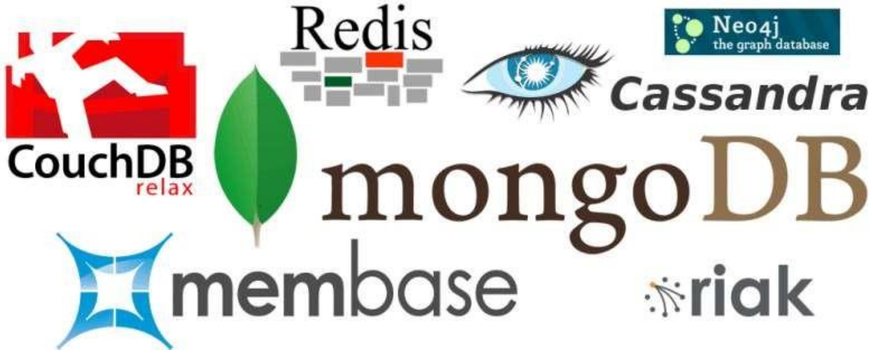


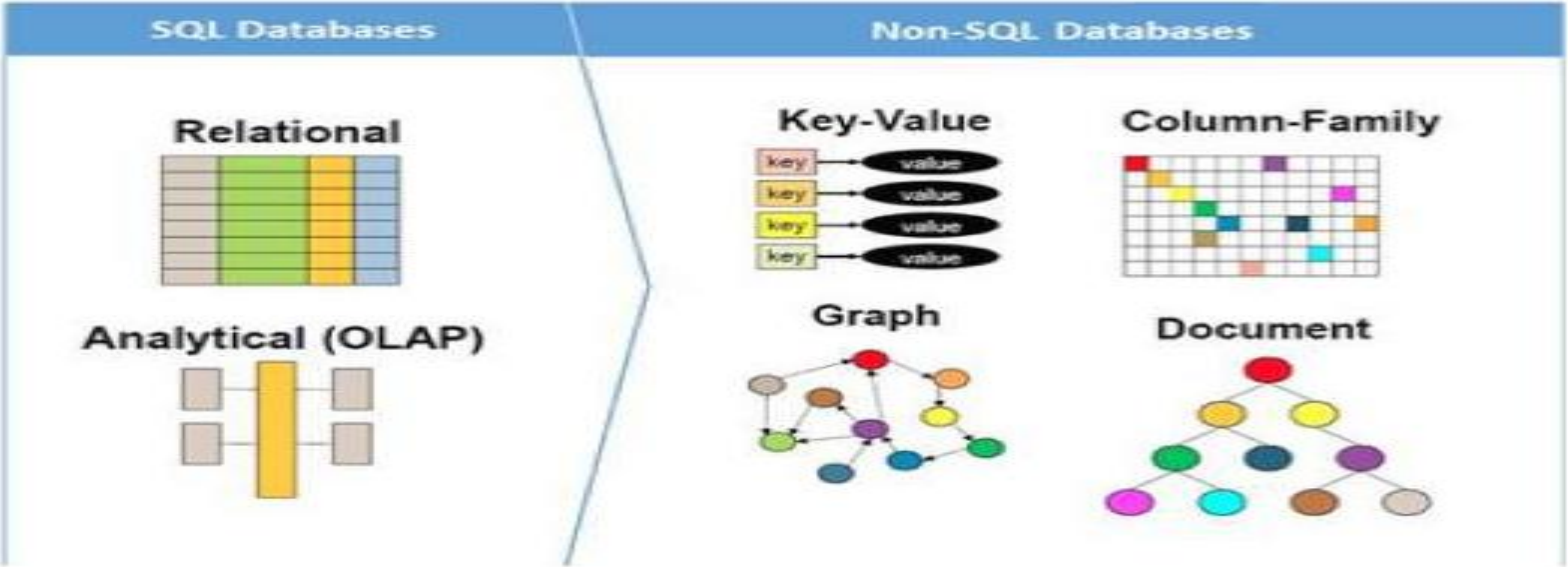
redis and it’s data types

http://redis.io/

Agenda

- What is redis (and what it’s not)
- Popularity of redis, who uses redis, history of redis
- Type of NoSQL databases, NoSQL history
- Different type of data structures supported by redis
  - String
  - List
  - Set
  - Hash
  - Sorted Set / ZSET
- redis CLI commands
- Database
- Expiry
- Transactions
- Mass insert / Bulk upload

noSQL: “Not Only SQL”



What’s Redis? (REmote DIctionary Server)

- Open-source (BSD), in-memory, persist-able key-value advanced datastore
- Key-value means something like

```
CREATE TABLE redis (
    k VARCHAR(512MB) NOT NULL,
    v VARCHAR(512MB),
    PRIMARY KEY (k)
);
```

- 6 data types, 160 commands, blazing fast
- Created in 2009 by @antirez
  (a.k.a Salvatore Sanfilippo)
- Source: https://github.com/antirez/redis
- Website: http://redis.io

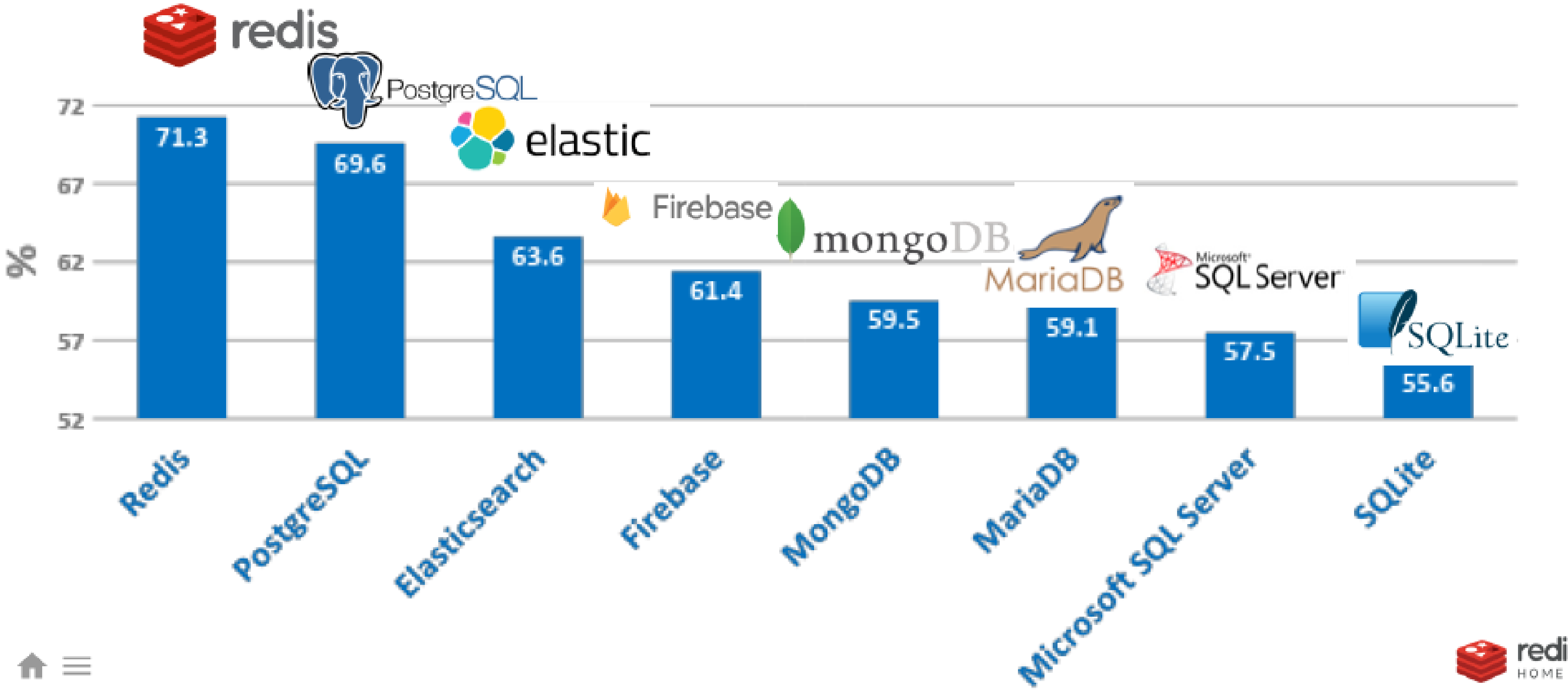
Redis Tops Database Popularity Rankings

G2 CROWD	……..#1 NoSQL in User Satisfaction and Market Presence
stackshare	……..#1 NoSQL among Top 10 Data Stores
DATADOG	……..#1 database on Docker
ClusterHQ DevOps.com	#1 NoSQL database deployed in containers
DB-ENGINES	………#1 in growth among top 3 NoSQL databases
UpScored	………#1 database in skill demand

Most Loved Databases 2017, 2018 & 2019

Stack Overflow survey, among >100K developers

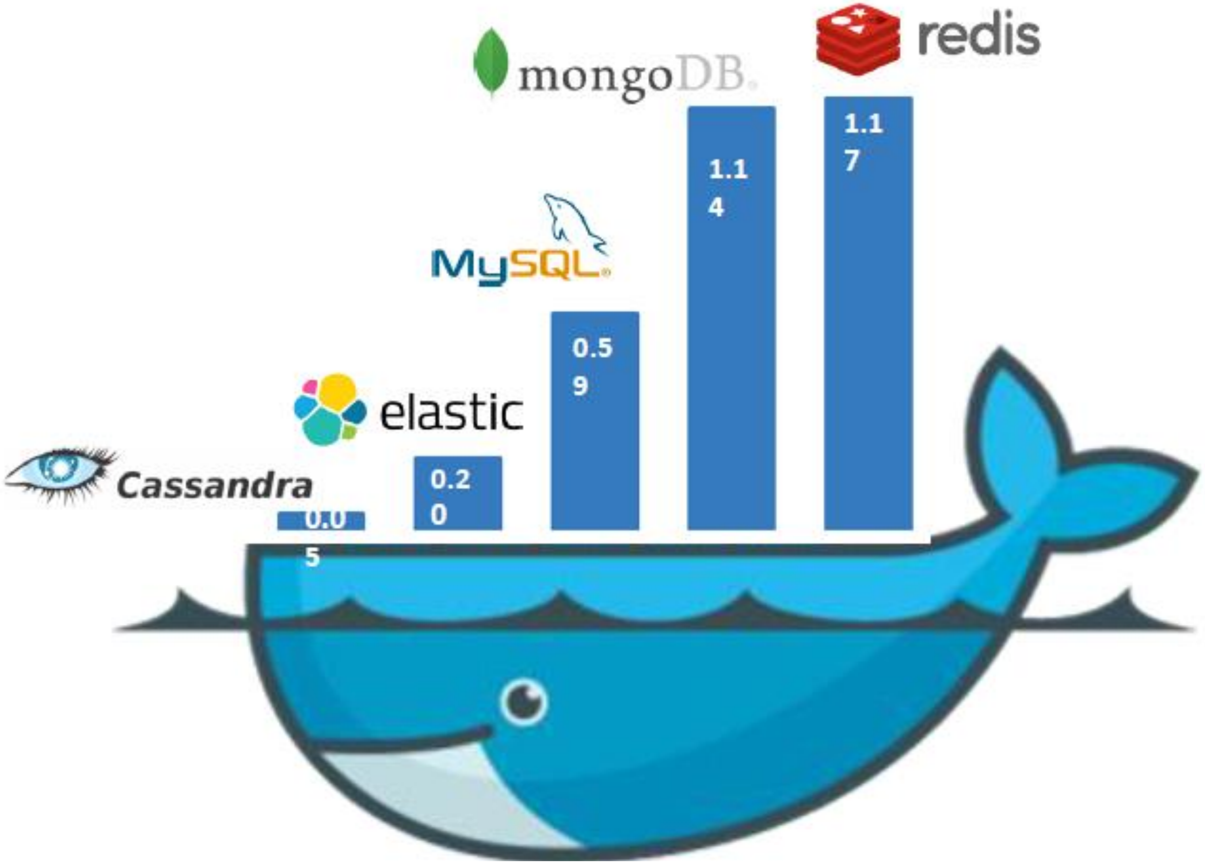
% of devs who expressed interest in continuing to develop with a database



redislabs
HOME OF REDIS

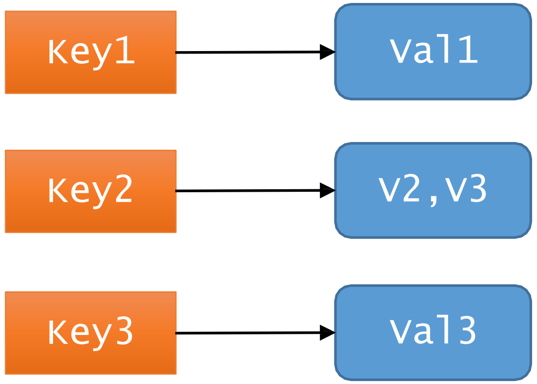
Most Popular Database Container

Number of containers (in Billions) launched at Docker Hub (as of Dec 2018)



