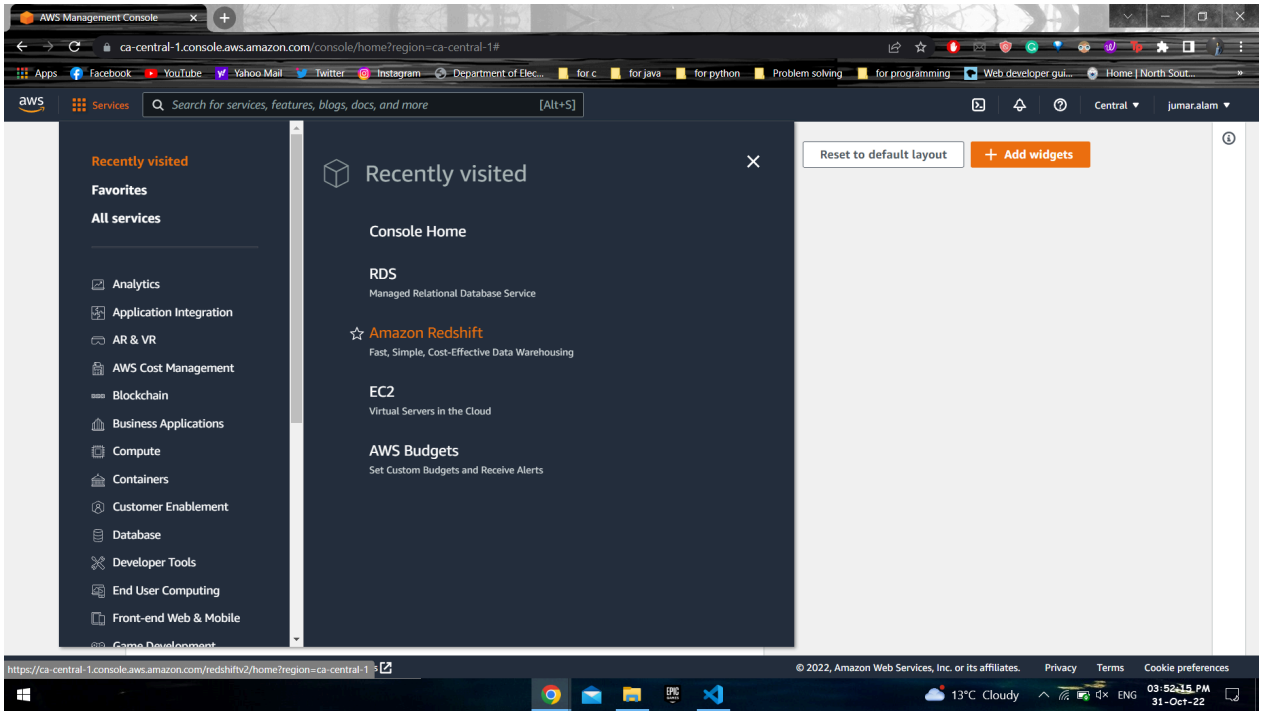
Assignment-6

Redis: 55 points

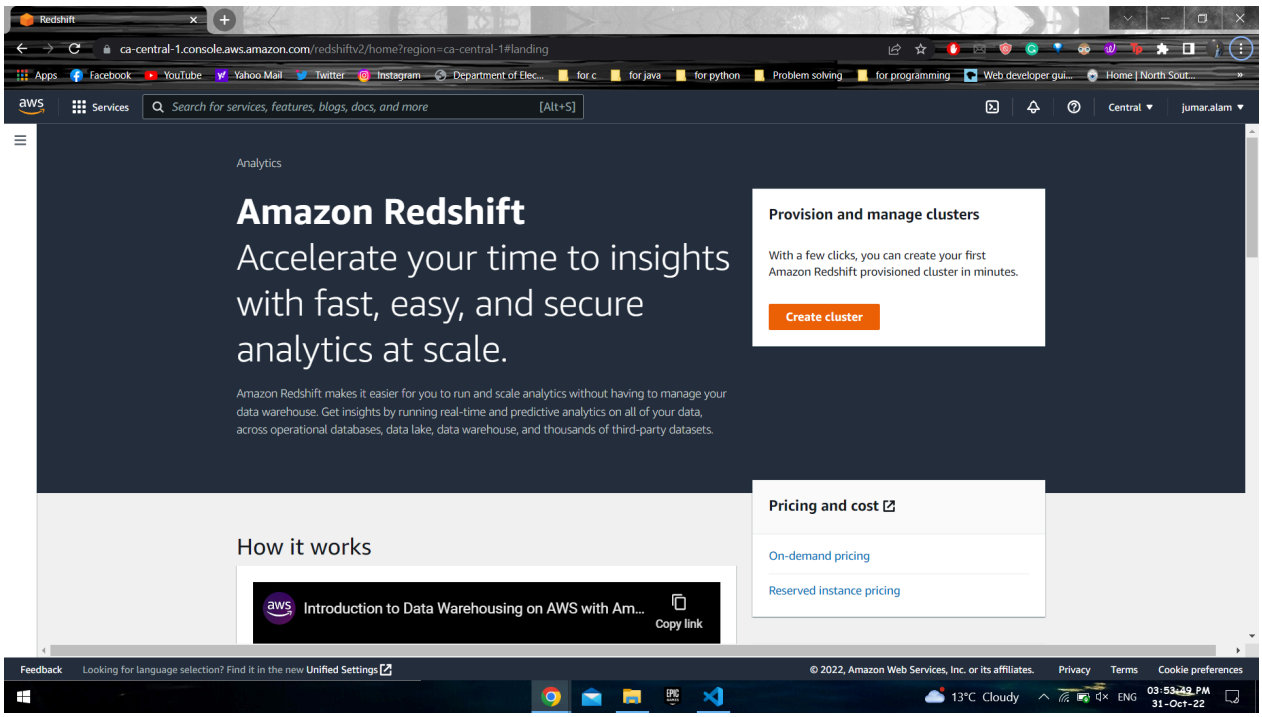
> **Submit a PDF with code listing, and screenshots showing outputs of insert(), delete(), and the queries. Screenshots