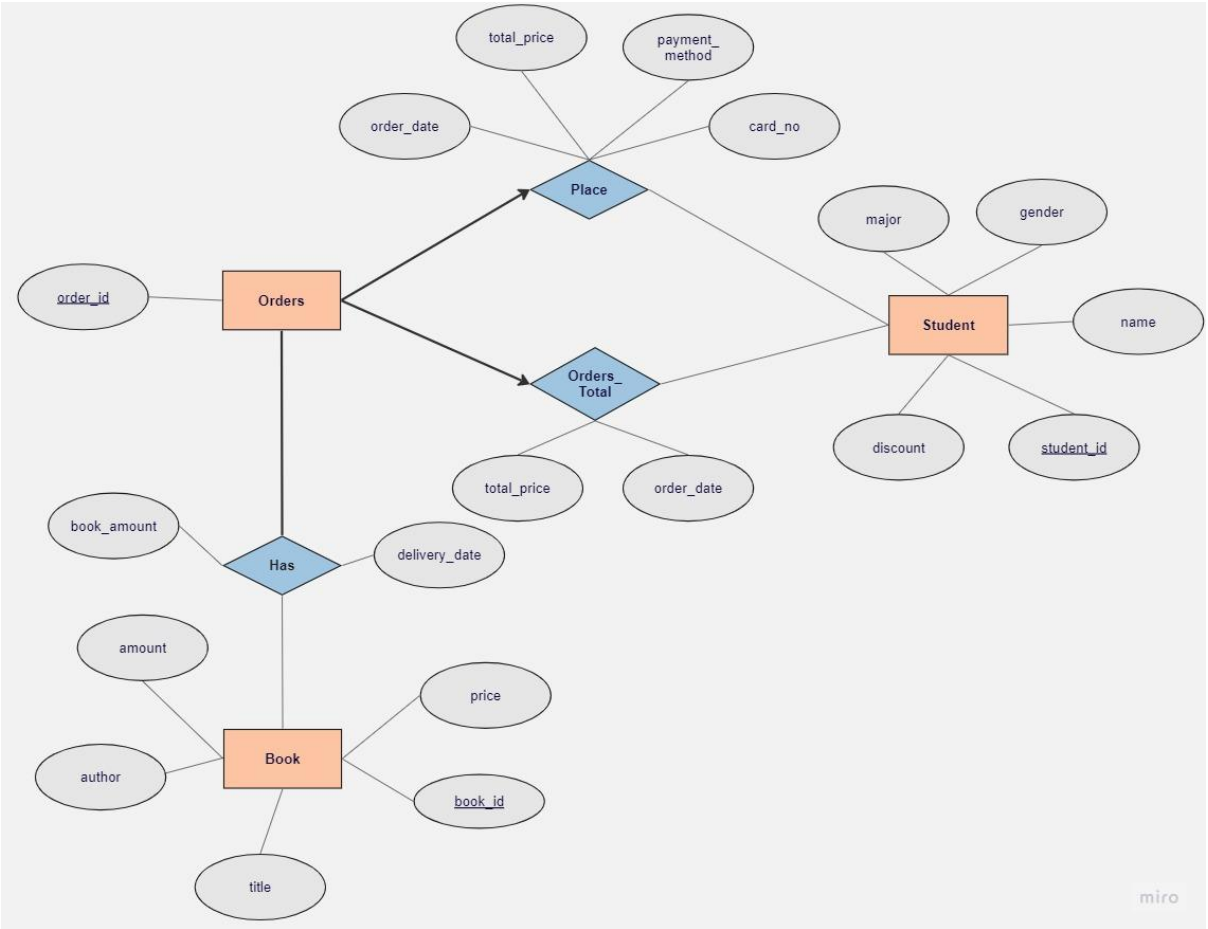
COMP2016:
Database Management

Group Project

Group No: 3
KADESSOVA Ayazhan, 21204276
ONG Jun Kye, 21201749
SHAMMO Shahtiya Khan, 21201633
LIM Jihye, 22508570

ER Diagram



- There are three entity sets: **Student**, **Orders** and **Book**. These entity sets record the students studying at the university, orders made by the university students and books available at the bookshop respectively.
- There are three relational sets: **Place**, **Has**, **Orders_Total.** These relational sets record orders placed by students, book that has been ordered by students and the total order price charged to students respectively.

Table Designs

1. Book Table

```
CREATE TABLE Book (
    book_id INT,
    title VARCHAR(30) NOT NULL,
    author CHAR(30) NOT NULL,
    price DECIMAL(10,2) NOT NULL,
    amount INT NOT NULL,
    PRIMARY KEY (book_id));
```

- The **Book** table is designed to have *book_id*, *title*, *author*, *price* and *amount* for the number of that books in the inventory.
- The primary key for the book table is *book_id* as the *book_id* attribute can easily identify the rest of the attributes in the **Book** table.
- *title*, *author*, *price* and *amount* attributes are set to **not null** so that the book is valid in the sense that each book in the table is valid.

