Show Submission Credentials

# P3. Cloud Storage - SQL and NoSQL Cloud Storage - SQL and NoSQL

Introduction

## SQL and NoSQL

Information

### Learning Objectives

This project will encompass the following learning objectives:

1. Compare the advantages and disadvantages of SQL and NoSQL databases and their suitable application domains.
2. Design and develop efficient database queries using the Java API for databases to fetch data from heterogeneous SQL and NoSQL databases.
3. Implement and evaluate MySQL indexing to improve the performance of MySQL queries.
4. Deploy Hibernate Application to learn how and when to use Object-Relational Mapping (ORM).
5. Deploy analysis using Java API for HBase as a column-based store.
6. Implement infrastructure as code using Terraform and Azure Resource Manager (ARM) to orchestrate and manage virtual machines, databases, and HBase clusters.

Information

### General Details

The following table contains general information about this project module:

| | |
|---|---|
| Prerequisites | Java, MySQL, and MySQL Connector |
| Primers | NoSQL Primer, HBase Basics, MySQL Primer |
| Applicable languages | Python for Q1 and Java for the rest of the questions |

| | |
|---|---|
| Applicable cloud platform | Azure Only |
| Tags Required | Key: `project` Value: `cloud-storage` |

**Danger**

### Special Deadline

You will only have **one week** to finish this section of Project 3, instead of the regular two-week period.

### Resource Tagging

For this project, assign the tag with Key: **project** and Value: **cloud-storage** for all resources.

### Grading Penalties

The following table outlines the violations of project rules and their corresponding grade penalties for this week's project.

These rules apply for the week when the current project is active.

| **Violation** | **Penalty of the project grade** |
|---|---|
| Incomplete submission of required files (including all the given Java files) | -10% |
| Submitting your Azure or Andrew credentials in your code for grading | -100% |
| Submitting only executables ( `.jar` , `.pyc` , etc.) without human-readable code ( `.py` , `.java` , `.sh` , etc.) | -100% |
| Attempting to hack/tamper the autograder in any way | -100% |
| Cheating, plagiarism, or unauthorized assistance (please refer to the university policy on academic integrity and our syllabus) | -200% & potential dismissal |

## Tagging and Budget Notes on Azure

We suggest that you tag your Azure resources. You will not incur any penalty if you do not tag resources in this project.

Be sure to use the correct subscription when you are using the Azure portal or Azure CLI. If you have multiple subscriptions, you can use the command below to change the default subscription.

```
az account list --output table --refresh
az account set --subscription <subscription_id>
```

**In this project, the recommended Azure budget is $50. Your subscription will be disabled for the entire course if you overspend the total subscription budget. You will NOT be able to continue working on Azure in this course and we will not increase the budget or reactivate your subscription.**

We suggest that you plan the budget carefully. Note that Azure Classroom subscription has its own cost monitoring portal which is different from the Cost Management + Billing (https://portal.azure.com/#blade/Microsoft_Azure_Billing/ModernBillingMenuBlade/BillingAccounts) blade used by most subscription types. **To check the remaining budget of Azure Classroom subscriptions, please visit Azure Education Hub (https://portal.azure.com/#blade/Microsoft_Azure_Education/EducationMenuBlade/overview)**.

Information

### Grade Distribution

| | |
|---|---|
| MySQL Task | 53 points |
| HBase Task | 32 points |
| Reflection and Discussion | 5 points |
| Manual Grading | 10 points |

## Overview

In recent years, people have gradually realized the power and value of data. Everybody is talking about "data": big data, data mining, data visualization, etc. But what is data? Let's loosely define data as simply a collection of raw facts and figures. Applications are responsible for generating, storing, analyzing, and consuming data, or some combination of these.

Data storage is important - without storage, data cannot be persisted, measured, and further analyzed to gather insights. In this project, we will be exploring and experiencing several common ways of storing and manipulating data. After this project, you should have a better understanding of a few common storage abstractions and be able to choose one over another in real-world scenarios.

In the first few sections, we will introduce relational databases.

Databases organize and structure data to make data quickly accessible and manipulatable by applications. To help clarify what unstructured or structured data means, consider the following example which shows how the same line of data can be stored and accessed in a file and database.

Here are various possible representations of a single record in a file, for example:

- comma-separated values (CSV) format: `Carnegie,Cloud Computing,A,2018`
- tab-separated values (TSV) format: `Carnegie\tCloud Computing\tA\t2018`
- a custom and verbose format: `Name: Carnegie, Course: Cloud Computing, Section: A, Year: 2018`

The box below shows a single line (row) in a database table of four columns, note that the first line is column names

```
+----------+-----------------+---------+------+
| Name     | Course          | Section | year |
+----------+-----------------+---------+------+
| Carnegie | Cloud Computing | A       | 2018 |
+----------+-----------------+---------+------+
```

As you can see, for a database, the data is represented in a more structured way, as a **table**, which makes it easier for users of this data to access specific parts of the data.

A **transaction** is any work that is done on the database. It should be noted that one transaction can have multiple operations performed on the database. Traditional relational databases satisfy the **ACID** properties for transactions.

**Atomicity** - A transaction is a complete unit. It either succeeds as a whole or fails as a whole.

**Consistency** - The state of the database has to be valid before and after a transaction completes. Some constraints or rules ensure the validity of data in a database. These are applied to check data validity after a transaction completes. If a transaction results in invalid data, the transaction is rolled back.

**Isolation** - Multiple transactions are allowed to run concurrently to improve performance. Hence, we must ensure that the transactions which are executed concurrently should leave the database in the same state as if the transactions were executed sequentially.

**Durability** - Transactions that are committed, completed successfully, should be persisted permanently regardless of server crashes or restarts.

**Please read the NoSQL primer (https://projects.sailplatform.org/s22-15619/nosql-primer) to understand the CAP theorem.**

In addition to traditional SQL databases, NoSQL databases are becoming a popular method of storing data. The motivation for NoSQL was driven by the big-data challenges of scale and performance. NoSQL solutions have become popular choices for certain types of applications that require performance at scale. NoSQL trades off some of the constraints (consistency, structure, etc.) that exist in SQL databases in exchange for performance and scalability. It is important to understand that the consistency mentioned in the CAP theorem is different from the consistency mentioned in the ACID properties of the database. ACID talks about transactional consistency whereas CAP theorem talks about the consistency of a distributed system where 2 replicated databases read the same value.

NoSQL databases trade off consistency for availability. NoSQL databases are explained using the **BASE** acronym.

**Basically Available** - The database guarantees high availability

**Soft state** - The state of the database can change over time even though there is no input. The changes made to the database will propagate in the system without blocking other requests from the user. This is because there may be changes going on due to eventual consistency. Hence, the state is considered soft

**Eventually consistent** - The data in the database may sometimes be stale but it will become consistent after a period of time.

In later sections, you will learn how to use one of the most popular NoSQL databases, **HBase**, to complete several queries that are quite similar to the ones you will do in the **MySQL** section. After completing all the tasks, you should not only be able to write SQL, HBase, MongoDB, or Neo4j commands to perform queries but more importantly, realize some of their advantages and disadvantages to help decide which one to use depending on the real-world scenario that you might encounter in your startup or career.

## The Scenario

You recently changed jobs and joined Carnegie SoShall (CS), a large firm that is planning to roll out a global social network. Before assigning you to a specific development team, the company wants to show your skills by working with Yelp's public dataset (https://www.yelp.com/dataset). Before adopting cloud storage services, Carnegie SoShall would like your help to evaluate the functionality and limitations in adopting SQL and NoSQL databases to query and perform basic analytics on different kinds of data.

## Provision VM instances using Terraform

Infrastructure as code (IaC) is the process of managing and provisioning data center infrastructure through machine-readable definition files, in contrast to the traditional interactive configuration tools or imperative commands.