What is redis

- Redis is a **Key Value NoSQL database**
- **Open source** (BSD licensed)
- **In memory** data structure store. All data is served from memory
  - Redis mantra - data served from memory, disk used for storage
- Offers **high performance**, replication, and a unique data model
- Supports **five different data structures** - strings, lists, sets, hashes, sorted sets (as value)
- Used as database, **cache** and message broker.
- Actually stands for REmote DIctionary Server
- Redis is often compared to memcached, which is a very high performance, key-value cache server.
- Supports built in replication, Lua scripting, on disk persistence, limited transaction
- Written in ANSI C, supports multiple platform



What redis is not

- redis is not a RDBMS
- Does not support Schema
- Does not support Joins (Stored Procs, Triggers etc)
- Does not support ACID Transactions, though supports limited transactions
- Does not support SQL

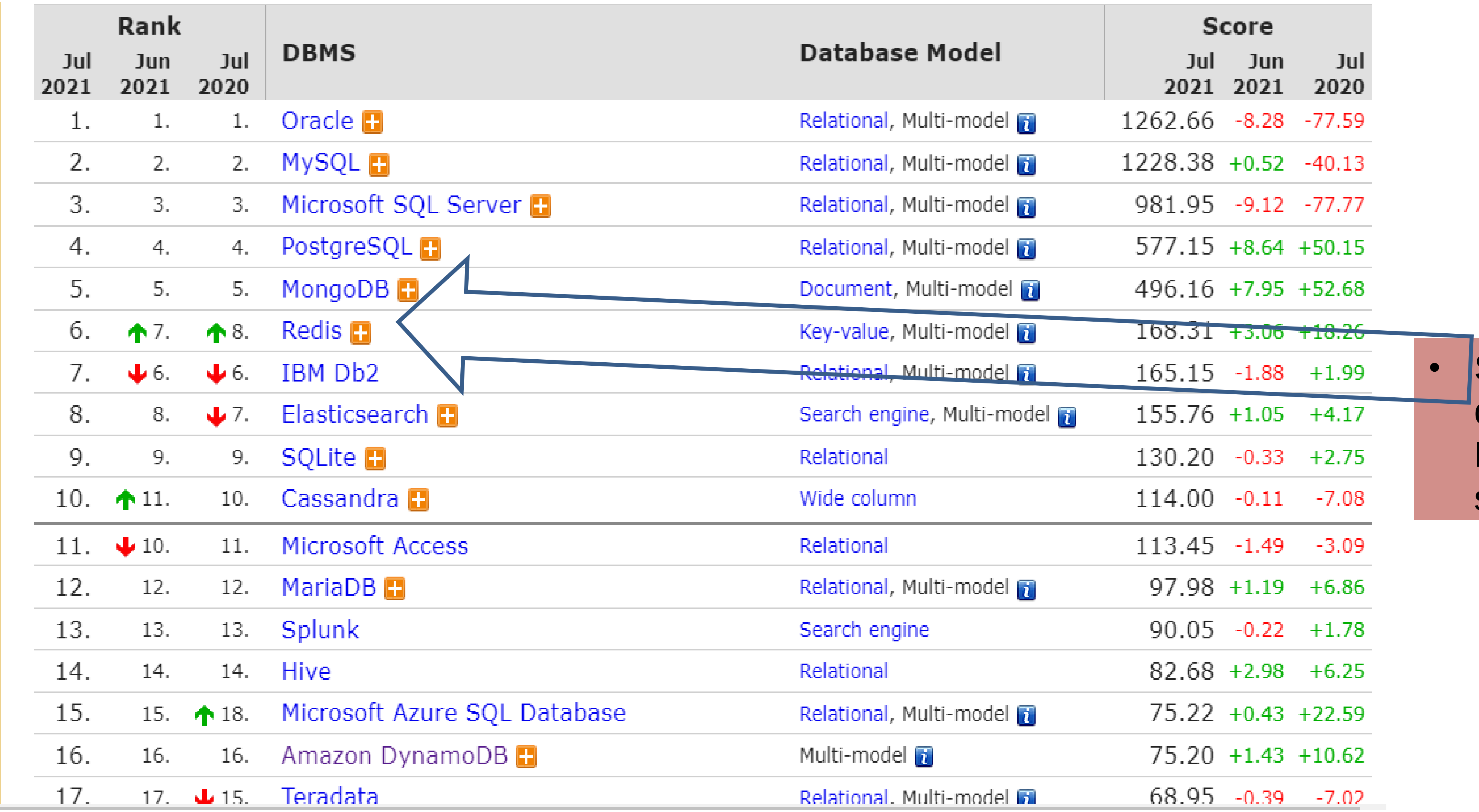
NoSQL History

- 1998 | Carlo Strozzi used the term NoSQL to name his lightweight, open-source relational database that did not expose the standard SQL interface.
- 2000 | Graph database Neo4j started
- 2004 | Google BigTable is started in 2004. Paper published in 2006.
- 2005 | CouchDB development started.
- 2007 | Research paper on Amazon Dynamo released (not AWS DynamoDB)
- 2007 | MongoDB started as a part of a open source cloud computing stack and first standalone release in 2009.
- 2008 | Facebook open sources the Cassandra project
- 2008 | Project Voldemort started
- 2009 | The term NoSQL was reintroduced by Eric Evans of rackspace. Redis initial release
- 2010 | Some NoSQL conferences NoSQL Matters, NoSQL Now!, INOSA

redis History

- Early 2009 - Redis project was started in early 2009 by an Italian developer named Salvatore Sanfilippo. Redis was initially written to improve the performance of LLOOGG, a real-time web analytics product out of Salvatore’s startup.
- June 2009 - Redis was stable, and had enough of a base feature set, to serve production traffic at LLOOGG (retired the MySQL installation)
- Redis rapidly grew in popularity. Salvatore fostered a great community, added features at a very rapid pace, and dealt with bugs.
- March 2010 - VMWare hired Salvatore to work full-time on Redis. (Redis itself remains BSD licensed.) VMWare hired Pieter Noordhuis, a key Redis contributor, to give the project an additional momentum boost.
- December 2012 - VMWare and EMC spins off Pivotal which would focus on Big Data and Cloud. Redis related effort moves to Pivotal.
- June 2015 - Redis Labs started sponsoring the development of Redis

Popularity of redis



Rank			DBMS	Database Model	Score		
Jul 2021	Jun 2021	Jul 2020			Jul 2021	Jun 2021	Jul 2020
1.	1.	1.	Oracle	Relational, Multi-model	1262.66	-8.28	-77.59
2.	2.	2.	MySQL	Relational, Multi-model	1228.38	+0.52	-40.13
3.	3.	3.	Microsoft SQL Server	Relational, Multi-model	981.95	-9.12	-77.77
4.	4.	4.	PostgreSQL	Relational, Multi-model	577.15	+8.64	+50.15
5.	5.	5.	MongoDB	Document, Multi-model	496.16	+7.95	+52.68
6.	↑ 7.	↑ 8.	Redis	Key-value, Multi-model	168.31	+3.06	+18.26
7.	↓ 6.	↓ 6.	IBM Db2	Relational, Multi-model	165.15	-1.88	+1.99
8.	8.	↓ 7.	Elasticsearch	Search engine, Multi-model	155.76	+1.05	+4.17
9.	9.	9.	SQLite	Relational	130.20	-0.33	+2.75
10.	↑ 11.	10.	Cassandra	Wide column	114.00	-0.11	-7.08
11.	↓ 10.	11.	Microsoft Access	Relational	113.45	-1.49	-3.09
12.	12.	12.	MariaDB	Relational, Multi-model	97.98	+1.19	+6.86
13.	13.	13.	Splunk	Search engine	90.05	-0.22	+1.78
14.	14.	14.	Hive	Relational	82.68	+2.98	+6.25
15.	15.	↑ 18.	Microsoft Azure SQL Database	Relational, Multi-model	75.22	+0.43	+22.59
16.	16.	16.	Amazon DynamoDB	Multi-model	75.20	+1.43	+10.62
17.	17.	↓ 15.	Teradata	Relational, Multi-model	68.95	-0.39	-7.02

- Second popular NoSQL database after MongoDB Most popular Key Value store

Source: http://db-engines.com/en/ranking

Who all are using Redis?

twitter
SONY LIV sports
guardian.co.uk
Pinterest
snapchat
flickr
Engine Yard
digg
craigslist
BLIZZARD ENTERTAINMENT
DISQUS
bump TECHNOLOGIES
github SOCIAL CODING
studiVZ
stackoverflow
mercado Libre

Who uses redis

Who's using Redis?

A list of well known companies using Redis:

Twitter GitHub Weibo Pinterest Snapchat Craigslist Digg StackOverflow Flickr

A notable use case for [Redis is at Twitter](), where it is used to cache the latest few hundred tweets for every active user.

React Chat
Redis Admin UI
Flickr
Instagram
Airbnb
Alibaba
Medium
Pintrest
Shopify
Square
Tumblr
Twitter
UserVoice
Vine

Craigslist
Stripe
rdio
Hulu
GitHub
Disqus
BitBucket
Parse
Songtive
StackOverflow
Lanyrd
Grooveshark
MixRadio
Kickstarter

Trello
NewsBlur
Coinbase
Dashlane
DataSift
Imgur
Gluten Freedom
Recommend
AppLovin
N5 Tech
Mozello
Appknox
Natue
Redis Labs

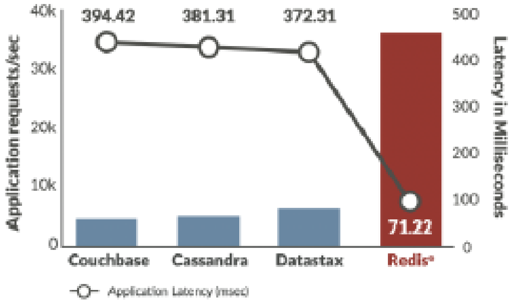
Source - http://techstacks.io/

Redis Top Differentiators

1

Performance

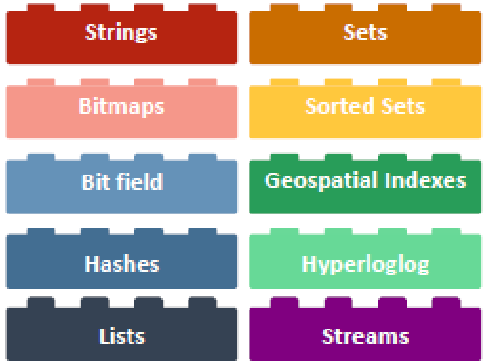
NoSQL Benchmark



2

Simplicity

Redis Data Structures



Strings

Sets

Bitmaps

Sorted Sets

Bit field

Geospatial Indexes

Hashes

Hyperloglog

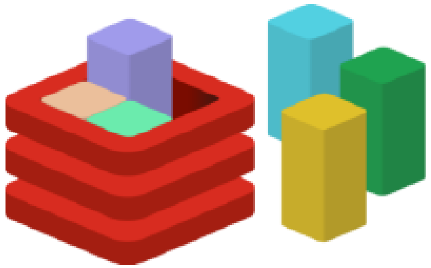
Lists

Streams

3

Extensibility

Redis Modules



Types of NoSQL databases

- **Redis**
- MemcacheDB
- Riak
- Amazon DynamoDB
- Oracle NoSQL database
- Berkley DB
- Azure Table Storage
- Voldermort
- Amazon SimpleDB

Key Value Store / Tuple Store

- MongoDB
- CouchDB
- CouchBase
- RavenDB
- Azure DocumentDB
- IBM Cloudant

Document Store

- BigTable
- Hadoop, HBase
- Hortonworks, Cloudera
- Cassandra
- Cloudata
- Accumolo
- Amazon SimleDB
- *IBM Informix*
- Google Cloud Platform BigTable

Wide Column Store / Column Family

- Neo4J
- OrientDB
- Titan
- HyperGraph DB
- FlockDB (Twitter)

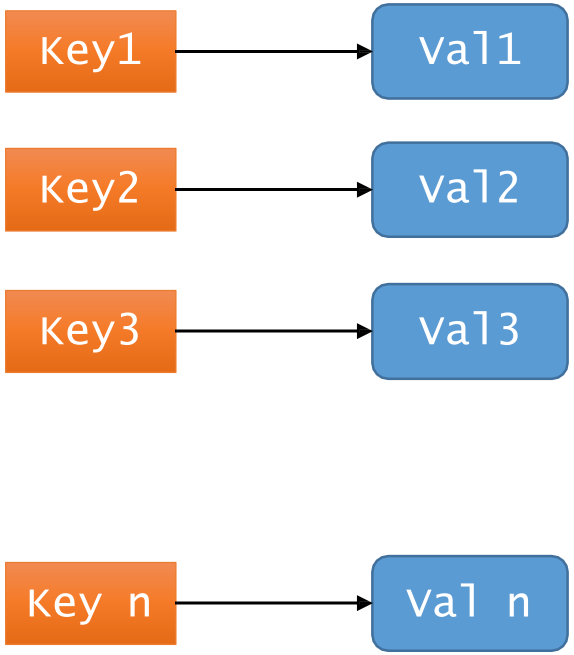
Graph Database

Source - http://nosql-database.org/

NOSQL

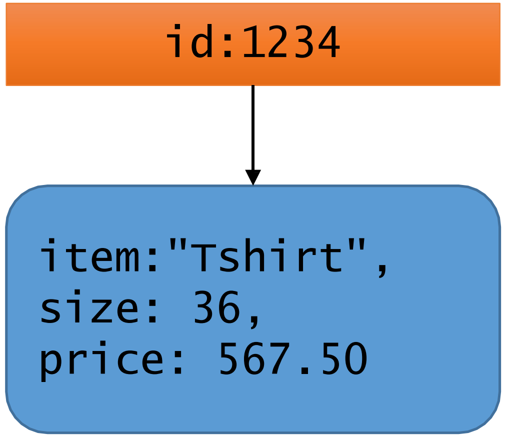
Types of NoSQL databases

Key Value Store / Tuple Store



Document Store

```
{
  id:1234,
  item:"Tshirt",
  size: 36,
  price: 567.50
}
```



