**Guided LAB 305.3.1 - Building Java CRUD application using DAO pattern, OOP concept and JDBC**

## Overview**:**

In this lab, we will demonstrate how to use or implement **DAO pattern and OOP concepts** with **JDBC applications,** including data transfer, database creation, and connecting to a database. We will use CRUD (create, read, update, and delete) operations.

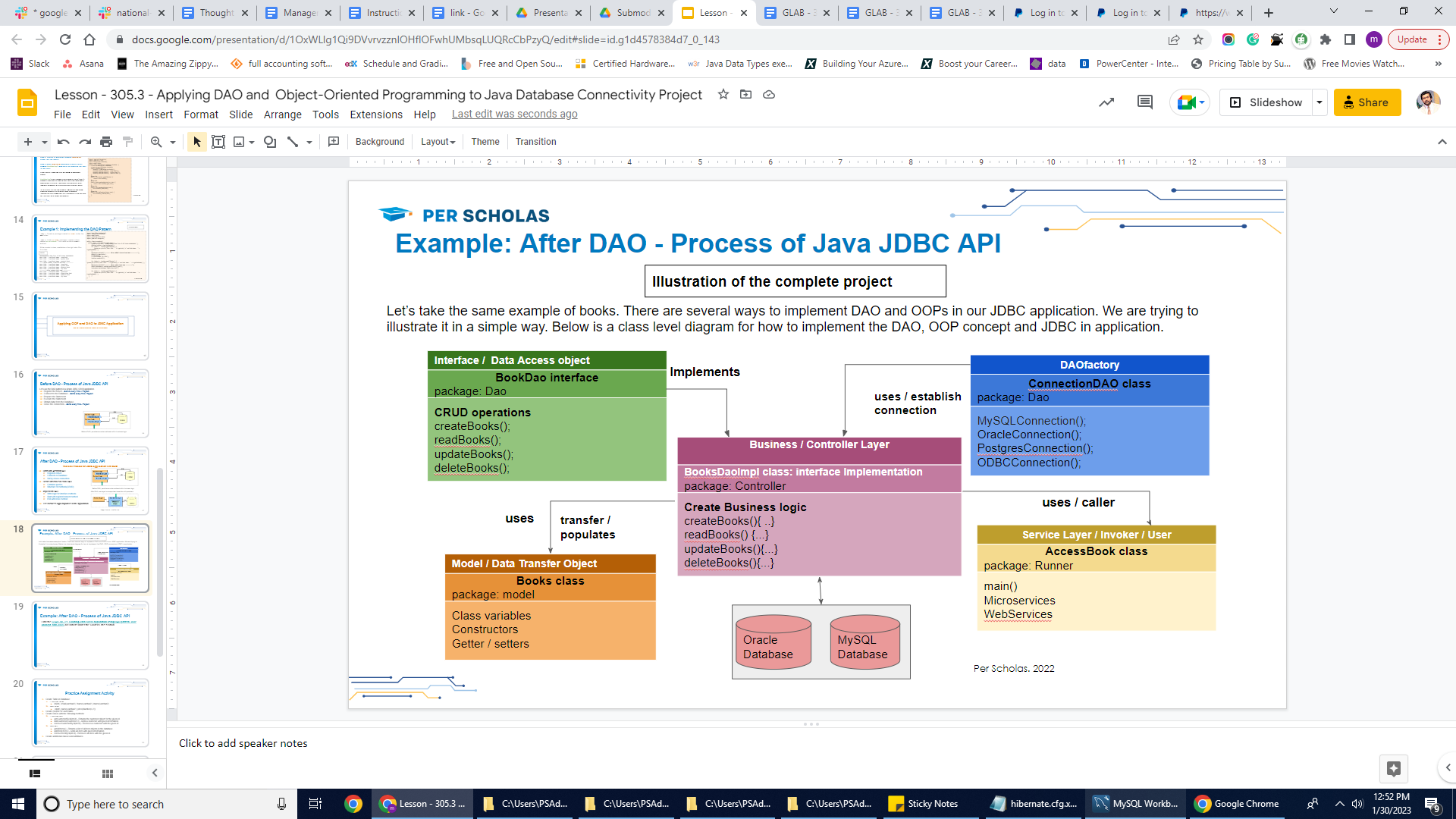
The biggest benefit from DAL is that it simplifies database access operations through some method calls like insert() and find() rather than making connections and executing some queries; this layer instead handles all database related calls and queries inside it.

The aim of this lab is to manage the access of a Book table in the database from a separate layer written in Java, this layer is usually referred to as the **Data Access Layer (DAL)**

## **Learning Objectives:**

* Explain how to use DAO pattern and OOP concepts with JDBC applications.
* Manage the access of a Book Table in the database from a separate layer written in Java.

