

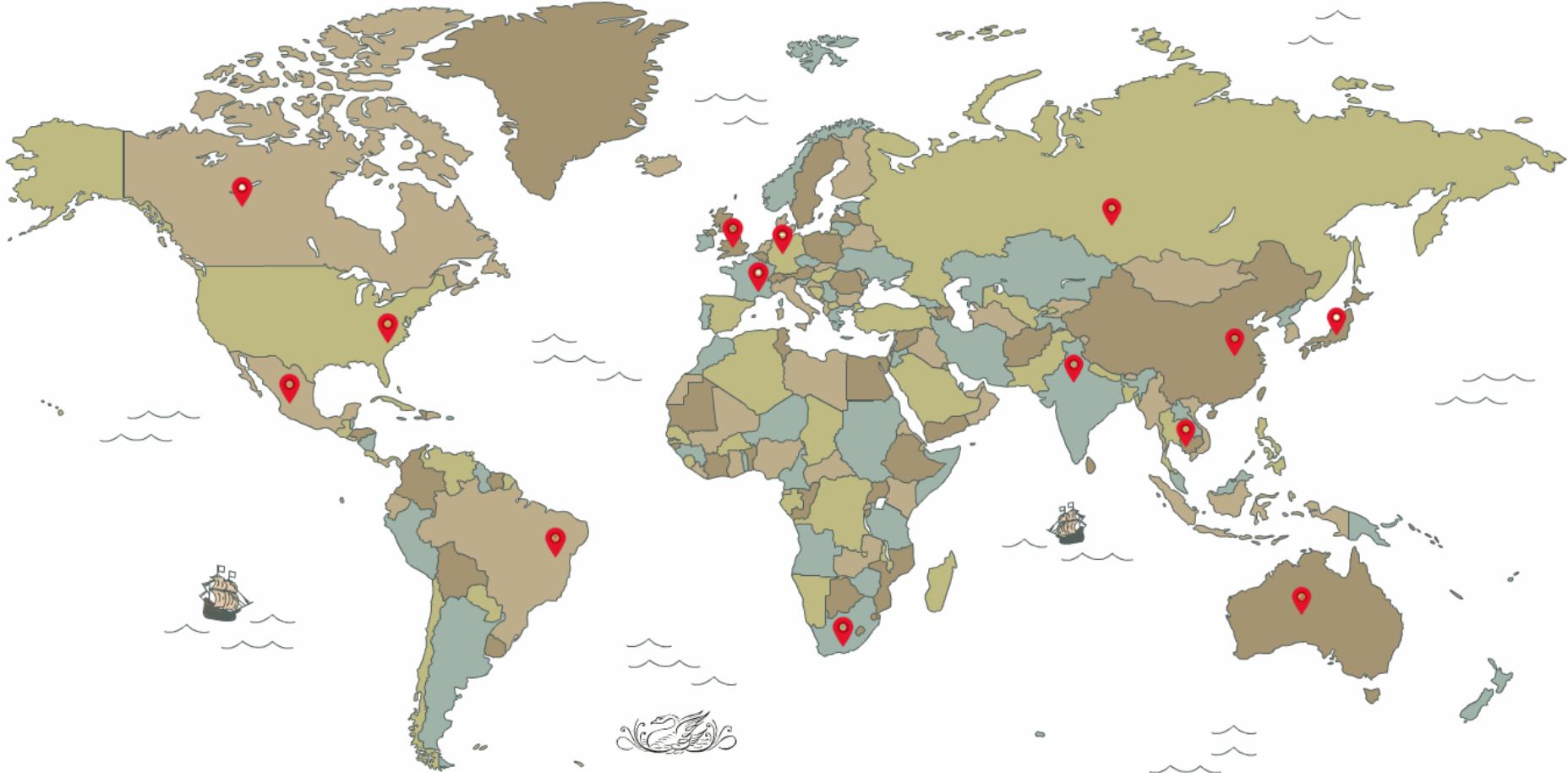


Key Value Store Database: Redis

Information Systems

Master Course: Advanced Information Systems 2016

Lecturer: Prof. Dr. Barbara Sprick, PhD. Ajinkya Prabhune



Key Value Store Database: Redis

Information Systems

Master Course: Advanced Information Sytems 2016

Lecturer: Prof. Dr. Barbara Sprick, PhD. Ajinkya Prabhune



HOCHSCHULE
SRH HEIDELBERG
Intelligence in Learning



Key Value Store Database: Redis

Information Systems

Master Course: Advanced Information Systems 2016

Lecturer: Prof. Dr. Barbara Sprick, PhD. Ajinkya Prabhune

Participants

Redis Team

Alpana Chaphalkar (11007146)

Ankit Dixit (11007129)

Jaydeep Galani (11007162)

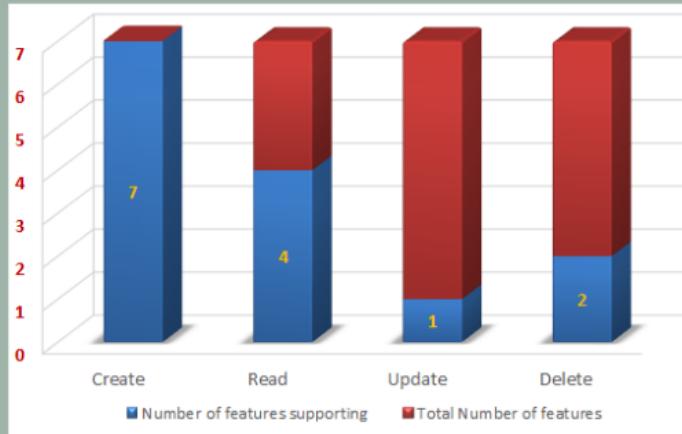
Mukhtar Baloch (11007170)

Redis Database Introduction

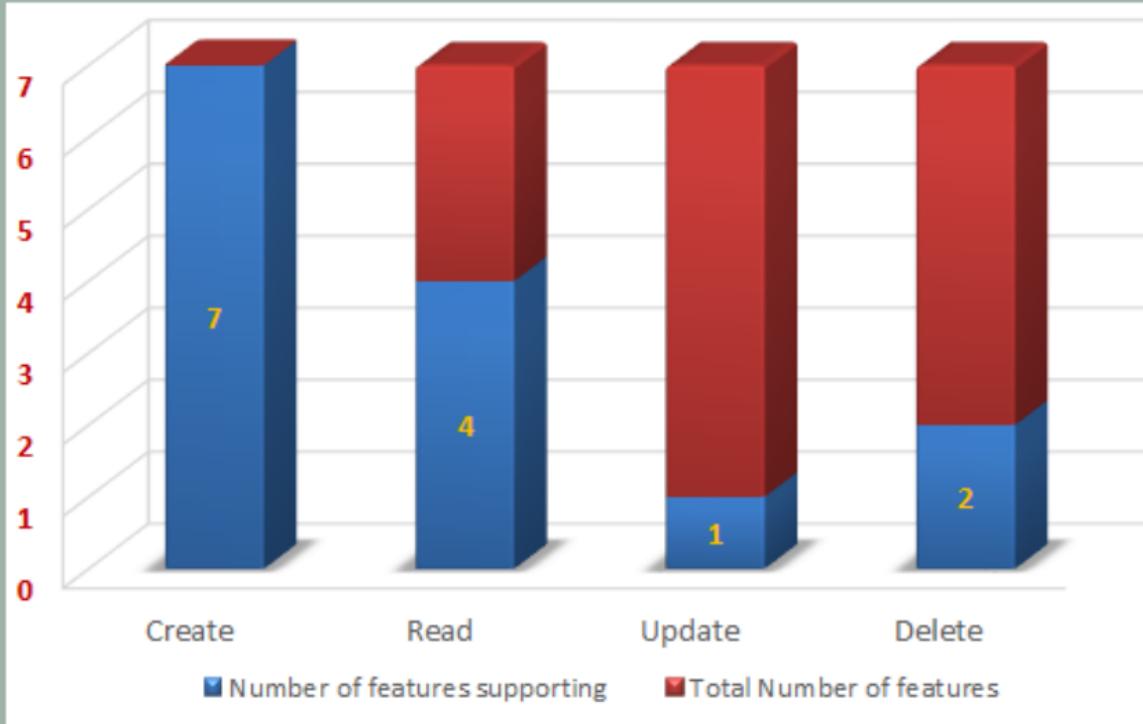
- In Memory Data Store
- Supports Consistency and partition tolerance
- Supports advanced data structures like List, Set, Hash
- Specifically Designed for fast access

Redis Datamodel

Sr. No.	Application Features	Data Structure Names
1	<i>User Login</i>	<i>Users ID</i>
		<i>User Login Profile</i>
2	<i>User Subscription</i>	<i>Subscription List</i>
3	<i>User Favorite List</i>	<i>Favorite List</i>
4	<i>Application Status</i>	<i>Application Status</i>
5	<i>User Search History</i>	<i>User Search History</i>
6	<i>Location Price Trend</i>	<i>Estate Types List</i>
		<i>Price Trend</i>
7	<i>Location Crime Trend</i>	<i>Crime Trend</i>

