Graph & Modern Databases

COMP1835

1

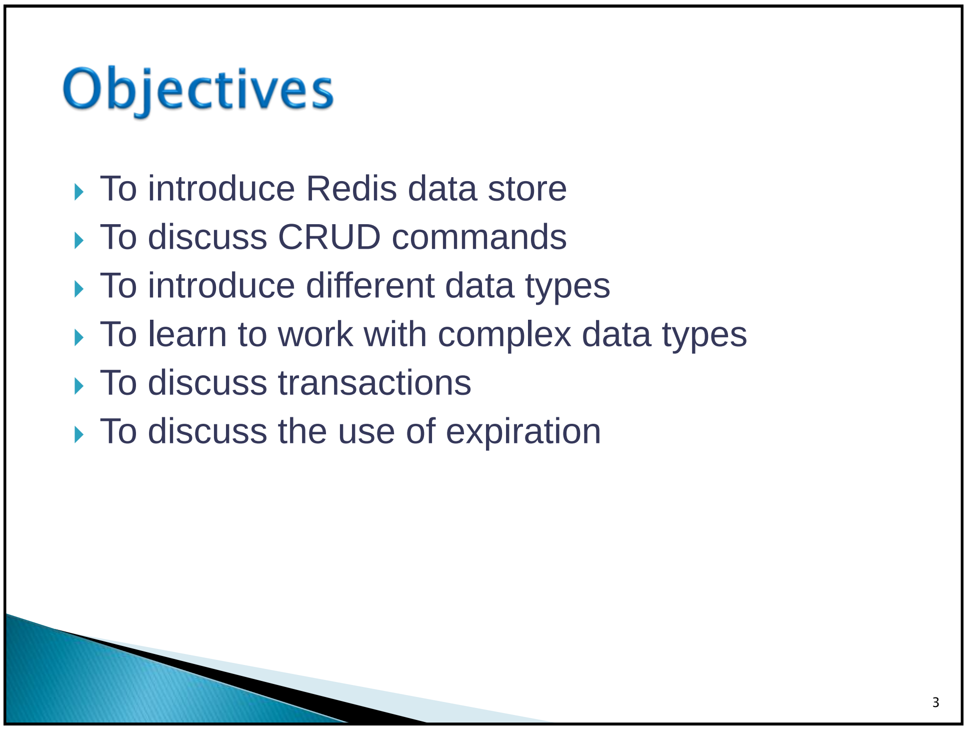
1

Redis

redis

2

2

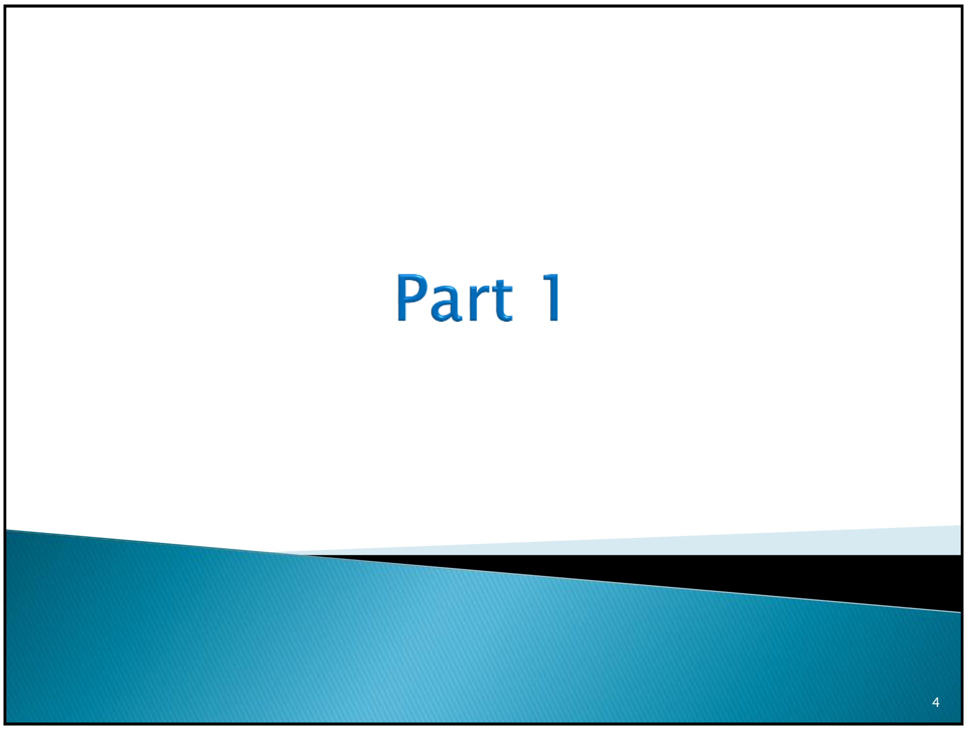


Objectives

- To introduce Redis data store
- To discuss CRUD commands
- To introduce different data types
- To learn to work with complex data types
- To discuss transactions
- To discuss the use of expiration

3

3



Part 1

4

4

Overview

- Redis is an open source, advanced **key-value store** and a solution for building high performance, scalable web applications.
- Redis stands for **RE**mote **DI**ctionary **S**erver.
- Redis has three main features that sets it apart:
  - Redis holds its database entirely in the memory, using the disk only for persistence.
  - Redis has a relatively rich set of data types when compared to many key-value data stores.
  - Redis can replicate data to any number of slaves.
- ‘Redis is like grease: it’s most often used to lubricate moving parts and keep them working smoothly by reducing friction and speeding up their overall function.

5

5

Facts about Redis

- Official Online Resources:
  - http://redis.io/.
- History:
  - Project started in 2009 by Salvatore Sanfilippo. Salvatore created it for his startup company LLOOGG (http://lloogg.com/). Though still an independent project, Redis primary author is employed by VMware, who sponsors its development.
- Technologies and Language:
  - Implemented in C.
- Access Methods:
  - Rich set of methods and operations. Can access via Redis command-line interface and a set of well-maintained client libraries for languages like Java, Python, Ruby, C, C++, Lua, Haskell, AS3, and more.
- Who Uses It:
  - Twitter, Github, Stackoverflow, Pinterest, Snapchat, Craigslist, Flickr, etc.
- In 2015 Redis has been ranked the #4 NoSQL database in user satisfaction and market presence based on user reviews.

6

6

Advantages

- Redis is exceptionally fast:
  - It can perform about 110,000 SETs per second, about 81,000 GETs per second.
- Redis supports rich data types:
  - It natively supports most of the datatypes such as list, set, sorted set, and hashes.
    - This makes it easy to solve a variety of problems since we know which problem can be handled better by which data type.
- Operations are atomic:
  - All Redis operations are atomic, which ensures that if two clients concurrently access, Redis server will receive the updated value.
- Redis is a multi-utility tool:
  - It can be used in a number of use cases such as caching, messaging-queues
  - Redis natively supports Publish/Subscribe, any short-lived data in your application, such as web application sessions, web page hit counts, etc.