In this project, you will practice using Terraform and the Azure Resource Manager (ARM) to orchestrate and manage virtual machines and HBase clusters on Azure.

To complete this project, you will work on a `Standard_B2ms` VM instance using the VHD file provided by us.

1. **You should use Azure Cloud Shell to launch the student VM.** If you haven't tried it before, please refer to this page (https://docs.microsoft.com/en-us/azure/cloud-shell/overview) to learn about it. You will be asked to choose between Bash and Powershell when you launch cloud shell for the first time, and you should choose 'bash' for this project.
2. Then, make sure you’re authenticated with Azure. To do this, either use the Azure Cloud Shell or Azure CLI on your local machine and log in with it.

```
az login
```

3. **Warning: As you may have multiple subscriptions, make sure you are using the correct subscription for this project.**

```
az account set --subscription $SUBSCRIPTION_ID
```

4. Run the following terraform commands to set up the student instance.

```
wget https://clouddeveloper.blob.core.windows.net/assets/cloud-storage/project/build-azure-vm-v2.tgz
tar -xvzf build-azure-vm-v2.tgz

# You will need to set the password needed to ssh into your VM
# The password must be at least 10 characters in length and must contain
# at least one digit, one non-alphanumeric character, and one upper or lower case letter.

export TF_VAR_password=<input_password>

terraform init

terraform apply
```

5. Obtain the instance public IP and virtual network ID from the output instructions of Terraform. You can note down the virtual network ID, which will be used when you provision HBase clusters in Task 2.
6. Open the URL `http://[INSTANCE_PUBLIC_IP]` in your web browser. Enter your submission username and submission password. Then click the Launch Project button of **P3. Cloud Storage - SQL and NoSQL**. Wait for several minutes until the log tells you that the instance is ready.
7. SSH into the VM as `clouduser` with the password you set when you provision the VM. **You should always use your local machine to ssh into the VM to prevent timeout.**

```
ssh clouduser@"$INSTANCE_PUBLIC_IP"
```

8. You can get the starter code for Task 1 and Task 2 under `/home/clouduser/Project_Database` .

## The Input Data

1. The Yelp data files are stored at `https://clouddeveloper.blob.core.windows.net/datasets/cloud-storage/` , which is a subset of Yelp's dataset (https://www.yelp.com/dataset). The Yelp dataset contains 5 TSV files that we will be working with. You can use the following bash commands to download the data into the directory `Project_Database` in your VM instance.

```
declare -a data_files=(businesses checkin review tip user)
for data_file in "${data_files[@]}"; do
  wget https://clouddeveloper.blob.core.windows.net/datasets/cloud-storage/yelp_academic_dataset_"$data_
file".tsv
done
```

**You should download the data to your VM instance only. Don't download this to your local machine as it will take too long and will be a costly operation.**

2. You can validate the TSV files with `wc -l *.tsv` and the expected output is as follows:

```
  174568 yelp_academic_dataset_businesses.tsv
  146351 yelp_academic_dataset_checkin.tsv
  153190 yelp_academic_dataset_review.tsv
  643783 yelp_academic_dataset_tip.tsv
  357182 yelp_academic_dataset_user.tsv
 1475074 total
```

## Data Schemas

These `.tsv` files contain a couple of GBs of real Yelp data on businesses, users, tips, and check-ins delimited by the tab character `\t`.

The first row in each TSV is the column header which shows the schema. You can check the first line of each file to discover the order of the columns.

```
head -1 *.tsv
```

You can view the files interactively by using `less`, e.g.

```
less yelp_academic_dataset_businesses.tsv
```

The scrolling and navigating features are powerful in `less`, e.g. you can scroll forward half a page by pressing `d`. For more details, you can read the manual (https://linux.die.net/man/1/less).

Task 1: Introduction to MySQL

# MySQL

**Before you move on, we require you to complete the MySQL primer.**

## Scenario

Carnegie SoShall wants you to evaluate MySQL on data loading and queries so that it can decide whether to adopt SQL [Task 1] or NoSQL [Task 2] databases for its needs.

## JDBC and MySQL Basics

### Introduction

The Java Database Connectivity (JDBC) API is the industry standard for database-independent connectivity between the Java programming language and a wide range of databases including SQL databases and other tabular data sources, such as spreadsheets or flat files. The JDBC API provides a call-level API for SQL-based database access.

JDBC technology allows you to use the Java programming language to exploit "Write Once, Run Anywhere" capabilities for applications that require access to data. Using JDBC drivers, you can connect all Carnegie SoShall's data even if they were in a heterogeneous environment.

In this task, you will use MySQL Connector, an official JDBC driver for MySQL, to connect to your local MySQL server and execute queries to answer questions.

### Establish Database Connections

When you are writing a Java program to perform any operations on a MySQL database, your program should first establish a connection to the database. To do so, use

```
Class.forName("com.mysql.jdbc.Driver");
Connection conn = DriverManager.getConnection(URL, DB_USER, DB_PWD);
```

The first line of code `Class.forName("com.mysql.jdbc.Driver")` loads and initializes the MySQL JDBC driver. The second line makes a `Connection` to the database by calling `DriverManager.getConnection(URL, DB_USER, DB_PWD)`. As you can see, three parameters are passed here. We leave you to search and find out what these parameters mean.

### Statement and ResultSet

In order to execute queries using your established database connection, you will need to create a `Statement`. A `Statement` is defined as an object used for executing a raw SQL query and returning the results. The returned results are often stored in a `ResultSet` object. An example of how to use `Statement` is given below.

```
Statement stmt = conn.createStatement();
String neighborhood = "Shadyside";
String sql = "SELECT name FROM businesses WHERE neighborhood = '" + neighborhood + "'";
ResultSet rs = stmt.executeQuery(sql);
```

To retrieve results from `ResultSet`, you can iterate through the `ResultSet` object by calling `rs.next()`:

```
while (rs.next()) {
    // rs.getString(1) is equivalent to rs.getString("neighborhood")
    System.out.println(rs.getString(1));
}
```

### PreparedStatement vs Statement

PreparedStatement represents a precompiled statement:

```
String neighborhood = "Shadyside";
PreparedStatement pstmt = conn.prepareStatement("SELECT name FROM businesses WHERE neighborhood = ? ");
pstmt.setString(1, neighborhood);
ResultSet rs = pstmt.executeQuery();
```

Executing a raw SQL query with `Statement` is the most basic JDBC usage you should learn. While `Statement` is useful, `PreparedStatement` is more powerful and preferred for the following reasons:

1. PreparedStatement is precompiled and DB-side cached. It leads to faster execution, *especially when the same query needs to be executed multiple times. Please keep this in mind if you need to build a high-performance server with MySQL database as the back-end in the future.*
2. PreparedStatement is more readable. We can pass different parameters at runtime using setter methods.
3. PreparedStatement helps to avoid SQL injection attacks. SQL injection is a common malicious code injection technique that your applications can be vulnerable to and easily be hacked when they lack protective measures.

   Suppose you have built a web server and the value of the field `neighborhood` is parsed from a request and used as follows:

   ```
   Statement stmt = conn.createStatement();
   String neighborhood = "Shadyside' or '1'='1"; # parsed from a malicious web request
   String sql = "SELECT name FROM businesses WHERE neighborhood = '" + neighborhood + "'";
   ResultSet rs = stmt.executeQuery(sql);
   ```

   Give it a try and see what happens.

   In order to prevent such an injection, PreparedStatement, by default, uses parameterization, built-in escaping of quotes, and other special characters. However, you should use the PreparedStatement properly to set values with setter methods and **NOT** by concatenating the string in the raw query.

