

Project #01 – Part 01 (v1.0)

Assignment: PhotoApp with AWS S3 and RDS

Submission: Gradescope (part 02)

Policy: individual work only, late work is accepted

Complete By: Part 01 and 02 by Friday October 6th @ 11:59pm CST

Late submissions: see syllabus for late policy... No submissions accepted after Sunday 10/8 @ 11:59pm

Pre-requisites: assignment “01 Getting Started”

Overview

We’re going to build a cloud-native application **PhotoApp** that works with images, allowing users to upload, download, and manipulate their own images. Here in project 01 we’ll lay the foundation by using AWS + Python to do the following:

1. Use S3 to store images
2. Use RDS (in particular MySQL) to keep track of users and their images
3. Use IAM to create users and policies for access

Note that the goal of this assignment is not to build something novel --- there are tons of apps that work with images. Instead, think of the images as *assets*, which could vary from law documents to large financial datasets, and think of this project as an architectural example of how to work with these assets.

Piazza, not email

There is simply too much email in the world. So please do not use email for communication with the course staff; in general all class-related emails will be ignored. The one exception are matters of a personal nature, in which case you can (and should) email the professor. Appropriate personal matters include family emergencies, classroom issues, and unresolved grading issues.

To provide another means of assistance, we will be using a forum-based web site called **Piazza**. Piazza is your replacement for email. Piazza allows students and staff to help one another, reducing the *time-to-answer*. The staff will check Piazza throughout the day — you should get in the habit of doing the same. Before posting, please search Piazza as the answer to your question may already be online. When posting, please follow these guidelines:

1. Look before you post — the main advantage of Piazza is that common questions are already answered, so search for an existing answer before you post a question. Posts are categorized to help you search, e.g. “HW”.

2. Post publicly — only post privately when asked by the staff, or when it's absolutely necessary (e.g. the question is of a personal nature). Private posts defeat the purpose of piazza, which is answering questions to the benefit of everyone.
3. Be respectful. Anonymous posts are allowed, but note that such posts are *not* anonymous to the staff.
4. Ask pointed questions — do not post a big chunk of code and then ask "help, please fix this". Staff and other students are willing to help, but we aren't going to type in that chunk of code to find the error. You need to narrow down the problem, and ask a pointed question, e.g. "on the 3rd line I get this error, I don't understand what that means..." .
5. Post a screenshot — sometimes a picture captures the essence of your question better than text. Piazza allows the posting of images, so don't hesitate to take a screenshot and post; see <http://www.take-a-screenshot.org/> .
6. Don't post large blocks of code / possible answerws — if you do, you just gave away the answer to the ENTIRE CLASS. Sometimes you will need to post code when asking a question --- in that case post only the fragment that denotes your question, and omit whatever details you can. If you must post a large block of code, then do so privately --- there's an option to create a private post ("visible to staff only").

You should have received an invitation to join Piazza (by email of course). If not, you can join via this [link](#).

Academic Conduct Policy

Northwestern publishes a basic guide to academic integrity, which can be found [here](#). In summary, here are NU's eight cardinal rules of academic integrity:

1. *Know your rights*
2. *Acknowledge your sources*
3. *Protect your work*
4. *Avoid suspicion*
5. *Do you own work*
6. *Never falsify a record or permit another person to do so*
7. *Never fabricate data, citations, or experimental results*
8. *Always tell the truth when discussing your work with your instructor*

School policies and more information can be found on NU's academic integrity [website](#). With regards to CS 211, unless stated otherwise, all work submitted for grading *must* be done individually. While we encourage you to talk and learn from the course staff, peers, and others, this interaction must be superficial with regards to all work submitted for grading. This means you cannot work in teams, you cannot work side-by-side, you cannot submit someone else's work (partial or complete) as your own. The use of AI (ChatGPT, Co-pilot, etc.) is currently forbidden.

Examples of what is not allowed? Downloading work from a github repository and submitting it as your own, whether partial or complete. Downloading answers from StackOverflow and submitting them as your own. Emailing your work to another student, or receiving work from another student. Sharing your screen with another student so they can see your work (and potentially submit it as their own). Participating in a screen share and taking screenshots or photos of someone else's work, and then using that as a guide to submit your own work. Copying answers posted to Piazza, making a few simple changes, and then submitting as your own work. Allowing someone else to write / type the answer for you. Using AI (ChatGPT, Co-pilot, etc.) to generate code for you which you then submit as your own.

Okay, so what is allowed? Talking to the instructor or course staff, and getting insights --- but not direct answers --- to help you solve the assignment. Talking to other students about the assignment, using diagrams / natural language / pseudo-code to convey ideas. Searching the internet for guidance, and using that guidance to help you form your own solution. If you do receive help / guidance, it's always best to cite your source by name or URL just in case there is a question as to where the work came from.

Before you start...

Before you start, you'll need three things. First, you need an AWS account. If you haven't already, you can sign up for a free account [here](#).

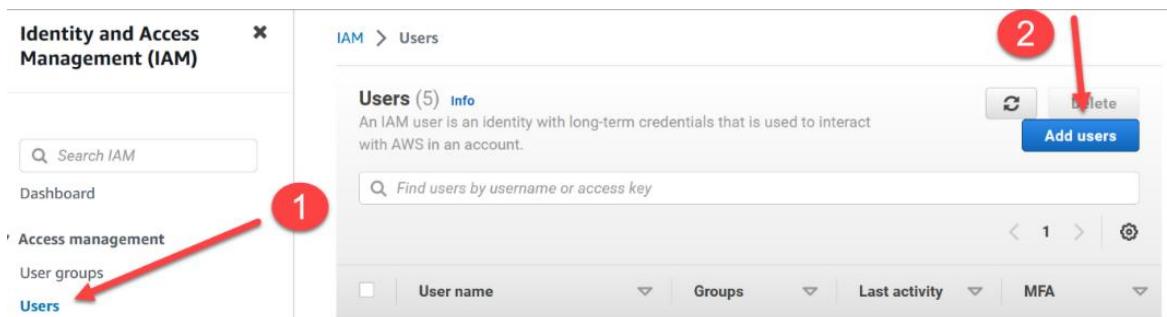
Second, you will need a Python programming environment. You can use whatever IDE you prefer, popular choices include VS Code, PyCharm, and Spyder. If you prefer not to install anything on your local machine, you are welcome to use [replit.com](#) --- a Python replit will be provided for each class project. If you haven't already, you can join our replit team [here](#).

Step 1: Setup AWS CLI

There are various ways to work with AWS. The two most common are logging in to the web site, and using the **Management Console**, Amazon's web-based GUI -----> This implies a visual interface with lots of navigation and button-clicking. Great for exploring and learning, but hard to remember what you did and hard to repeat over and over again. Once you learn your way around, most would agree that a **Command-Line Interface (CLI)** is preferred, especially when you can save those commands into files for repeated execution. For example, once the CLI is setup on your local machine, you can use a command window / terminal to make a new S3 bucket as follows:

```
aws s3 mb s3://new-bucket-nu-cs310
```

To setup your computer for CLI access, start by logging into AWS, opening the Management Console, and search for "IAM" to bring up the Identity and Account Management dashboard. Click on "Users" along the left of the list of users that have access to your AWS account:



Click the "Add users" button, and then continue on the next page...

As discussed in the textbook, create a new user named “mycli” and setup their permissions as shown below:

Specify user details

User details

User name: mycli
The user name can have up to 64 characters. Valid characters: A-Z, a-z, 0-9, and + = , . @ _ - (hyphen)

Provide user access to the AWS Management Console - *optional*
If you're providing console access to a person, it's a [best practice](#) to manage their access in IAM Identity Center.

If you are creating programmatic access through access keys or service-specific credentials for AWS CodeCommit or Amazon Keypairs, you can generate them after you create this IAM user.
[Learn more](#)

Cancel **Next**

Set permissions

Add user to an existing group or create a new one. Using groups is a best-practice way to manage user's permissions by job functions. [Learn more](#)

Permissions options

Add user to group
Add user to an existing group, or create a new group. We recommend using groups to manage user permissions by job function.

