AS101 Lab Exercises


Lab: Key-value Operations

Objective

After successful completion of this Lab module you will have:

- Connected to a cluster
- Write and read records using simple and complex values
- Used advanced key-value techniques

Lab Overview

The lab exercises add functionality to a simple Twitter- like console application (tweetaspike) using Aerospike as the database.

In this Lab, we will focus on key-value operations and techniques. You will add code to:

- Create users and Tweets
- Read all the Tweets for a user
- Use an advance Key-value feature

The exercises shell is located in your cloned GitHub directory

`~/exercises/Key-valueOperations/<language>`

Make sure you have your server up and you know its IP address

In your cloned or downloaded repository, you will find the following directories:

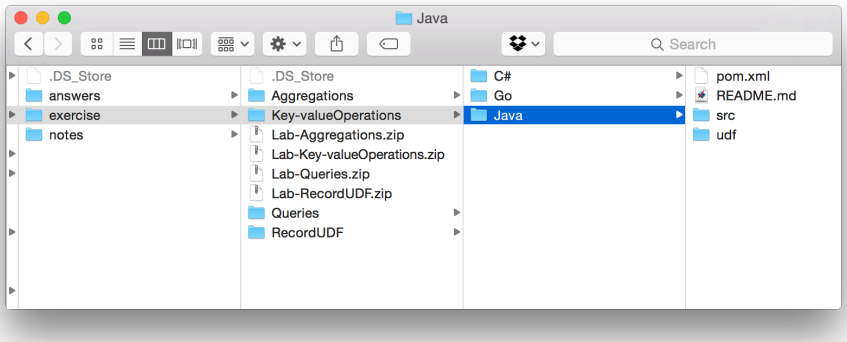
- Answers
- Exercise
- Notes

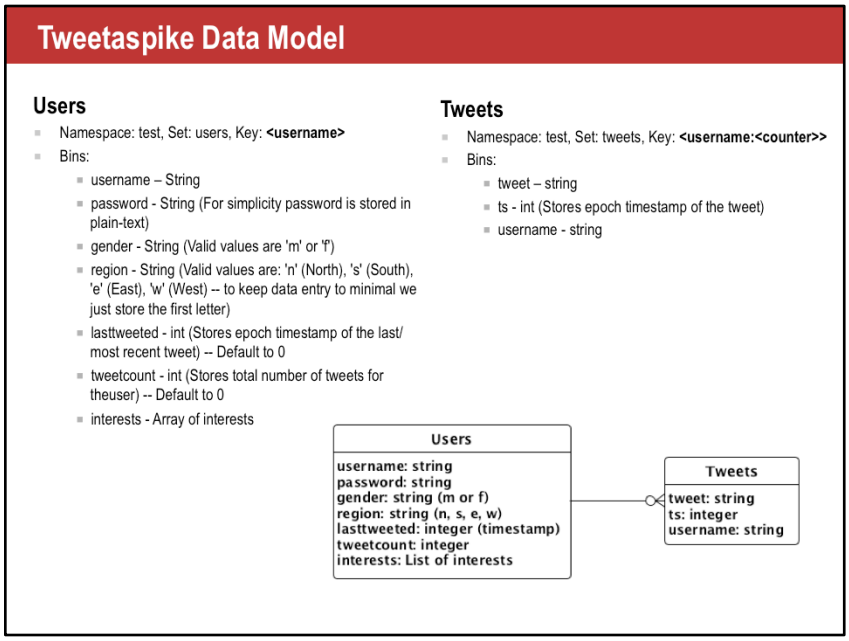
In the exercise directory, select the subdirectory for your programming language:

- C#
- Java
- Go
- PHP
- Ruby
- Node.js
- Python

The exercises for this module are in the Key-valueOperations directory and your will find a Project/ Solution/Codebase that is partly complete. Your tasks is to complete the code as outlined in each exercise.

Make sure you have your server up and you know its IP address





Users

Namespace: test, Set: users, Key: <username>

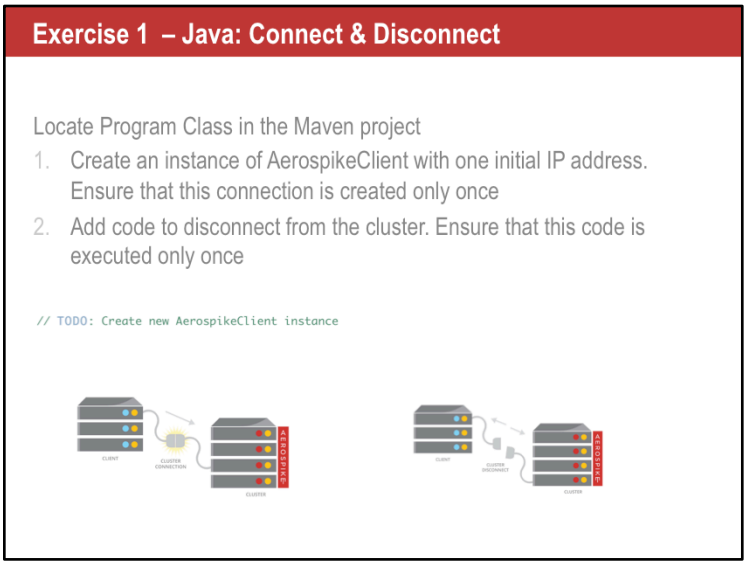
Bin name	Type	Comment
username	String	
password	String	For simplicity password is stored in plain text
region	String	Valid values are: 'n' (North), 's' (South), 'e' (East), 'w' (West) to keep data entry to minimal we just store the first le] er
las] weeted	Integer	Stores epoch Nmestamp of the last/most recent tweet Default to 0
tweetcount	Integer	Stores total number of tweets for the user – Default 0
Interests	List	A list of interests

Tweets

Namespace: test, Set: tweets, Key: <username:<counter>>

Bin name	Type	Comment
tweet	String	Tweet text
ts	Integer	Stores epoch Nmestamp of the tweet
username	String	User name of the tweeter


Java Exercises



In this exercise you will connect to a Cluster by creating an AerospikeClient instance, passing a single IP address and port to the constructor. The IP address and port should be to a node in your own cluster.

Ensure that you only create one client instance at the start of the program. The AerospikeClient is thread safe and creates a pool of worker threads, this means you DO NOT need to create your own connection or thread pool.

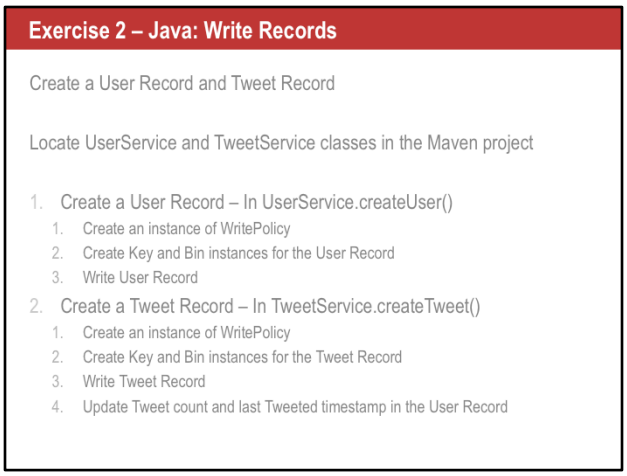
1. In the constructor for the class Program, add code similar to this;

```
// Establish a connection to Aerospike cluster
this.client = new AerospikeClient("192.168.1.15", 3000);
```

Make sure you have your server up and you know its IP address

2. At the end of method work(), add code, similar to this, to disconnect from the cluster. This should only be done once. After close() is called, the client instance cannot be used.

```
if (client != null && client.isConnected()) {
     // Close Aerospike server connection
     client.close();
}
```



Create a User Record. In UserService.createUser(), add code similar to this:

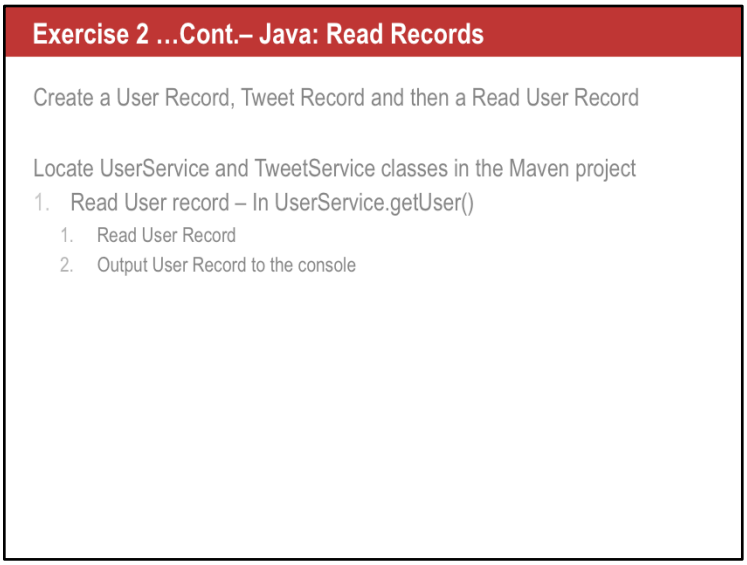
1. Create WritePolicy
2. Create ey and Bin instances
3. Write a user record using the Key and Bins

```
// Write record
WritePolicy wPolicy = new WritePolicy();
wPolicy.recordExistsAction = RecordExistsAction.UPDATE;

Key key = new Key("test", "users", username);
Bin bin1 = new Bin("username", username);
Bin bin2 = new Bin("password", password);
Bin bin3 = new Bin("gender", gender);
Bin bin4 = new Bin("region", region);
Bin bin5 = new Bin("lasttweeted", 0);
Bin bin6 = new Bin("tweetcount", 0);
Bin bin7 = new Bin("interests",
           Arrays.asList(interests.split(",")));

client.put(wPolicy, key, bin1, bin2, bin3, bin4,
                         bin5, bin6, bin7);
```

Create a Tweet Record. In TweetService.createTweet(), add code similar to this:

1. Read user record
2. Create WritePolicy
3. Create ey and Bin instances

```
// Check if username exists
userKey = new Key("test", "users", username);
userRecord = client.get(null, userKey);

WritePolicy wPolicy = new WritePolicy();
wPolicy.recordExistsAction = RecordExistsAction.UPDATE;

tweetKey = new Key("test", "tweets", username + ":"
                                   + nextTweetCount);
Bin bin1 = new Bin("tweet", tweet);
Bin bin2 = new Bin("ts", ts);
Bin bin3 = new Bin("username", username);

client.put(wPolicy, tweetKey, bin1, bin2, bin3);
console.printf("\nINFO: Tweet record created!\n");

// Update tweet count and last tweet'd timestamp in the user
// record
long ts = getTimeStamp();
updateUser(client, userKey, wPolicy, ts, nextTweetCount);
```



Read a User Record. In UserService.getUser(), add code, similar to this, to:

1. Read a User record
2. Output the User record to the console

```
// Check if username exists
userKey = new Key("test", "users", username);
userRecord = client.get(null, userKey);
if (userRecord != null) {
      console.printf("\nINFO: User record read successfully! Here are the details:\n");
      console.printf("username:    " + userRecord.getValue("username") + "\n");
      console.printf("password:    " + userRecord.getValue("password") + "\n");
      console.printf("gender:      " + userRecord.getValue("gender") + "\n");
      console.printf("region:      " + userRecord.getValue("region") + "\n");
      console.printf("tweetcount:  " + userRecord.getValue("tweetcount") + "\n");
      console.printf("interests:   " + userRecord.getValue("interests") + "\n");
} else {
      console.printf("ERROR: User record not found!\n");
}
```

Exercise 3 – Java: Batch Read

Batch Read Tweets for a given user

Locate UserService class in the Maven project

1. In UserService.batchGetUserTweets()
   1. Read User Record
   2. Determine how many Tweets the user has
   3. Create an array of Tweet Key instances -- keys[tweetCount]
   4. Initiate Batch Read operation
   5. Output Tweets to the console

Read all the tweets for a given user. In UserService.batchGetUserTweets(), add code similar to this:

1. Read a user record

```
userKey = new Key("test", "users", username);
userRecord = client.get(null, userKey);
```

2. Get the tweet count

```
// Get how many tweets the user has
int tweetCount = (Integer) userRecord.getValue("tweetcount");
```

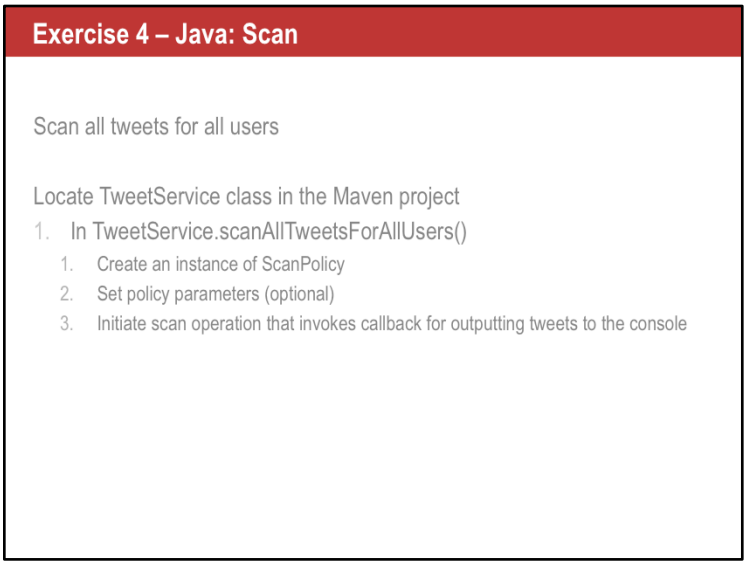
3. Create a "list" of tweet keys

```
// Create an array of keys so we can initiate batch read
// operation
Key[] keys = new Key[tweetCount];
for (int i = 0; i < keys.length; i++) {
     keys[i] = new Key("test", "tweets",
                (username + ":" + (i + 1)));
}
console.printf("\nHere's " + username + "'s tweet(s):\n");
```

4. Perform a Batch operation to read all the tweets

5. Then print out the tweets

```
// Initiate batch read operation
if (keys.length > 0){
     Record[] records = client.get(new Policy(), keys);
     for (int j = 0; j < records.length; j++) {
          console.printf(records[j].getValue("tweet").toString() + "\n");
     }
}
```



Scan all the tweets for all users – warning – there could be a large result set.

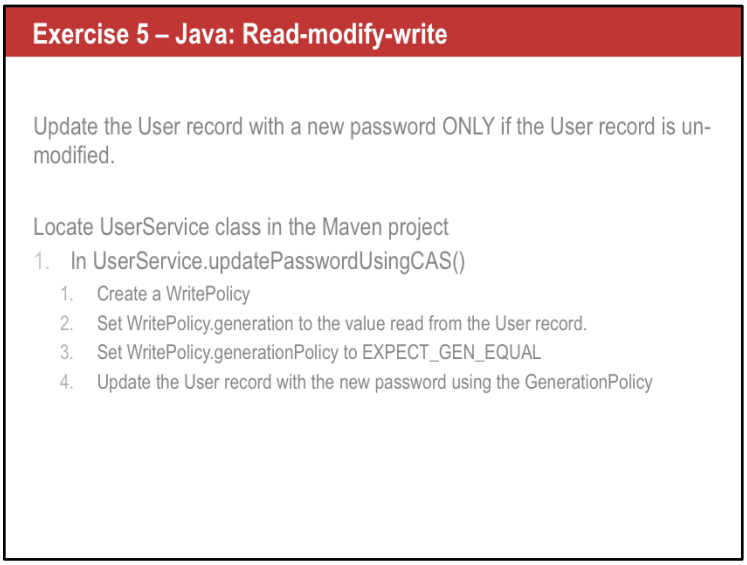
In the TweetService.scanAllTweetsForAllUsers(), add code similar to this:

1. Create ScanPolicy
2. Set policy parameters

```
// Java Scan
ScanPolicy policy = new ScanPolicy();
policy.concurrentNodes = true;
policy.priority = Priority.LOW;
policy.includeBinData = true;
```

3. Perform a Scan operation and process the results

```
client.scanAll(policy, "test", "tweets", new ScanCallback() {

     public void scanCallback(Key key, Record record)
                                   throws AerospikeException {

     console.printf(record.getValue("tweet") + "\n");

     }
}, "tweet");
```

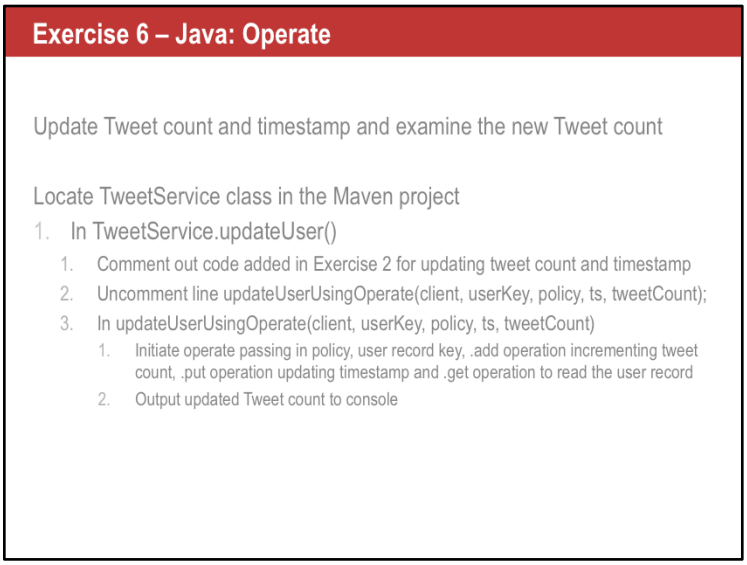


Update the User record with a new password ONLY if the User record is un-modified

In UserService.updatePasswordUsingCAS(), add code similar to this:

1. Create WritePolicy
2. Set WritePolicy.generaon to the value read from the User record.
3. Set WritePolicy.generaonPolicy to EXPECT_GEN_EQUAL
4. Update the User record with the new password using the GeneraonPolicy

```
// Check if username exists
userKey = new Key("test", "users", username);
userRecord = client.get(null, userKey);
if (userRecord != null)
{
      // Get new password
      String password;
      console.printf("Enter new password for " + username + ":");
      password = console.readLine();

      WritePolicy writePolicy = new WritePolicy();
      // record generation
      writePolicy.generation = userRecord.generation;
      writePolicy.generationPolicy = GenerationPolicy.EXPECT_GEN_EQUAL;
      // password Bin
      passwordBin = new Bin("password", Value.get(password));
      client.put(writePolicy, userKey, passwordBin);

      console.printf("\nINFO: The password has been set to: " + password);
}
```



Aerospike can perform multiple operations on a record in one transaction. Update the tweet count and timestamp in a user record and read the new tweet count.

In TweetService.updateUser()

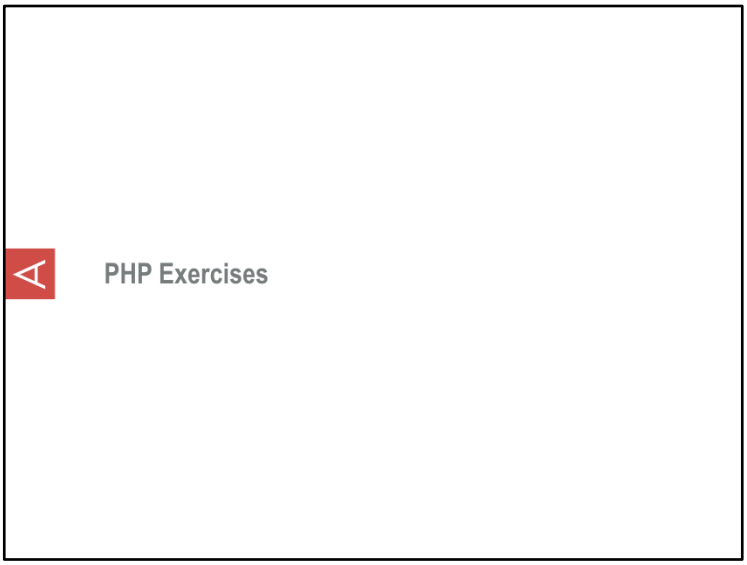
1. Comment out the code added in exercise 2

2. Uncomment the line:

```
// TODO: Update tweet count and last tweeted timestamp in the user record using Operate
// Exercise 6
// updateUserUsingOperate(client, userKey, policy, ts);
```

3. In TweetService.updateUserUsingOperate() add code similar to this:

```
Record record = client.operate(policy, userKey,
     Operation.add(new Bin("tweetcount", 1)),
     Operation.put(new Bin("lasttweeted", ts)),
     Operation.get());
```


PHP Exercises

Exercise 1 – PHP: Connect & Disconnect

Locate Program Class in PHP project

1. Create an instance of Aerospike with one initial IP address. Ensure that this connection is created only once
2. Add code to disconnect from the cluster. Ensure that this code is executed only once

```
// @todo create a config array describing the Aerospike cluster
// @todo: create an instance of Aerospike named $client
```

```
// @todo close the connection to the Aerospike cluster
```

In this exercise you will connect to a Cluster by creating an Aerospike instance, passing a $config structure to the constructor, that has a single IP address and port. These should be to a node in your own cluster.

