Homework Assignment #2

Apache Cassandra Exploration

Deadline: 10/19/25 (Sunday) 11:45PM

Growing startup

As the ZotPets service begins to gain traction after its initial launch, the business is starting to grow. Your boss has heard that PostgreSQL can scale up but not out, so she is getting worried about possibly hitting a wall in terms of the size of the supportable customer base as the platform expands. As a result, your boss wants to explore the possible use of Apache Cassandra instead of PostgreSQL for ZotPets's backend database. Your boss wants you to get the company started down this path technically. Your second assignment is thus to prepare a subset of the data to be migrated from PostgreSQL to Cassandra for a proof-of-concept (POC) project, generating different Cassandra data model designs to answer a handful of queries, and then report back about (1) what you did and how and (2) the various implications of what you did.

Cassandra setup

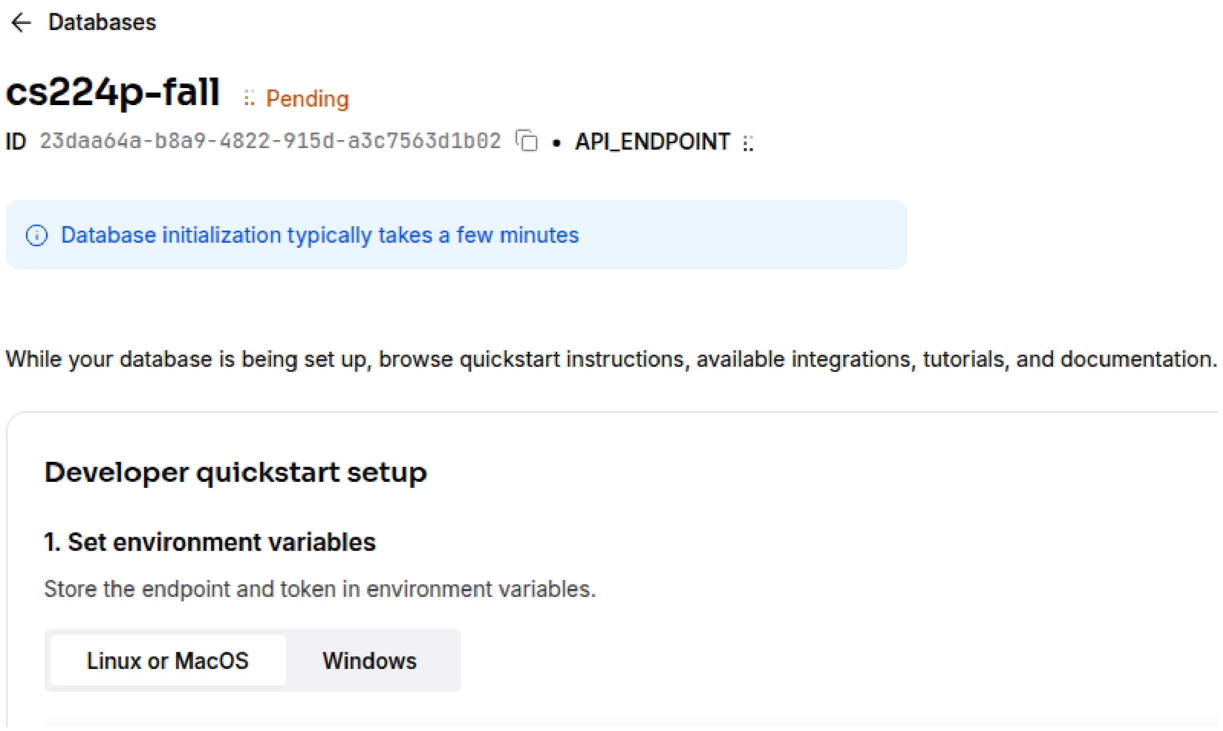
For this assignment, we will be using an online DBaaS service called DataStax Astra; it provides a "serverless" cloud-based implementation of Apache Cassandra. In this tutorial, we will use a small example database called "Hoofers" to walk you through the steps needed to get started with Astra/Cassandra. This example contains 3 tables, namely "Boats", "Sailors", and "Reserves". We will also be using your PostgreSQL database from Assignment #1 to create and export CSV files, so make sure you keep your PostgreSQL database on hand (or rerun a prefix of your HW1 script to recreate it).

1. Go to https://astra.datastax.com/ and sign up for a free Astra account.

2. After verifying your email and answering a few questions, you will be directed to your DataStax dashboard. Your free account is allowed up to $25 for free for Astra service use, and you should not reach that limit while doing this assignment!

