**! STOP!**

**Required Viewing**

Before starting this assignment, you must watch two videos:

1. **Week 3 Fundamentals Lecture Video – Hive Architecture and SQL**  
   Explains how Hive works, including the architecture (clients, HiveServer2, Driver, Compiler/Optimizer, Metastore) and the data model (managed vs. external tables, schema-on-read, ACID support). This gives you the background needed to understand what you are learning.  
   Link: <https://youtu.be/_trCi95_3lQ>
2. **Week 3 Assignment Walkthrough Video**  
   Shows you step by step how to complete the tasks, including each command and the expected output.

It is not enough to just run commands without checking results. You must verify that your commands execute correctly. If they do not, you will lose points.

Watching both videos is mandatory. The fundamentals video explains what you’re learning and why it matters, and the walkthrough video shows you how to complete the assignment.

## Week 3 Assignment: Hands-on Hive

# Conceptual Foundations

Before beginning the assignment, watch the instructor-led fundamentals video, which introduces and explains the key concepts for this week: <https://youtu.be/_trCi95_3lQ>

**Deliverable:** Write a 3–4 paragraph summary that demonstrates your understanding of the material presented in the video. Your writeup should explain the main ideas in your own words, highlight why these concepts are important, and connect them to the technologies you will be working with in the assignment.

### Objective: Resize your Google Cloud VM

**Instructions:**

1. Watch the following video to learn the basics of resizing a Google Cloud VM:  
   <https://youtu.be/2pJUiT_0Wv4>
2. Stop your virtual machine.
3. Resize the VM to **8 cores and 16 GB RAM**.
4. Restart the virtual machine once the resize is complete.

**Deliverable:** Submit a screenshot showing your VM successfully resized to **8 cores and 16 GB RAM**. A screenshot of the Google Cloud Console UI is acceptable.

# Objective: Hands-on Experience with Apache Hive

In this assignment, you will gain hands-on experience with **Apache Hive**, a powerful data warehousing tool that enables you to run SQL-like queries on large datasets stored in Hadoop. Hive simplifies the complexity of Hadoop’s **MapReduce** framework by providing an intuitive, SQL-based interface.

By the end of this assignment, you will:

* Understand how to create and work with Hive tables.
* Learn how to load data into Hive for querying.
* Gain experience in running SQL queries to extract insights from big datasets.

**Hive Table Types: Managed vs. External**

Hive supports two types of tables:

* **Managed Tables**: Hive controls both the metadata and the data. If you drop a managed table, the data will be deleted along with the table.
* **External Tables**: Hive only manages the metadata, while the data remains outside of Hive’s control. If you drop an external table, the data remains intact.

In this assignment, you will primarily work with **managed tables**. For external tables, the **CREATE TABLE** statement changes slightly, including the **LOCATION** clause and the keyword **EXTERNAL** to specify the data’s location outside of Hive's control.

### Objective: Familiarize with the core functionalities of Hive MANAGED Tables (Part 1 video)

#### **1. Environment Initialization**

* Navigate to the required directory and start your Docker containers:
* cd dsc650-infra/bellevue-bigdata/hadoop-hive-spark-hbase
* docker-compose up -d
* Access the master container:
* docker-compose exec master bash
* Load the grades.csv into HDFS:
* hdfs dfs -put /data/grades.csv /

#### **2. Hive Managed Table Creation & Data Loading**

In this section, you will create a managed table in Hive, which means Hive will manage both the metadata and the actual data storage. You’ll define the structure of the table, load a dataset into it, and then query the data. This process gives you hands-on experience with how Hive stores and manages data, using SQL-like queries for analysis. By the end of this section, you will understand how to create tables, load data from HDFS into Hive, and run basic queries to retrieve and analyze the data.

* Start a Hive session:
* hive
* In the Hive CLI, create a table:

|  |
| --- |
| CREATE TABLE grades(  `Last name` STRING,  `First name` STRING,  `SSN` STRING,  `Test1` DOUBLE,  `Test2` INT,  `Test3` DOUBLE,  `Test4` DOUBLE,  `Final` DOUBLE,  `Grade` STRING)  ROW FORMAT DELIMITED  FIELDS TERMINATED BY ','  STORED AS TEXTFILE  tblproperties("skip.header.line.count"="1"); |

* Load data into the Hive Managed table:

|  |
| --- |
| LOAD DATA INPATH '/grades.csv' INTO TABLE grades; |

* Run a query to view the data:

|  |
| --- |
| SELECT \* FROM grades; |

**Deliverable:** Screenshot of the query results.

#### **3. Further Exploration with SQL on Hive**

You’re now encouraged to experiment with SQL queries on your data. This is important because it gives you the opportunity to explore the data, practice your SQL skills, and gain insights from the dataset.

* Run 3 different SQL commands on the grades data.

**Deliverable:** Screenshots of each command and its output.

#### **4. Working with a New Dataset**

For this part, you're applying the skills you learned to a new dataset. This gives you experience in working with different types of data and allows you to explore real-world datasets to extract insights.

* Choose a CSV dataset of your choice from an open data set. Some data sources to consider:
  1. [Kaggle Datasets](https://www.kaggle.com/datasets)
  2. [UCI Machine Learning Repository](https://archive.ics.uci.edu/ml/index.php)
  3. [Government Datasets](https://www.data.gov/)
  4. [AWS Public Datasets](https://registry.opendata.aws/)

**Download and Load Your Dataset into Docker and HDFS**

In this step, you will download a CSV file and load it into your Docker container, ensuring the file is in the correct format before moving it to HDFS. Follow these steps:

1. Download the file onto your local computer.
2. Rename your file so that it does not include any white spaces.
3. Proceed to GitHub and create a new public repository.
4. Upload your CSV file to the repository.
5. Click on the CSV file in GitHub, then click **RAW**.
6. Copy the URL.