Ensure that you only create one client instance at the start of the program. The Aerospike is thread safe and creates a pool of worker threads, this means you DO NOT need to create your own connection or thread pool.

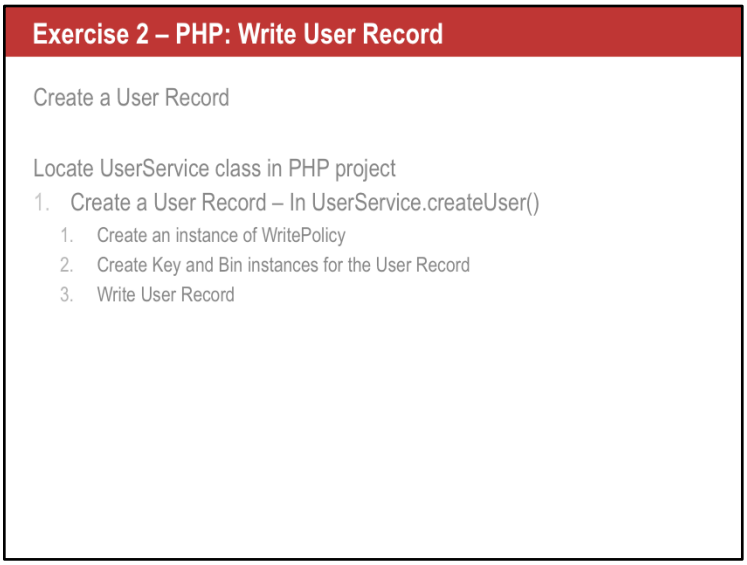
1. In the constructor for the class Program, add code similar to this;

```
$config = array("hosts" => array(array("addr" => $HOST_ADDR,
                "port" => $HOST_PORT)));
$client = new Aerospike($config, false);
if (!$client->isConnected()) {
    echo standard_fail($client);
    echo colorize("Connection to Aerospike cluster failed!
Please check the server settings and try again!\n", 'red',
true);
    exit(2);
}
```

Make sure you have your server up and you know its IP address

2. At the end of the program, add code, similar to this, to disconnect from the cluster. This should only be done once. After close() is called, the client instance cannot be used.

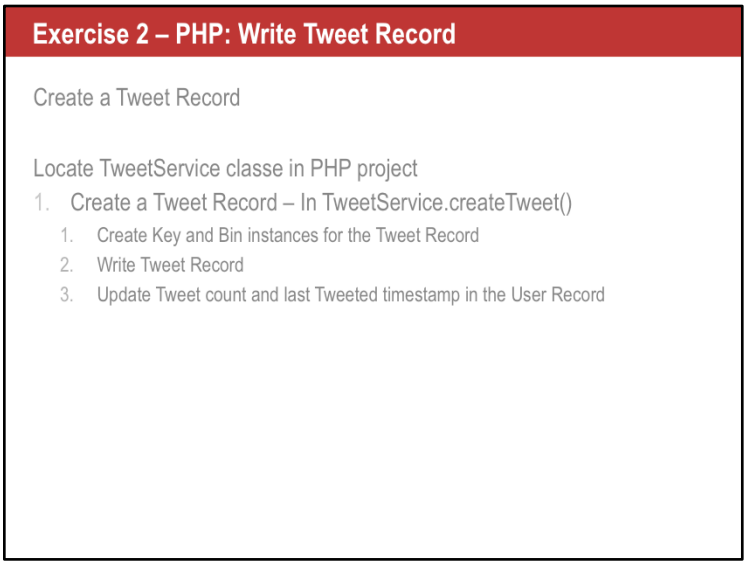
```
$client->close();
```



Create a User Record. In UserService.createUser(), add code similar to this:

1. Create an Array of Bin values from the user input
2. Create a Key
3. Write a user record using the Key and Bins
4. Check result code for errors

```php
// Get username
echo colorize("Enter username (or hit Return to skip): ");
$username = trim(readline());
if ($username == '') return;
$bins = array('username' => $username);
// Get password
echo colorize("Enter password for $username: ");
$bins['password'] = trim(readline());
// Get gender
echo colorize("Select gender (f or m) for $username: ");
$bins['gender'] = substr(trim(readline()), 0, 1);
// Get region
echo colorize("Select region (north, south, east or west) for
$username: ");
$bins['region'] = substr(trim(readline()), 0, 1);
// Get interests
echo colorize("Enter comma-separated interests for  $username:
");
$bins['interests'] = explode(',', trim(readline()));
echo colorize("Creating user record >", 'black', true);
$key = $this->getUserKey($username);
if ($this->annotate) display_code(__FILE__, __LINE__, 6);
$status = $this->client->put($key, $bins);
if ($status !== Aerospike::OK) {
    // throwing an \Aerospike\Training\Exception
    echo fail();
    throw new Exception($this->client, "Failed to create user
$username");
}
```



Create a Tweet Record. In TweetService.createTweet(), add code similar to this:

1. Read user record

```
// Check if username exists
$record = $this->getUser($username);
$ubins = $record['bins'];
$tweet_count = isset($ubins['tweetcount']) ? $ubins['tweetcount'] : 0;
$tweet_count++;
```

2. Create ey and Bin instances

```
$bins = array();
// Get a tweet
echo colorize("Enter tweet for $username: ");
$bins['tweet'] = trim(readline());
$bins['ts'] = time() * 1000;
$bins['username'] = $username;
```

3. Write a tweet record using the Key and Bins

```
echo colorize("Creating tweet record >", 'black', true);
$key = $this->getTweetKey($username, $tweet_count);
$status = $this->client->put($key, $bins);
if ($status !== Aerospike::OK) {
    // throwing an \Aerospike\Training\Exception
    echo fail();
    throw new Exception($this->client, "Failed to create the tweet");
}
```

4. Update the user record with tweet count

```
echo success();
return $this->updateUser($username, $bins['ts'], $tweet_count);
```

Exercise 2 PHP: Read Records

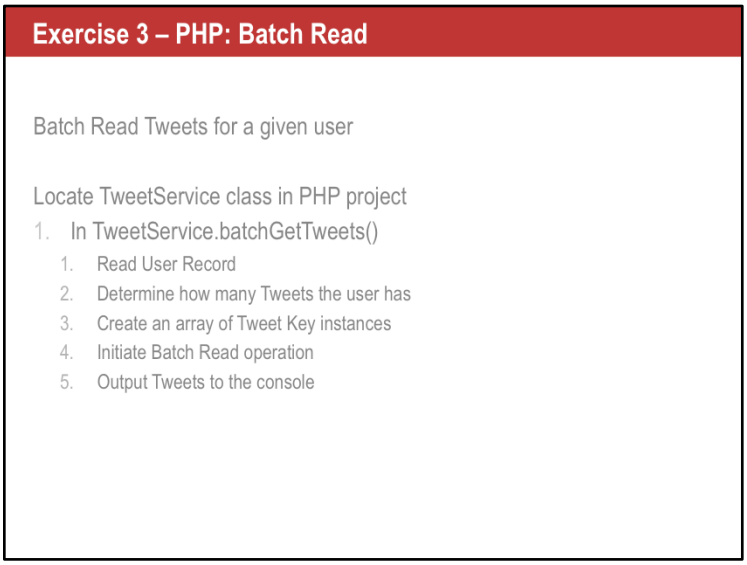
Read User Record

Locate BaseService class in PHP project

1. Read User record – In BaseService.getUser()
    1. Read User Record

Read a User Record. In BaseService.getUser(), add code, similar to this, to:

```
if ($username == '') {
    // Get username
    echo colorize("Enter username (or hit Return to skip): ");
    $username = trim(readline());
}
if ($username != '') {
    $key = $this->getUserKey($username);
    if ($this->annotate) display_code(__FILE__, __LINE__, 6);
    $status = $this->client->get($key, $record);
    if ($status !== Aerospike::OK) {
// throwing an \Aerospike\Training\Exception
throw new Exception($this->client, "Failed to get the user
$username");
    }
    return $record;
} else {
    throw new \Exception("Invalid input provided for username in
UserService::getUser()");
}
```



Read all the tweets for a given user. In TweetService.batchGetTweets(), add code similar to this:

1. Read a user record
2. Get the tweet count
3. Create a "list" of tweet keys

```
$record = $this->getUser($username, array('tweetcount'));
$tweet_count = intval($record['bins']['tweetcount']);
if ($this->annotate) display_code(__FILE__, __LINE__, 4);
$keys = array();
for ($i = 1; $i <= $tweet_count; $i++) {
    $keys[] = $this->getTweetKey($username, $i);
}
```

4. Perform a Batch operation to read all the tweets

```
echo colorize("Batch-reading the user's tweets >", 'black',
true);
if ($this->annotate) display_code(__FILE__, __LINE__, 6);
$status = $this->client->getMany($keys, $records);
if ($status !== Aerospike::OK) {
    echo fail();
    // throwing an \Aerospike\Training\Exception
    throw new Exception($this->client, "Failed to batch-read the
tweets for $username");
}
echo success();
```

5. Then print out the tweets

```
echo colorize("Here are $username's tweets:\n", 'blue', true);
foreach ($records as $record) {
    echo colorize($record['bins']['tweet'], 'black')."\n";
}
```

Exercise 4 – PHP: Scan

Scan all tweets for all users

Locate TweetService class in the PHP project

1. In TweetService.scanAllTweets()
    1. Initiate scan operation that invokes callback for outputting tweets to the console

Scan all the tweets for all users – warning – there could be a large result set.

In the TweetService.scanAllTweets(), add code similar to this:

1. Perform a Scan operation
2. process the results

```
$status = $this->client->scan('test','tweets', function
($record) {
    var_dump($record['bins']['tweet']);
}, array('tweet'));
if ($status !== Aerospike::OK) {
    // throwing an \Aerospike\Training\Exception
    throw new Exception($this->client, "Failed to scan
test.tweets");
}
```

Exercise 5 – PHP: Read-modify-write

Update the User record with a new password ONLY if the User record is un-modified.

Locate UserService class in the PHP project

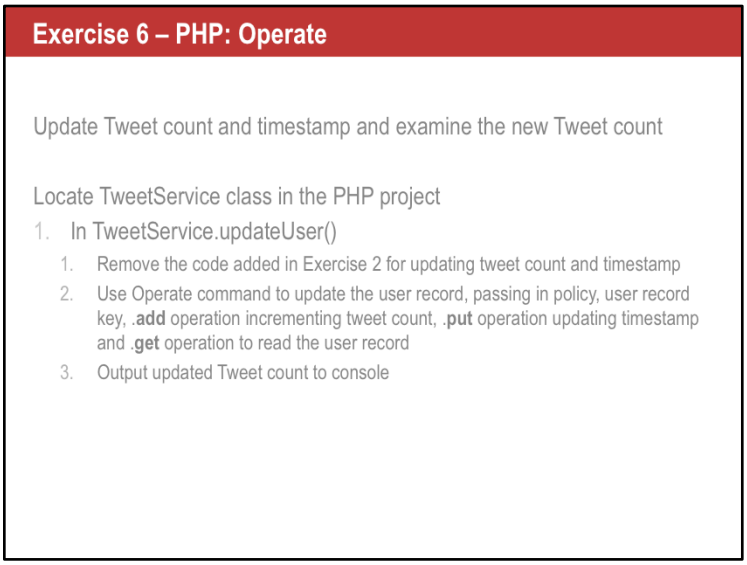
1. In UserService.updatePasswordUsingCAS()
    1. Create a Write policy array
    2. Set Write policy generation to the value read from the User record.
    3. Set Write policy's generation policy to POLICY_GEN_EQ
    4. Update the User record with the new password using the Write policy array

Update the User record with a new password ONLY if the User record is un-modified

In UserService.updatePasswordUsingCAS(), add code similar to this:

1. Create a Write policy array
2. Set Write policy generation to the value read from the User record.
3. Set Write policy's generation policy to POLICY_GEN_EQ
4. Update the User record with the new password using the Generation count from the meta data

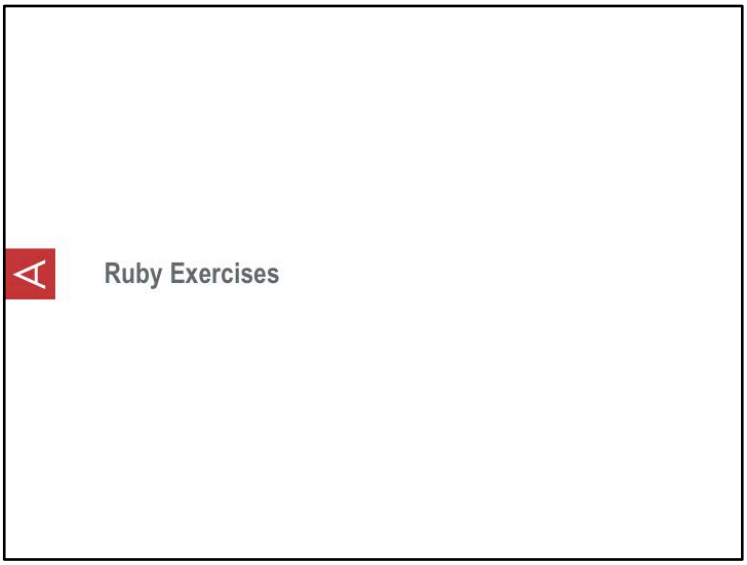
```
$key = $this->getUserKey($username);
if ($this->annotate) display_code(__FILE__, __LINE__, 1);
$status = $this->client->exists($key, $metadata);
if ($status !== Aerospike::OK) {
    // throwing an \Aerospike\Training\Exception
    echo fail();
    throw new Exception($this->client, "Failed to retrieve
metadata for the record");
}
echo success();
var_dump($metadata);

echo colorize("Updating the user's password if the generation
matches >", 'black', true);
if ($this->annotate) display_code(__FILE__, __LINE__, 4);
$bins = array('password' => $new_password);
$policy = array(Aerospike::OPT_POLICY_GEN =>
    array(Aerospike::POLICY_GEN_EQ, $metadata['generation']));
$status = $this->client->put($key, $bins, $metadata['ttl'],
$policy);
if ($status !== Aerospike::OK) {
    // throwing an \Aerospike\Training\Exception
    echo fail();
    throw new Exception($this->client, "Writing with
POLICY_GEN_EQ failed due to generation mismatch");
}
```



Aerospike can perform multiple operations on a record in one transaction. Update the tweet count and timestamp in a user record and read the new tweet count.

In TweetService.updateUser()

1. Delete the code added in Exercise 2


Ruby Exercises

Exercise 1 – Ruby: Connect & Disconnect

Locate the Training module in the Ruby project

1. Create an instance of the Aerospike client in the Training module's init method. Ensure that this connection is created only once
2. Add code to disconnect from the cluster in the Training.finish method. Ensure that this code is executed only once

```
# todo: configure and instantiate the Aerospike client
#@client = # assign the client to the module instance variable
```

```
# todo: close the client's connection to the cluster
```

In this exercise you will connect to a Cluster by creating an Aerospike client instance, using a single IP address and port. These should be to a node in your own cluster.

Ensure that you only create one client instance at the start of the program. The Aerospike is thread safe and creates a pool of worker threads, this means you DO NOT need to create your own connection or thread pool.

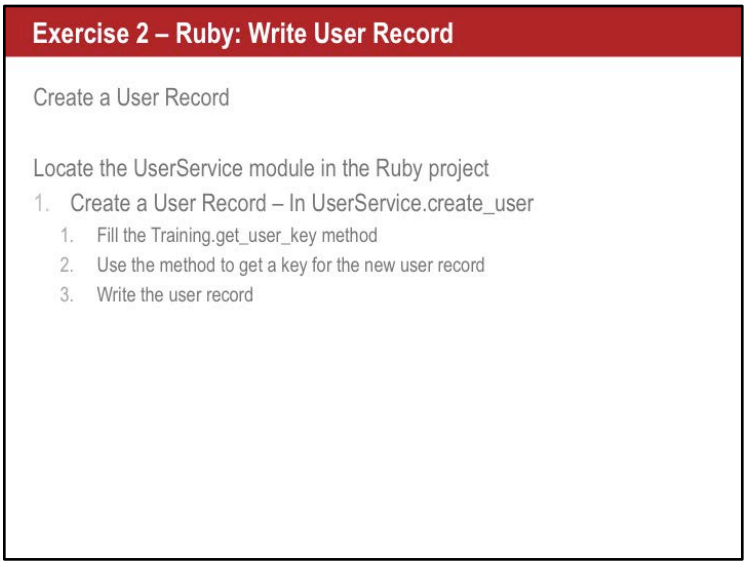
1. In the Training.init, add code similar to this;

```
@client = host ? Client.new(Host.new(host, port)) : Client.new
```

Make sure you have your server up and you know its IP address

2. In Training.finish, add code, similar to this, to disconnect from the cluster. This should only be done once. After close() is called, the client instance cannot be used.

```
@client->close
```

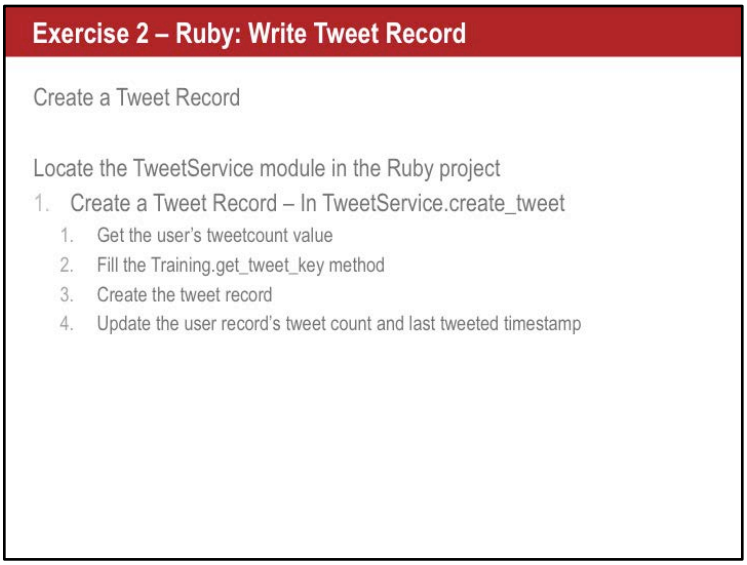


Create a User Record. In UserService.create_user, add code similar to this:

1. Fill the Training.get_user_key method
2. Use the method to get a key for the new user record
3. Write the user record

```
def get_user_key(username)
  Key.new(self.namespace, self.set_name, username)
end
```