   ```
   String neighborhood = "Shadyside";
   // WRONG USAGE
   PreparedStatement pstmt = conn.prepareStatement("SELECT name FROM businesses WHERE neighborhood = '" + n
   eighborhood + "'");
   ResultSet rs = pstmt.executeQuery();

   String neighborhood = "Shadyside";
   // CORRECT USAGE
   PreparedStatement pstmt = conn.prepareStatement("SELECT name FROM businesses WHERE neighborhood = ? ");
   pstmt.setString(1, neighborhood);
   ResultSet rs = pstmt.executeQuery();
   ```

### Clean up

Remember to clean up your resources after you are done using them! Close your `ResultSet`, `Statement/PreparedStatement` and `Connection`. Failing to clean up may exhaust the CPU and memory resources, which you would want to guard against when you work on the team project. We leave the research of best practices to you.

## Object-Relational Mapping (ORM)

Your team leader at Carnegie SoShall believes that the business model of a successful global social network will inevitably evolve over time and there will always be reasons to update the existing database schemas to meet the new requirements. In order to build a system that is highly evolvable, the team leader has two conditions: - The system must be designed with high flexibility and maintainability so that updating the schemas will not slow down the development process. - Avoid database vendor lock-in. The system should be easily portable from one database management system to another, e.g. from a local PostgreSQL server to a remote MySQL server on Azure.

## Problems with JDBC

JDBC enables Java programs to interact with any SQL-compliant database and offers the flexibility to reduce the performance overhead for your application. However, another overhead that also matters in the real world is the programming overhead for developers.

As an application written with JDBC becomes more and more complicated over time, it will result in a large amount of boilerplate SQL code. As a result, building complex SQL queries by putting boilerplate components together can be time-consuming and error-prone for developers.

Additionally, unless you have a deep understanding of SQL standards and dialects such that you can write SQL queries in a SQL-compliant approach, you may easily introduce some SQL code written in a SQL dialect that is only supported in some RDBMSs which leads to database vendor lock-in.

Moreover, as the business logic evolves over time, it may be necessary to change and update the existing DB schemas. With an application written with JDBC, you may find that a change of the schemas can cause a domino effect on the previously written code. Each time you update the schemas, you have to review, update and test the affected code all over your application.

For example, suppose you are following the object-oriented programming (OOP) approach to implement your Java application and there is a domain class (https://en.wikipedia.org/wiki/Domain_model) named `Course` :

```
public class Course {
    String courseId;
    String name;
}
```

To persist the Course-related data, you use the following MySQL table,

```
CREATE TABLE `course` (
    `course_id` varchar(22) not null,
    `name` varchar(140) not null,
    primary key (course_id)
);
```

If you use JDBC APIs and SQL queries to retrieve and save the data, your code may look something like:

```
Course course = new Course("s22-15319", "Cloud Computing Course");

insertCoursePstmt = conn.prepareStatement("INSERT INTO `course` (`courseId`, `name`) VALUES (?, ?);");
// map the fields of the `Course` class to the columns of the `course` table
insertCoursePstmt.setString(1, course.getCourseId());
insertCoursePstmt.setString(2, course.getName());
// map a `Course` object to one row of the `course` table
insertCoursePstmt.executeUpdate();
```

With the code above, you are manually creating a mapping between the `Course` class and the `course` table together. Each field of the `Course` class is mapped to a column in the `course` table, and a `Course` object is mapped to a single unique row in the `course` table.

Now, let's say each course can have multiple projects (one-to-many association). We define a new domain class `Project` and a new SQL table for projects. The `project` schema can be defined as:

```
CREATE TABLE `project` (
    `project_id` varchar(22) not null,
    `name` varchar(140) not null,
    `course_id` varchar(22) not null,
    primary key (project_id)
    foreign key (course_id) references course(course_id)
);
```

The entity-relation diagram will be as follows:

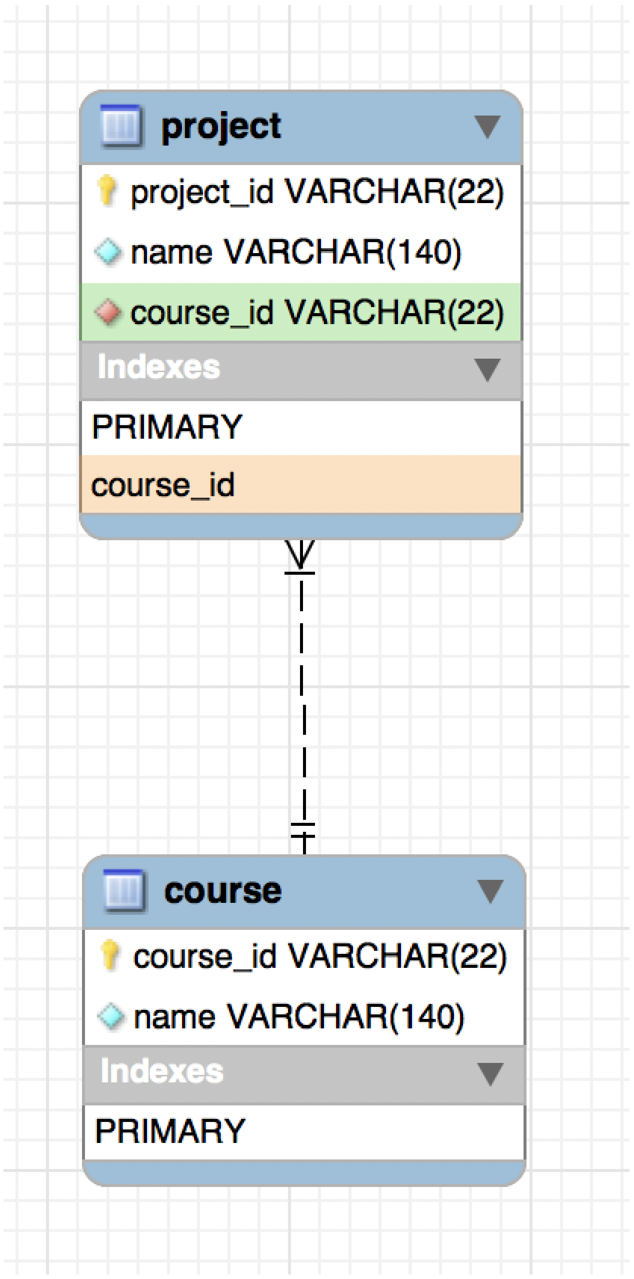

**Figure 1:** Course and Project Domain class Entity-Relation Diagram

**Problem: How do you represent foreign keys in the domain classes?**

```
public class Course {
    String courseId;
    String name;
}

public class Project {
    String projectId;
    String name;
}
```

Object-oriented languages represent associations using object references whereas an RDBMS represents associations as foreign keys. To reflect this relationship in the domain classes, you have to refer to `Project` in the definition of `Course` class.

```
public class Course {
    String courseId;
    String name;
}

public class Project {
    String projectId;
    String name;
    Course course;
}
```

Now, suppose there is 1 course consisting of 2 projects, imagine the SQL queries you have to write to persist them into the databases.

```
Course course = new Course("s22-15319", "Cloud Computing Course");
Project project1 = new Project("1", "Sequential Programming");
project1.setCourse(course);
Project project2 = new Project("2", "Parallel Programming");
project2.setCourse(course);

// plain-SQL solutions
insertCoursePstmt = conn.prepareStatement("INSERT INTO `course` (`courseId`, `name`) VALUES (?, ?);");
insertCoursePstmt.setString(1, course.getCourseId());
insertCoursePstmt.setString(2, course.getName());
insertCoursePstmt.executeUpdate();

PreparedStatement insertProjectPstmt = conn.prepareStatement("INSERT INTO `project` (`project_id`, `name`, `co
urse_id`) VALUES (?, ?, ?);");
for (Project project : new Project[]{project1, project2}) {
    insertProjectPstmt.setString(1, project.getProjectId());
    insertProjectPstmt.setString(2, project.getName());
    // Association mismatch:
    // Object-oriented languages represent associations using object references
    // where as an RDBMS represents an association as a foreign key.
    Course foreignKeyCourse = project.getCourse();
    insertProjectPstmt.setString(3, foreignKeyCourse.getCourseId());
    insertProjectPstmt.executeUpdate();
}
```

To map the objects to the relations is a common implementation challenge, not only because of the tedious work that may be required but also because object models and relational models fundamentally follow different paradigms and do not work very well together.