2. Student Table

```
CREATE TABLE Student (
    student_id INT,
    name VARCHAR(30) NOT NULL,
    gender VARCHAR(30) NOT NULL,
    major VARCHAR(30) NOT NULL,
    discount DECIMAL(10,2) NOT NULL,
    PRIMARY KEY (student_id)
);
```

- The **Student** table is designed to have *student_id*, *name* (student's name), *gender*, *major* and *discount*(discount level) attributes for each student.
- The primary key for the student table is *student_id* as the *student_id* attribute can easily identify the rest of the attributes in the **Student** table.
- *gender, major,* and *discount* attributes are set to **not null** so that the student is valid in the sense that each student in the table is valid.

3. Orders Table (has Place relationship)

```
CREATE TABLE Orders (
    order_id INT,
    student_id INT NOT NULL,
    order_date DATE NOT NULL,
    total_price DECIMAL(10,2) NOT NULL,
    payment_method VARCHAR(20) NOT NULL,
    card_no VARCHAR(16),
```

```
    order_delivered VARCHAR(20) DEFAULT 'pending',
    PRIMARY KEY (order_id),
    FOREIGN KEY (student_id) REFERENCES Student(student_id)
);
```

- The **Orders** table is designed to have *order_id, student_id, order_date, total_price, payment_method, card_no, order_delivered* attributes for each order.
- The primary key for the orders table is *order_id* as the *order_id* attribute can easily identify the rest of the attributes in the **Orders** table.
- *student_id* is a foreign key referring to **Student** to make sure only students listed in Students relation should be allowed to order books.
- *order_date, total_price,* and *payment_method* attributes are set to **not null** so that the order is valid, whereas card_no can be **null** considering cases where the payment method is not a card.
- *order_delivered* which means delivery status is 'pending' as default, and only if order is delivered (all books in order are delivered), it will be changed as 'delivered'.

4. Orders_Book Table (has Has relationship)

```
CREATE TABLE Orders_Book (
    order_id INT,
    book_id INT,
    book_amount INT,
    delivery_date DATE NOT NULL,
    PRIMARY KEY (order_id, book_id),
    FOREIGN KEY (book_id) REFERENCES Book(book_id)
);
```

- The **Orders_Book** table(Has Relationship) is designed to have *order_id, book_id, book_amount, delivery_date* attributes for each book order.
- The primary key for the Orders_Book table is *order_id, book_id* from Book table and Orders table.
- *book_id* is a foreign key referring to **Book** to make sure only books listed in the Book table should be allowed to be ordered.
- Total participation constraint can't be captured for orders since the constraint of orders is many and total participation.

5. Orders_Total Table (has Orders_Total relationship)

```
CREATE TABLE Orders_Total (
    order_id INT,
```

```
    student_id INT NOT NULL,
    order_date DATE NOT NULL,
    total_price DECIMAL(10,2) NOT NULL,
    PRIMARY KEY (order_id),
    FOREIGN KEY (student_id) REFERENCES Student(student_id)
);
```

- The **Orders_Total** table is designed to have *order_id, student_id, order_date, total_price* attributes for each book order.
- The primary key for the Orders_Total table is *order_id.*
- *student_id* is a foreign key referring to **Student** to make sure only students listed in Students relation would be allowed to be inserted into **Orders_Total**.
- *student_id, order_date,* and *total_price* attributes are set to **not null** so that the order is valid.
- This table is used to check and update the student's discount level (0.0, 0.1 or 0.2) based on total ordered amount in a year when a new order is inserted into Order by using *student_id, total_price, order_date* attributes. If a student ordered more than 2000 in a year, discount is set to 0.2, if 1000-2000, discount is set to 0.1, else 0.
- The checking and update on student's discount level is automatically executed by trigger.

Constraint on Orders_Book table

```
ALTER TABLE Orders_Book
ADD CONSTRAINT FK_ORDERS_BOOK_ORDER_ID
FOREIGN KEY (order_id) REFERENCES Orders(order_id) ON DELETE CASCADE
INITIALLY DEFERRED DEFERRABLE;
```

- Since order_id references the Orders table, it can't be created until the Orders table exists. So we can't directly use order_id as foreign key in the Orders_Books table, and the constraint FK_ORDERS_BOOKS_ORDER_ID is added to defer integrity checking.
- It is ensured that if a record is deleted from the Orders table, all corresponding records in the Orders_Book table are also deleted by using ON DELETE CASCADE.