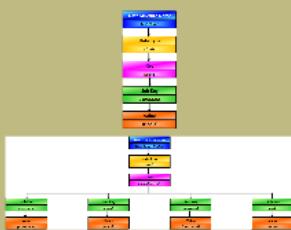


Sr. No.	Application Features	Data Structure Names
1	<i>User Login</i>	<i>Users ID</i>
		<i>User Login Profile</i>
2	<i>User Subscription</i>	<i>Subscription List</i>
3	<i>User Favorite List</i>	<i>Favorite List</i>
4	<i>Application Status</i>	<i>Application Status</i>
5	<i>User Search History</i>	<i>User Search History</i>
6	<i>Location Price Trend</i>	<i>Estate Types List</i>
		<i>Price Trend</i>
7	<i>Location Crime Trend</i>	<i>Crime Trend</i>



User Login

Data Structure



CRUD Operations

Create

- > HMSET users (username) {user_id}
- > HMSET users:{user_id} username (username) email (email) password (password) auth (auth)

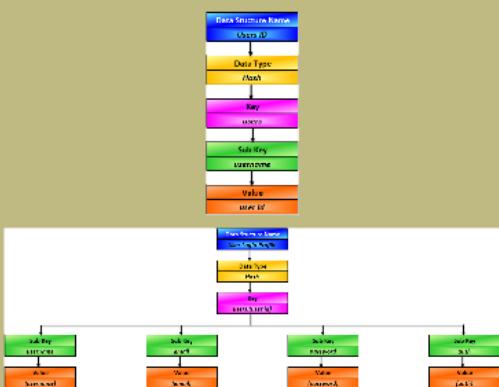
Read

- > HGETALL users
- > HGETALL users:{user_id}

Update

- > HSET users:{user_id} password (password)

Data Structure



CRUD Operations

Create

- > HMSET users (username) (user_id)
- > HMSET users:(user_id) username (username) email (email) password (password) auth (auth)

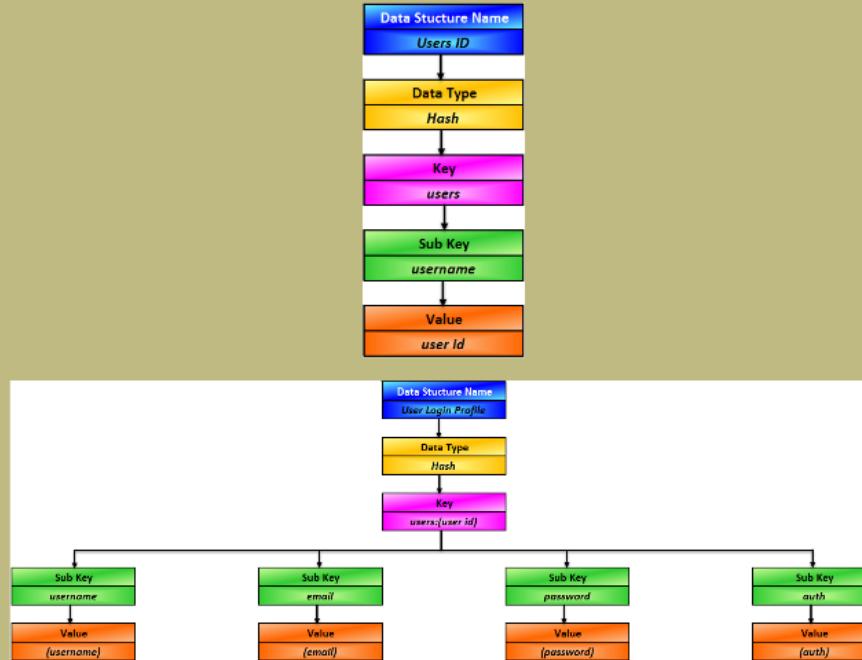
Read

- > HGETALL users
- > HGETALL users:(user_id)

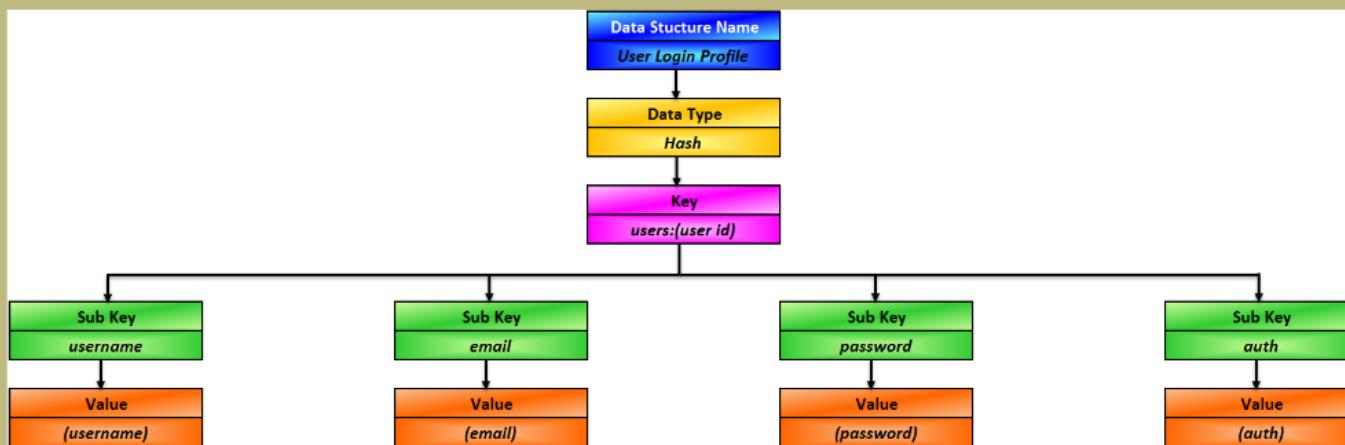
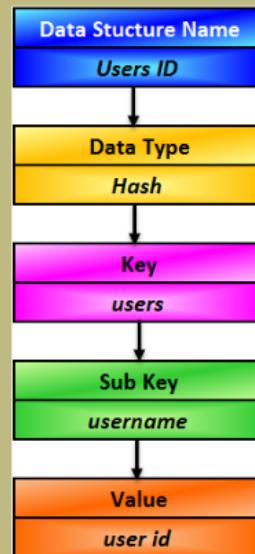
Update

- > HSET users:(user_id) password (password)

Data Structure



Data Structure



CRUD Operations

Create

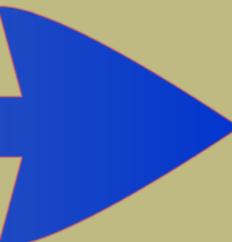
- > HMSET users (username) (user_id)
- > HMSET users:(user_id) username (username) email (email) password (password) auth (auth)

Read

- > HGETALL users
- > HGETALL users:(user_id)

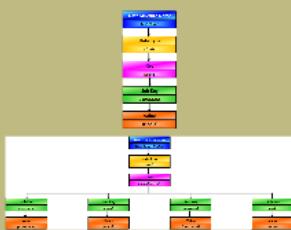
Update

- > HSET users:(user_id) password (password)



User Login

Data Structure



CRUD Operations

Create

- > HMSET users (username) {user_id}
- > HMSET users:(user_id) username (username) email (email) password (password) auth (auth)

Read

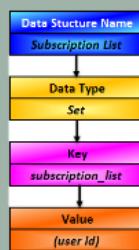
- > HGETALL users
- > HGETALL users:(user_id)

Update

- > HSET users:(user_id) password (password)

Subscription List

Data Structure



CRUD Operations

Create

> SADD subscription_list (user_id)

Read

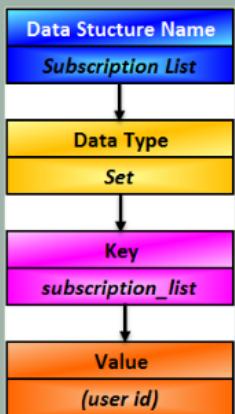
> SMEMBERS subscription_list

Delete

> SREM subscription_list (user_id)

Subscription List

Data Structure



CRUD Operations

Create

> `SADD subscription_list (user_id)`

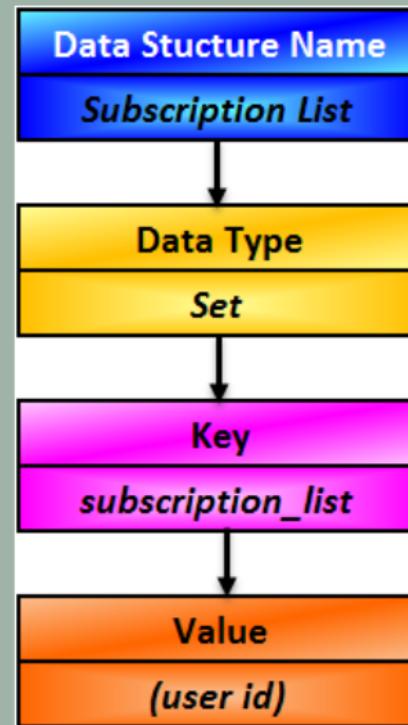
Read

> `SMEMBERS subscription_list`

Delete

> `SREM subscription_list (user_id)`

Data Structure



CRUD Operations

Create

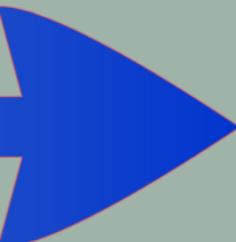
> SADD subscription_list (user_id)