Copy permissions
Copy all group memberships, attached managed policies, and inline policies from an existing user.

Attach policies directly
Attach a managed policy directly to a user. As a best practice, we recommend attaching policies to a group instead. Then, add the user to the appropriate group.

Permissions policies (1/1065)
Choose one or more policies to attach to your new user.

Filter distributions by text, property or value

Policy name	Type	Atts
AccessAnalyzerServiceRolePolicy	AWS managed	0
<input checked="" type="checkbox"/> AdministratorAccess	AWS managed ...	6

Now review and create the new user:

Review and create

Review your choices. After you create the user, you can view and download the autogenerated password, if enabled.

User details

User name: mycli	Console password type: None	Require password reset: No
------------------	-----------------------------	----------------------------

Permissions summary

Name	Type	Used as
AdministratorAccess	AWS managed - job function	Permissions policy

Tags - optional
Tags are key-value pairs you can add to AWS resources to help identify, organize, or search for resources. Choose any tags you want to associate with this user.

No tags associated with the resource.
Add new tag
You can add up to 50 more tags.

Create user

Continued on next page...

Once you create your “mycli” user, they will be listed as one of your users. Click on the name:

The screenshot shows the AWS IAM 'Users' page. At the top, there's a search bar with placeholder text 'Find users by username or access key'. Below it is a table header with columns: 'User name', 'Groups', 'Last activity', and 'MFA'. A red arrow points to the 'User name' column for the 'mycli' row, which has a checked checkbox. The 'mycli' row also shows 'None' under Groups, 'Never' under Last activity, and 'None' under MFA.

Now let's get the user's access and secret key (ignoring best practices for now):

Access key best practices & alternatives
Avoid using long-term credentials like access keys to improve your security. Consider the following use cases and alternatives.

This screenshot shows the 'Security credentials' creation wizard for the 'mycli' user. It lists several options for using access keys:

- Command Line Interface (CLI)
You plan to use this access key to enable the AWS CLI to access your AWS account.
- Local code
You plan to use this access key to enable application code in a local development environment to access your AWS account.
- Application running on an AWS compute service
You plan to use this access key to enable application code running on an AWS compute service like Amazon EC2, Amazon ECS, or AWS Lambda to access your AWS account.
- Third-party service
You plan to use this access key to enable access for a third-party application or service that monitors or manages your AWS resources.
- Application running outside AWS
You plan to use this access key to enable an application running on an on-premises host, or to use a local AWS client or third-party AWS plugin.
- Other
Your use case is not listed here.

A warning message at the bottom says: "Alternatives recommended Use AWS CloudShell, a browser-based CLI, to run commands. Learn more" with a link. It also says: "• Use the AWS CLI V2 and enable authentication through a user in IAM Identity Center. Learn more" with another link. A checkbox at the bottom is checked: "I understand the above recommendation and want to proceed to create an access key." Red arrows point from the text "Ignoring best practices for now" in the previous step to the 'Command Line Interface (CLI)' radio button and the 'I understand...' checkbox.

mycli

This screenshot shows the 'mycli' user details page. It includes a 'Summary' section with ARN, console access status, and access key information. Below it are tabs for 'Permissions', 'Groups', 'Tags', and 'Security credentials', with 'Security credentials' being the active tab. The 'Access Advisor' tab is also visible. A red box highlights the 'Security credentials' tab.

Scroll down to “Create access key”, and create an access key for CLI access:

This screenshot shows the 'Create access key' wizard. It starts with a note about access keys and a 'Create access key' button. Below it is a section titled 'No access keys' with a note about best practices and a 'Create access key' button. A red box highlights the first 'Create access key' button.

After creating the access key, you'll see this dialog --- stop and keep this dialog open.

Now let's install the AWS CLI SDK v2, here's the link:

<https://docs.aws.amazon.com/cli/latest/userguide/install-cliv2.html>

Once installed, open a command window (Windows) or terminal (Mac). Configure your local machine by running "aws configure". Enter the access and secret keys --- note the small buttons on the GUI page to copy each of these to the clipboard. For the region enter "us-east-2", and "json" for the default format. Your local computer should now be setup for CLI access. Test using these commands:

```
aws --version
```

```
aws s3 mb s3://new-bucket-YOUR-NAME-RANDOM-NUMBER
```

The second command creates a bucket; recall that bucket names must be unique in each region, so you need to personalize the bucket name. To confirm that the bucket was created, use the Management Console (search for "S3"), or better yet, use the CLI to list the contents of your S3:

```
aws s3 ls
```

Step 2: Create photoapp bucket

Step 2 is to create a new S3 bucket for your **PhotoApp** application. Using the Management Console (search for "S3"), or use the CLI. The name of your bucket should start with "photoapp" then follow with your name and if necessary a unique identifier of some sort. Example: our staff bucket is "photoapp-nu-cs310".

For testing purposes, we want your bucket to be publicly readable. After creating your bucket, in the Management Console find your bucket and click on the name:

<< next page >>

The screenshot shows the AWS Management Console Buckets page. At the top, there is an 'Account snapshot' section with metrics like Total storage (7.7 MB), Object count (18), and Average object size (440.7 KB). Below this is a table titled 'Buckets (1)'. The table has columns for Name, AWS Region, Access, and Creation date. A single row is shown for the bucket 'photoapp-nu-cs310', which is located in 'US East (Ohio) us-east-2' with 'Public' access. A red arrow points to the 'photoapp-nu-cs310' entry in the table.

Retrieve access keys

This screenshot shows the 'Access key' dialog from the AWS IAM service. It displays two access keys: 'Access key' (with value AKIAWPEBNNUOSVN3DEEUP) and 'Secret access key' (with value obscured by asterisks). There are 'Copy' and 'Show' buttons for each key. Below the keys is a section titled 'Access key best practices' with a bulleted list of security tips. At the bottom are 'Download .csv file' and 'Done' buttons.

Switch to the Permissions tab:

The screenshot shows the AWS S3 console. At the top, it says "Amazon S3 > Buckets > photoapp-nu-cs310". Below that is the bucket name "photoapp-nu-cs310" with an "Info" link. A red box highlights the "Permissions" tab in the navigation bar, which also includes "Objects", "Properties", "Metrics", "Management", and "Access Points".