7

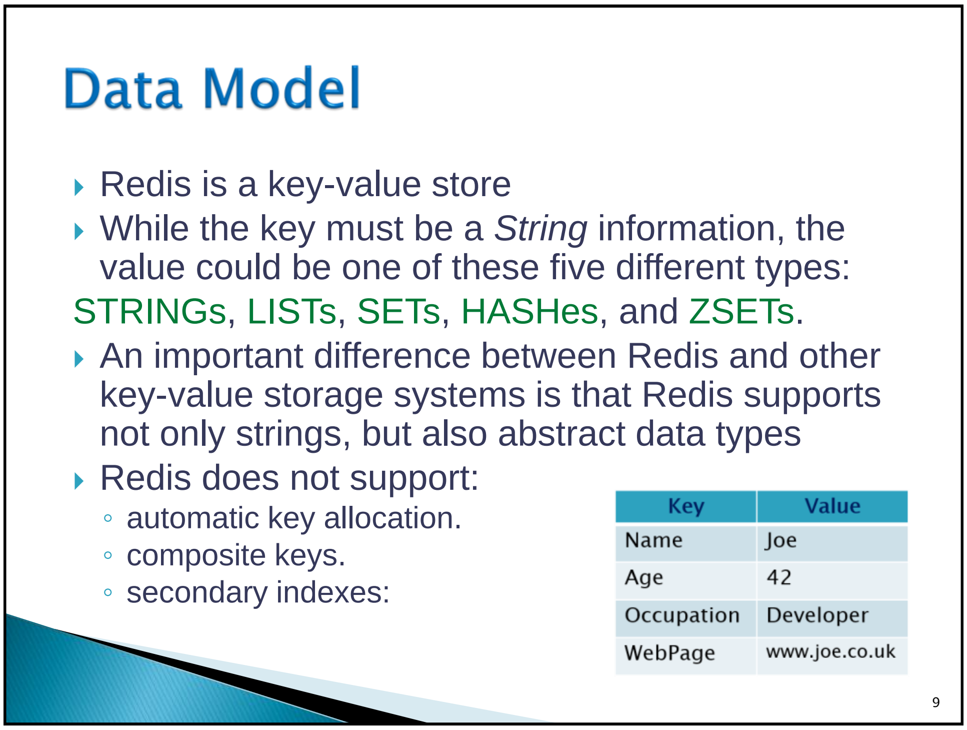
7

More ..

- Redis is not just a key-value store, but a different evolution path in the key-value DBs:
  - It supports more complex data types, though not to the degree that document-oriented database would
- Redis is an in-memory database but persistent on disk database
  - It represents a different trade off where very high write and read speed is achieved with the limitation of data sets that can't be larger than the memory.
  - To enhance the speed Redis purposely compromises the durability
    - In Redis, in the event of system failure or crash, Redis writes to disk but may fall behind and lose the data which is not stored.
- Redis supports set-based query operations but not with the granularity or type support you would find in a relational database.
- Redis is more of a toolkit of useful data structures algorithms and processes than a member of any specific database genre.

8

8



Data Model

- Redis is a key-value store
- While the key must be a *String* information, the value could be one of these five different types:

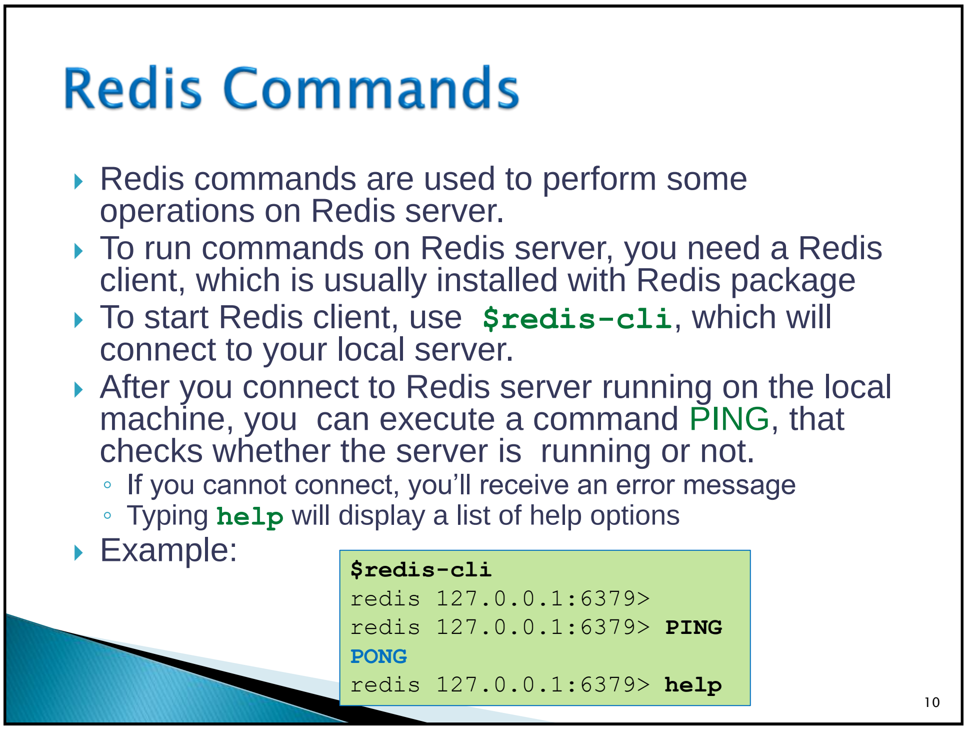
STRINGs, LISTs, SETs, HASHes, and ZSETs.

