрей. И, наконец, с помощью команды LLEN выясним длину списка после всех проведенных операций. Как и ожидалось – она равна 3.

Сценарий, который мы продемонстрировали, является типичным примером использования этой структуры данных. Особенностью этого типа структур является то, что один ключ ссылается на список определенным образом упорядоченных и обрабатываемых значений. Значения могут быть чем угодно. Можно использовать списки для хранения записей журнала или пути, по которому пользователь перемещается по вашему сайту. Если вы создаете игру, вы можете использовать список для хранения последовательности действий пользователя и т.п.

Еще одна структура данных – **множества**. Множества используются для хранения уникальных неупорядоченных значений. Redis предостав-



ляют для манипуляций над множествами классический набор теоретико-множественных операций (объединение, пересечение, разность и т.п.). Множества не упорядочены по определению. Классическими примерами множеств могут быть группы друзей.

Рассмотрим подробнее список стандартных команд для работы с множествами.

Стандартные команды для работы с множествами

SADD <key> <value>

– добавляет к множеству новые значения

SCARD <key>

– определяет мощность множества (количество элементов)

SMEMBERS <key>

– выбирает все элементы множества

SINTER <key1> <key2>

– выполняет операцию пересечения множеств

SINTERSTORE <key1> <key2> <key3>

– выполняет операцию пересечения множеств <key2> и <key3> и сохраняет результат в <key1>



Стандартные команды для работы с множествами

`SDIFF <key1> <key2>`

– выполняет операцию вычитания множеств

`SDIFFSTORE <key1> <key2> <key3>`

– выполняет операцию вычитания множеств <key2> и <key3> и сохраняет результат в <key1>

`SUNION <key1> <key2>`

– выполняет операцию объединения множеств

`SUNIONSTORE <key1> <key2> <key3>`

– выполняет операцию объединения множеств <key2> и <key3> и сохраняет результат в <key1>



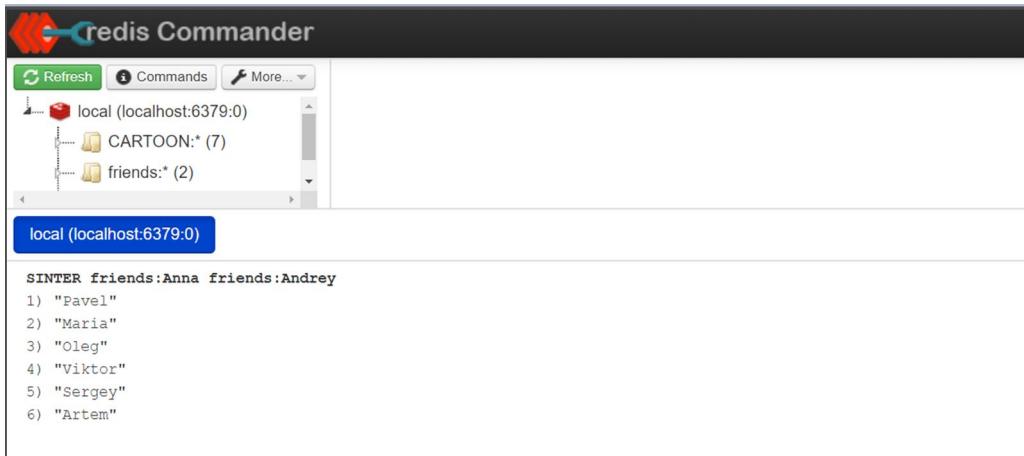
Теперь продемонстрируем примеры использования операций над множествами на примере поиска общих друзей Анны и Андрея. Для начала зададим множество друзей Анны и Андрея. Для создания множества друзей Анны используем команду `SADD`. Аналогичным образом создали список друзей Андрея. Теперь можем выяснить количество друзей. Определим количество

The screenshot shows the Redis Commander interface. The left sidebar lists keys under 'local (localhost:6379:0)': 'CARTOON:*' (7) and 'friends:*' (2). The main window displays the command history:

```
SADD friends:Anna Semen Ivan Sergey Pavel Viktor Maria Anton Oleg Artem Kat Ilya John
12
SADD friends:Andrey Sergey Pavel Viktor Maria Alexander Oleg Artem Eva Roman Sofia Inessa Inga
12
SCARD friends:Anna
12
SCARD friends:Andrey
12
```

друзей Анны с помощью команды `SCARD`. Напомним, что эта команда выдает кардинальность множества, то есть количество элементов. Как видим – у Анны 12 друзей. Аналогичным образом определяем количество друзей Андрея. Их тоже 12. Далее попробуем поэкспериментировать с теоретико-множественными операциями.

