AT82.02

DATA MODELING AND MANAGEMENT

LAB05: KEY-VALUE (REDIS)

Outline

- Scenario: TikTock E-Commerce
- Introduction to Key-Value Model
- Redis: Getting started
- Redis Practice

Project: RDB to NoSQL migration

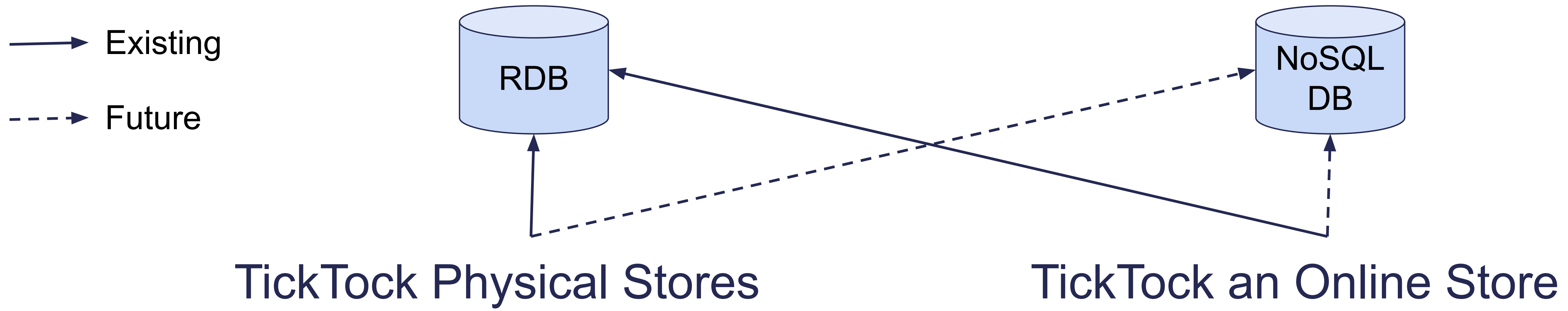
Scenario :

TickTock is an office supply company which has several store locations throughout the world and an online store website with relational database.

With flexibility of schema, performance and scalability features, TickTock is moving to NOSQL database.

You, as a db admin should design and implement query for core functions.

Project: RDB to NoSQL migration



Require query statement for these core functions

- **Support e-commerce functions for customers:** (U1) Search product, (U2) Add product(s) to cart, and (U3) Checkout
  Note that: The payment process is excluded to simplify the system.

- **Support daily operation for store staff:** (U4) Add new products, (U5) Update product information, (U6) Delete duplicate product, (U9) Search insight customer behavior

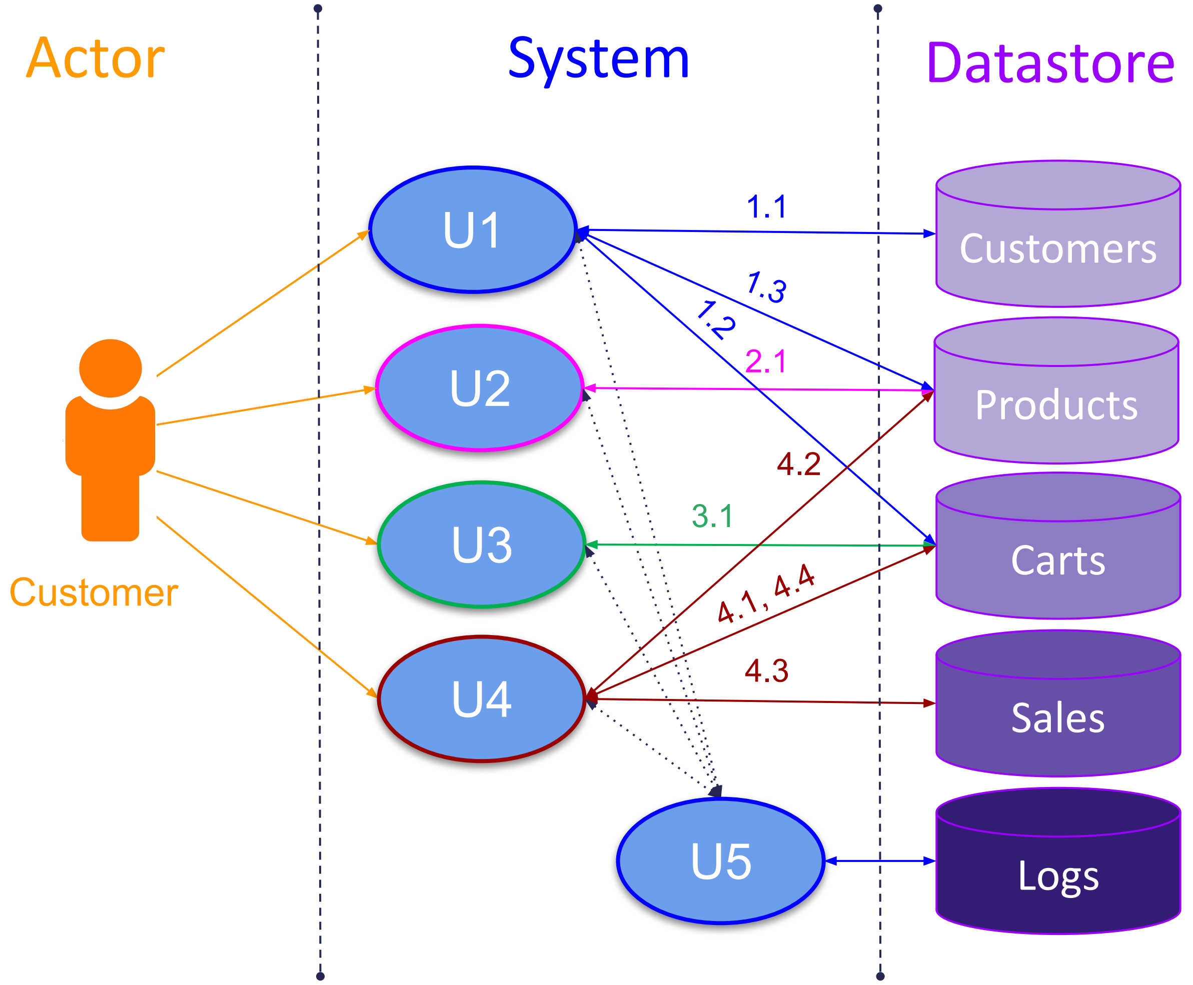
- **Support summary report for owner:** (U7) View popular product report by month, (U8) Total sales, count , average sales

Existing Data in RDB

Data	Description	Fields
Product data	Information about products	product id, name, description ,price, quantity in stock, product keywords
Customer data	Information about customers	email as username as custid,customer name, address, birthdate, password, bank account
Cart data	Information about cart and tentative purchased products	Cartid, product id, quantity, price
Sales data	Information about order and purchased products	the item(s) purchased, information on the customer who made the purchase, several other details regarding the sale.
Log data	Information about customers' activities	Action datetime, Action type, Action description, customer Action type = {''}1

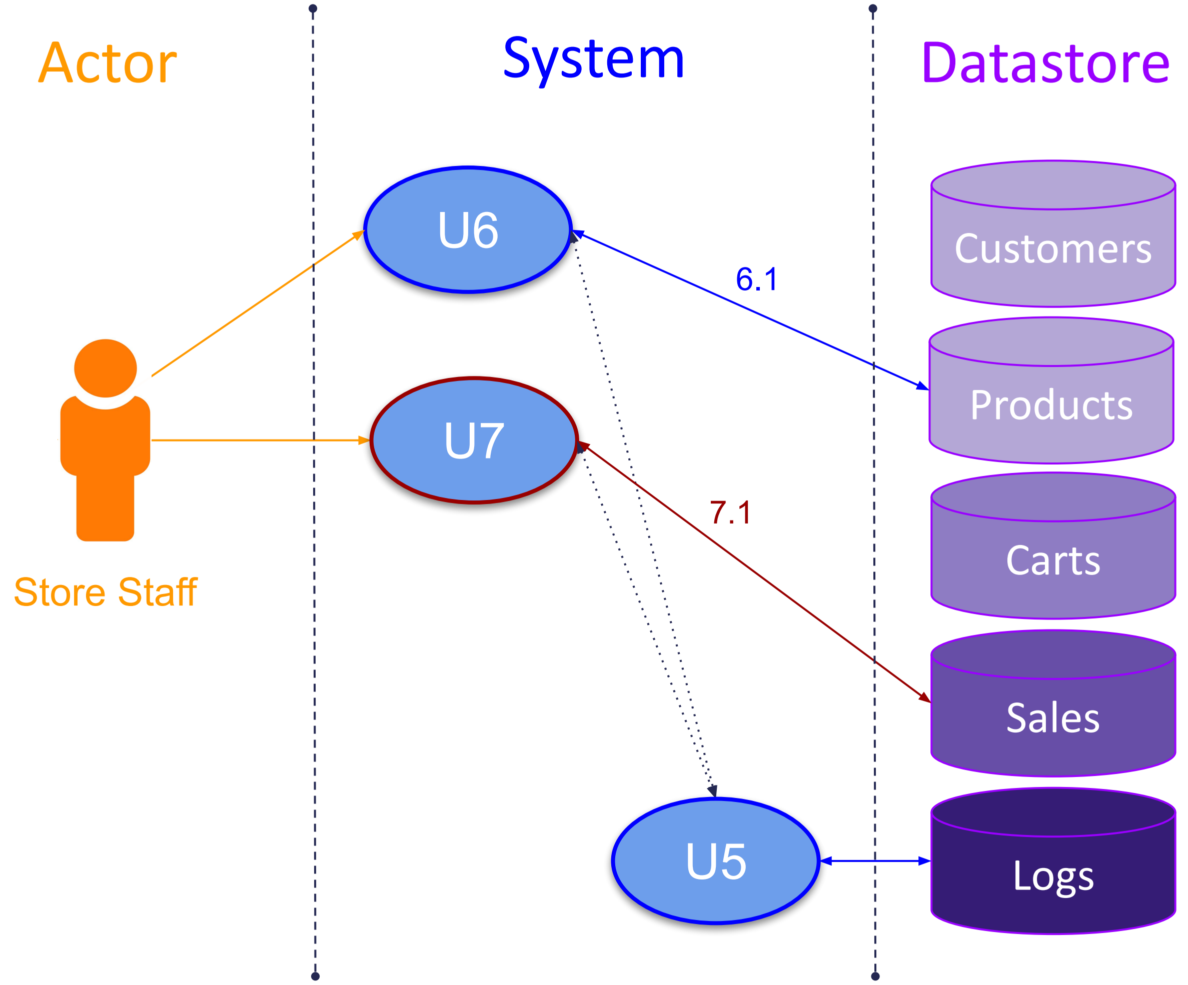
Customer Use Cases & Associated Datastore