should be uniquely distinguishable for each submission. Be careful of plagiarism from online sources/peers.**

Amazon Redshift

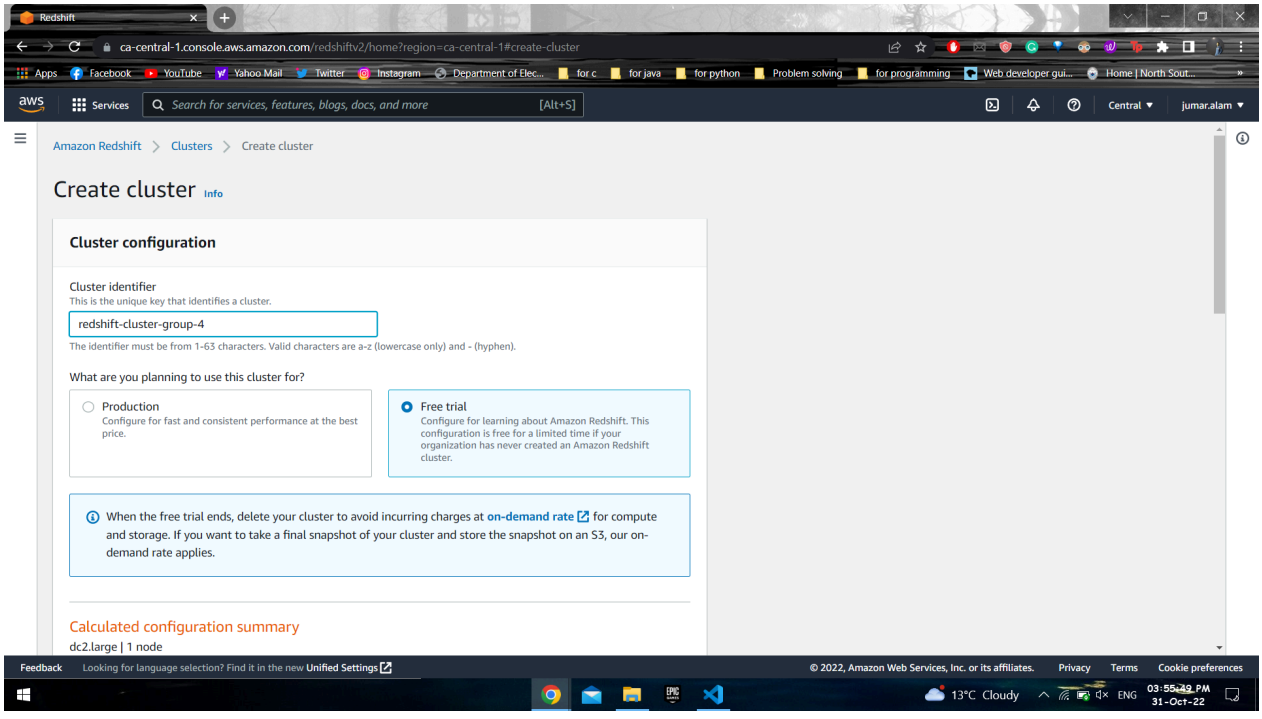
Login to AWS console, click Services and then Amazon Redshift