Read

> SMEMBERS subscription_list

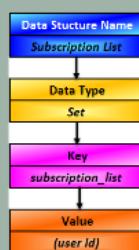
Delete

> SREM subscription_list (user_id)



Subscription List

Data Structure



CRUD Operations

Create

> SADD subscription_list (user_id)

Read

> SMEMBERS subscription_list

Delete

> SREM subscription_list (user_id)

User Favorite List

Data Structure



CRUD Operations

Create

```
HMSET favourite_list:(user id):(random number) estate_id (estate id) time_stamp (date:time) estate_latitude (estate latitude) estate_longitude (estate longitude) user_latitude (user latitude) user_longitude (user longitude)
```

Read

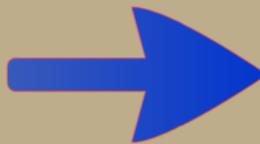
```
HGETALL favourite_list:(user id):(random number)
```

Delete

```
DEL favourite_list:(user id):(random number)
```

USER FAVORITE LIST

Data Structure



CRUD Operations

Create

```
HMSET favourite_list:(user id):(random number) estate_id (estate id) time_stamp (date:time) estate_latitude (estate latitude) estate_longitude (estate longitude) user_latitude (user latitude) user_longitude (user longitude)
```

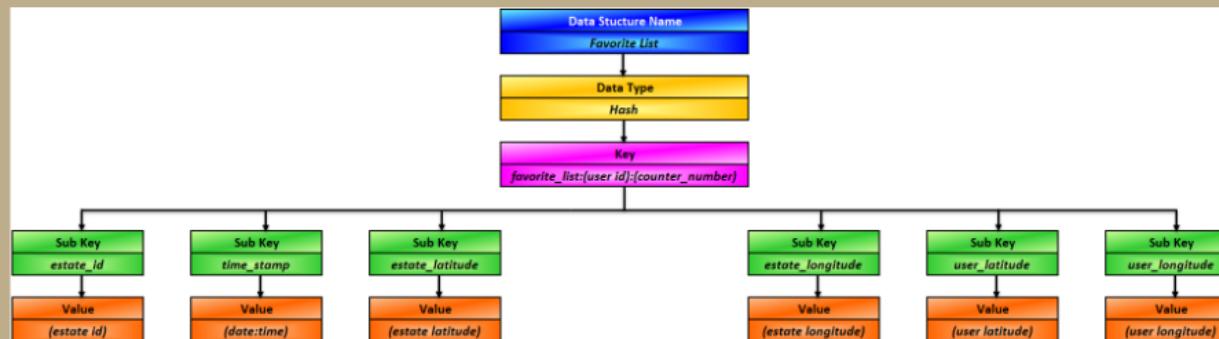
Read

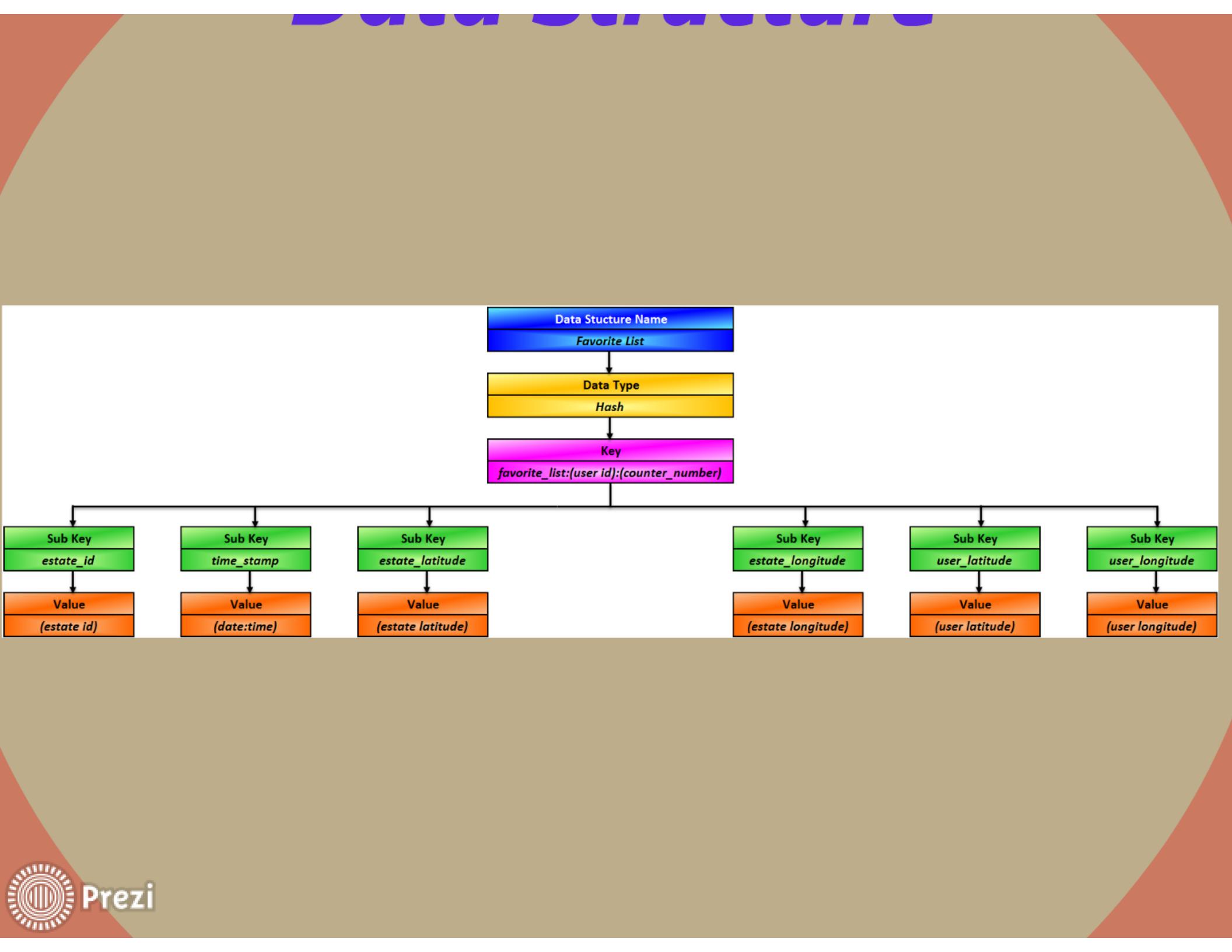
```
HGETALL favourite_list:(user id):(random number)
```

Delete

```
DEL favourite_list:(user id):(random number)
```

Data Structure





CRUD Operations

Create

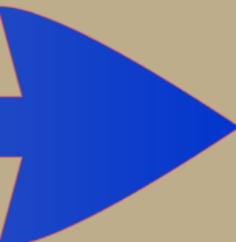
```
HMSET favourite_list:(user id):(random  
number) estate_id (estate id) time_stamp  
(date:time) estate_latitude (estate latitude)  
estate_longitude (estate longitude)  
user_latitude (user latitude) user_longitude  
(user longitude)
```

Read

```
HGETALL favourite_list:(user id):(random  
number)
```

Delete

```
DEL favourite_list:(user id):(random number)
```



User Favorite List

Data Structure



CRUD Operations

Create

```
HMSET favourite_list:(user id):(random number) estate_id (estate id) time_stamp (date:time) estate_latitude (estate latitude) estate_longitude (estate longitude) user_latitude (user latitude) user_longitude (user longitude)
```

Read

```
HGETALL favourite_list:(user id):(random number)
```

Delete

```
DEL favourite_list:(user id):(random number)
```

User Search History

Data Structure



CRUD Operations

Create

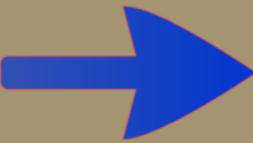
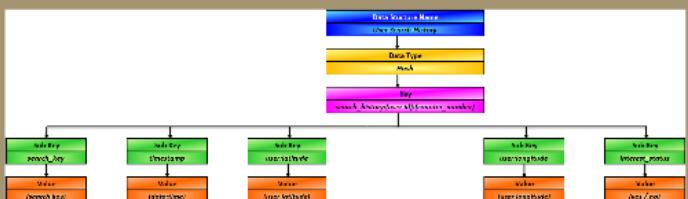
```
> HMSET search_history:(user_id):  
(random_number) search_key (search_key)  
timestamp (timestamp) user_latitude  
(user_latitude) user_longitude (user_longitude)  
interest_status (0/1)
```