3. Go to the "Databases" tab and create a database called "cs224p-fall". You will need to choose between two deployment types. Both the "Severless (Vector)" and the "Serveless (Non-Vector)" deployment types will work for this assignment. This instruction will use the "Serverless (Vector)" option. For the "Provider" and "Region" sections, choose Google Cloud and us-east1.

Note: your database might take a few minutes to be successfully created. While it is being created, you will see "pending" next to the database name.



← Databases

cs224p-fall Pending

ID 23daa64a-b8a9-4822-915d-a3c7563d1b02 • API_ENDPOINT

Database initialization typically takes a few minutes

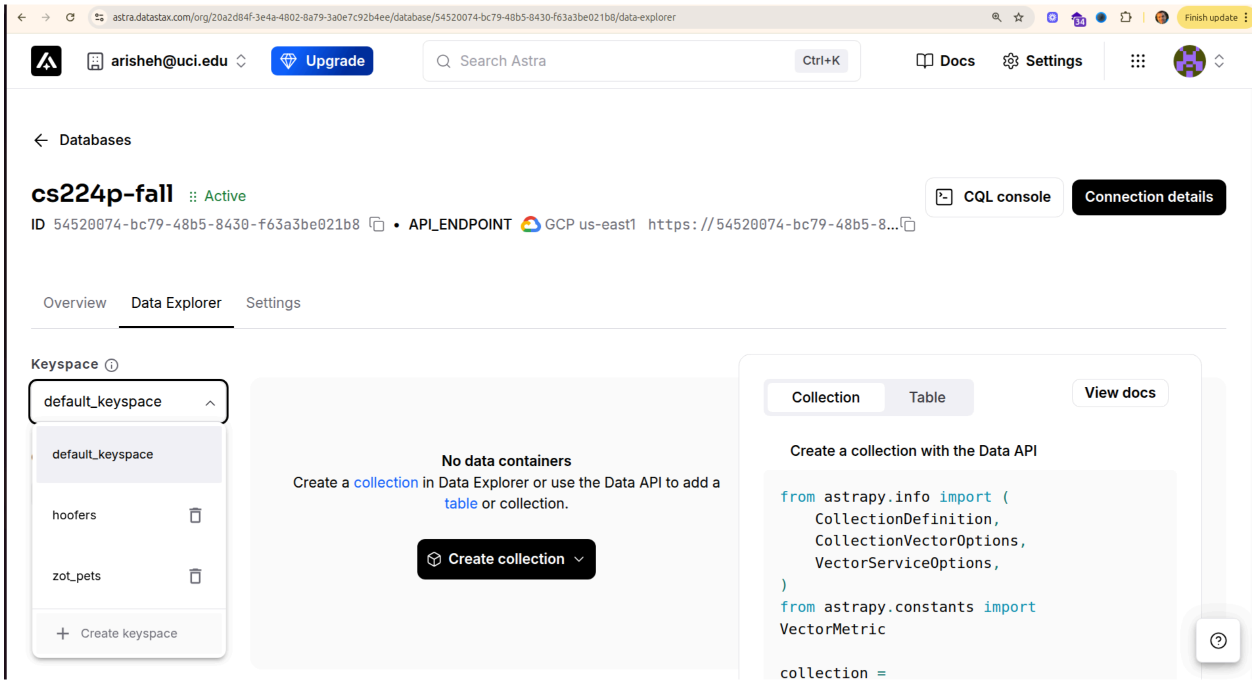
While your database is being set up, browse quickstart instructions, available integrations, tutorials, and documentation.

Developer quickstart setup

1. Set environment variables

Store the endpoint and token in environment variables.

4. Once you receive confirmation that your database was created, head over to the dashboard by clicking on your database name. Go to "Data Explorer" and create a keyspace (also known as namespace) called "hoofers", which we will load data into later. It will also take some time for the keyspace to be successfully created. Note Astra only allows creating a keyspace via the UI.



astra.datastax.com/org/20a2d84f-3e4a-4802-8a79-3a0e7c92b4ee/database/54520074-bc79-48b5-8430-f63a3be021b8/data-explorer
arisheh@uci.edu
Upgrade
Search Astra
Ctrl+K
Docs
Settings
Databases

cs224p-fall Active

ID 54520074-bc79-48b5-8430-f63a3be021b8 • API_ENDPOINT GCP us-east1 https://54520074-bc79-48b5-8...