Use Cases		Action
U1	Log in	1.1 Authentication & get customer info. 1.2 get Cart information 1.3 get latest price 1.4 Activate U5 with action='U1'
U2	Find Product	2.1 Get products by keywords 2.2 Activate U5 with action='U2'
U3	Update Shopping Cart	3.1(1) add a product to cart 3.1(2) remove products from cart 3.1(3) update quantity 3.2 Activate U5 with action='U3(n)'
U4	Checkout	4.1 Retrieve cart information 4.2 Retrieve latest price 4.3 Add Sales information 4.4 Remove purchased products from cart 4.5 Activate U5 with action='U4'
U5	Keep log	Insert action into logs datastore



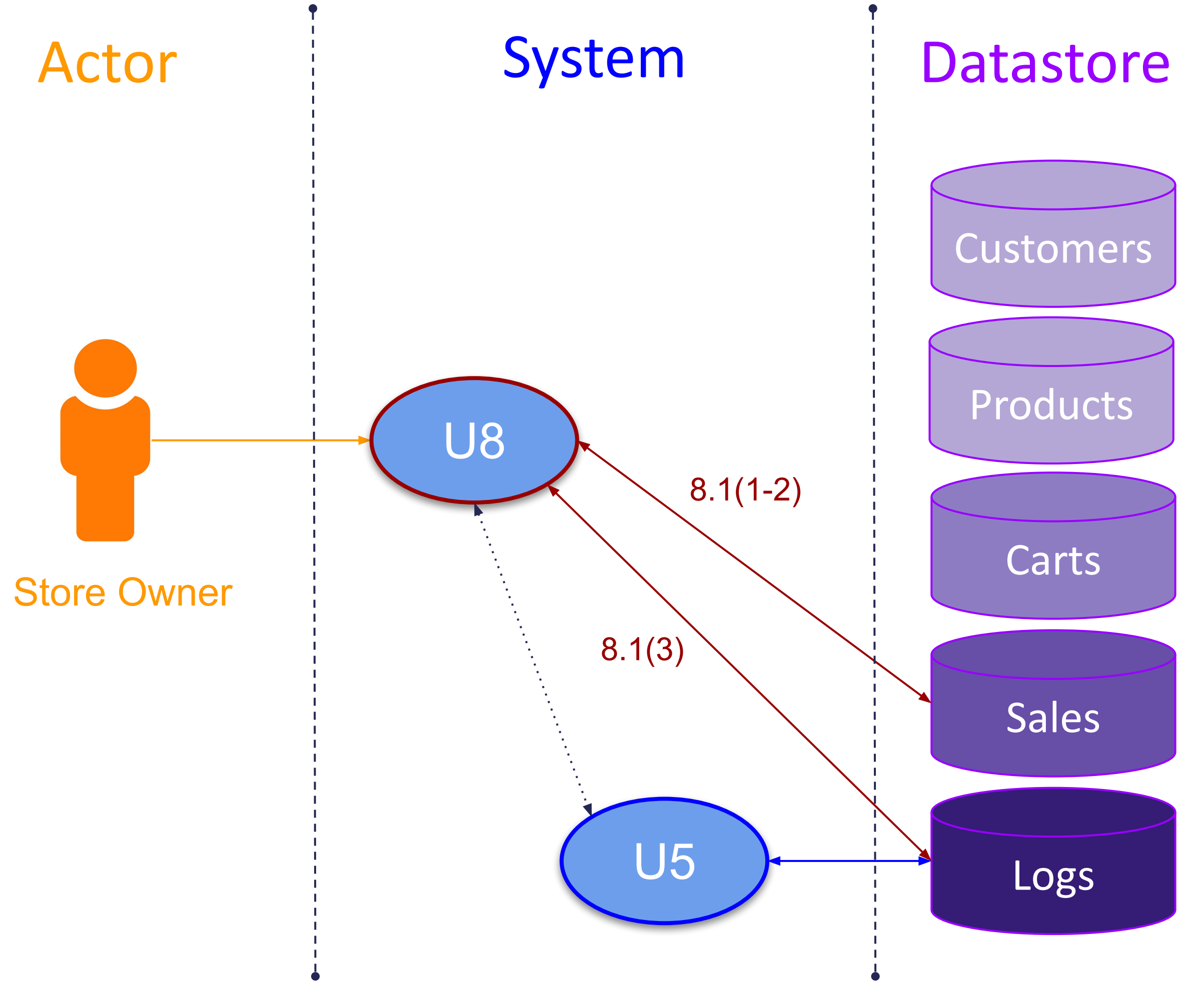
Store Staff Use Cases & Associated Datastore

	Use Cases	Action
U6	Update Product and sale information	6.1(1) add a product 6.1(2) update bestseller for products 6.1(3) delete last-5year-sale document from sales 6.2 Activate U5 with action='U6'
U7	View Daily Report	For a given date 7.1(1) Get count sales for a given date 7.1(2) Get Sum total sales 7.1(3) The most/least purchased product 7.1(4) The average purchase cost per customer 7.2 Activate U5 with action='U7'
U5	Keep log	Insert action into logs datastore