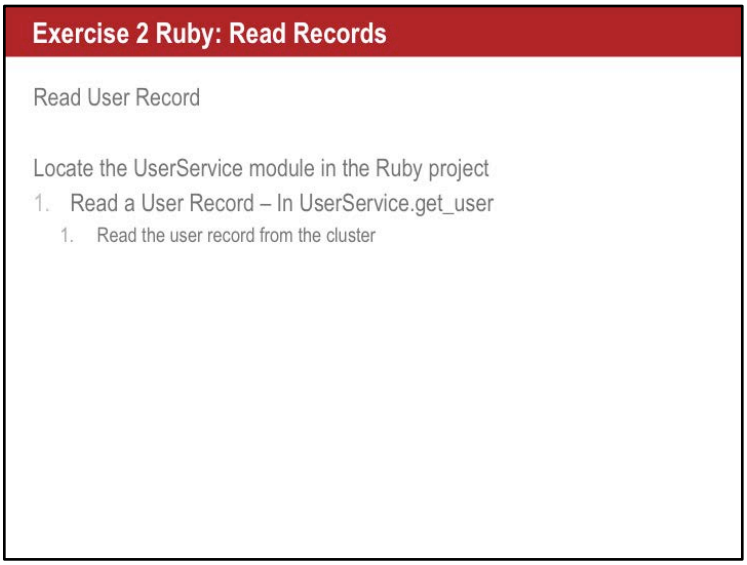
```
   def create_user(client)
     puts "\nCreate a new user".colorize(:color => :blue, :mode
=> :bold)
     print 'Enter username (or hit Return to skip):
'.colorize(:blue)
     username = gets.chomp
     bins = { 'username' => username }
     return if username.length < 1
     print "Enter password for #{username}: ".colorize(:blue)
     bins['password'] = gets.chomp
     print "Select gender (f or m) for #{username}:
".colorize(:blue)
     bins['gender'] = gets.chomp
     print "Select region (north, south, east or west) for
#{username}: ".colorize(:blue)
     bins['region'] = gets.chomp
     print "Enter comma-separated interests for #{username}:
".colorize(:blue)
     bins['interests'] = gets.chomp.split(',')

     print "Creating user record >".colorize(:color
=> :black, :mode => :bold)
     begin
       key = Key.new(self.namespace, self.set_name, username)
       client.put(key, bins, self.write_policy)
       self.yep
     rescue
       self.nope
       puts "Connection to Aerospike cluster failed! Please
check the server settings and try again!".colorize(:color
=> :red, :mode => :bold)
     end
   end
```



Create a Tweet Record. In TweetService.create_tweet, add code similar to this:

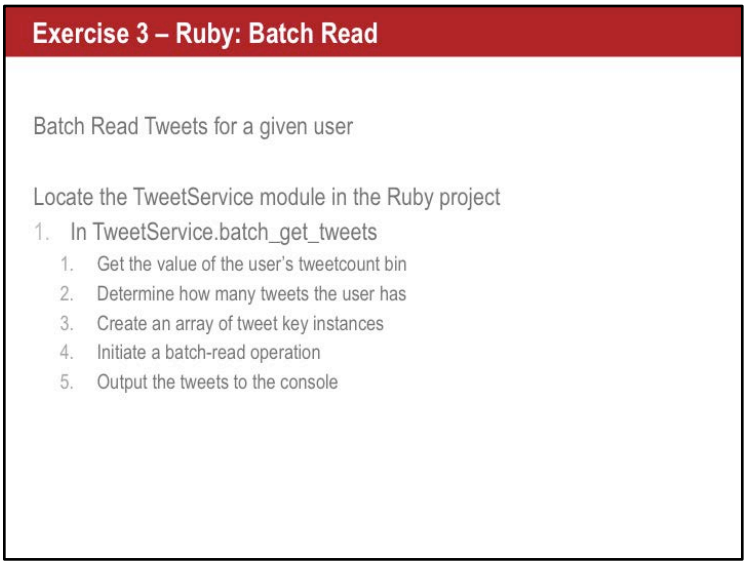
1. Get the user's tweetcount value
2. Fill the Training.get_tweet_key method
3. Create the tweet record
4. Update the user record's tweet count and last tweeted timestamp

```
  def get_tweet_key(username, id)
    Key.new(self.namespace, 'tweets', "#{username}:#{id}")
  end

    def create_tweet(client, username = nil)
      puts "\nCreate a new tweet".colorize(:color => :blue, :mode
=> :bold)
      unless username
        print "Enter username (or hit Return to skip): ".colorize(:blue)
        username = gets.chomp
      end
      unless username == ''
        key = self.get_user_key(username)
        rec = client.get(key, ['tweetcount'])
        bins = rec.bins unless rec.nil?
        tweet_count = bins.nil? ? 1 : bins['tweetcount'] + 1
      end
      ts = (Time.now.to_f * 1000).round
      bins = {'ts' => ts}
      print "Enter tweet for #{username}: ".colorize(:blue)
      bins['tweet'] = gets.chomp
      bins['username'] = username
      print "Creating tweet record >".colorize(:color => :black, :mode
=> :bold)
      key = self.get_tweet_key(username, tweet_count)
      begin
        client.put(key, bins)
        self.yep
      rescue Exception => e
        self.nope
        puts "Failed to create the tweet".colorize(:color => :red, :mode
=> :bold)
        pp e
        return
      end
      key = self.get_user_key(username)
      bins = {'tweetcount' => tweet_count, 'lasttweeted' => ts}
      print "Updating the user record >".colorize(:color => :black, :mode
=> :bold)
      begin
        client.put(key, bins)
        self.yep
      rescue
        self.nope
        print "Failed to update the user record >".colorize(:color
=> :black, :mode => :bold)
      end
    end
```



In UserService.get_user, add code, similar to this, to:

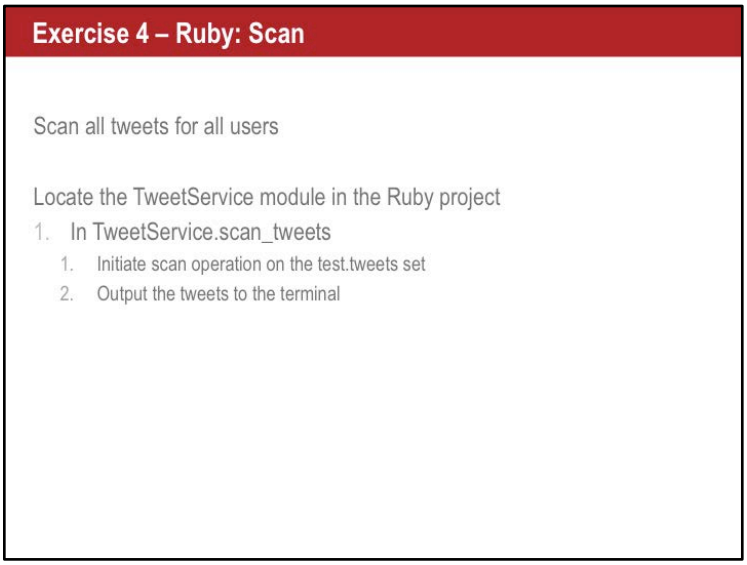
```
  def get_user(client)
      puts "\nCreate a user".colorize(:color
=> :blue, :mode => :bold)       print 'Enter username
(or hit Return to skip): '.colorize(:blue)
username = gets.chomp
      key = self.get_user_key(username)
      rec = client.get(key)
      if rec
        puts "Record.key (Key)".colorize(:mode
=> :bold)
        pp rec.key
        puts "Record.generation
(Fixnum)".colorize(:mode => :bold)         pp
rec.generation
        puts "Record.expiration
(Fixnum)".colorize(:mode => :bold)         pp
rec.expiration
        puts "Record.bins (Hash)".colorize(:mode
=> :bold)
        pp rec.bins
      else
        puts "There is no such user
#{username}".colorize(:red)       end
    end
```



Read all the tweets for a given user. In TweetService.batch_get_tweets, add code similar to this:

1. Get the value of the user's tweetcount bin
2. Determine how many tweets the user has
3. Create an array of tweet key instances
4. Initiate a batch-read operation
5. Output the tweets to the console

```
    def batch_get_tweets(client, username = nil)
      puts "\nGet the user's tweet".colorize(:color => :blue, :mode
=> :bold)
      unless username
        print "Enter username (or hit Return to skip): ".colorize(:blue)
        username = gets.chomp
      end
      unless username == ''
        key = self.get_user_key(username)
        rec = client.get(key, ['tweetcount'])
        unless rec
          puts "There is no such user #{username}".colorize(:red)
          return
        end
        bins = rec.bins
        tweet_count = bins.nil? ? 0 : bins['tweetcount']
        keys = []
        (1..tweet_count).each do |i|
          keys.push(self.get_tweet_key(username, i))
        end
        print "Batch-reading the user's tweets >".colorize(:color
=> :black, :mode => :bold)
        begin
          recs = client.batch_get(keys)
          self.yep
        rescue Exception => e
          self.nope
          puts "Failed to batch-read the tweets for
#{username}".colorize(:color => :red, :mode => :bold)
          pp e
          return
        end
        puts "Here are #{username}'s tweets:".colorize(:color
=> :blue, :mode => :bold)
        recs.each do |r|
          puts r.bins['tweet']
        end
      end
    end
```



Scan all the tweets for all users – warning – there could be a large result set.

In the TweetService.scan_tweets, add code similar to this:

1. Initiate scan operation on the test.tweets set
2. Output the tweets to the terminal

```
    def scan_tweets(client)
      puts "\nScan for tweets".colorize(:color
=> :blue, :mode => :bold)
      begin
        policy = ScanPolicy.new(:fail_on_cluster_change =>
true)
        recordset = client.scan_all(self.namespace,
'tweets', [], policy)

        recordset.each do |rec|
          puts rec.bins['tweet']
        end
      rescue
        puts "Failed to scan test.tweets".colorize(:color
=> :red, :mode => :bold)
      end
    end
```

Exercise 5 – Ruby: Read-modify-write

Update the User record with a new password ONLY if the user record is unmodified.

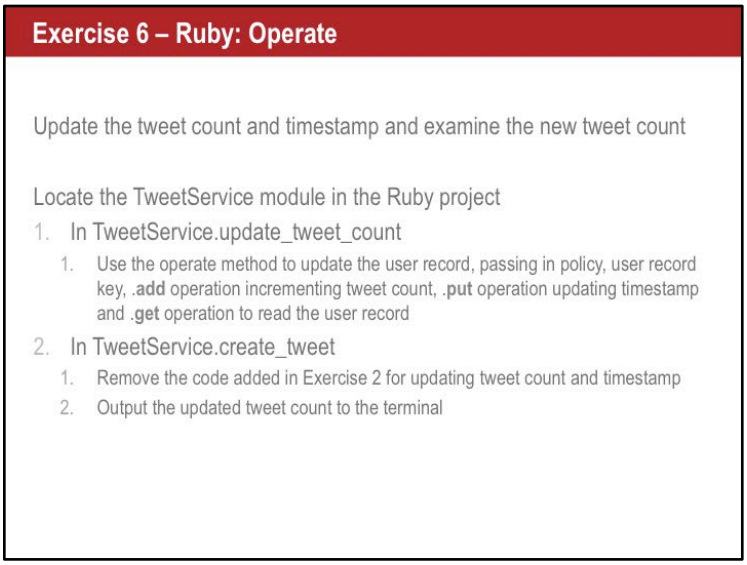
Locate the UserService module in the Ruby project

1. In UserService.check_and_set_password
    1. Get the generation of the user record
    2. Instantiate a WritePolicy
    3. Set the WritePolicy.generation to the value read from the user record
    4. Set WritePolicy's generation_policy to GenerationPolicy::EXPECT_GEN_EQUAL
    5. Update the user record with the new password

Update the User record with a new password ONLY if the User record is une modified In UserService.check_and_set_password, add code similar to this:

1. Get the generation of the user record
2. Instantiate a WritePolicy
3. Set the WritePolicy.generation to the value read from the user record
4. Set WritePolicy's generation_policy to GenerationPolicy::EXPECT_GEN_EQUAL
5. Update the user record with the new password

```
        key = self.get_user_key(username)
        rec = client.get_header(key)
        generation = rec.generation


        write_policy = WritePolicy.new
        write_policy.generation_policy =
GenerationPolicy::EXPECT_GEN_EQUAL
        write_policy.generation = generation
        client.put(key, bins, write_policy)
```



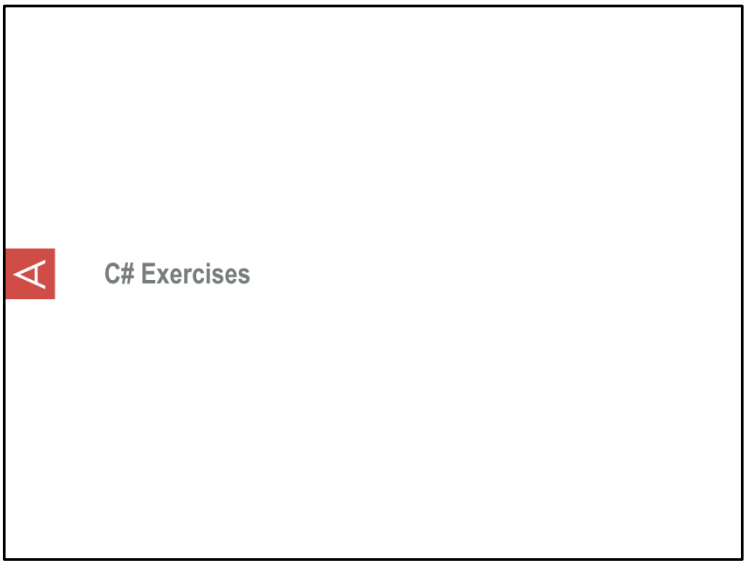
In TweetService.update_tweet_count

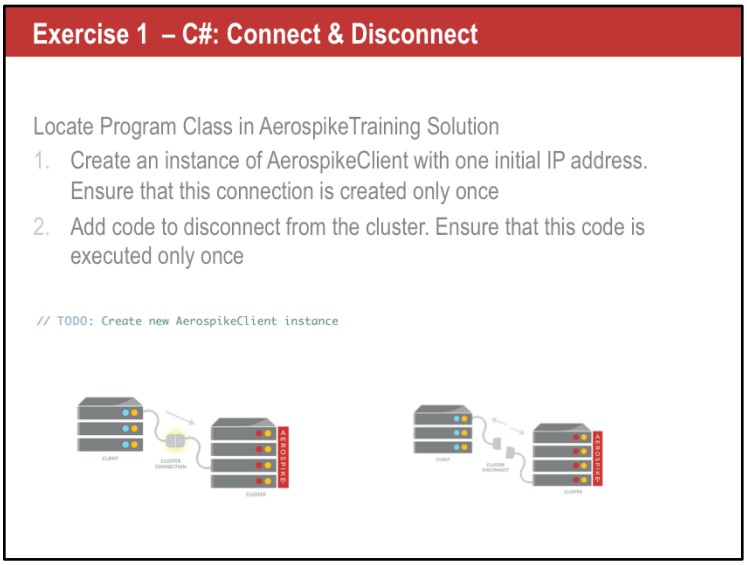
1. Use the operate method to update the user record, passing in policy, user record key, .add operation incrementing tweet count, .put operation updating timestamp and .get operation to read the user record

```
ops = [
 Operation.add(Bin.new('tweetcount', 1)),
 Operation.put(Bin.new('lasttweeted', (Time.now.to_f *
 1000).round))
]
client.operate(key, ops)
```

In TweetService.create_tweet

1. Remove the code added in Exercise 2 for updating tweet count and timestamp
2. Output the updated tweet count to the terminal


C# Exercises



In this exercise you will connect to a Cluster by crea ng an AerospikeClient instance, passing a single IP address and port to the constructor. The IP address and port should be to a node in your own cluster.

Ensure that you only create one client instance at the start of the program. The AerospikeClient is thread safe and creates a pool of worker threads, this means you DO NOT need to create your own connecNon or thread pool.

1. In the Main() method for the class Program, add code similar to this;

```
// Specify IP of one of the nodes in the cluster
string asServerIP = "172.16.159.206";
// Specity Port that the node is listening on
int asServerPort = 3000;
// Establish connection
client = new AerospikeClient(asServerIP, asServerPort);
```

Make sure you have your server up and you know its IP address

2. At the end of Main() method, add code, similar to this, to disconnect from the cluster. This should only be done once. Ae er close() is called, the client instance cannot be used.

```
if (client != null && client.Connected)
{
      // Close Aerospike server connection
      client.Close();
}
```

Exercise 2 – C#: Write Records

Create a User Record and Tweet Record

Locate UserService and TweetService classes in AerospikeTraining Solution

1. Create a User Record – In UserService.createUser()
   1. Create an instance of WritePolicy
   2. Create Key and Bin instances for the User Record
   3. Write User Record
2. Create a Tweet Record – In TweetService.createTweet()
   1. Create an instance of WritePolicy
   2. Create Key and Bin instances for the Tweet Record
   3. Write Tweet Record
   4. Update Tweet count and last Tweeted timestamp in the User Record

Create a User Record. In UserService.createUser(), add code similar to this:

1. Create a WritePolicy

```
// Write record
WritePolicy wPolicy = new WritePolicy();
wPolicy.recordExistsAction = RecordExistsAction.UPDATE;
```

2. Create Key and Bin instances

```
Key key = new Key("test", "users", username);
Bin bin1 = new Bin("username", username);
Bin bin2 = new Bin("password", password);
Bin bin3 = new Bin("gender", gender);
Bin bin4 = new Bin("region", region);
Bin bin5 = new Bin("lasttweeted", 0);
Bin bin6 = new Bin("tweetcount", 0);
Bin bin7 = Bin.AsList("interests",
           interests.Split(',').ToList<object>());
```

3. Write a user record using the Key and Bins

```
client.Put(wPolicy, key, bin1, bin2, bin3,
                              bin4, bin5, bin6, bin7);
```

Create a Tweet Record. In TweetService.createTweet(), add code similar to this:

1. Read User record

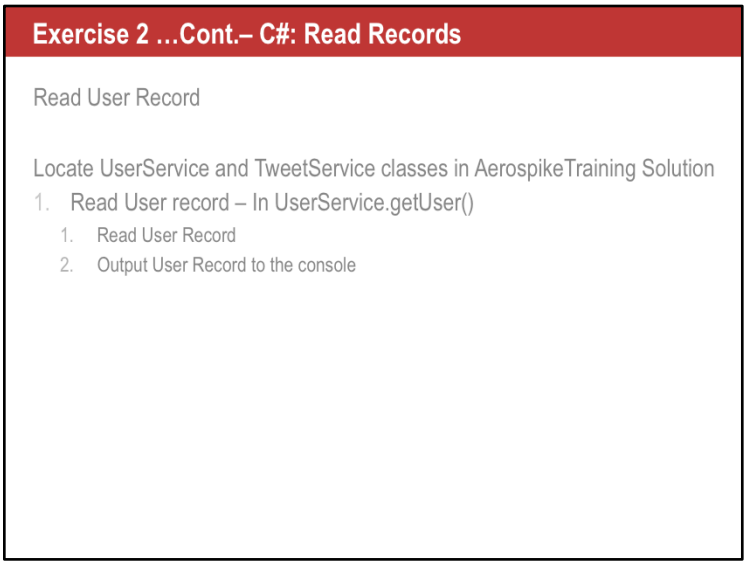
```
userRecord = client.Get(null, new Key("test", "tweets", username));
```

2. Create a WritePolicy

```
WritePolicy wPolicy = new WritePolicy();
wPolicy.recordExistsAction = RecordExistsAction.UPDATE;
```

3. Create Key and Bin instances

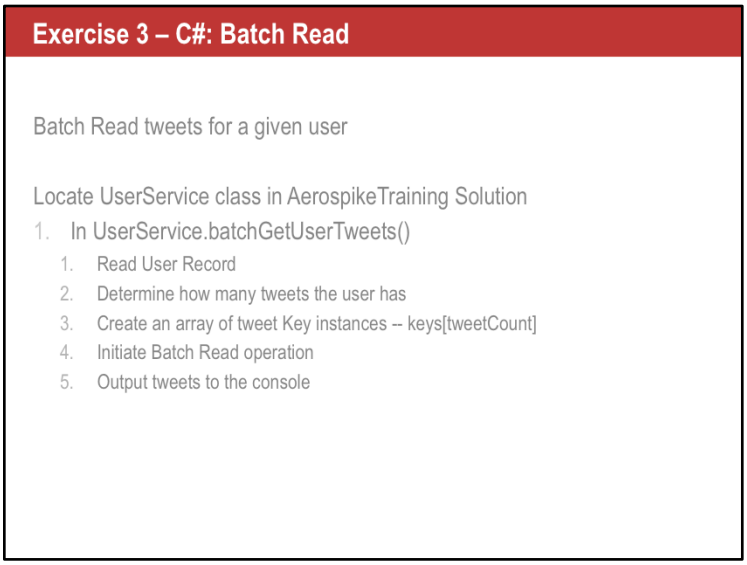
```
long ts = getTimeStamp();
tweetKey = new Key("test", "tweets", username + ":"
                                        + nextTweetCount);
Bin bin1 = new Bin("tweet", tweet);
Bin bin2 = new Bin("ts", ts);
Bin bin3 = new Bin("username", username);

client.Put(wPolicy, tweetKey, bin1, bin2, bin3);
Console.WriteLine("\nINFO: Tweet record created!");

// Update tweet count and last tweet'd timestamp
// in the user record
updateUser(client, userKey, wPolicy, ts, nextTweetCount);
```