- An important difference between Redis and other key-value storage systems is that Redis supports not only strings, but also abstract data types
- Redis does not support:
  - automatic key allocation.
  - composite keys.
  - secondary indexes:

Key	Value
Name	Joe
Age	42
Occupation	Developer
WebPage	www.joe.co.uk

9

9



Redis Commands

- Redis commands are used to perform some operations on Redis server.
- To run commands on Redis server, you need a Redis client, which is usually installed with Redis package
- To start Redis client, use **`$redis-cli`**, which will connect to your local server.
- After you connect to Redis server running on the local machine, you can execute a command PING, that checks whether the server is running or not.
  - If you cannot connect, you'll receive an error message
  - Typing **`help`** will display a list of help options
- Example:

```
$redis-cli
redis 127.0.0.1:6379>
redis 127.0.0.1:6379> PING
PONG
redis 127.0.0.1:6379> help
```

10

10

CRUD commands

- Syntax:
```
127.0.0.1:6379> COMMAND KEY_NAME
```
- SET (Setting a Key):
```
127.0.0.1:6379> SET greeting "Hello World"
```
- GET (Getting a Key):
```
127.0.0.1:6379> GET greeting
"Hello World"
```
- DEL (Deleting a Key)
```
127.0.0.1:6379> GET greeting
"Hello World" // getting a key
127.0.0.1:6379> DEL greeting
(integer) 1 // key just got deleted
127.0.0.1:6379> GET greeting
(nil) // since key is deleted therefore, result is nil.
```