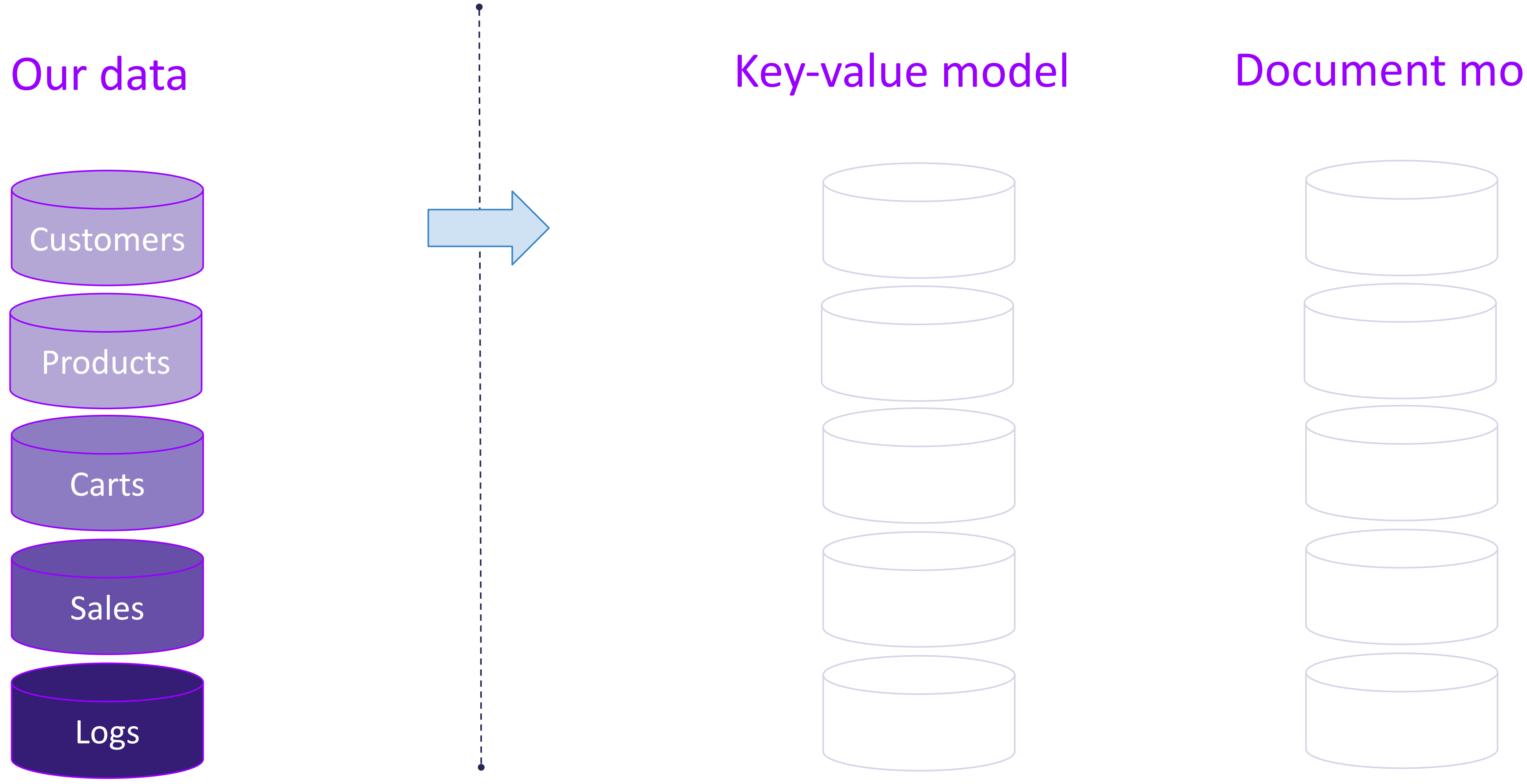


Store Owner Use Cases & Associated Datastore

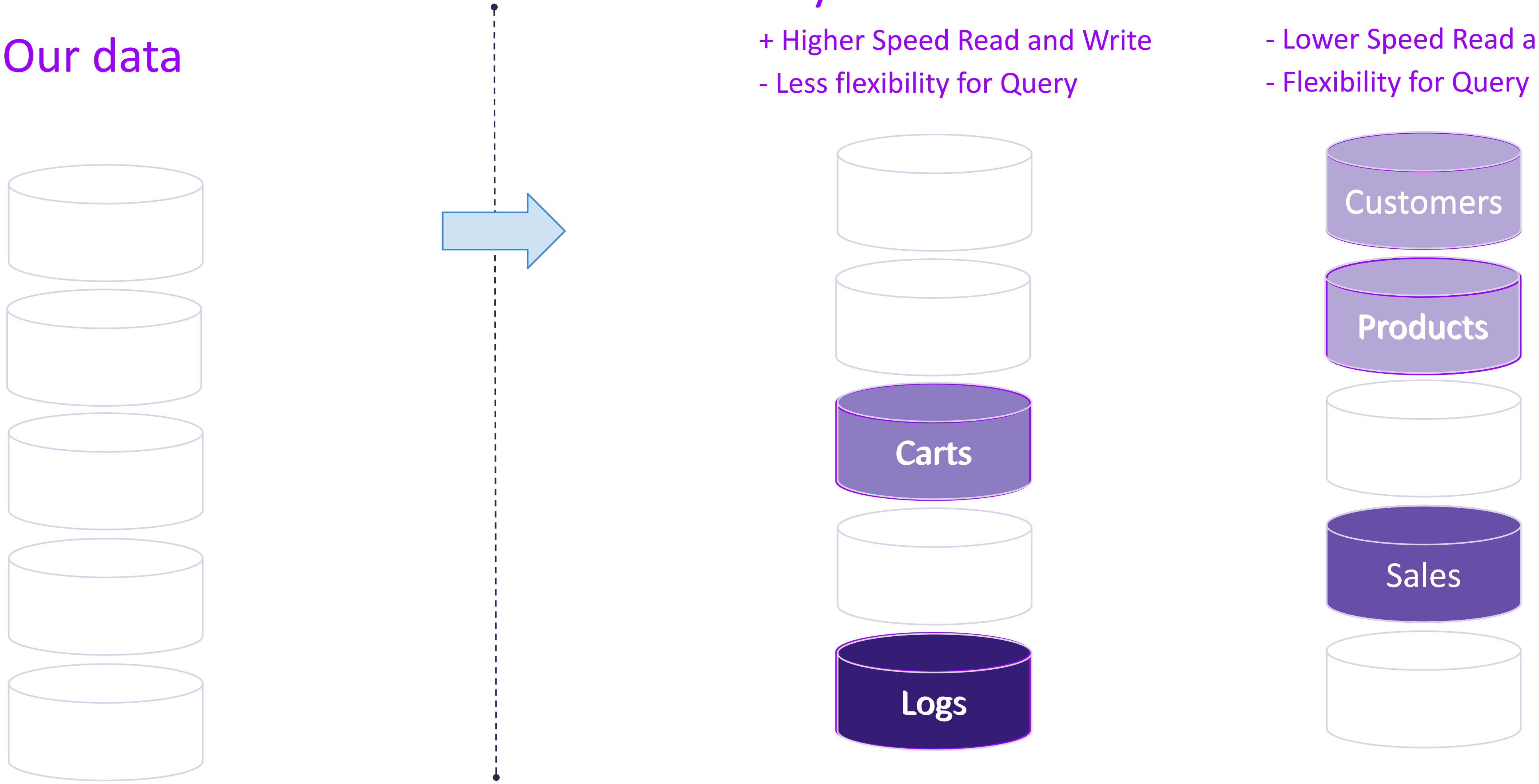
Use Cases		Action
U8	View Summary Report	Group by month, store type (i.e.,Online, Physical store) 8.1(1) Get the most popular product 8.1(2) Get total sales, count, average sales 8.2 Activate U5 with action='U8'
U5	Keep log	Insert action into logs datastore



Data Modeling



Data Modeling

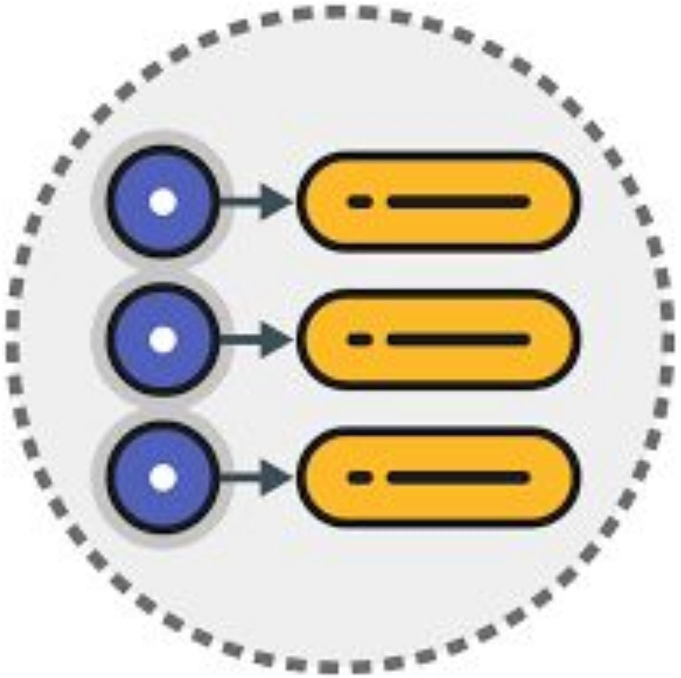


Key-Value Model

Key-Value Model

Key	Value
Henry	Name: Henry Country: France Age: 30 years
Goals101	Name: BestGoals Sport: Football Facebook: URL Video Object

Key-Value Model



The simplest model: Keys and Values

- No Schema
- Keys: synthetic or auto-generated
- Values: any object type (e.g., String, JSON, BLOB) stored as uninterpreted block, thus the keys are the only way to retrieve stored data.

Query operations for stored objects are associated with a key:

- PUT, GET, DELETE

Benefits vs. Limitations

BENEFITS

Extremely fast retrieval using the key

Virtually no restriction on the type of data that can be stored:

- Text (for example, the HTML code for a Web page)
- Any type of multimedia binary (still images, audio, and video).

LIMITATIONS

Cannot search within stored values rather than always retrieving by the key

Cannot update parts of a “value” while it’s in the database. You must replace the entire value with a new copy if modifications are needed.

Applications & Use Cases

Best suited for applications where access is only through the key.

They are used for Web sites that include thousands of pages, large image databases, and large catalogs. They are also particularly useful for keeping Web app session information.

Redis Use Cases

Redis Data Types	Example use cases
Redis String	**Session Cache:** Many websites leverage Redis Strings to create a session cache to speed up their website experience by caching HTML fragments or pages. **Queues:** Any application that deals with traffic congestion, messaging, data gathering, job management, or packer routing should consider a Redis Queue, as this can help you manage your queue size by rate of arrival and departure for resource distribution.
Redis Lists	**Social Networking Sites:** Social platforms like Twitter use Redis Lists to populate their timelines or homepage feeds, and can customize the top of their feeds with trending tweets or stories. **Leaderboards:** Forums like Reddit and other voting platforms leverage Redis Lists to add articles to the leaderboard and sort by most voted entries.

Redis Use Cases

Redis Data Types	**Example use cases**
Redis Sets	**Analyzing Ecommerce Sales:** Many online stores use Redis Sets to analyze customer behavior, such as searches or purchases for a specific product category or subcategory. For example, an online bookstore owner can find out how many customers purchased medical books in Psychology. **Inappropriate Content Filtering:** For any app that collects user input, it's a good idea to implement content filtering for inappropriate words, and you can do this with Redis Sets by adding words you'd like to filter to a SET key and the SADD command.
Redis Sorted Sets	**Q&A Platforms:** Many Q&A platforms like Stack Overflow and Quora use Redis Sorted Sets to rank the highest voted answers for each proposed question to ensure the best quality content is listed at the top of the page. **Task Scheduling Service:** Redis Sorted Sets are a great tool for a task scheduling service, as you can associate a score to rank the priority of a task in your queue. For any task that does not have a score noted, you can use the WEIGHTS option

Redis Use Cases

Redis Data Types	**Example use cases**
Redis Hash	**User Profiles:** Many web applications use Redis Hashes for their user profiles, as they can use a single hash for all the user fields, such as name, surname, email, password, etc. **User Posts:** Social platforms like Instagram leverage Redis Hashes to map all the archived user photos or posts back to a single user. The hashing mechanism allows them to look up and return values very quickly, fit the data in memory, and leverage data persistence in the event one of their servers dies.

Sources: https://scalegrid.io/blog/top-redis-use-cases-by-core-data-structure-types/

Getting familiarization with Redis Enterprise Cloud (free version)

redis

Key-Value Store

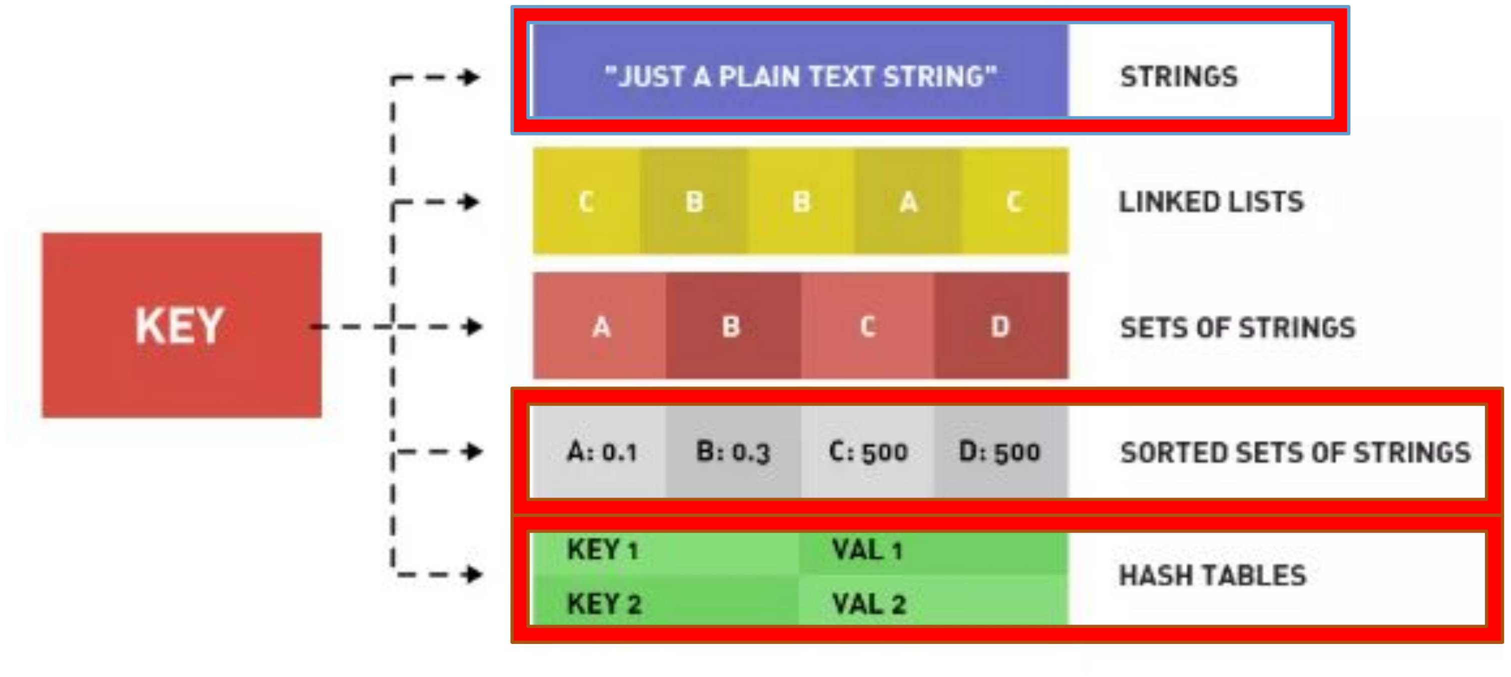
- What is Redis?
- Redis Practice Architecture
- Redis Commands
- Implement Redis for User session and shopping cart

What is Redis?