Read

```
> HGETALL search_history:(user_id):  
(random_number)
```

User Search History

Data Structure



CRUD Operations

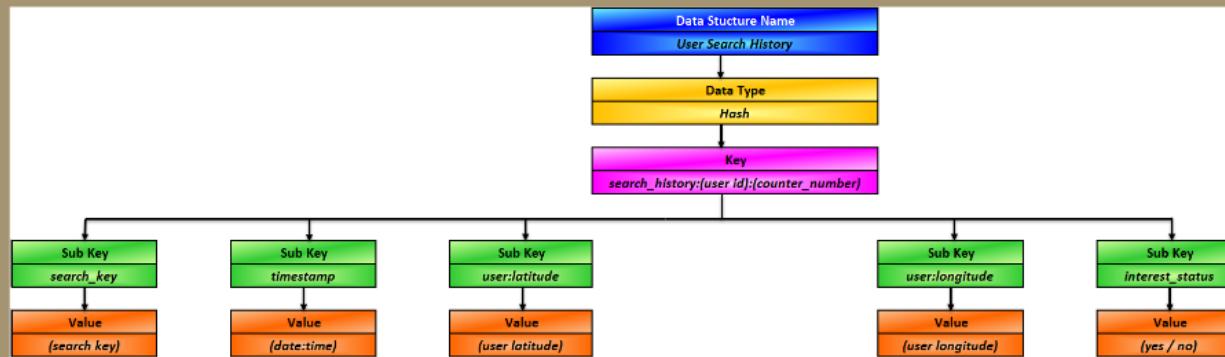
Create

```
> HMSET search_history:(user_id):  
(random_number) search_key (search_key)  
timestamp (timestamp) user_latitude  
(user_latitude) user_longitude (user_longitude)  
interest_status (0/1)
```

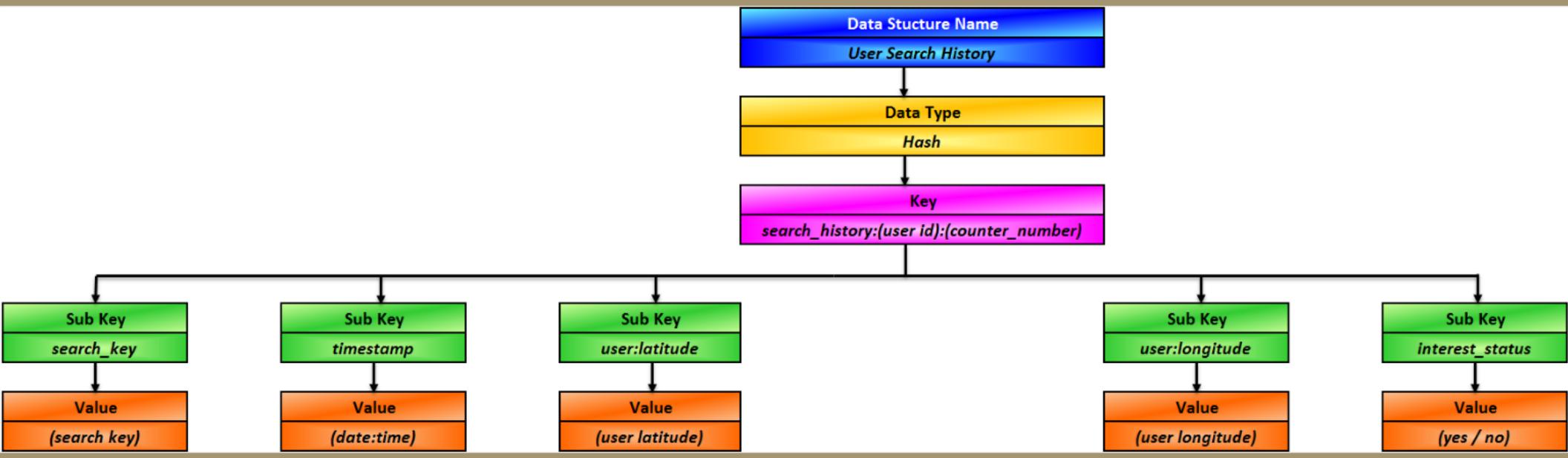
Read

```
> HGETALL search_history:(user_id):  
(random_number)
```

Data Structure



Data Structure



CRUD Operations

Create

```
> HMSET search_history:(user_id):  
  (random_number) search_key (search_key)  
  timestamp (timestamp) user_latitude  
  (user_latitude) user_longitude (user_longitude)  
  interest_status (0/1)
```

Read

```
> HGETALL search_history:(user_id):  
  (random_number)
```



User Search History

Data Structure



CRUD Operations

Create

```
> HMSET search_history:(user_id):  
(random_number) search_key (search_key)  
timestamp (timestamp) user_latitude  
(user_latitude) user_longitude (user_longitude)  
interest_status (0/1)
```

Read

```
> HGETALL search_history:(user_id):  
(random_number)
```

Trends

Data Structure



CRUD Operations

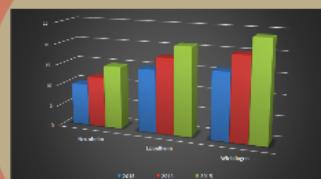
Create

```
> LPUSH estate_types (estate type)
> HMSET price_trend:(year):(estate_type)
  (address_id) (price)
```

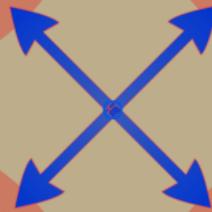
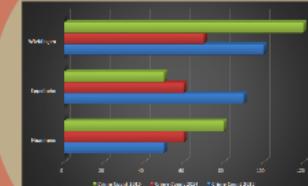
Read

```
> LRANGE estate_types 0 -1
> HGETALL price_trend:(year):(estate_type)
```

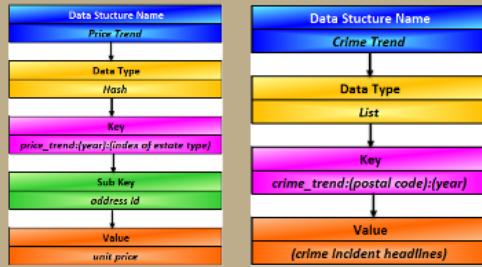
Price Trend



Crime Trend



Data Structure



CRUD Operations

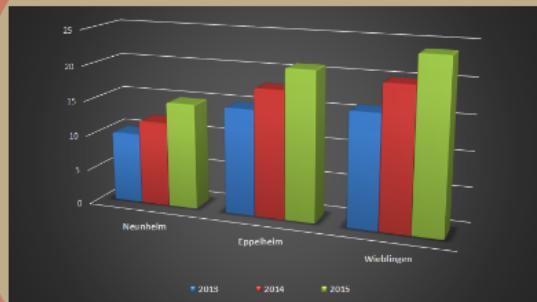
Create

> LPUSH estate_types (estate type)
> HMSET price_trend:(year):(estate_type) (address_id) (price)

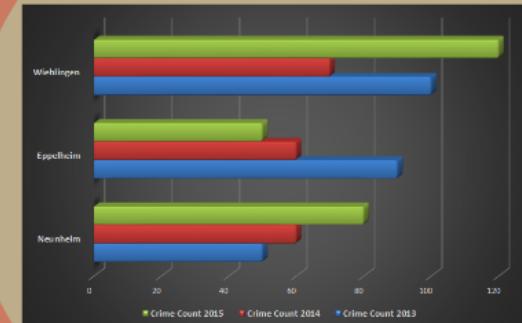
Read

> LRANGE estate_types 0 -1
> HGETALL price_trend:(year):(estate_type)