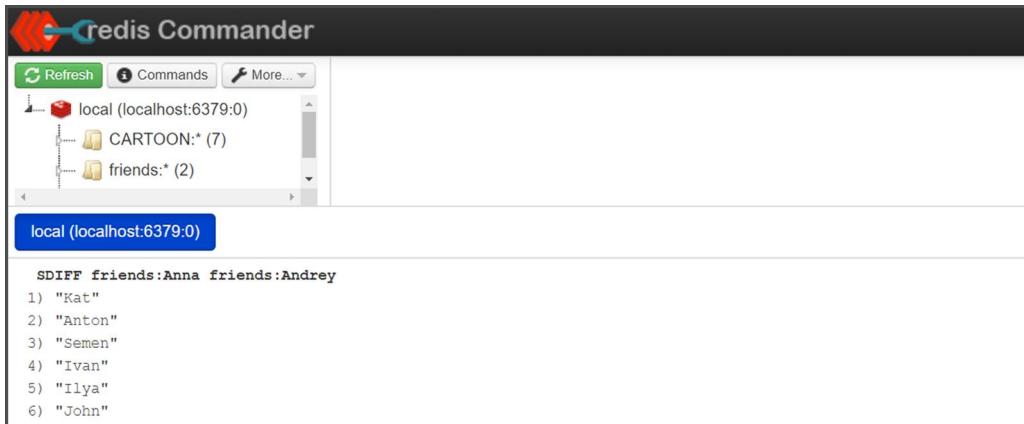
С помощью команды **SINTER** (что соответствует пересечению множеств) определили общих друзей Анны и Андрея. Результат вы можете видеть на экране.



The screenshot shows the Redis Commander interface. On the left, there's a tree view of databases: 'local (localhost:6379:0)' which contains 'CARTOON:' (7) and 'friends:' (2). The 'friends:' database is selected. In the main pane, the command 'SINTER friends:Anna friends:Andrey' is run, and its output is displayed:

```
SINTER friends:Anna friends:Andrey
1) "Pavel"
2) "Maria"
3) "Oleg"
4) "Viktor"
5) "Sergey"
6) "Artem"
```

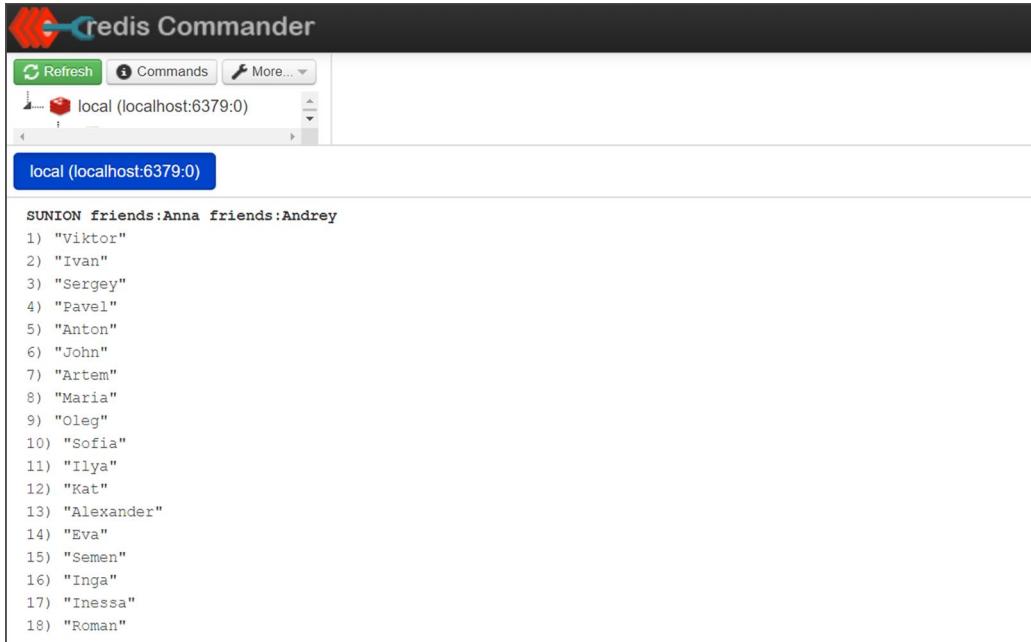
С помощью команды **SDIFF** (что соответствует вычитанию множеств) определили друзей Анны, с которыми не дружит Андрей. Результат команды — личные друзья Анны.



The screenshot shows the Redis Commander interface. The tree view is identical to the previous one. In the main pane, the command 'SDIFF friends:Anna friends:Andrey' is run, and its output is displayed:

```
SDIFF friends:Anna friends:Andrey
1) "Kat"
2) "Anton"
3) "Semen"
4) "Ivan"
5) "Ilya"
6) "John"
```