- A Key-Value Store.
- Stores and manipulates all data in memory that can be used as a database, cache, and message broker.
- Supports basic data structures such as strings, hashes, lists, sets, and sorted sets with range queries.
- More advanced data structures like bitmaps, hyperloglogs, and geospatial indexes with radius queries are also supported.

Redis Data Types: String, List, Set, Sorted Set, Hash

Redis Data Types



One key to rule them all.

STRING: Binary-safe string data with max allowed size 512 MB

LIST: Lists in Redis are implemented using a linked list. They are collections of string elements, sorted by insertion order.

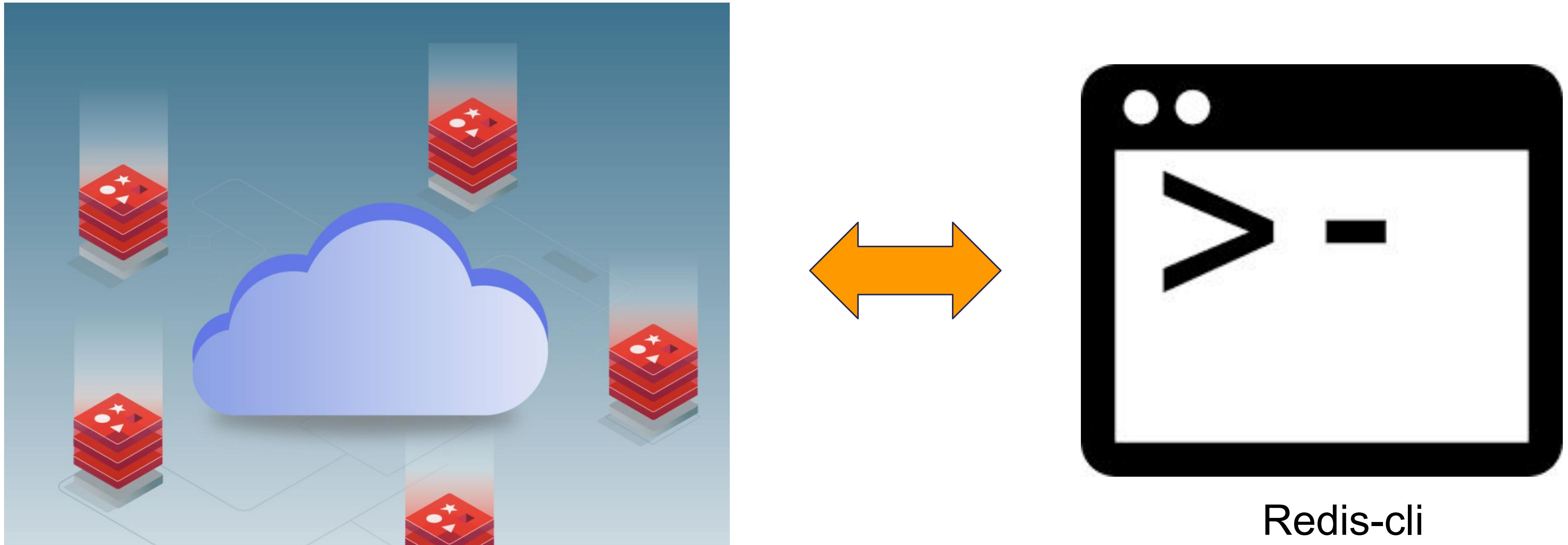
SET: A collection of unique strings with no ordering.

SORTED SET: A collection of unique strings ordered by user defined scoring

HASH: Unordered hash table of keys to values

Redis Command: https://redis.io/commands

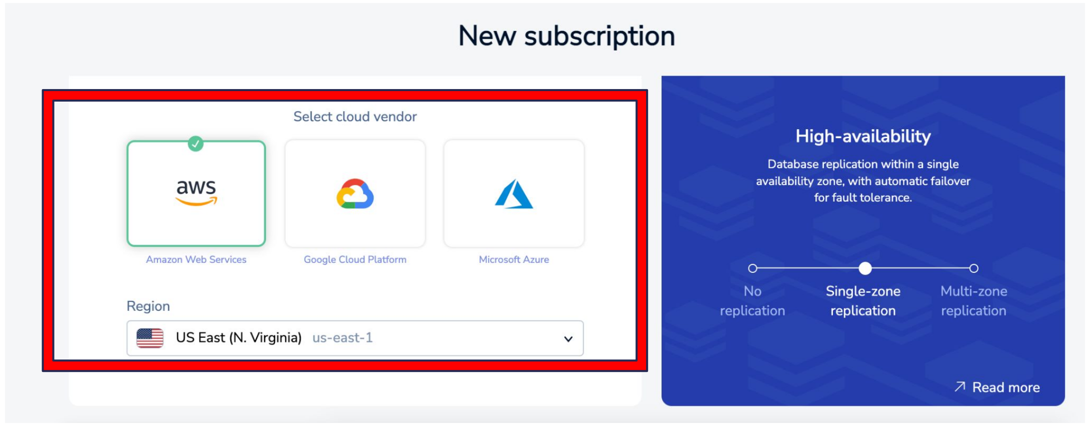
Redis Practice Lab Architecture



Redis Enterprise Cloud

Redis Enterprise Cloud (Free Version)

Register at https://redislabs.com/try-free/



New subscription

Select cloud vendor

[✓ Amazon Web Services] Google Cloud Platform Microsoft Azure

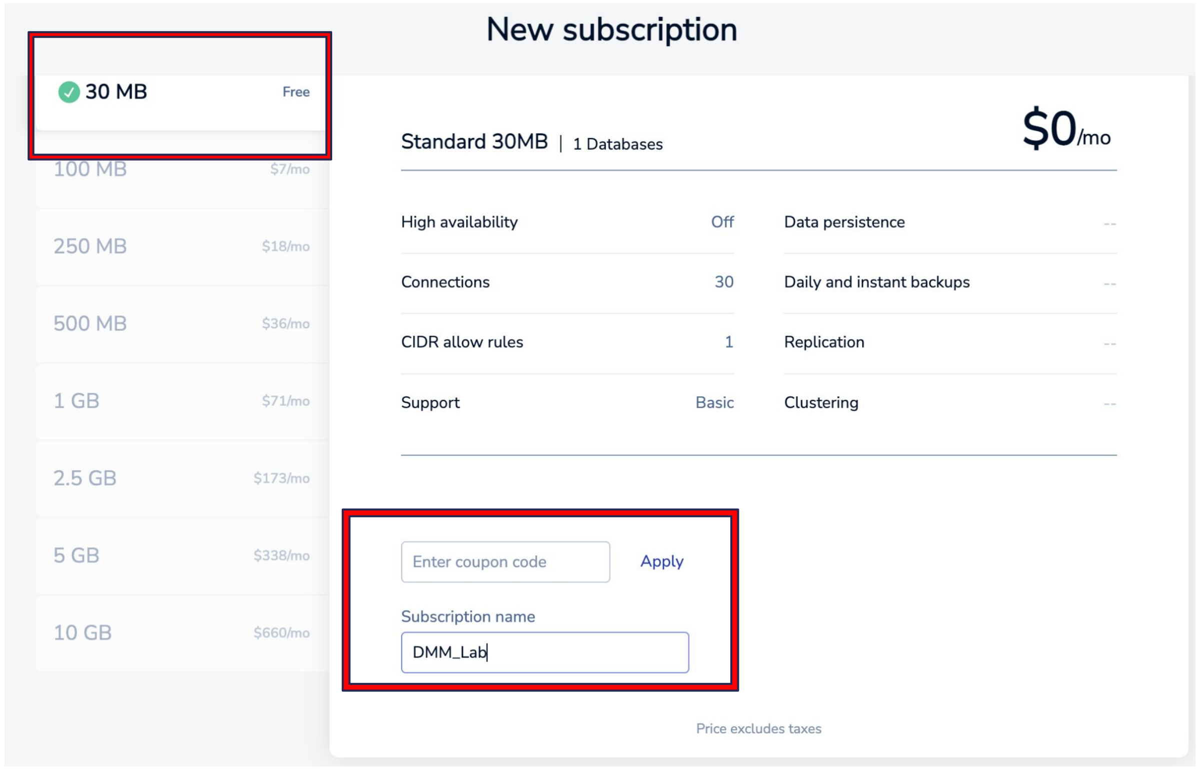
Region: US East (N. Virginia) us-east-1

High-availability

Database replication within a single availability zone, with automatic failover for fault tolerance.

No replication [✓ Single-zone replication] Multi-zone replication

Read more



New subscription

✓ 30 MB	Free
100 MB	$7/mo
250 MB	$18/mo
500 MB	$36/mo
1 GB	$71/mo
2.5 GB	$173/mo
5 GB	$338/mo
10 GB	$660/mo