No data containers

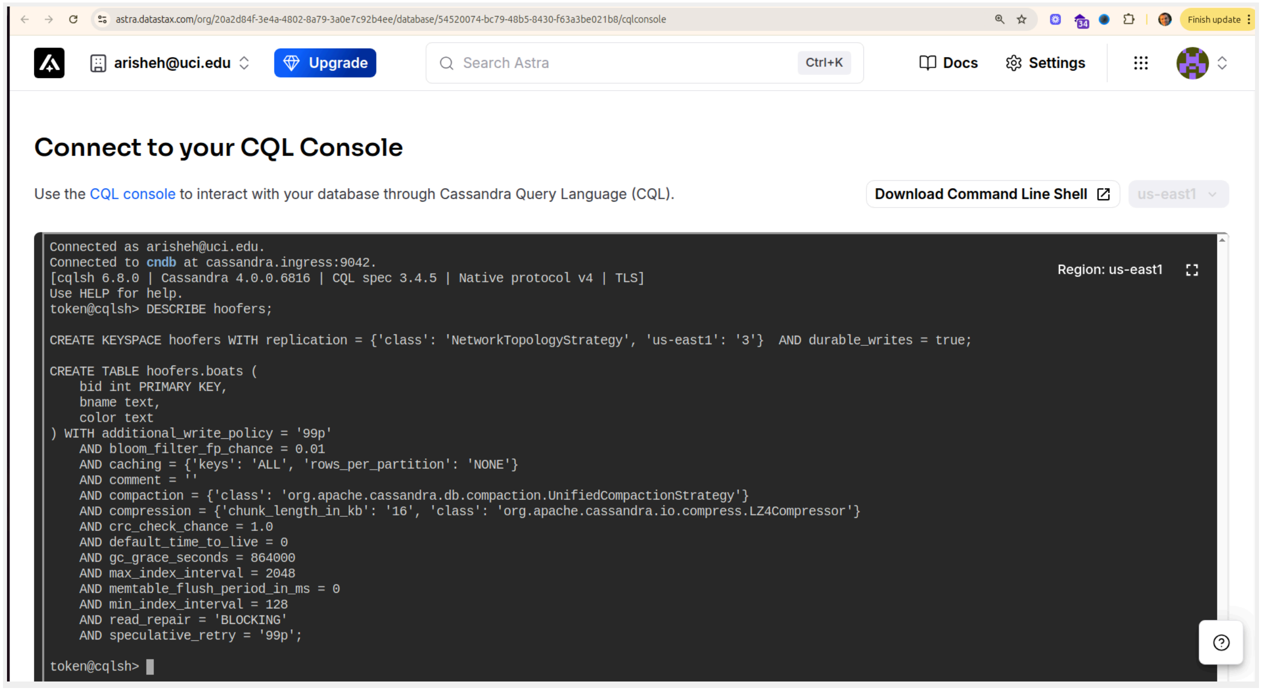
Create a collection in Data Explorer or use the Data API to add a table or collection.

Create a collection with the Data API

```
from astrapy.info import (
    CollectionDefinition,
    CollectionVectorOptions,
    VectorServiceOptions,
)
from astrapy.constants import
VectorMetric

collection =
```

5. Open the CQL console by clicking "CQL Console". This is where you will be running your queries. In the console, execute the following command to verify if the keyspace has been successfully created.

```
DESCRIBE hoofers;
```


astra.datastax.com/org/20a2d84f-3e4a-4802-8a79-3a0e7c92b4ee/database/54520074-bc79-48b5-8430-f63a3be021b8/cqlconsole
arisheh@uci.edu
Upgrade
Search Astra
Ctrl+K
Docs
Settings
Connect to your CQL Console
Use the CQL console to interact with your database through Cassandra Query Language (CQL).
Download Command Line Shell
us-east1
Region: us-east1

```
Connected as arisheh@uci.edu.
Connected to cndb at cassandra.ingress:9042.
[cqlsh 6.8.0 | Cassandra 4.0.0.6816 | CQL spec 3.4.5 | Native protocol v4 | TLS]
Use HELP for help.
token@cqlsh> DESCRIBE hoofers;

CREATE KEYSPACE hoofers WITH replication = {'class': 'NetworkTopologyStrategy', 'us-east1': '3'}  AND durable_writes = true;

CREATE TABLE hoofers.boats (
    bid int PRIMARY KEY,
    bname text,
    color text
) WITH additional_write_policy = '99p'
    AND bloom_filter_fp_chance = 0.01
    AND caching = {'keys': 'ALL', 'rows_per_partition': 'NONE'}
    AND comment = ''
    AND compaction = {'class': 'org.apache.cassandra.db.compaction.UnifiedCompactionStrategy'}
    AND compression = {'chunk_length_in_kb': '16', 'class': 'org.apache.cassandra.io.compress.LZ4Compressor'}
    AND crc_check_chance = 1.0
    AND default_time_to_live = 0
    AND gc_grace_seconds = 864000
    AND max_index_interval = 2048
    AND memtable_flush_period_in_ms = 0
    AND min_index_interval = 128
    AND read_repair = 'BLOCKING'
    AND speculative_retry = '99p';

token@cqlsh>
```

6. For purposes of loading data into Datastax Astra, we will be using a software tool called the **DataStax Bulk Loader**. This is to make sure that you learn how to manually create tables in CQL before loading data, and also that you experience the task of uploading bulk data into a database service in the cloud (a skill whose relevance is rapidly increasing over time).