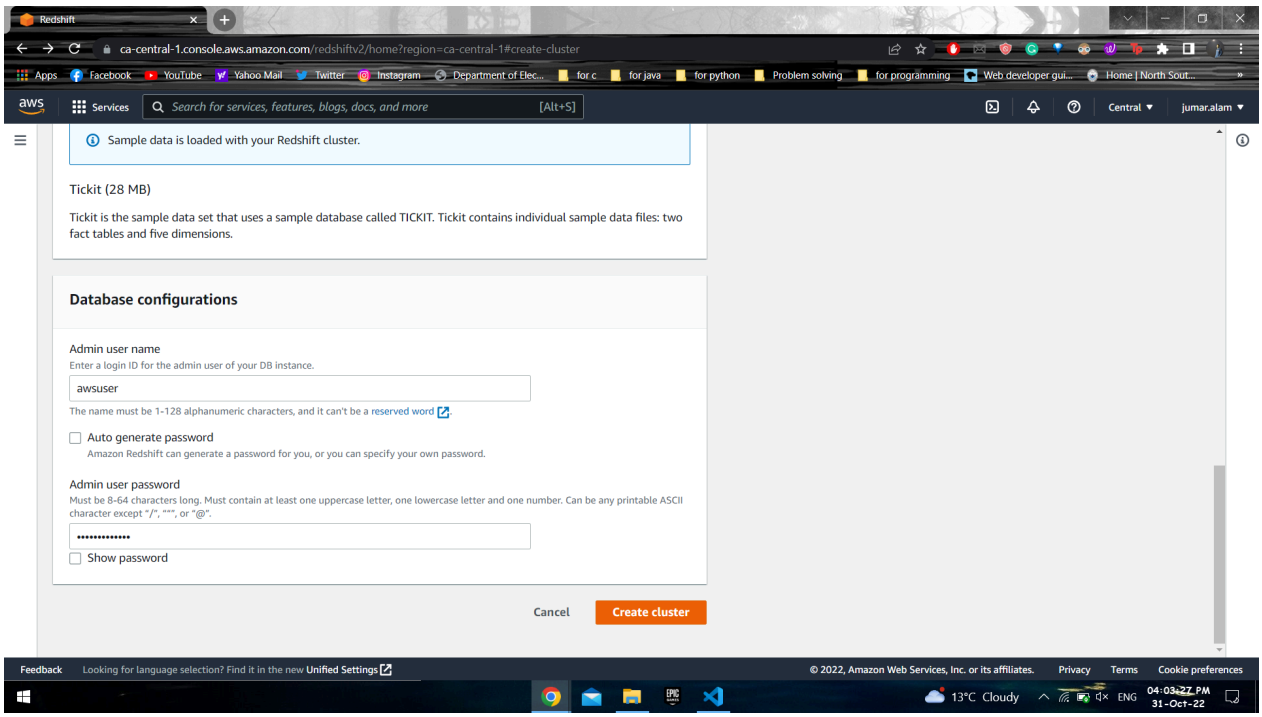
Click on Create Cluster



For cluster identifier, use a unique identifier ad select free trial:

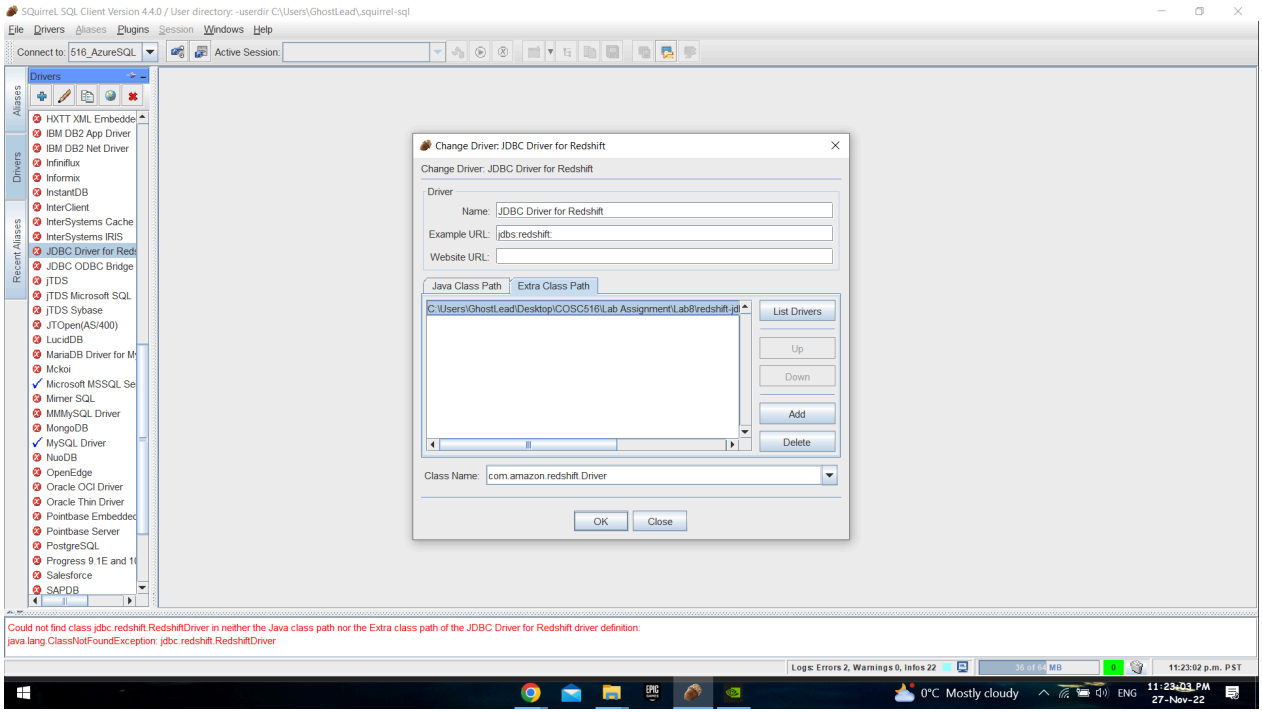


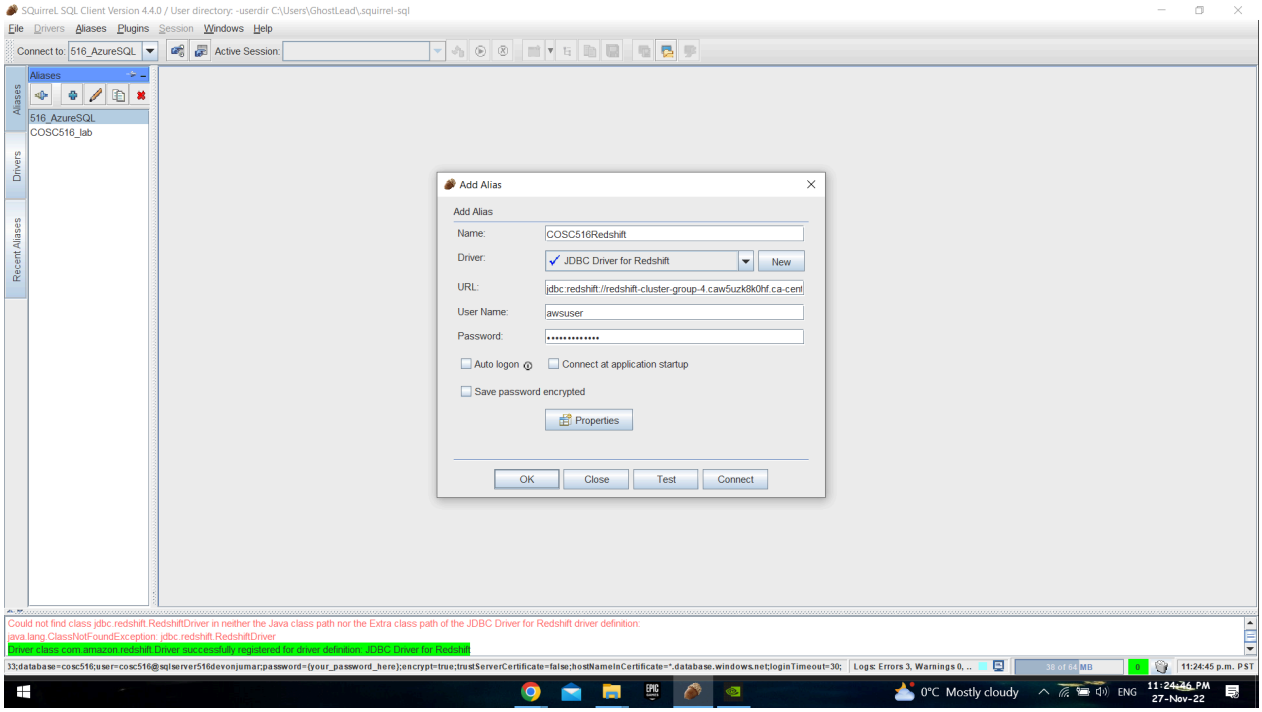
Enter login ID and password for admin user of database. Clock Create Cluster



Configure VPC and Network access like you did earlier in Amazon RDS

You can use SQuirrel again to access the database:





Now do the following tasks with the help of the starter code provided below

1. Write the method connect() to make a connection to the database. [5]
2. Method close() to close the connection to the database. [5]
3. Method drop() to drop all the tables from the database. Note: The database schema name will be dev. [5]
4. Method create() to create the database dev and the tables. [5]
5. Write the method insert() to add the standard TPC-H data into the database. The DDL files are in the ddl folder. Hint: Files are designed so can read entire file as a string and execute it as one statement. May need to divide up into batches for large files. [10]
6. Write the method query1() that returns the most recent top 10 orders with the total sale and the date of the order for customers in America. [5]
7. Write the method query2() that returns the customer key and the total price a customer spent in descending order, for all urgent orders that are not failed for all customers who are outside Europe and belong to the largest market segment. The largest market segment is the market segment with the most customers. [10]
8. Write the method query3() that returns a count of all the line items that were ordered within the six years starting on April 1st, 1997 group by order priority. Make sure to sort by order priority in ascending order. [10]
9. Try to implement some basic ML techniques for classifying the total price using traditional techniques such as SVM, Random Forest, Linear Regression using Redshift. This is not a part of the evaluation but will help you learn the ML features of Redshift.