11

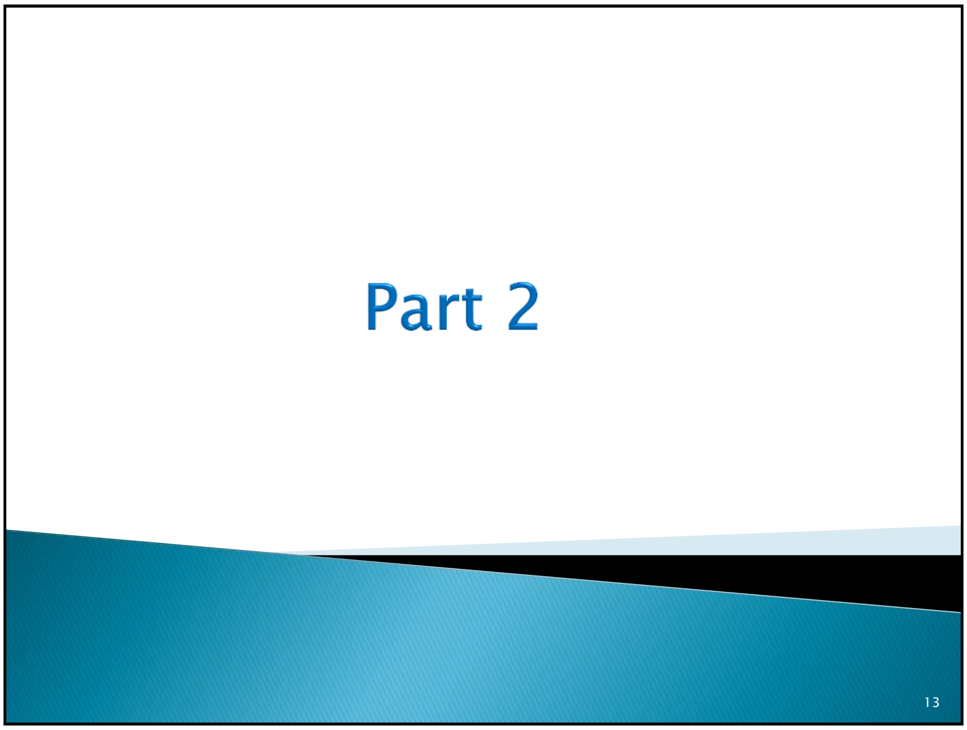
11

Questions

- Name two advantages of using Redis data store.
- What are the limitations of Redis data store?
- Does Redis provide both **speed and durability**?

12

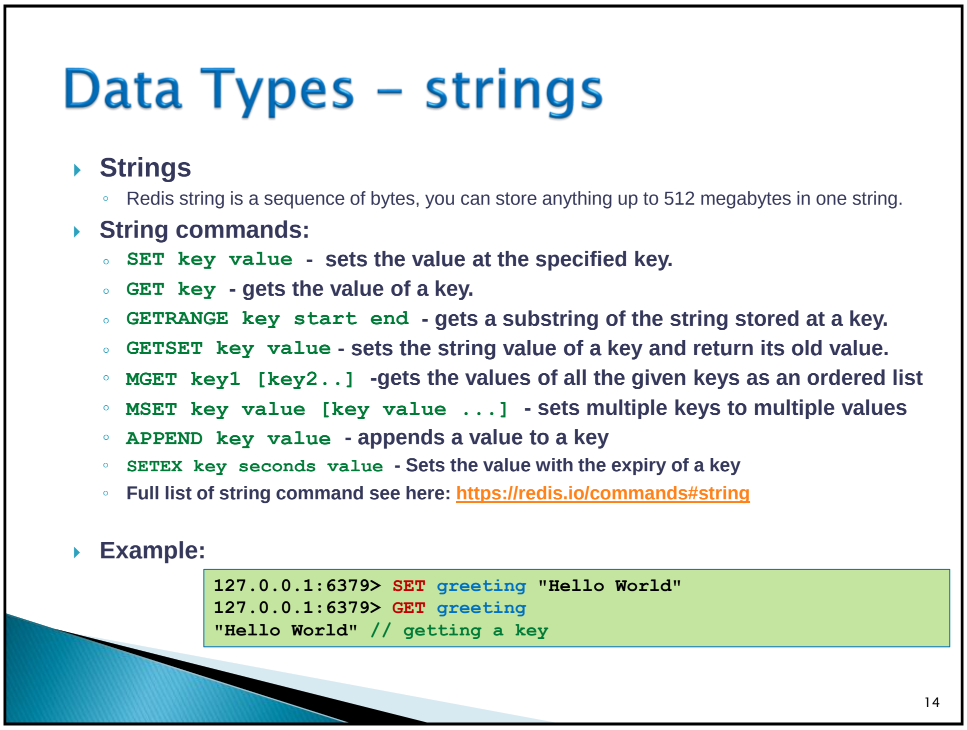
12



Part 2

13

13



Data Types – strings

- **Strings**
  - Redis string is a sequence of bytes, you can store anything up to 512 megabytes in one string.
- **String commands:**
  - **`SET key value` - sets the value at the specified key.**
  - **`GET key` - gets the value of a key.**
  - **`GETRANGE key start end` - gets a substring of the string stored at a key.**
  - **`GETSET key value` - sets the string value of a key and return its old value.**
  - **`MGET key1 [key2..]` -gets the values of all the given keys as an ordered list**
  - **`MSET key value [key value ...]` - sets multiple keys to multiple values**
  - **`APPEND key value` - appends a value to a key**
  - **`SETEX key seconds value` - Sets the value with the expiry of a key**
  - **Full list of string command see here: https://redis.io/commands#string**
- **Example:**

```
127.0.0.1:6379> SET greeting "Hello World"
127.0.0.1:6379> GET greeting
"Hello World" // getting a key
```