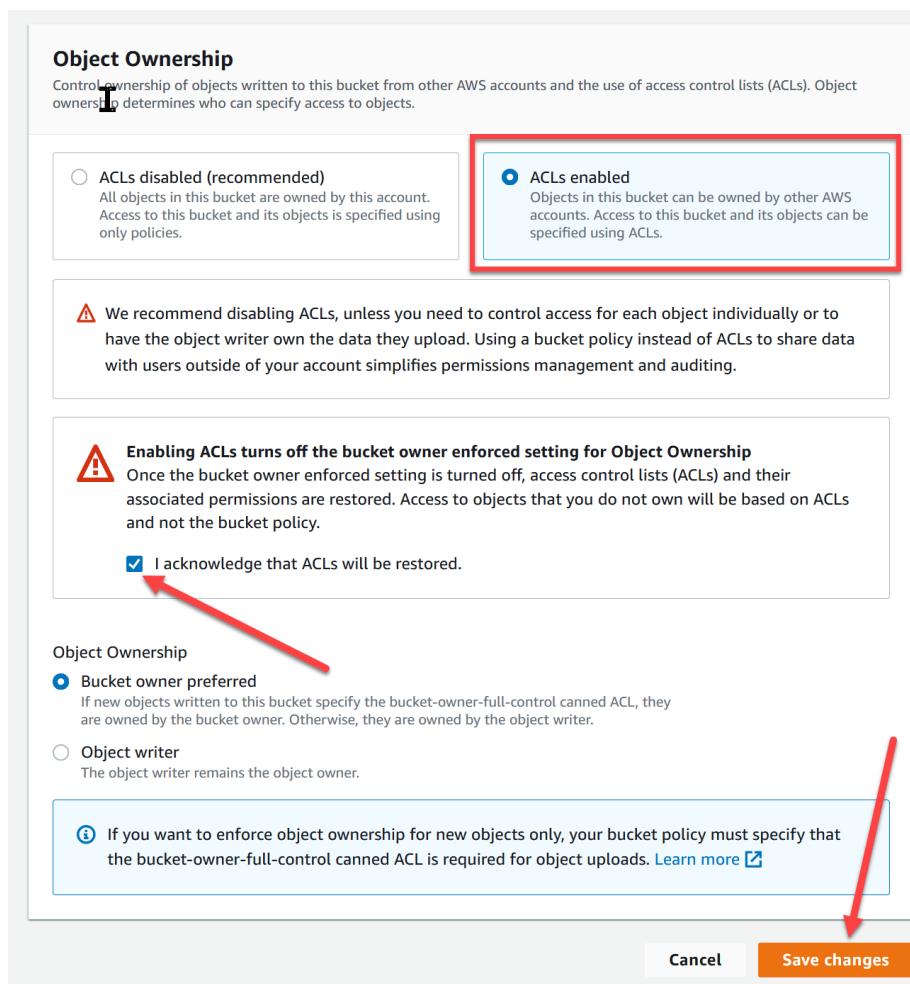
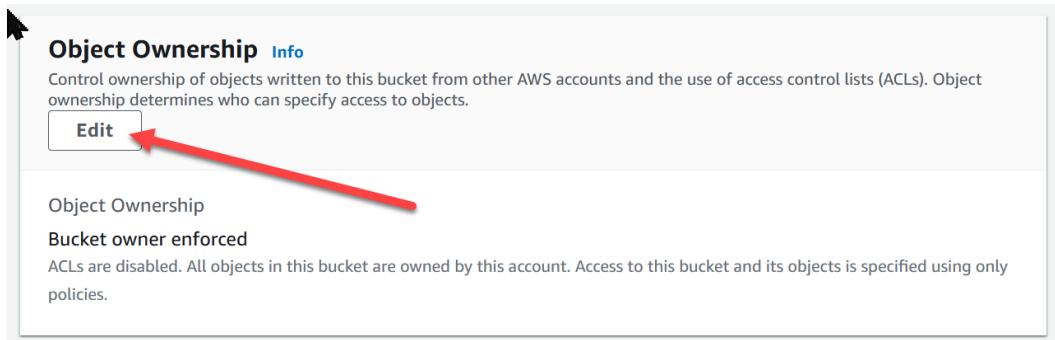
Edit the public access settings:

This screenshot shows the "Block public access (bucket settings)" configuration page. It includes a descriptive text about blocking public access and a "Learn more" link. Below is an "Edit" button with a red arrow pointing to it. Under "Block all public access", there is a green "On" radio button with a red arrow pointing to it, and a link to "Individual Block Public Access settings for this bucket".

Uncheck the option to block all public access, thereby ALLOWING public access, and Save changes:

This screenshot shows the "Block public access" settings dialog. It has a checkbox for "Block all public access" which is unchecked. A red arrow points to this checkbox. Below it is a detailed description of what turning on this setting does. There are four additional checkboxes listed: "Block public access to buckets and objects granted through new access control lists (ACLs)", "Block public access to buckets and objects granted through any access control lists (ACLs)", "Block public access to buckets and objects granted through new public bucket or access point policies", and "Block public and cross-account access to buckets and objects through any public bucket or access point policies". A red arrow points to the "Save changes" button at the bottom right.

Now we need to enable ACLs, or access control lists. ACLs are how permissions work in most operating systems, where access is controlled by 3 aspects: List, Read, and Write. We want to enable ACLs for the bucket, so do the following and save changes:



<< continued on next page >>

At this point the bucket is now public, and its objects *can* be public.

The last step is to actually make objects public. We need to configure the bucket's ACL to allow public access:

Permissions overview

Access

Objects can be public

Access control list (ACL)
Grant basic read/write permissions to other AWS accounts. [Learn more](#)

The console displays combined access grants for duplicate grantees
To see the full list of ACLs, use the Amazon S3 REST API, AWS CLI, or AWS SDKs.

Grantee	Objects	Bucket ACL
Bucket owner (your AWS account) Canonical ID: 6975725107b0a013082f9dfef83b8f60f317b7da74adf63424a018c8d4399a23	List, Write	Read, Write
Everyone (public access) Group: http://acs.amazonaws.com/groups/global/AllUsers	-	-
Authenticated users group (anyone with an AWS account) Group: http://acs.amazonaws.com/groups/global/AuthenticatedUsers	-	-
S3 log delivery group Group: http://acs.amazonaws.com/groups/s3/LogDelivery	-	-

Edit access control list (ACL) [Info](#)

Access control list (ACL)
Grant basic read/write permissions to other AWS accounts. [Learn more](#)

Grantee	Objects	Bucket ACL
Bucket owner (your AWS account) Canonical ID: 6975725107b0a013082f9dfef83b8f60f317b7da74adf63424a018c8d4399a23	<input checked="" type="checkbox"/> List <input checked="" type="checkbox"/> Write	<input checked="" type="checkbox"/> Read <input checked="" type="checkbox"/> Write
Everyone (public access) Group: http://acs.amazonaws.com/groups/global/AllUsers	<input checked="" type="checkbox"/> List <input checked="" type="checkbox"/> Write	<input checked="" type="checkbox"/> Read <input checked="" type="checkbox"/> Write
Authenticated users group (anyone with an AWS account) Group: http://acs.amazonaws.com/groups/global/AuthenticatedUsers	<input type="checkbox"/> List <input type="checkbox"/> Write	<input type="checkbox"/> Read <input type="checkbox"/> Write
S3 log delivery group Group: http://acs.amazonaws.com/groups/s3/LogDelivery	<input type="checkbox"/> List <input type="checkbox"/> Write	<input type="checkbox"/> Read <input type="checkbox"/> Write

1 When you grant access to the Everyone or Authenticated users group grantees, anyone in the world can access the objects in this bucket.
[Learn more](#)

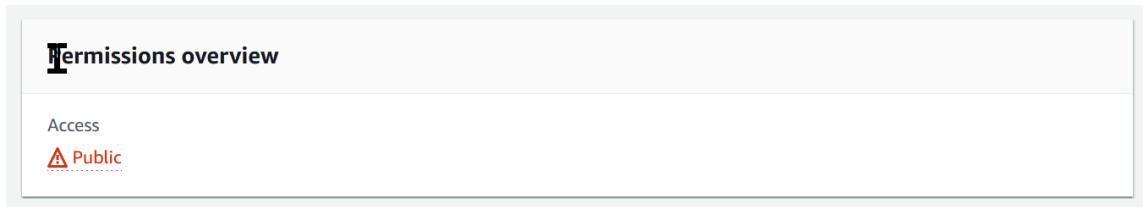
2 I understand the effects of these changes on my objects and buckets.

3 **Add grantee**

Access for other AWS accounts
No other AWS accounts associated with the resource.