Read a User Record. In UserService.getUser(), add code, similar to this, to:

1. Read a User record
2. Output the User record to the console

```
// Check if username exists
userKey = new Key("test", "users", username);
userRecord = client.Get(null, userKey);
if (userRecord != null)
{
    Console.WriteLine("\nINFO: User record read successfully! Here are the details:\n");
    Console.WriteLine("username:     " + userRecord.GetValue("username"));
    Console.WriteLine("password:     " + userRecord.GetValue("password"));
    Console.WriteLine("gender:       " + userRecord.GetValue("gender"));
    Console.WriteLine("region:       " + userRecord.GetValue("region"));
    Console.WriteLine("tweetcount:   " + userRecord.GetValue("tweetcount"));
    List<object> interests = (List<object>) userRecord.GetValue("interests");
    Console.WriteLine("interests:    " + interests.Aggregate((x, y) => x + "," + y));
}
else
{
    Console.WriteLine("ERROR: User record not found!");
}
```



Read all the tweets for a given user. In UserService.batchGetUserTweets(), add code similar to this:

1. Read a user record

```
userKey = new Key("test", "users", username);
userRecord = client.Get(null, userKey);
```

2. Get the tweet count

```
// Get how many tweets the user has
int tweetCount = int.Parse(userRecord.GetValue("tweetcount").ToString());
```

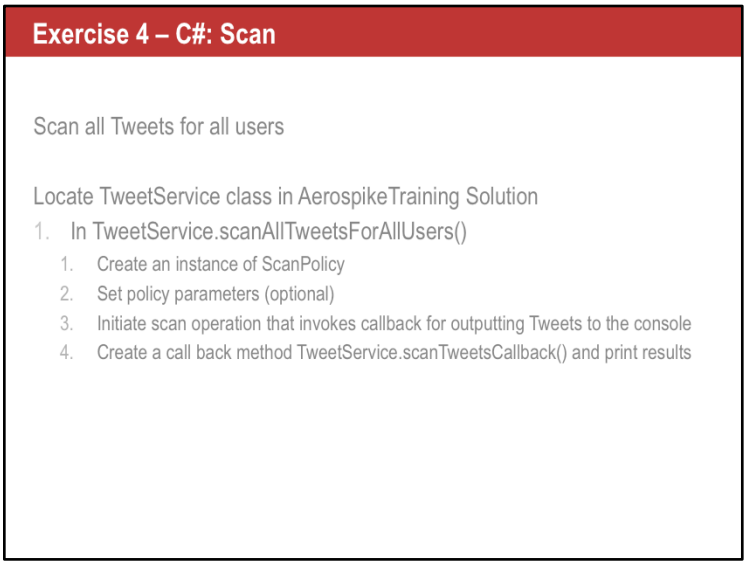
3. Create a “list” of tweet keys

```
// Create an array of keys so we can initiate batch read operation
Key[] keys = new Key[tweetCount];
for (int i = 0; i < keys.Length; i++)
{
    keys[i] = new Key("test", "tweets", (username + ":" + (i + 1)));
}
Console.WriteLine("\nHere's " + username + "'s tweet(s):\n");
```

4. Perform a Batch opera on to read all the tweets

```
// Initiate batch read operation
Record[] records = client.Get(null, keys);
for (int j = 0; j < records.Length; j++)
{
    Console.WriteLine(records[j].GetValue("tweet"));
}
```

5. Then print out the tweets



Scan all the tweets for all users – warning – there could be a large result set.

In the TweetService.scanAllTweetsForAllUsers(), add code similar to this:

1. Create a ScanPolicy
2. Set policy parameters

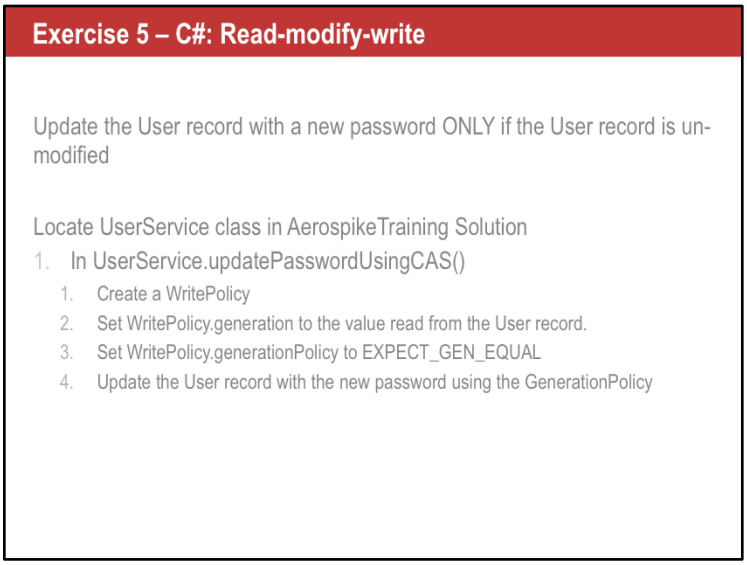
```
ScanPolicy policy = new ScanPolicy();
policy.concurrentNodes = true;
policy.priority = Priority.LOW;
policy.includeBinData = true;
```

3. Perform a Scan opera on

```
client.ScanAll(policy, "test", "tweets", scanTweetsCallback, "tweet");
```

4. Create a call back method TweetService.scanTweetsCallback() and print the results

```
public void scanTweetsCallback(Key key, Record record)
{
     Console.WriteLine(record.GetValue("tweet"));
} //scanTweetsCallback
```

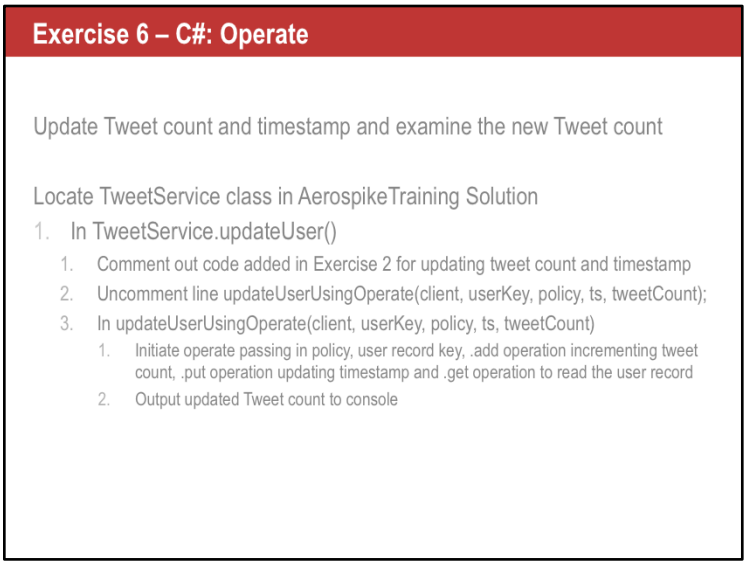


Update the User record with a new password ONLY if the User record is un modified

In UserService.updatePasswordUsingCAS(), add code similar to this:

1. Create a WritePolicy
2. Set WritePolicy.genera on to the value read from the User record.
3. Set WritePolicy.genera onPolicy to EXPECT_GEN_EQUAL
4. Update the User record with the new password using the Genera onPolicy

```
// Check if username exists
userKey = new Key("test", "users", username);
userRecord = client.Get(null, userKey);
if (userRecord != null)
{
    // Get new password
    string password;
    Console.WriteLine("Enter new password for " + username + ":");
    password = Console.ReadLine();

    WritePolicy writePolicy = new WritePolicy();
    // record generation
    writePolicy.generation = userRecord.generation;
    writePolicy.generationPolicy = GenerationPolicy.EXPECT_GEN_EQUAL;
    // password Bin
    passwordBin = new Bin("password", password);
    client.Put(writePolicy, userKey, passwordBin);

    Console.WriteLine("\nINFO: The password has been set to: " + password);
}
```



Aerospike can perform mulNple opera ons on a record in one transacNon. Update the tweet count and Nmestamp in a user record and read the new tweet count.

In TweetService.updateUser()

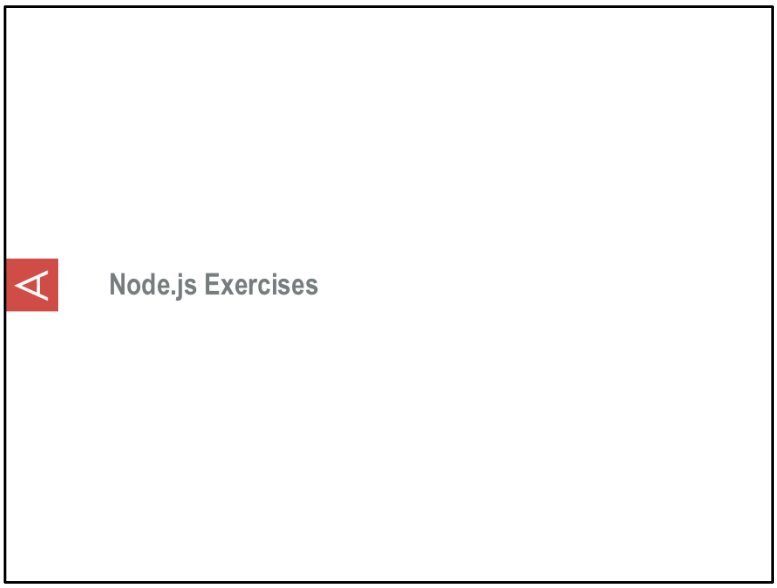
1. Comment out the code added in exercise 2

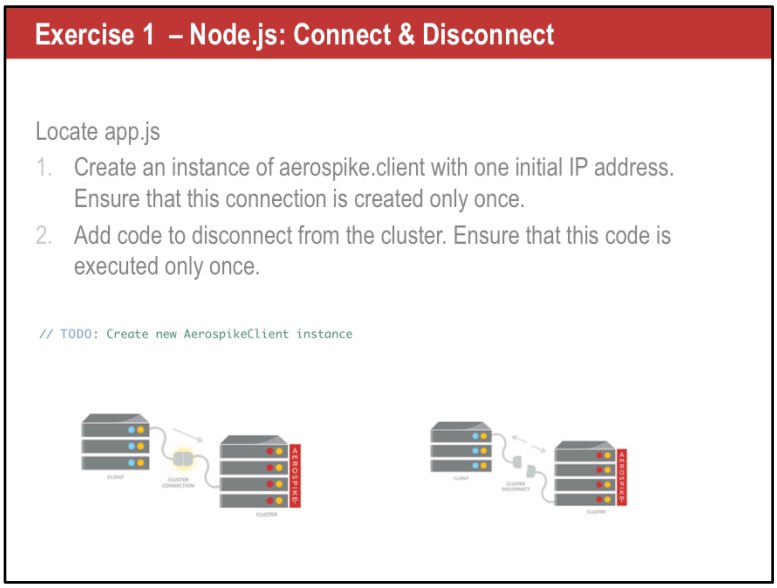
```
// TODO: Update tweet count and last tweeted timestamp in the user record using Operate
// Exercise 6
// updateUserUsingOperate(client, userKey, policy, ts);
```

2. Uncomment the line:

3. In TweetService.updateUserUsingOperate() , add code similar to this:

```
Record record = client.Operate(policy, userKey,
     Operation.Add(new Bin("tweetcount", 1)),
     Operation.Put(new Bin("lasttweeted", ts)),
     Operation.Get());
Console.WriteLine("INFO: The tweet count now is: " + record.GetValue("tweetcount"));
```


Node.js Exercises



In this exercise you will connect to a Cluster by crea ng an Aerospike client instance, passing a single IP address and port to the constructor. The IP address and port should be to a node in your own cluster.

Ensure that you only create one client instance at the start of the program. The Aerospike client is thread safe and creates a pool of worker threads, this means you DO NOT need to create your own connecNon or thread pool.

1. In app.js add code similar to this;

```
// Connect to the Aerospike Cluster
var client = aerospike.client({
    hosts: [ { addr: '172.16.159.172', port: 3000 } ]
}).connect(function(response) {
    // Check for errors
    if ( response.code == aerospike.status.AEROSPIKE_OK ) {
      // Connection succeeded
      console.log("Connection to the Aerospike cluster succeeded!");
    }
    else {
      // Connection failed
      console.log("Connection to the Aerospike cluster failed. Please check cluster IP and Port
settings and try again.");
      process.exit(0);
    }
});
```

Make sure you have your server up and you know its IP address

2. Add a *process.on* exit funcNon and call close() to disconnect from the cluster. This should only be done once. Ae er close() is called, the client instance cannot be used.

```
// Setup tear down
process.on('exit', function() {
  if (client != null) {
    client.close();
    // console.log("Connection to Aerospike cluster closed!");
  }
});
```

Exercise 2 – Node.js: Write Records

Create a User Record and Tweet Record

Locate user_service.js

1. Create a User Record – In exports.createuser
   1. Create Key and Bin instances for the User Record
   2. Write User Record

Create a User Record. In exports.createUser, add code similar to this:

1. Create Key and Bin instances

```
var key = {
  ns:  "test",
  set: "users",
  key: answers.username
};

var bins = {
  username: answers.username,
  password: answers.password,
  gender: answers.gender,
  region: answers.region,
  lasttweeted: 0,
  tweetcount: 0,
  interests: answers.interests.split(",")
};
```

2. Write a user record using the Key and Bins

```
client.put(key, bins, function(err, rec, meta) {
  // Check for errors
  if ( err.code === 0 ) {
    console.log("INFO: User record created!");

    // Create tweet record
    tweet_service.createTweet(client);
  }
  else {
    console.log("ERROR: User record not created!");
    console.log(err);
  }
});
```

Exercise 2 – ..Cont Node.js: Write Records..

Create a User Record and Tweet Record

Locate tweet_service.js

1. Create a Tweet Record – In export.createTweet
   1. Create Key and Bin instances for the Tweet Record
   2. Write Tweet Record
   3. Update tweet count and last tweeted timestamp in the User Record

Create a Tweet Record. In exports.createTweet add code similar to this:

1. Create Key and Bin instances

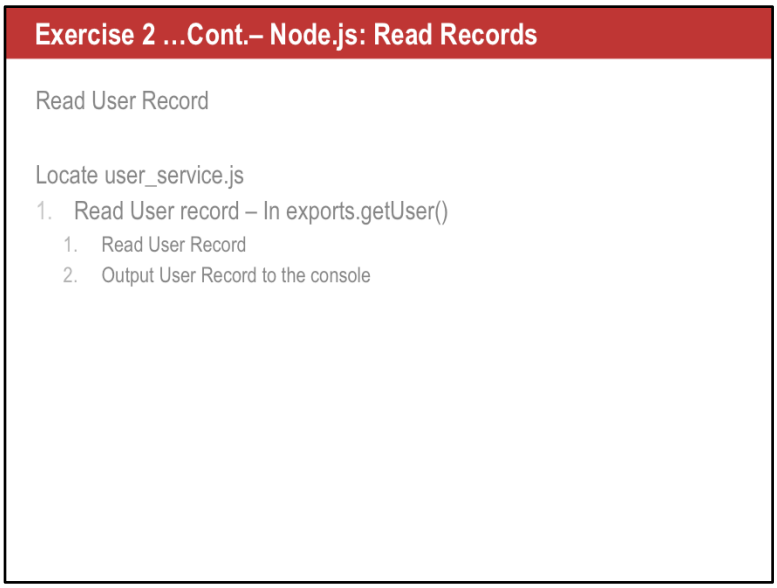
```
// Write Tweet record
var tweet_key = {
  ns:  "test",
  set: "tweets",
  key: userrecord.username + ":" + tweet_count
};

var bins = {
  username: userrecord.username,
  tweet: answer.tweet,
  ts: ts
};
```

2. Write a tweet record using the Key and Bins

```
client.put(tweet_key, bins, function(err, tweetrecord, meta) {
  // Check for errors
  if ( err.code === 0 ) {
    console.log("INFO: Tweet record created!");
```

3. Update the user record with tweet count

```
    // Update tweetcount and last tweet'd timestamp in the user record
    updateUser(client, user_key, ts, tweet_count);
  }
  else {
    console.log("ERROR: Tweet record not created!");
    console.log("",err);
  }
```



Read a User Record. In exports.getUser, add code, similar to this, to:

1. Read a User record
2. Output the User record to the console

```
// Read User record
var key = {
  ns:  "test",
  set: "users",
  key: answer.username
};

client.get(key, function(err, rec, meta) {
  // Check for errors
  if ( err.code === 0 ) {
    console.log("INFO: User record read successfully! Here are the details:");
    console.log("username:   " + rec.username);
    console.log("password:   " + rec.password);
    console.log("gender:     " + rec.gender);
    console.log("region:     " + rec.region);
    console.log("tweetcount: " + rec.tweetcount);
    console.log("lasttweeted: " + rec.lasttweeted);
    console.log("interests:  " + rec.interests);
  }
  else {
    console.log("ERROR: User record not found!");
  }
});
```

Exercise 3 – Node.js: Batch Read

Batch Read tweets for a given user

Locate user_service.js

1. In exports.batchGetUserTweets
   1. Read User Record
   2. Determine how many tweets the user has
   3. Create an array of tweet Key instances -- keys[tweetCount]
   4. Initiate Batch Read operation
   5. Output tweets to the console

Read all the tweets for a given user. In the funcNon exports.batchGetUserTweets, add code similar to this:

1. Read a user record

```
// Read User record
var key = {
  ns:  "test",
  set: "users",
  key: answer.username
};

client.get(key, function(err, userrecord, meta) {
  // Check for errors
  if ( err.code === 0 ) {
```

2. Get the tweet count

```
    var tweet_count = userrecord.tweetcount;
    var tweet_keys = [];
```

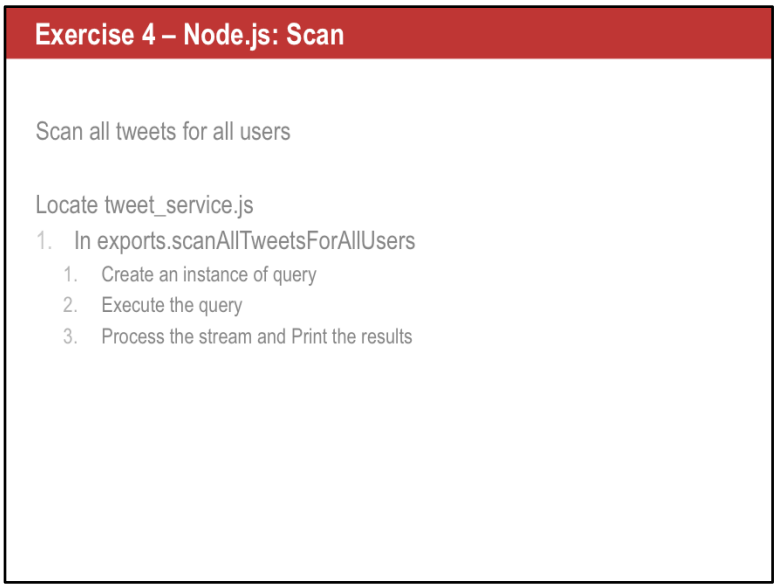
3. Create a “list” of tweet keys

```
    for(var i=1;i<=tweet_count;i++)  {
      tweet_keys.push({ns: "test", set: "tweets", key: answer.username + "
    }
```

4. Perform a Batch opera on to read all the tweets

5. Then print out the tweets

```
    client.batchGet(tweet_keys, function (err, results) {
      // Check for errors
      if ( err.code === 0 ) {
        for(var j=0;j<results.length;j++)  {
          console.log(results[j].record.tweet);
        }
      }
      else {
        console.log("ERROR: Batch Read Tweets For User failed\n", err);
      }
    });

  }
  else {
    console.log("ERROR: User record not found!");
  }
});

  });
```



Scan all the tweets for all users – warning – there could be a large result set.