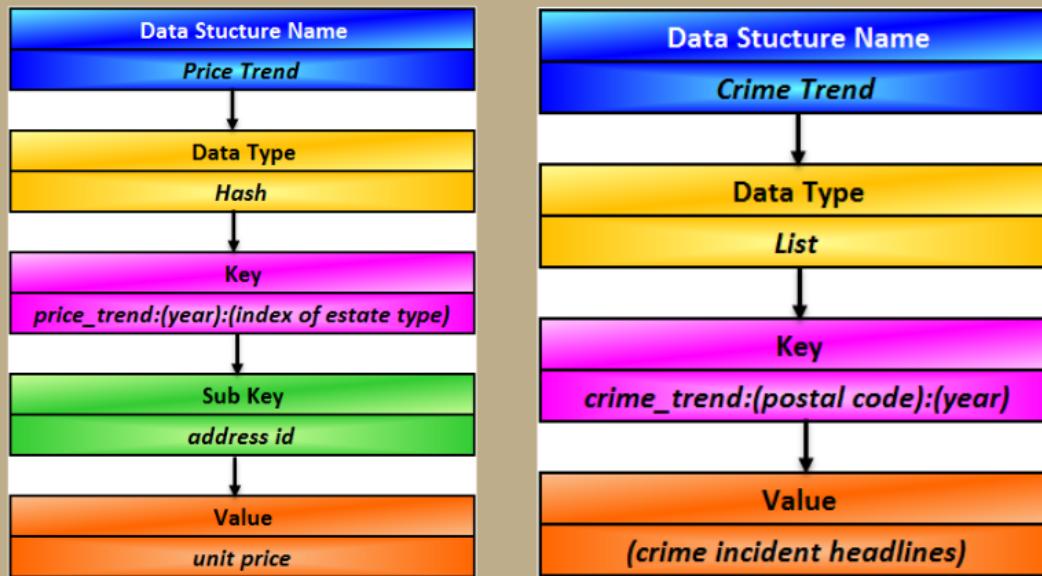
Price Trend

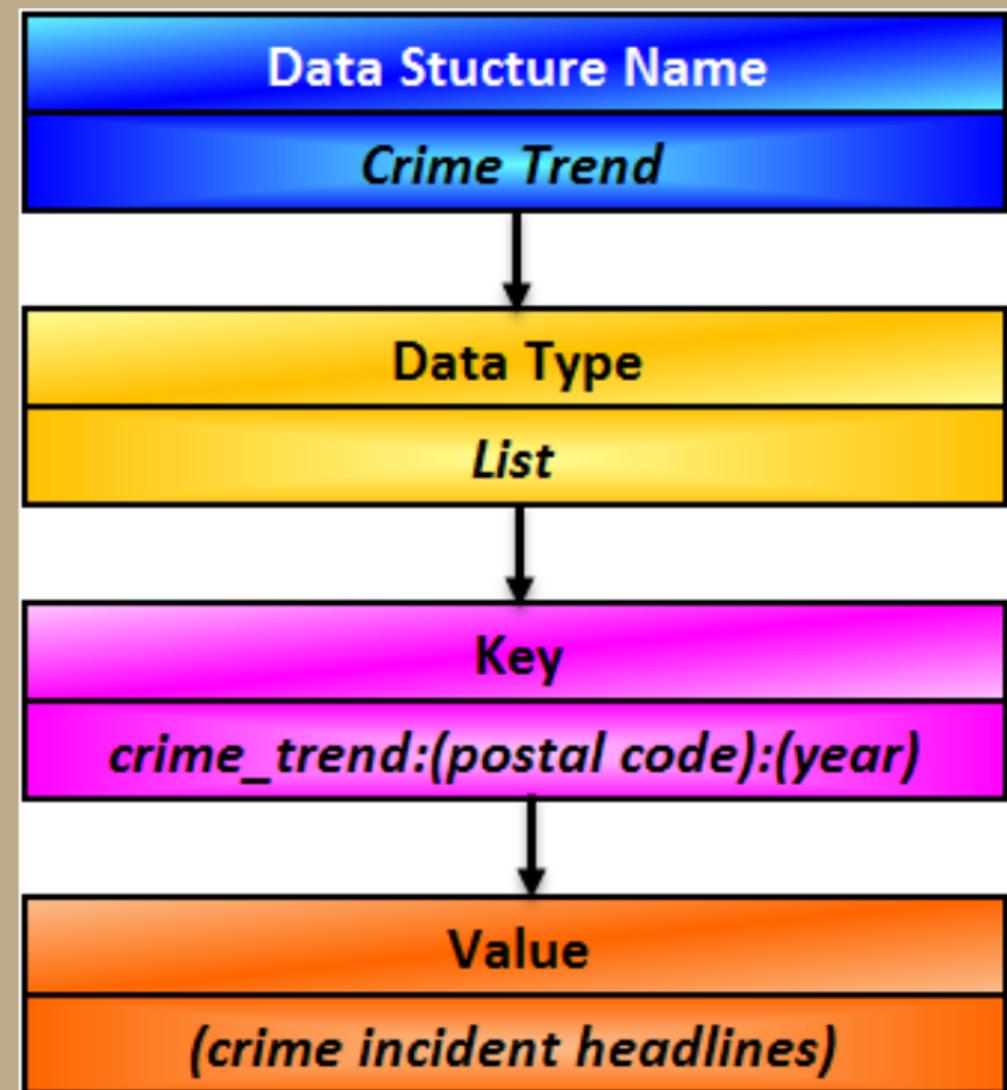
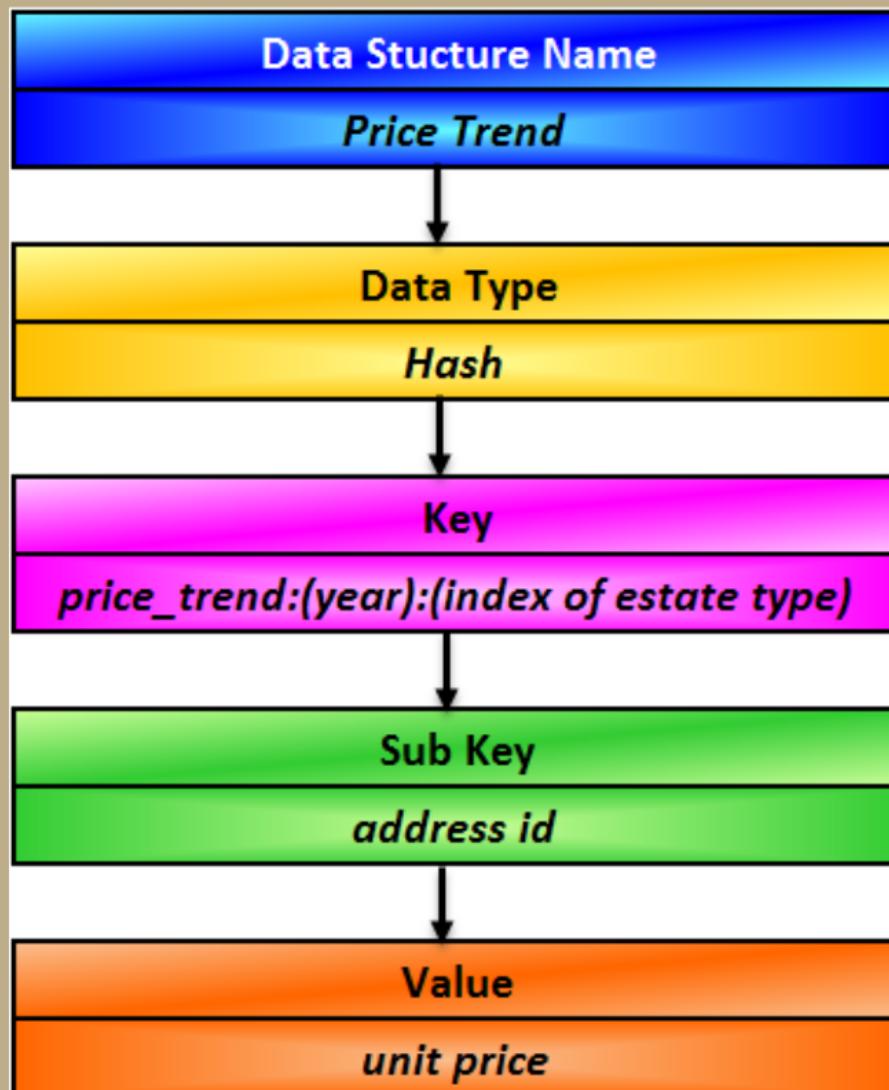


Crime Trend



Data Structure





CRUD Operations

Create

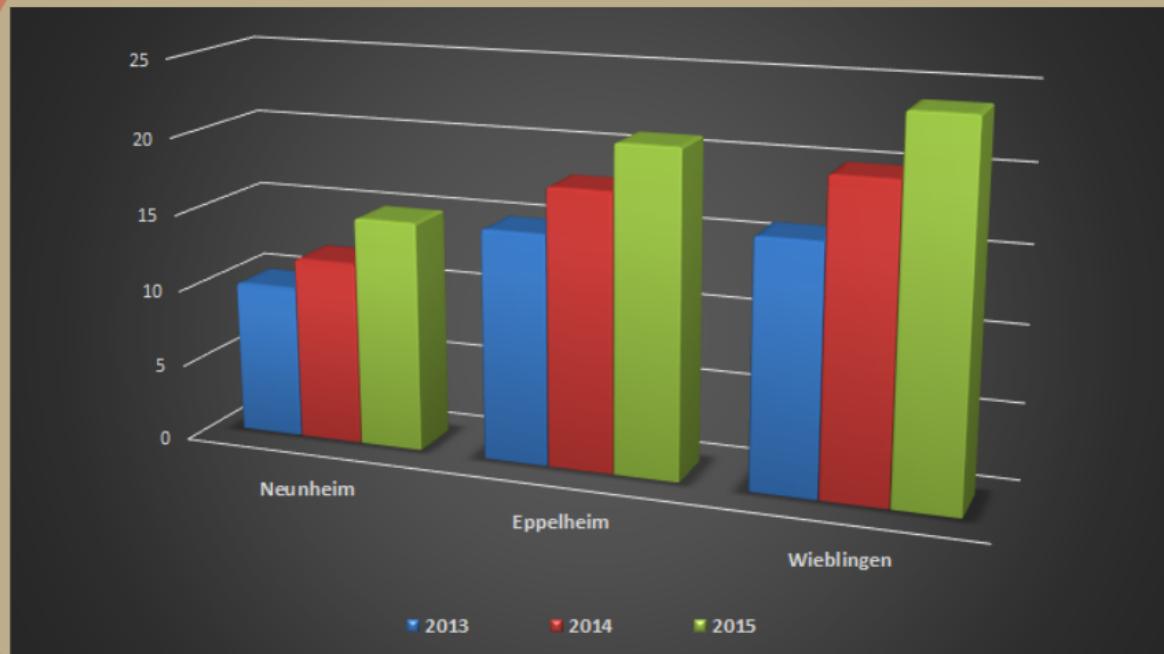
- > LPUSH estate_types (estate type)
- > HMSET price_trend:(year):(estate_type)
(address_id) (price)