Cancel **Save changes**

At this point, the bucket “Permission overview” should say that all access is now Public:



If all is well, you should now be able to open a browser and view the contents of your new bucket:

<http://new-bucket-nu-cs310.s3.us-east-2.amazonaws.com/>

Replace “new-bucket-nu-cs310” with the name of your bucket, and you should see something like this:

A screenshot of a web browser displaying the XML response from an S3 bucket. The URL in the address bar is "new-bucket-nu-cs310.s3.us-east-2.amazonaws.com". The page content shows an XML document structure:

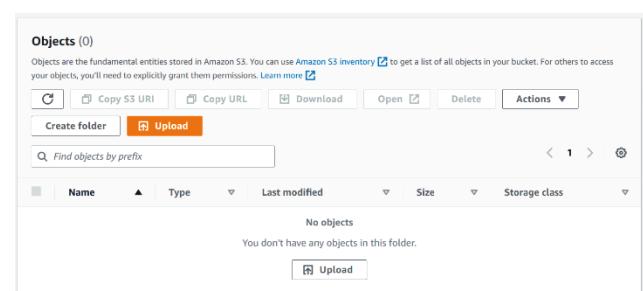
```
<ListBucketResult xmlns="http://s3.amazonaws.com/doc/2006-03-01/">
<Name>new-bucket-nu-cs310</Name>
<Prefix/>
<Marker/>
<MaxKeys>1000</MaxKeys>
<IsTruncated>false</IsTruncated>
</ListBucketResult>
```

If it's not working, delete the bucket you created and start again? Did you create in the correct region? Are you configured for the correct region if using the CLI?

Step 3: Manually upload some photos

Step 3 is to manually upload some photos to the bucket. We're doing this to make sure things are setup properly, then in a later step we'll upload programmatically using Python. But one step at a time...

Using the Management Console (or CLI), create a new folder in your bucket named “test”. Note that AWS will automatically add the “/” as part of the folder name. Click on the folder name to navigate into the folder. Use the Upload button to upload one or more images to your test folder, don't worry about the file names, it can be whatever you want, though upload JPG images since we know those will display correctly in replit.



It turns out that uploads do *not* inherit bucket permissions, so if we want the newly-upload images to be publicly readable, we must explicitly grant that permission. For each image you uploaded, click on the image in the Management Console, change to the Permissions tab, click the ACL edit button, and change the permissions to allow Read access for Everyone:

The screenshot shows the AWS S3 object details for 'degu.jpg'. The 'Permissions' tab is active. An 'Edit' button is highlighted with a red arrow. A red box highlights the 'Read' permission for the 'Everyone (public access)' group.

Go ahead and test that everyone has read access to your image(s). Browse to your bucket like before, adding "test/filename.jpg" to the URL:

<http://new-bucket-nu-cs310.s3.us-east-2.amazonaws.com/test/degu.jpg>

If all is well, your image should appear in the browser. Now we want to repeat this 3 more times... First, create three folders with random, unique names using the UUID python module. Startup any python environment, repl.it, or an online python compiler such as <https://www.online-python.com/>, and run the following code each time you need a unique name:

```
import uuid

print(uuid.uuid4())
```

Create 3 folders in your bucket. Something like this:

	Name	Type	Last modified	Size	Storage class
	52592157-9216-4d4c-882f-3f3013864930/	Folder	-	-	-
	6b0be043-1265-4c80-9719-fd8dbcda8fd4/	Folder	-	-	-
	ab099de4-ea33-4237-8c78-5584dc591231/	Folder	-	-	-
	img.txt	txt	March 20, 2023, 15:27:16 (UTC 05:00)	50.0 B	Standard
	test/	Folder	-	-	-

<< continued on next page >>

Navigate into each folder, and upload at least two image (JPG) files. The filename should also be randomly generated using the UUID module; note you can upload a file with its existing name and then rename the object once it's in the bucket (select the object and then under "Actions" select "Rename object"). Don't forget to also change the permissions on the object so it can be publicly accessed --- a short-cut is to select the object and then under "Actions" select "Make public using ACL".

Objects are the fundamental entities stored in Amazon S3. You can use [Amazon S3 inventory](#) to get a list of all objects in your bucket. For others to access your objects, you'll need to explicitly grant them permissions. [Learn more](#)

Actions ▾

Name	Type	Last modified	Size	Storage class
<input checked="" type="checkbox"/> 195e06c8-6005-4006-97f2-1c7c19b3414b.jpg	jpg	March 25, 2023, 08:58:12 (UTC-05:00)	42.5 KB	Standard

At this point you should have FOUR folders (including "test"), and inside each folder at least two image (JPG) files. A final test is to browse to your root-level bucket URL, and you should receive an XML document that lists every folder and every object key:

<http://photoapp-nu-cs310.s3.us-east-2.amazonaws.com/>

Something like this:

```

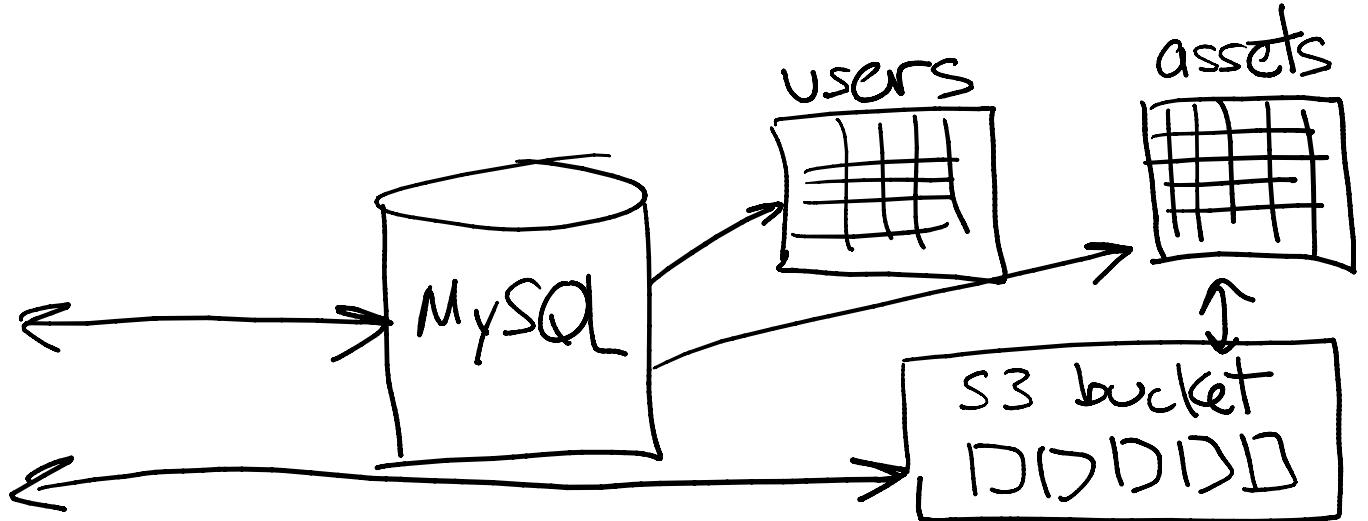
    <ListBucketResult xmlns="http://s3.amazonaws.com/doc/2006-03-01/">
        <Name>photoapp-nu-cs310</Name>
        <Prefix/>
        <Marker/>
        <MaxKeys>1000</MaxKeys>
        <IsTruncated>false</IsTruncated>
        <Contents>
            <Key>52592157-9216-4d4c-882f-3f3013864930/<Key>
            <LastModified>2023-03-25T13:37:47.000Z</LastModified>
            <ETag>"d41d8cd98f00b204e9800998ecf8427e"</ETag>
            <Size>0</Size>
            <StorageClass>STANDARD</StorageClass>
        </Contents>
        <Contents>
            <Key>52592157-9216-4d4c-882f-3f3013864930/195e06c8-6005-4006-97f2-1c7c19b3414b.jpg</Key>
            <LastModified>2023-03-25T13:50:12.000Z</LastModified>
            <ETag>"e9ec7990943728e66fdde155981493e84"</ETag>
            <Size>43534</Size>
            <StorageClass>STANDARD</StorageClass>
        </Contents>
        <Contents>
            <Key>52592157-9216-4d4c-882f-3f3013864930/564055dc-f109-4df8-bdf1-27a629d4a0c6.jpg</Key>
            <LastModified>2023-03-25T13:57:10.000Z</LastModified>
            <ETag>"9cac1fd5cd2f5feadc5f8b38f1735473"</ETag>
            <Size>5438</Size>
            <StorageClass>STANDARD</StorageClass>
        </Contents>
        <Contents>
            <Key>52592157-9216-4d4c-882f-3f3013864930/a68cab2e-7d23-4af3-bbb1-f067befc6712.jpg</Key>
            <LastModified>2023-03-25T13:56:36.000Z</LastModified>
            <ETag>"e73d5e337b65182f8f9a5f2f68a63884"</ETag>
            <Size>60222</Size>
            <StorageClass>STANDARD</StorageClass>
        </Contents>
        <Contents>
            <Key>600be043-1265-4c80-9719-fd8dbcda8fd4/<Key>
            <LastModified>2023-03-25T13:33:42.000Z</LastModified>
            <ETag>"d41d8cd98f00b204e9800998ecf8427e"</ETag>
            <Size>0</Size>
            <StorageClass>STANDARD</StorageClass>
        </Contents>
    
```