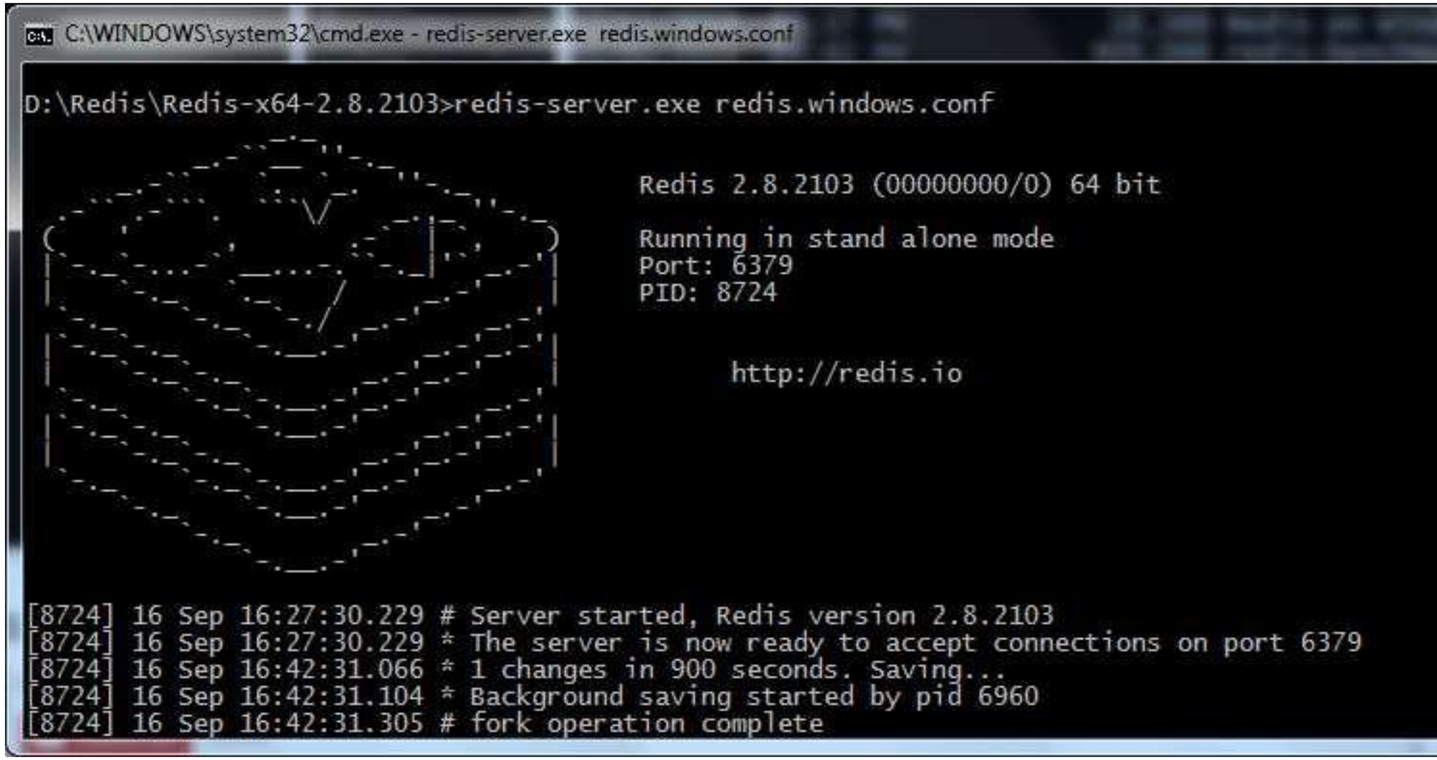
Wide Column Store / Column Family

Graph Database

redis – Server and CLI/Client

```
redis-server.exe redis.windows.conf
```

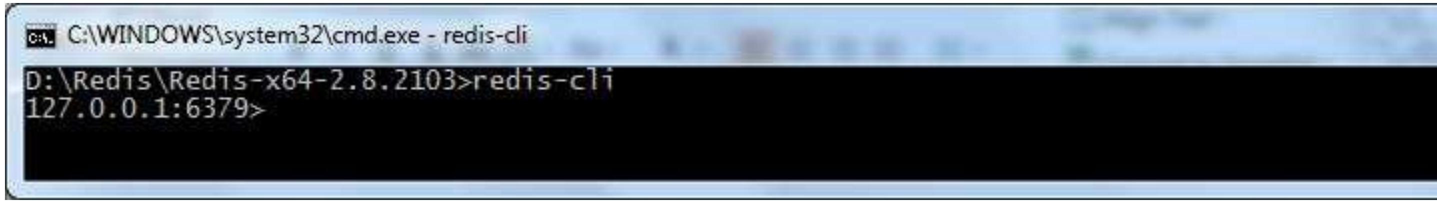
- Runs on default port, 6379



C:\WINDOWS\system32\cmd.exe - redis-server.exe redis.windows.conf

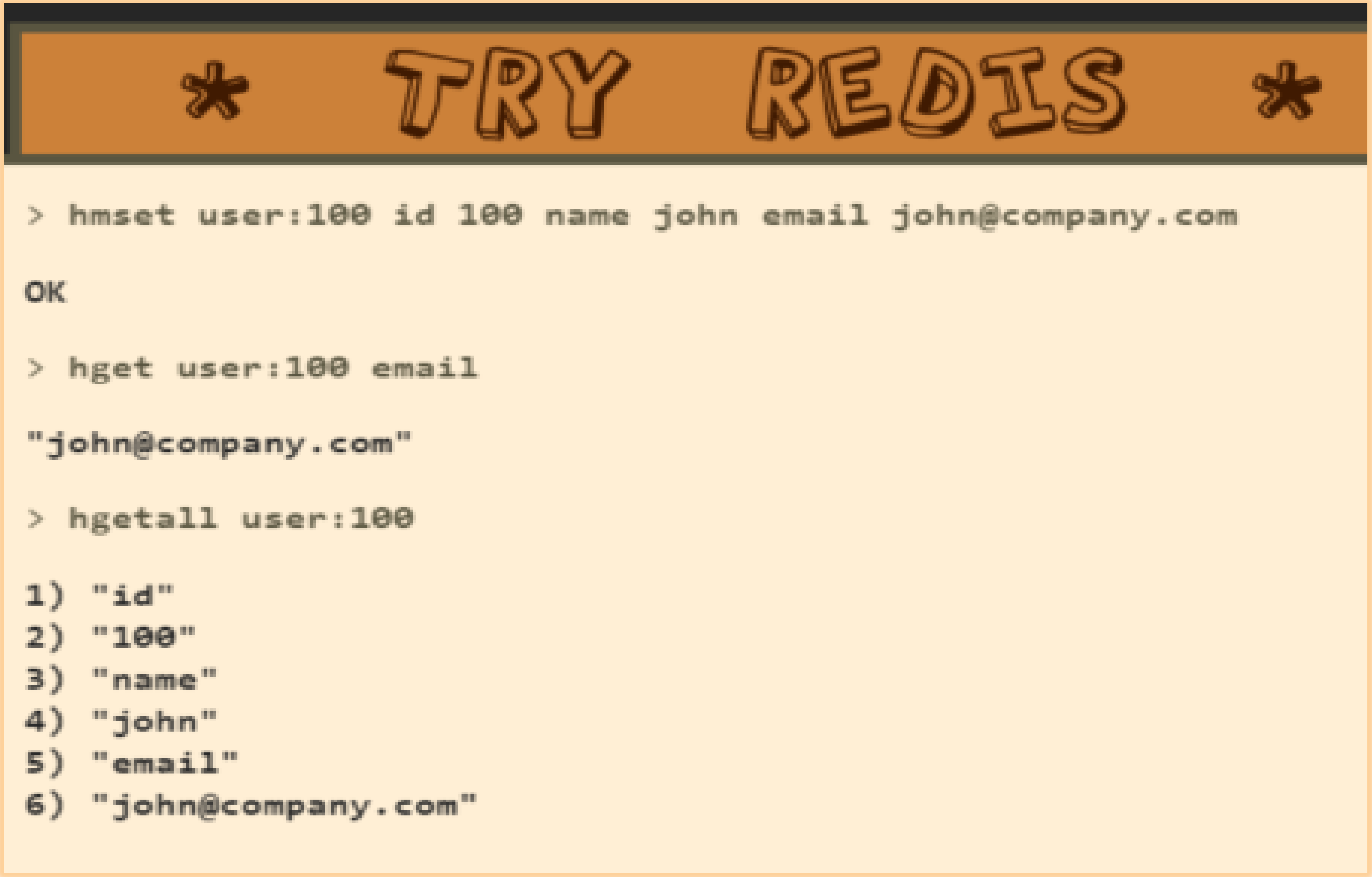
```
D:\Redis\Redis-x64-2.8.2103>redis-server.exe redis.windows.conf

                                    Redis 2.8.2103 (00000000/0) 64 bit

                                    Running in stand alone mode
                                    Port: 6379
                                    PID: 8724

                                          http://redis.io

[8724] 16 Sep 16:27:30.229 # Server started, Redis version 2.8.2103
[8724] 16 Sep 16:27:30.229 * The server is now ready to accept connections on port 6379
[8724] 16 Sep 16:42:31.066 * 1 changes in 900 seconds. Saving...
[8724] 16 Sep 16:42:31.104 * Background saving started by pid 6960
[8724] 16 Sep 16:42:31.305 # fork operation complete
```

```
D:\Redis\Redis-x64-2.8.2103>redis-cli
127.0.0.1:6379>
```



```
D:\Redis\Redis-x64-2.8.2103>redis-cli
127.0.0.1:6379>
```

- Connects to default port, 6379 and awaits for commands



```
> hmset user:100 id 100 name john email john@company.com

OK

> hget user:100 email

"john@company.com"

> hgetall user:100

1) "id"
2) "100"
3) "name"
4) "john"
5) "email"
6) "john@company.com"
```

First set of commands

```
127.0.0.1:6379> ping
PONG

127.0.0.1:6379> echo "hello from redis"
"hello from redis"

127.0.0.1:6379> quit
D:\Redis\Redis-x64-2.8.2103>
```

Logical Data Model

Data Model

- *Key*
  - Printable ASCII

- *Value*
  - Primitives
    - Strings
  - Containers (of strings)
    - Hashes
    - Lists
    - Sets
    - Sorted Sets

Redis data types

Redis Data Type		Contains	Read/write ability
String	“I’m a string!” 0 1 1 0 0 0 0 ...	Binary-safe strings (up to 512 MB), Integers or Floating point values, Bitmaps.	Operate on the whole string, parts, increment/decrement the integers and floats, get/set bits by position.
Hash	Key1 Value1 Key2 Value2	Unordered hash table of keys to string values	Add, fetch, or remove individual ítems by key, fetch the whole hash.
List	A ↔ C ↔ B ↔ C	Doubly linked list of strings	Push or pop items from both ends, trim based on offsets, read individual or multiple items, find or remove items by value.
Set	D B C A	Unordered collection of unique strings	Add, fetch, or remove individual items, check membership, intersect, union, difference, fetch random items.
Sorted Set	B: 0.1 D: 0.3 A: 250 C: 250	Ordered mapping of string members to floating-point scores, ordered by score	Add, fetch, or remove individual items, fetch items based on score ranges or member value.
Geospatial index	Value Lat.: 20.63373 Lon.: -103.55328	Sorted set implementation using geospatial information as the score	Add, fetch or remove individual items, search by coordinates and radius, calculate distance.
HyperLogLog	0 1 1 0 0 0 0 1 0 1 ...	Probabilistic data structure to count unique things using 12Kb of memory	Add individual or multiple items, get the cardinality.

