3.6 Transaction Management

This section will guide you to:

- Understand how to use JDBC transaction management.

Development Environment:

- Mysql 5.7
- Eclipse IDE
- Java 1.8

This guide has three subsections, namely:

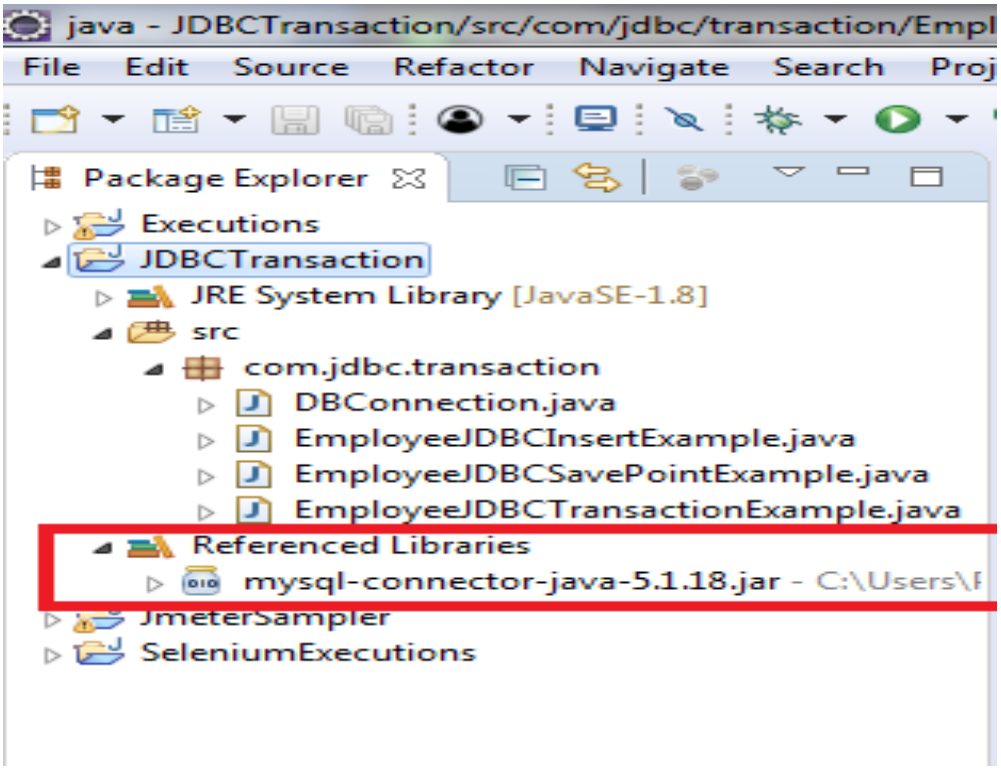
3.6.1. Writing a program to perform JDBC transaction management using Auto-Commit Mode.
3.6.2. Writing a program to perform JDBC transaction management by disabling setAutoCommit().
3.6.3. Pushing the code to your GitHub repositories.

Step 3.6.1: Writing a program to perform JDBC transaction management using Auto-Commit Mode.

- By default, when we create a database connection, it runs in **auto-commit** mode. It means that whenever we execute a query, the commit is fired automatically. So, every SQL query we fire is a transaction and if we are running DML or DDL queries, the changes are getting saved in the database after every SQL statement is executed .
- Sometimes we want a group of SQL queries to be part of a transaction, so that we can commit them when all the queries run successfully. If we get any exception, we have a choice to rollback all the queries executed as part of the transaction.
- Let's understand with a simple example where we want to utilize JDBC transaction management support for data integrity. Let's say we have "transaction_management" database and employee information saved in two tables. Example: I am using MySQL database.
- Create two tables 'employee' and 'address' in 'transaction_management' database using the credentials below:

```
CREATE TABLE transaction_management.employee (
 empId int(11) unsigned NOT NULL,
 name varchar(20) DEFAULT NULL,
 PRIMARY KEY (`empId`)) ENGINE=InnoDB DEFAULT CHARSET=utf8;

CREATE TABLE transaction_management.address (
 empId int(11) unsigned NOT NULL,
 address varchar(20) DEFAULT NULL,
 city varchar(5) DEFAULT NULL,
 country varchar(20) DEFAULT NULL,
 PRIMARY KEY (`empId`)) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

- Open Eclipse
- Create Java Project. Ex: JDBCTransaction
- Download “mysql-connector-java-5.1.18.jar”
- Add External jar “mysql-connector-java-5.1.18.jar” into the project



- Create a class called “DBConnection.java” and give the database credentials as below:
  - **DB_URL:** jdbc:mysql://localhost:3307/transaction_management
  - **DB_DRIVER_CLASS:** com.mysql.jdbc.Driver
  - **DB_USERNAME:** The username of database (here: **root)**
  - **DB_PASSWORD:** Password for the username (here: **root)**