14

14

Data Types – integers

- Although Redis stores strings, it recognizes integers and provides some simple operations for them.
- For example, if we want to keep a running total of how many key/value pairs are in our dataset, we can create a count and then increment it with the **`INCR`** command.
- Example:

```
127.0.0.1:6379> SET count 2
OK
127.0.0.1:6379> INCR count
(integer) 3
127.0.0.1:6379> GET count
"3"
```

- Although **`GET`** returns count as a string, **`INCR`** recognized it as an integer and added one to it.
- If the value can't be resolved to an integer, Redis will return an error.
  - You can also increment by any integer (**`INCRBY`**) or decrement (**`DECR, DECRBY`**).

Complex Data Types

- Redis supports many complex data types, storing lists, hashes, sets, and sorted sets natively, which make it very attractive data store
- Different complex data types have appropriate commands associate with them, which generally follow a good pattern.
  - Set's commands begin with **`S`**,
  - Hashes' commands begin with **`H`**,
  - Sorted set's commands begin with **`Z`**.
  - List's commands generally start with either an **`L`** (for left) or an **`R`** (for right), depending on the direction of the operation (such as **`LPUSH`**).

Data Types – Hashes

- Hashes
  - A Redis hash is a collection of key value pairs. Redis Hashes are maps between string fields and string values. Hence, they are used to represent objects
- Hash commands:
  - `HGET key field` -Gets the value of a hash field stored at the specified key.
  - `HGETALL key` - Gets all the fields and values stored in a hash at the specified key
  - `HVALS key` - Gets all the values in a hash
  - `HLEN key` - Gets the number of fields in a hash
  - `HMGET key field1 [field2]` - Gets the values of all the given hash fields
  - `HSET key field value` - Sets the string value of a hash field
  - `HMSET key field1 value1 [field2 value2 ]` - Sets multiple hash fields to multiple values
  - Full list of hash command see here: https://redis.io/commands#hash
- Example:

```
127.0.0.1:6379> HMSET user:1 username nosql password nosqlpass
credits 15
OK
127.0.0.1:6379> HGETALL user:1
1) "username"
2) "nosql"
3) "password"
4) "nosqlpass"
5) "credits"
6) "15"
```

17

Data Types –Lists

- Lists
  - Redis Lists are simply lists of strings, sorted by insertion order. You can add elements to a Redis List on the head (L) or on the tail (R).
- Lists commands:
  - `LINDEX key index-` Gets an element from a list by its index
  - `LLEN key-` Gets the length of a list
  - `LPUSH key value1 [value2]` – Adds one or multiple values to the beginning of the list
  - `RPUSH key value1 [value2]` - Appends one or multiple values to the end of the list
  - `LRANGE key start stop -` Gets a range of elements from a list
  - Full list of list commands see here https://redis.io/commands#list
  - All list operations in Redis use a zero based index.

Example:

```
127.0.0.1:6379> LPUSH tutorials redis
(integer) 1
127.0.0.1:6379> LPUSH tutorials mongodb
(integer) 2
127.0.0.1:6379> LPUSH tutorials mysql
(integer) 3
127.0.0.1:6379> LRANGE tutorials 0 10
1) "mysql"
2) "mongodb"
3) "redis"
```

18

18

Data Types – Sets

- Sets
  - Redis Sets are **an unordered** collection of **unique** strings.
    - Unique means sets does not allow repetition of data in a key.
  - Sets are an excellent choice for performing complex operations between two or more key values, such as unions or intersections.
- Sets commands:
  - `SADD key member1 [member2]` **- Adds one or more members to a set**
  - `SCARD key` **- Gets the number of members in a set**
  - `SMEMBERS key` - Gets all the members in a set
  - Full list sets commands see here: https://redis.io/commands#set
  - Example:

```
127.0.0.1:6379> SADD tutorials redis
(integer) 1
127.0.0.1:6379> SADD tutorials mongodb
(integer) 1
127.0.0.1:6379> SADD tutorials mysql
(integer) 1
127.0.0.1:6379> SADD tutorials mysql
(integer) 0
127.0.0.1:6379> SMEMBERS tutorials
1) "mysql"
2) "mongodb"
3) "redis
```