*Redis data types internals: https://cs.brown.edu/courses/cs227/archives/2011/slides/mar07-redis.pdf

redis Data Structures

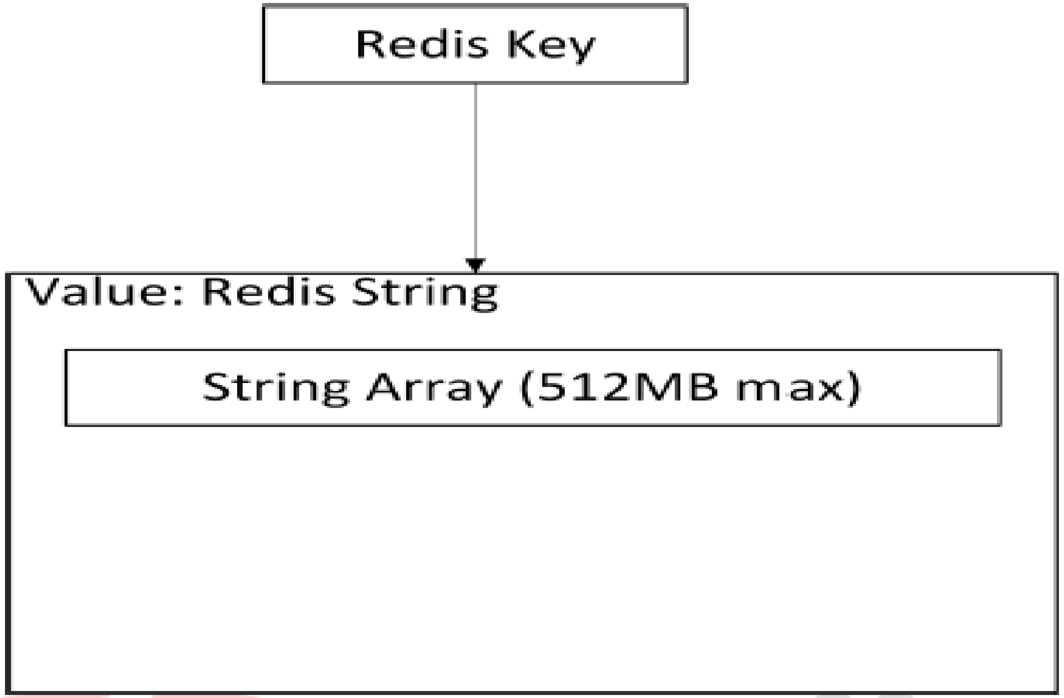
- String
- List
- Set
- Hash
- Sorted Set / ZSET

Logical Data Model

Data Model

- *Key*
  - ○ Printable ASCII

- *Value*
  - ○ Primitives
    - ■ **Strings**
  - ○ Containers (of strings)
    - ■ Hashes
    - ■ Lists
    - ■ Sets
    - ■ Sorted Sets



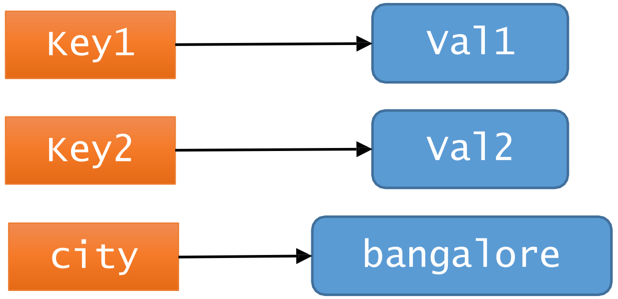
Redis Key

Value: Redis String

String Array (512MB max)

String

- STRINGs are similar to strings that we see in other languages or other key-value stores.
- String values could be added, read, deleted and updated for a key.



Key1 → Val1

Key2 → Val2

city → bangalore

Operations	Details
SET	Sets the value stored at the given key
GET	Fetches the data stored at the given key
DEL	Deletes the value stored at the given key (works for all types)
SETNX	Sets the value of a key, only if the key does not exist
MSET	Sets the value of multiple keys
MGET	Fetches the data of multiple keys
INCR	Increase the integer value of a key by one
INCRBY	Increase the integer value of a key by the given amount
DECR	Decrease the integer value of a key by one
DECRBY	Decrease the integer value of a key by the given amount
STRLEN	Gets the length of the value stored in a key

String

```
127.0.0.1:6379> set city bangalore
OK

127.0.0.1:6379> get city
"bangalore"

127.0.0.1:6379> append city ", India"
(integer) 16

127.0.0.1:6379> get city
"bangalore, India"

127.0.0.1:6379> del city
(integer) 1

127.0.0.1:6379> get city
(nil)
```

```
127.0.0.1:6379> mset key1 val1 key2 val2 key3 val3
OK

127.0.0.1:6379> mget key1 key2 key3
1) "val1"
2) "val2"
3) "val3“

127.0.0.1:6379> setnx city "den haag"
(integer) 1

127.0.0.1:6379> setnx city "blore"
(integer) 0

127.0.0.1:6379> strlen city
(integer) 8
```

String (Integer)

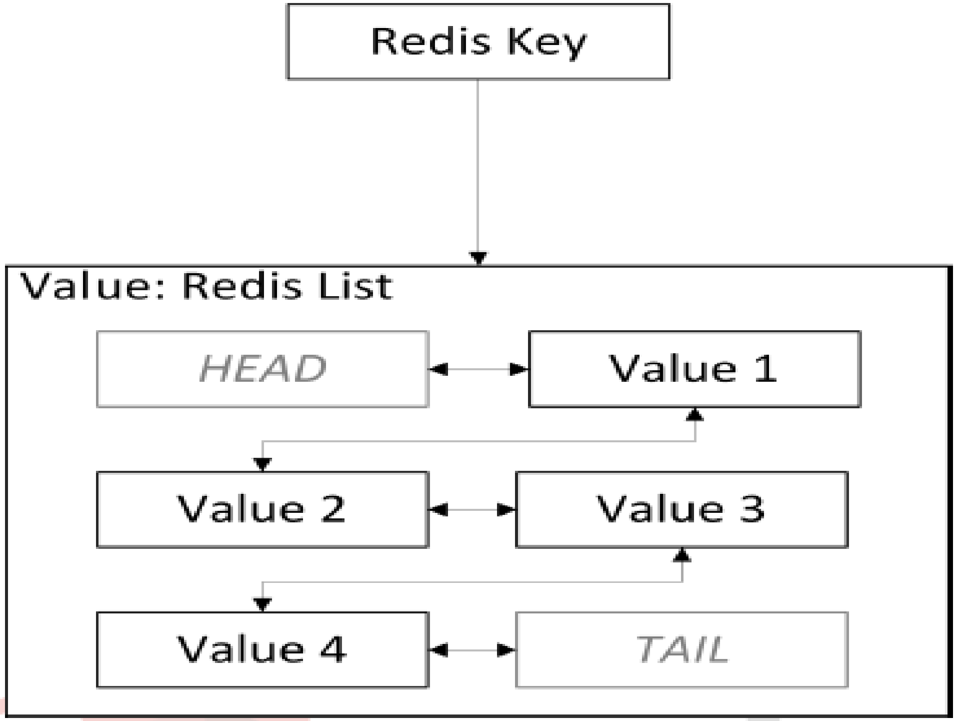
```
127.0.0.1:6379> set counter 100
OK
127.0.0.1:6379> get counter
"100"
127.0.0.1:6379> incr counter
(integer) 101
127.0.0.1:6379> get counter
"101"
127.0.0.1:6379> incrby counter 5
(integer) 106
127.0.0.1:6379> decr counter
(integer) 105
127.0.0.1:6379> decrby counter 10
(integer) 95
```

Logical Data Model

Data Model

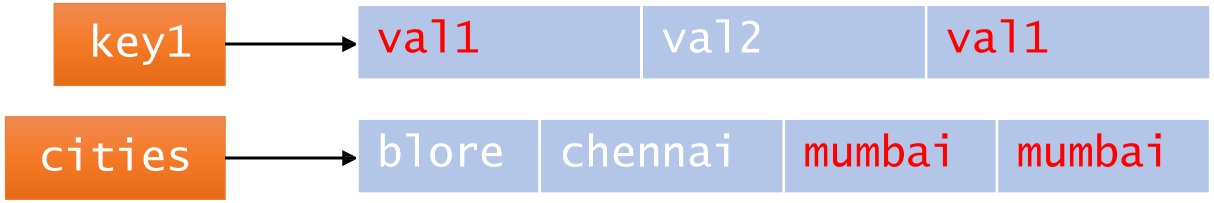
- *Key*
  - Printable ASCII

- *Value*
  - Primitives
    - Strings
  - Containers (of strings)
    - Hashes
    - **Lists**
    - Sets
    - Sorted Sets



List

- Redis Lists are simply sequence of strings
- Ordered by insertion order
- Allows duplicates
- It is possible to add elements to a Redis List pushing new elements on the head (on the left) or on the tail (on the right) of the list.
- Implemented via Linked Lists



Key	Value
cities	blore, chennai, mumbai
roles	admin, general_user, guest
weekends	Saturday, Sunday
promoted_emps	Bill, Satya, Satish, Steve
prime_nos	1,3,5,7,11,13,17,19

Operations	Details
LPUSH	Pushes the value onto the left end (head) of the list
RPUSH	Pushes the value onto the right end (tail) of the list
LPOP	Pops the value from the left end (head) of the list and returns it
RPOP	Pops the value from the right end (tail) of the list and returns it
LRANGE	Fetches a range of values from the list
LINDEX	Fetches an item at a given position in the list

List – command examples

```
127.0.0.1:6379> rpush cities blore
(integer) 1
127.0.0.1:6379> rpush cities chennai
(integer) 2
127.0.0.1:6379> rpush cities mumbai
(integer) 3

127.0.0.1:6379> lrange cities 0 3
1) "blore"
2) "chennai"
3) "mumbai

127.0.0.1:6379> lrange cities 0 -1
1) "blore"
2) "chennai"
3) "mumbai"

127.0.0.1:6379> lindex cities 1
"chennai"
```

```
127.0.0.1:6379> rpush cities mumbai
(integer) 4

127.0.0.1:6379> lrange cities 0 4
1) "blore"
2) "chennai"
3) "mumbai"
4) "mumbai“

127.0.0.1:6379> lpush cities chennai
(integer) 5
127.0.0.1:6379> lrange cities 0 5
1) "chennai"
2) "blore"
3) "chennai"
4) "mumbai"
5) "mumbai"
127.0.0.1:6379>
```

List – command examples

```
127.0.0.1:6379> lrange cities 0 5
1) "chennai"
2) "blore"
3) "chennai"
4) "mumbai"
5) "mumbai"

127.0.0.1:6379> rpop cities
"mumbai"

127.0.0.1:6379> lrange cities 0 5
1) "chennai"
2) "blore"
3) "chennai"
4) "mumbai"
```

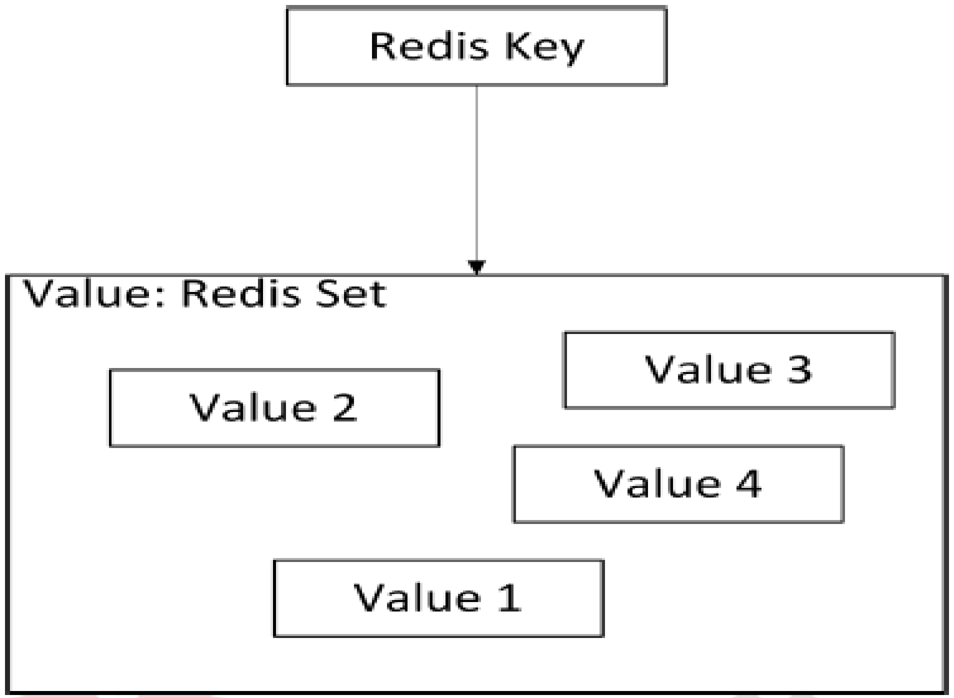
```
127.0.0.1:6379> rpop cities
"mumbai"

127.0.0.1:6379> lrange cities 0 5
1) "chennai"
2) "blore"
3) "chennai"
```

Logical Data Model

Data Model

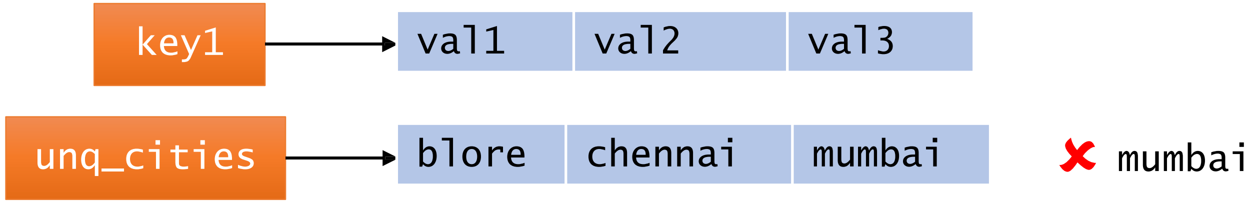
- *Key*
  - Printable ASCII

- *Value*
  - Primitives
    - Strings
  - Containers (of strings)
    - Hashes
    - Lists
    - **Sets**
    - Sorted Sets



Set

- Similar to Lists, Redis Sets are simply sequence of strings
- Does not allow duplicates. Redis SETs use a hash table to keep all strings unique (though there are no associated values).
- Redis SETs are unordered, so we can't push and pop items from the ends like LISTs.