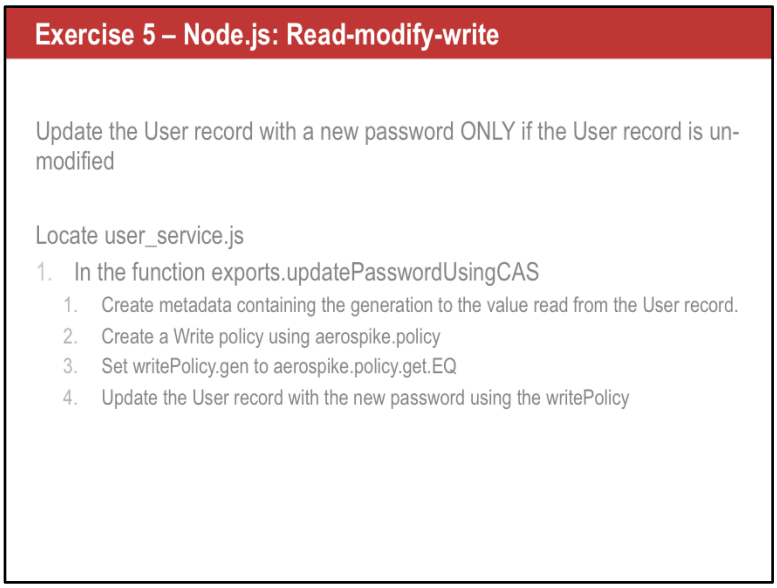
In the funcNon exports.scanAllTweetsForAllUsers, add code similar to this:

1. Create a instance of a query
2. Execute the query

```
var query = client.query('test', 'tweets');
var stream = query.execute();
```

3. Process the stream and print the results

```
stream.on('data', function(record){
     console.log(record.bins.tweet);
});
stream.on('error', function(err){
     console.log('ERROR: Scan All Tweets For All Users failed: ',err);
});
stream.on('end', function(){
     // console.log('INFO: Scan All Tweets For All Users completed!');
});
```

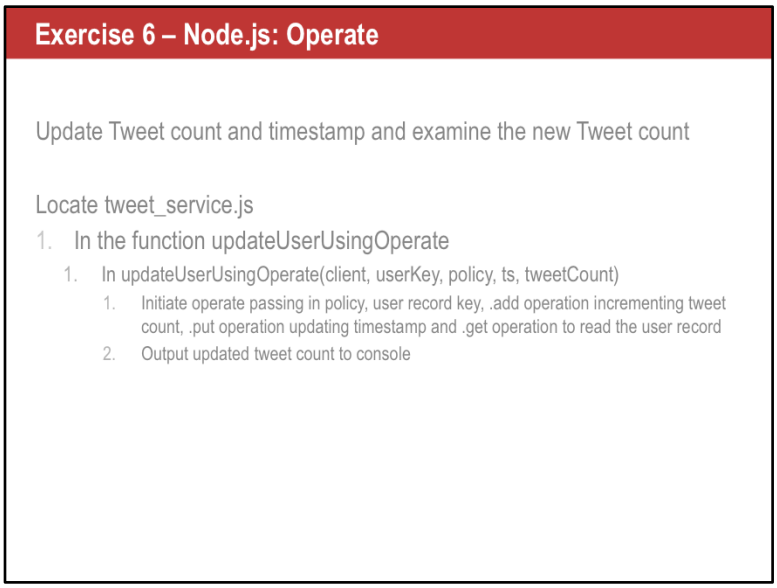


Update the User record with a new password ONLY if the User record is un modified

In exports.updatePasswordUsingCAS, add code similar to this:

1. Create metadata containing the genera on to the value read from the User record.
2. Create a Write policy using aerospike.policy
3. Set writePolicy.gen to aerospike.policy.get.EQ
4. Update the User record with the new password using the writePolicy

```
// Set the generation count to the current one from the user record.
// Then, setting writePolicy.gen to aerospike.policy.gen.EQ will ensure we don't have a 'dirty-
read' when updating user's password
var metadata = {
  gen: meta.gen
}

var writePolicy = aerospike.policy;
writePolicy.key = aerospike.policy.key.SEND;
writePolicy.retry = aerospike.policy.retry.NONE;
writePolicy.exists = aerospike.policy.exists.IGNORE;
writePolicy.commitLevel = aerospike.policy.commitLevel.ALL;
// Setting writePolicy.gen to aerospike.policy.gen.EQ will ensure we don't have a 'dirty-read'
when updating user's password
writePolicy.gen = aerospike.policy.gen.EQ;

var bin = {
  password: answer2.password
};

client.put(key, bin, metadata, writePolicy, function(err, rec) {
  // Check for errors
  if ( err.code === 0 ) {
    console.log("INFO: User password updated successfully!");
  }
  else {
    console.log("ERROR: User password update failed:\n", err);
  }
});
```



Aerospike can perform mulNple opera ons on a record in one transacNon. Update the tweet count and Nmestamp in a user record and read the new tweet count.

In updateUser

1. Comment out the code added in exercise 2

2. Uncomment the line:

```
// TODO: Update tweet count and last tweeted timestamp in the user record using operate
// Exercise 6
// console.log("TODO: Update tweet count and last tweeted timestamp in the user record using
operate");
// updateUserUsingOperate(client, user_key, ts);
```

3. In updateUserUsingOperate , add code similar to this:

```
// Update User record
var operator = aerospike.operator;
var operations = [operator.incr('tweetcount', 1),operator.write('lasttweeted',
ts),operator.read('tweetcount')];
client.operate(user_key, operations, function(err, bins, metadata, key) {
  // Check for errors
  if ( err.code === 0 ) {
    console.log("INFO: The tweet count now is: " + bins.tweetcount);
  }
  else {
    console.log("ERROR: User record not updated!");
    console.log(err);
  }
});
```


Go Exercises

Exercise 1 – Go: Connect & Disconnect

Locate tweetaspike.go

1. Create an instance of AerospikeClient with one initial IP address. Ensure that this connection is created only once.
2. Add code to disconnect from the cluster. Ensure that this code is executed only once.

```
// TODO: Create new AerospikeClient instance
```

In this exercise you will connect to a Cluster by crea ng an AerospikeClient instance, passing a single IP address and port to the constructor. The IP address and port should be to a node in your own cluster.

Ensure that you only create one client instance at the start of the program. The AerospikeClient is thread safe and creates a pool of worker threads, this means you DO NOT need to create your own connecNon or thread pool.

1. In the main() funcNon add code similar to this;

```
fmt.Println("INFO: Connecting to Aerospike cluster...")
// Establish connection to Aerospike server
client, err := NewClient("54.90.203.181", 3000)
panicOnError(err)
```

Make sure you have your server up and you know its IP address

2. Add a “defer” and call Close() to disconnect from the cluster. This should only be done once. Ae er close() is called, the client instance cannot be used.

```
defer client.Close()
```

Exercise 2 – Go: Write Records

Create a User Record and Tweet Record

Locate tweetaspike.go

1. Create a User Record – In CreateUser()
   1. Create an instance of WritePolicy
   2. Create Key and Bin instances for the User Record
   3. Write User Record
2. Create a Tweet Record – In CreateTweet()
   1. Create an instance of WritePolicy
   2. Create Key and Bin instances for the Tweet Record
   3. Write Tweet Record
   4. Update tweet count and last tweeted timestamp in the User Record

Create a User Record. In CreateUser(), add code similar to this:

1. Create a WritePolicy

```
// Write record
wPolicy := NewWritePolicy(0, 0) // generation = 0, expiration = 0
wPolicy.RecordExistsAction = UPDATE
```

2. Create Key and Bin instances

```
key, _ := NewKey("test", "users", username)
bin1 := NewBin("username", username)
bin2 := NewBin("password", password)
bin3 := NewBin("gender", gender)
bin4 := NewBin("region", region)
bin5 := NewBin("lasttweeted", 0)
bin6 := NewBin("tweetcount", 0)
arr := strings.Split(interests, ",")
bin7 := NewBin("interests", arr)
```

3. Write a user record using the Key and Bins

```
err := client.PutBins(wPolicy, key, bin1, bin2, bin3,
                              bin4, bin5, bin6, bin7)
panicOnError(err)
```

Create a Tweet Record. In CreateTweet(), add code similar to this:

1. Create a WritePolicy

```
// Write record
wPolicy := NewWritePolicy(0, 0) // generation = 0, expiration = 0
wPolicy.RecordExistsAction = UPDATE
// Create timestamp to store along with the tweet so we can
// query, index and report on it
timestamp := getTimeStamp()
```

2. Create Key and Bin instances

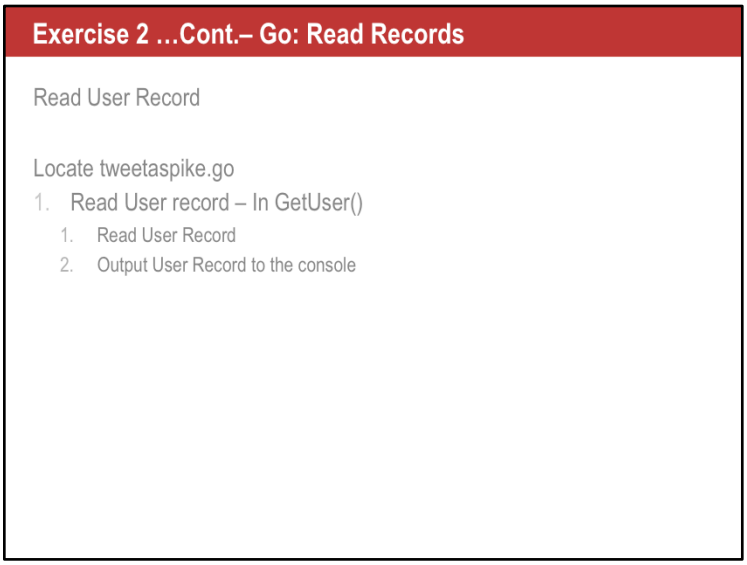
```
keyString := fmt.Sprintf("%s:%d", username, tweetCount)
tweetKey, _ := NewKey("test", "tweets", keyString)
bin1 := NewBin("tweet", tweet)
bin2 := NewBin("ts", timestamp)
bin3 := NewBin("username", username)
```

3. Write a tweet record using the Key and Bins

```
err := client.PutBins(wPolicy, tweetKey, bin1, bin2, bin3)
panicOnError(err)
fmt.Printf("\nINFO: Tweet record created! with key: %s, %v, %v, %v\n",
                        keyString, bin1, bin2, bin3)
```

4. Update the user record with tweet count

```
// Update tweet count and last tweet'd timestamp in the user record
updateUser(client, userKey, nil, timestamp, tweetCount)
```



Read a User Record. In GetUser(), add code, similar to this, to:

1. Read a User record
2. Output the User record to the console

```
// Check if username exists
userKey, _ := NewKey("test", "users", username)
userRecord, err := client.Get(nil, userKey)
panicOnError(err)
if userRecord != nil {
      fmt.Printf("\nINFO: User record read successfully! Here are the details:\n")
      fmt.Printf("username:   %s\n", userRecord.Bins["username"].(string))
      fmt.Printf("password:   %s\n", userRecord.Bins["password"].(string))
      fmt.Printf("gender:     %s\n", userRecord.Bins["gender"].(string))
      fmt.Printf("region:     %s\n", userRecord.Bins["region"].(string))
      fmt.Printf("tweetcount: %d\n", userRecord.Bins["tweetcount"].(int))
      fmt.Printf("interests:  %v\n", userRecord.Bins["interests"])
} else {
      fmt.Printf("ERROR: User record not found!\n")
}
```

Exercise 3 – Go: Batch Read

Batch Read tweets for a given user

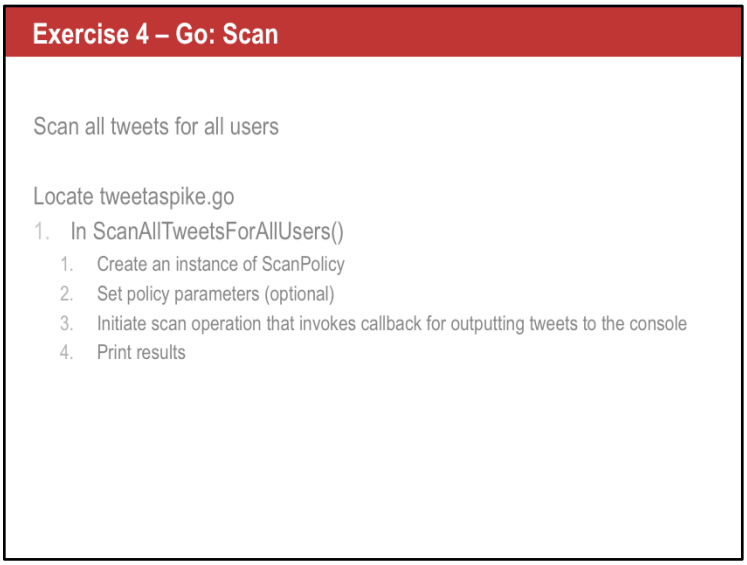
Locate tweetaspike.go

1. In BatchGetUserTweets()
    1. Read User Record
    2. Determine how many tweets the user has
    3. Create an array of tweet Key instances -- keys[tweetCount]
    4. Initiate Batch Read operation
    5. Output tweets to the console

Read all the tweets for a given user. In BatchGetUserTweets(), add code similar to this:

1. Read a user record
2. Get the tweet count
3. Create a “list” of tweet keys
4. Perform a Batch opera on to read all the tweets
5. Then print out the tweets

```
userKey, _ := NewKey("test", "users", username)
userRecord, err := client.Get(nil, userKey)
panicOnError(err)

if userRecord != nil {
      // Get how many tweets the user has
      tweetCount := userRecord.Bins["tweetcount"].(int)

      // Create an array of keys so we can initiate batch read
      // operation
      keys := make([]*Key, tweetCount)

      for i := 0; i < len(keys); i++ {
            keyString, _ := fmt.Scanf("%s:%d", username, i+1)
            key, _ := NewKey("test", "tweets", keyString)
            keys[i] = key
      }

      fmt.Printf("\nHere's %s's tweet(s):\n", username)

// Initiate batch read operation
      if len(keys) > 0 {
            records, err := client.BatchGet(NewPolicy(), keys)
            panicOnError(err)
            for _, element := range records {
                  fmt.Println(element.Bins["tweet"])
            }
      }
}
```



Scan all the tweets for all users – warning – there could be a large result set.

In the ScanAllTweetsForAllUsers(), add code similar to this:

1. Create a ScanPolicy
2. Set policy parameters

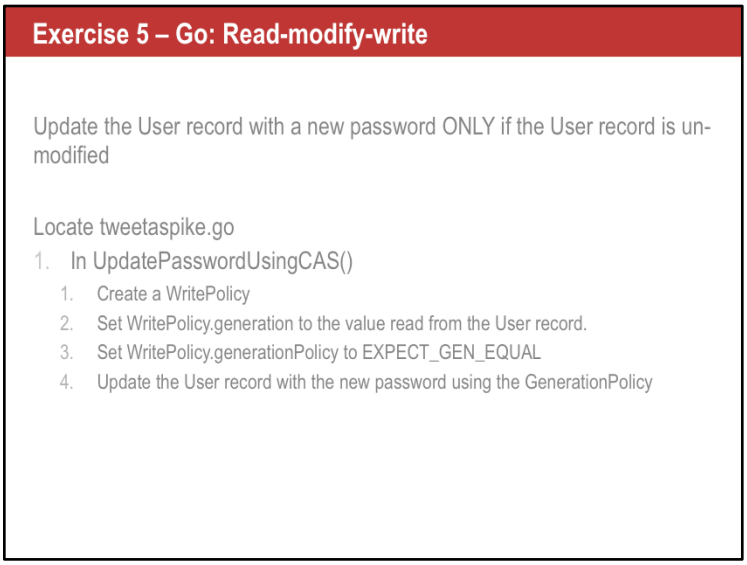
```
policy := NewScanPolicy()
policy.ConcurrentNodes = true
policy.Priority = LOW
policy.IncludeBinData = true
```

3. Perform a Scan opera on

```
records, err := client.ScanAll(policy, "test", "tweets", "tweet")
panicOnError(err)
```

4. Print the results

```
for element := range records.Records {
     fmt.Println(element.Bins["tweet"])
}
```

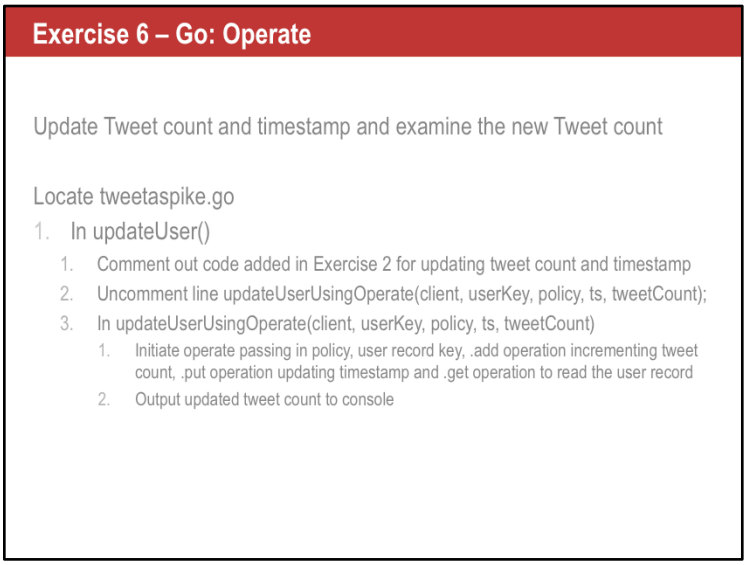


Update the User record with a new password ONLY if the User record is un modified

In UpdatePasswordUsingCAS(), add code similar to this:

1. Create a WritePolicy
2. Set WritePolicy.genera on to the value read from the User record.
3. Set WritePolicy.genera onPolicy to EXPECT_GEN_EQUAL
4. Update the User record with the new password using the Genera onPolicy

```
// Check if username exists
userKey, _ := NewKey("test", "users", username)
userRecord, err := client.Get(nil, userKey)
panicOnError(err)
if err == nil {
      // Get new password
      var password string
      fmt.Print("Enter new password for %s:", username)
      fmt.Scanf("%s", &password)

      writePolicy := NewWritePolicy(0, 0) // generation = 0, expiration = 0
      // record generation
      writePolicy.Generation = userRecord.Generation
      writePolicy.GenerationPolicy = EXPECT_GEN_EQUAL
      // password Bin
      passwordBin := NewBin("password", password)
      err = client.PutBins(writePolicy, userKey, passwordBin)
      panicOnError(err)
      fmt.Printf("\nINFO: The password has been set to: %s", password)
} else {
      fmt.Printf("ERROR: User record not found!")
}
```



Aerospike can perform mulNple opera ons on a record in one transacNon. Update the tweet count and Nmestamp in a user record and read the new tweet count.