## Instructions:

Let's take an example of Books. With the DAO design pattern, we have the following components.

* The **Books** class is the **Model** class or **the transfer object,**which is transferred from one layer to the other.
* The **BookDoa** is an **interface** that provides a flexible design.
* The **BooksDaoImp** is a regular class that implements the **interface,** which isthe business layer or controller layer for CRUD operation.
* The **ConnectionDao** is a class where we can specify database connection credentials and Register JDBC driver for each database.
* The **Accessbook** is a class where we can use business logic or invoke any specific operation from business logic. We can use the **main()** method in this class.

## Prerequisite: Creating a Database.

For this lab, create a new database named ***"library"*** and create a table named ***"books"*** under the ***library*** database by using the below SQL statements.

Run the below SQL statement in MySQL:

| CREATE database library;  CREATE TABLE books (  id int PRIMARY KEY auto\_increment,  isbn int Unique Key NOT NULL,  bookName VARCHAR(50) ); |
| --- |

### **Step 1: Setup New project**

### Create a new project by using the Eclipse IDE and add a JDBC driver to the project. You have two choices; either you can add a JDBC jar file or a JDBC Maven dependency. Both options are given below.

**Option A:** [Click here to download the MySql JDBC driver jar.](https://dev.mysql.com/downloads/connector/j/8.0.html)

**Option B:** Add below MySQL maven dependency in pom.xml file

| <dependency>  <groupId>mysql</groupId>  <artifactId>mysql-connector-java</artifactId>  <version>8.0.21</version>  </dependency> |
| --- |

### **Step 2:** Create a package named **"model"**under the ***src*** folder.

### **Step 3:** Under **model** package, create a class named **Books**, and write the code below in that class.

| package model;  public class Books {  private int id;  private int isbn;  private String bookName;  // ----Constructor---  public Books() {  }  public Books(int isbn, String bookName) {  this.isbn = isbn;  this.bookName = bookName;  }  public Books(int id, int isbn, String bookName) {  this.isbn = isbn;  this.bookName = bookName;  This.id = id  }  // getter setter methods  public int getId() {  return id;  }  public void setId(int id) {  this.id = id;  }    public int getIsbn() {  return isbn;  }  public void setIsbn(int isbn) {  this.isbn = isbn;  }  public String getBookName() {  return bookName;  }  public void setBookName(String bookName) {  this.bookName = bookName;  }  } |
| --- |

* The above class is our model class. It has two variables/properties to keep things simple.
* This layer should contain a simple class called Data Transfer Object (DTO). This object is just a simple mapping to the table. Every column in the table would be a member variable in the class.
* A good practice is to provide a default empty constructor, a full constructor with all variables, and a full constructor without the id parameter.

### Step 4: Create a package named "Daointerface," under src folder, create a class named ConnectionDAO under the “Daointerface” package

In this class we would provide connection parameters like database JDBC URL, username and password as final variables, provide a method to return a Connection object or null if it failed to connect, or it may be better throw a runtime exception in that case

Add below code in the **"ConnectionDAO"** class:

| package Daointerface;  import java.sql.Connection;  import java.sql.DriverManager;  import java.sql.PreparedStatement;  import java.sql.ResultSet;  import java.sql.SQLException;  public class ConnectionDAO {  static Connection con = null;  protected PreparedStatement ps = null;  protected ResultSet rs = null;  public static Connection getConnection() throws ClassNotFoundException  {  final String DBURL = "jdbc:mysql://localhost:3305/library";  final String DBUSERNAME = "root";  final String DBPASSWORD = "password";  try {  con = DriverManager.getConnection(DBURL, DBUSERNAME, DBPASSWORD);  System.out.println("Connected Database Successfully");  }  catch (SQLException e) {  System.out.println(e);  e.printStackTrace();  }  return con;  }    public void disconnect()  {  try {  if(rs != null)  {  rs.close();  }  if(ps != null)  {  ps.close();  }  if(con != null)  {  con.close();  }    }catch (Exception e) {  // TODO: handle exception  }    }  } |
| --- |

### Step 5: Create an interface named BookDao under the “Daointerface” package.

## The BookDao interface defines methods for performing CRUD operations on the books table.

## Add the code below in the BookDao interface:

| package DAOinterface;  import java.sql.SQLException;  import java.util.List;  import model.Books;  public interface BookDao {  */\*\* This is the method to be used to list down all the records from the books table.\*/*  **List<Books> getAllBooks() throws ClassNotFoundException, SQLException;**  */\*\* This method to be used to create a record in the books table. \*/*  **void saveBook(List<Books> BookList);**  */\*\* This is the method to be used to delete a record from the Student table corresponding to a passed books id. \*/*  **boolean deleteBook(int id);**  */\*\* This is the method to be used to update a record into the books table. \*/*  **boolean updateBook(Books book, int id);**  } |
| --- |

Here, we have defined the *CRUD* operations that will be exposed to our business layer.