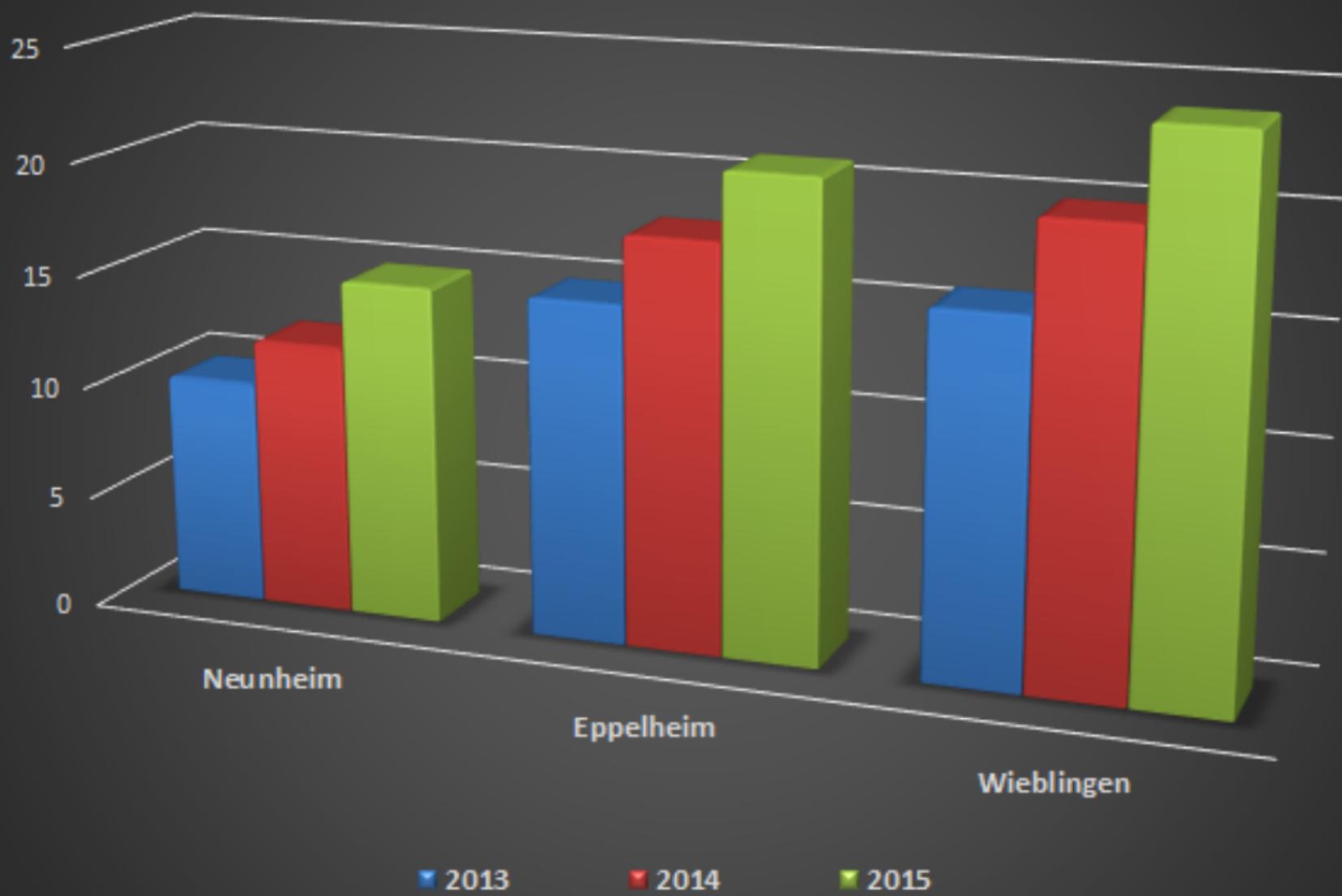
Read

- > LRANGE estate_types 0 -1
- > HGETALL price_trend:(year):(estate_type)