The conceptual and technical difficulties to map objects or class definitions to database tables defined by a relational schema is called "object-relational impedance mismatch", and the association mismatch above is one such example. There are many attempts at solving this problem.

**Solution:**

Object-Relational Mapping (ORM) is one of the solutions. Although ORM is not an agreed-upon standard and there are many debates over ORM, it is highly recommended with agile development.

Object-Relational Mapping (ORM) is a widely adopted technique to remedy the challenges when the application is written in an object-oriented fashion and the data is persisted in an RDBMS. ORM automatically converts data between object-oriented programming and relational databases by associating user-defined domain classes with database tables, and instances of the classes (objects) with rows in the corresponding tables.

Here is a sample snippet with ORM:

```
Course course = new Course("s22-15619", "Cloud Computing Course (Graduate)");
Project project1 = new Project("phase1", "Twitter Analytics Web Service Phase 1");
project1.setCourse(course);
Project project2 = new Project("phase2", "Twitter Analytics Web Service Phase 2");
project2.setCourse(course);

// ORM can abstract plain-SQL queries away from your application
SessionFactory sessionFactory = buildSessionFactory();
Session session = sessionFactory.getCurrentSession();
session.beginTransaction();
session.save(course);
for (Project project : new Project[]{project1, project2}) {
    session.save(project);
}
session.getTransaction().commit();
```

You can run this sample by executing the following commands:

```
# Create sample_db and initialize tables
mysql -pdbroot --local-infile=1 < hibernate_sample_db.sql
mvn package -Dmaven.test.skip=true

# Plain-SQL solution
java -cp target/database_tasks.jar edu.cmu.cs.sample.Main plainSQL

# ORM solution
java -cp target/database_tasks.jar edu.cmu.cs.sample.Main ORM
```

There are two different solutions for inserting data into the database in this sample, - plain-SQL and - ORM.

After executing the above code, you can check the data in MySQL. Note that you will get errors if you try to run the sample again and insert the same data because of the primary key constraints.

### Benefits of ORM

1. Separation of concerns. ORM decouples the persistence and the business logic code so that developers don't get distracted by writing all the CRUD query operations from scratch.
2. Productivity. You can focus on a single programming language rather than switching between the OOP paradigm and the declarative SQL paradigm.
3. Flexibility to meet evolving business requirements. When you need to update the schemas, the major modification required will be in the scope of the definition of the domain classes. There will be fewer SQL statements that need to be updated. Although ORM cannot eliminate the schema update problem, it may ease the difficulty to update the schema, especially when used together with data migration tools.
4. Vendor independence. ORM can abstract the application away from the underlying SQL database and SQL dialect.

Two of the most popular ORM tools are Hibernate in Java and Django ORM in Python.

## Load Data

Before starting off with any of the MySQL tasks, let us first set up the database with the data. MySQL is installed and running on the VM image supplied for this project.

### Connecting to the MySQL database

Information

To start a MySQL CLI client and connect to the running MySQL server on your instance, use the following command:

```
mysql -u clouduser --password=dbroot
```

If the username of the MySQL database is the same as the username on your Linux machine, the command can be simplified to:

```
mysql -pdbroot
```

In the commands above, the username is `clouduser` and the password is `dbroot`.

Using a plain-text password on the CLI can be insecure, and you will get a warning from MySQL.

```
[Warning] Using a password on the command line interface can be insecure.
```

If you are working on a less secure machine, you must remove the plain password from your command and MySQL will ask you for the password and you can type the password safely.

```
mysql -p
```

Before loading the data into a database, you need to describe each table in a schema. A schema encapsulates the structure and relationship of the data. We will refer to the column headers of the TSV files to design the schemas. Please read the SQL script `create_yelp_database.sql` to understand the schemas before you move on.

You can load the data files into the database with the following command:

```
# make sure you run the below command from the same location as your .tsv files
# remember to modify the path to "create_yelp_database.sql"
# please note that this takes a few minutes
# --local-infile=1 enables "LOAD DATA LOCAL INFILE", otherwise the server will reject
# the infile loading due to security concerns

mysql -pdbroot --local-infile=1 < path/to/create_yelp_database.sql
```

The data loading may take around 40 - 60 minutes. In rare cases, the loading process could take more than 60 minutes. You can continue to read on while MySQL completes loading the data.

After the command has been completed successfully and the tables have been created, you can inspect and verify the schemas with the following commands:

```
# set the working database
USE yelp_db;
# list the tables
SHOW TABLES;
# list all the indexes (including primary keys and foreign keys) in the database
# `table_schema` is the name of the database
SELECT * FROM INFORMATION_SCHEMA.STATISTICS WHERE table_schema = 'yelp_db';
# show the schema of a table
DESCRIBE businesses;
```

To verify whether the data has been successfully loaded into the database, you can list the first 10 records of the table using the SELECT command:

```
SELECT * FROM businesses LIMIT 10;
```

To make the output clearer with fewer columns, you can specify the column names to select.

```
SELECT name, city, state, stars FROM businesses LIMIT 10;
```

MySQL allows you to make queries from the database using keywords such as SELECT, WHERE and LIKE. For example, the equivalent SQL command to find the records that contain "Pittsburgh" is:

```
SELECT COUNT(*) FROM businesses WHERE city LIKE '%Pittsburgh%';
```

You need to investigate the usage of `%` before and after "Pittsburgh", and figure out whether the query above is case-sensitive or not.

## MySQL Task

For this task, you need to add in code for

- Question 1 in `runner.sh` and
- Questions 2-7 in `/home/clouduser/Project_Database/src/main/java/edu/cmu/cs/cloud/MySQLTasks.java`

In `MySQLTasks.java`, the `DB_USER` is the same as your username on your machine, `clouduser`, which can be retrieved by `System.getProperty("user.name")` in Java.

Before you start writing your own queries, you can run a demo of how Java interacts with MySQL by calling:

```
mvn clean package -Dmaven.test.skip=true
java -cp target/database_tasks.jar edu.cmu.cs.cloud.MySQLTasks demo
```

This should print the current total number of rows in the `businesses` table if it exists.

Note:

1. `mvn package` implicitly runs `mvn test`. As there are test cases in the next tasks that we will not finish in this task, we want to skip the tests for now.
2. You can use `-Dmaven.test.skip=true` to skip the tests when running `mvn package`.
3. However, please use this command with caution and do not abuse this command to fully beat the purpose of Test-Driven Development in the next tasks. There is a graded Test-Driven Development task later.

After completing each question in this section, you can execute your queries and check your results by either of the following commands after compiling:

```
./runner.sh -r question_id
java -cp target/database_tasks.jar edu.cmu.cs.cloud.MySQLTasks question_id
```

### Notes and Hints (Read these carefully)

1. Figure out whether the SQL `LIKE` operator is case sensitive or not by default.
2. Remember to output all your answers in one line per result.
3. **You are not allowed to expose MySQL credentials** in any of the questions.
4. Using `LIMIT` operator to get the maximum value will be regarded as a hardcode.
5. You should write **a single SQL query** that returns the final answer. It is **NOT** allowed to do **any further processing (e.g. filtering, indexing) with Java code** after executing the SQL query. You will not earn full points during manual grading if you violate this rule.
    - For example, if you are asked to return only one record, that's exactly what you should do. The following is NOT allowed: use SQL to return a batch of records, invoke `resultSet.next()` n times, and then use the `n+1` record, here you are actually doing the filtering with Java. We want you to learn how to return records directly from databases without further processing.
    - We expect a single query for each question, not more. You can use nested queries instead of consecutive queries. We expect you to learn how to write complex queries in this project instead of splitting a question into pieces when it can be solved in one query.
6. Since the next task builds upon this task, you do not have to terminate the instance.