### 

### Step 6: Create a package named "Controller", under the src folder and create a class named BookDaoImpl under the “Controller” package.

Having defined our **BookDao** interface, let’s implement it. **BookDaoImpl** class will be our business layer.

### BookDaoImpl.java class:

| package Controller;  import java.sql.\*;  import java.util.ArrayList;  import java.util.List;  import DAOinterface.BookDao;  import DAOinterface.ConnectionDAO;  import model.Books;  public class BookDaoImpl extends ConnectionDAO implements BookDao {  public void saveBook(List<Books> BookList) {  try {  Connection con = ConnectionDAO.getConnection();  for(Books b: BookList) {  String sqlQuery = "INSERT INTO books (isbn,bookName) VALUES (?,?)";  PreparedStatement prepStmt = con.prepareStatement(sqlQuery);  prepStmt.setInt(1, b.getIsbn());  prepStmt.setString(2, b.getBookName());  int affectedRows = prepStmt.executeUpdate();  System.out.println(affectedRows + " row(s) affected !!");  }  }  catch (ClassNotFoundException e)  {  } catch (SQLException throwables) {  throwables.printStackTrace();  }  }  public List<Books> getAllBooks() {  try {  Connection connection = ConnectionDAO.getConnection();  Statement stmt = connection.createStatement();  ResultSet rs = stmt.executeQuery("SELECT \* FROM books");  List boollist = new ArrayList();  while(rs.next())  {  Books b = new Books();  b.setIsbn( rs.getInt("isbn") );  b.setBookName( rs.getString("bookName") );  boollist.add(b);  }  return boollist;  } catch (SQLException ex) {  ex.printStackTrace();  System.err.format("SQL State: %s\n%s", ex.getSQLState(), ex.getMessage());  } catch (ClassNotFoundException e) {  e.printStackTrace();  }  return null;  }  public boolean updateBook(Books bookObj, int id){  try {  Connection connection = ConnectionDAO.getConnection();  PreparedStatement ps = connection.prepareStatement("UPDATE books SET isbn=?, bookName=? WHERE id=?");  ps.setInt(1, bookObj.getIsbn());  ps.setString(2, bookObj.getBookName());  ps.setInt(3, id);  int i = ps.executeUpdate();  if(i == 1) {  return true;  }  }  catch (ClassNotFoundException e)  {  e.printStackTrace();  }  catch (SQLException e) {  e.printStackTrace();  System.err.format("SQL State: %s\n%s", e.getSQLState(), e.getMessage());  }  return false;  }  public boolean deleteBook(int id) {  try {  Connection connection = ConnectionDAO.getConnection();  PreparedStatement ps = connection.prepareStatement("DELETE FROM books WHERE id=?");  ps.setInt(1, id);  int i = ps.executeUpdate();  if(i == 1) {  return true;  }  } catch (ClassNotFoundException e)  {  e.printStackTrace();  }  catch (SQLException ex) {  System.err.format("SQL State: %s\n%s", ex.getSQLState(), ex.getMessage());  }  return false;  }  } |
| --- |

Our **BookDaoImpl** class supports ***select, insert, update,*** and ***delete*** operations. The business layer remains unaware of the actual persistence logic.

### 

### Step 6: Create a package named "Runner" under src folder, create a class named AccessBook under the “Runner” package, and then declare main() method in this class as shown below.

| package Runner;  import Controller.BookDaoImpl;  import DAOinterface.BookDao;  import model.Books;  public class AccessBook {  public static void main(String[] args) {  *// CRUD and business logic will invoke here*  }  } |
| --- |

Let's test our business layer, which we defined in the above (BookDaoImpl) class.

### **Step 6.1: In**sert Books Record

### By using the saveBook (List<Books> BookList) method, let's add a few records in the database. Add the below code in AccessBook class.

| package Runner;  import Controller.BookDaoImpl;  import DAOinterface.BookDao;  import model.Books;  import java.sql.SQLException;  import java.util.ArrayList;  public class AccessBook {  public static void main(String[] args) {  *// creating object*  BookDao bookDao = new BookDaoImpl();  System.out.println("--------- inserting book records ----------");    ArrayList<Books> BookList = new ArrayList<Books>();  Books b1 = new Books();  b1.setIsbn(120);  b1.setBookName("Java Book");  BookList.add(b1);  Books b2 = new Books();  b2.setIsbn(300);  b2.setBookName("Python Book");  BookList.add(b2);  Books b3 = new Books();  b3.setIsbn(365);  b3.setBookName("JavaScript Book");  BookList.add(b3);  Books b4 = new Books();  b4.setIsbn(256);  b4.setBookName("SQL Book");  BookList.add(b4);  bookDao.saveBook(BookList);  }  } |
| --- |

Run your **Accessbook** class.