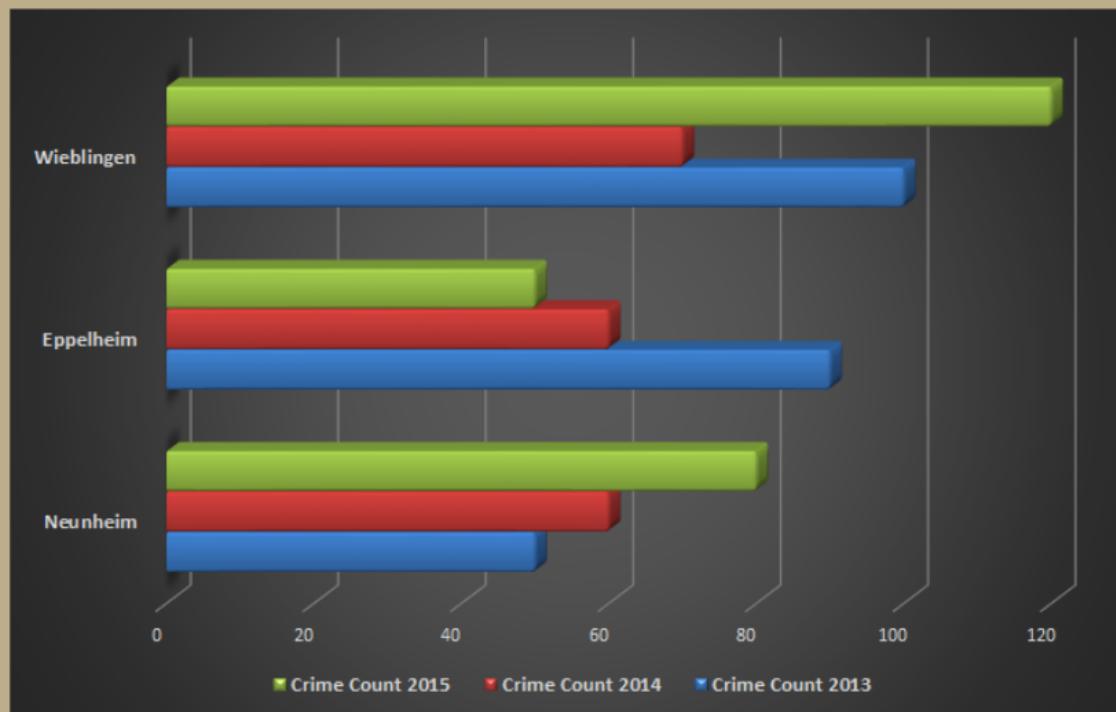


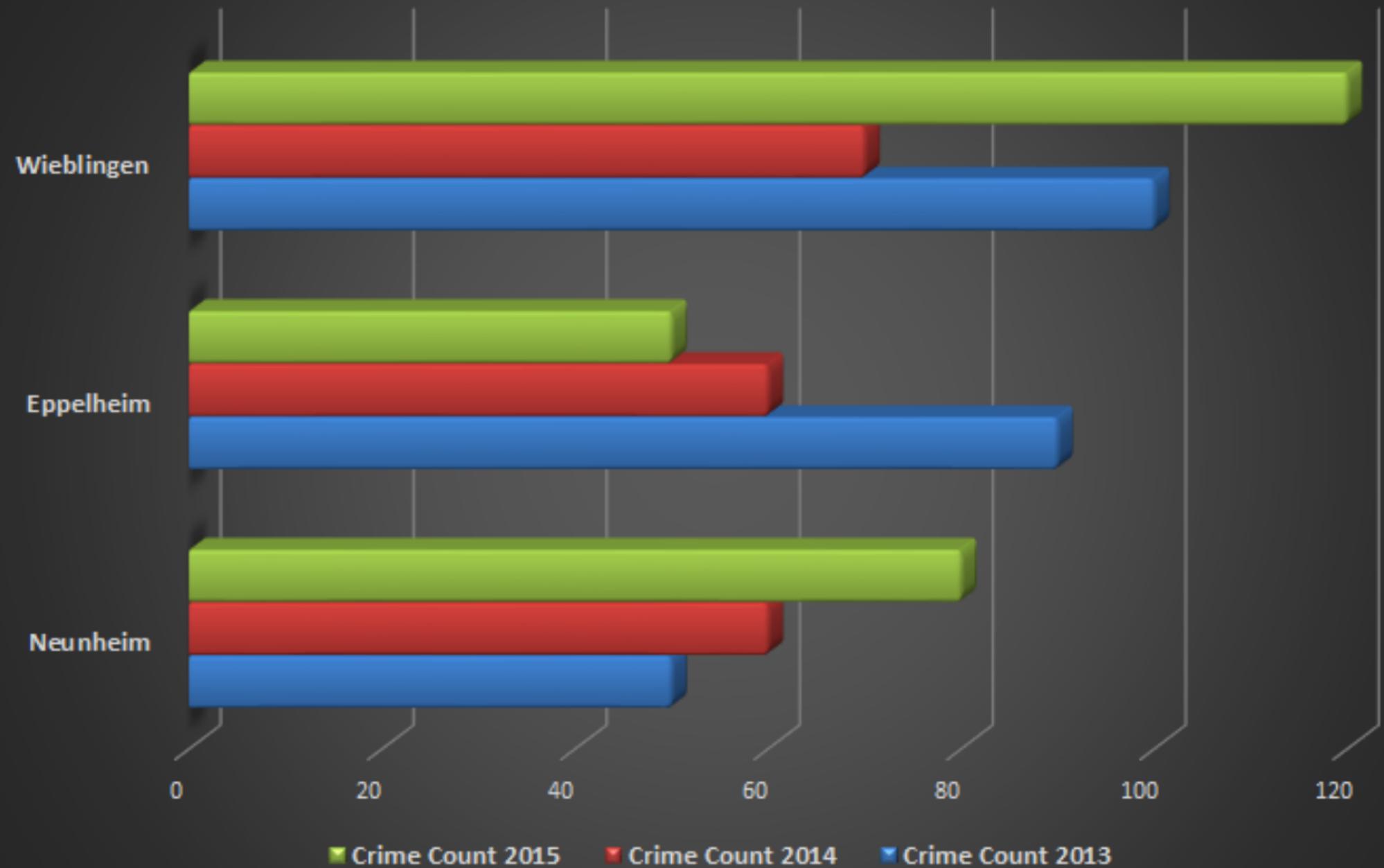
Price Trend





Crime Trend





Trends

Data Structure



CRUD Operations

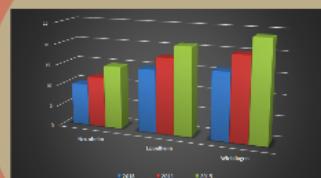
Create

```
> LPUSH estate_types (estate type)
> HMSET price_trend:(year):(estate_type)
  (address_id) (price)
```

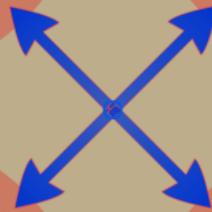
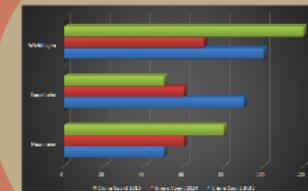
Read

```
> LRANGE estate_types 0 -1
> HGETALL price_trend:(year):(estate_type)
```

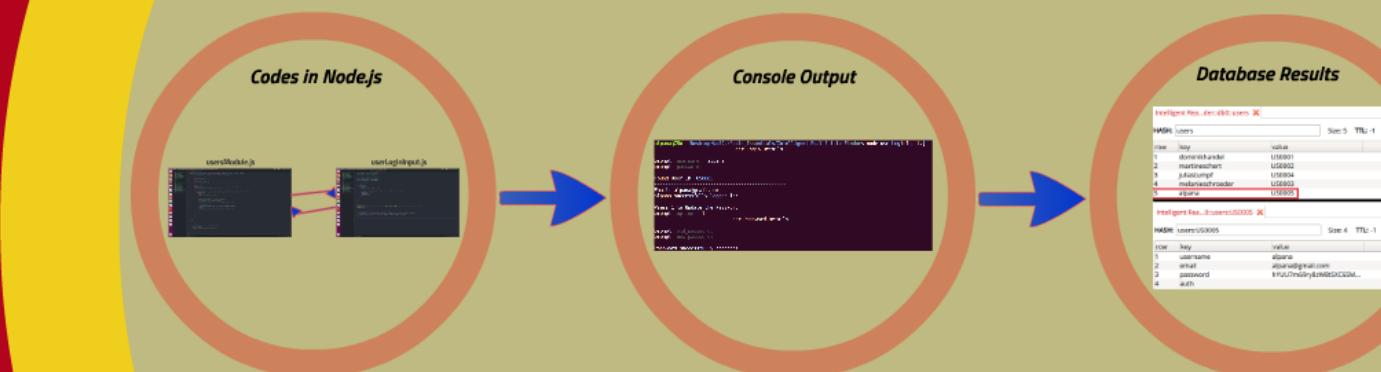
Price Trend



Crime Trend



User Login Feature Implementation



Implementation

Codes in Node.js

Two code editors are shown side-by-side. The left editor contains the code for the user module, and the right editor contains the user login input logic.

Console Output

A terminal window displays the execution of the userLoginInput.js script. It shows the user entering their password and the script outputting the user's details.

```
alpana@alpana-OptiPlex-5070:~/Desktop/NodeJS_Essentials/Intelligent Real Estate Platform$ node userLoginInput.js
User login: alpana
Password: alpana
User ID: 100005
Email: alpana@gmail.com
Access level: Super Admin
Press 1 to Update the Password
User Password Details
Old password: 
New password: 
Re-enter new password: 
Password successfully changed!
```

Database Results

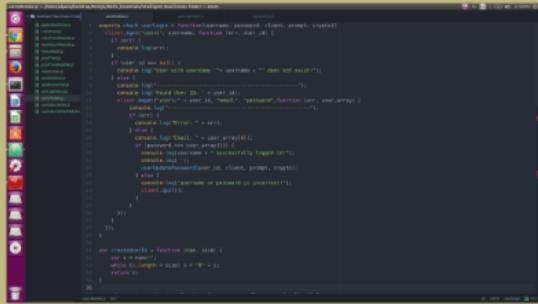
The Redis interface shows two tables: 'users' and 'userUS0005'. The 'users' table lists five users with their IDs. The 'userUS0005' table shows the details for the user with ID 100005.

HASH: users	Size: 5 TTL: -1
row key value	
1 dominikanhandel US0001	
2 martineschert US0002	
3 julieschumpf US0004	
4 melanieischoeder US0003	
5 alpana US0005	

HASH: userUS0005	Size: 4 TTL: -1
row key value	
1 username alpana	
2 email alpana@gmail.com	
3 password hYUU7m69ry8zWBLSXCB6SM...	
4 auth	

Codes in Node.js

usersModule.js



```
const bcrypt = require('bcryptjs');
const jwt = require('jsonwebtoken');
const User = require('../models/User');
const errorHandler = require('../utils/errorHandler');

// Create a new user
exports.createUser = async (req, res) => {
  const { name, email, password } = req.body;
  if (!name || !email || !password) {
    return res.status(400).json({ message: 'All fields are required' });
  }
  try {
    const user = await User.create({ name, email, password });
    const token = jwt.sign({ user }, process.env.JWT_SECRET);
    res.status(201).json({ user, token });
  } catch (err) {
    errorHandler(err, res);
  }
};

// Login a user
exports.loginUser = async (req, res) => {
  const { email, password } = req.body;
  if (!email || !password) {
    return res.status(400).json({ message: 'Email and password are required' });
  }
  try {
    const user = await User.findOne({ email });
    if (!user) {
      return res.status(401).json({ message: 'User not found' });
    }
    const isMatch = await bcrypt.compare(password, user.password);
    if (!isMatch) {
      return res.status(401).json({ message: 'Incorrect password' });
    }
    const token = jwt.sign({ user }, process.env.JWT_SECRET);
    res.status(200).json({ user, token });
  } catch (err) {
    errorHandler(err, res);
  }
};

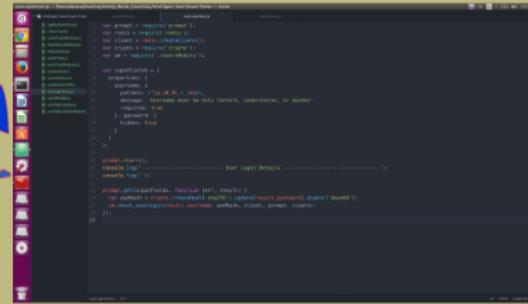
// Get all users
exports.getAllUsers = async (req, res) => {
  try {
    const users = await User.find();
    res.status(200).json(users);
  } catch (err) {
    errorHandler(err, res);
  }
};

// Get a single user
exports.getUser = async (req, res) => {
  const { id } = req.params;
  try {
    const user = await User.findById(id);
    if (!user) {
      return res.status(404).json({ message: 'User not found' });
    }
    res.status(200).json(user);
  } catch (err) {
    errorHandler(err, res);
  }
};

// Update a user
exports.updateUser = async (req, res) => {
  const { id } = req.params;
  const { name, email, password } = req.body;
  if (!id) {
    return res.status(400).json({ message: 'User ID is required' });
  }
  try {
    const user = await User.findByIdAndUpdate(id, { name, email, password });
    if (!user) {
      return res.status(404).json({ message: 'User not found' });
    }
    res.status(200).json(user);
  } catch (err) {
    errorHandler(err, res);
  }
};

// Delete a user
exports.deleteUser = async (req, res) => {
  const { id } = req.params;
  if (!id) {
    return res.status(400).json({ message: 'User ID is required' });
  }
  try {
    const user = await User.findByIdAndDelete(id);
    if (!user) {
      return res.status(404).json({ message: 'User not found' });
    }
    res.status(200).json({ message: 'User deleted successfully' });
  } catch (err) {
    errorHandler(err, res);
  }
};
```