## ORM Scenario

After the completion of the MySQL task, Carnegie SoShall wants you to finish a prototype using ORM to demonstrate its portability to different databases and the flexibility to meet evolving business needs. Carnegie SoShall shares a Yelp startup story with you and asks you to wear the hat of a Yelp developer to solve the challenges.

In the old days when Yelp was a small startup in 2005, the most valuable business data was stored as the `businesses` table in a SQLite3 database `yelp_db.sqlite3`. In this legacy database, there are only 11 columns in the `businesses` table with the basic business information. You can view the legacy schema with the following commands:

```
$ sqlite3 yelp_db.sqlite3
SQLite version 3.11.0 2016-02-15 17:29:24
Enter ".help" for usage hints.

sqlite> .schema businesses

CREATE TABLE `businesses` (
    `business_id` varchar(22) not null,
    `name` varchar(140) not null,
    `state` varchar(2) default null,
    `city` varchar(140) default null,
    `address` varchar(140) not null,
    `postal_code` varchar(10) not null,
    `hours` LONGTEXT default null,
    `review_count` int(10) default 0 not null,
    `stars` float(2,1) default 0.0 not null,
    `longitude` varchar(12) not null,
    `latitude` varchar(12) not null,
     primary key (business_id)
);
```

The Yelp dev team implemented a complex application with Hibernate, and the business logic code is simplified in `YelpApp.java`.

The simplified `YelpApp` application implements an API which queries the business with the most reviews in Pittsburgh from the database as a `Business` object, serializes the object into JSON and prints it.

```
{"address":"649 Penn Ave","business_id":"JLbgvGM4FXh9zNP4O5ZWjQ","city":"Pittsburgh","hours":"['Monday 17:0-2
3:0', 'Tuesday 17:0-23:0', 'Wednesday 11:30-14:0', 'Thursday 11:30-14:0', 'Friday 11:30-14:0', 'Friday 17:0-0:
0', 'Saturday 10:30-14:0', 'Sunday 17:0-22:0']","latitude":"40.4431445863","longitude":"-80.0011037908","nam
e":"Meat & Potatoes","postal_code":"15222","review_count":1249,"stars":4.0,"state":"PA"}
```

After great success and several rounds of funding,

1. The development team of Yelp decided to add more business information to the `business` table (more than once), which eventually results in the latest schema. The following columns were not included in the legacy schema:

    ```
    `neighborhood` varchar(140) default null,
    `open` tinyint(1) not null,
    `attributes` LONGTEXT default null,
    `categories` LONGTEXT default null
    ```

2. The dev team also decided to switch to a MySQL database. The new `businesses` table is created with the following schema:

```
create table `businesses` (
    `neighborhood` varchar(140) default null,
    `business_id` varchar(22) not null,
    `hours` LONGTEXT default null,
    `open` tinyint(1) not null,
    `address` varchar(140) not null,
    `attributes` LONGTEXT default null,
    `categories` LONGTEXT default null,
    `city` varchar(140) default null,
    `review_count` int(10) default 0 not null,
    `name` varchar(140) not null,
    `longitude` varchar(12) not null,
    `state` varchar(2) default null,
    `stars` float(2,1) default 0.0 not null,
    `latitude` varchar(12) not null,
    `postal_code` varchar(10) not null,
    primary key (business_id)
);
```

*The updated data was already loaded to the new `businesses` table (actually by you in the previous steps in the MySQL task).*

The current configuration in `YelpApp.java` makes use of the SQLite3 legacy database. In order to use `Hibernate`, two sets of configurations have been defined:

1. **The database connection configuration:** The configuration is set at `src/main/resources/hibernate.properties`, such as the SQL dialect, the driver class and the connection URL for the SQLite3 database.
2. **The mapping between domain classes and database tables**: In this simplified scenario, there is only one mapping between the `Business` class (`Business.java`) and the `businesses` table. It is set through `YelpApp.java` using `configuration.addAnnotatedClass(Business.class)`. The `@Table(name = "businesses")` annotation in the `Business` class specified the mapped table as `businesses`. There are 11 fields in the `Business` class that will match the number of the columns in the table.

## ORM Task

In order to update the configuration to accommodate the new changes made by the dev team, your task is

1. To plug in the MySQL database, with database url as `jdbc:mysql://localhost/yelp_db?useSSL=false`, to the `YelpApp` app and
2. Update the definition of the `Business` class to meet the schema in the `businesses` table in MySQL.

You need to update the following two files only:

- `src/main/resources/hibernate.properties`
- `src/main/java/edu/cmu/cs/cloud/Business.java`

To address the complexity and difficulty to update existing business logic code upon any schema update, you are **NOT ALLOWED** to change `YelpApp.java`.

For more context, explanation, and hints, you can refer to the comments in

- `src/main/resources/hibernate.properties`,
- `src/main/java/edu/cmu/cs/cloud/Business.java` and
- `src/main/java/edu/cmu/cs/cloud/YelpApp.java`

After completing this task, you can check that `YelpApp` will return the data following the **new** schema:

```
mvn clean package -Dmaven.test.skip=true && java -cp target/database_tasks.jar edu.cmu.cs.cloud.YelpApp

 {"address":"649 Penn Ave","business_id":"JLbgvGM4FXh9zNP4O5ZWjQ","city":"Pittsburgh","hours":"{'Monday': '17:
00-23:00', 'Tuesday': '17:00-23:00', 'Friday': '17:00-0:00', 'Wednesday': '11:30-14:00', 'Thursday': '11:30-1
4:00', 'Sunday': '17:00-22:00', 'Saturday': '10:30-14:00'}","latitude":"40.44","longitude":"-80.00","name":"Me
at & Potatoes","postal_code":"15222","review_count":1476,"stars":4.0,"state":"PA","attributes":"{'GoodForMea
l': {'dessert': False, 'latenight': False, 'lunch': False, 'dinner': True, 'brunch': False, 'breakfast': Fals
e}, 'Alcohol': 'full_bar', 'Caters': False, 'HasTV': False, 'GoodForKids': False, 'NoiseLevel': 'average', 'Wi
Fi': 'free', 'RestaurantsAttire': 'casual', 'RestaurantsReservations': True, 'OutdoorSeating': True, 'Business
AcceptsCreditCards': True, 'RestaurantsPriceRange2': 3, 'BikeParking': True, 'RestaurantsTableService': True,
'RestaurantsDelivery': False, 'Ambience': {'romantic': False, 'intimate': False, 'classy': False, 'hipster': F
alse, 'divey': False, 'touristy': False, 'trendy': True, 'upscale': False, 'casual': False}, 'RestaurantsTakeO
ut': True, 'RestaurantsGoodForGroups': True, 'WheelchairAccessible': True, 'BusinessParking': {'garage': True,
'street': True, 'validated': False, 'lot': False, 'valet': False}}","categories":"['Specialty Food', 'Steakhou
ses', 'Meat Shops', 'American (New)', 'Gastropubs', 'Restaurants', 'Food']","neighborhood":"Downtown","open":t
rue}
```

As you can notice, despite the update of the schema, the sample API keeps backward compatibility without any modification with the help of ORM. The new version of the JSON result is a superset of the previous version, and you can assume the clients of the API will not be affected. Although such zero-touch maintenance is rare in real life due to the complexity of real-world applications, this simplified example demonstrates the productivity ORM can offer when used properly.