Standard 30MB | 1 Databases — $0/mo

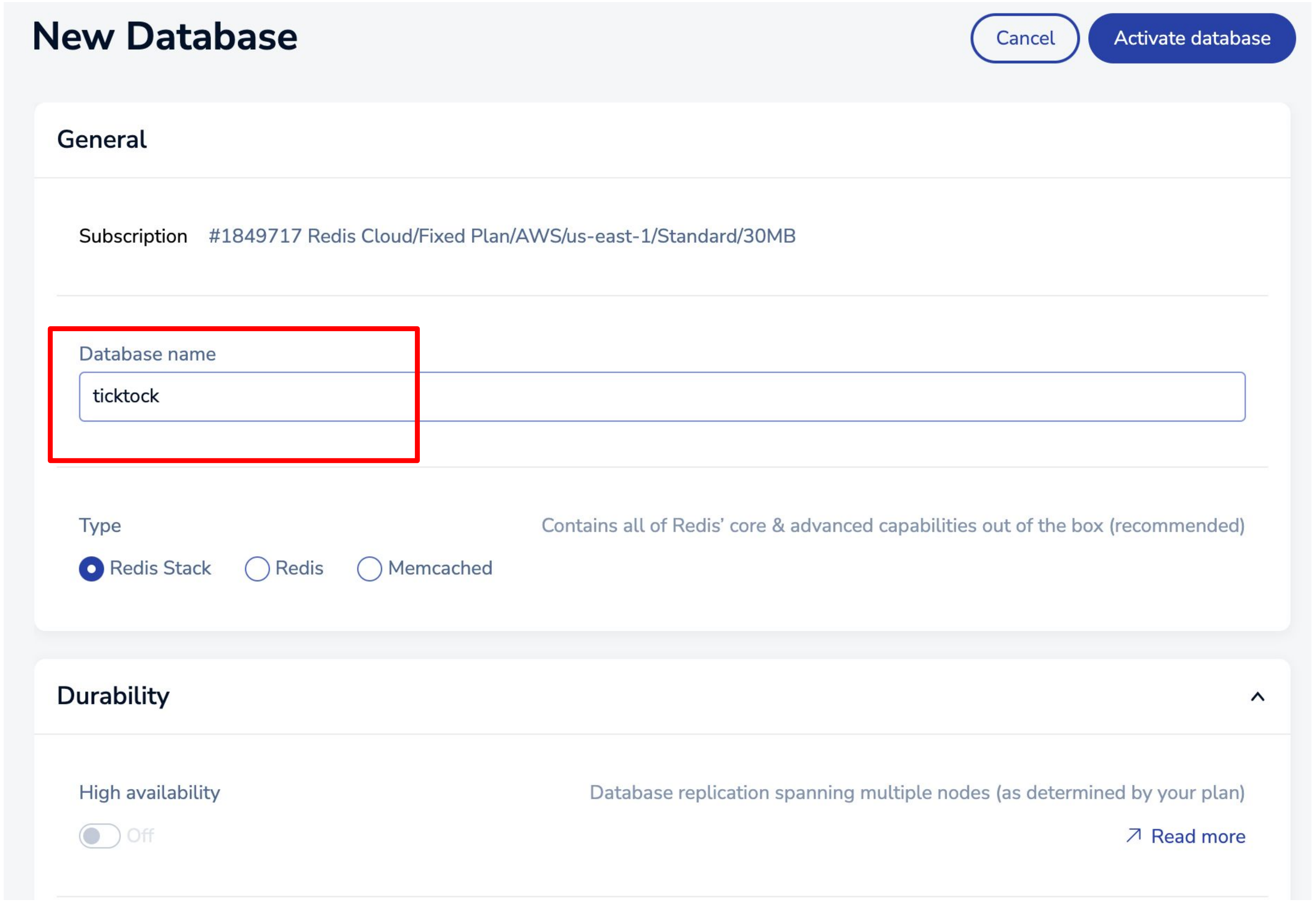
High availability	Off	Data persistence	--
Connections	30	Daily and instant backups	--
CIDR allow rules	1	Replication	--
Support	Basic	Clustering	--

Enter coupon code — Apply

Subscription name: DMM_Lab

Price excludes taxes

Redis Enterprise Cloud



New Database

General

Subscription #1849717 Redis Cloud/Fixed Plan/AWS/us-east-1/Standard/30MB

Database name: ticktock

Type — Contains all of Redis' core & advanced capabilities out of the box (recommended)

[✓ Redis Stack] Redis Memcached

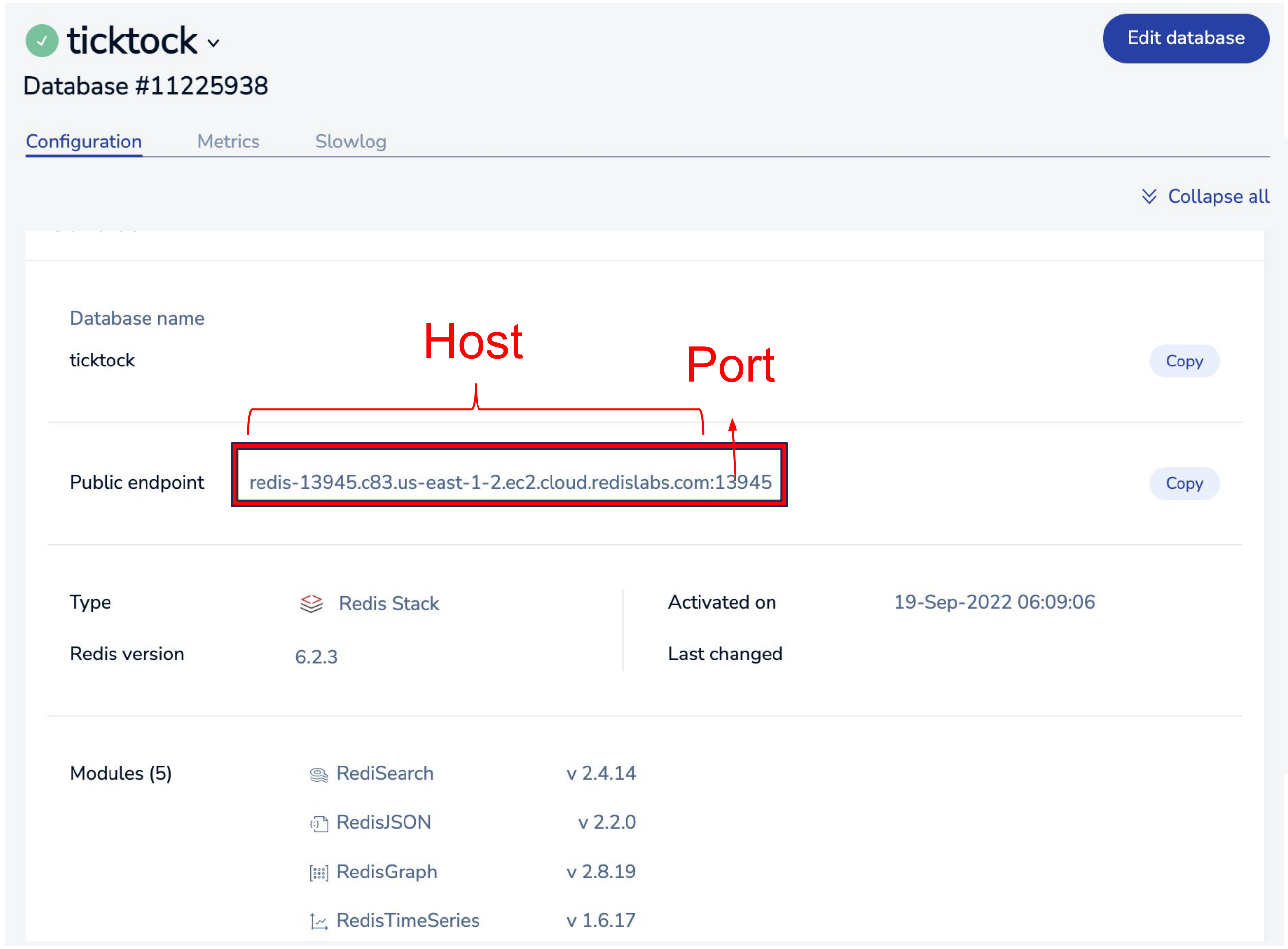
Durability

High availability — Database replication spanning multiple nodes (as determined by your plan)

Off

Enter the name and click **Activate Database**.

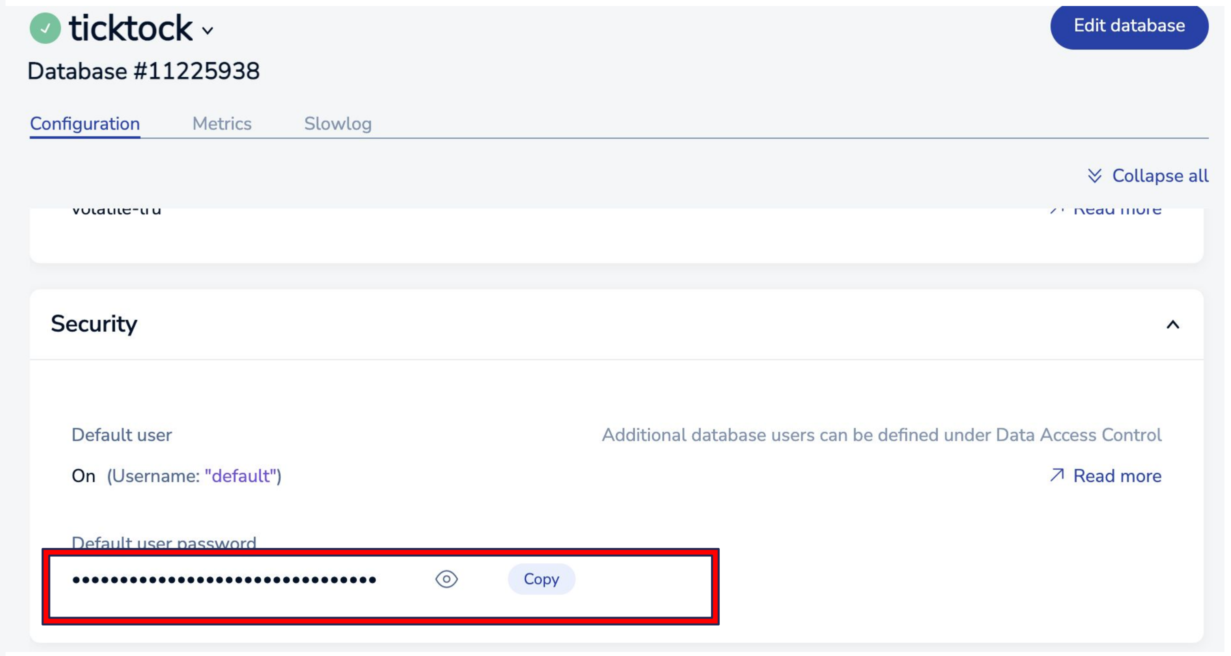
Redis Enterprise Cloud



Database #11225938

Database name	ticktock		
Public endpoint	redis-13945.c83.us-east-1-2.ec2.cloud.redislabs.com:13945		
Type	Redis Stack	Activated on	19-Sep-2022 06:09:06
Redis version	6.2.3	Last changed	

Modules (5)		
	RediSearch	v 2.4.14
	RedisJSON	v 2.2.0
	RedisGraph	v 2.8.19
	RedisTimeSeries	v 1.6.17



Database #11225938

volatile-lru

Security

Default user

On (Username: "default")