7. Download DSBulk from the following link: https://downloads.datastax.com/#bulk-loader and unpack the downloaded distribution. Note: DSBulk requires a Java Runtime Environment. (If you do not have a JRE, make sure to download that as well, but most of you probably have Java set up already.)

8. Add the downloaded file to your PATH so that you are able to use the "dsbulk" command. This process can differ depending on your OS. a. For MAC/Linux users: on the command line run the following:

```
export PATH=path-to-unpacked-location/dsbulk-1.10.0/bin:$PATH
```

(You might want to add this to your favorite shell's .xxxrc file. Please make sure the .xxxrc file has the correct PATH of bin, and reload the terminal once changes have been made to .xxxrc by saying, e.g., source ~/.bashrc .) b. For Windows users, the following link might help.

https://www.architectryan.com/2018/03/17/add-to-the-path-on-windows-10/

9. Verify that dsbulk is working properly by running the command:

```
$ dsbulk --version
```

```
DataStax Bulk Loader v1.11.0
```

Let's test out the workflow involved in loading data into Astra by first exporting data from PostgreSQL and then importing it using DSBulk by experimenting with the 'hoofers' data.

10. Create a database called "hoofers" in PostgreSQL. Run the following SQL script in PostgreSQL to create the tables and insert the data: HoofersDB.sql

11. Export the "Boats" table into a CSV file; below are some useful hints on how to do so in PostgreSQL either using pdAdmin or command line:

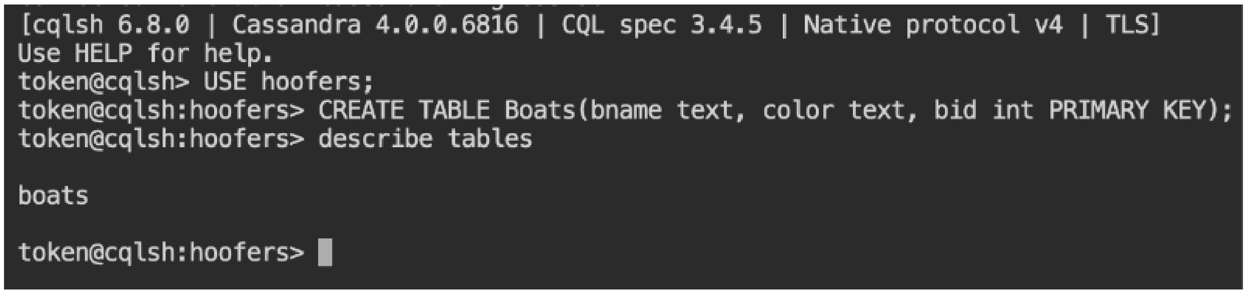
Using pSQL command line:

```
SQL
-- Suppose you want to fully export a table named Person into a csv file called
persons.csv, this file will be created at runtime --
\copy Person to '/desired-path/persons.csv' DELIMITER ',' CSV HEADER;

-- Now suppose you instead want to export the results of a query on the Person
table into a csv file called results.csv, this file will be created
at runtime --

\copy (SELECT * FROM Person P WHERE P.person_id = ‘1234’) to
'/desired-path/persons.csv' DELIMITER ',' CSV HEADER;

-- The process would be the same for exporting the results of a JOIN query --

\copy (SELECT * FROM Person P, Username U WHERE P.uid = U.uid) to
'/desired-path/persons.csv' DELIMITER ',' CSV HEADER;
```

Now that we have the CSV file, we can begin the process of loading data into Astra.
12. In the Astra CQL console, first execute the following command to switch to hoofers keyspace.

```
USE hoofers;
```

