CS 327E Final Project: Milestone 2, due Thursday 05/12
Due date is not flexible

1. Create a new Jupyter notebook for this milestone and name it `milestone2.ipynb`. All work for this milestone should be done in a notebook cell unless otherwise specified.

2. If you are using Postgres, skip to step 3.

   If you are using MySQL, read the manual to learn about the `max_allowed_packet` variable and run these commands in the mysql cli to set this variable:

   ```
   SHOW VARIABLES LIKE 'max_allowed_packet';

   SET PERSIST max_allowed_packet=536870912;

   SELECT v.VARIABLE_NAME, g.VARIABLE_VALUE current_value,
           p.VARIABLE_VALUE as persist_value
   FROM performance_schema.variables_info v
   JOIN performance_schema.persisted_variables p
   USING(VARIABLE_NAME)
   JOIN performance_schema.global_variables g USING(VARIABLE_NAME);
   ```

   The SHOW command gets the current value of the `max_allowed_packet` variable. The SET command increases the value of `max_allowed_packet` to 512M in bytes. The SELECT statement returns the current and persisted values of `max_allowed_packet`.

   Note: The above three commands should be run on the database server (not from JupyterLab).

3. Copy the loader code you wrote for Milestone 1 into the current notebook and modify the logic as follows:

   Instead of executing one insert statement at a time using the execute() method, write batch inserts to the database in sizes of 5000, 50,000 and 500,000.

   If you are using MySQL, use the mysql connector's executemany() method. See code sample for details.

   If you are using Postgres, use the psycopg connector's execute_batch() method. See code sample for details. Note: execute_batch() is only available in psycopg2, so run `pip install psycopg2` prior to running the code sample.

   Continue to measure the execution time of your code using the `%%timeit` magic function.

Add a short Markdown comment above each of the code blocks that describes the batch size of the test.

Your code should produce the following output for the small batch size (5000):

```
5000 records inserted successfully into Person table
10000 records inserted successfully into Person table
15000 records inserted successfully into Person table
20000 records inserted successfully into Person table
...
1000000 records inserted successfully into Person table
MySQL/Postgres connection is closed
1min 31s ± 2.76 s per loop (mean ± std. dev. of 7 runs, 1 loop each)
```

Your code should produce the following output for the medium batch size (50,000):

```
50000 records inserted successfully into Person table
100000 records inserted successfully into Person table
150000 records inserted successfully into Person table
200000 records inserted successfully into Person table
...
1000000 records inserted successfully into Person table
MySQL/Postgres connection is closed
1min 31s ± 2.02 s per loop (mean ± std. dev. of 7 runs, 1 loop each)
```

Your code should produce the following output for the large batch size (500,000):

```
500000 records inserted successfully into Person table
1000000 records inserted successfully into Person table
MySQL/Postgres connection is closed
...
500000 records inserted successfully into Person table
1000000 records inserted successfully into Person table
MySQL/Postgres connection is closed
1min 31s ± 1.6 s per loop (mean ± std. dev. of 7 runs, 1 loop each)
```

Note: while your timing results may differ from those reported here, you should notice a relative speed-up from your baseline results (single inserts in Milestone 1).

You should end up with 8,000,000 records in the Person table after each run. Make sure you empty out the table between each test (either by recreating the table or truncating it).

4. Follow this guide to request CPU quota increases for your GCP project.

5. Go to the GCP console and stop your database server. Once your instance has stopped, edit the instance and go to the compute-optimized machine family. Change the machine series from `N1` to `C2` and the machine type from `n1-standard-8` to `c2-standard-30`. Save your changes and restart your database server. You have upgraded your server to a machine with 30 CPUs and 120GB of RAM.

6. Back in your notebook, run a new load on the upgraded server as follows:

   - empty out the table
   - create a short Markdown comment that describes this test
   - copy the loader code for the large batch size (500,000) into a new cell and run it

   Your code should produce the following output:

   ```
   500000 records inserted successfully into Person table
   1000000 records inserted successfully into Person table
   MySQL/Postgres connection is closed
   ...
   500000 records inserted successfully into Person table
   1000000 records inserted successfully into Person table
   MySQL/Postgres connection is closed
   1min 24s ± 3.48 s per loop (mean ± std. dev. of 7 runs, 1 loop each)
   ```

   Note: your timing results may differ from those reported here.

7. Open the GCP console and stop your database server. Go back to your Jupyter notebook instance and open a terminal window. Run the following export command, replacing `<your_gcp_project_id>` with your actual GCP project id.