Operations	Details
SADD	Adds the item to the set
SMEMBERS	Returns the entire set of items
SISMEMBER	Checks if an item is in the set
SREM	Removes the item from the set, if it exists

Set – command examples

```
127.0.0.1:6379> sadd ucities chennai kolkata  blore
(integer) 3

127.0.0.1:6379> sadd ucities delhi
(integer) 1

127.0.0.1:6379>  smembers ucities
1) "delhi"
2) "blore"
3) "kolkata"
4) "Chennai"

127.0.0.1:6379> sismember ucities blore
(integer) 1

127.0.0.1:6379> sismember ucities hyderabad
(integer) 0
```

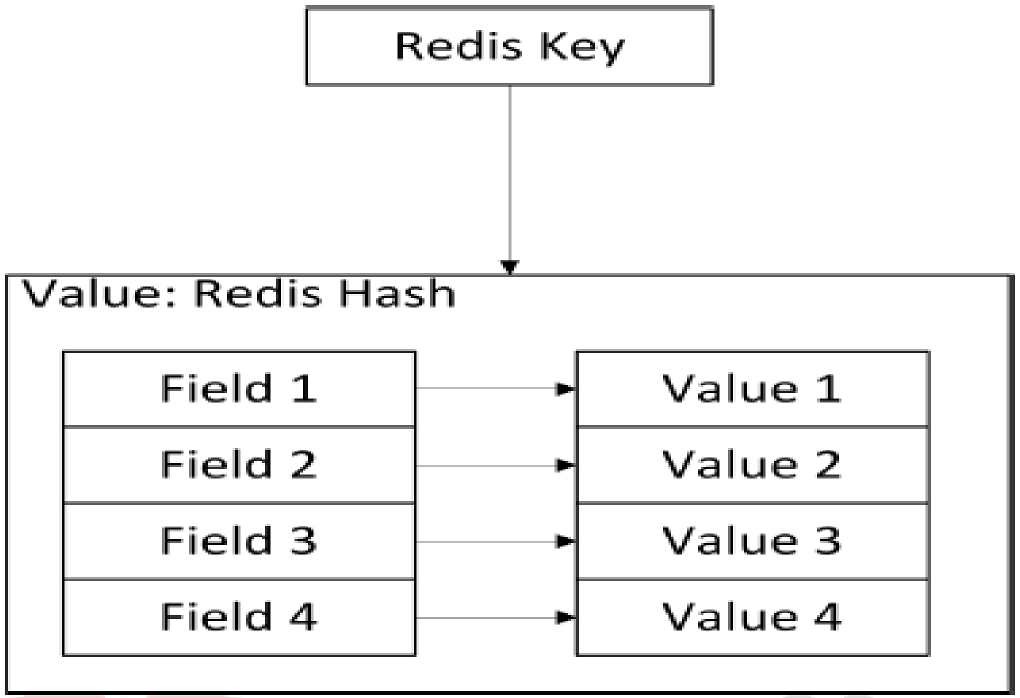
```
127.0.0.1:6379> sadd ucities blore
(integer) 0

127.0.0.1:6379> srem ucities blore
(integer) 1

127.0.0.1:6379> smembers ucities
1) "kolkata"
2) "chennai"
3) "delhi"

127.0.0.1:6379> srem ucities blore
(integer) 0
```

Logical Data Model

Data Model

- *Key*
    - Printable ASCII

- *Value*
    - Primitives
        - Strings
    - Containers (of strings)
        - **Hashes**
        - Lists
        - Sets
        - Sorted Sets



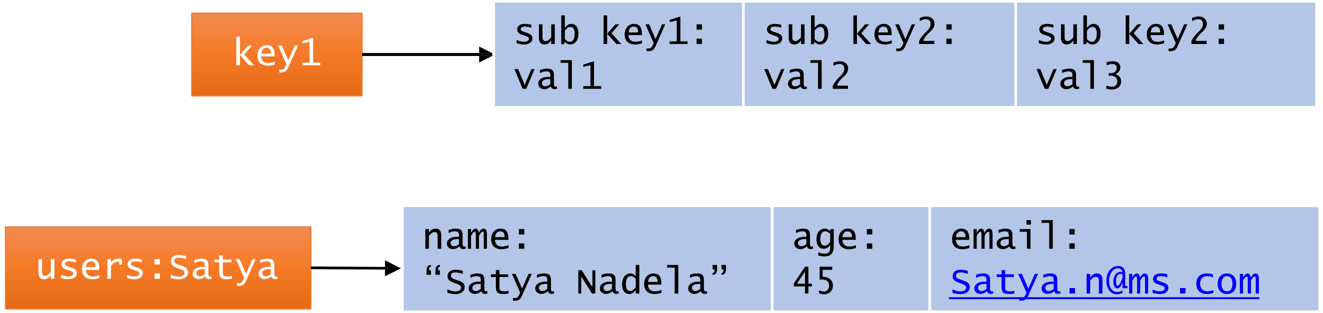
Redis Key

Value: Redis Hash

Field 1	Value 1
Field 2	Value 2
Field 3	Value 3
Field 4	Value 4

Hash

- Whereas LISTs and SETs in Redis hold sequences of items, Redis HASHes store a mapping of keys to values
- Suitable for storing an object



key1 → sub key1: val1 | sub key2: val2 | sub key2: val3

users:Satya → name: “Satya Nadela” | age: 45 | email: Satya.n@ms.com

Operations	Details
HSET	Stores the value at the key in the hash (single key and value)
HGET	Fetches the value at the given hash key
HGETALL	Fetches the entire hash
HDEL	Removes a key from the hash, if it exists
HINCRBY	Increases the value of the sub-key by the specified value
HKEYS	Returns all sub-keys for a specific key
HVALS	Returns all values for a specific key
HMSET	Stores multiple key value pairs in the hash
HLEN	Returns the number of fields in a hash
HEXISTS	Determine if the hash field exists

Hash – command examples

```
127.0.0.1:6379> hset users:Sata name "Satya Nadela"
(integer) 1
127.0.0.1:6379> hset users:Sata age 45
(integer) 1
127.0.0.1:6379> hset users:Sata email satya.m@ms.com
(integer) 1

127.0.0.1:6379> hget users:Sata email
"satya.m@ms.com"

127.0.0.1:6379> hgetall users:Sata
1) "name"
2) "Satya Nadela"
3)"age"
4) "45"
5) "email"
6) "satya.m@ms.com"
```

```
127.0.0.1:6379> hdel users:Sata email
(integer) 1

127.0.0.1:6379> hgetall users:Sata
1) "name"
2) "Satya Nadela"
3)"age"
4) "45"

127.0.0.1:6379> hkeys users:Sata
1) "name"
2) "age"

127.0.0.1:6379> hvals users:Sata
1) "Satya Nadela"
2) "50"
```

Hash – command examples

```
127.0.0.1:6379> hincrby users:Sata age 5
(integer) 50

127.0.0.1:6379> hget users:Sata age
"50“


127.0.0.1:6379> hgetall users:Sata
1) "name"
2) "Satya Nadela"
3) "age"
4) "50"


127.0.0.1:6379> hlen users:Sata
(integer) 2
```

```
127.0.0.1:6379> hexists users:Sata name
(integer) 1

127.0.0.1:6379> hexists users:Sata email
(integer) 0
```

Hash – command examples

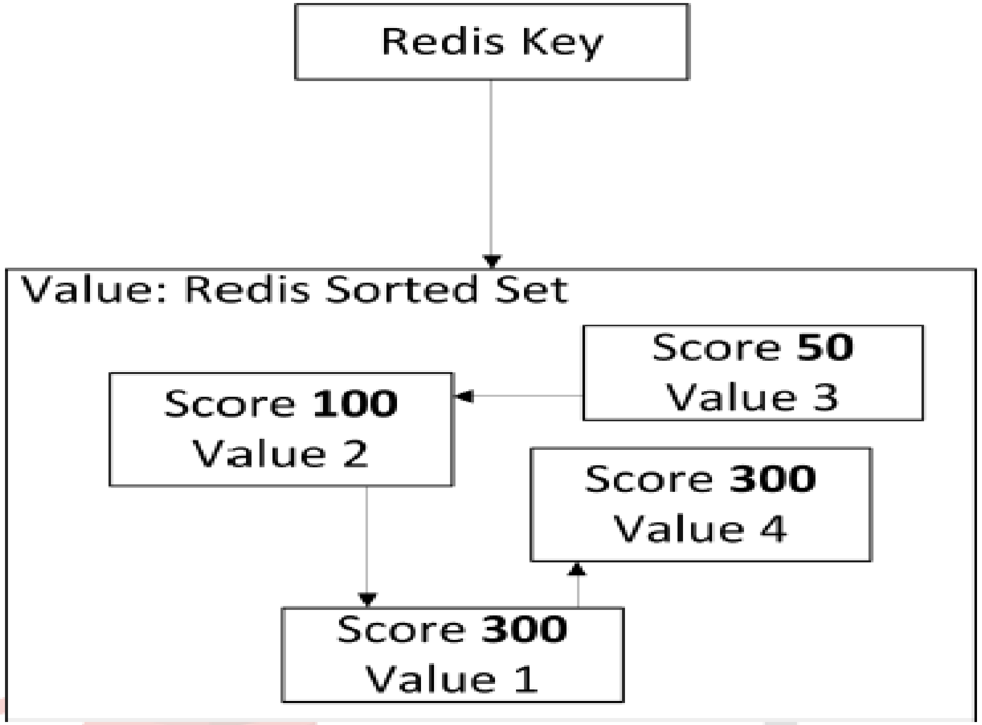
```
127.0.0.1:6379> hmset users:Sunder name "Sunder P"  age 43 designation CEO

OK


127.0.0.1:6379> hmset users:XYZ name "XYZ PQR" age  21 designation Engineer email xyz@gmail.com mobile
  "+(91) 12345 67890"

OK


127.0.0.1:6379> keys users*

1) "users:Sata"

2) "users:XYZ"

3) "users:Sunder"
```

Logical Data Model

Data Model

- *Key*
  - Printable ASCII

- *Value*
  - Primitives
    - Strings
  - Containers (of strings)
    - Hashes
    - Lists
    - Sets
    - **Sorted Sets**



Sorted Set (ZSET)

- Whereas LISTs and SETs in Redis hold sequences of items, Redis HASHes store a mapping of keys to values
- ZSETs offer the ability to store a mapping of members to scores (similar to the keys and values of HASHes). These mappings allow us to manipulate the numeric scores, 2 and fetch and scan over both members and scores based on the sorted order of the scores.