**Important**: If you do not click **RAW**, you will download the webpage’s HTML instead of the CSV file. This will result in an incorrect data load, so ensure the URL points to the correct CSV file.

* Enter your Docker container's master node:

docker exec -it master bash

* Download the file using the wget command, replacing **URL** with the raw GitHub URL

wget **URL**

* Verify the file downloaded by running:

ls

* You should see the file listed. Next, check the file’s content to confirm it is a CSV:

cat **file\_name**

*replace* ***file\_name*** *with the name of your file.*

* **If** you see HTML instead of CSV data, this means you did **NOT** use the **RAW** link to download the file as instructed. Delete the file and re-download the correct one:

rm -rf **file\_name**

wget **URL**

* Once verified, load the data into HDFS:

hdfs dfs -put **file\_name** /

*Replace* ***file\_name*** *with the actual name of your CSV file.*

Return to the Hive CLI and create a table for your dataset, then load the data. Remember, you can choose to define each column as a STRING for simplicity or use the appropriate data types for a greater challenge. Visit the [Hive Language Manual](https://cwiki.apache.org/confluence/display/Hive/LanguageManual+Types) to learn about different data types available in Hive.

* Run 3 SQL commands on your new data.

**Deliverable:** Screenshots of each SQL command and its output.

* Run EXPLAIN on your 3 SQL commands.

After running your 3 SQL queries, you will now analyze how Hive executes those queries behind the scenes by using the **EXPLAIN** command. This helps you understand the query execution plan, particularly how Hive translates your SQL into a MapReduce job.

Run the **EXPLAIN** command for each of your 3 SQL queries. For example:

EXPLAIN SELECT \* FROM table;

**What EXPLAIN does**: The output of the EXPLAIN command shows how Hive will execute the SQL query. This includes the MapReduce (MR) execution plan, which details how Hive will distribute and process the data across the cluster.

**Deliverable:** Screenshots the results of the EXPLAIN command for each query. Try to explain what the MapReduce execution is doing under the hood for each query. Consider the following:

* How many stages are involved in the MapReduce job?
* Is the data being sorted or aggregated?
* What is the shuffle process doing, if applicable?

This exercise will help you better understand how Hive optimizes and executes SQL queries using Hadoop's distributed processing framework.

### Objective: Familiarize with the core functionalities of Hive EXTERNAL Tables (Part 2 Video)

Video Walkthrough Link: <https://youtu.be/wsk93urZqtU>

#### **1. Environment Initialization**

* Navigate to the required directory and start your Docker containers:
* cd dsc650-infra/bellevue-bigdata/hadoop-hive-spark-hbase
* docker-compose up -d
* Access the master container:
* docker-compose exec master bash
* Create a Folder to demonstrate Schema-on-Read

hdfs dfs -mkdir /rawdata

* Load the grades.csv into HDFS:
* hdfs dfs -put /data/grades.csv /rawdata
* Verify data is loaded to hdfs

hdfs dfs -ls /rawdata

#### **2. Hive External Table Creation & Data Loading**

In this section, you will create an external table in Hive, which means Hive will manage the metadata but the actual data storage will remain outside Hive, in its original location. You’ll define the structure of the table and point it to data that already exists in HDFS, rather than loading the data into Hive’s warehouse directory. This approach is useful when you want Hive to query and analyze data without taking ownership of it, ensuring the files remain accessible to other tools or workflows. By the end of this section, you will understand how to create external tables, link them to existing HDFS directories, and run basic queries to retrieve and analyze the data.

* Start a Hive session:

hive

* In the Hive CLI, create a table:

|  |
| --- |
| CREATE **EXTERNAL TABLE** grades\_ext(  `Last name` STRING,  `First name` STRING,  `SSN` STRING,  `Test1` DOUBLE,  `Test2` INT,  `Test3` DOUBLE,  `Test4` DOUBLE,  `Final` DOUBLE,  `Grade` STRING  )  ROW FORMAT DELIMITED  FIELDS TERMINATED BY ','  STORED AS TEXTFILE  **LOCATION '/rawdata'**  TBLPROPERTIES ("skip.header.line.count"="1"); |

* Run a query to view the data:

|  |
| --- |
| SELECT \* FROM grades\_ext; |

* Drop the table

|  |
| --- |
| DROP TABLE grades\_ext; |

* Validate the table no longer exists

|  |
| --- |
| SHOW TABLES; |

* Exist Hive

exit;

* Validate the data remains untouched

hdfs dfs -ls /rawdata

We loaded data into HDFS and created an external Hive table on top of it to query the records, demonstrating Hive’s schema-on-read capability. After dropping the table, we confirmed that the underlying data in HDFS was untouched, highlighting how Hive manages only the metadata for external tables. This exercise shows the power of schema-on-read, allowing us to apply structure for analysis without altering or moving the raw data.

**Deliverable:** Screenshots of each SQL command and its output.

## Shutting Down

Ensure all Docker containers are turned off with docker-compose down for each directory. If you’re using google cloud, please shut down your virtual machine to preserve cloud costs.