### What to Submit

1. It is good practice to always include comments to **explain complicated queries** and **adopt a readable style** as you are writing the code. This is reflected in your code style and comments. We will manually grade your last submission of each task.
2. Make sure the following files are under the `~/Project_Database/` directory:
    - references file
    - runner.sh
    - submitter
3. Make sure the following files are under the `~/Project_Database/src/main/java/edu/cmu/cs/cloud/` directory:
    - Business.java
    - MySQLTasks.java
    - YelpApp.java
    - HBaseTasks.java

### How to Submit

Note: You should only execute the submitter from the student VM and not from your local machine. Executing the submitter with your own machine is subject to an unbounded debugging scope of the OS environment and such debugging is not in the scope of the project learning. The teaching staff will not be able to support you if you attempt to run the code locally. However, you may run your java/python code locally for testing purposes without submission.

You can submit your answers and code by running the `submitter`

Please follow these steps to submit your project:

```
export HISTIGNORE=export* # so that the following export commands will not be tracked into bash history
export SUBMISSION_USERNAME=your_submission_username
export SUBMISSION_PASSWORD=your_submission_pwd
./submitter -t mysql
```

Task 2: Introduction to NoSQL (HBase)

# HBase

**Before you move on, we require you to complete the NoSQL primer and HBase Basics primer.**

## Scenario

Now that you have explored relational databases, Carnegie SoShall wants you to explore NoSQL database solutions for its applications. The NoSQL database you will explore in this task is a distributed database called HBase. For your task, you will first define a table schema with pre-split regions, then design a rowkey pattern that evenly loads data into the pre-split regions, and finally perform a set of queries that retrieve data.

## Region Splits

As the primer stated, an HBase region (HRegion) is defined as a contiguous subset of rows when the HBase table grows beyond a preset threshold. Each HBase region is served by a region server (HRegionServer). Regions help HBase distribute loads away from one centralized server.

However, HBase doesn't have prior knowledge of the rowkey patterns of your data and, therefore, doesn't know how to split the region accordingly. Therefore, by default, HBase creates only one region for a table. Decisions on region splitting are based highly on the distribution of the keys in your data. Rather than guessing and leaving it up to the user to deal with the consequences, HBase provides you with the tools to manage the split points from the client.

With a process called **pre-splitting**, you can create a table with many regions by supplying the split points at the time of table creation. Because pre-splitting will ensure that the initial load is more evenly distributed throughout the cluster, you should always consider using it if you know your key distribution beforehand.

You can read Apache HBase Region Splitting and Merging (https://blog.cloudera.com/apache-hbase-region-splitting-and-merging/) to learn more about pre-splitting, which is a quick explanation of the HBase Region split policy.

From the `hbase shell` , there are 2 ways to create a table in HBase with pre-splitted regions:

1. **Defining the startKey and endKey by yourself**. The following command can create a table and split the table into 4 regions:
    - Region 1: no startKey, endKey is 10
    - Region 2: startKey is 10, endKey is 20
    - Region 3: startKey is 20, endKey is 30
    - Region 4: startKey is 30, no endKey

    ```
    > create 'table', 'cf', SPLITS=> ['10', '20', '30']
    ```

2. **Using predefined split algorithms** such as `HexStringSplit` and `UniformSplit`. You can refer to the Java implementation (https://github.com/apache/hbase/blob/master/hbase-server/src/main/java/org/apache/hadoop/hbase/util/RegionSplitter.java) for an example. The following command shows how to split the table into 5 regions by using `HexStringSplit`.

    ```
    > create 'table', 'cf', { NUMREGIONS => 5, SPLITALGO => 'HexStringSplit' }
    ```

## RowKey Design

Once the table schema is defined and the table is pre-split, you will need to design a rowkey pattern that evenly loads data into the pre-splitted regions. An analogy here is that pre-splitting a table is like a Java interface, which sets the paradigm for region splitting, while your rowkey design is like the implementation class, which actually implements the paradigm by evenly distributing rowkeys into different regions.

By default, rows in HBase are sorted lexicographically by row key. This design is optimized only for SCAN operations, allowing you to store and retrieve rows that are related to each other or placed next to each other. That being said, poorly designed row keys are still a common source of hotspotting.

### Hotspotting

Given the distributed nature of HBase regions, **hotspotting** occurs when a large amount of client traffic is directed to one node, or only a few nodes, of a cluster. This traffic may represent reads, writes, or other operations.

Hotspotting can overwhelm a single machine responsible for hosting a region,

1. Causing performance degradation and potentially leading to region unavailability.
2. Have adverse effects on other regions hosted within the same region server as the host suffering from hotspotting is unable to service the requested load.

Therefore, it is important to design data access patterns such that the cluster is fully and evenly utilized.

To prevent hotspotting, it is crucial to design row keys such that the records are evenly distributed across regions. Next, we will introduce two techniques for avoiding hotspotting by redesigning row keys, namely salting and hashing, along with their advantages and disadvantages.

**Technique 1: Salting**

Salting in this case has nothing to do with cryptography. Salting, in this case, refers to adding a randomly assigned prefix to the row key to cause it to sort differently than it otherwise would. The number of possible prefixes to be used should equal the number of regions you want to spread the data across. Salting can be helpful if you have few "hot" row key patterns which come up over and over among other more evenly distributed rows.

For example, suppose you have the following list of row keys, and your table is split such that a single region corresponds to a single letter of the alphabet. Prefix 'a' is in one region, prefix 'b' is in another, and so on. In this table, all rows starting with 'f' are in the same region. This example focuses on rows with keys as follows:

```
foo0001
foo0002
foo0003
foo0004
```

Now, imagine that you would like to spread these across four different regions. You decide to use four different salts: a, b, c, and d. In this scenario, each of these letter prefixes will be in a different region. After applying the salts, you have the following row keys. Since you can now write to four separate regions, you theoretically have a writing throughput that is four times higher than the throughput of writing to the same region.

```
a-foo0003
b-foo0001
c-foo0004
d-foo0002
```

**Technique 2: Hashing**

Instead of a random assignment, you could use a one-way hash to randomize rowkey. Using a deterministic hash allows the client to reconstruct the complete rowkey and use a `GET` operation to retrieve the row as normal. It is due to hashing that makes `GET` operation the most efficient way to query HBase. You definitely don't want to `SCAN` the entire table to find a row by its key.

Suppose you have the same sets of row keys, and your table is split by hex string into four regions. Given a hash function `hash`, the data can be written to four separate regions. You can expect to have evenly distributed storage with a large enough dataset, promised by the functionality of the hash algorithm.

```
hash(foo0001) = 0x1bfe -> region 1
hash(foo0002) = 0x420a -> region 2
hash(foo0003) = 0xab53 -> region 3
hash(foo0004) = 0xff98 -> region 4
```

## Provision an HBase cluster using Azure Resource Manager (ARM)

### Information

Before you getting started, make sure your subscription has enough HDI quota limits.

```
az hdinsight list-usage --location eastus
```

The Core limit is expected to be 40. However, if the limit is less, a value more than 24 will suffice as well. If your quota is 0, you should register Microsoft.HDInsight as a resource provider.

```
az provider register --namespace Microsoft.HDInsight
```

You can track this process by executing the following command.

```
az provider show --namespace Microsoft.HDInsight --output table
```

Once it is successful, you can expect to have 40 quota limits within 5 minutes.

Download the ARM template to provision an HBase cluster within the same virtual network as the submitter VM instance.