**Result on Console**

--------- inserting book records ----------

Connected Database Successfully

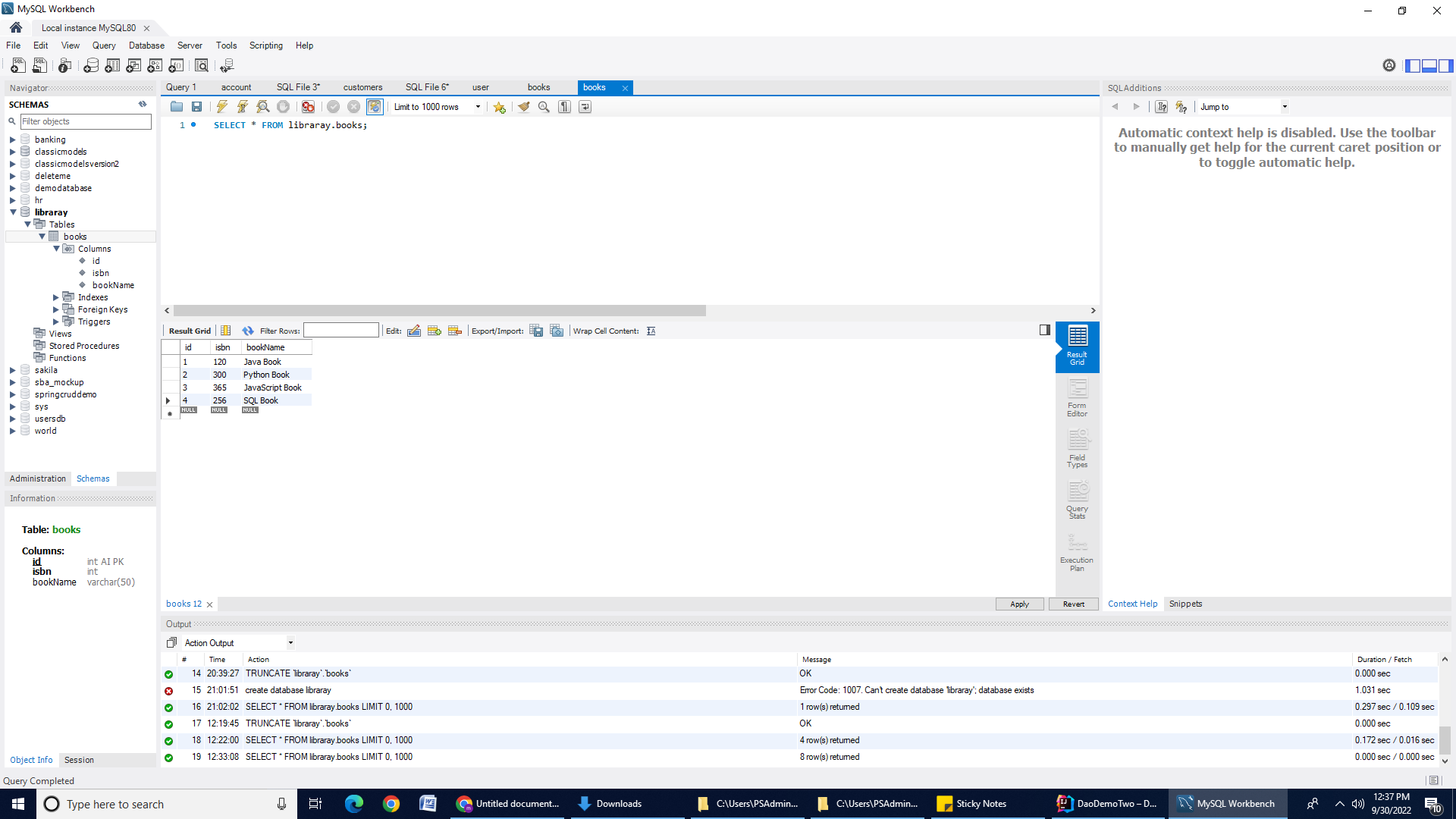
1 row(s) affected !!

1 row(s) affected !!

1 row(s) affected !!

1 row(s) affected !!