19

Operations with SETS

- Create two sets

```
127.0.0.1:6379> SADD fruit1 apples oranges
(integer) 2
127.0.0.1:6379> SADD fruit2 apples pears
(integer) 2
```

- To find the intersection of sets – values that are in both, we use the **`SINTER`** command.

```
127.0.0.1:6379> SINTER fruit1 fruit2
1)"apples"
```

- To remove any matching values in one set from another by finding the difference, use **`SDIFF`**:

```
127.0.0.1:6379> SDIFF fruit1 fruit2
1) "pears"
```

- To build a union of sets with values from both (since it's a set, any duplicates are dropped), use **`SUNION:`**

```
127.0.0.1:6379> SUNION fruit1 fruit2
1) "apples"
2) "oranges"
3) "pears"
```

20

20

Data Types – Sorted Sets

- Sorted Sets
  - Redis Sorted Sets are similar to Redis Sets, non-repeating collections of Strings. The difference is, every member of a Sorted Set is associated with a score, that is used in order to take the sorted set ordered, from the smallest to the greatest score. While members are unique, the scores may be repeated.
- Sorted Sets commands:
  - `ZADD key score1 member1 [score2 member2]`-Adds one or more members to a sorted set, or updates its score, if it already exists
  - `ZCARD key` - Gets the number of members in a sorted set
  - `ZRANGE key start stop [WITHSCORES]`- Returns a range of members in a sorted set, by index
  - Full list sets commands see here: https://redis.io/commands#sorted_set
  - Example:

```
127.0.0.1:6379> ZADD tutorials 1 redis
(integer) 1
127.0.0.1:6379> ZADD tutorials 2 mongodb
(integer) 1
127.0.0.1:6379> ZADD tutorials 3 mysql
(integer) 1
127.0.0.1:6379> ZADD tutorials 4 mysql
(integer) 0
127.0.0.1:6379> ZRANGE tutorials 0 10 WITHSCORES
1) "redis"
2) "1"
3) "mongodb"
4) "2"
5) "mysql"
6) "4"
```

21

Questions

- What is **Redis-cli**?
- How can you save multiple values under one key in Redis?
- How to get value from Redis database?
- How will you delete a key from Redis?

22

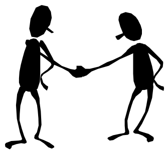
22

Part 3

23

23

Transactions



- Redis transactions allow the execution of a group of commands in a single step.
- Redis transactions have the following properties:
  - All commands in a transaction are sequentially executed as a single isolated operation.
  - It is not possible that a request issued by another client is served in the middle of the execution of a Redis transaction.
- Redis transaction are also atomic.
  - Atomic means either all of the commands or none are processed.

24

24

Transactions

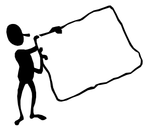
- In Redis transactions are wrapped in **MULTI** block atomic commands.
  - Wrapping several operations in a single block will complete either successfully or not at all.
- When using **MULTI**, the commands aren't actually executed when we define them. Instead, they are queued and then executed in sequence.
- Similar to **ROLLBACK** in SQL, you can stop a transaction with the **DISCARD** command, which will clear the transaction queue.
  - Unlike **ROLLBACK**, it won't revert the database; it will simply not run the transaction at all.
  - The effect is identical, although the underlying concept is a different mechanism (transaction rollback vs. operation cancellation).
- Example:

```
127.0.0.1:6379> MULTI
OK
127.0.0.1:6379> SET count 2
QUEUED
127.0.0.1:6379> SET student Patel
QUEUED
127.0.0.1:6379> INCR count
QUEUED
127.0.0.1:6379> EXEC
1)OK
2)(integer) 3
```