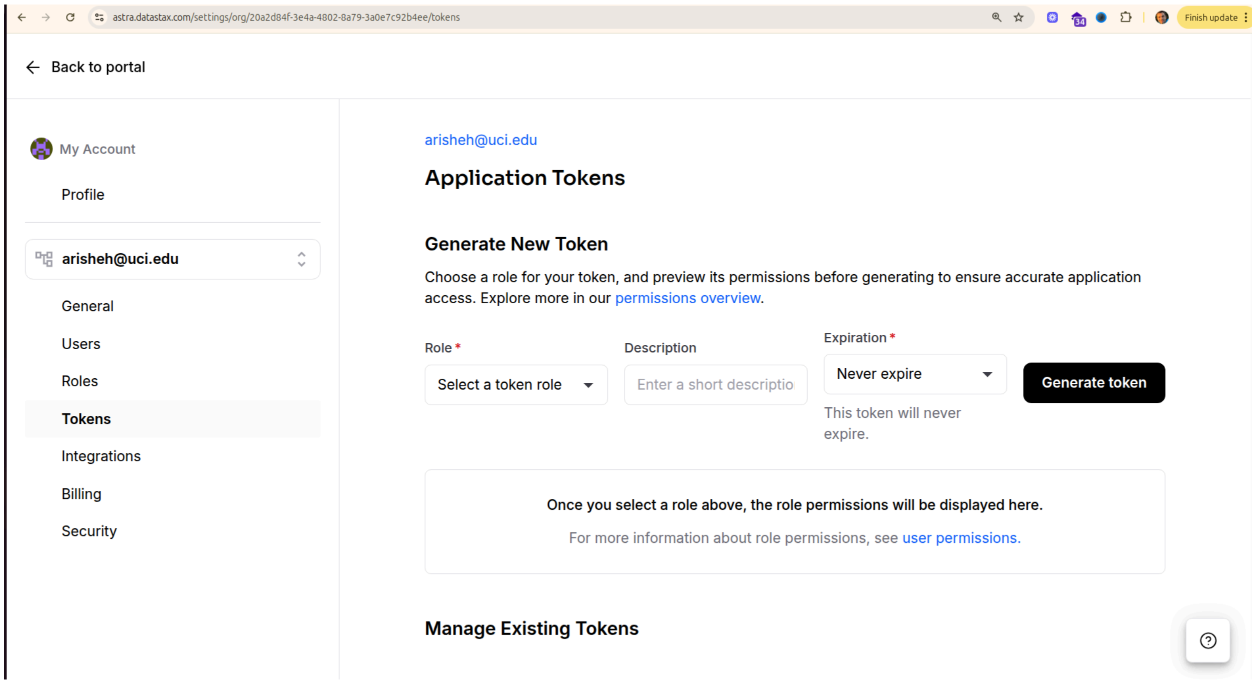
Then create a table called "Boats" by translating the PostgreSQL CREATE TABLE statement given in the script.



```
[cqlsh 6.8.0 | Cassandra 4.0.0.6816 | CQL spec 3.4.5 | Native protocol v4 | TLS]
Use HELP for help.
token@cqlsh> USE hoofers;
token@cqlsh:hoofers> CREATE TABLE Boats(bname text, color text, bid int PRIMARY KEY);
token@cqlsh:hoofers> describe tables

boats

token@cqlsh:hoofers>
```

```SQL
CREATE TABLE Boats(bname text, color text, bid int PRIMARY KEY);
```

Next we need to load data into Cassandra. This involves the following four steps:
13. Generate a token:
Click on "Settings" on top right of the page, then select the "Tokens" tab from the left bar to go to the Application Tokens page.



astra.datastax.com/settings/org/20a2d84f-3e4a-4802-8a79-3a0e7c92b4ee/tokens

arisheh@uci.edu

Application Tokens

Generate New Token

Choose a role for your token, and preview its permissions before generating to ensure accurate application access. Explore more in our permissions overview.