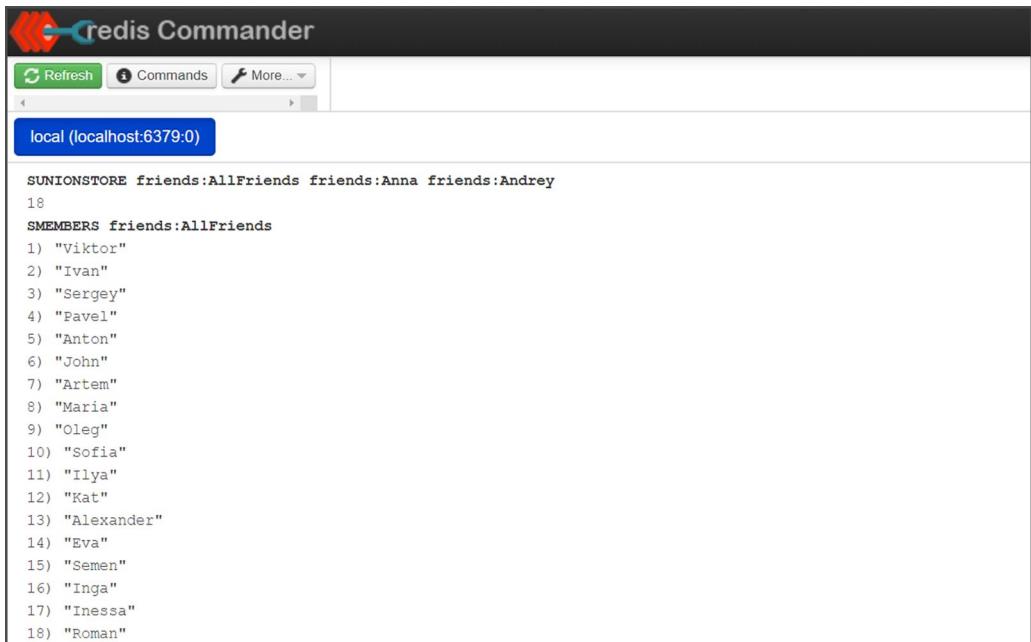
С помощью команды **SUNION** (что соответствует операции объединения множеств) определили всех друзей Анны и Андрея.



The screenshot shows the Redis Commander interface with a single connection named "local (localhost:6379:0)". In the command input field, the command `SUNION friends:Anna friends:Andrey` was entered. The output window displays a list of 18 names, each preceded by a number from 1 to 18, representing the combined set of friends for both Anna and Andrey.

```
SUNION friends:Anna friends:Andrey
1) "Viktor"
2) "Ivan"
3) "Sergey"
4) "Pavel"
5) "Anton"
6) "John"
7) "Artem"
8) "Maria"
9) "Oleg"
10) "Sofia"
11) "Ilya"
12) "Kat"
13) "Alexander"
14) "Eva"
15) "Semen"
16) "Inga"
17) "Inessa"
18) "Roman"
```

С помощью команды **SUNIONSTORE** (что соответствует операции объединения множеств) не только определили всех друзей Анны и Андрея, но и сохранили результат в множестве с ключом **friends:AllFriends**. В результате работы команды выводится количество элементов в сохраненном множестве. Обратите внимание, что ключ этого множества упоминается как первый operand команды **SUNIONSTORE**. С помощью команды **SMEMBERS**, которая



The screenshot shows the Redis Commander interface with a single connection named "local (localhost:6379:0)". First, the command `SUNIONSTORE friends:AllFriends friends:Anna friends:Andrey` was run, which resulted in 18 elements being added to the set at key "friends:AllFriends". Then, the command `SMEMBERS friends:AllFriends` was run to list all 18 members of the set.

```
SUNIONSTORE friends:AllFriends friends:Anna friends:Andrey
18
SMEMBERS friends:AllFriends
1) "Viktor"
2) "Ivan"
3) "Sergey"
4) "Pavel"
5) "Anton"
6) "John"
7) "Artem"
8) "Maria"
9) "Oleg"
10) "Sofia"
11) "Ilya"
12) "Kat"
13) "Alexander"
14) "Eva"
15) "Semen"
16) "Inga"
17) "Inessa"
18) "Roman"
```

в данном случае является аналогом команды **GET** применительно к множествам, вывели содержимое множества, полученного в результате предыду-

щей операции объединения. Хочется обратить внимание, что все теоретико-множественные операции Redis выполняет очень эффективно.

И, наконец, последняя из перечисленных структур Redis – **упорядоченные множества**. Это, пожалуй, – самая мощная из структур Redis. Упорядоченные множества очень похожи на множества, но имеют дополнительные поля в виде рангов. Ранги предоставляют возможности для упорядочивания элементов множества.

Основные команды для упорядоченных множеств

`ZADD <key> <rank-value> <rank-value> ...`

– формирует ранжированное множество

`ZCOUNT <key> <rank-from> <rank-to>`

– выдает количество элементов множества с указанным диапазоном рангов

`ZRANK <key> <value>`

– выдает порядковый номер заданного значения в ранжированном списке элементов множества



Приведем примеры использования команд для ранжированных множеств. С помощью команды ZADD создали ранжированный список друзей Арины. Обратите внимание на то, в каком порядке перечисляются операнды. Сначала упоминается ранг элемента множества, и только потом элемент, ему соответствующий. Например, у Ивана ранг – 2, у Сергея – 4 и т.п. Сле-

The screenshot shows the Redis Commander application. The left pane displays a tree structure of databases: 'local (localhost:6379:0)' which contains 'CARTOON:' (7 elements) and 'friends:' (4 elements). The right pane shows the command history for the 'local' database:

```
ZADD friends:Arina 2 Ivan 4 Sergey 3 Pavel 6 Viktor 5 Marina 7 Anton 9 Olga 11 Semen 10 Ilya  
9  
ZCOUNT friends:Arina 9 11  
3  
ZRANK friends:Arina Viktor  
4
```

дующий пример демонстрирует агрегатные возможности Redis. С помощью команды ZCOUNT определили количество друзей Арины с рангом от 9 до 11. Как оказалось – их трое. Последняя из упомянутых команд – определение

ранга для элемента множества. Для определения ранга используется команда `ZRANK`. Обратите внимание, что в качестве ранга она выдает не то число, которое ему было приписано при определении, а его порядковый номер в упорядоченном по возрастанию рангов множестве (причем нумерация начинается с 0). Так, например, Виктор, которому формально в качестве ранга соответствует значение 6, в результате работы команды получил значение 4. Попробуем объяснить этот результат.