```
package com.jdbc.transaction;

import java.sql.Connection;

import java.sql.DriverManager;

import java.sql.SQLException;


public class DBConnection {


        public final static String DB_DRIVER_CLASS =

        "com.mysql.jdbc.Driver";

        public final static String DB_URL =

        "jdbc:mysql://localhost:3307/transaction_management";

        public final static String DB_USERNAME = "root";

        public final static String DB_PASSWORD = "root";


        public static Connection getConnection() throws

           ClassNotFoundException, SQLException {


                Connection con = null;


                // load the Driver Class

                Class.forName(DB_DRIVER_CLASS);


                // create the connection now

                con = DriverManager.getConnection(DB_URL,
```

```
                DB_USERNAME, DB_PASSWORD);

            System.out.println("DB Connection created
            successfully");
            return con;
        }
    }
```

- DBConnection is the class used by other classes for MYSQL database connection.
- Create another class called “EmployeeJDBCInsertExample.java”

```
package com.jdbc.transaction;
import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.SQLException;

public class EmployeeJDBCInsertExample {

    public static final String INSERT_EMPLOYEE_QUERY =
  "insert into Employee (empId, name) values (?,?)";

    public static final String INSERT_ADDRESS_QUERY = "insert into
    Address (empId, address, city, country) values (?,?,?,?)";

    public static void main(String[] args) {

        Connection con = null;
        try {
```

```
                con = DBConnection.getConnection();

                insertEmployeeData(con, 1, "Pankaj");

                insertAddressData
                (con, 1, "Albany Dr", "San Jose", "USA");
            } catch (SQLException | ClassNotFoundException e) {
                e.printStackTrace();
            } finally {

                try {
                    if (con != null)
                        con.close();
                } catch (SQLException e) {
                    e.printStackTrace();
                }
            }
        }

    public static void insertAddressData(Connection con, int id,
     String address, String city, String country) throws SQLException {
            PreparedStatement stmt =
            con.prepareStatement(INSERT_ADDRESS_QUERY);
            stmt.setInt(1, id);
            stmt.setString(2, address);
            stmt.setString(3, city);
            stmt.setString(4, country);

            stmt.executeUpdate();
```

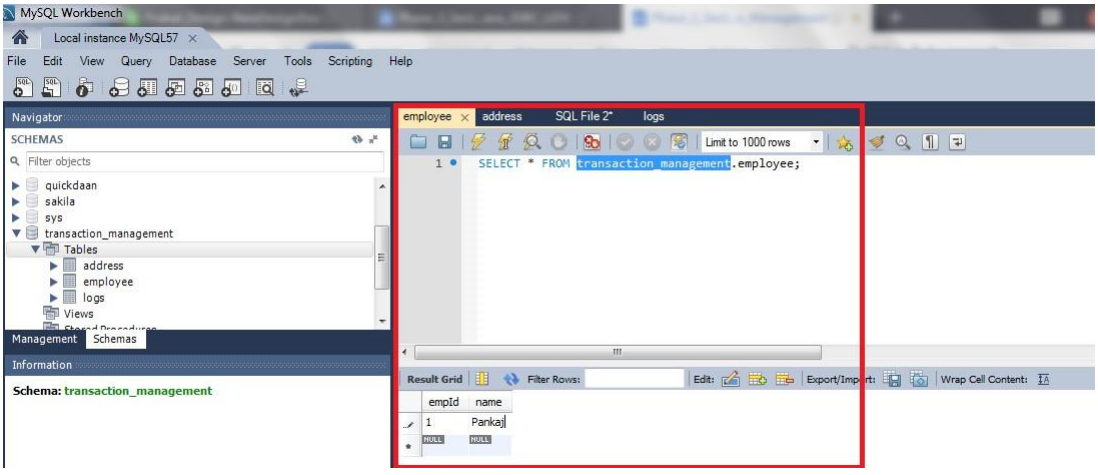
```
                System.out.println("Address Data inserted successfully
                for ID=" + id);
                stmt.close();
        }

        public static void insertEmployeeData(Connection con, int id,
         String name) throws SQLException {
                PreparedStatement stmt =
                con.prepareStatement(INSERT_EMPLOYEE_QUERY);
                stmt.setInt(1, id);
                stmt.setString(2, name);

                stmt.executeUpdate();