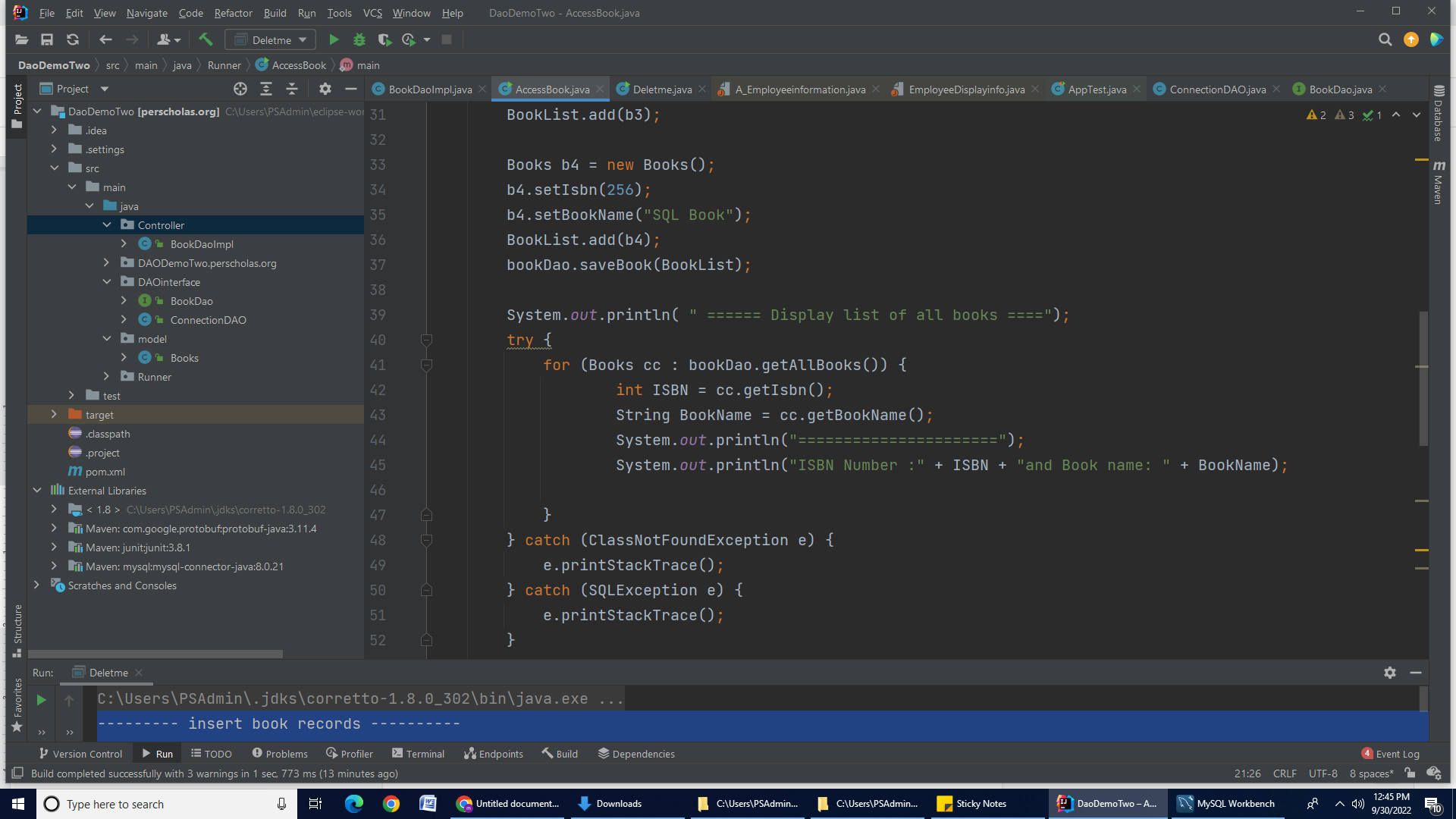
**Result in MySQL:**



### Step 6.2: Retrieve Books

Let's get all the books from the database. We can invoke/use the **getAllBooks()** method in the main() method.Add the code below in the AccessBook class.

| System.out.println( " ====== Display list of all books ====");  try {  for (Books cc : bookDao.getAllBooks()) {  int ISBN = cc.getIsbn();  String BookName = cc.getBookName();  System.out.println("======================");  System.out.println("ISBN Number :" + ISBN + "and Book name: " + BookName);  }  } catch (ClassNotFoundException e) {  e.printStackTrace();  } catch (SQLException e) {  e.printStackTrace();  } |
| --- |



Run your **AccessBook** class.