Упорядочим друзей Арины по рангам и получим отсортированный список, который виден на слайде. Каждому элементу списка сопоставим номер. Нумерацию начнем с нуля. У Ивана будет номер 0, а Виктору в этом списке, действительно, соответствует номер 4.

Друзья Арины (упорядоченные по рангам)

- 0: Ivan(2)
- 1: Pavel(3)
- 2: Sergey(4)
- 3: Marina(5)
- 4: Viktor(6)**
- 5: Anton(7)
- 6: Olga(9)
- 7: Ilya(10)
- 8: Semen(11)



Самым очевидным примером использования упорядоченных множеств являются системы рейтингов. На самом деле, упорядоченные множества подойдут для любых случаев, когда вы имеете значения, которые вы хотите упорядочить, используя для этого целочисленные «весовые коэффициенты», и которыми вы хотите управлять на основании этих коэффициентов.

4 За пределами структур данных

Несмотря на то, что пять структур данных формируют основу Redis, в системе также есть команды, не относящиеся к структурам данных. Например:

- команды задающие/снимающие ограничения по времени существования ключей;
- команды, связанные с подпиской и публикаций сообщений;
- команды сортировки.

В этом разделе мы рассмотрим именно эти команды. Начнем с команд, задающих временные ограничения для ключей. Redis позволяет назначать ключам срок существования. Вы можете использовать абсолютные значения времени (количество секунд, прошедших с 1 января 1970 года) или оставшееся время существования ключа в секундах. Такие команды оперируют ключами, поэтому не имеет значения, какая структура данных при этом используется. Кроме того, существуют команды, позволяющие узнать, сколько еще будет существовать указанный ключ и ассоциированное с ним значение. Есть команды, позволяющие снять ограничения по времени существования ключа. И есть команды, позволяющие задавать ограничения по времени в момент объявления ключа. Эта особенность Redis позволяет автоматически избавляться от значений, которые в силу особенностей той или иной предметной области, по истечении определенного времени морально устаревают. На

Команды, определяющие время существования ключей

TTL <key> – выводит оставшееся время существования ключа

PERSIST <key> – снимает ограничения по времени

EXPIRE <key> <number>

– задает оставшуюся продолжительность существования ключа в секундах

EXPIREAT <key> <number>

– задает абсолютное значение времени в секундах начиная с 1 января 1970 года

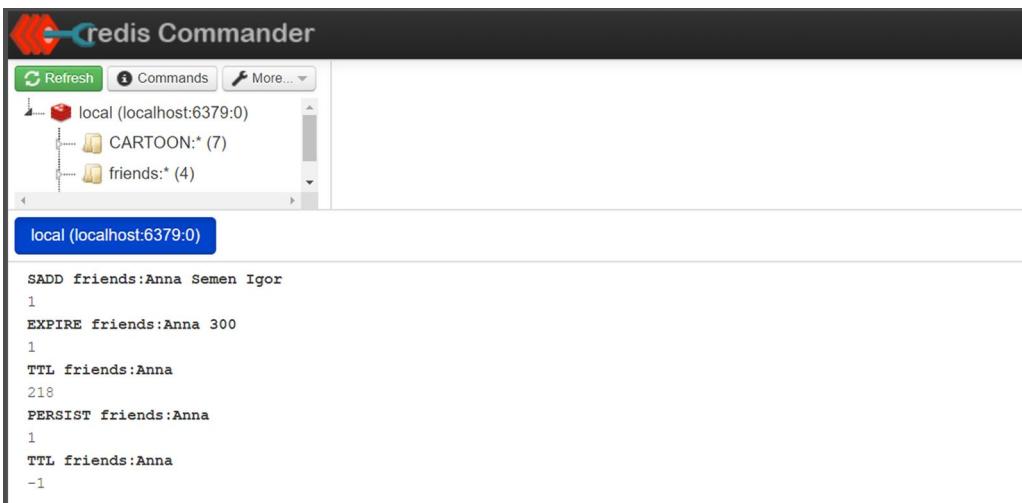
SETEX <key> <number> <value>

– задает строковое значение и указывает оставшуюся продолжительность существования



слайде приведен краткий перечень основных команд, определяющий время существования ключей.

Рассмотрим несколько примеров использования этих команд. В начале эксперимента зададим какой-нибудь ключ (например, `friends:Anna`) и со-поставим ему в качестве значения множество приятелей Анны. Затем с по-мощью команды `EXPIRE` зададим время существования ключа в секундах. В ответ получим немного странное значение 1. Это означает, что время суще-ствования было успешно установлено. Если бы указанный ключ не существо-вал — вернулось бы значение 0.