   ```
   export PROJECT=<your_gcp_project_id>
   ```

   Run the following two gcloud commands to create a disk image and a database server with high-network bandwidth:

   ```
   gcloud compute images create debian-11-gvnic \
      --source-image-family=debian-11 \
      --source-image-project=debian-cloud \
      --guest-os-features=GVNIC
   ```

```
gcloud beta compute instances create final-project-high-bw \
   --zone=us-central1-a \
   --network-performance-configs=total-egress-bandwidth-tier=TIER_1 \
   --network-interface=nic-type=GVNIC \
   --image=projects/$PROJECT/global/images/debian-11-gvnic \
   --machine-type=c2-standard-30 \
```

```
--create-disk=auto-delete=yes,boot=yes,device-name=instance-1,image=projects/$P
ROJECT/global/images/debian-11-gvnic,mode=rw,size=50,type=projects/$PROJECT/zon
es/us-central1-a/diskTypes/pd-ssd
```

When you copy-and-paste the above command into your terminal window, be sure that the `--create-disk` parameter doesn't get broken up into multiple lines.

Also, while the above command is running, you will see a warning about the 50G disk size. This is expected and can be safely ignored.

8. From the GCP console, ssh into final-project-high-bw. Follow the appropriate setup guide for your system to install and configure the DBMS on your high-bandwidth server:

   MySQL
   Postgres

   Ensure that you skip the initial VM creation section of the guide and proceed directly to the installation section.

   If you are using Postgres, restart your JupyterLab instance once you have completed the setup and updated the .bash_profile file with the new IP address.

   If you are using MySQL, re-run the commands in step 2 for adjusting the allowed packet size with your new high-bandwidth server.

9. Back in your notebook, run the load on the high-bandwidth server as follows:

   - create the database/schema and table using the `create_table.sql` script
   - create a short Markdown comment that describes this test
   - copy the loader code for the large batch size (500,000) into a new cell and run it

   Your code should produce the following output:

   ```
   500000 records inserted successfully into Person table
   1000000 records inserted successfully into Person table
   MySQL/Postgres connection is closed
   ...
   500000 records inserted successfully into Person table
   1000000 records inserted successfully into Person table
   MySQL/Postgres connection is closed
   34.4 s ± 458 ms per loop (mean ± std. dev. of 7 runs, 1 loop each)
   ```

   Note: your timing results may differ from those reported here.

   As usual, you should end up with 8,000,000 records in the Person table.

10. Summarize your results in a 1-2 page report (saved as `final_project_report.pdf`). Your report should follow the template provided below:

Section 1: Overview
- Provide background information on the project to help the reader understand what it was about and what methodology was used.

Section 2: Baseline run (Milestone 1)
- Provide a short description of what the baseline run entailed.
- Provide the time results you obtained.
- Include any observations you made and/or technical challenges you encountered during this stage.

Section 3: Batch runs (Milestone 2)
- Provide a short description of what the three batch runs entailed.
- Provide the time results you obtained for each run.
- Include any observations you made and/or technical challenges you encountered during this stage.

Section 4: Hardware upgrade runs (Milestone 2)
- Provide a short description of the two c2-standard-30 tests you ran.
- Provide the time results you obtained.
- Include any observations you made and/or technical challenges you encountered.

Section 5: Potential future improvements
- How would you expand on this study?
- What experiments would you consider if the timeline allowed for it?

Additional guidance on technical report writing:
- Include code samples or snippets to illustrate concepts.
- Keep paragraphs short: 5 sentences max.
- Keep sentences short: Aim to stay under 45 words.

CS 327E Milestone 2 Rubric
Due Date: 05/12/22 Due date is not flexible

Code blocks to insert in 5000, 50,000, and 500,000 batch sizes **-5** each execute() rather than executemany()/execute_batch() **-5** incorrect CSV file parsing logic used **-5** incorrect or incomplete output produced (e.g. missing %%timeit function) **-3** missing or incorrect try/except/finally blocks **-2** each missing markdown comment	30
Code block to insert 500,000 batch size data into high-memory server **-5** incorrectly/no increased CPU quota **-5** incorrect machine configuration changes **-5** incorrect or incomplete output produced (e.g. missing %%timeit function) **-5** used execute() rather than executemany()/execute_batch() **-3** missing or incorrect try/except/finally blocks **-2** missing markdown comment	25
Code block to insert 500,000 batch size data into high-bandwidth server **-5** missing/incorrect setup of server instance **-5** incorrect or incomplete output produced (e.g. missing %%timeit function) **-5** used execute() rather than executemany()/execute_batch() **-3** missing or incorrect try/except/finally blocks **-3** missing or incorrect database/schema and table creation **-2** missing Markdown	25
Report **-4** each missing/incomplete section (e.g. missing time results, observations/technical challenges, or irrelevant code snippets)	20
`milestone2.ipynb` and `final_project_report.pdf` pushed to your group's private repo on GitHub. Your project **will not** be graded without this submission.	**Required**

`submission.json` submitted into Canvas. Your project **will not** be graded without this submission. The file should have the following schema:
 	
`{`
`    "commit-id": "your most recent commit ID from GitHub",`
`    "project-id": "your project ID from GCP"`
`}`
 	
Example:
 	
`{`
`    "commit-id": "dab96492ac7d906368ac9c7a17cb0dbd670923d9",`
`    "project-id": "some-project-id"`
`}`
	Required

Total Credit:	**100**