Constraint on Orders_Total table

```
ALTER TABLE Orders_Total
ADD CONSTRAINT FK_ORDERS_Total_ORDER_ID
```

```
FOREIGN KEY (order_id) REFERENCES Orders(order_id) INITIALLY DEFERRED
DEFERRABLE;
```

- Since order_id references the Orders table, it can't be created until the Orders table exists. So we can't directly use order_id as foreign key in the Orders_Total table, and the constraint FK_ORDERS_Total_ORDER_ID is added to defer integrity checking.

Source Codes

a. groupX_dbdrop.sql:

```
PROMPT DROP TABLES;

DROP TABLE Book CASCADE CONSTRAINT;
DROP TABLE Student CASCADE CONSTRAINT;
DROP TABLE Orders CASCADE CONSTRAINT;
DROP TABLE Orders_Book CASCADE CONSTRAINT;
DROP TABLE Orders_Total CASCADE CONSTRAINT;

COMMIT;
```

b. groupX_dbinsert.sql

```
CREATE TABLE Book (
    book_id INT,
    title VARCHAR(30) NOT NULL,
    author CHAR(30) NOT NULL,
    price DECIMAL(10,2) NOT NULL,
    amount INT NOT NULL,
    PRIMARY KEY (book_id));

CREATE TABLE Student (
    student_id INT,
    name VARCHAR(30) NOT NULL,
    gender VARCHAR(30) NOT NULL,
    major VARCHAR(30) NOT NULL,
    discount DECIMAL(10,2) NOT NULL,
    PRIMARY KEY (student_id)
);

CREATE TABLE Orders (
    order_id INT,
    student_id INT NOT NULL,
    order_date DATE NOT NULL,
    total_price DECIMAL(10,2) NOT NULL,
    payment_method VARCHAR(20) NOT NULL,
    card_no VARCHAR(16),
    order_delivered VARCHAR(20) DEFAULT 'pending',
    PRIMARY KEY (order_id),
    FOREIGN KEY (student_id) REFERENCES Student(student_id)
);
```

```sql
CREATE TABLE Orders_Total (
    order_id INT,
    student_id INT NOT NULL,
    order_date DATE NOT NULL,
    total_price DECIMAL(10,2) NOT NULL,
    PRIMARY KEY (order_id),
    FOREIGN KEY (student_id) REFERENCES Student(student_id)
);


CREATE TABLE Orders_Book (
    order_id INT,
    book_id INT,
    book_amount INT,
    delivery_date DATE NOT NULL,
    PRIMARY KEY (order_id, book_id),
    FOREIGN KEY (book_id) REFERENCES Book(book_id)
);

COMMIT;

ALTER TABLE Orders_Book
ADD CONSTRAINT FK_ORDERS_BOOK_ORDER_ID
FOREIGN KEY (order_id) REFERENCES Orders(order_id) ON DELETE CASCADE
INITIALLY DEFERRED DEFERRABLE;


COMMIT;

-- dont add on delete cascade because we delete in the trigger
update_student_discount_del
ALTER TABLE Orders_Total
ADD CONSTRAINT FK_ORDERS_Total_ORDER_ID
FOREIGN KEY (order_id) REFERENCES Orders(order_id) INITIALLY DEFERRED
DEFERRABLE;

COMMIT;

PROMPT INSERT Book TABLE;

INSERT INTO Book VALUES (1, 'To Kill a Mockingbird', 'Harper Lee',
180.99, 50);
```

```
INSERT INTO Book VALUES (2, '1984', 'George Orwell', 200.05, 30);
INSERT INTO Book VALUES (3, 'Pride and Prejudice', 'Jane Austen',
220.05, 40);
INSERT INTO Book VALUES (4, 'The Great Gatsby', 'F. Scott Fitzgerald',
290.05, 20);
INSERT INTO Book VALUES (5, 'One Hundred Years of Solitude', 'Gabriel
García Márquez', 300.05, 60);
INSERT INTO Book VALUES (6, 'The Catcher in the Rye', 'J.D. Salinger',
240.99, 35);
INSERT INTO Book VALUES (7, 'The Lord of the Rings', 'J.R.R. Tolkien',
350.99, 25);
INSERT INTO Book VALUES (8, 'Brave New World', 'Aldous Huxley', 180.99,
30);
INSERT INTO Book VALUES (9, 'Animal Farm', 'George Orwell', 150.05,
50);
INSERT INTO Book VALUES (10, 'The Hobbit', 'J.R.R. Tolkien', 170.05,
45);

-- Insert into Student table
PROMPT INSERT Student TABLE;
INSERT INTO Student VALUES (1, 'John Smith', 'Male', 'English', 0.00);
INSERT INTO Student VALUES (2, 'Sarah Johnson', 'Female', 'Biology',
0.00);
INSERT INTO Student VALUES (3, 'David Chen', 'Male', 'Computer
Science', 0.00);
INSERT INTO Student VALUES (4, 'Emily Wong', 'Female', 'History',
0.00);
INSERT INTO Student VALUES (5, 'Michael Kim', 'Male', 'Mathematics',
0.00);