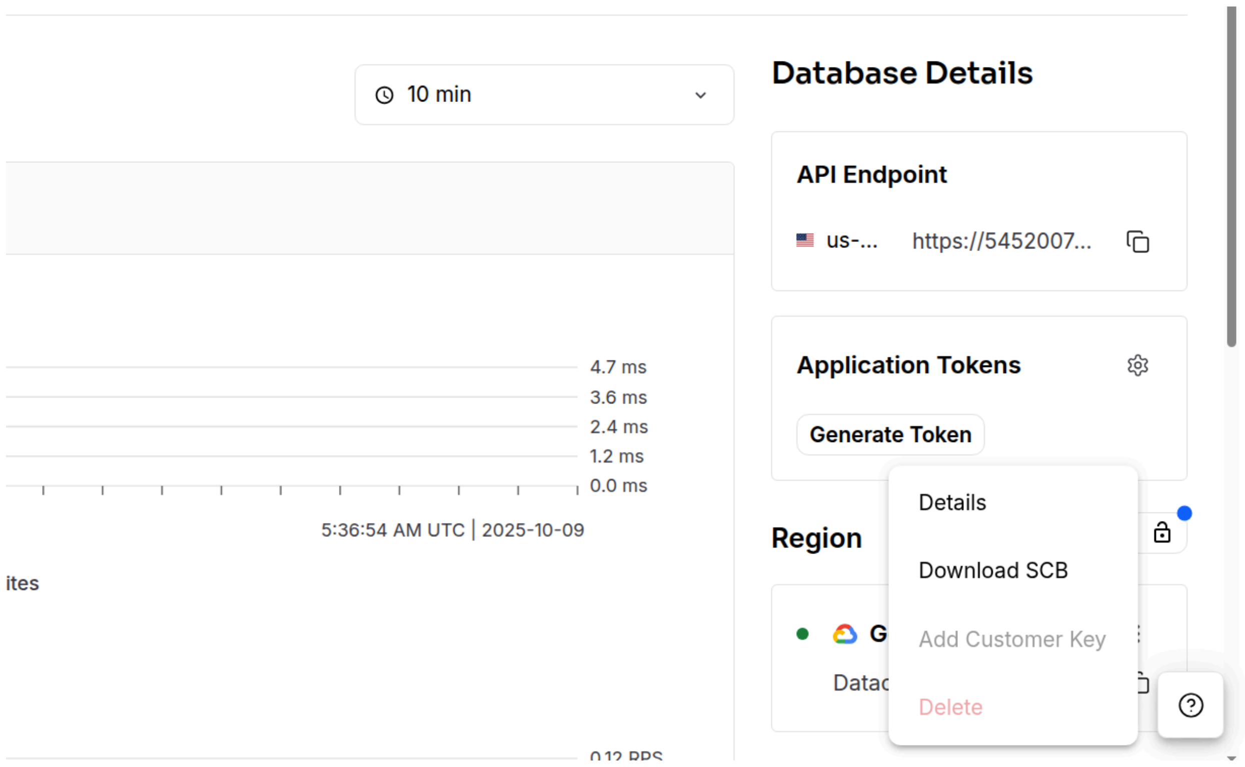
Role * Select a token role
Description Enter a short descriptio
Expiration * Never expire
This token will never expire.

Once you select a role above, the role permissions will be displayed here.
For more information about role permissions, see user permissions.

Manage Existing Tokens

You will be shown the token details with a link to download these as a csv file. Click "Download Token Details". In the downloaded file, the first two fields - client ID and client secret – will be used as credentials for the bulk load command.

14. Go back to your database's Dashboard and download the Secure Connect Bundle(SCB) by clicking the "..." in the Region panel - [Secure Connect DataStax Astra Documentation.]

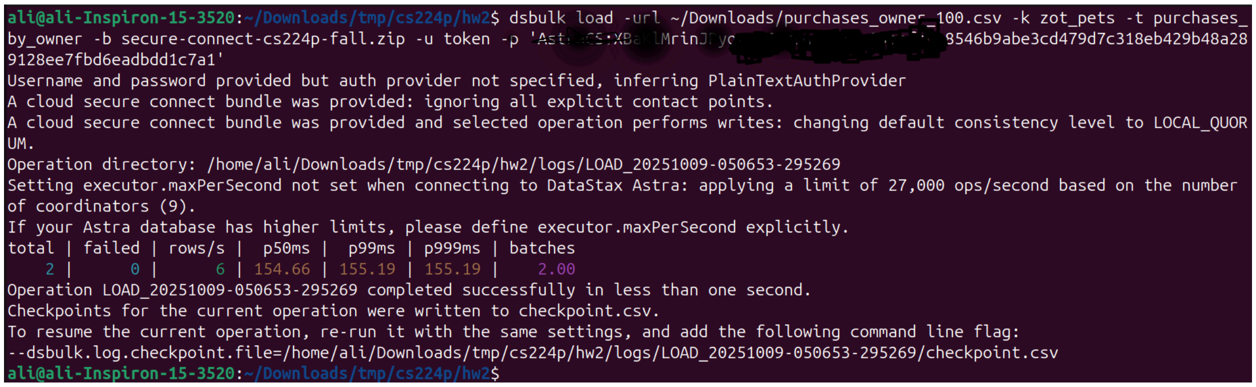


15. The downloaded result is a compressed .zip file (use Firefox or Chrome).

16. [Recommended] Put the secure connect bundle, dsbulk and the boats.csv file in the same folder, and navigate to this location in your terminal window.