**Output on Console**

====== Display list of all books ====

Connected Database Successfully

======================

ISBN Number :120and Book name: Java Book

======================

ISBN Number :300and Book name: Python Book

======================

ISBN Number :365and Book name: JavaScript Book

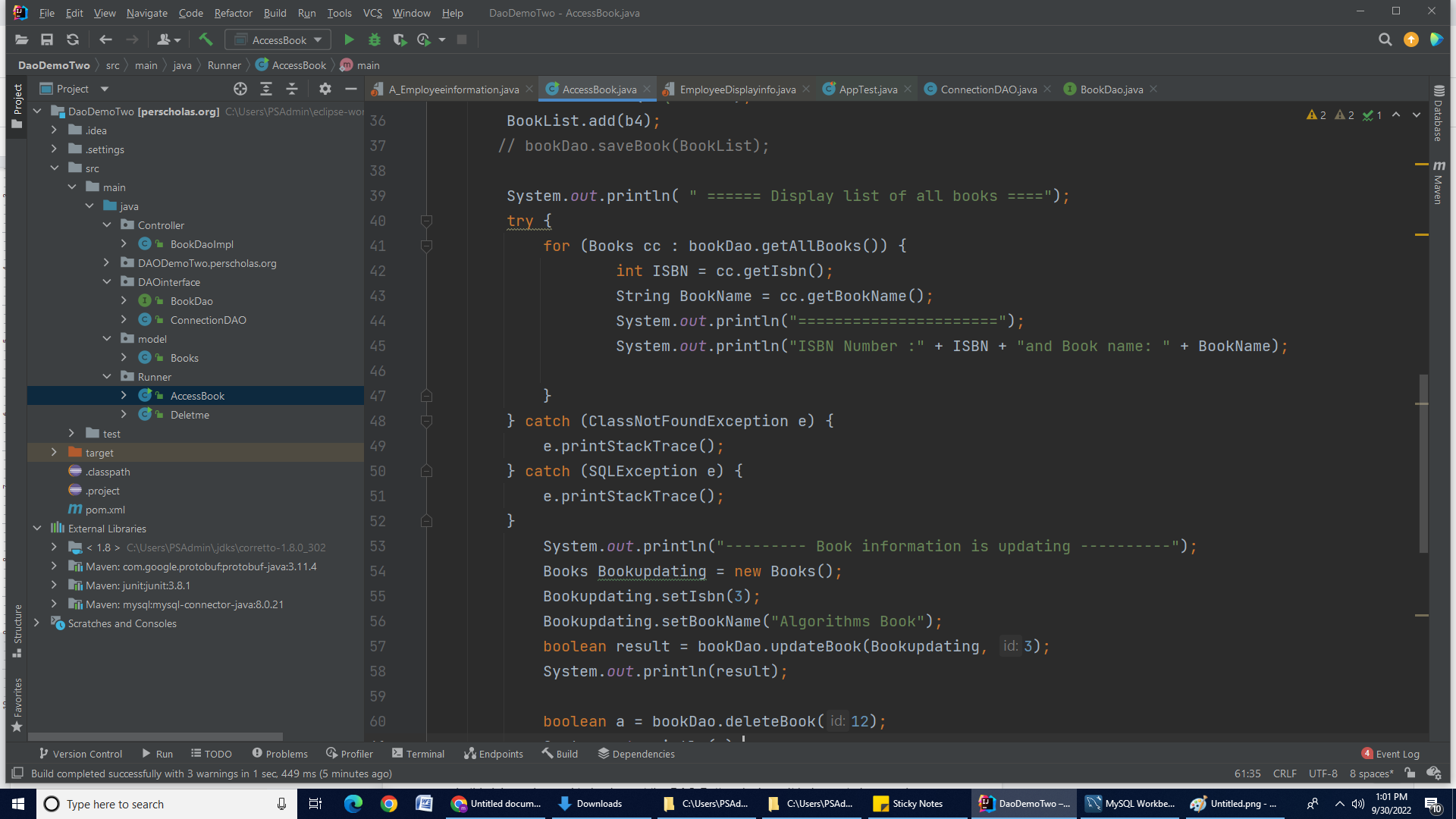
======================

ISBN Number :256and Book name: SQL Book

### Step 6.3: Update Book record

We can invoke/use **updateBook (Books bookObj, int id)** method in **AccessBook** class. Add the below code in **AccessBook** class.

| System.out.println("----Book information is updating -----");  Books Bookupdating = new Books();  Bookupdating.setIsbn(3);  Bookupdating.setBookName("Algorithms Book");  boolean result = bookDao.updateBook(Bookupdating, 3);  System.out.println(result); |
| --- |