To find the resource ID of a resource on Azure, you may go to the Azure portal and view the properties tab of the resource. Note that an Azure Resource ID is represented with namespaces, e.g., the resource ID of a virtual network will be `/subscriptions/<subscriptionId>/resourceGroups/<resourceGroupName>/providers/Microsoft.Network/virtualNetworks/<virtualNetworkName>`.

```
mkdir azure-hbase && cd azure-hbase
wget https://clouddeveloper.blob.core.windows.net/assets/cloud-storage/project/azure-hbase.tgz
tar -xvzf azure-hbase.tgz

# in parameters.json, set the values of the following parameters:
# `virtualNetworkId`
# note that the value should be the full resource ID of the virtual network
# you can find the resource ID in the properties tab of the virtual network "cloud-computing-vnet"

RESOURCE_GROUP_NAME="cloud-storage-rg"
az deployment group create --name "cloud-course-hdinsight-cluster" --resource-group "$RESOURCE_GROUP_NAME" --t
emplate-file "template.json" --parameters "parameters.json"

# You will be prompted with the following where you should type your password.
# The password must be at least 10 characters in length and must contain
# at least one digit, one non-alphanumeric character, and one upper or lower case letter.
Please provide securestring value for 'clusterLoginPassword' (? for help): <type_your_password>
Please provide securestring value for 'sshPassword' (? for help): <type_your_password>
```

**Note: It might take around 20 minutes to provision an HBase cluster, so please be patient!**

### Information

If you accidentally terminate the close/disrupt the command line after you created the deployment, you can use the following command to check its status. `az deployment group show --resource-group "$RESOURCE_GROUP_NAME" --name "cloud-course-hdinsight-cluster"`

### Information

Cluster provision approximately takes **20 minutes**. You may want to keep reading the write-up at the same time. When the provisioning finishes, you will see a message in the `JSON` format that includes the result `"provisioningState": "Succeeded"`.

## Load Data

Information

If you get a **DNS SPOOFING** warning, follow the steps below to solve this issue

```
- Edit the .ssh/known_hosts file
- Remove the entry that corresponds to the HDInsight cluster
- Try to SSH again
```

1. SSH into the HBase cluster and create an HBase table via the HBase shell. You can find the SSH user and DNS in Azure portal `(Resource group -> cloud-storage-rg -> HDInsight cluster -> Properties)`. Then, you need to design the rowkey by yourself and execute your command to create the table with **10 pre-split regions**.

```
ssh sshuser@CLUSTER_ID-ssh.azurehdinsight.net
hbase shell

# (in the HBase shell)
# Please complete the following command to split the table into 10 regions
> create 'business', 'data'

# you can type `exit` to quit the HBase shell
```

2. Download the dataset to the HBase cluster and transfer the dataset to HDFS.

```
# Note that the first line of `yelp_academic_dataset_business.tsv`, is not a record but the header row.
# You MUST get rid of the header line first so that you will not import the header line into HBase.
# We recommend that you use `tail` to skip the header row

wget https://clouddeveloper.blob.core.windows.net/datasets/cloud-storage/yelp_academic_dataset_businesse
s.tsv

tail -n +2 yelp_academic_dataset_businesses.tsv > yelp_academic_dataset_businesses_no_header.tsv

# Create a new directory in HDFS
hadoop fs -mkdir /input

# Put the dataset to HDFS
hadoop fs -put yelp_academic_dataset_businesses_no_header.tsv /input

# To check if the file is successfully stored to HDFS, use the following command (or you may specify you
r own file path):
hadoop fs -ls /input/yelp_academic_dataset_businesses_no_header.tsv
```

3. `YetAnotherImportTsv` is provided in this project which uses MapReduce jobs to load data into an HBase table. Note that the `YetAnotherImportTsv` program provided in this project has some minor differences compared to the one in the primer, such as the data schema. **Download YetAnotherImportTsv to the HBase cluster.**

```
wget https://clouddeveloper.blob.core.windows.net/s22-cloud-developer/cloud-storage/YetAnotherImportTsv.
tgz
tar -xvzf YetAnotherImportTsv.tgz
```

4. **Read the source code and update the configuration.** The `hbase.zookeeper.quorum` property is a comma-separated list of hosts on which ZooKeeper servers are running. For an HDInsight cluster, go to the HDInsight dashboard and click the "Properties" tab. Click the virtual network under the "Virtual Network" heading to go to the virtual network page. The virtual network page lists the devices connected together with their private IP addresses, and the ZooKeeper nodes have the names with the keyword `zookeepernode`. As there are 3 ZooKeeper nodes on HDInsight, the value of `hbase.zookeeper.quorum` will be `private_ip_1,private_ip_2,private_ip_3`.

5. **You will need to evenly distribute the rowkey to the 10 regions that you pre-split.** Please note that, evenly distributing the data, in this case, does not mean that each region will have the same number of records. It only means that the data is spread across regions uniformly. Design the rowkey and update the configuration in `YetAnotherImportTsv`:

```
// TODO: Design a new rowkey to evenly load data into 10 regions.
String rowKey = columns[ROW_KEY_INDEX];

// TODO: update the Zookeeper address(es)
String zkAddr = "";
conf.set("hbase.zookeeper.quorum", zkAddr);
conf.set("hbase.zookeeper.property.clientport", "2181");
conf.set("hbase.cluster.distributed", "true");
// Linux-based HDInsight clusters use /hbase-unsecure as the znode parent
conf.set("zookeeper.znode.parent","/hbase-unsecure");
```

6. After loading data into the database, you can access the HBase Master UI to get the information of all tables. Refer to this tutorial (https://docs.microsoft.com/en-us/azure/hdinsight/hbase/apache-hbase-tutorial-get-started-linux) to learn how to access the HBase Master UI. Note that your username is **admin**, and your password is the **clusterLoginPassword** that you had set. As you can see in the following screenshot, you can get the table region information from the HBase Master UI.

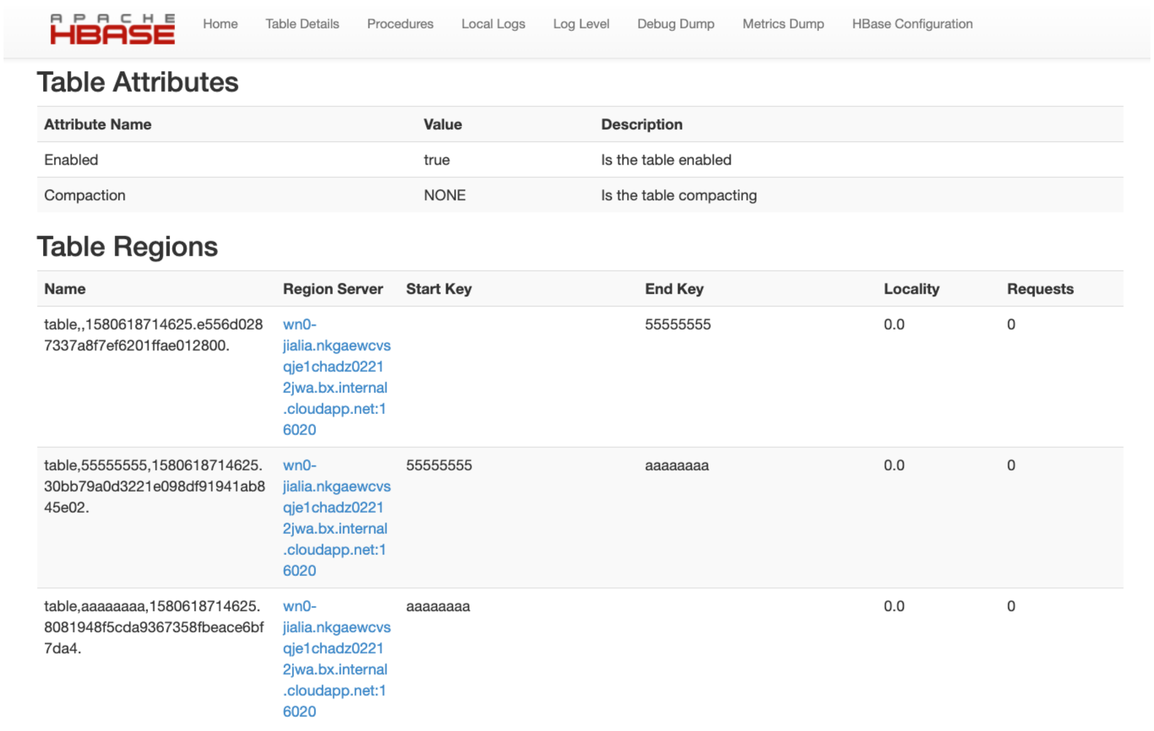

Table Attributes

| Attribute Name | Value | Description |
|---|---|---|
| Enabled | true | Is the table enabled |
| Compaction | NONE | Is the table compacting |