17. Bulk load the boats data using the following command:

```None
% dsbulk load -url boats.csv -header true -k hoofers -t boats
-b "/path/to/secure-connect-cs224p-fall.zip" -u CLIENTID -p CLIENTSECRET

-url: the path to the data file (which, for us, is boats.csv)
-header: specifies that the first row in the data file is a header with
information about the attributes
-k: the keyspace name
-t: the table name
-b: the path to the application bundle
-u and -p are the client id and secret values that will be found in the app
token JSON file from
```



```
ali@ali-Inspiron-15-3520:~/Downloads/tmp/cs224p/hw2$ dsbulk load -url ~/Downloads/purchases_owner_100.csv -k zot_pets -t purchases_
by_owner -b secure-connect-cs224p-fall.zip -u token -p '[redacted]8546b9abe3cd479d7c318eb429b48a28
9128ee7fbd6eadbdd1c7a1'
Username and password provided but auth provider not specified, inferring PlainTextAuthProvider
A cloud secure connect bundle was provided: ignoring all explicit contact points.
A cloud secure connect bundle was provided and selected operation performs writes: changing default consistency level to LOCAL_QUOR
UM.
Operation directory: /home/ali/Downloads/tmp/cs224p/hw2/logs/LOAD_20251009-050653-295269
Setting executor.maxPerSecond not set when connecting to DataStax Astra: applying a limit of 27,000 ops/second based on the number
of coordinators (9).
If your Astra database has higher limits, please define executor.maxPerSecond explicitly.
total | failed | rows/s |  p50ms |  p99ms | p999ms | batches
    2 |      0 |      6 | 154.66 | 155.19 | 155.19 |    2.00
Operation LOAD_20251009-050653-295269 completed successfully in less than one second.
Checkpoints for the current operation were written to checkpoint.csv.
To resume the current operation, re-run it with the same settings, and add the following command line flag:
--dsbulk.log.checkpoint.file=/home/ali/Downloads/tmp/cs224p/hw2/logs/LOAD_20251009-050653-295269/checkpoint.csv
ali@ali-Inspiron-15-3520:~/Downloads/tmp/cs224p/hw2$
```

You are now armed and ready to start this assignment! Good luck!

Getting started

Core to ZotPets's business model are its users (including pet owners and groomers), the pets they own, and the appointments they schedule. Your boss has identified this subset of ZotPets's overall schema as a good target for your POC work. She wants you to test-drive Cassandra with some basic tables to explore its key-related functionality, which she's heard takes some getting used to. Then she would like you to explore its "no joins" (or more accurately, pre-joined) approach to schema design. For data, she wants you to export tables and query results from PostgreSQL into CSV files and then upload them into the tables of the Cassandra keyspace that you will be designing, defining, and querying in the course of your POC work.

1. Understanding Your Cassandra Environment

Before working with ZotPets data, you should have a sample keyspace called "Hoofers" from the setup document. Use the DESCRIBE CQL command to get information about that keyspace.

A. How many copies of the data does the Hoofers keyspace maintain?
B. Which cloud region does it reside in?
C. What should the read and write quorum sizes (R & W) be for consistent reads and writes?

2. Initial Table Creation in CQL

To get acquainted with CQL, your first task is to translate the PostgreSQL CREATE TABLE DDL for the user, pet, and appointment tables from the provided zot_pets.ddl file into equivalent CQL statements. First, create a new keyspace named zot_pets, then use the DDL file from homework 1 to create three tables: user, pet, appointment. You should keep the table structures as similar as possible, using the same column names, equivalent data types (e.g., INT, VARCHAR, TIMESTAMP), and the same primary keys defined in the DDL file.

3. Data Export and Import

Next, you will move the data from PostgreSQL to Cassandra.In PostgreSQL, use the COPY command to export the user, pet, and appointment tables into three separate CSV files with headers. Once you have the CSV files, use the Cassandra "dsbulk" utility to upload the data into the corresponding Astra tables you created in the previous step.

4. Querying on a Non-Primary Key
With all your data loaded into Astra, let us try a simple filter (WHERE) query. In a relational database, you could easily query any column. In Cassandra, things are different, as the system is designed to prevent potentially slow or resource-intensive queries that don't use the primary key. Write a CQL query to look up the name, owner_id, and dob of all pets whose species is 'Dog'. Use the pet table you created earlier. Try running your query "as is". Based on the resulting error message, use an appropriately modified command to "force" Cassandra to execute this query and show the result.