****

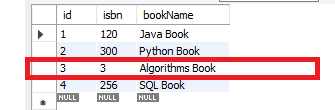
**Output on Console**

--------- Book information is updating ----------

Connected Database Successfully

True

**Result in MySQL:** Id 3 record is updated.

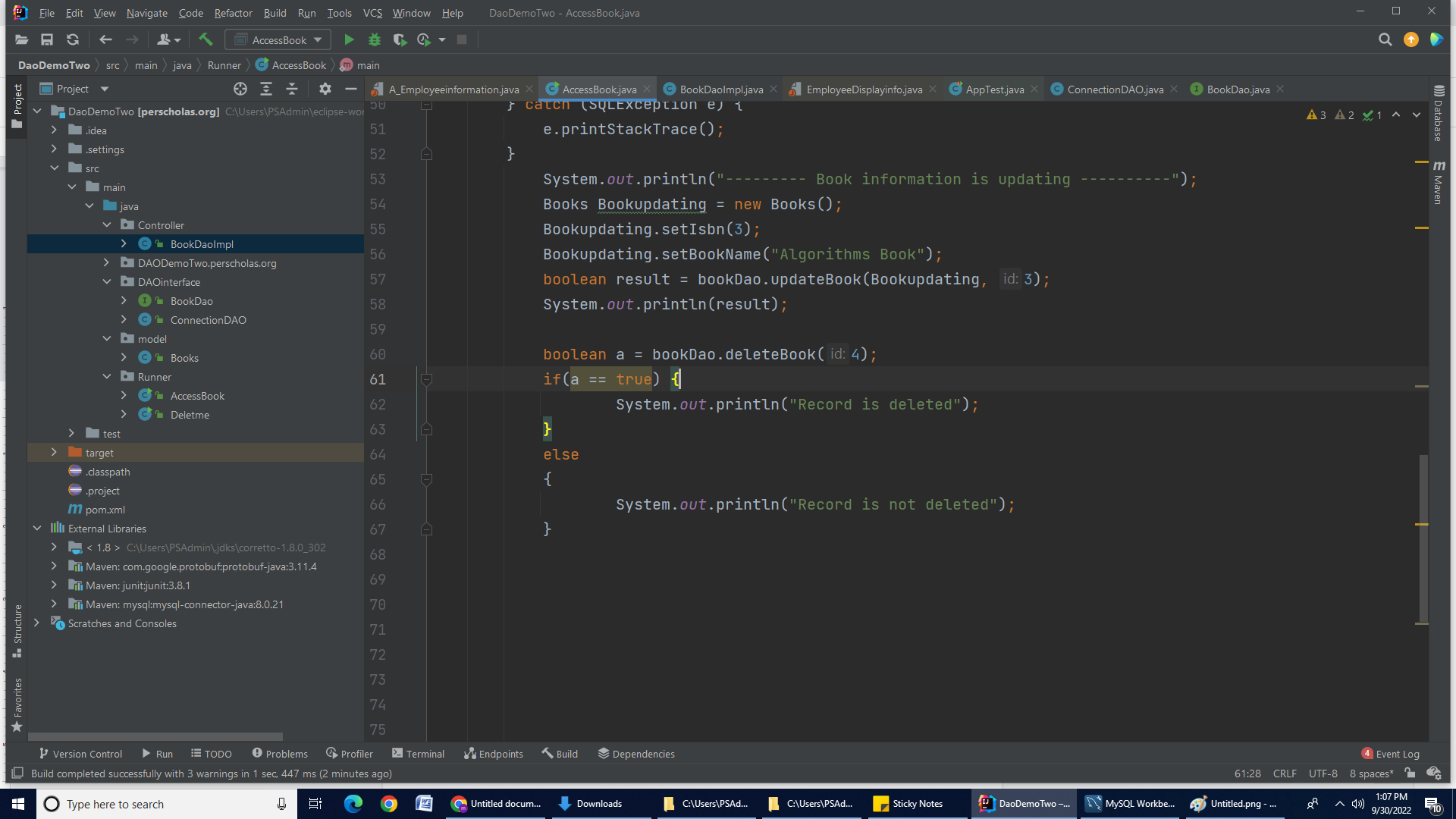


### 

### Step 6.4: Delete Book record by id

We can invoke/use the **deleteBook(int id)** method in **AccessBook** class. We can pass **id** to this method. Add the below code in **AccessBook** class.