                System.out.println("Employee Data inserted
                successfully for ID=" + id);
                stmt.close();
        }
}
```

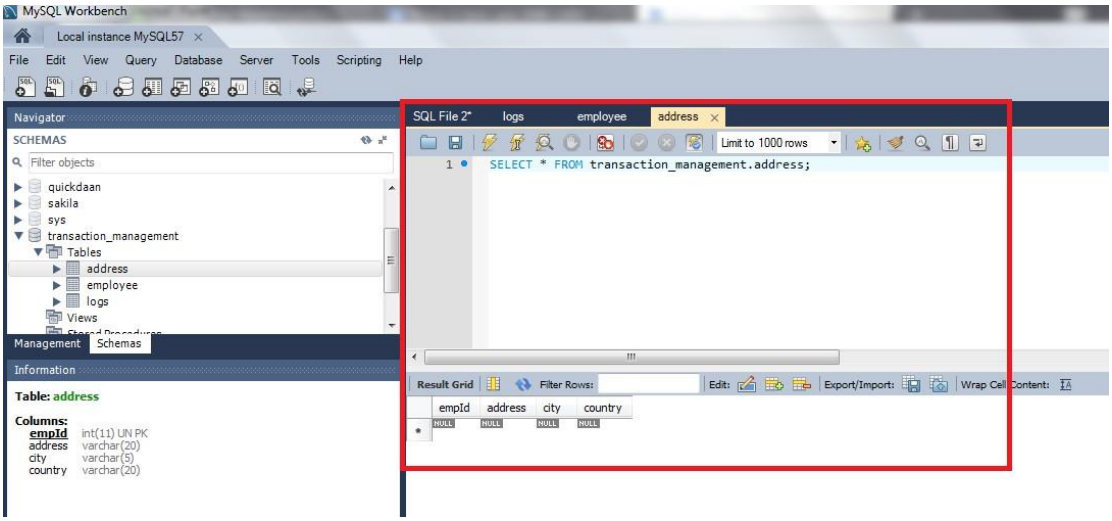
- By running the "EmployeeJDBCInsertExample.java" program, we will get the following output:

```
DB Connection created successfully
Employee Data inserted successfully for ID=1
com.mysql.jdbc.MysqlDataTruncation: Data truncation: Data too
long for column 'city' at row 1
        at com.mysql.jdbc.MysqlIO.checkErrorPacket(MysqlIO.java:2939)
        at com.mysql.jdbc.MysqlIO.sendCommand(MysqlIO.java:1623)
        at com.mysql.jdbc.MysqlIO.sqlQueryDirect(MysqlIO.java:1715)
        at com.mysql.jdbc.Connection.execSQL(Connection.java:3249)
```

```
at com.mysql.jdbc.PreparedStatement.executeInternal
(PreparedStatement.java:1268)
at com.mysql.jdbc.PreparedStatement.executeUpdate
(PreparedStatement.java:1541)
at com.mysql.jdbc.PreparedStatement.executeUpdate
(PreparedStatement.java:1455)
at com.mysql.jdbc.PreparedStatement.executeUpdate
(PreparedStatement.java:1440)
at com.journaldev.jdbc.transaction.EmployeeJDBCInsertExample.
insertAddressData(EmployeeJDBCInsertExample.java:45)
at com.journaldev.jdbc.transaction.EmployeeJDBCInsertExample.
main(EmployeeJDBCInsertExample.java:23)
```

- As you can see, SQLException is only raised when we are trying to insert data into the address table, because the value is bigger than the size of the column.
- If you look at the content in the employee and address tables, you will notice that data is present only in the employee table.





- By running the program again, it will try to insert employee information into the employee table again and will throw the below exception.