Additional database users can be defined under Data Access Control

Default user password

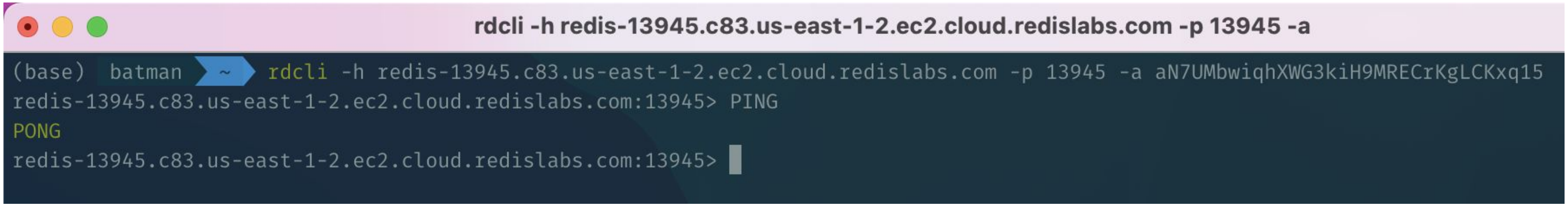
••••••••••••••••••••••••••••••••

Our remote server is ready, let’s connect with CLI

Install Redis-cli without Installing Redis-server

1. **Install Nodejs: https://nodejs.org/en/download/**
2. **From your command line, install the Node.js version of redis-cli:**
   ```
   > npm install -g redis-cli
   ```
3. **Connect to redis server:**
   ```
   > rdcli -h <your redis host name> -p <your redis port number> -a
   <your redis password>
   ```

Connect to Redis Enterprise Cloud using redis-cli



```
rdcli -h redis-13945.c83.us-east-1-2.ec2.cloud.redislabs.com -p 13945 -a
```

```
(base)  batman  ~  rdcli -h redis-13945.c83.us-east-1-2.ec2.cloud.redislabs.com -p 13945 -a aN7UMbwiqhXWG3kiH9MRECrKgLCKxq15
redis-13945.c83.us-east-1-2.ec2.cloud.redislabs.com:13945> PING
PONG
redis-13945.c83.us-east-1-2.ec2.cloud.redislabs.com:13945> 
```

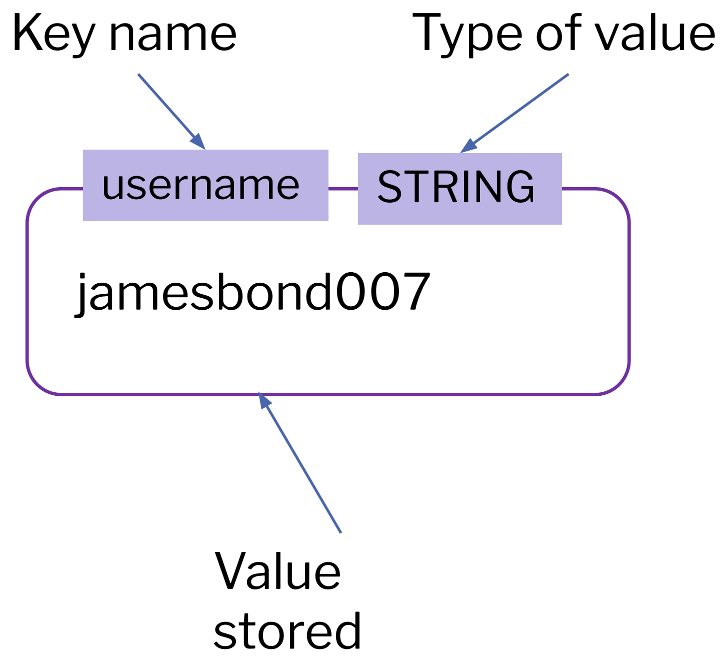
NOTE:
While you copy the endpoints from the Cloud server (from previous slide) make sure you remove the port number like in the above example.

Test connection:

- rdcli -h redis-13945.c83.us-east-1-2.ec2.cloud.redislabs.com -p 13945 -a aN7UMbwiqhXWG3kiH9MRECrKgLCKxq15
- redis-13945.c83.us-east-1-2.ec2.cloud.redislabs.com:13945> ping
- PONG

Redis STRING

Redis *String* type is the simple type of value.



Command	Meaning
SET	Set the value stored at the given key
GET	Retrieves the data stored at the given key
DEL	Delete the value stored at the given key (use for all types)

Note: **MSET** and **MGET** commands are used to set or retrieve the value of multiple keys in a single command

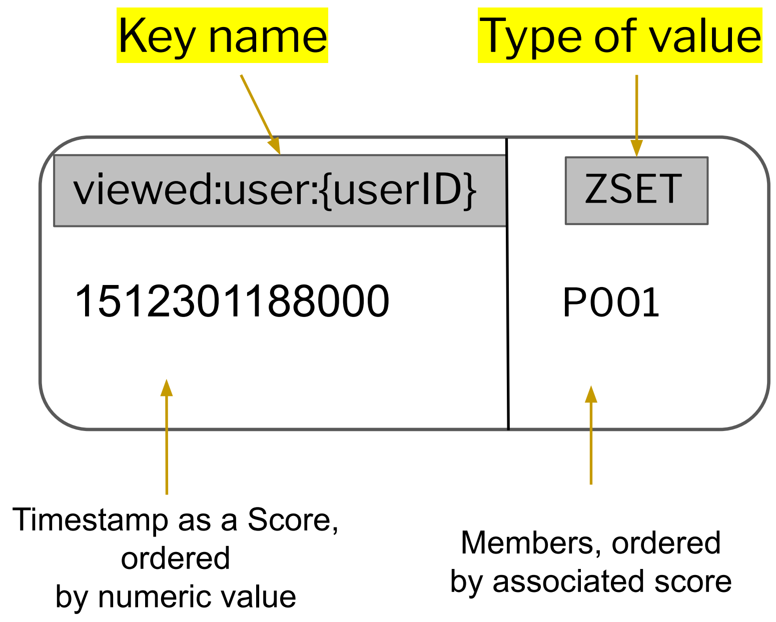
Redis STRING example

```
> SET login:session-1 "{user_id:swaarup}"
OK

> MSET login:session-1
"{user_id:swarup}" login:session-2
“{user_id:saam}”

OK

> GET login:session-2
“{user_id:saam}”

> MGET login:session-1 login:session-2


> DEL login:session-2
(integer) 1
```

```
> SET view_count 10
OK

> INCR view_count
(integer) 11

> INCR view_count
(integer) 12
```

Redis SORTED SET (ZSET)

Redis SORTED SET is a collection of unique strings ordered by user defined scoring. Every element in a sorted set is associated with a floating-point value (called score).



Key name
Type of value
viewed:user:{userID}
ZSET
1512301188000
P001
Timestamp as a Score, ordered by numeric value
Members, ordered by associated score

Command	Meaning
ZADD	Adds the value with the given score to the ZSET
ZRANGE	Retrieves the values in the ZSET from their position in the sorted order
ZRANGEBYSCORE	Retrieves the values in the ZSET based on a range of scores
ZREM	Remove the value from the ZSET, If it exists

Redis SORTED SET (ZSET) example

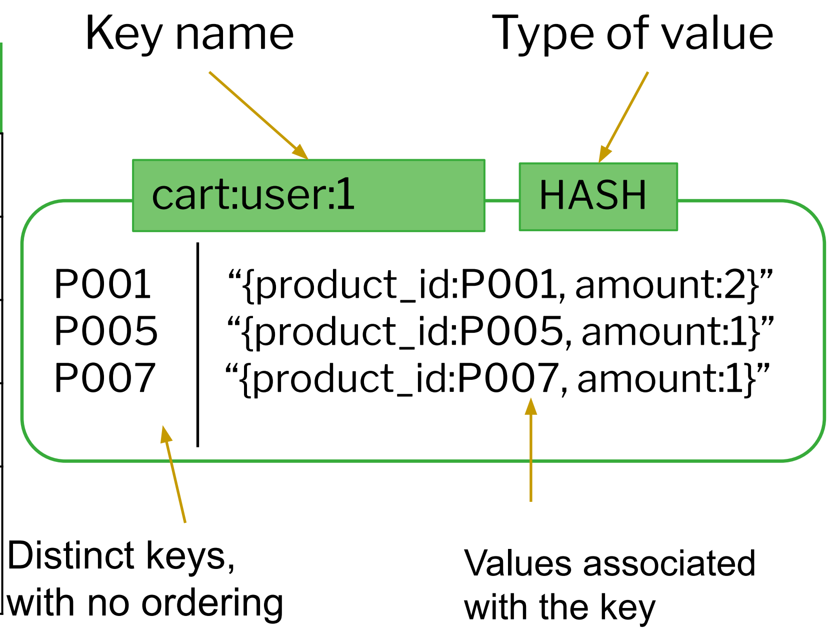
```
> ZADD viewed:user:saam 1590215629000 P001
(integer) 1  //success

> ZADD viewed:user:saam 1 P002
(integer) 1

> ZADD viewed:user:saam 1590215629004 P003
(integer) 1

> ZADD viewed:user:saam 1590215629006 P004
(integer) 1

>ZADD viewed:user:saam 1590215629008 P001
(integer) 0   //cannot add duplicate value
```

```
> ZRANGE viewed:user:saam 0 -1
1) "P002"
2) "P003"
3) "P004"
4) "P001"  // recently


//keep last 3 viewed products
> ZREMRANGEBYRANK viewed:user:saam  0
-4
(integer) 1

> ZRANGE viewed:user:saam 0 -1
1) "P003"
2) "P004"
3) "P001"
```

Redis HASH

Redis HASH is a collection of key-value pairs. The value which stored in HASH can be strings and numbers.

Command	Meaning
HSET	Stores the value at the key in the hash
HGET	Retrieves the value at the given hash key
HGETALL	Retrieves the entire hash
HDEL	Remove the key from the hash, if it exists
HLEN	Returns the number of fields contained in the hash stored at key.