] folder

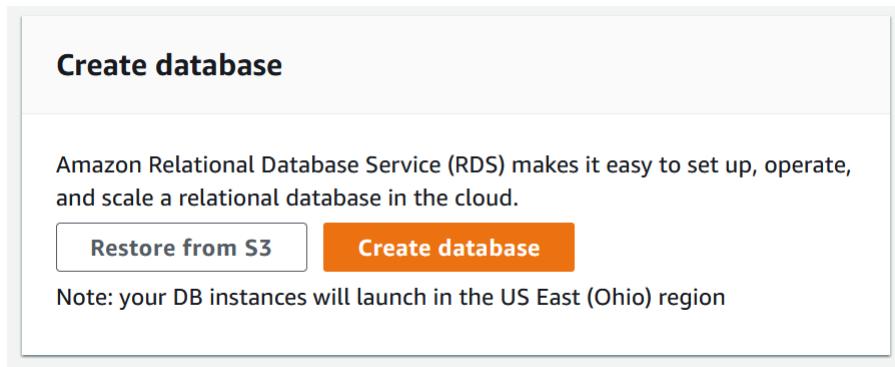
Image Key

Step 4: Setup AWS RDS and MySQL database server

Our application needs a database to keep track of users, and for each user to keep track of their images in S3. The database will consist of two tables, **users** and **assets**, as depicted below:



We're going to use **MySQL** as our relational database, which is hosted as part of AWS RDS. The first step is to configure RDS to run MySQL and host our database. Be ready, it's a tedious process. Open the Management Console and search for "RDS". Scroll down to where you see "Create database", and click the big button:



<< next page >>

On the top-half of the page select “Standard create” and “MySQL”:

The screenshot shows the 'Create database' page in the AWS RDS console. In the 'Choose a database creation method' section, the 'Standard create' option is selected and highlighted with a red box. In the 'Engine options' section, the 'MySQL' engine type is selected and highlighted with a red box.

As you scroll down, be sure to select “Free tier”:

The screenshot shows the continuation of the 'Create database' page. The 'Edition' section has 'MySQL Community' selected. Below it, there's a note about known issues. Under 'Filters', 'Show versions that support the Multi-AZ DB cluster' is selected. Under 'Engine Version', 'MySQL 8.0.32' is chosen. The 'Templates' section at the bottom has three options: 'Production', 'Dev/Test', and 'Free tier'. The 'Free tier' option is selected and highlighted with a red box, with a red arrow pointing to it from the bottom right. A second red arrow points to the 'Free tier' option from the left side of the screen.

Scrolling down, set the database instance identifier to the name of your MySQL server --- some variation of MySQL is fine --- and provide a password for the server's admin account:

Settings

DB instance identifier [Info](#)
Type a name for your DB instance. The name must be unique across all DB instances owned by your Account.
 mysql-nu-cs310 →

Credentials Settings

Master username [Info](#)
Type a login ID for the master user of your DB instance.
 admin →

Manage master credentials in AWS Secrets Manager
Manage master user credentials in Secrets Manager. RDS can generate a password for you and manage it throughout its lifecycle.

ⓘ If you manage the master user credentials in Secrets Manager, some RDS features will not be available. [Learn more](#) ↗

Auto generate a password
Amazon RDS can generate a password for you, or you can specify your own password.

Master password [Info](#)
 ***** →

Constraints: At least 8 printable ASCII characters. Can't contain any of the following: / (slash), ' (single quote), " (double quote), & (ampersand), ! (exclamation point), @ (at sign).

Confirm master password [Info](#)

Reduce the allocated storage to the minimum, and disable storage autoscaling:

Instance configuration

The DB instance configuration options below are limited to those supported by the engine that you selected above.

ⓘ **Amazon RDS Optimized Writes - new** [Info](#)
 Show instance classes that support Amazon RDS Optimized Writes

DB instance class [Info](#)
 Standard classes (includes m classes)
 Memory optimized classes (includes r and x classes)
 Burstable classes (includes t classes)

db.t3.micro
2 vCPUs 1 GiB RAM Network: 2,085 Mbps →

Include previous generation classes

Storage

Storage type [Info](#)
 General Purpose SSD (gp2) →
Baseline performance determined by volume size

Allocated storage [Info](#)
 20 → GiB
The minimum value is 20 GiB and the maximum value is 6,144 GiB

Storage autoscaling [Info](#)
Provides dynamic scaling support for your database's storage based on your application's needs.

Enable storage autoscaling
Enabling this feature will allow the storage to increase after the specified threshold is exceeded.

Almost done... Next, enable the database server so it's publicly accessible --- the default values for the other settings should be correct:

Connectivity [Info](#) [Edit](#)

Compute resource
Choose whether to set up a connection to a compute resource for this database. Setting up a connection will automatically change connectivity settings so that the compute resource can connect to this database.

Don't connect to an EC2 compute resource
Don't set up a connection to a compute resource for this database. You can manually set up a connection to a compute resource later.

Connect to an EC2 compute resource
Set up a connection to an EC2 compute resource for this database.

Network type [Info](#)
To use dual-stack mode, make sure that you associate an IPv6 CIDR block with a subnet in the VPC you specify.

IPv4
Your resources can communicate only over the IPv4 addressing protocol.

Dual-stack mode
Your resources can communicate over IPv4, IPv6, or both.

Virtual private cloud (VPC) [Info](#)
Choose the VPC. The VPC defines the virtual networking environment for this DB instance.

vpc-0ccf3cdb378341daf [Edit](#)

Only VPCs with a corresponding DB subnet group are listed.

DB subnet group [Info](#)
Choose the DB subnet group. The DB subnet group defines which subnets and IP ranges the DB instance can use in the VPC that you selected.

default-vpc-0ccf3cdb378341daf [Edit](#)

Public access [Info](#)

Yes
RDS assigns a public IP address to the database. Amazon EC2 instances and other resources outside of the VPC can connect to your database. Resources inside the VPC can also connect to the database. Choose one or more VPC security groups that specify which resources can connect to the database.

No
RDS doesn't assign a public IP address to the database. Only Amazon EC2 instances and other resources inside the VPC can connect to your database. Choose one or more VPC security groups that specify which resources can connect to the database.

Scrolling down... Change “Database authentication” to allow both “Password and IAM”:

Database authentication

Database authentication options [Info](#)

Password authentication
Authenticates using database passwords.

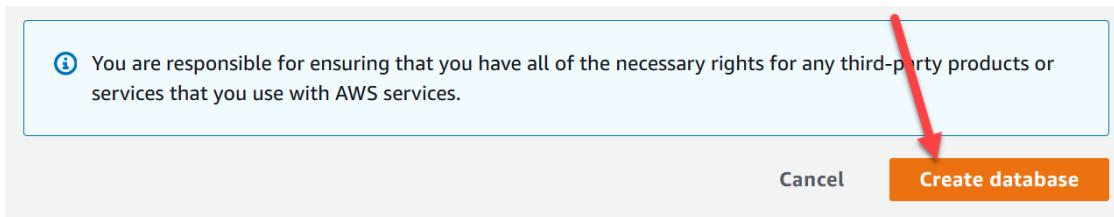
Password and IAM database authentication
Authenticates using the database password and user credentials through AWS IAM users and roles.