```
com.mysql.jdbc.exceptions.MySQLIntegrityConstraintViolationExceptio
n: Duplicate entry '1' for key 'PRIMARY'
        at com.mysql.jdbc.SQLError.createSQLException
        (SQLError.java:931)
        at com.mysql.jdbc.MysqlIO.checkErrorPacket(MysqlIO.java:2941)
        at com.mysql.jdbc.MysqlIO.sendCommand(MysqlIO.java:1623)
        at com.mysql.jdbc.MysqlIO.sqlQueryDirect(MysqlIO.java:1715)
        at com.mysql.jdbc.Connection.execSQL(Connection.java:3249)
        at com.mysql.jdbc.PreparedStatement.executeInternal
        (PreparedStatement.java:1268)
        at com.mysql.jdbc.PreparedStatement.executeUpdate
        (PreparedStatement.java:1541)
        at com.mysql.jdbc.PreparedStatement.executeUpdate
        (PreparedStatement.java:1455)
        at com.mysql.jdbc.PreparedStatement.executeUpdate
        (PreparedStatement.java:1440)
        at com.journaldev.jdbc.transaction.EmployeeJDBCInsertExample.
        insertEmployeeData(EmployeeJDBCInsertExample.java:57)
```

```
        at com.journaldev.jdbc.transaction.EmployeeJDBCInsertExample.
        main(EmployeeJDBCInsertExample.java:21)
```

- Now, there is no way we can save the data in the address table for the Employee. Since this program leads to data integrity issues, we need transaction management to insert data into both the tables successfully or rollback everything if any exception arises.

Step 3.6.2: Writing a program to perform JDBC transaction management by disabling setAutoCommit().

- JDBC API provides the method setAutoCommit() through which we can disable the auto commit feature of the connection (should disable when it's required because the transaction will not be committed unless we call the commit() method on connection).
- Let's write another program where we will use JDBC transaction management feature to make sure data integrity is not violated.

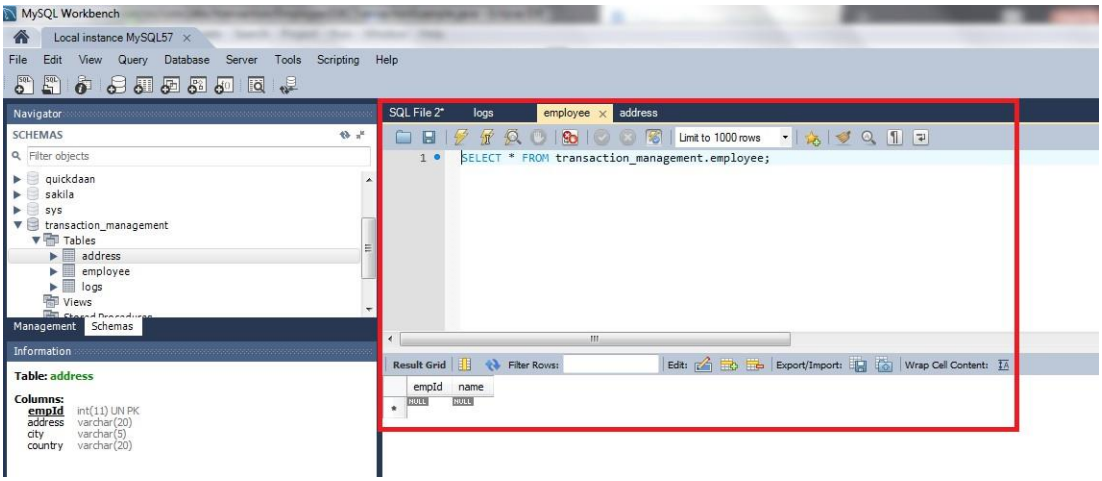
```
package com.jdbc.transaction;

import java.sql.Connection;
import java.sql.SQLException;
public class EmployeeJDBCTransactionExample {

    public static void main(String[] args) {

        Connection con = null;
        try {
            con = DBConnection.getConnection();

            //set auto commit to false
            con.setAutoCommit(false);