Key name
Type of value
cart:user:1
HASH
P001
P005
P007
"{product_id:P001, amount:2}"
"{product_id:P005, amount:1}"
"{product_id:P007, amount:1}"
Distinct keys, with no ordering
Values associated with the key

Note: **HMSET** and **HMGET** commands are used to set or retrieve the value of multiple keys in a single command

Redis HASH example

```
> HSET cart:user:saam P001 "{product_id:P001, amount:2}"
(integer) 1   //success

> HSET cart:user:saam P005 "{product_id:P005, amount:1}"
(integer) 1

> HGET cart:user:saam P005
"{product_id:P005, amount:1}"

> HGETALL cart:user:saam
1) "P001"
2) "{product_id:P001, amount:2}"
3) "P005"
4) "{product_id:P005, amount:1}"
```

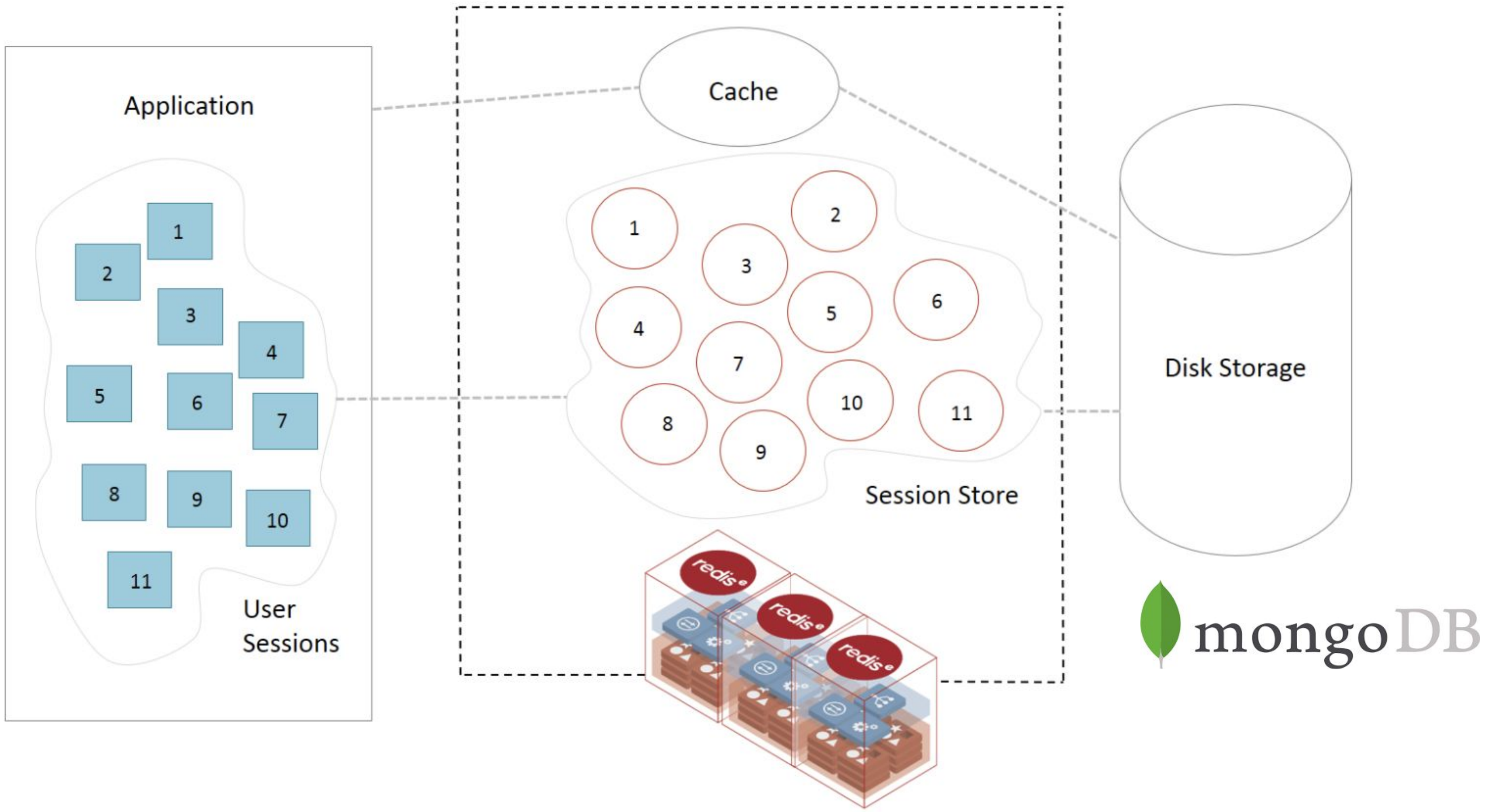
```
> HLEN cart:user:saam
2
> HDEL cart:user:saam P005
(integer) 1

> HGETALL cart:user:saam
1) "P001"
2) "{product_id:P001, amount:2}"

> DEL cart:user:saam
(integer
) 1
```

Redis in E-Commerce System

Functions

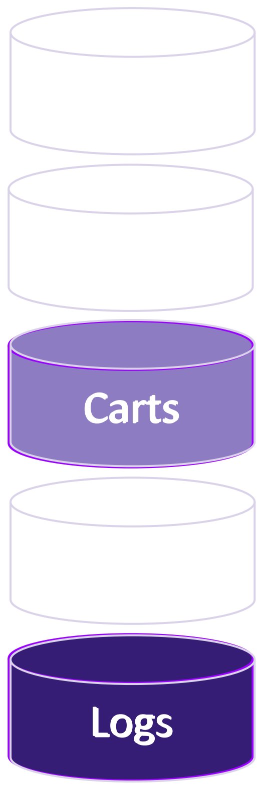
1. User Session Management : Login Sessions
2. User Behavior Log Management: Viewed Products Log
3. Shopping Cart Management

Designing Cache and Session Store with Redis



Source: https://redislabs.com/blog/cache-vs-session-store/

Summary Key-Value Data Model



Data		Key	Value	Data type
Login Session	sessionID user_id	login:{sessionID}	user_id	String
	timestamp sessionID	recent	timestamp sessionID	Sorted Set
Viewed Products Log	userID timestamp product_id	viewed:{email}	timestamp produc_id	Sorted Set
Shopping Cart	userID product_id amount price	cart:{email}	product_id amount price	Hash

STRING

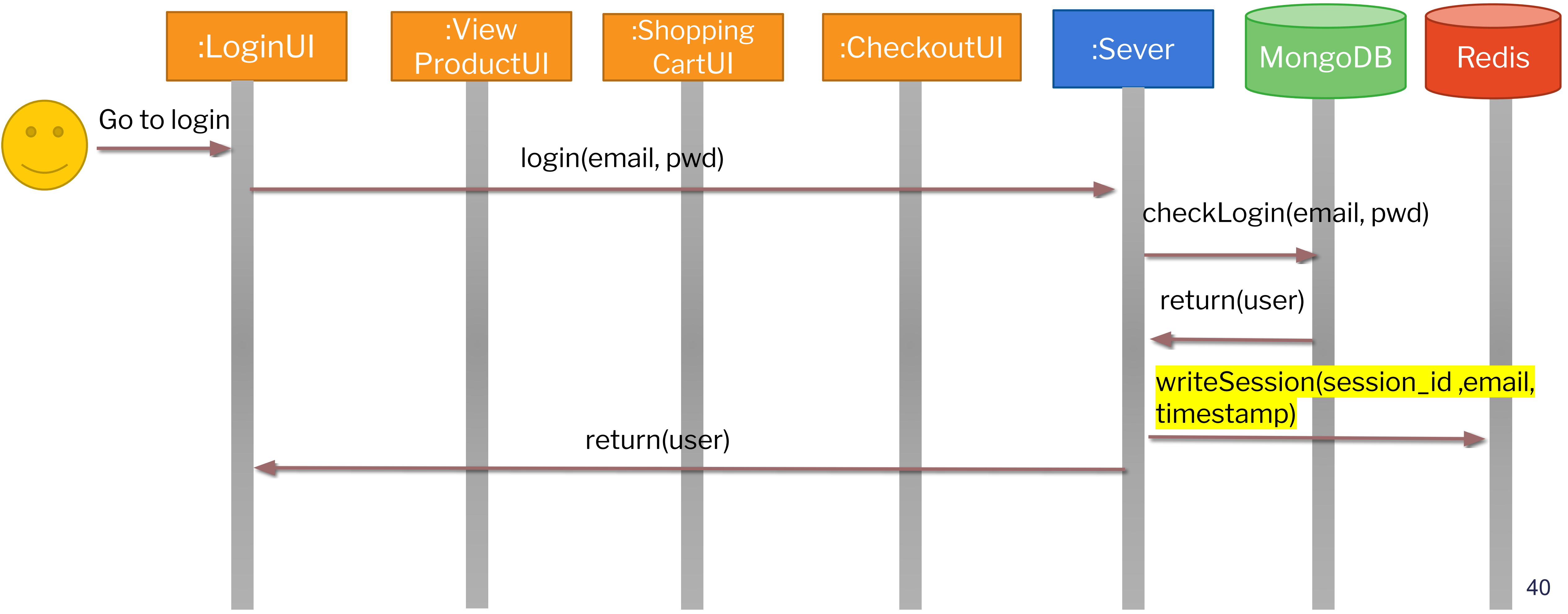
LIST

SET

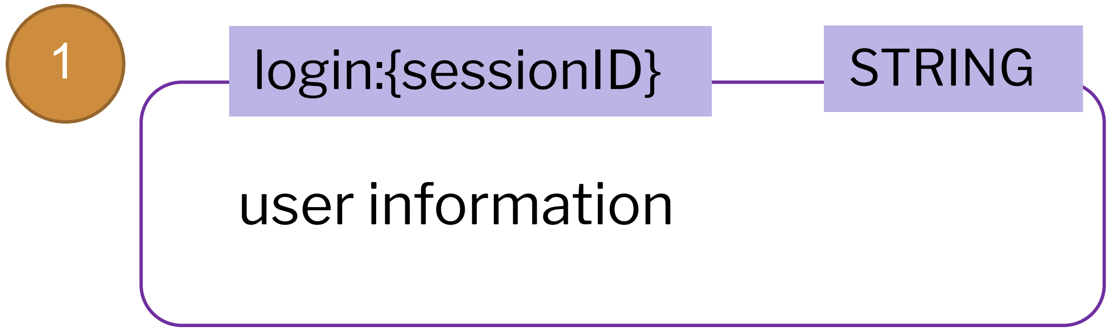
SORTED SET

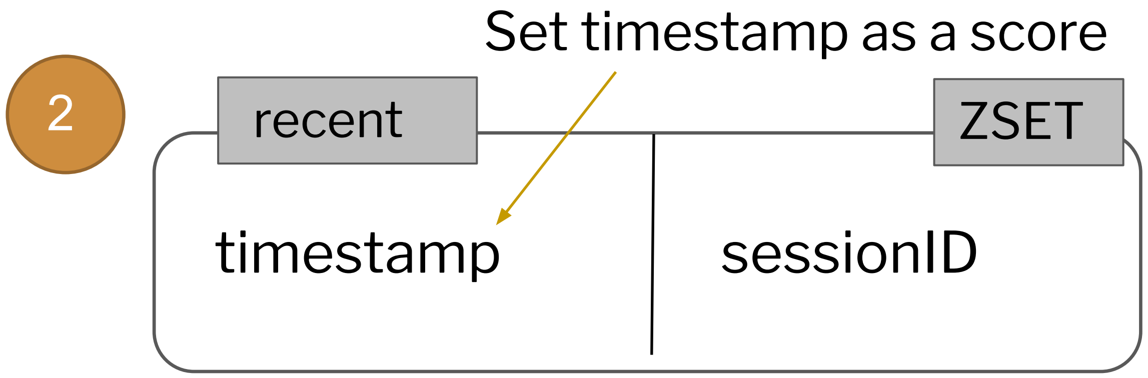
HASH

U1. Login Session Management



(U1) writeSession(session_id ,user_id, timestamp)