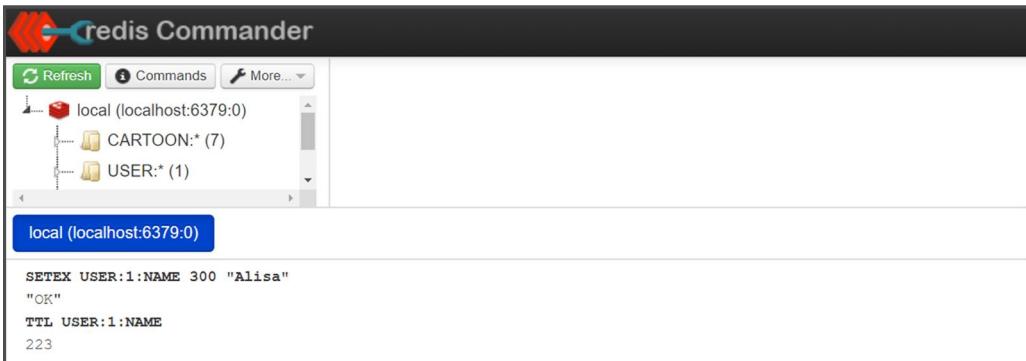


Следующая команда (`TTL`) позволяет определить оставшееся время су-ществования ключа. Если же такого ключа не существует – команда выдаст в качестве результата -2. Если ключ существует, но оставшееся время опреде-лить невозможно (ограничение по времени не установлено) – команда выдаст в качестве результата -1.

Следующая команда (`PERSIST`) снимает ограничения по времени для за-данного ключа. Значение 1 говорит о нормальном завершении команды, зна-чение 0 – что такого ключа не существует или срок действия не был устано-вен.

Убедиться в том, что ограничения были сняты, можно опять с помощью команда `TTL`. Она выдаст значение -1, что говорит о том, что ограничения по времени не установлены.

И, наконец, последняя команда из упомянутого списка (`SETEX`) позволяет одновременно задать ключ, ассоциированное значение в виде строки и ука-зать время существования ключа. Зададим время существования для ключа `USER:1:NAME` (300 секунд), затем через несколько секунд выполним команду `TTL` и убедимся, что ключ все еще существует, и время действия ключа было установлено.



В последнем случае следует помнить, что простое обновление значения по ключу сотрет старое ограничение по времени. Чтобы этого не случилось, при обновлении значений следует также указывать время существования ключей.

Следующая группа команд – команды, связанные с публикацией сообщений и подпиской на каналы. Для того, чтобы публиковать сообщения и подписываться на каналы существует ряд простых команд. Например, команда

Команды публикации сообщений и подписки на каналы

SUBSCRIBE <channel1> <channel2> <channel3> ...

– позволяет подписаться на указанные каналы

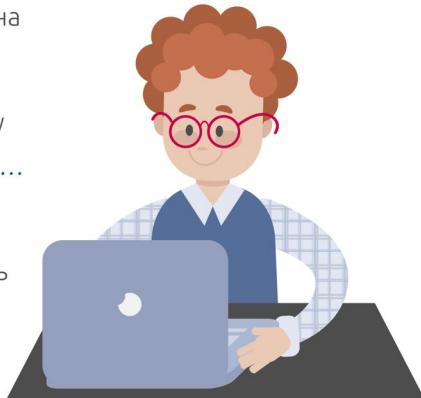
PSUBSCRIBE <pattern> – позволяет подписаться на каналы, соответствующие указанному шаблону

PUBLISH <channel> <message> – позволяет опубликовать сообщение по указанному каналу

UNSUBSCRIBE <channel1> <channel2> <channel3> ...

– позволяет прекратить подписку на указанные каналы

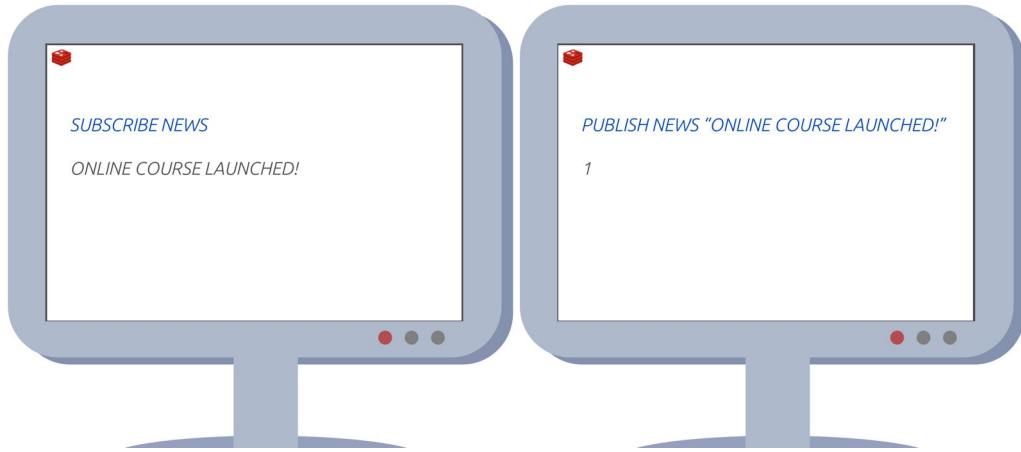
PUNSUBSCRIBE <pattern> – позволяет прекратить подписку на каналы, соответствующие указанному шаблону