Password and Kerberos authentication
Choose a directory in which you want to allow authorized users to authenticate with this DB instance using Kerberos Authentication.

To save \$, you want to uncheck automated backups, under “Additional configuration”:

▼ Additional configuration
Database options
Initial database name [Info](#)
If you do not specify a database name, Amazon RDS does not create a database.
DB parameter group [Info](#)
default.mysql8.0
Option group [Info](#)
default:mysql-8-0
Backup
☐ Enable automated backups
Creates a point-in-time snapshot of your database

Finally, at the very bottom, click “Create database”, and take a break. AWS will take at least 5 minutes to create your new database...



In the Management console, monitor the “Status” column to see when the creation process has completed. You may need to enlarge your browser window (or shrink the page):

Databases							<input checked="" type="checkbox"/> Group resources	C	Modify	Actions ▾	Restore from S3
							Filter by databases				
DB identifier	▲	Role ▾	Engine	▼	Region & AZ ▾	Size ▾	Status ▾	Actions ▾	Restore from S3		
mysql-nu-cs310		Instance	MySQL Community	us-east-2a	db.t3.micro		Creating	-			

<< continued on next page >>

Your database server is ready when its status changes to “Available”. When available, click on the database instance ID so you can view its properties, in particular its endpoint and to confirm it’s publicly accessible:

The screenshot shows the AWS RDS 'Databases' page for the database 'mysql-nu-cs310'. The 'Summary' tab is selected, displaying basic information like DB identifier, CPU, Status (Available), and Class. Below the summary, the 'Connectivity & security' tab is active, showing details under three sections: 'Endpoint & port', 'Networking', and 'Security'. The 'Endpoint' field in the first section is highlighted with a red box and has a red arrow pointing from it to the 'Publicly accessible' field in the third section, which is also highlighted with a red box. The 'Networking' section shows the VPC and subnet group, while the 'Security' section shows VPC security groups and the certificate authority.

Endpoint	Availability Zone	Security
mysql-nu-cs310.cb1xaky37wq8.us-east-2.rds.amazonaws.com	us-east-2a	VPC security groups default (sg-09c0fdcca7e28e109) Active
Port	VPC	Publicly accessible Yes
3306	vpc-0ccf3cdb378341daf	Certificate authority Info rds-ca-2019
	Subnet group	Certificate authority date August 22, 2024, 12:08 (UTC-05:00)
	Subnets	DB instance certificate expiration date August 22, 2024, 12:08 (UTC-05:00)
	Subnets	
	Network type	
	IPv4	

The endpoint is the hostname for your database server --- this is the machine you'll connect to when connecting to your database. If the database is not “Publicly accessible”, you will be unable to connect --- in this case delete the database instance and start over...

Are we done with the setup of the database server? Unfortunately, no. There's one more step, critically important because it turns out the database server is still not accessible. Read on...

By default AWS is blocking incoming network traffic to the server for security reasons. Normally you could ping the machine to make sure it's up and running --- but ping traffic is blocked. It's our job to decide how much to “open up” our database server to the internet. For simplicity, we're going to open the server to the entire world so it can be accessed by any machine.

First, confirm your database is available (search for “RDS”, and click “databases” along the left-side):

Databases

Group resources Modify Actions

Filter by databases

DB identifier	Role	Engine	Region & AZ	Size	Status
mysql-nu-cs310	Instance	MySQL Community	us-east-2a	db.t3.micro	Available

Click on the database server identifier, and you'll see it's properties. Focus on Security:

RDS > Databases > mysql-nu-cs310

mysql-nu-cs310

Summary

DB identifier mysql-nu-cs310	CPU 3.32%	Status Available
Role Instance	Current activity 0 Connections	Engine MySQL Community

Connectivity & security Monitoring Logs & events Configuration Maintenance & backups

Connectivity & security

Endpoint & port	Networking	Security
Endpoint mysql-nu-cs310.cb1xaky37wq8.us-east-2.rds.amazonaws.com	Availability Zone us-east-2a	VPC security groups default (sg-09c0fdcca7e28e109) Active
	VPC	

Click on the VPC (Virtual Private Cloud) security group, and click the tab for “Inbound rules”:

Security Groups (1/1) Info

Actions

Filter security groups

Name	Security group ID	Security group name	VPC ID	Description	Owner
-	sg-09c0fdcca7e28e109	default	vpc-0ccf3cdb378341daf	default VPC security gr...	444800541605

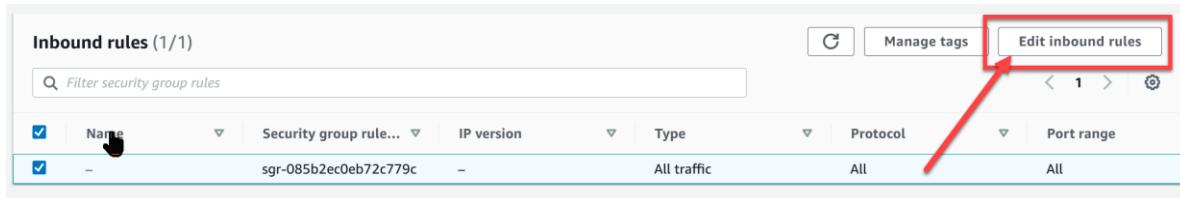
Inbound rules Outbound rules Tags

You can now check network connectivity with Reachability Analyzer Run Reachability Analyzer

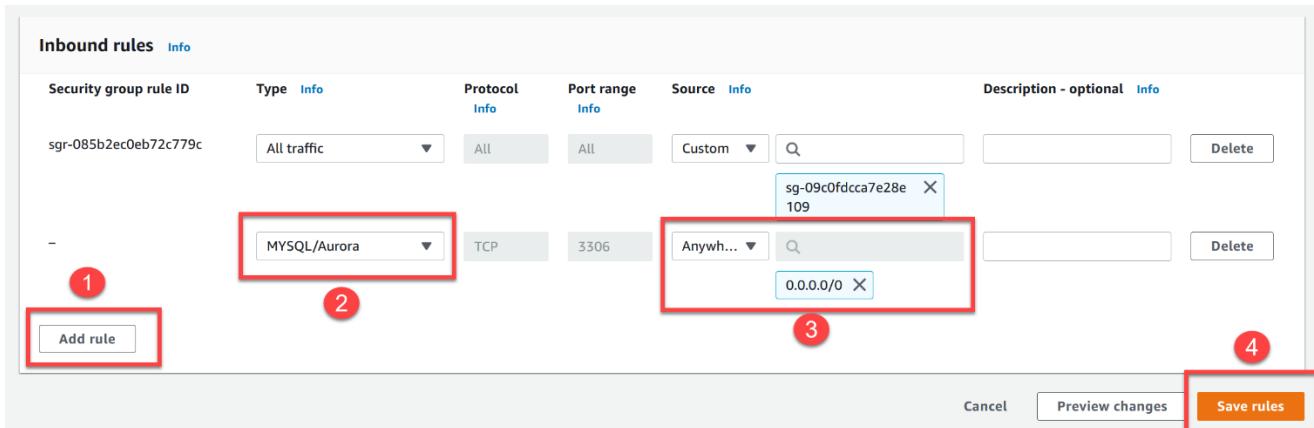
Inbound rules (1/1)

Name	Security group rule...	IP version	Type	Protocol	Port range
-	sgr-085b2ec0eb72c779c	-	All traffic	All	All

Interestingly, even though it lists “All traffic”, it’s not what we think. Click “Edit inbound rules” so we can add another rule to open up the server to all IP addresses:



In the dialog that opens, click “Add rule”, change Type to “MySQL/Aurora”, and change Source to “Anywhere-IPv4”. Then click “Save Rules”:



Done with RDS and MySQL server setup!