In updateUser()

1. Comment out the code added in exercise 2

2. Uncomment the line:

```
// TODO: Update tweet count and last tweeted timestamp in the user record using Operate
// Exercise 6
// updateUserUsingOperate(client, userKey, policy, ts);
```

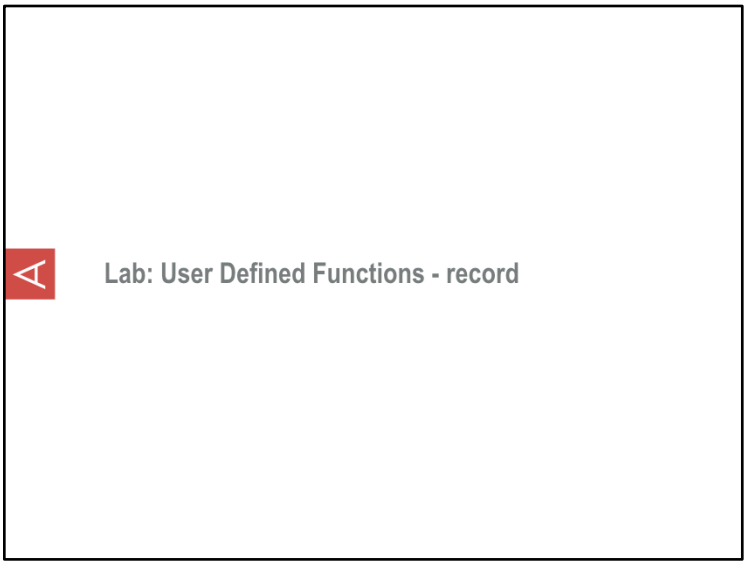
3. In updateUserUsingOperate() , add code similar to this:

```
record, err := client.Operate(policy, userKey,
     AddOp(NewBin("tweetcount", 1)),
     PutOp(NewBin("lasttweeted", timestamp)),
     GetOp())
panicOnError(err)

fmt.Printf("\nINFO: The tweet count now is: %d\n",
                     record.Bins["tweetcount"])
```

Summary

You have learned how to:

- Connect to Cluster
- Write and Read Records
- Batch Read Records
- Read-Modify-Write
- Operate
- Handle errors correctly


Lab: User Defined Functions - record

Objective

After successful completion of this Lab module you will have:

- Coded a Record UDF
- Registered the UDF with a cluster
- Invoked the UDF from your C#, Go, PHP, Ruby, Node.js or Java application

Lab Overview

The lab exercise augments "tweetaspike" by using a Record UDF. Here we will focus on a Record UDF that updates user password.

You will:

- Write a user defined function, in Lua, to update the user password
- Register the UDF
- Execute the UDF from your application

The application shell is located in your cloned GitHub directory

```
~/exercises/RecordUDF/<language>
```

Make sure you have your server up and you know its **IP address**

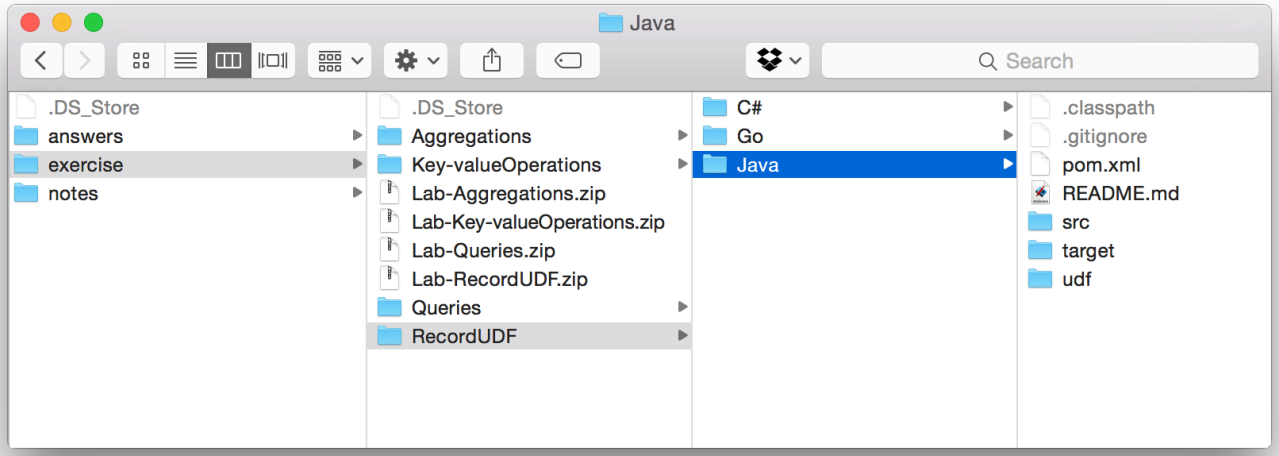
In your cloned or downloaded repository you will find the following directories:

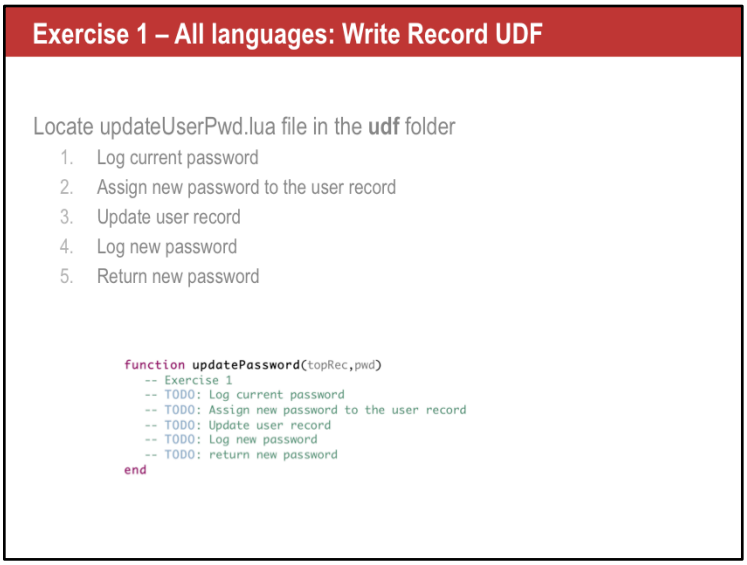
- Answers
- Exercise
- Notes

In the exercise directory, select the subdirectory for your programming language:

- C#
- Java
- Node.js
- Go
- PHP
- Ruby

The exercises for this module are in the UDF directory and your will find a Project/SoluNon/Codebase that is partly complete. Your tasks is to complete the code as outlined in each exercise.

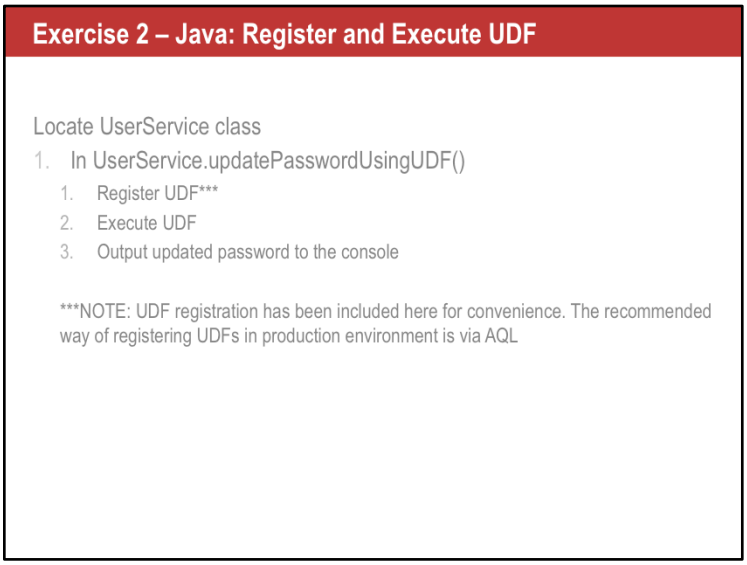




In this exercise you will create a record UDF that:

1. Logs the current password
2. Assigns a new password to the record, passed in via the pwd parameter
3. Updates the user record by calling **aerospike:update**(topRec)
4. Logs the new password
5. Returns the new password to the client

```
function updatePassword(topRec,pwd)
   -- Log current password
   debug("current password: " .. topRec['password'])
   -- Assign new password to the user record
   topRec['password'] = pwd
   -- Update user record
   aerospike:update(topRec)
   -- Log new password
   debug("new password: " .. topRec['password'])
   -- return new password
   return topRec['password']
end
```



In this exercise you will register and invoke the UDF created in Exercise 1.

We will programma cally register the UDF for convenience.

In UserService.updatePasswordUsing UDF(), locate these comments and add your code:

1. Register the UDF with an API call

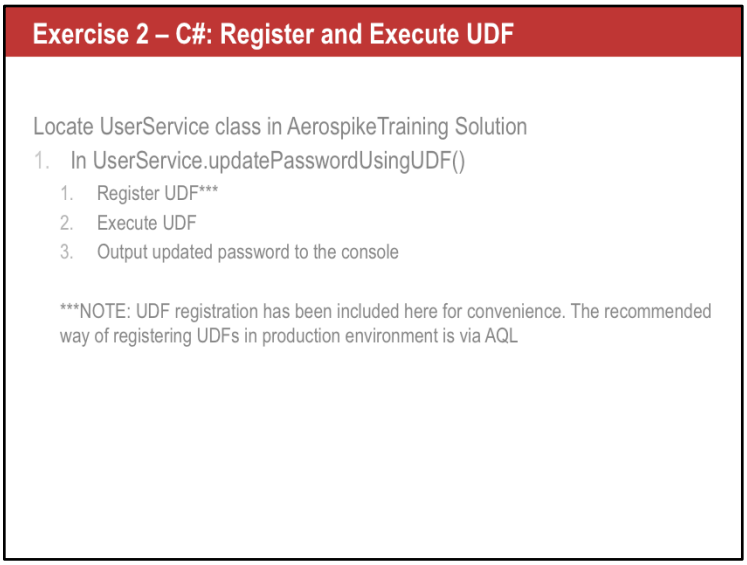
```
LuaConfig.SourceDirectory = "udf";
File udfFile = new File("udf/updateUserPwd.lua");
RegisterTask rt = client.register(null, udfFile.getPath(),
                                  udfFile.getName(), Language.LUA);
rt.waitTillComplete(100);
```

2. Execute the UDF passing the new password, as a parameter, to the UDF

```
String updatedPassword = (String) client.execute(null, userKey,
                    "updateUserPwd", "updatePassword",
                         Value.get(password));
```

3. Output the return from the UDF to the console

```
console.printf("\nINFO: The password has been set to: " + updatedPassword);
```



In this exercise you will register and invoke the UDF created in Exercise 1.

We will programma cally register the UDF for convenience.

In UserService.updatePasswordUsing UDF(), locate these comments and add your code:

1. Register the UDF with an API call

```
string luaDirectory = @"..\..\udf";
LuaConfig.PackagePath = luaDirectory + @"\?.lua";
string filename = "updateUserPwd.lua";
string path = Path.Combine(luaDirectory, filename);
RegisterTask rt = client.Register(null, path, filename, Language.LUA);
rt.Wait();
```

2. Execute the UDF passing the new password, as a parameter, to the UDF

```
string updatedPassword = client.Execute(null, userKey, "updateUserPwd", "updatePassword",
Value.Get(password)).ToString();
```

3. Output the return from the UDF to the console

```
Console.WriteLine("\nINFO: The password has been set to: " + updatedPassword);
```

Exercise 2 – PHP: Register and Execute UDF

Locate UserService class

1. In UserService.updatePasswordUsingUDF()
   1. Ensure the UDF module is registered
   2. Execute UDF

NOTE: UDF registration has been included here for convenience. The recommended way of registering UDFs in production environment is via AQL

In this exercise you will register and invoke the UDF created in Exercise 1.

We will programma cally register the UDF for convenience.

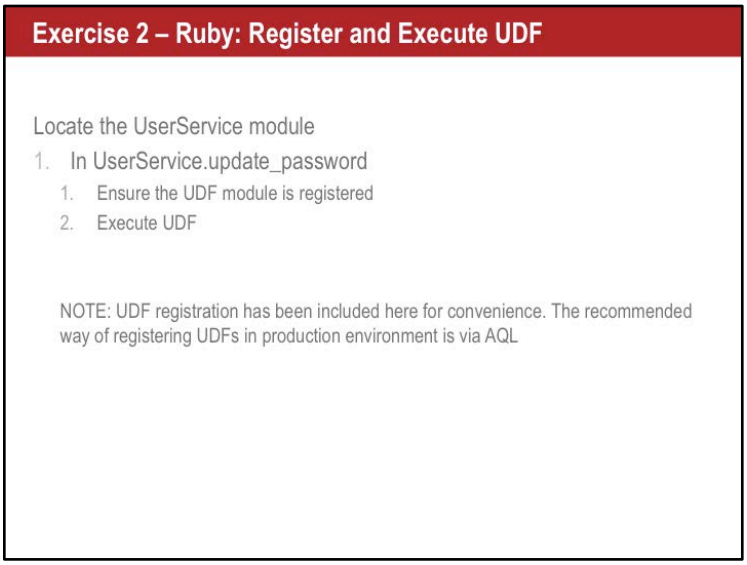
In UserService.updatePasswordUsing UDF(), locate these comments and add your code:

1. Ensure the UDF module is registered

```
$ok = $this->ensureUdfModule('udf/updateUserPwd.lua',
'updateUserPwd.lua');
if ($ok) echo success();
else echo fail();
$this->display_module('udf/updateUserPwd.lua');
```

2. Execute the UDF passing the new password as a parameter, to the UDF

```
echo colorize("Updating the user record >", 'black', true);
$key = $this->getUserKey($username);
if ($this->annotate) display_code(__FILE__, __LINE__, 7);
$args = array($new_password);
$status = $this->client->apply($key, 'updateUserPwd',
'updatePassword', $args);
if ($status !== Aerospike::OK) {
    // throwing an \Aerospike\Training\Exception
    echo fail();
    throw new Exception($this->client, "Failed to update
password for user $username");
}
```



In this exercise you will register and invoke the UDF created in Exercise 1.

We will programma cally register the UDF for convenience.

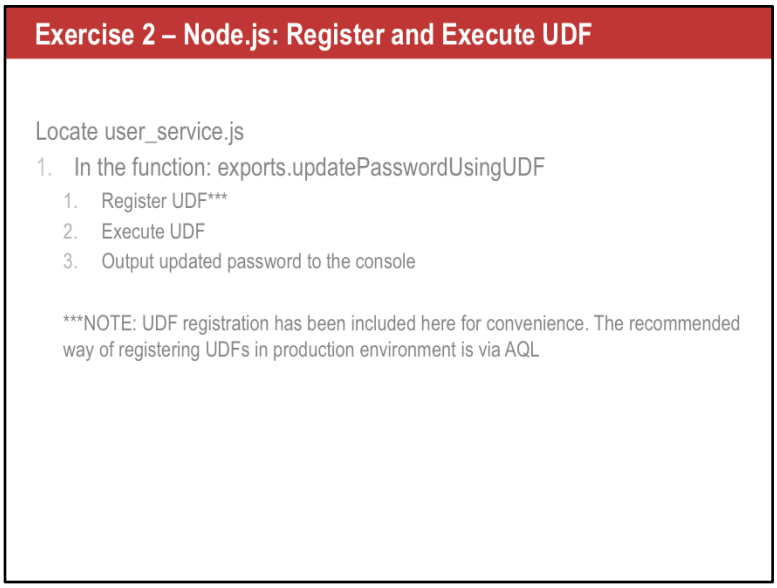
In UserService.update_password, locate these comments and add your code:

1. Ensure the UDF module is registered

```
task = client.register_udf_from_file(module_path,
module_name, Language::LUA)
task.wait_till_completed
```

2. Execute the UDF passing the new password as a parameter, to the UDF

```
client.execute_udf(key, "updateUserPwd", "updatePassword",  [new_password])
```



In this exercise you will register and invoke the UDF created in Exercise 1.

We will programma cally register the UDF for convenience.

In the funcNon module.updatePasswordUsingUDF add your code:

1. Register the UDF with an API call

```
// Register UDF
client.udfRegister('udfs/updateUserPwd.lua', function(err) {
  if ( err.code === 0 ) {
    var UDF = {module:'updateUserPwd', funcname:
      'updatePassword', args: [answers.password]};
    var key = {
      ns:  "test",
      set: "users",
      key: answers.username
    };
```

2. Execute the UDF passing the new password, as a parameter, to the UDF

```
    client.execute(key, UDF, function(err) {
      // Check for errors
      if ( err.code === 0 ) {
           console.log("INFO: User password updated successfully!");
      }
      else {
           console.log("ERROR: User password update failed\n", err);
      }
    });
```

3. Output the return from the UDF to the console

```
  } else {
    // An error occurred
     console.error("ERROR: updateUserPwd UDF registeration failed:\n", err);
  }
});
```

Exercise 2 – Go: Register and Execute UDF

Locate tweetaspike.go

1. In the function: UpdatePasswordUsingUDF()
   1. Register UDF***
   2. Execute UDF
   3. Output updated password to the console

***NOTE: UDF registration has been included here for convenience. The recommended way of registering UDFs in production environment is via AQL

In this exercise you will register and invoke the UDF created in Exercise 1.

We will programma cally register the UDF for convenience.

In UpdatePasswordUsingUDF(), locate these comments and add your code:

1. Register the UDF with an API call

```
regTask, err := client.RegisterUDFFromFile(nil, "udf/updateUserPwd.lua",
                        "updateUserPwd.lua", LUA)
panicOnError(err)
// wait until UDF is created
for {
     if err := <-regTask.OnComplete(); err == nil {
          break
     }
}
```

2. Execute the UDF passing the new password, as a parameter, to the UDF

```
updatedPassword, err := client.Execute(nil, userKey, "updateUserPwd",
                                        "updatePassword", NewValue(password))
panicOnError(err)
```

3. Output the return from the UDF to the console

```
fmt.Printf("\nINFO: The password has been set to: %s\n", updatedPassword)
```

Summary

You have learned:

- Code a record UDF
- Register the UDF module
- Invoke a record UDF


Lab: Queries

Objectives

After successful completion of this Lab module you will have:

- Created a secondary index
- Prepared a statement
- Executed a query
- Processed the results

Lab Overview

The Lab exercises augment the"tweetaspike" application by allowing us to:

1) Query Tweets for a given username
2) Query users based on number of Tweets

The application shell is located in your cloned GitHub directory

`~/exercises/Queries/<language>`

Make sure you have your server up and you know its **IP address**

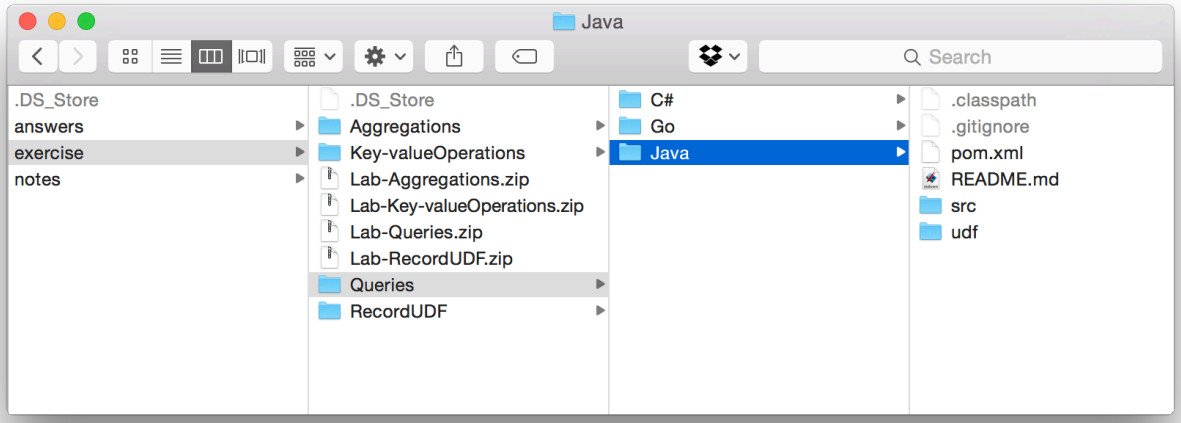
On your cloned or downloaded repository, you will find the following directories:

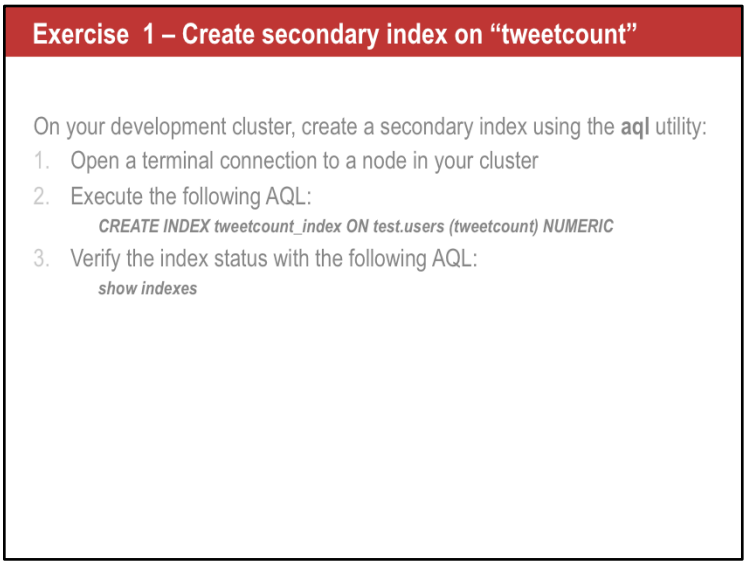
- Answers
- Exercise
- Notes

In the exercise directory, select the subdirectory for your programming language:

- C#
- Java
- Go
- Node.js
- PHP
- Python

The exercises for this module are in the Queries directory and your will find a Project/SoluNon/ Codebase that is partly complete. Your tasks is to complete the code as outlined in each exercise.