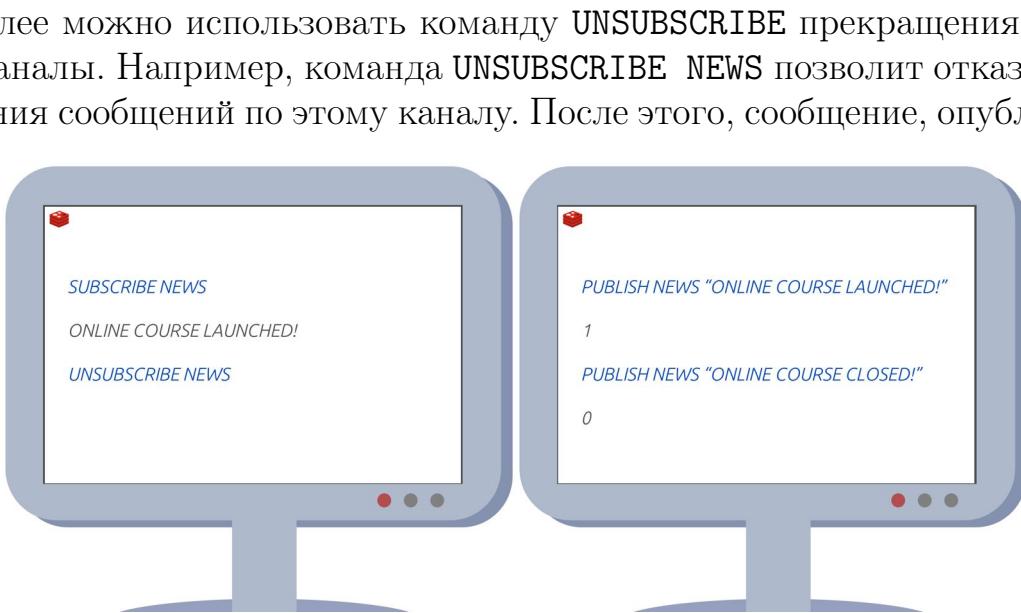
SUBSCRIBE позволяет подписаться на указанные каналы. Подписка в таком контексте означает готовность пользоваться получать сообщения, передаваемые по этому каналу. Команда **PUBLISH** позволяет публиковать сообщение по указанному каналу. Команда **UNSUBSCRIBE** позволяет прекратить подписку на указанные каналы. Кроме того, существует группа команд, позволяющая осуществлять подписку/публикацию в соответствии с шаблонами.

Вы можете сами убедиться в работоспособности этих команд на конкретных примерах если будете использовать Redis, который вы установили на своем компьютере. Откройте две различные сессии в Redis и выполните следующие команды.

В окне первой сессии подпишемся на канал (назовем его NEWS). Теперь, в окне второй сессии, опубликуем сообщение в канал NEWS. Обратите внимание, что команда PUBLISH вернула некоторое целочисленное значение. Это значение показывает число подписчиков, получивших сообщение. Если вы вернетесь в первое окно, вы должны увидеть сообщение, полученное через канал NEWS.



Далее можно использовать команду UNSUBSCRIBE прекращения подписки на каналы. Например, команда UNSUBSCRIBE NEWS позволит отказаться от получения сообщений по этому каналу. После этого, сообщение, опубликованное во втором окне, например: PUBLISH NEWS "ONLINE COURSE CLOSED!" не дойдет до первого пользователя, т.к. он к этому моменту уже отказался от подписки на этот канал.



Последняя из обещанных в этом фрагменте команд – команда сортировки. Это – одна из наиболее мощных команд в Redis. Она позволяет сортировать значения в списке, множестве или упорядоченном множестве. Как ни странно это звучит – это имеет смысл даже в последнем случае, так как элементы упорядоченных множеств упорядочены по значению своих весовых коэффициентов, а не по хранящимся значениям. В немного упрощенной форме синтаксис этой команды выглядит так, как показано на слайде. Можно

задать порядок сортировки с помощью знакомых опций ASC и DESC, указать количество выводимых значений, пропустить значения в начале выводимого списка и т.п.

Команда сортировки

SORT <key> [BY pattern][LIMIT offset count] [GET pattern][ASC/DESC] [ALPHA] [STORE destination]

<*key*> – ключ списка или множества
pattern – структура, указывающая фрагмент элемента списка или множества
offset – количество пропущенных значений в начале отсортированного списка
count – количество выводимых значений отсортированного списка
ASC – опция сортировки по возрастанию (используется по умолчанию)
DESC – опция сортировки по убыванию
ALPHA – опция лексикографической сортировки
destination – ключ, с которым будет ассоциировано полученное отсортированное множество