The good news is you only have to do this once, the AWS RDS MySQL database server can host any number of databases. To confirm everything is working properly, the simplest approach is to login to repl.it.com, open the team, and then open the project “Test-MySQL-in-AWS”. It’s a small python program that will connect to your database server, and if successful, output the current date and time GMT. You’ll need to copy-paste your database server endpoint and master user password (and master user name if you changed it from “admin”):

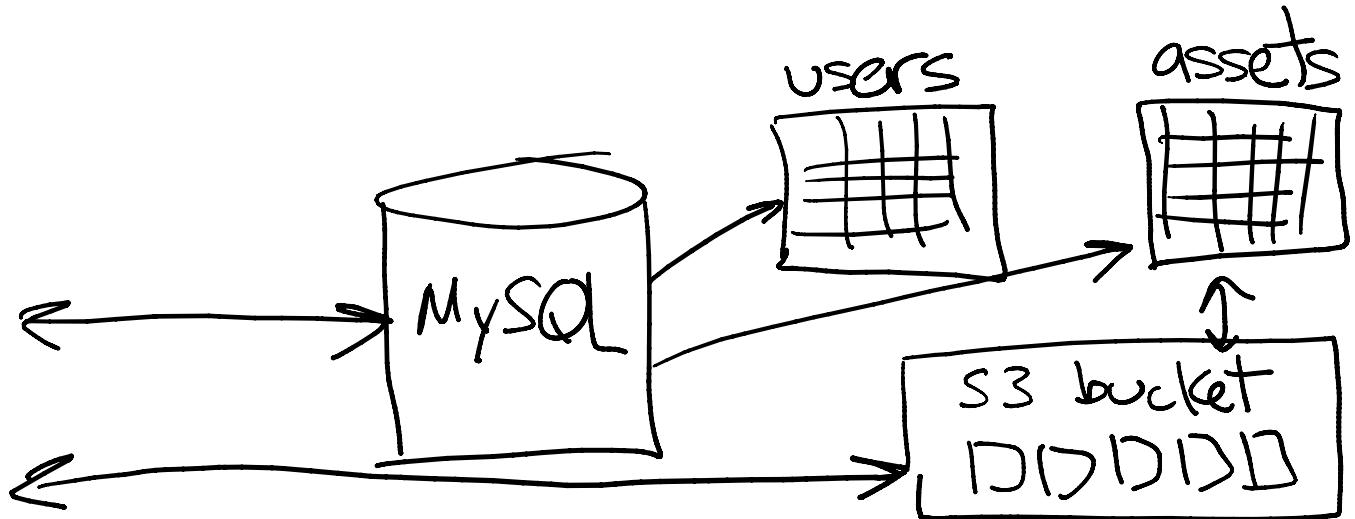
```
ENDPOINT="TODO"  
USER="admin"  
PASSWORD="TODO"
```

Click “Run” and if all is well, you’ll see date and time. If you plan to take a break from working on the assignment, select your database instance and under “Actions”, select “Stop temporarily” --- this will pause the server, free the assigned VM instance, and save \$. However, you’ll need to manually restart when you want to resume working. Using AWS CLI, where the DB_INSTANCE_NAME is something like *mysql-nu-cs310*:

```
aws rds stop-db-instance --db-instance-identifier DB_INSTANCE_NAME  
...  
aws rds start-db-instance --db-instance-identifier DB_INSTANCE_NAME
```

Step 5: Create photoapp database

Now that MySQL is setup in AWS RDS, the next step is to create our photoapp database with two tables **users** and **assets**, to keep track of users, and for each user to keep track of their images in S3:



The **users** table will keep track of each user's email, last name, first name, and the name of their bucket folder. Every user is identified by a unique user id; ids will start at 80001 and increase by 1. In DB design terms, userid is the *primary key*. Example: the CS310 photoapp database consists of 3 rows in the users table:

userid	email	lastname	firstname	bucketfolder
80001	pooja.sarkar@company.com	sarkar	pooja	6b0be043-1265-4c80-9719-fd8dbcda8fd4
80002	e_ricci@email.com	ricci	emanuele	ab099de4-ea33-4237-8c78-5584dc591231
80003	li_chen@domain.com	chen	li	52592157-9216-4d4c-882f-3f3013864930

The **assets** table keeps track of each image in the bucket: it's original name ("assetname"), it's key name ("bucketkey"), and the user who's to which this image belongs to ("userid"). Each asset has a unique asset id; ids start at 1001 and increase by 1. In DB design terms, assetid is the *primary key*, and userid is a *foreign key* to the users table. Example: the CS310 photoapp database contains 3 images per user, for a total of 9 rows in the assets table:

assetid	userid	assetname	bucketkey
1001	80001	A3-mac-2016.JPG	6b0be043-1265-4c80-9719-fd8dbcda8fd4/af986381-55ac-4bf2-85b3-ff4a29047226.jpg
1002	80001	A3-verve-Megan-on-bow.jpg	6b0be043-1265-4c80-9719-fd8dbcda8fd4/62ec70d5-ba0c-4822-9017-3864b1d1fb47.jpg
1003	80001	NOOD-A3-water-photo-2016.jpg	6b0be043-1265-4c80-9719-fd8dbcda8fd4/2c20b346-6f2b-4bca-b770-772bd0aa9ad3.jpg
1004	80002	ajax.jpg	ab099de4-ea33-4237-8c78-5584dc591231/a4365e93-5c41-4997-8414-389cf874509e.jpg
1005	80002	Darwin.jpg	ab099de4-ea33-4237-8c78-5584dc591231/fdcc04a8-4a73-494a-978f-41e0dba9eeb3.jpg
1006	80002	Degu.JPG	ab099de4-ea33-4237-8c78-5584dc591231/58732c6b-2e1c-4503-944b-c63f7abb546c.jpg
1007	80003	All-Day-Joke.jpg	52592157-9216-4d4c-882f-3f3013864930/a68cab2e-7d23-4af3-bbb1-f067befc6712.jpg
1008	80003	NU-purple-background.png	52592157-9216-4d4c-882f-3f3013864930/564055dc-f109-4df8-bdf1-27a629d4a0c6.jpg
1009	80003	NU-white-background.jpg	52592157-9216-4d4c-882f-3f3013864930/195e06c8-6005-4006-97f2-1c7c19b3414b.jpg

NOTE: because MySQL is multi-platform and potentially case-sensitive, the convention is to use lowercase for all table and column names. We follow that convention in our database design.

The database and tables are created by executing SQL. Unfortunately, there's no tool within the AWS Management Console for executing the necessary SQL to create the database. You'll need to install a local tool on your laptop to interact with the MySQL instance running in AWS. I would recommend either of the following:

1. **MySQL Workbench**: free, runs on any platform, very popular. Download link is <https://dev.mysql.com/downloads/workbench/> . [M1 Monterey [users](#); M1 Big Sur [users](#).]
2. **Visual Studio code**: if you're already using VS Code, there are a number of extensions you can install to connect to MySQL servers and execute SQL. *MySQL Management Tool* (Jun Han) works fine; there are many others.