Operations	Details
ZADD	Adds member with the given score to the ZSET
ZRANGE	Fetches the items in the ZSET from their positions in sorted order
ZRANGEBYSCORE	Fetches items in the ZSET based on a range of scores
ZREM	Removes the item from the ZSET, if it exists
ZCARD	Returns the number of members in the ZSET

Sorted Set (ZSET) – command examples

```
127.0.0.1:6379> zadd city_ranking 1 'New York'
(integer) 1
127.0.0.1:6379> zadd city_ranking 2 'Minneapolis'
(integer) 1
127.0.0.1:6379> zadd city_ranking 3 'Bangalore'
(integer) 1
127.0.0.1:6379> zadd city_ranking 4 'Amsterdam'
(integer) 1
127.0.0.1:6379> zadd city_ranking 5 'Warshaw'
(integer) 1

127.0.0.1:6379> zrange city_ranking 0 -1
1) "New York"
2) "Minneapolis"
3) "Bangalore"
4) "Amsterdam"
5) "Warshaw"
```

```
127.0.0.1:6379> zrangebyscore city_ranking 2 4
1) "Minneapolis"
2) "Bangalore"
3) "Amsterdam"

127.0.0.1:6379> zrem city_ranking bangalore
(integer) 0
127.0.0.1:6379> zrem city_ranking Bangalore
(integer) 1

127.0.0.1:6379> zrange city_ranking 0 -1
1) "New York"
2) "Minneapolis"
3) "Amsterdam"
4) "Warshaw“

127.0.0.1:6379> zcard city_ranking
(integer) 4
```

Other Commands

```
127.0.0.1:6379> keys *
1) "user:ani"
2) "users:Sata"
3) "ctr"
4) "ucities"
5) "friend"
6) "counter"
7) "cities“

127.0.0.1:6379> keys user*
1) "user:ani"
2) "users:Sata"
```

Server related commands

Operations	Details
INFO	Gets detailed information and statistics about the redis server like redis version, OS name, Process Id, port, how long the server has been up, config file name, information about connected clients, memory used by redis, persistence information, replication data, etc.
DBSIZE	Returns the no of keys in the database
CLIENT LIST	Gets the list of client connections
CLIENT SETNAME	Sets the current connection name
CLIENT GETNAME	Gets the current connection name
CLIENT KILL	Kills the connection of a client
CONFIG GET	Gets the value of a configuration parameter
CONFIG SET	Sets the value of a configuration parameter
FLUSHDB	Removes all keys from the current database
FLUSHALL	Removes all keys from all databases

Server related commands

```
127.0.0.1:6379> info
#  Server
redis_version:2.8.2103
redis_git_sha1:00000000
redis_git_dirty:0
redis_build_id:2a4e86cac6830caf
redis_mode:standalone
os:Windows
arch_bits:64
multiplexing_api:winsock_IOCP
process_id:8724
run_id:5bbf82e6f0382362bfa0bf378cb87fa7109e6269
tcp_port:6379
uptime_in_seconds:153348
uptime_in_days:1
hz:10
lru_clock:16491038
config_file:D:\Redis\Redis-x64-2.8.2103\redis.windows.conf
```

```
127.0.0.1:6379> dbsize
(integer) 9
```

Database in redis

- Different types of data, typically for different application or even modules could be stored in different redis Database.
- In Redis, databases are identified by an integer index, not by a database name.
- By default, a client is connected to database 0
- To connect and switch to a different database use SELECT (such as SELECT 3) – all subsequent commands will then use database 3, until you issue another SELECT.
- Each Redis database has its own keyspace.
- The number of databases which is available can be configured in redis.conf — by default, it is set to 16. Simply set it to a higher number if you need more

```
127.0.0.1:6379> set city bangalore
OK
127.0.0.1:6379> use 1
(error) ERR unknown command 'use'
127.0.0.1:6379> select 1
OK
127.0.0.1:6379[1]> keys *
(empty list or set)
127.0.0.1:6379[1]> set city kolkata
OK
127.0.0.1:6379[1]> get city
"kolkata"
127.0.0.1:6379[1]> select 0
OK
127.0.0.1:6379> get city
"bangalore"
127.0.0.1:6379> dbsize
(integer) 14
```

Expiry

```
127.0.0.1:6379> set emsg Hello
OK
127.0.0.1:6379> expire emsg 30
(integer) 1
127.0.0.1:6379> ttl emsg
(integer) 26
127.0.0.1:6379> get emsg
"Hello"
127.0.0.1:6379> ttl emsg
(integer) 16
127.0.0.1:6379> get emsg
"Hello"
127.0.0.1:6379> ttl emsg
(integer) 8
127.0.0.1:6379> get emsg
"Hello"
127.0.0.1:6379> ttl emsg
(integer) -2
127.0.0.1:6379> get emsg
(nil)
```

- Set a timeout on key. After the timeout has expired, the key will automatically be deleted.
- A key with an associated timeout is often said to be volatile in Redis terminology.

Operations	Details
EXPIRE	Sets a key’s time to live or timeout. After the timeout expires, the key gets deleted
TTL	Returns the time to live (TTL) for a key

Transaction

- Set a timeout on key. After the timeout has expired, the key will automatically be deleted.
- A key with an associated timeout is often said to be volatile in Redis terminology.

Operations	Details
MULTI	Marks the start of a transaction block. Similar to Begin Transaction in SQL. All commands issued after MULTI are not executed and are queued up.
EXEC	Executes all commands issued after MULTI
DISCARD	Discards all commands issued after MULTI
WATCH	Watch the given keys to determine the execution of MULTI/EXEC block.

Transaction

```
127.0.0.1:6379> set debit_account 120000
OK
127.0.0.1:6379> set credit_account 40000
OK

127.0.0.1:6379> multi
OK
127.0.0.1:6379> decrby debit_account 30000
QUEUED
127.0.0.1:6379> incrby credit_account 30000
QUEUED
127.0.0.1:6379> exec
1) (integer) 90000
2) (integer) 70000
127.0.0.1:6379>
```

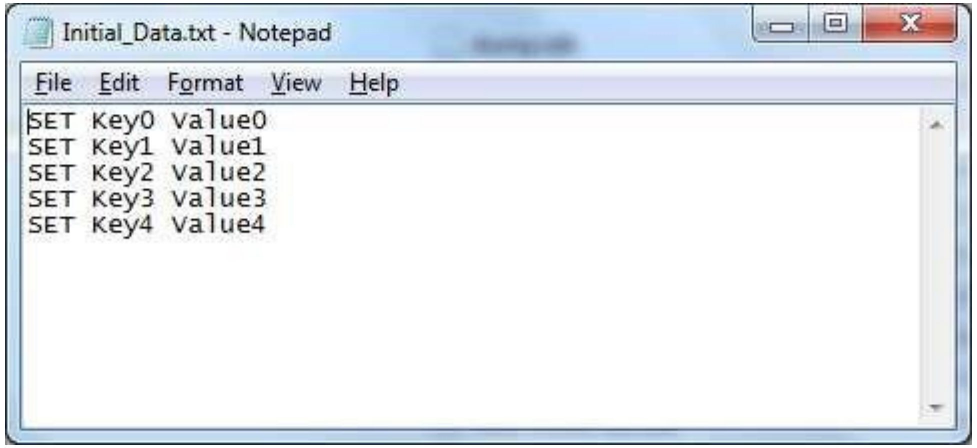
```
127.0.0.1:6379> set debit_account 120000
OK
127.0.0.1:6379> set credit_account 40000
OK

127.0.0.1:6379> multi
OK
127.0.0.1:6379> decrby debit_account 30000
QUEUED
127.0.0.1:6379> incrby credit_account 30000
QUEUED
127.0.0.1:6379> discard
OK

127.0.0.1:6379> get debit_account
"120000"
127.0.0.1:6379> get credit_account
"40000"
```

Mass insertion of data (bulk upload)

- Sometimes Redis instances needs to be loaded with big amount of preexisting or user generated data in short amount of time, so that millions of keys will be created quickly - called mass insertion
- In 2.6 or later versions of Redis the redis-cli utility supports a new mode called pipe mode that was designed in order to perform mass insertion.

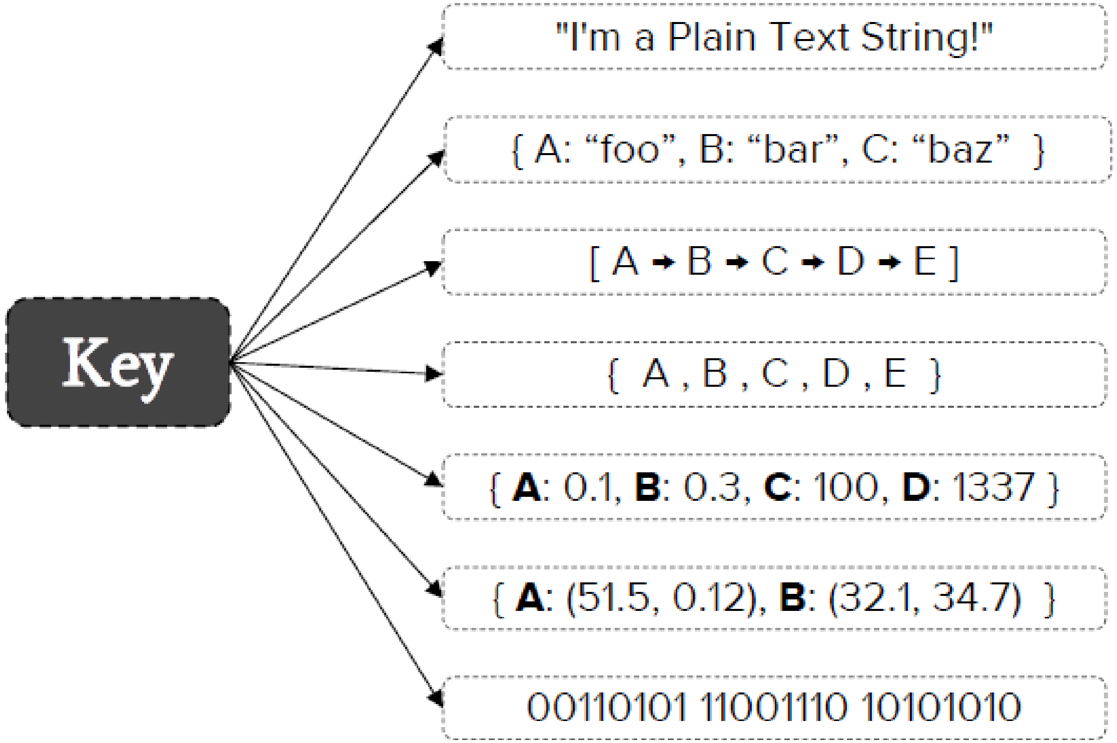
Using the pipe mode the command to run looks like the following:

```
D:\Redis\Redis-x64-2.8.2103>type initial_data.txt | redis-cli  --pipe
All data transferred. Waiting for the last reply...
Last reply received from server.
errors: 0, replies: 5

D:\Redis\Redis-x64-2.8.2103>redis-cli
127.0.0.1:6379> keys Key*
1) "Key0"
2) "Key4"
3) "Key3"
4) "Key2"
5) "Key1"
```



```
SET Key0 Value0
SET Key1 Value1
SET Key2 Value2
SET Key3 Value3
SET Key4 Value4
```

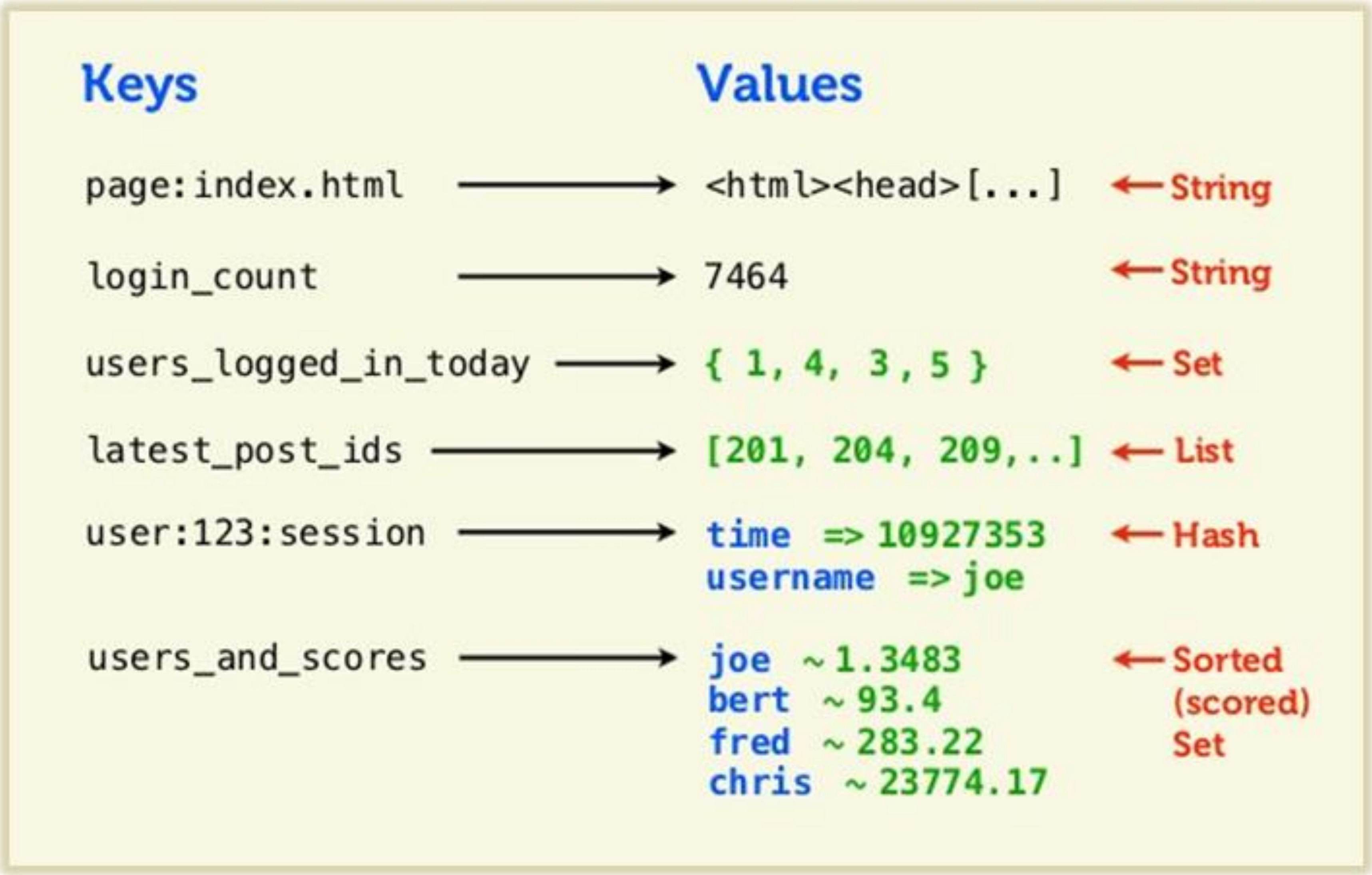
A Quick Recap of Redis



Key

"I'm a Plain Text String!"	Strings / Bitmaps / BitFields
{ A: “foo”, B: “bar”, C: “baz” }	Hash Tables (objects!)
[A → B → C → D → E]	Linked Lists
{ A , B , C , D , E }	Sets
{ **A**: 0.1, **B**: 0.3, **C**: 100, **D**: 1337 }	Sorted Sets
{ **A**: (51.5, 0.12), **B**: (32.1, 34.7) }	Geo Sets
00110101 11001110 10101010	HyperLogLog