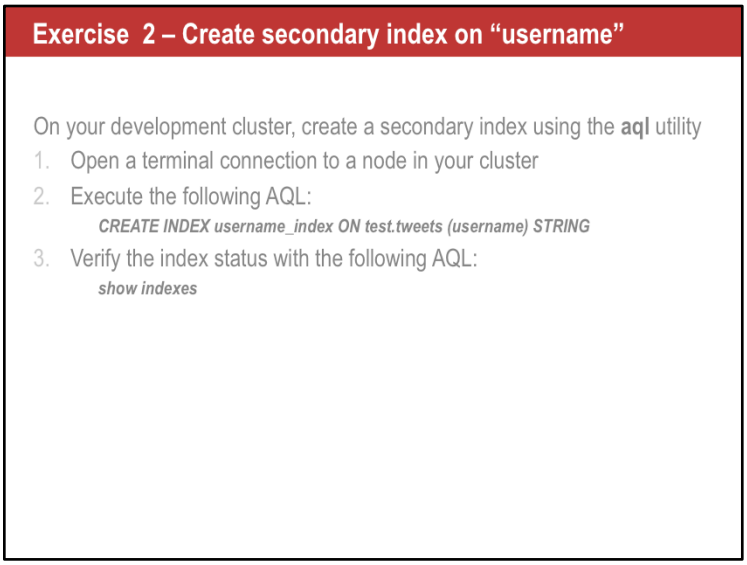


Logon on to your server instance and run **aql** to create a numeric index on *tweetcount*.
At the prompt, enter the command:

CREATE INDEX tweetcount_index ON test.users (tweetcount) NUMERIC

Verify that the index has been created with the command:

show indexes



Logon on to your server instance and run **aql** to create a string index on *username*. At the prompt, enter the command:

CREATE INDEX username_index ON test.tweets (username) STRING

Verify that the index has been created with the command:

show indexes


Java Exercises

Exercise 3 – Java: Query tweets for a given username

Locate class TweetService in the Maven project

In TweetService.queryTweetsByUsername():

1. Create String array of bins to retrieve. In this example, we want to display tweets for a given user.
2. Create Statement instance. On this Statement instance:
   1. Set namespace
   2. Set name of the set
   3. Set array of bins to retrieve
   4. Set equality Filter for username
3. Execute query passing in null policy and instance of Statement created above
4. Iterate through returned RecordSet and output tweets to the console

In TweetService.queryTweetsByUsername(), locate these comments and add your code:

1. Create a list of Bins to retrieve
2. Create a statement
   1. Set the Namespace
   2. Set the Set name
   3. Set the index name (opNonal)
   4. Set the array of bins (from above)
   5. Set the Filter to qualify the user name
3. Execute the query from your code
4. Iterate through the RecordSet returned from the query
5. Close the record set

```
if (username != null && username.length() > 0) {
     String[] bins = { "tweet" };

     Statement stmt = new Statement();
     stmt.setNamespace("test");
     stmt.setSetName("tweets");
     stmt.setIndexName("username_index");
     stmt.setBinNames(bins);
     stmt.setFilters(Filter.equal("username", username));

     console.printf("\nHere's " + username + "'s tweet(s):\n");

     rs = client.query(null, stmt);
     while (rs.next()) {
          Record r = rs.getRecord();
          console.printf(r.getValue("tweet").toString() + "\n");
     }
} else {
     console.printf("ERROR: User record not found!\n");
}
} finally {
     if (rs != null) {
          // Close record set
          rs.close();
```

Exercise 4 – Java: Query users based on number of tweets

Locate class TweetService in AerospikeTraining Solution

In TweetService.queryUsersByTweetCount():

1. Create String array of bins to retrieve. In this example, we want to output which user has how many tweets.
2. Create Statement instance. On this Statement instance:
    1. Set namespace
    2. Set name of the set
    3. Set array of bins to retrieve
    4. Set range Filter for min--max tweetcount
3. Execute query passing in null policy and instance of Statement created above
4. Iterate through returned RecordSet and for each record, output text in format "<username> has <#> tweets"

In TweetService.queryUsersByTweetCount(), locate these comments and add your code:

1. Create String array of bins to retrieve. In this example, we want to output which user has how many tweets.

```
String[] bins = { "username", "tweetcount", "gender" };
```

2. Create Statement instance. On this Statement instance:
    1. Set namespace
    2. Set name of the set
    3. Set name of the index
    4. Set array of bins to retrieve
    5. Set range Filter for min max tweetcount

```
Statement stmt = new Statement();
stmt.setNamespace("test");
stmt.setSetName("users");
stmt.setIndexName("tweetcount_index");
stmt.setBinNames(bins);
stmt.setFilters(Filter.range("tweetcount", min, max));
```

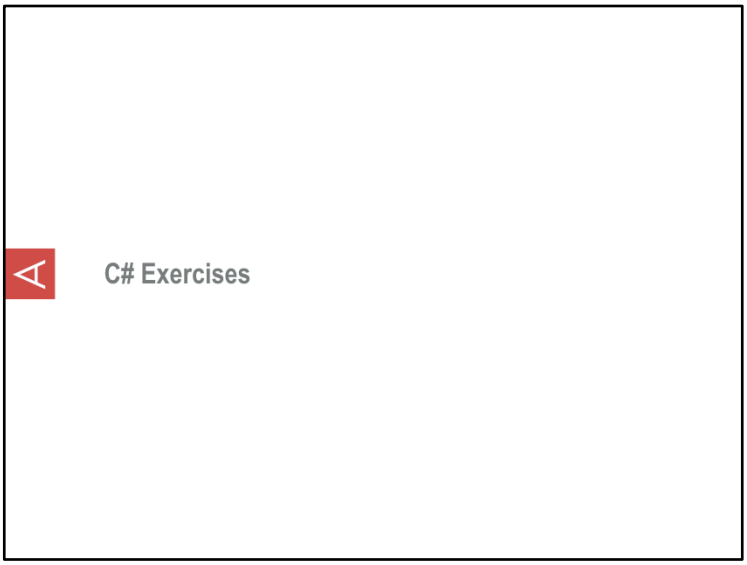
3. Execute query passing in null policy and instance of Statement created above

```
rs = client.query(null, stmt);
```

4. Iterate through returned RecordSet and for each record, output text in format "<username> has <#> tweets"

```
while (rs.next()) {
      Record r = rs.getRecord();
      console.printf(r.getValue("username") + " has "
                  + r.getValue("tweetcount") + " tweets\n");
}
} finally {
      if (rs != null) {
            // Close record set
            rs.close();
```

5. Close the RecordSet


C# Exercises

Exercise 3 – C#: Query tweets for a given username

Locate class TweetService in AerospikeTraining Solution

In TweetService.queryTweetsByUsername():

1. Create String array of bins to retrieve. In this example, we want to display tweets for a given user.
2. Create Statement instance. On this Statement instance:
   1. Set namespace
   2. Set name of the set
   3. Set name of the index
   4. Set array of bins to retrieve
   5. Set equality Filter for username
3. Execute query passing in null policy and instance of Statement created above
4. Iterate through returned RecordSet and output tweets to the console

In TweetService.queryTweetsByUsername(), locate these comments and add your code:

1. Create a list of Bins to retrieve
2. Create a statement
   1. Set the Namespace
   2. Set the Set name
   3. Set the index name (opNonal)
   4. Set the array of bins (from above)
   5. Set the Filter to qualify the user name
3. Execute the query from your code
4. Iterate through the RecordSet returned from the query
5. Close the record set

```
if (username != null && username.Length > 0)
{
    string[] bins = { "tweet" };
    Statement stmt = new Statement();
    stmt.SetNamespace("test");
    stmt.SetSetName("tweets");
    stmt.SetIndexName("username_index");
    stmt.SetBinNames(bins);
    stmt.SetFilters(Filter.Equal("username", username));

    Console.WriteLine("\nHere's " + username + "'s tweet(s):\n");

    rs = client.Query(null, stmt);
    while (rs.Next())
    {
        Record r = rs.Record;
        Console.WriteLine(r.GetValue("tweet"));
    }
}
else
{
    Console.WriteLine("ERROR: User record not found!");
}
}
finally
{
    if (rs != null)
    {
        // Close record set
        rs.Close();
```

Exercise 4 – C#: Query users based on number of tweets

Locate class TweetService in AerospikeTraining Solution

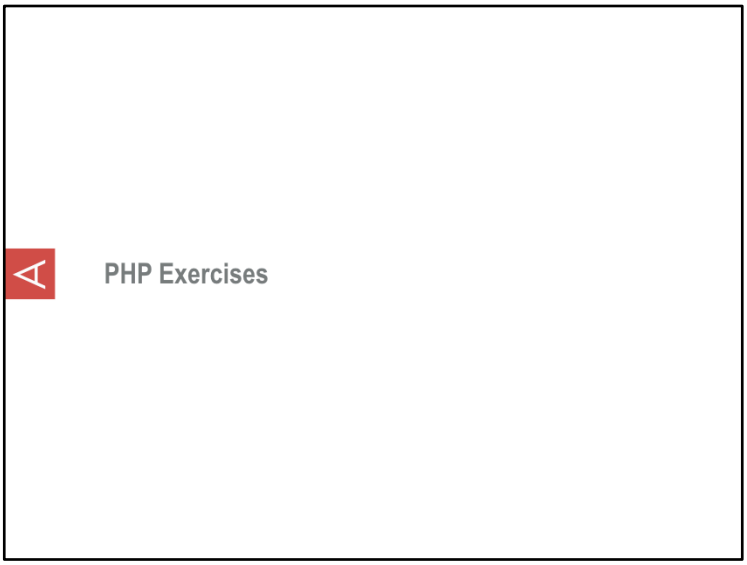
In TweetService.queryUsersByTweetCount():

1. Create String array of bins to retrieve. In this example, we want to output which user has how many Tweets.
2. Create Statement instance. On this Statement instance:
    1. Set namespace
    2. Set name of the set
    3. Set name of the index
    4. Set array of bins to retrieve
    5. Set range Filter for min--max tweetcount
3. Execute query passing in null policy and instance of Statement created above
4. Iterate through returned RecordSet and for each record, output text in format "<username> has <#> tweets"

In TweetService.queryUsersByTweetCount(), locate these comments and add your code:

1. Create String array of bins to retrieve. In this example, we want to output which user has how many tweets.
2. Create Statement instance. On this Statement instance:
    1. Set namespace
    2. Set name of the set
    3. Set name of the index
    4. Set array of bins to retrieve
    5. Set range Filter for min max tweetcount
3. Execute query passing in null policy and instance of Statement created above
4. Iterate through returned RecordSet and for each record, output text in format "<username> has <#> tweets"
5. Close the RecordSet

```
string[] bins = { "username", "tweetcount" };


Statement stmt = new Statement();
stmt.SetNamespace("test");
stmt.SetSetName("users");
stmt.SetIndexName("tweetcount_index");
stmt.SetBinNames(bins);
stmt.SetFilters(Filter.Range("tweetcount", min, max));

Console.WriteLine("\nList of users with " + min
                        + "-" + max + " tweets:\n");

rs = client.Query(null, stmt);

while (rs.Next())
{
    Record r = rs.Record;
    Console.WriteLine(r.GetValue("username")
       + " has " + r.GetValue("tweetcount") + " tweets");
}
}
finally
{
    if (rs != null)
    {
     rs.Close();
```


PHP Exercises

Exercise 3 – PHP: Query tweets for a given username

Locate class TweetService in AerospikeTraining Solution

In TweetService.queryTweetsByUsername():

1. Create a Filter predicate
2. Execute a query using:
    1. Namespace
    2. name of the set
    3. Filter for username

In TweetService.queryTweetsByUsername() add your code:

1. Create a Filter predicate
2. Execute a query using:
    1. the Namespace
    2. the Set name
    3. the Filter predicate to qualify the user name

```
$where = $this->client->predicateEquals('username', $username);
$status = $this->client->query('test','tweets', $where, function
($record) {
    var_dump($record['bins']['tweet']);
}, array('tweet'));
if ($status !== Aerospike::OK) {
    // throwing an \Aerospike\Training\Exception
    throw new Exception($this->client, "Failed to query
test.tweets");
}
```

Exercise 4 – PHP: Query users based on number of tweets

Locate class TweetService in the PHP project

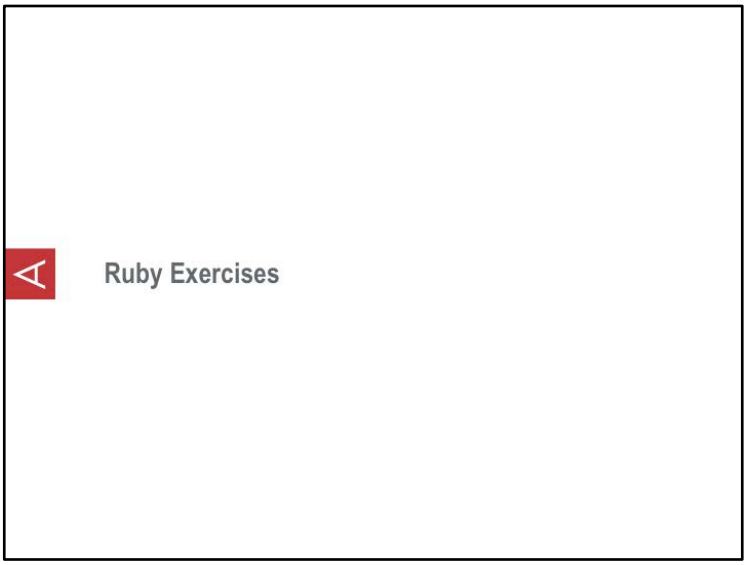
In TweetService.queryUsersByTweetCount():

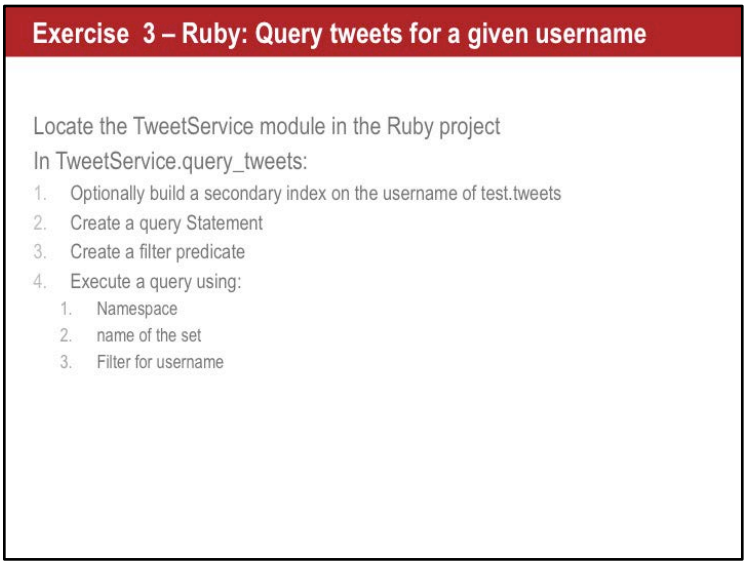
1. Create a range filter predicate for min--max tweetcount.
2. Execute the query using:
    1. The namespace
    2. The set
    3. The range filter predicate
3. Execute query passing in null policy and instance of Statement created above
4. Iterate through returned RecordSet and for each record, output text in format "<username> has <#> tweets"

In TweetService.queryUsersByTweetCount(), add your code:

1. Create a range filter predicate for min max tweetcount.
2. Execute the query using:
    1. The namespace
    2. The set
    3. The range filter predicate
3. Execute query passing in null policy and instance of Statement created above
4. Iterate through returned RecordSet and for each record, output text in format "<username> has <#> tweets"

```
$where = $this->client->predicateBetween('tweetcount', $min,
$max);
$status = $this->client->query('test','users', $where, function
($rec) {
    echo colorize("{$rec['bins']['username']} has
{$rec['bins']['tweetcount']} tweets", 'black')."\n";
});
if ($status !== Aerospike::OK) {
    // throwing an \Aerospike\Training\Exception
    throw new Exception($this->client, "Failed to query
test.users");
}
```

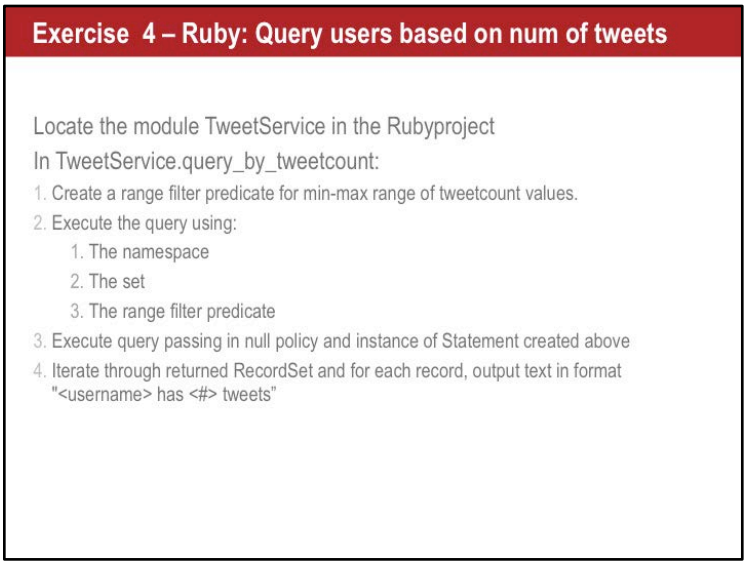

Ruby Exercises



In TweetService.query_tweets addyour code:

1. Create a Filter predicate
2. Execute a query using:
    1. the Namespace
    2. the Set name
    3. the Filter predicate to qualify the user name

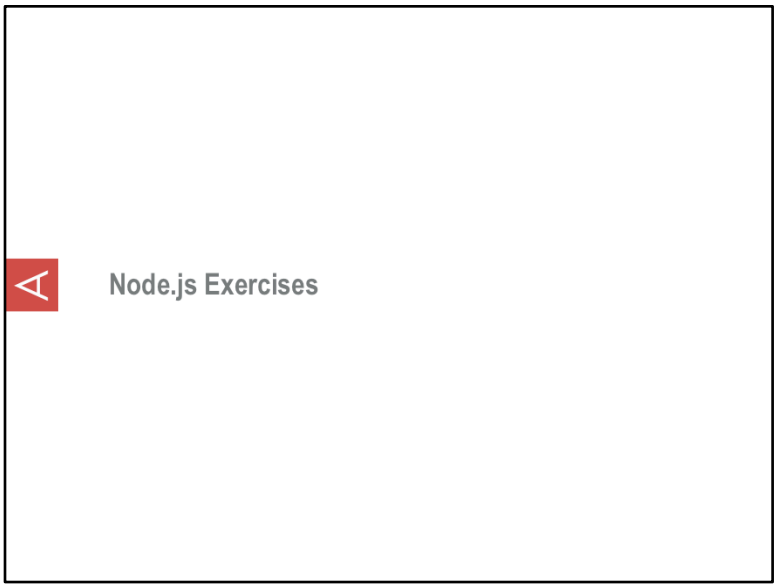
```
statement = Statement.new('test', 'tweets', ['tweet'])
statement.filters << Filter.Equal('username', username)
results = client.query(statement)
```