userLoginInput.js



```
const express = require('express');
const router = express.Router();
const User = require('../models/User');
const jwt = require('jsonwebtoken');
const errorHandler = require('../utils/errorHandler');

// User login input
router.post('/login', async (req, res) => {
  const { email, password } = req.body;
  if (!email || !password) {
    return res.status(400).json({ message: 'Email and password are required' });
  }
  try {
    const user = await User.findOne({ email });
    if (!user) {
      return res.status(401).json({ message: 'User not found' });
    }
    const isMatch = await bcrypt.compare(password, user.password);
    if (!isMatch) {
      return res.status(401).json({ message: 'Incorrect password' });
    }
    const token = jwt.sign({ user }, process.env.JWT_SECRET);
    res.status(200).json({ user, token });
  } catch (err) {
    errorHandler(err, res);
  }
});

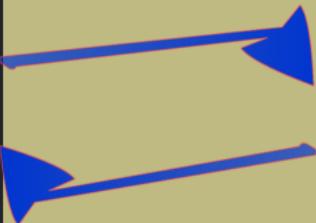
module.exports = router;
```

usersModule.js

```
userModule.js 1 exports.check_userLogin = function(username, password, client, prompt, crypto){  
2   client.get("users", username, function (err, user_id) {  
3     if (err) {  
4       console.log(err);  
5     }  
6     if (user_id === null) {  
7       console.log("User with username '" + username + "' does not exist");  
8     } else {  
9       console.log("-----");  
10      console.log("Found User ID: " + user_id);  
11      client.get("users" + user_id, "email", "password",function (err, user_array) {  
12        if (err) {  
13          console.log("Error: " + err);  
14        } else {  
15          console.log("Email: " + user_array[0]);  
16          if (password === user_array[0]) {  
17            console.log(username + " successfully logged in!");  
18            console.log("-----");  
19            userUpdatePassword(user_id, client, prompt, crypto);  
20          } else {  
21            console.log("username or password is incorrect");  
22          }  
23        }  
24      });  
25    }  
26  });  
27 }  
28  
29 var createdUserId = function (num, size) {  
30   var s = num+"";  
31   while (s.length < size) s = "0" + s;  
32   return s;  
33 }  
34  
35 }  
36  
37 module.exports = usersModule;
```

userLoginInput.js

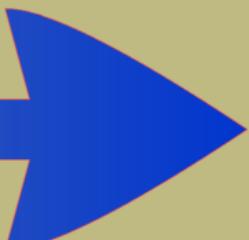
```
userLoginInput.js 1 var prompt = require('prompt');  
2 var redis = require('redis');  
3 var client = redis.createClient();  
4 var crypto = require('crypto');  
5 var sm = require('./usersModule');  
6  
7 var inputFields = {  
8   properties: {  
9     username: {  
10       pattern: /^[a-zA-Z\_\.\-]+$/i,  
11       message: "username must be only letters, underscores, or dashes",  
12       required: true  
13     },  
14     password: {  
15       hidden: true  
16     }  
17   }  
18 };  
19 prompt.start();  
20 console.log("----- User Login Details -----");  
21 console.log("");  
22  
23 prompt.getInputFields(function (err, result) {  
24   var pwHash = crypto.createHash('sha256').update(result.password).digest('base64');  
25   sm.check_userLogin(result.username, pwHash, client, prompt, crypto);  
26 });  
27  
28 }  
29  
30 module.exports = userLoginInput;
```



Console Output

```
alpana@Z50:~/Desktop/NoSQL/Redis_Essentials/Intelligent Real Estate Finder$ node userLoginInput.js
----- User Login Details -----
prompt: username: alpana
prompt: password:
-----
Found User ID: US0005
-----
Email: alpana@gmail.com
alpana successfully logged in!

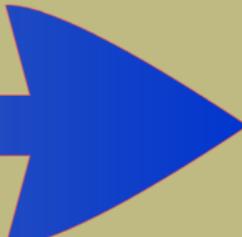
Press 1 to Update the Password
prompt: option: 1
----- User Password Details -----
prompt: old_password:
prompt: new_password:
-----
Password successfully changed!
```



```
alpana@Z50:~/Desktop/NoSQL/Redis_Essentials/Intelligent Real Estate Finder$ node userLoginInput.js
----- User Login Details -----
prompt: username: alpana
prompt: password:
-----
Found User ID: US0005
-----
Email: alpana@gmail.com
alpana successfully logged in!

Press 1 to Update the Password
prompt: option: 1
----- User Password Details -----
prompt: old_password:
prompt: new_password:
-----
Password successfully changed!
```

Database Results

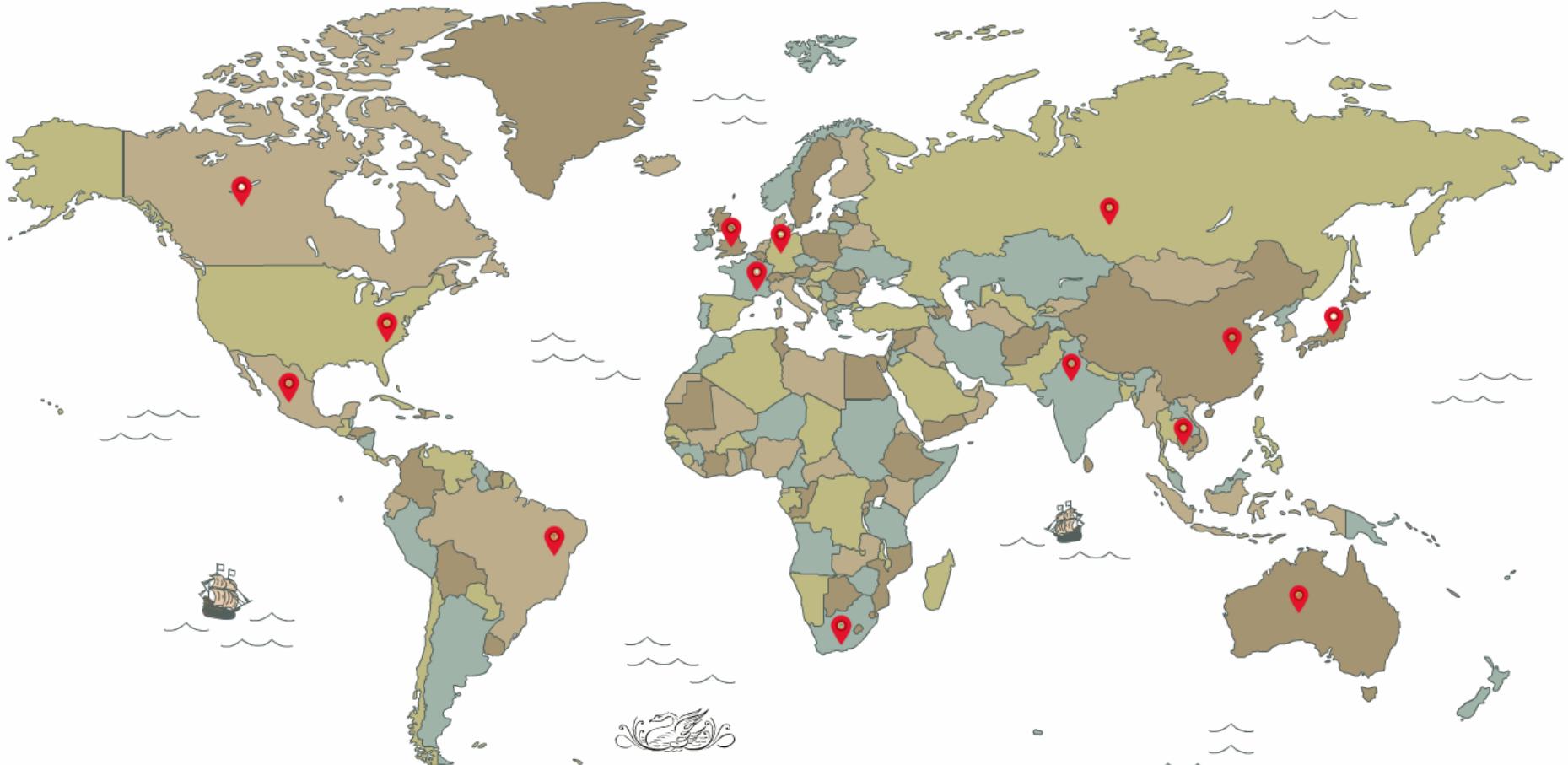


Intelligent Reader::db0::users X		
HASH: users		Size: 5 TTL: -1
row	key	value
1	dominikhandel	US0001
2	martineschert	US0002
3	juliastumpf	US0004
4	melanieschroeder	US0003
5	alpana	US0005

Intelligent Reader::db0::users:US0005 X		
HASH: users:US0005		Size: 4 TTL: -1
row	key	value
1	username	alpana
2	email	alpana@gmail.com
3	password	hYUU7m69ry8zW8tSXC6SM...
4	auth	

Ideas on extending the application

- Detail Analysis of Price Trend and Crime Trend
- For example, using PUB / SUB functionality of Redis database, to broadcast live updates and other information to all the users who are currently logged in.



Key Value Store Database: Redis

Information Systems

Master Course: Advanced Information Sytems 2016

Lecturer: Prof. Dr. Barbara Sprick, PhD. Ajinkya Prabhune