Starter:

```
import java.math.BigDecimal;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.ResultSetMetaData;
import java.sql.SQLException;
import java.sql.Statement;
import java.util.Arrays;
import java.util.List;
import java.util.Scanner;
import java.io.File;


/**
 * Performs SQL DDL and SELECT queries on a MySQL database hosted on AWS RDS.
 */
public class AmazonRedshift
{
	/**
	 * Connection to database
	 */
	private Connection con;

	/**
	 * TODO: Fill in AWS connection information.
	 */
	//private String url = ;
	//private String uid = ;
	//private String pw = ;

	/**
	 * Main method is only used for convenience.  Use JUnit test file to verify your answer.
	 *
	 * @param args
	 *                  none expected
	 * @throws SQLException
	 *                  if a database error occurs
	 */
	public static void main(String[] args) throws SQLException
```

```
	{
		AmazonRedshift q = new AmazonRedshift();
		q.connect();
		q.drop();
		q.create();
		q.insert();
		q.query1();
		q.query2();
		q.query3();

		q.close();
	}

	/**
	 * Makes a connection to the database and returns connection to caller.
	 *
	 * @return
	 *		connection
	 * @throws SQLException
	 *		if an error occurs
	 */
	public Connection connect() throws SQLException
	{
		// TODO: For connect to work you must configure your AWS connection info in the private instance variables at the top of the file.
		// If connection fails, make sure to modify your VPC rules to along inbound traffic to the database from your IP.
		System.out.println("Connecting to database.");
		// Note: Must assign connection to instance variable as well as returning it back to the caller
		//con = ;
		return con;
	}


	/**
	 * Closes connection to database.
	 */
	public void close()
	{
		System.out.println("Closing database connection.");

	}
```

```
	public void drop()
	{
		System.out.println("Dropping all the tables");

	}


	public void create() throws SQLException
	{
		System.out.println("Creating Tables");
	}


	public void insert() throws SQLException
	{
		System.out.println("Loading TPC-H Data");

	}



	/**
	 * Query returns the most recent top 10 orders with the total sale and the date of the order in `America`.
	 *
	 * @return
	 *			ResultSet
	 * @throws SQLException
	 *			if an error occurs
	 */
	public ResultSet query1() throws SQLException
	{
		System.out.println("Executing query #1.");

		return ;
	}

	/**
	 * Query returns the customer key and the total price a customer spent in descending order, for all urgent orders that are not failed for all customers who are outside Europe belonging to the highest market segment.
	 *
	 * @return
	 *			ResultSet
	 * @throws SQLException
```

```
         *                  if an error occurs
         */
        public ResultSet query2() throws SQLException
        {
                System.out.println("Executing query #2.");

                return ;
        }

        /**
         * Query returns all the lineitems that was ordered within the six years from January 4th,
1997 and the orderpriority in ascending order.
         *
         * @return
         *                  ResultSet
         * @throws SQLException
         *                  if an error occurs
         */
        public ResultSet query3() throws SQLException
        {
                System.out.println("Executing query #3.");
                return;
        }

        /*
         * Do not change anything below here.
         */
        /**
   * Converts a ResultSet to a string with a given number of rows displayed.
   * Total rows are determined but only the first few are put into a string.
   *
   * @param rst
   *                ResultSet
   * @param maxrows
   *                maximum number of rows to display
   * @return
   *                String form of results
   * @throws SQLException
   *                if a database error occurs
   */
  public static String resultSetToString(ResultSet rst, int maxrows) throws SQLException
  {
     StringBuffer buf = new StringBuffer(5000);
     int rowCount = 0;
     ResultSetMetaData meta = rst.getMetaData();
```

```
        buf.append("Total columns: " + meta.getColumnCount());
        buf.append('\n');
        if (meta.getColumnCount() > 0)
            buf.append(meta.getColumnName(1));
        for (int j = 2; j <= meta.getColumnCount(); j++)
            buf.append(", " + meta.getColumnName(j));
        buf.append('\n');

        while (rst.next())
        {
            if (rowCount < maxrows)
            {
                for (int j = 0; j < meta.getColumnCount(); j++)
                {
                        Object obj = rst.getObject(j + 1);

                        buf.append(obj);
                    if (j != meta.getColumnCount() - 1)
                        buf.append(", ");
                }
                buf.append('\n');
            }
            rowCount++;
        }
        buf.append("Total results: " + rowCount);
        return buf.toString();
    }

    /**
     * Converts ResultSetMetaData into a string.
     *
     * @param meta
     *                ResultSetMetaData
     * @return
     *                string form of metadata
     * @throws SQLException
     *                if a database error occurs
     */
    public static String resultSetMetaDataToString(ResultSetMetaData meta) throws
SQLException
    {
            StringBuffer buf = new StringBuffer(5000);
            buf.append(meta.getColumnName(1)+" ("+meta.getColumnLabel(1)+",
"+meta.getColumnType(1)+"-"+meta.getColumnTypeName(1)+",
"+meta.getColumnDisplaySize(1)+", "+meta.getPrecision(1)+", "+meta.getScale(1)+")");
```

```
            for (int j = 2; j <= meta.getColumnCount(); j++)
                buf.append(", "+meta.getColumnName(j)+" ("+meta.getColumnLabel(j)+",
"+meta.getColumnType(j)+"-"+meta.getColumnTypeName(j)+",
"+meta.getColumnDisplaySize(j)+", "+meta.getPrecision(j)+", "+meta.getScale(j)+")");
            return buf.toString();
    }
}
```