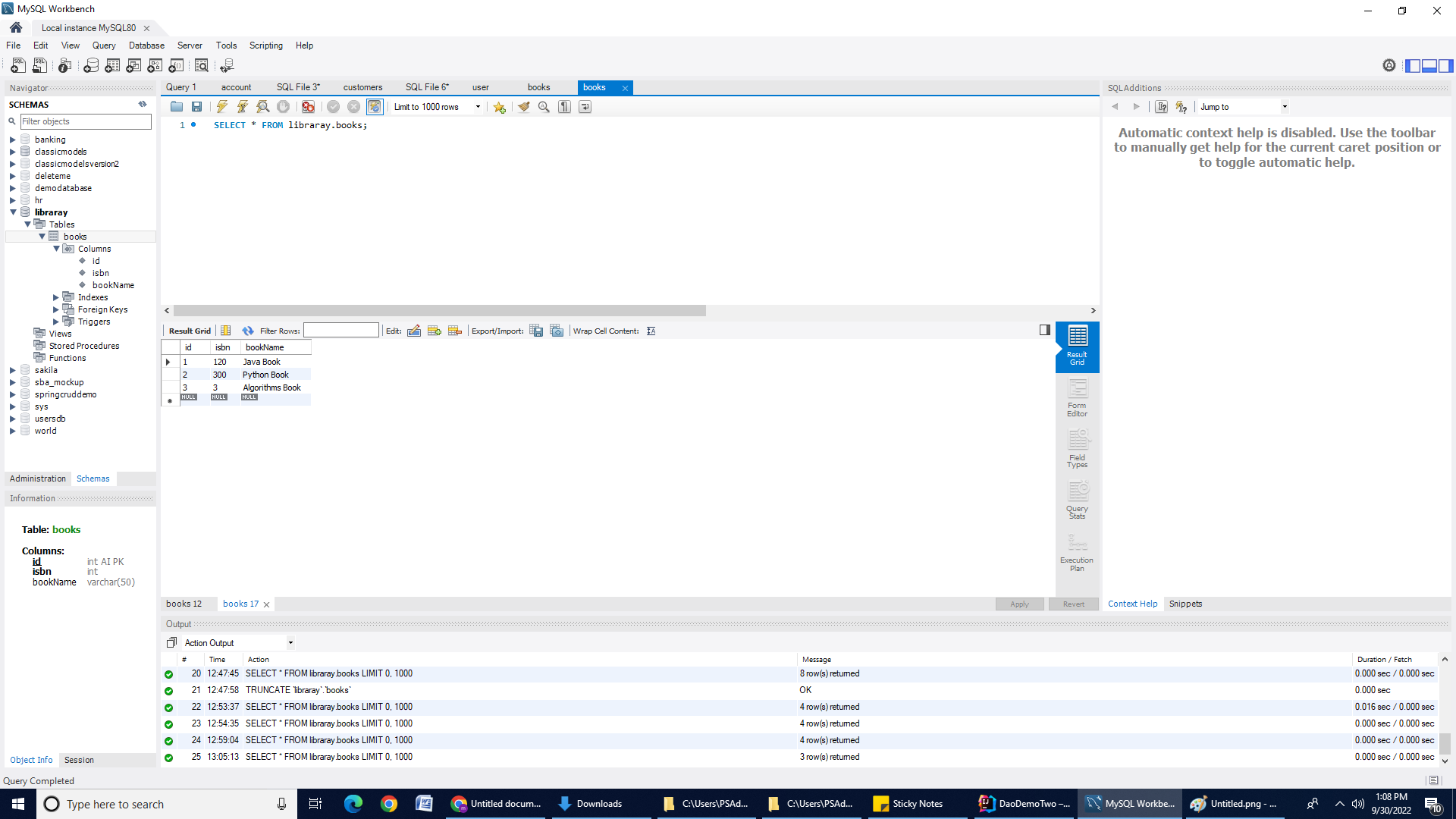
| boolean a = bookDao.deleteBook(4); *// 4 is a id of book*  if(a == true) {  System.out.println("Record is deleted");  }  else  {  System.out.println("Record is not deleted");  } |
| --- |



**Output on Console**

### Connected Database Successfully

### Record is deleted

**Result in MySQL:** You will notice that Record 4 is deleted from the database.

### Conclusion:

In this lab, we learned to implement the DAO Pattern in Java. It helps us to have a clear boundary between our persistence and business layers, promoting loose-coupling.

**Submission Instructions:**

Include the following deliverables in your submission -

* + Submit your source code or screenshot using the Start Assignment button in the top-right corner of the assignment page in Canvas.

**CANVAS STAFF USE ONLY: Canvas Submission Guideline:**

| **Instructions for Canvas Assignment Creation** |
| --- |
| **Assignment Name: GLAB - 305.3.1 - Building Java CRUD application using DAO pattern, OOP concept and JDBC**  **Points:** **100**  **Assignment Group: Module 305 - JDBC, ORM, and Hibernate - (Not Graded)**  **Display Grade As: Completed/Incomplete**  **Do not count this assignment towards the final grade: Checked**  **Submission Types: File Uploads**  **Allowed Attempts: Unlimited**  **Everything else is the default.** |

## 