Redis data types - Examples



Keys	Values	
page:index.html	<html><head>[...]	← String
login_count	7464	← String
users_logged_in_today	{ 1, 4, 3 , 5 }	← Set
latest_post_ids	[201, 204, 209,..]	← List

user:123:session	time => 10927353
	username => joe
		← Hash

users_and_scores	joe ~ 1.3483 bert ~ 93.4 fred ~ 283.22 chris ~ 23774.17	← Sorted (scored) Set

Shopping Cart Example

Relational Model

carts

CartID	User
1	james
2	chris
3	james

cart_lines

Cart	Product	Qty
1	28	1
1	372	2
2	15	1
2	160	5
2	201	7

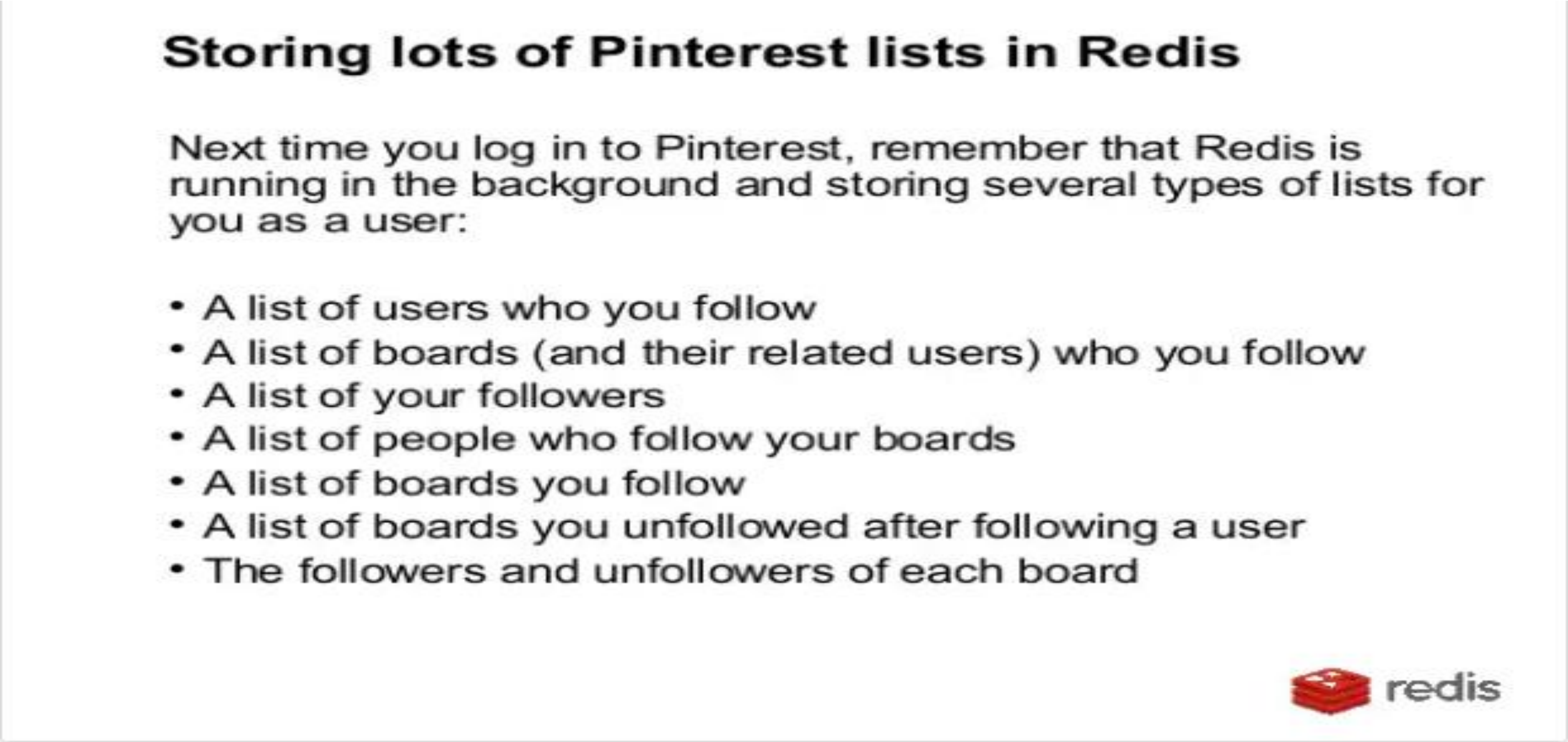
```
UPDATE cart_lines
SET    Qty = Qty + 2
WHERE  Cart=1 AND Product=28
```

Redis Model

```
set  carts_james ( 1 3 )
set  carts_chris ( 2 )
hash cart_1 {
    user        : "james"
    product_28 : 1
    product_372: 2
}
hash cart_2 {
    user        : "chris"
    product_15 : 1
    product_160: 5
    product_201: 7
}
```

```
HINCRBY cart_1 product_28 2
```

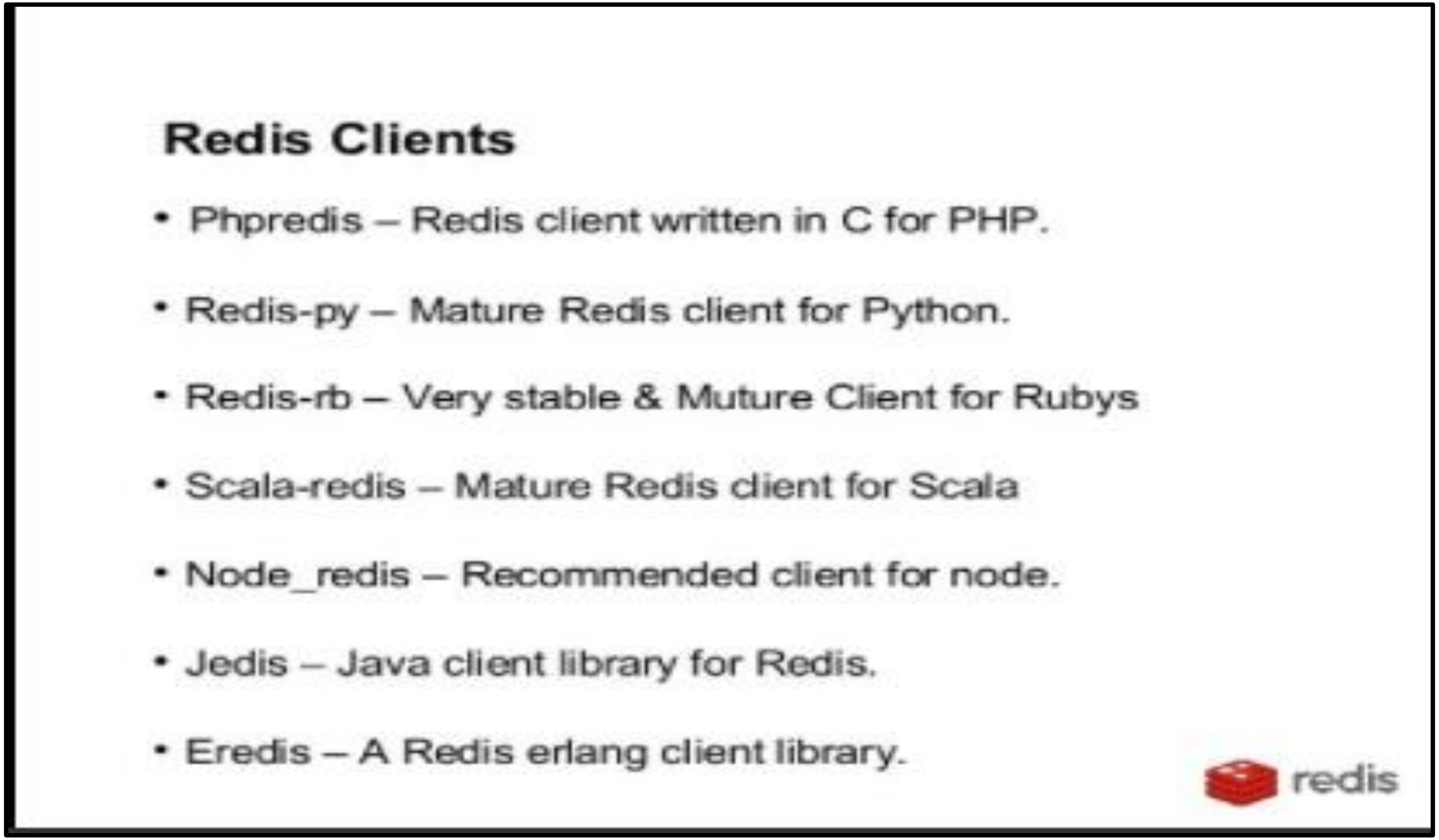
Redis Use Cases



Storing lots of Pinterest lists in Redis

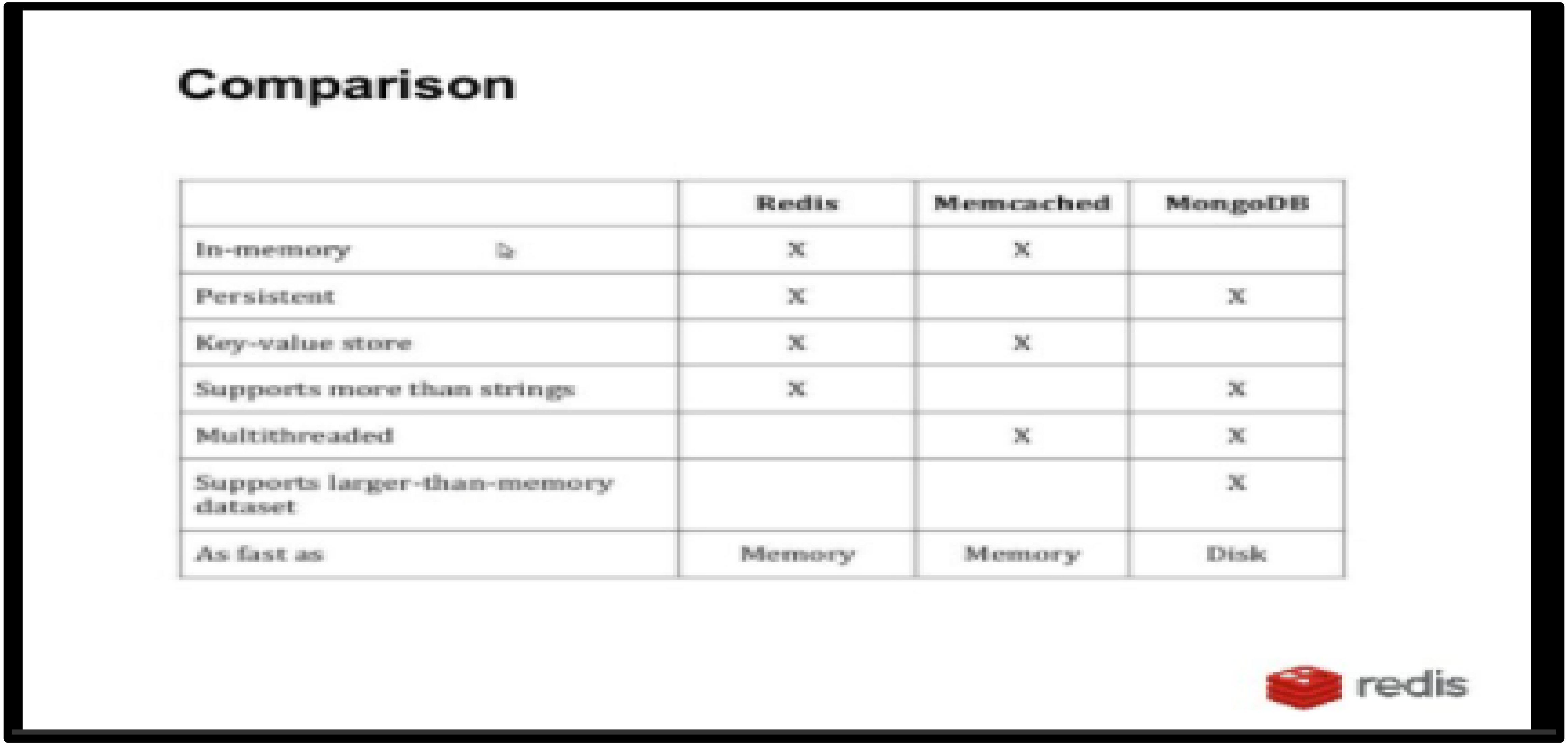
Next time you log in to Pinterest, remember that Redis is running in the background and storing several types of lists for you as a user:

- A list of users who you follow
- A list of boards (and their related users) who you follow
- A list of your followers
- A list of people who follow your boards
- A list of boards you follow
- A list of boards you unfollowed after following a user
- The followers and unfollowers of each board



Redis Clients

- Phpredis – Redis client written in C for PHP.
- Redis-py – Mature Redis client for Python.
- Redis-rb – Very stable & Muture Client for Rubys
- Scala-redis – Mature Redis client for Scala
- Node_redis – Recommended client for node.
- Jedis – Java client library for Redis.
- Eredis – A Redis erlang client library.