            EmployeeJDBCInsertExample.insertEmployee
```

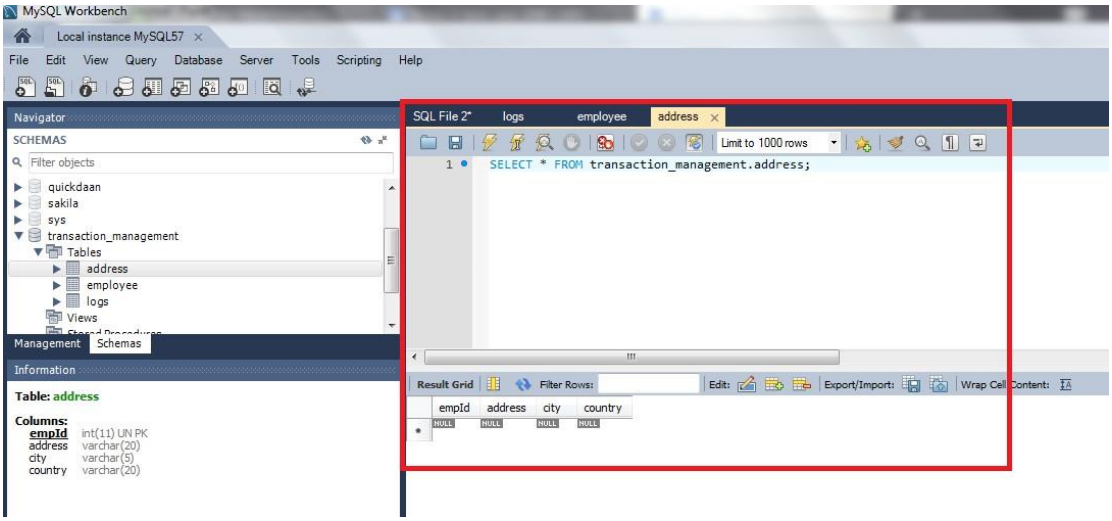
```
                        Data(con, 1, "Pankaj");
                        EmployeeJDBCInsertExample.insertAddress
                        Data(con, 1, "Albany Dr", "San Jose", "USA");

                        //now commit transaction
                        con.commit();

                } catch (SQLException e) {
                        e.printStackTrace();
                        try {
                                con.rollback();
                                System.out.println("JDBC
                                Transaction rolled back successfully");
                        } catch (SQLException e1) {
                                System.out.println("SQLException in
                                rollback"+e.getMessage());
                        }
                } catch (ClassNotFoundException e) {
                        e.printStackTrace();
                } finally {
                        try {
                                if (con != null)
                                        con.close();
                        } catch (SQLException e) {
                                e.printStackTrace();
                        }
                }
        }
}
```

- Please make sure you remove the earlier inserted data from both the tables before running this program. By running this program, you will get the following output:

