Task 6:

1. Demonstrate the usage of Hive User Defined Function (UDF) to manipulate data in a

Hive table.

What are we doing in this experiment :

1. Create a **Java-based UDF**
2. **Compile** it into a .jar file
3. **Upload** it to Hive
4. **Register** the UDF in Hive
5. Use the function on a table column (like customer\_name or product\_name) and show the transformation

**What is a Hive UDF?**

A **User Defined Function (UDF)** in Hive is:

* A custom function written by you (usually in **Java**, but Python is also possible with Hive-on-Spark)
* It **extends Hive’s built-in capabilities**
* You use it to **transform or process data** in custom ways that built-in Hive functions (like UPPER(), SUBSTR(), CONCAT()) can’t handle.

## STEPS to Use the UDF in Hive

We’ll follow this 6-step plan inside our **Cloudera VM**:

| **Step** | **Task** |
| --- | --- |
| 1️ | Upload the Java file to Cloudera using **WinSCP** |
| 2️ | Compile the Java file into a .jar using javac |
| 3️ | Package the .class file into a .jar using jar |
| 4️ | Start **Hive** and register the .jar |
| 5️ | Create a **temporary function** in Hive for your UDF |
| 6️ | Test it on a sample table or dataset |

## STEP 1: Upload ReverseCase.java to Cloudera VM using **WinSCP**

1. Open **WinSCP**
2. Connect to your Cloudera VM using the IP address.
3. Upload the file ReverseCase.java to the guest OS.

Step 2: Compile the Java file — into a folder named myudfs, using the following command :

mkdir -p myudfs

mv ReverseCase.java myudfs/

javac -classpath `hadoop classpath`:/usr/lib/hive/lib/\* myudfs/ReverseCase.java

Explanation :

* What is “mkdir -p myudfs”

| **Command Part** | **Meaning** |
| --- | --- |
| mkdir | Command to create a directory |
| -p | Creates parent directories if needed (avoids error if already exists) |
| myudfs | The folder named after your Java package myudfs; |
| *Hive needs the class inside a proper package directory (myudfs) to match the Java package.* |  |

-what is “mv ReverseCase.java myudfs/”

| **Command Part** | **Meaning** |
| --- | --- |
| mv | Move command — also used for renaming |
| ReverseCase.java | Source Java file to move |
| myudfs/ | Destination folder for the file |
| *We are organizing the file so it compiles correctly into myudfs.ReverseCase.* |  |

-what is “javac -classpath `hadoop classpath`:/usr/lib/hive/lib/\* myudfs/ReverseCase.java”

| **Command Part** | **Meaning** |
| --- | --- |
| javac | Java compiler |
| -classpath | Tells the compiler where to find the required libraries |
| `hadoop classpath` | Adds Hadoop JARs to classpath (used by Hive) |
| :/usr/lib/hive/lib/\* | Adds all Hive JARs to classpath |
| myudfs/ReverseCase.java | Path to the source file to compile |
| *This compiles the Java file into myudfs/ReverseCase.class.* |  |

Step 3: Create the jar with the correct package path, using the following command

jar -cf reversecase.jar myudfs/ReverseCase.class

Explanation :

| **Command Part** | **Meaning** |
| --- | --- |
| jar | Java Archiver tool |
| -c | Create a new archive |
| -f | Specify archive filename |
| reversecase.jar | Output JAR file |
| myudfs/ReverseCase.class | File to include in the JAR |
| *Now the JAR contains the correct folder and class path: myudfs.ReverseCase.* |  |

Step 4: Restart Hive and re-add the jar, using the following commands

First, start the hive, using the following command :

Hive

Now, inside Hive, give the following command :

ADD JAR /home/cloudera/reversecase.jar;

CREATE TEMPORARY FUNCTION reverse\_case AS 'myudfs.ReverseCase';

Explanation :

-what is “ADD JAR /home/cloudera/reversecase.jar;”

| **Part** | **Meaning** |
| --- | --- |
| ADD JAR | Tells Hive to load an external Java JAR file |
| /home/cloudera/reversecase.jar | Path to your compiled UDF JAR |
| *Makes your UDF available to Hive for this session.* |  |

-what is “CREATE TEMPORARY FUNCTION reverse\_case AS 'myudfs.ReverseCase';”

| **Part** | **Meaning** |
| --- | --- |
| CREATE TEMPORARY FUNCTION | Defines a function usable in current Hive session |
| reverse\_case | Hive function name (you can call this in queries) |
| 'myudfs.ReverseCase' | Fully-qualified class name inside the JAR |
| *This maps your Java method to a Hive function.* |  |

STEP 5: Test the UDF on a Hive Table

### A. Create a Simple Test Table

At the hive> prompt, enter the following command:

CREATE TABLE test\_names(name STRING);

Note : This creates a table with just one column called name

B. Insert Sample Data into the Table, using the following command :

INSERT INTO test\_names VALUES

('Alice'),

('Bob'),

('Cloudera'),

('BigData'),

('hIVE');

Note : This adds 5 names into the table, which we’ll manipulate using our UDF.

C. Query Using Your UDF

Now, let’s apply our UDF reverse\_case() to these names, using the following command:

SELECT name, reverse\_case(name) AS reversed\_name FROM test\_names;

What this does:

* Shows the **original name** and the **reversed-case version** side by side.

Task completed, we have:

* Created a Hive table
* Inserted sample data
* Successfully tested our own **Java-based UDF** on it