In TweetService.query_by_tweet_count, add your code:

1. Create a range filter predicate for min-max tweetcount.
2. Execute the query using:
    1. The namespace
    2. The set
    3. The range filter predicate
3. Execute query passing in null policy and instance of Statement created above
4. Iterate through returned RecordSet and for each record, output text in format "<username> has <#> tweets"

```
statement = Statement.new('test', 'users', ['username', 'tweetcount'])
statement.filters << Filter.Range('tweetcount', min, max) unless min == 0
&& max == 0
results = client.query(statement)
```


Node.js Exercises

Exercise 3 – Node.js: Query tweets for a given username

Locate tweet_service.js

In tweet_service.js modify the function exports.queryTweetsByUsernamequeryTweetsByUsername(:

1. Create String array of bins to retrieve. In this example, we want to display tweets for a given user.
2. Create Statement instance. On this Statement instance:
   1. Set namespace
   2. Set name of the set
   3. Set equality Filter for username
3. Execute query passing in null policy and instance of Statement created above
4. Process the streamand output tweets to the console

In the funcNon:
queryTweetsByUsername, add your code:

1. Create a statement with
   1. the Namespace
   2. the Set name
   3. the bins (“tweet”)
   4. Set the Filter to qualify the user name

```
var statement = {filters:[aerospike.filter.equal('username',
answer.username)]};
```

2. Execute the query from your code

```
var query = client.query('test', 'tweets', statement);
var stream = query.execute();
```

3. Iterate through the RecordSet returned from the query

```
stream.on('data', function(record)  {
  console.log(record.bins.tweet);
});
stream.on('error', function(err)  {
  console.log('ERROR: Query Tweets By Username failed: ',err);
});
stream.on('end', function()  {
  // console.log('INFO: Query Tweets By Username completed!');
  queryUsersByTweetCount(client);
});
```

Exercise 4 – Node.js: Query users based on tweets

Locate tweet_service.js

In the function: queryUsersByTweetCount:

1. Create Statement instance. On this Statement instance:
   1. Set namespace
   2. Set name of the set
   3. Set array of bins to retrieve
   4. Set range Filter for min--max tweetcount
2. Execute query passing in null policy and instance of Statement created above
3. Process the stream and output text in format "<username> has <#> tweets"

In the funcNon exports.queryUsersByTweetCount, add your code:

1. Create Statement with:
   1. the namespace
   2. the Set
   3. the bins to retrieve
   4. a range Filter for min max tweetcount

```
var statement = {filters:[aerospike.filter.range('tweetcount',
answers.min, answers.max)]};
statement.select = ['username', 'tweetcount'];
```

2. Execute query passing in null policy and instance of Statement created above

```
var query = client.query('test', 'users', statement);
var stream = query.execute();
stream.on('data', function(record)  {
  console.log(record.bins.username + ' === ' +
record.bins.tweetcount);
});
```

3. Process the stream and output text in format "<username> has <#> tweets"

```
stream.on('error', function(err)  {
  console.log('ERROR: Query Users By Tweet Count Range
failed:\n',err);
});
stream.on('end', function()  {
  // console.log('INFO: Query Users By Tweet Count Range
completed!');
});
```


Go Exercises

Exercise 3 – Go: Query tweets for a given username

Locate tweetaspike.go

In queryTweetsByUsername():

1. Create String array of bins to retrieve. In this example, we want to display tweets for a given user.
2. Create Statement instance. On this Statement instance:
    1. Set namespace
    2. Set name of the set
    3. Set name of the index
    4. Set array of bins to retrieve
    5. Set equality Filter for username
3. Execute query passing in null policy and instance of Statement created above
4. Iterate through returned RecordSet and output tweets to the console

In the funcNon: queryTweetsByUsername(), locate these comments and add your code:

1. Create a statement with
    1. the Namespace
    2. the Set name
    3. the bins (“tweet”)
    4. Set the Filter to qualify the user name
2. Execute the query from your code
3. Iterate through the RecordSet returned from the query
4. Close the record set

```
if len(username) > 0 {
     stmt := NewStatement("test", "tweets", "tweet")
     stmt.Addfilter(NewEqualFilter("username", username))

     fmt.Printf("\nHere's " + username + "'s tweet(s):\n")


     recordset, err := client.Query(nil, stmt)
     panicOnError(err)
L:
     for {
          select {
          case rec, chanOpen := <-recordset.Records:
               if !chanOpen {
                    break L
               }
               fmt.Println(rec.Bins["tweet"])
          case err := <-recordset.Errors:
               panicOnError(err)
          }
     }
     recordset.Close()

} else {
     fmt.Printf("ERROR: User record not found!\n")
}
```

Exercise 4 – Go: Query users based on number of tweets

Locate tweetaspike.go

In the function: queryUsersByTweetCount():

1. Create String array of bins to retrieve. In this example, we want to output which user has how many tweets.
2. Create Statement instance. On this Statement instance:
   1. Set namespace
   2. Set name of the set
   3. Set name of the index
   4. Set array of bins to retrieve
   5. Set range Filter for min--max tweetcount
3. Execute query passing in null policy and instance of Statement created above
4. Iterate through returned RecordSet and for each record, output text in format "<username> has <#> tweets"

In queryUsersByTweetCount(), locate these comments and add your code:

1. Create Statement with:
   1. the namespace
   2. the Set
   3. the bins to retrieve
   4. a range Filter for min max tweetcount

```
stmt := NewStatement("test", "users",
            "username", "tweetcount", "gender")
stmt.Addfilter(NewRangeFilter("tweetcount", min, max))
```

2. Execute query passing in null policy and instance of Statement created above

```
recordset, err := client.Query(nil, stmt)
panicOnError(err)
```

3. Iterate through returned RecordSet and for each record, output text in format "<username> has <#> tweets"

```
L:
for {
      select {
      case rec, chanOpen := <-recordset.Records:
            if !chanOpen {
                  break L
            }
            fmt.Printf("%s has %d tweets\n", rec.Bins["username"],
                                                rec.Bins["tweetcount"])
      case err := <-recordset.Errors:
            panicOnError(err)
      }
}
```

4. Close the RecordSet

```
recordset.Close()
```

Summary

You have learned:

- How to create a secondary index
- How to create a Statement
- Execute a query on a secondary index
- Process the results from a query


Lab: Aggregations

Objective

After successful completion of this Lab module you will have:

- Coded a Stream UDF
- Register the UDF with a cluster
- Executed Aggregation from your C#, PHP, Node.js or Java application

Lab Overview

The lab exercise augments “tweetaspike” by using a Stream UDF.
Here we will create a Stream UDF that aggregates number of users with tweet count between min-max range by region – North, South, East and West .

The application shell is located in your cloned GitHub directory

```
~/exercises/Aggregations/<language>
```

Make sure you have your server up and you know its IP address

In your cloned or downloaded repository, you will find the following directories:

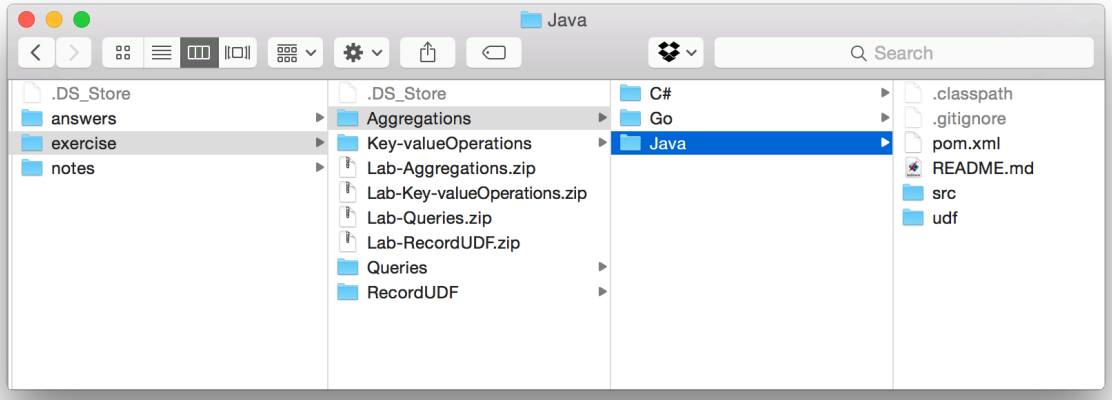
- Answers
- Exercise
- Notes

In the exercise directory, select the subdirectory for your programming language:

- C#
- Java
- Node.js
- PHP
- Python

The exercises for this module are in the Aggrega ons directory and your will find a Project/SoluNon/ Codebase that is partly complete. Your tasks is to complete the code as outlined in each exercise.

Make sure you have your server up and you know its IP address



Exercise 1 – Write Stream UDF

Locate aggregationByRegion.lua file under udf folder in AerospikeTraining Solution

1. Code main function 'sum' to process incoming record stream and pass it to aggregate function 'aggregate_stats', then to reduce function 'reduce_stats'
2. Code aggregate function 'aggregate_stats' to examine value of 'region' bin and increment respective counters
3. Code reduce function 'reduce_stats' to merge maps

In this exercise you will create a Stream UDF module that:

- Aggregates (sums) tweets by region – The aggregate_stats() funcNon is invoked one for each element in the stream.
- Reduces the aggrega ons into a single Map of values – The reduce_stats() funcNon is invoked once for each data parNNon, once for each node in the cluster, and finally once on the client.
- The sum() funcNon configures the stream processing, and it is the funcNon invoked by the Client.

```
local function aggregate_stats(map,rec)
  -- Examine value of 'region' bin in record rec and increment respective counter in the map
  if rec.region == 'n' then
      map['n'] = map['n'] + 1
  elseif rec.region == 's' then
      map['s'] = map['s'] + 1
  elseif rec.region == 'e' then
      map['e'] = map['e'] + 1
  elseif rec.region == 'w' then
      map['w'] = map['w'] + 1
  end
  -- return updated map
  return map
end
local function reduce_stats(a,b)
  -- Merge values from map b into a
  a.n = a.n + b.n
  a.s = a.s + b.s
  a.e = a.e + b.e
  a.w = a.w + b.w
  -- Return updated map a
  return a
end
function sum(stream)
  -- Process incoming record stream and pass it to aggregate function, then to reduce function
return stream : aggregate(map{n=0,s=0,e=0,w=0},aggregate_stats) : reduce(reduce_stats)
end
```

Exercise 2 – Java: Register and Execute UDF

Locate UserService class in the Maven project

In UserService.aggregateUsersByTweetCountByRegion()

1. Register UDF
2. Create String array of bins to retrieve. In this example, we want to display total users that have tweets between min-max range by region.
3. Create Statement instance. On this Statement instance:
    1. Set namespace
    2. Set name of the set
    3. Set array of bins to retrieve
    4. Set min--max range Filter on tweetcount
4. Execute aggregate query passing in the Statement, UDF module and function name.
5. Examine returned ResultSet and output result to the console in format "Total Users in <region>: <#>"

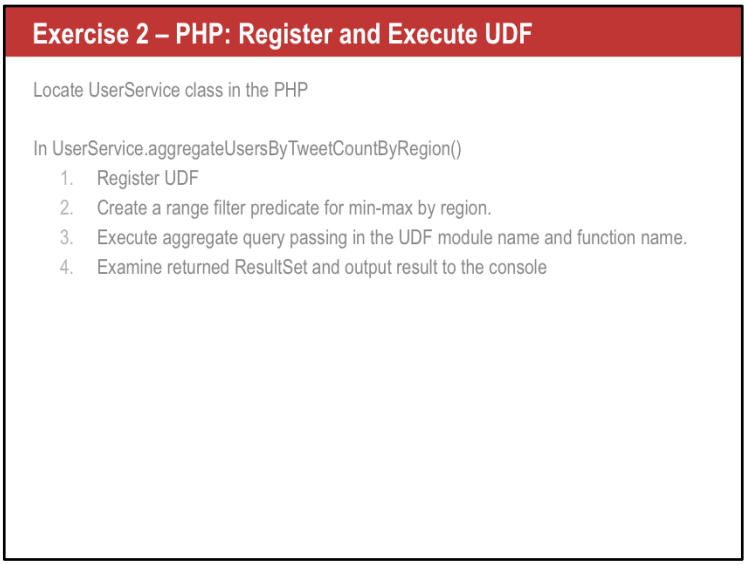
In this exercise you will register and invoke the UDF created in Exercise 1.

We will programma cally register the UDF for convenience .

In UserService.aggregateUsersByTweetCountByRegion(), add your code to look like this:

1. Register the UDF with an API call
2. Create the Bin array
3. Prepare the Statement
4. Execute the aggrega on
5. Examine the ResultSet
6. Close the result set

```
// NOTE: UDF registration has been included here for convenience and to demonstrate the syntax.
// The recommended way of registering UDFs in production env is via AQL
      LuaConfig.SourceDirectory = "udf";
      File udfFile = new File("udf/aggregationByRegion.lua");

      RegisterTask rt = client.register(null, udfFile.getPath(),
      udfFile.getName(), Language.LUA);
      rt.waitTillComplete(100);

      String[] bins = { "tweetcount", "region" };
      Statement stmt = new Statement();
      stmt.setNamespace("test");
      stmt.setSetName("users");
      stmt.setBinNames(bins);
      stmt.setFilters(Filter.range("tweetcount", min, max));
      rs = client.queryAggregate(null, stmt, "aggregationByRegion", "sum");
      if (rs.next()) {
            Map<Object, Object> result = (Map<Object, Object>) rs
                        .getObject();
            console.printf("\nTotal Users in North: " + result.get("n") + "\n");
            console.printf("Total Users in South: " + result.get("s") + "\n");
            console.printf("Total Users in East: " + result.get("e") + "\n");
            console.printf("Total Users in West: " + result.get("w") + "\n");
      }
} finally {
      if (rs != null) {
            // Close result set
            rs.close();
      }
```



In this exercise you will register and invoke the UDF created in Exercise 1.

We will programma cally register the UDF for.

In UserService.aggregateUsersByTweetCountByRegion(), add your code to look like this:

1. Register the UDF with an API call

```
$ok = $this->ensureUdfModule('udf/aggregationByRegion.lua',
'aggregationByRegion.lua');
```

2. Create a range filter predicate

```
$where = $this->client->predicateBetween('tweetcount',
$min, $max);
$args = array();
echo colorize("Call the aggregation stream UDF >", 'black',
true);
```

3. Execute the aggrega on

```
$status = $this->client->aggregate('test', 'users', $where,
'aggregationByRegion', 'sum', $args, $returned);
if ($status !== Aerospike::OK) {
    echo fail();
    // throwing an \Aerospike\Training\Exception
    throw new Exception($this->client, "Failed to execute
the stream UDF");
}
```

4. Examine the ResultSet

```
echo success();
var_dump($returned);
```

Exercise 2 – C#: Register and Execute UDF

Locate UserService class in AerospikeTraining Solution

In UserService.aggregateUsersByTweetCountByRegion()

1. Register UDF
2. Create String array of bins to retrieve. In this example, we want to display total users that have tweets between min-max range by region.
3. Create Statement instance. On this Statement instance:
    1. Set namespace
    2. Set name of the set
    3. Set array of bins to retrieve
    4. Set min--max range Filter on tweetcount
4. Execute aggregate query passing in <null> policy and instance of Statement, .lua filename of the UDF and lua function name.
5. Examine returned ResultSet and output result to the console in format "Total Users in <region>: <#>"

In this exercise you will register and invoke the UDF created in Exercise 1.

We will programma cally register the UDF for convenience.

In UserService.aggregateUsersByTweetCountByRegion(), add your code to look like this:

1. Register the UDF with an API call

```
// NOTE: UDF registration has been included here for convenience and to demonstrate
// the syntax. The recommended way of registering UDFs in production env is via AQL
string luaDirectory = @"..\..\udf";
LuaConfig.PackagePath = luaDirectory + @"\?.lua";
string filename = "aggregationByRegion.lua";
string path = Path.Combine(luaDirectory, filename);
RegisterTask rt = client.Register(null, path, filename, Language.LUA);
rt.Wait();
```

2. Create the Bin array
3. Prepare the Statement

```
string[] bins = { "tweetcount", "region" };
Statement stmt = new Statement();
stmt.SetNamespace("test");
stmt.SetSetName("users");
stmt.SetIndexName("tweetcount_index");
stmt.SetBinNames(bins);
stmt.SetFilters(Filter.Range("tweetcount", min, max));

Console.WriteLine("\nAggregating users with " + min + "-" + max + " tweets by region. Hang on...\n");
```

4. Execute the aggrega on

```
rs = client.QueryAggregate(null, stmt, "aggregationByRegion", "sum");
```

5. Examine the ResultSet

```
if (rs.Next())
{
    Dictionary<object, object> result = (Dictionary<object, object>)rs.Object;
    Console.WriteLine("Total Users in North: " + result["n"]);
    Console.WriteLine("Total Users in South: " + result["s"]);
    Console.WriteLine("Total Users in East: " + result["e"]);
    Console.WriteLine("Total Users in West: " + result["w"]);
}
}
finally
{
    if (rs != null)
```

6. Close the result set

Exercise 2 – node.js: Register and Execute UDF

Locate user_service.js

In user_service.js modify the function *exports.aggregateUsersByTweetCountByRegion*

1. Register UDF
2. Create Statement instance. On this Statement instance:
   1. Set namespace
   2. Set name of the set
   3. Set the bins to retrieve
   4. Set min--max range Filter on tweetcount
3. Execute aggregate query passing in <null> policy and instance of Statement, .lua filename of the UDF and lua function name.
4. Process the stream and output result to the console in format "Total Users in <region>: <#>"

In this exercise you will register and invoke the UDF created in Exercise 1.

We will programma cally register the UDF for convenience.

In user_service.js modify the funcNon *exports.aggregateUsersByTweetCountByRegion*, add your code to look like this:

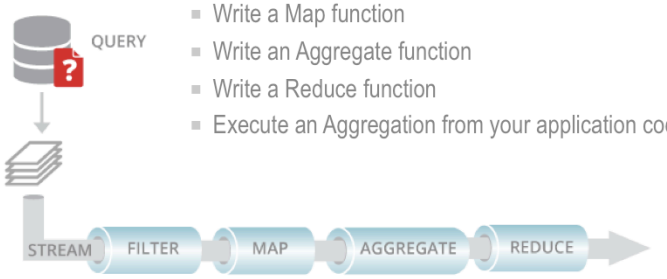
1. Register the UDF with an API call
2. Prepare the Statement
3. Execute the query
4. Process the stream

```
// NOTE: UDF registration has been included in here for convenience and to demonstrate the syntax.
// NOTE: The recommended way of creating indexes in production env is via AQL.
//Register UDF
client.udfRegister('udfs/aggregationByRegion.lua', function(err) {
  if ( err.code === 0 ) {
var statement = {filters:[aerospike.filter.range('tweetcount', answers.min, answers.max)],
aggregationUDF: {module: 'aggregationByRegion', funcname: 'sum'}};
    var query = client.query('test', 'users', statement);
    var stream = query.execute();

    stream.on('data', function(result)  {
      console.log('Total Users In East:  ', result.e);
      console.log('Total Users In West:  ', result.w);
      console.log('Total Users In North: ', result.n);
      console.log('Total Users In South: ', result.s);
    });
    stream.on('error', function(err)  {
      console.log('ERROR: Aggregation Based on Tweet Count By Region failed: ',err);
    });
    stream.on('end', function()  {
      console.log('INFO: Aggregation Based on Tweet Count By Region completed!');
    });
  } else {
    // An error occurred
    console.log('ERROR: aggregationByRegion UDF registeration failed: ', err);
  }
});
```

Summary

You have learned how to:

- Write a Stream UDF
- Write a Filter function
- Write a Map function
- Write an Aggregate function
- Write a Reduce function
- Execute an Aggregation from your application code




AEROSPIKE