You'll see demos of how to use these tools in class, but the general idea is to open a connection to your MySQL database instance, and then open a query window and execute the SQL. To open a connection, you'll need to provide your endpoint (e.g. mysql-nu-cs310.cb1xaky37wq8.us-east-2.rds.amazonaws.com), master user name (e.g. admin), and master user password. If you're asked for the port number, use 3306; if asked for a security certificate, it's optional so you can ignore. Once connected, you want to open a query window and execute SQL. As a test, execute the following SQL query to display the current date and time in GMT:

```
select now();
```

You're ready to create to the photoapp database. Execute the following SQL:

```
CREATE DATABASE photoapp;
```

Now we need to create the tables. Here's the SQL, but it's also available online in the file [create-tables.sql](#) so you can more easily copy-paste:

```
USE photoapp;

DROP TABLE IF EXISTS assets;
DROP TABLE IF EXISTS users;

CREATE TABLE users
(
    userid      int not null AUTO_INCREMENT,
    email       varchar(128) not null,
    lastname    varchar(64) not null,
    firstname   varchar(64) not null,
    bucketfolder varchar(48) not null, -- random, unique name (UUID)
    PRIMARY KEY (userid),
    UNIQUE      (email),
    UNIQUE      (bucketfolder)
);

ALTER TABLE users AUTO_INCREMENT = 80001; -- starting value

CREATE TABLE assets
(
    assetid     int not null AUTO_INCREMENT,
    userid      int not null,
```

```

assetname      varchar(128) not null,    -- original name from user
bucketkey      varchar(128) not null,    -- random, unique name in bucket
PRIMARY KEY (assetid),
FOREIGN KEY (userid) REFERENCES users(userid),
UNIQUE        (bucketkey)
);

ALTER TABLE assets AUTO_INCREMENT = 1001;  -- starting value

```

The last step is to execute SQL to populate your database with M users (one user row for each folder you created), and N assets (one asset row for each image you uploaded). Given the earlier instructions, you should have at least 3 users ($M \geq 3$) and 6 assets ($N \geq 6$) in your database. How do you insert row into the database? Use an SQL INSERT action query. Here are examples of inserting into the users and assets table from our CS310 database (this SQL is available in [insert-rows.sql](#)). Note that the userid in the users table (and the assetid in the assets table) are automatically generated to ensure they are unique, and so those values are omitted when inserting into the users table (and assets table):

```

USE photoapp;

INSERT INTO
users(email, lastname, firstname, bucketfolder)
values('pooja.sarkar@company.com', 'sarkar', 'pooja',
'6b0be043-1265-4c80-9719-fd8dbcda8fd4');

INSERT INTO
assets(userid, assetname, bucketkey)
values(80001,
'A3-mac-2016.JPG',
'6b0be043-1265-4c80-9719-fd8dbcda8fd4/af986381-55ac-4bf2-85b3-ff4a29047226.jpg');

```

Create fictitious user names and emails, then associate each user with one of the folders in your S3 bucket. Likewise, associate the image within each bucket with the appropriate user (based on which user owns which folder). Example: if we insert “pooja” first, she’ll have the userid 80001. We associate her with one of the folders, and then associate the image(s) in that folder with her. If this is the first asset inserted, it will have assetid 1001. Repeating this for each folder and each image in the S3 bucket, we end up with the tables populated as shown earlier.

As you insert and populate your database, you can check your work using `SELECT *` as shown:

```

USE photoapp;

SELECT * FROM users;

```

userid	email	lastname	firstname	bucketfolder
80001	pooja.sarkar@company.com	sarkar	pooja	6b0be043-1265-4c80-9719-fd8dbcda8fd4
80002	e_ricci@email.com	ricci	emanuele	ab099de4-ea33-4237-8c78-5584dc591231
80003	li_chen@domain.com	chen	li	52592157-9216-4d4c-882f-3f3013864930

```
USE photoapp;
SELECT * FROM assets;
```

assetid	userid	assetname	bucketkey
1001	80001	A3-mac-2016.JPG	6b0be043-1265-4c80-9719-fd8dbcda8fd4/af986381-55ac-4bf2-85b3-ff4a29047226.jpg
1002	80001	A3-verve-Megan-on-bow.jpg	6b0be043-1265-4c80-9719-fd8dbcda8fd4/62ec70d5-ba0c-4822-9017-3864b1d1fb47.jpg
1003	80001	NOOD-A3-water-photo-2016.jpg	6b0be043-1265-4c80-9719-fd8dbcda8fd4/2c20b346-6f2b-4bca-b770-772bd0aa9ad3.jpg
1004	80002	ajax.jpg	ab099de4-ea33-4237-8c78-5584dc591231/a4365e93-5c41-4997-8414-389cf874509e.jpg
1005	80002	Darwin.jpg	ab099de4-ea33-4237-8c78-5584dc591231/fdcc04a8-4a73-494a-978f-41e0dba9eeb3.jpg
1006	80002	Degu.JPG	ab099de4-ea33-4237-8c78-5584dc591231/58732c6b-2e1c-4503-944b-c63f7abb546c.jpg
1007	80003	All-Day-Joke.jpg	52592157-9216-4d4c-882f-3f3013864930/a68cab2e-7d23-4af3-bbb1-f067befc6712.jpg
1008	80003	NU-purple-background.png	52592157-9216-4d4c-882f-3f3013864930/564055dc-f109-4df8-bdf1-27a629d4a0c6.jpg
1009	80003	NU-white-background.jpg	52592157-9216-4d4c-882f-3f3013864930/195e06c8-6005-4006-97f2-1c7c19b3414b.jpg

If you make a mistake, not to worry. Simply re-run the first SQL code and rebuild the database ([create-tables.sql](#)); this deletes and recreates an empty database. Now re-insert the data, correcting any errors you made earlier. You can repeat this as many times as necessary.

Step 6: Controlling access to your database

Right now you are connecting to your MySQL server using the administrator account. This is okay when creating a database, but certainly not a best practice for general-purpose read/write access. When we start programming against the database using Python, we want user accounts with more limited permissions.

User accounts and permissions are defined in two ways: (1) through MySQL, and (2) through AWS IAM. We're going to use both methods, but to start we're going to focus on #1 so you see how to create accounts using MySQL --- this way you'll know how to do this in the future if you happen to be working with MySQL outside of AWS. Here's the SQL to create two user accounts, photoapp-read-only and photoapp-read-write. The former can execute SELECT queries to retrieve data, and the latter can execute both SELECT and ACTION queries to both retrieve and modify data (this is available online in the file [add-users.sql](#)). Do *not* change the user names as shown, and do *not* change the passwords, we need these as below for grading purposes:

```
USE photoapp;
DROP USER IF EXISTS 'photoapp-read-only';
DROP USER IF EXISTS 'photoapp-read-write';

CREATE USER 'photoapp-read-only' IDENTIFIED BY 'abc123!!';
CREATE USER 'photoapp-read-write' IDENTIFIED BY 'def456!!';

GRANT SELECT, SHOW VIEW ON photoapp.* 
    TO 'photoapp-read-only';
GRANT SELECT, SHOW VIEW, INSERT, UPDATE, DELETE ON photoapp.* 
    TO 'photoapp-read-write';

FLUSH PRIVILEGES;
```

Test and make sure the user accounts were created successfully. The best way to test is to create another connection to your photoapp database --- using MySQL Workbench or VS Code (or whatever tool you are using to execute SQL) --- and connect using the user name **photoapp-read-only** and the password **abc123!!**. You should be able to successfully connect, and execute this SQL:

```
USE photoapp;  
  
SELECT * FROM users;  
SELECT * FROM assets;
```

Repeat, creating another connection with the username **photoapp-read-write** and the password **def456!!**. Congratulations, and well done!

Step 7: Pause your RDS server?

At this point you may want to take a break, and in that case you should temporarily stop your database server instance to save \$. Using the AWS Management Console, select your database instance and under “Actions”, select “Stop temporarily” --- this will pause the server, free the assigned VM instance, and save \$. However, you’ll need to manually restart when you want to resume working. Alternatively, you can use the AWS CLI, where the DB_INSTANCE_NAME is something like *mysql-nu-cs310*:

```
aws rds stop-db-instance --db-instance-identifier DB_INSTANCE_NAME  
...  
aws rds start-db-instance --db-instance-identifier DB_INSTANCE_NAME
```

Next steps?

This is NOT the end of Project 01. But the handout is 25 pages at this point, so I’m going to break the handout into parts 01 and 02. This is the end of part 01, but there’s a part 02 where we’ll program in Python to tie all this together and build the foundation of our PhotoApp application. Watch for the release of part 02 on Canvas and Piazza. The due date remains the same: Friday October 6th, 11:59pm. Final submission details will appear in part 02.