Продемонстрируем некоторые возможные варианты использования команды SORT. И начнем, разумеется, с простейших случаев. Для начала рассмотрим примеры использования команды SORT для множества числовых значений. С помощью команды SADD создадим множество, состоящее из простых чисел. В нашем множестве 8 элементов. Простые числа были добавлены в произвольном порядке. Воспользуемся командой SORT в простейшей фор-

```

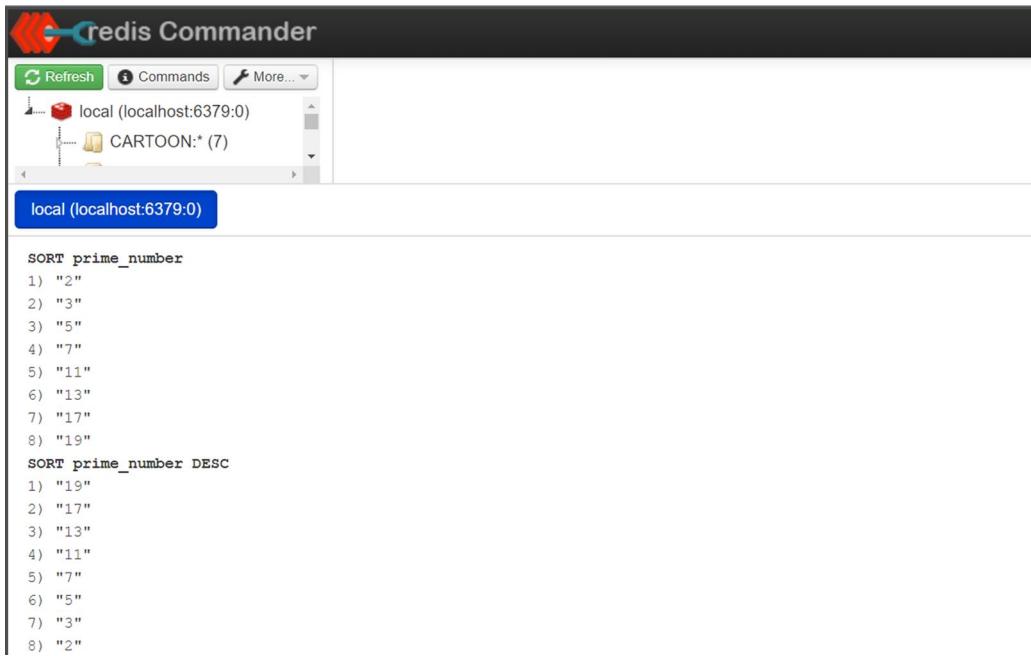
Redis Commander
local (localhost:6379:0)
SADD prime_number 13 3 19 17 11 7 2 5
8
SORT prime_number
1) "2"
2) "3"
3) "5"
4) "7"
5) "11"
6) "13"
7) "17"
8) "19"

```

ме без всяких дополнительных параметров и выведем это множество в виде упорядоченного списка по возрастанию. Мы не указывали опцию ASC. Тем не менее она будет использована, так как является опцией по умолчанию.

Затем выведем это множество в виде упорядоченного списка по убыванию. При этом в команде SORT нам придется воспользоваться дополнительной

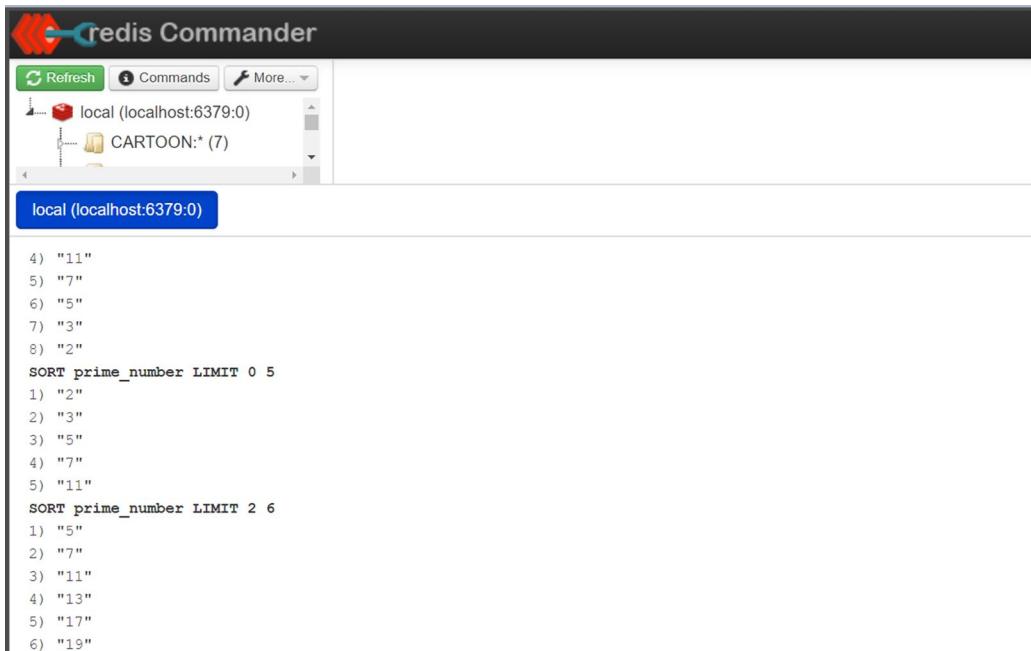
опцией DESC, которая указывает команде, что требуется выводить список по убыванию.