Example Data:

Key	Value
login:session-1	"{email:'june@gmail.com'}"
login:session-2	"{email:'tcrawford@hotmail.com'}"

Key	Value
recent	1511533205001 session-1
recent	1511532142401 session-2

Example Commands:

```
SET login:session-1 "{email:'june@gmail.com'}"
GET login:session-1
DEL login:session-1
```

```
ZADD recent 1511533205001 session-1
ZRANGE recent 0 -1
ZREMRANGEBYRANK recent 0 -51
```

U1. Login Session Management

1. User information

Create login session as a string: use command SET and set login:{sessionID} as a key.

```
> SET login:qHsXwI9URyRms81l7mjHOw  "{email:'june@gmail.com'}"
```

2. Log access

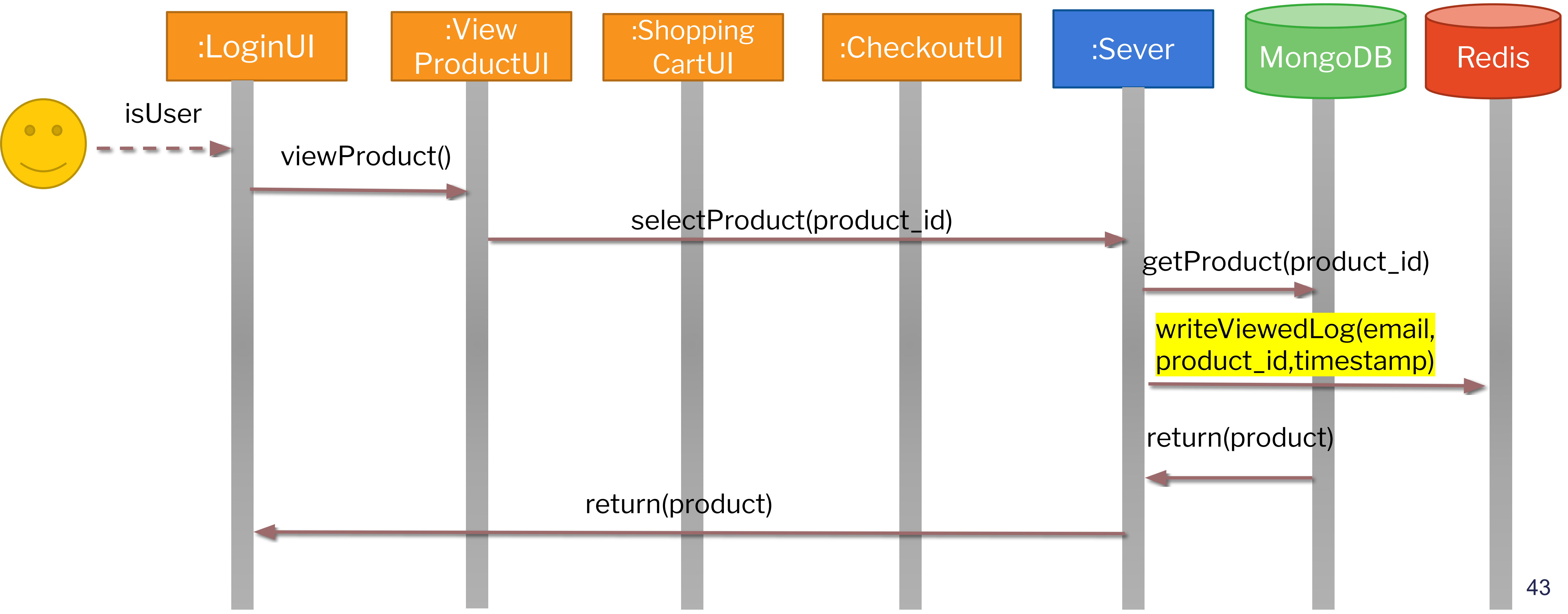
Create recent login session: use command ZADD and set recent as a key.

```
> ZADD recent 1511533205001 session-1
//expire session each 1 hour
```

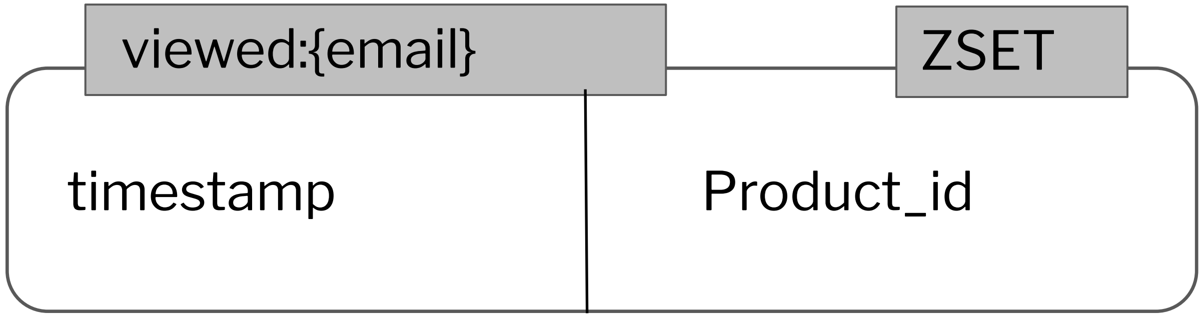
3. Set expire session: use command EXPIRE key second

```
 > EXPIRE login:qHsXwI9URyRms81l7mjHOw 3600
  // keep only the top 100 recent session
 > ZREMRANGEBYRANK recent 0 -101
```

U2. Recently Viewed Products Log



(U2.) writeViewedLog(user_id, product_id,timestamp)



Store viewed product log inside a sorted set with a key like “viewed:{email}” In sorted set, we store timestamp as a score and product_id

Example Data:

Key	Value
viewed:june@gmail.com	1511533205001 P001
viewed:tcrawford@hotmail.com	1511532142401 P005

Example Commands:

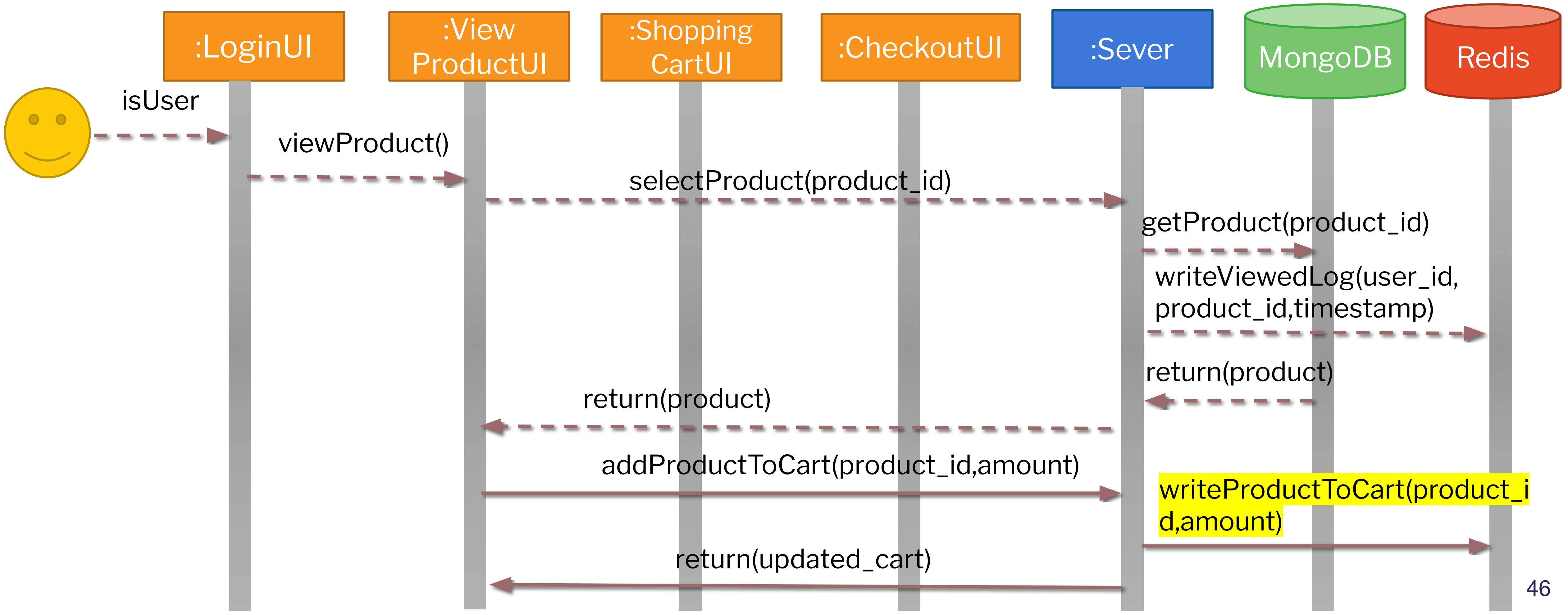
```
ZADD viewed:june@gmail.com 1511533205001 P001
ZRANGE viewed:tcrawford@hotmail.com 0 -1 withscores
ZREMRANGEBYRANK viewed:user:1 0 -11
```

Task: Recently Viewed Products Log

Instruction:

1. Create viewed product log of product_id P001 and P005 for “session-1”: use command ZADD and set viewed as a key.
2. Remove old items of viewed product, keeping the most recent 25

U3. Shopping Cart Management



U3. writeProductToCart(product_id,amount)

cart:{email} HASH

product_id : “CartItem Detail”
product_id : “CartItem Detail”
product_id : “CartItem Detail”

Store a cart object inside a hash with a key like "cart:user:{userID}". Inside this hash, we store the product_id as a key and the value is the item details in JSON format.

Example Data:

Key	Value

cart:june@gmail.com	`P003: "{product_id:P003,amount:2}"`
	`P004: "{product_id:P004,amount:10}"`

Example Commands:

```
HSET cart:june@gmail.com P004 "{product_id:P004,amount:10}"
HDEL cart:june@gmail.com P004
HGETALL cart:user:1
```

U3. writeProductToCart(product_id,amount)

Add Product P001 in HASH: use command **HSET** and set **cart:{email}** as a key.

```
> HSET cart:june@gmail.com P001 "{product_id:P001,amount:2}"
```

Add Product P002 and P003 in HASH : use command **HMSET**

```
> HMSET cart:june@gmail.com P002 "{product_id:P002,amount:1}" P003
"{product_id:P003,amount:5}"
```

Delete Product Item in Hash: use command **HDEL**

```
> HDEL cart:june@gmail.com P001
```

Get all data in Hash: use command **HGETALL**

```
> HGETALL cart:june@gmail.com
```

Thank you.