Table Regions

| Name | Region Server | Start Key | End Key | Locality | Requests |
|---|---|---|---|---|---|
| table,,1580618714625.e556d0287337a8f7ef6201ffae012800. | wn0-jialia.nkgaewcvsqje1chadz02212jwa.bx.internal.cloudapp.net:16020 | | 55555555 | 0.0 | 0 |
| table,55555555,1580618714625.30bb79a0d3221e098df91941ab845e02. | wn0-jialia.nkgaewcvsqje1chadz02212jwa.bx.internal.cloudapp.net:16020 | 55555555 | aaaaaaaa | 0.0 | 0 |
| table,aaaaaaaa,1580618714625.8081948f5cda9367358fbeace6bf7da4. | wn0-jialia.nkgaewcvsqje1chadz02212jwa.bx.internal.cloudapp.net:16020 | aaaaaaaa | | 0.0 | 0 |

**Figure 2:** HBase Master UI

1. After designing the row key and loading data to HBase, you will need to implement the `rowKeyDesignTask()` method in `HBaseTasks.java`, which prints out the number of rows in each region in a comma separated format. Comments in the starter code will give you more detailed instructions.

## Running the program

You can use the following commands to install Maven on the HDInsight cluster, package the JAR and start the MapReduce job to load the business data into the HBase cluster.

```
sudo apt update
sudo apt-get install -y maven
mvn clean package
yarn jar target/import_tsv.jar edu.cmu.cc.utils.YetAnotherImportTsv
```

**To connect to the HBase cluster from the submitter instance successfully, both the submitter instance and the HBase cluster should be in the same virtual network (as you set in the `parameters.json` of the ARM template when you provision the HBase cluster).**

Please note that after loading data into the database, you will need to complete the questions in `HBaseTasks.java`. Update the configuration in `HBaseTasks.java` first (similar to how you update the configuration for `YetAnotherImportTsv`).

You can run a demo by typing:

```
mvn clean package
java -cp target/database_tasks.jar edu.cmu.cs.cloud.HBaseTasks demo
```

This should print the number of businesses (6361) that are located in Pittsburgh (case sensitive).

If you need to re-design your rowkey, and want to reload the business data to the HBase cluster, you should drop the table before reloading the data. This can be done using the following commands:

```
disable 'business'
drop 'business'
```

After completing each question in `HBaseTasks.java` , you can execute your queries and check your output with either of the commands after compiling:

```
./runner.sh -r question_id
java -cp target/database_tasks.jar edu.cmu.cs.cloud.HBaseTasks question_id
```

After finishing every question, you can test everything together by calling

```
./runner.sh
```

Information

### Notes

You should write **a single query** that returns the final answer. It is **NOT** allowed to do **any further processing** (e.g. filtering, indexing, **math operations**) using Java code after executing the query. The only exception is for **q10()** where you are allowed to sort the result using Java code.

You should first pre-split the table using `hbase-shell` and design row keys before loading data. The `rowKeyDesignTask()` performs neither region-splitting nor rowkey design. It only allows you to visualize the results of your rowkey design so that our grader can verify that the data has been evenly split across regions.

## What to Submit

1. It is good practice to always include comments to **explain complicated queries** and **adopt a readable style** as you are writing the code. This is reflected in your code style and comments. We will manually grade your last submission of each task.
2. Create `design.pdf` to explain your rowkey design (Please include your pre-split command). Make sure the following files are under the `~/Project_Database/` directory.
3. Make sure the following files are under the `~/Project_Database/` directory:
    - references file
    - runner.sh
    - submitter
4. Make sure the following files are under the `~/Project_Database/src/main/java/edu/cmu/cs/cloud/` directory:
    - Business.java
    - MySQLTasks.java
    - YelpApp.java
    - HBaseTasks.java

## How to Submit

Note: You should only execute the submitter from the student VM and not from your local machine. Executing the submitter with your own machine is subject to an unbounded debugging scope of the OS environment and such debugging is not in the scope of the project learning. The teaching staff will not be able to support you if you attempt to run the code locally. However, you may run your Java code locally for testing purposes without submission.

You can submit your answers and code by running the `submitter`

Please follow these steps to submit your project:

```
export HISTIGNORE=export* # so that the following export commands will not be tracked into bash history
export SUBMISSION_USERNAME=your_submission_username
export SUBMISSION_PASSWORD=your_submission_pwd
./submitter -t hbase
```

Information

### Did you know?

In this project module, you explored how to apply the `SCAN` operation with column filters to obtain useful data in HBase. You might have noticed that the `SCAN` operation in HBase can be slow. This is even more apparent when the table size is larger since `SCAN` is an `O(N)` operation, where `N` is the total number of lines in the table. However, the `GET` operation is `O(logN)` on average. Hence, the speed of the `GET` operation is reasonable compared to `SCAN` even when the table size grows.

For a large HBase table (more than 100M lines), `GET` will be at least 10000 times faster than a full table `SCAN`. Since you can only specify a rowkey for `GET`, in order to optimize the throughput using `GET`, your HBase schema should be carefully designed. At the same time, you can also specify the "startrow" and "endrow" of the `SCAN` operation to largely improve the speed of `SCAN`. Refer to this link (https://blog.cloudera.com/blog/2013/04/how-scaling-really-works-in-apache-hbase/) for more information. **Note that this will be particularly useful in the team project for 15619!**

Warning

### Terminate Resource Group

You may want to take a long break before continue working on this project, or you finish Project 3 SQL and NoSQL Part early so there are plenty of time before you could work on P3 Cloud Storage - Heterogeneous Storage on the Cloud. Then you should clean up all the resources. You should run

```
az group delete --name "cloud-storage-rg"
```

to terminate all the resources. This command would delete everything under this resource group, including the HDInsight cluster. Double-check the Azure console to verify that you have no active resources.

### Terminate HDInsight Cluster

HDInsight clusters are very expensive and can deplete your budget very quickly. You should terminate the cluster once you have finished Task 2. You can delete the cluster by the following command

```
az deployment group delete --name "cloud-course-hdinsight-cluster" --resource-group ${RESOURCE_GROUP_NAM
E}
```

Project Reflection Task (Mandatory, graded)

## Project Reflection Task (Mandatory, graded)

Upon completing this project you will make one (1) post in the [Forum] P3. Cloud Storage - SQL and NoSQL (https://projects.sailplatform.org/cloud-forum/topic/publish/1031/) to reflect on your experience before the project deadline. **Before you publish the post, please double-check that the category is the currrent module name "P3 Cloud Storage SQL and NoSQL", which should be under the course title "S22 Cloud Computing"**.

Consider the following topics when creating your post, however, you should never share any code snippets in your reflection:

- Describe your approach to solving each task in this project. Explain alternative approaches that you decided not to take and why.
- Describe any interesting problems that you had overcome while completing this project.
- If you were going to do the project over again, how would you do it differently, and why?

After completing this task, confirm that your ***Reflection Score*** has been automatically updated on the scoreboard before the project deadline.

Project Survey

## Project Survey

Please leave us feedback for this project here. This will help us strengthen this project for future offerings.

Project Discussion

## Project Discussion Task (Mandatory, graded)

After this project has been completed (after the project deadline), all reflection posts by all students will become visible for review.

Your task is to reply and provide feedback to **3** posts in the [Forum] P3. Cloud Storage - SQL and NoSQL (https://projects.sailplatform.org/cloud-forum/category/1031/), within **7** days after the project deadline.

After completing this task, confirm that your ***Discussion Score*** has been automatically updated on the scoreboard within 7 days after the project deadline.