25

25

Namespaces



- Very often there is a need to separate keys by a namespace.
- In Redis a namespace is called a database and is keyed by number.
  - So far, we've always interacted with the default namespace 0 (also known as database 0).
  - To switch to another namespace you use **SELECT** command
  - You can move keys from one namespace to another with **MOVE** command.

```
127.0.0.1:6379> SET greeting hello
OK
127.0.0.1:6379> GET greeting
"hello"
127.0.0.1:6379> SELECT 1
OK
127.0.0.1:6379> GET greeting
(nil)
127.0.0.1:6379> SET greeting "ciao"
OK
127.0.0.1:6379> GET greeting
"ciao"
```

```
127.0.0.1:6379> SELECT 0
OK
127.0.0.1:6379> GET greeting
"hello"
127.0.0.1:6379> SELECT 1
OK
127.0.0.1:6379> GET greeting
"ciao"
```

26

26

Expiry

- A common use case for a key-value system like Redis is as a fast-access cache for data that's more expensive to retrieve or compute.
- Expiration helps keep the total key set from growing unbounded, by tasking Redis to delete a key value after a certain time has passed.
- Marking a key for expiration requires the **EXPIRE** command, an existing key, and a time to live in seconds.
  - Redis also provides a shortcut command called **SETEX** to set the keys with expiration.

Example:

```
127.0.0.1:6379> SET ice "I'm melting…"
OK
127.0.0.1:6379> EXPIRE ice 10      //setting 10 sec expiry
(integer) 1

//check immediately
127.0.0.1:6379> EXISTS ice
(integer) 1

//check after 10sec
127.0.0.1:6379> EXISTS ice
(integer) 0
```

27

Persistence

- Redis is very fast, but not very durable
- However, it provides a few persistence options:
  - **Snapshotting:**
    - Point-in-time snapshots of your dataset are created at specified intervals.
  - **The Append Only File:**
    - In that case Redis logs every write operation received by the server that will be played again at server startup, reconstructing the original dataset.
    - Commands are logged using the same format as the Redis protocol itself, in an append-only fashion. Redis is able to rewrite the log in the background when it gets too big.
  - It is possible to **combine both AOF** and **snapshots** in the same instance.
    - In this case, when Redis restarts the AOF file will be used to reconstruct the original dataset since it is guaranteed to be the most complete.
  - You can **disable persistence completely**, if you want your data to just exist as long as the server is running.

28

28

Persistence – continued

- **Snapshotting:**
  - By default Redis saves snapshots of the dataset on disk, in a binary file called **dump.rdb.**
  - You can configure Redis to have it save the dataset every **N** seconds if there are at least **M changes** in the dataset,
  - Or you can manually call the **SAVE** or **BGSAVE** commands.
  - Example: `127.0.0.1:6379> SAVE 60 1000`
    - this will make Redis automatically dump the dataset to disk **every 60 seconds** if at least **1000 keys changed**
- **Append-only file:**
  - The append-only file is an alternative, fully-durable strategy for Redis. It became available in version 1.1.
    - You can turn on the AOF in your configuration file using: `appendonly yes`
    - In that case every time Redis receives a command that changes the dataset (e.g. **`SET`**) it will append it to the AOF.
    - When you restart Redis it will re-play the AOF to rebuild the state.

More Info here https://redis.io/topics/persistence

29

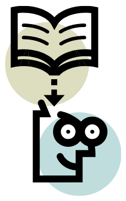
Questions

- What is the purpose of a Namespace in Redis?
- What is the difference between SET and MSET commands in Redis?
- What is the purpose of SELECT command in Redis?
- How to create a key that expires after 300 seconds and hold a string value?

30

Further reading

- Carlson J. Redis in Action. Manning Publications, 2013
- Kreibich J. Redis: The Definitive Guide: Data modeling, caching, and messaging, O'Reilly Media, 2015
- Macedo T., Oliveira F. Redis Cookbook: Practical Techniques for Fast Data Manipulation, O'Reilly Media, 2011
- Redis Documentation https://redis.io/



Essentials

- ✓ Introduced Redis data store
- ✓ Discussed CRUD commands
- ✓ Introduced different data types
- ✓ Learned to work with complex data types
- ✓ Discussed Redis transactions
- ✓ Discussed the use of the expiration