The screenshot shows the Redis Commander interface. In the left sidebar, there are two entries: 'local (localhost:6379:0)' and 'CARTOON:*(7)'. The main window displays a command history and output area. The output shows two commands: 'SORT prime_number' and 'SORT prime_number DESC'. The first command outputs a list of prime numbers from 2 to 19 in ascending order. The second command outputs the same list in descending order, starting with 19 and ending with 2.

```
SORT prime_number
1) "2"
2) "3"
3) "5"
4) "7"
5) "11"
6) "13"
7) "17"
8) "19"
SORT prime_number DESC
1) "19"
2) "17"
3) "13"
4) "11"
5) "7"
6) "5"
7) "3"
8) "2"
```

Напишем еще один запрос и выведем первые 5 простых чисел по возрастанию. Для этого пришлось использовать конструкцию LIMIT. Первый параметр после ключевого слова LIMIT укажет количество пропущенных значений в начале списка (в нашем случае – 0), второй – количество выводимых значений. В следующем запросе выведем 6 простых числа начиная с 3-го. Для

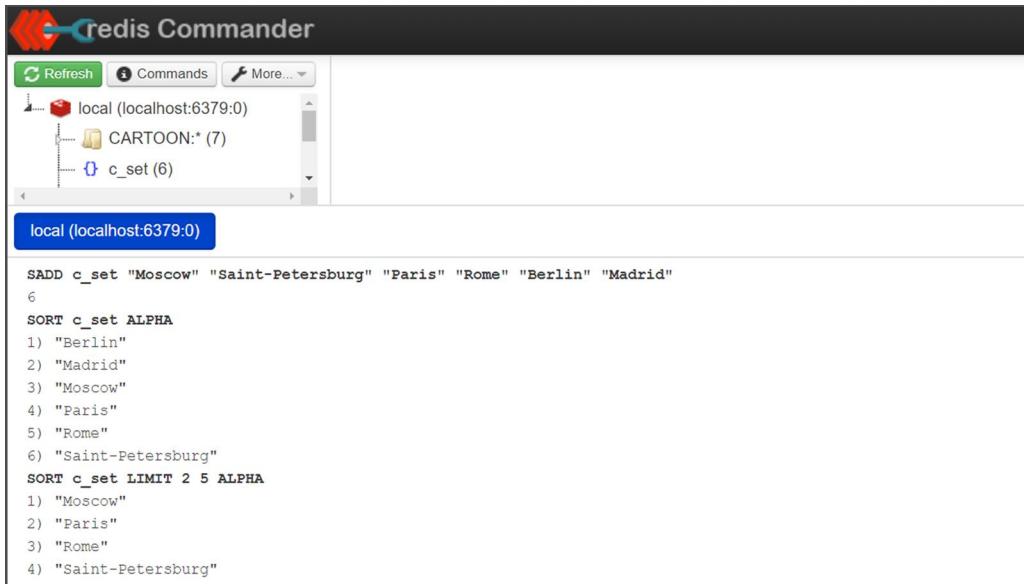


The screenshot shows the Redis Commander interface with the same sidebar entries. The main window shows three separate command executions. The first execution shows the first 5 prime numbers (2, 3, 5, 7, 11) using 'LIMIT 0 5'. The second execution shows the first 6 prime numbers (2, 3, 5, 7, 11, 13) using 'LIMIT 2 6'. The third execution shows the last 6 prime numbers (11, 13, 17, 19) using 'LIMIT 0 6'.

```
4) "11"
5) "7"
6) "5"
7) "3"
8) "2"
SORT prime_number LIMIT 0 5
1) "2"
2) "3"
3) "5"
4) "7"
5) "11"
SORT prime_number LIMIT 2 6
1) "5"
2) "7"
3) "11"
4) "13"
5) "17"
6) "19"
```

ЭТОГО ОПЯТЬ ИСПОЛЬЗУЕМ КОНСТРУКЦИЮ LIMIT. Первый параметр после ключевого слова LIMIT укажет количество пропущенных значений в начале списка (в этом случае – 2), второй – количество выводимых значений.

Теперь рассмотрим, как используется команда **SORT** для множества строк. Создадим множество из названий городов (Москва, Санкт-Петербург, Париж, Рим, Берлин, Мадрид). Затем отсортируем это множество по названиям городов по возрастанию в лексикографическом порядке. Для этого в команде **SORT** используем опцию **ALPHA**. Опцию **ASC** не указываем. Она используется по умолчанию. Затем напишем команду, в которой сказано от-



The screenshot shows the Redis Commander interface. In the left sidebar, there is a tree view with a single node 'local (localhost:6379:0)' expanded to show two sub-nodes: 'CARTOON:' (7) and 'c_set' (6). The main pane displays a command history window with the following content:

```
local (localhost:6379:0)

SADD c_set "Moscow" "Saint-Petersburg" "Paris" "Rome" "Berlin" "Madrid"
6
SORT c_set ALPHA
1) "Berlin"
2) "Madrid"
3) "Moscow"
4) "Paris"
5) "Rome"
6) "Saint-Petersburg"
SORT c_set LIMIT 2 5 ALPHA
1) "Moscow"
2) "Paris"
3) "Rome"
4) "Saint-Petersburg"
```

сортировать множество городов, пропустить первые два элемента, а затем вывести 5 элементов отсортированного множества. Результат вы можете видеть на слайде. Первые два элемента, действительно, были пропущены, а вот пяти элементов не нашлось, и команда вывела то, что было, т.е. оставшиеся 4 элемента множества.

Команда сортировки может быть применена и в значительно более сложных случаях, например, когда элементами списка или множества являются не скалярные значения, а объекты более сложной структуры.

Однако рассмотрение таких случаев, так же, как и многие другие аспекты использования Redis, не входит в программу нашего курса, а любознательным слушателям можно посоветовать читать официальную документацию Redis – <https://redis.io/documentation>.