```
DB Connection created successfully
Employee Data inserted successfully for ID=1
com.mysql.jdbc.MysqlDataTruncation: Data truncation: Data too long
for column 'city' at row 1
        at com.mysql.jdbc.MysqlIO.checkErrorPacket(MysqlIO.java:2939)
        at com.mysql.jdbc.MysqlIO.sendCommand(MysqlIO.java:1623)
        at com.mysql.jdbc.MysqlIO.sqlQueryDirect(MysqlIO.java:1715)
        at com.mysql.jdbc.Connection.execSQL(Connection.java:3249)
        at com.mysql.jdbc.PreparedStatement.executeInternal
        (PreparedStatement.java:1268)
        at com.mysql.jdbc.PreparedStatement.executeUpdate
        (PreparedStatement.java:1541)
        at com.mysql.jdbc.PreparedStatement.executeUpdate
        (PreparedStatement.java:1455)
        at com.mysql.jdbc.PreparedStatement.executeUpdate
        (PreparedStatement.java:1440)
        at com.journaldev.jdbc.transaction.EmployeeJDBCInsertExample.
        insertAddressData(EmployeeJDBCInsertExample.java:45)
        at com.journaldev.jdbc.transaction.EmployeeJDBCTransaction
        Example.main(EmployeeJDBCTransactionExample.java:19)
JDBC Transaction rolled back successfully
```

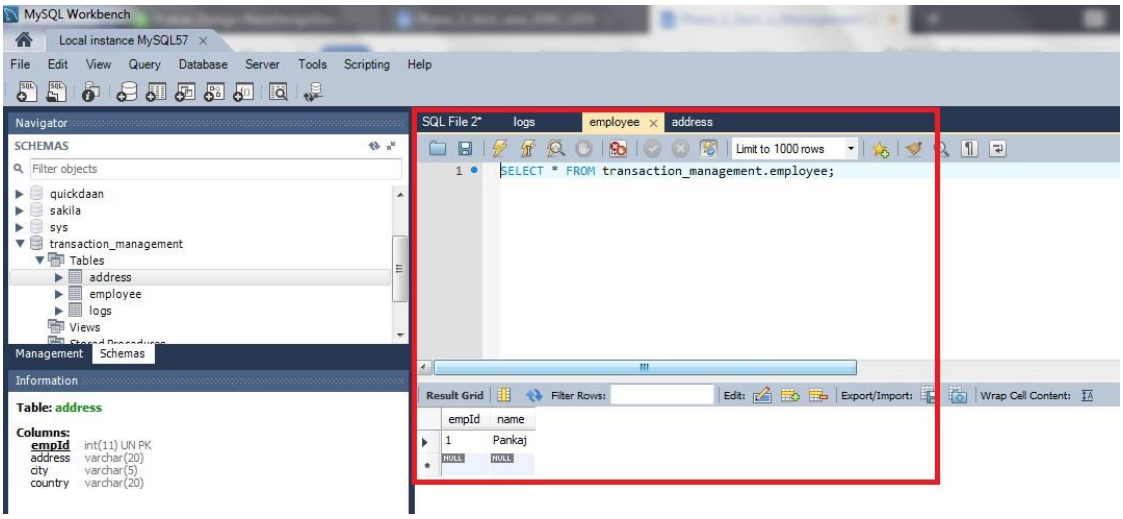
- If you look into the database tables, you will notice that data is not inserted into both employee and address table.

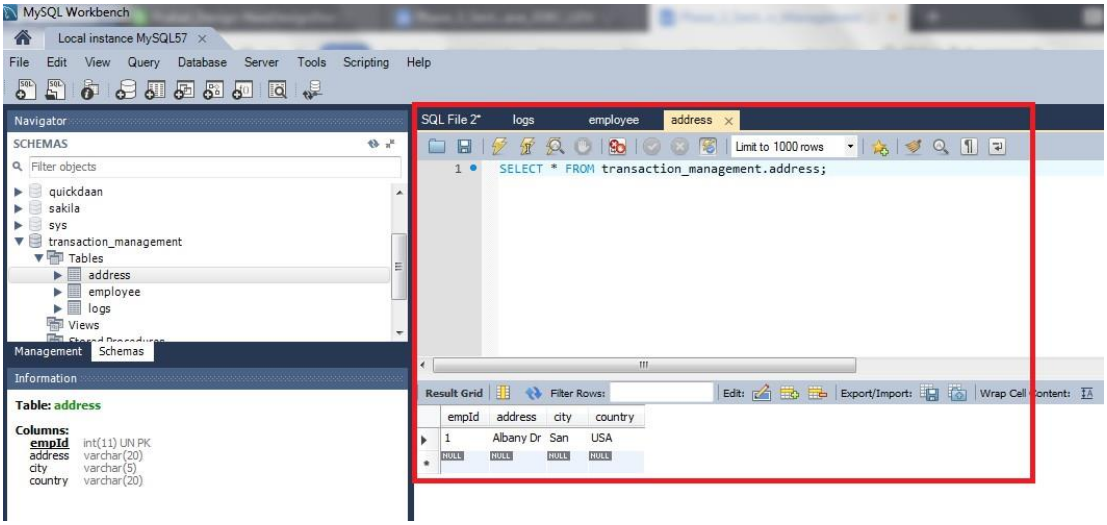




- Now we can change the city value, (here changed “San Jose” to “san” since city column size is 5) so that it can fit in the column and rerun the program to insert data into both the tables.

```
DB Connection created successfully

Employee Data inserted successfully for ID=1

Address Data inserted successfully for ID=1
```





- Notice that connection is committed only when both the inserts are executed successfully. If any of them throws an exception, we are rolling back the complete transaction.

Step 3.6.3: Pushing the code to your GitHub repositories

Open your command prompt and navigate to the folder where you have created your files.
cd jdbc_Demo
Initialize your repository using the following command:
git init
Add all the files to your git repository using the following command:
git add .
Commit the changes using the following command:
git commit . -m “Changes have been committed.”
Push the files to the folder you initially created using the following command:
git push -u origin master