Comparison

	Redis	Memcached	MongoDB
In-memory	X	X	
Persistent	X		X
Key-value store	X	X	
Supports more than strings	X		X
Multithreaded		X	X
Supports larger-than-memory dataset			X
As fast as	Memory	Memory	Disk

atomicity

single threaded, so no concurrency problems

transactions and lua scripts to run multiple operations atomically

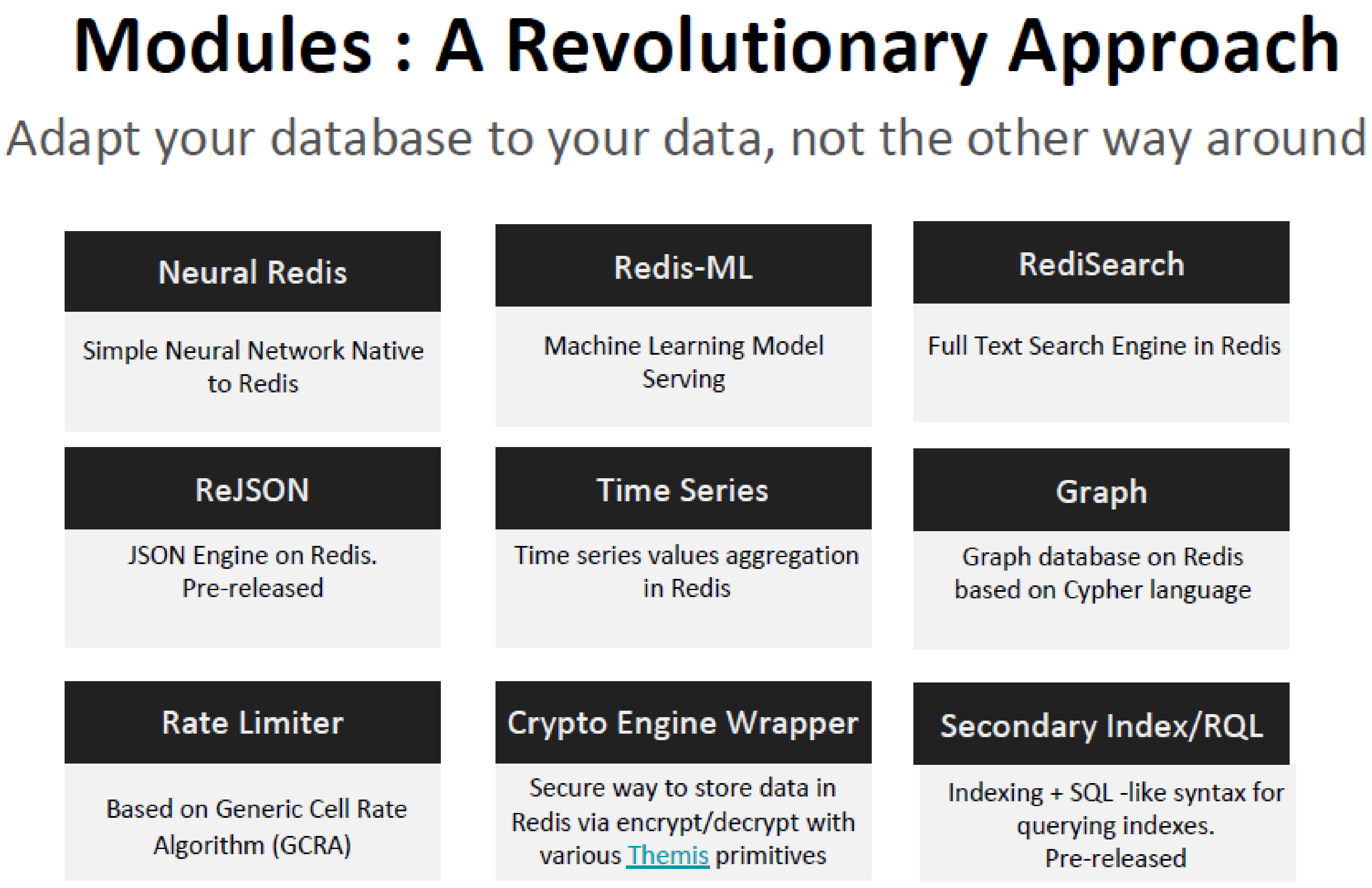
Redis keeps everything in memory all the time

Does that mean if the server goes down I will lose my data?

NO*

*unless you didn't configure it properly

Redis in ML



Modules : A Revolutionary Approach

Adapt your database to your data, not the other way around

Neural Redis

Simple Neural Network Native to Redis

Redis-ML

Machine Learning Model Serving

RediSearch

Full Text Search Engine in Redis

ReJSON

JSON Engine on Redis. Pre-released

Time Series

Time series values aggregation in Redis

Graph

Graph database on Redis based on Cypher language

Rate Limiter

Based on Generic Cell Rate Algorithm (GCRA)

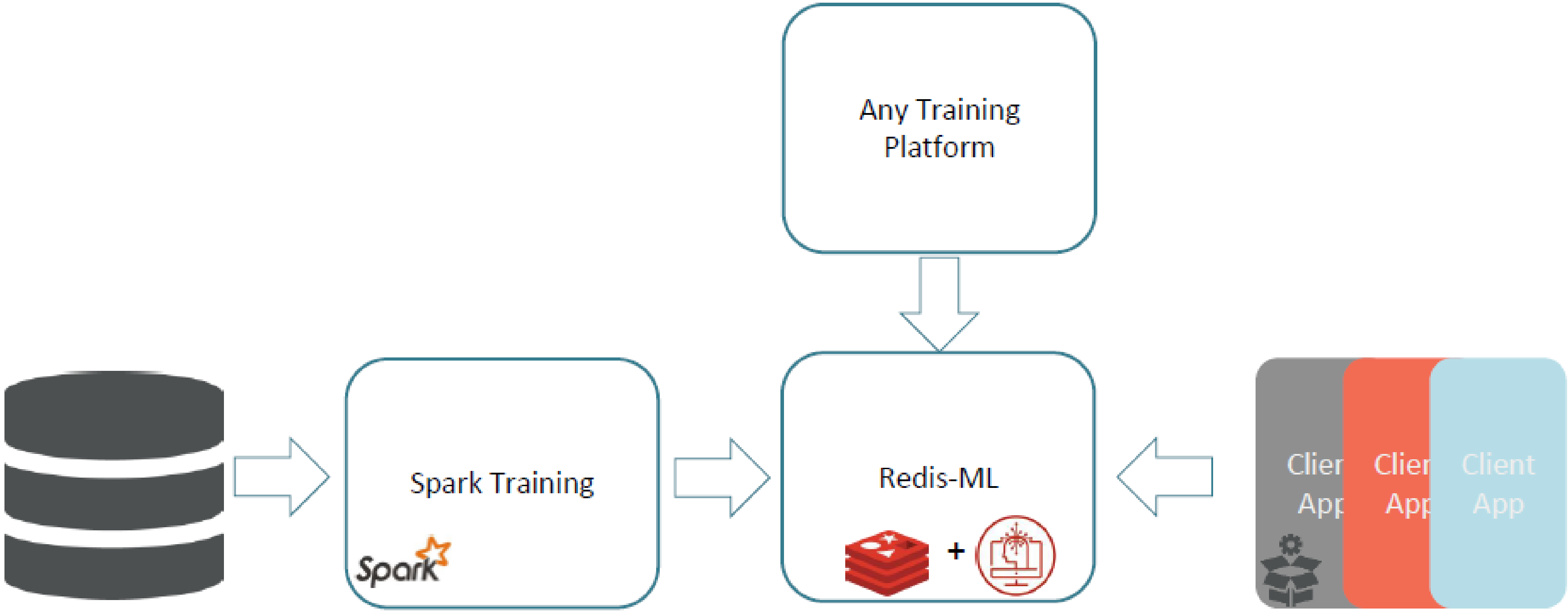
Crypto Engine Wrapper

Secure way to store data in Redis via encrypt/decrypt with various Themis primitives

Secondary Index/RQL

Indexing + SQL -like syntax for querying indexes. Pre-released

A Simpler Machine Learning Lifecycle



Data loaded into Spark

Model is saved in Redis-ML

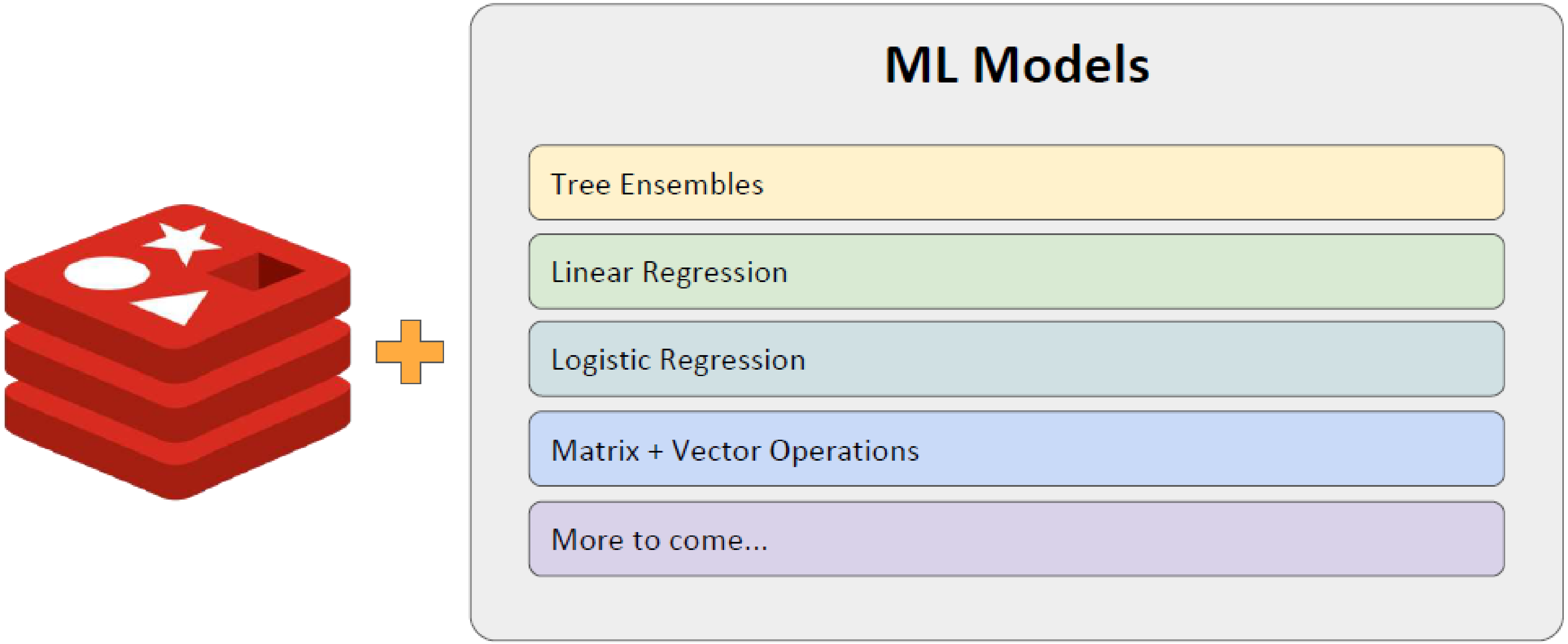
Serving Client

redisconf17

Redis-ML – ML Serving Engine

- Store training output as “hot model”
- Perform evaluation directly in Redis
- Easily integrate existing C/C++ ML libs
- Can be tuned on-the-fly
- Enjoy the performance, scalability and HA of Redis

Redis-ML



ML Models

- Tree Ensembles
- Linear Regression
- Logistic Regression
- Matrix + Vector Operations
- More to come...