5. Designing a Table for the Query
Unlike a relational database, where you model data first and write queries later, in Cassandra, queries are brought to the forefront. The "force" command used in question 4 was a "band-aid" solution that should be avoided in production, as it can lead to slow and unpredictable performance. Instead, the correct approach is to create a new, denormalized table designed specifically for your query.

Create a new table (e.g., `pets_by_species`) with a primary key designed to efficiently find pets by their species. The primary key should be (species, `owner_id, pet_id`), with the table partitioned only on species. Load the original pet.csv data into this new table using dsbulk. Perform the same query from question 4 (finding all dogs) on this new table. This time, it should execute without needing ALLOW FILTERING. Briefly explain why changing the partitioning key made the query efficient. Briefly explain why it was necessary to include `owner_id` and `pet_id` in your primary key and not just use species.

6. Designing for Sort Order
In addition to being aware of how data is partitioned, Cassandra users must also be very aware of how their data is physically sorted. Suppose your query from question 4 changes slightly. Now, you are interested in finding the name and dob (date of birth) of the 5 youngest pets of the species '`Dog`'. This requires sorting the results.

A. Write the new query in CQL (ORDER BY dob DESC LIMIT 5) and try to execute it on the table you created in question 5 `(pets_by_species)`. Observe the error Cassandra gives you.
B. Unfortunately, there is no command that will force Cassandra to run this query. The solution, once again, is a new table design. Create another new table (e.g., `pets_by_species_sorted_by_age`) with a partitioning key on its species field and clustering keys on `dob, owner_id, and pet_id`. Note that the order of the keys matters here. To satisfy the query, you must also specify the clustering order to sort by dob in descending order.
C. Load the pet.csv data into this new table.
D. Execute your top-5 query on this new table. It should now succeed.
E. Briefly explain why adding dob as the primary clustering key with a specified descending order allowed Cassandra to run your query.

F. Finally, clean up your keyspace by dropping the tables you created in question 5 and question 6.

7. Designing for Application Queries

Now that you understand the implications of primary and clustering keys, you are ready to tackle some parameterized queries that your boss wants ZotPets to support! For each of the following four query requirements, your task is to design and create one new Cassandra table that can efficiently answer that specific query. You will need to think carefully about what data to include (denormalization may be required, similar to how `species` was added to the `pet` data in the last table) and how to structure the primary key for each table. Use the same column names and data types from the original DDL where applicable.

The desired queries:

A. List the `pet_id` and `name` of all pets registered by a particular pet owner (`owner_id`), returning the pets in order from oldest to youngest (`dob` ascending).
B. Count the number of services offered for a particular `service_type` (e.g., 'Bath', 'Haircut').
C. List the details of all appointments (`appointment_id`, `pet_owner_id`, `pet_id`) handled by a specific groomer (`groomer_id`), ordering the results by the appointment time (`appointment_datetime`) from most recent to oldest.
D. For all purchases made by a specific pet owner (`pet_owner_id`) within a given time range, find the price of the most expensive item they bought (`MAX(price)`).

8. Executing and Verifying the Queries

Your boss is impressed with your table designs and now wants to see them in action. To see the fruits of your data modeling labor, you will now run concrete versions of the parameterized queries from question 7. For each of the four scenarios below, you must perform the entire workflow:

1. Write a PostgreSQL query (using joins if necessary) to produce the required denormalized data.
2. Export the result of that query to a new, uniquely named CSV file.
3. Use `dsbulk` to load the new CSV file into the appropriate Cassandra table you created in question 7.
4. Run the final CQL query in Cassandra to get the answer.

The Test Queries and Parameters:

A. (For table `pets_by_owner_sorted_by_age`) List the `pet_id`, `name`, and `dob` of all pets registered by the pet owner with `owner_id = 10`.

B. (For table `services_by_type`) Find out how many services have been created with the `service_type` of 'Bath'.
C. (For table `appointments_by_groomer`) List the `appointment_id`, `pet_owner_id`, and `pet_id` of the 10 most recent appointments handled by the groomer with `groomer_id = 6`.
D. (For table `purchases_by_owner`) For the pet owner with `pet_owner_id = 5`, find the price of the most expensive item they bought during September 2021 to September 2025 (from `'2021-09-01'` to `'2025-09-30'`).

What to turn in?

For this homework, you should submit two files:

- A PDF file that includes all the queries, commands, and execution results
- A TXT file that includes all the SQL, CQL queries and dsbulk commands that you ran

Notes:

- Write queries and results in the order of questions in the homework. For example, your first query and its results should belong to question 1.
- SQL queries should be run before CQL commands. Keep the order.
- We can run your queries during grading. So make sure the sequence of commands and queries for each question is correct.