INSERT INTO Student VALUES (6, 'Jessica Lee', 'Female', 'Chemistry',
0.00);
INSERT INTO Student VALUES (7, 'Daniel Rodriguez', 'Male', 'Physics',
0.00);
INSERT INTO Student VALUES (8, 'Avery Taylor', 'Female', 'Psychology',
0.00);
INSERT INTO Student VALUES (9, 'Kevin Patel', 'Male', 'Economics',
0.00);
INSERT INTO Student VALUES (10, 'Sophia Kim', 'Female', 'Sociology',
0.00);
COMMIT;

-- Check if there is credit card when payment option is credit card
CREATE OR REPLACE TRIGGER check_credit_card
```

```
BEFORE INSERT ON Orders
FOR EACH ROW
BEGIN
  -- Check if payment method is credit card and card number is valid
  IF :NEW.payment_method = 'Credit Card' AND ( :NEW.card_no IS NULL )
THEN
    -- Raise an error if the card number is not valid
    RAISE_APPLICATION_ERROR(-20001, 'Invalid credit card number');
  END IF;
END;
/

-- Create a trigger to check if there is enough amount of the book in
the Book table Before we Insert it to Orders_Book table
CREATE OR REPLACE TRIGGER check_book_amount
BEFORE INSERT ON Orders_Book
FOR EACH ROW
DECLARE
  book_amount NUMBER;
BEGIN
  -- Check if there is enough amount of the book in the Book table
  SELECT amount INTO book_amount FROM Book WHERE book_id =
:NEW.book_id;
  IF :NEW.book_amount > book_amount THEN
    -- Raise an error if there is not enough amount of the book
    RAISE_APPLICATION_ERROR(-20001, 'Not enough books');
  END IF;
END;
/

-- update student discount after new order
CREATE OR REPLACE TRIGGER update_student_discount
AFTER INSERT OR UPDATE ON Orders
FOR EACH ROW
DECLARE
  v_total_price DECIMAL(10,2);
  v_total_new DECIMAL(10,2):= 0;

BEGIN
  -- initial new
  DBMS_OUTPUT.PUT_LINE('v_total_new=' || v_total_new);

  SELECT SUM(total_price)
```

```
  INTO v_total_price
  FROM Orders_Total
  WHERE student_id = :NEW.student_id
    AND order_date >= ADD_MONTHS(TRUNC(SYSDATE, 'YEAR'), -12);

-- check if v_total_price is empty or not.
-- if we try to add empty -> v_total_new becaomes empty and the trigger
does not work
-- if v_total_price not empty, we can add
IF v_total_price > 0 THEN
    v_total_new := v_total_price + :NEW.total_price;
  ELSE
    v_total_new := :NEW.total_price;
  END IF;

-- check in terminal
  DBMS_OUTPUT.PUT_LINE('initial new ' || :NEW.total_price);
  DBMS_OUTPUT.PUT_LINE(' after, v_total_new= ' || v_total_new);

  -- set discount
  IF v_total_new > 2000 THEN
    UPDATE Student
    SET discount = 0.20
    WHERE student_id = :NEW.student_id;
  ELSIF v_total_new > 1000 THEN
    UPDATE Student
    SET discount = 0.10
    WHERE student_id = :NEW.student_id;
  ELSE
    UPDATE Student
    SET discount = 0
    WHERE student_id = :NEW.student_id;
  END IF;
END;
/

-- Create a trigger to update the amount of books in the Book table
after we Insert it to Orders_Book table
CREATE OR REPLACE TRIGGER update_amount
AFTER INSERT ON Orders_Book
FOR EACH ROW
DECLARE
  v_book_amount NUMBER;
```

```
BEGIN
  v_book_amount := :NEW.book_amount;

  -- Update book amount
  UPDATE Book
  SET amount = amount - v_book_amount
  WHERE book_id = :NEW.book_id;
END;
/

-- Create a trigger to update the total price of the order after we
delete it from orders_book {when we cancel order}
CREATE OR REPLACE TRIGGER add_book_amount
AFTER DELETE ON Orders_Book
FOR EACH ROW
DECLARE
  v_book_amount INT;
BEGIN
  v_book_amount := :OLD.book_amount;

  -- Add book amount back
  UPDATE Book
  SET amount = amount + v_book_amount
  WHERE book_id = :OLD.book_id;
END;
/

-- update student discount after an order has been deleted
CREATE OR REPLACE TRIGGER update_student_discount_del
AFTER DELETE ON Orders
FOR EACH ROW
DECLARE
  v_total_price DECIMAL(10,2);
  v_total_new DECIMAL(10,2):= 0;

BEGIN

  DBMS_OUTPUT.PUT_LINE('v_total_new=' || v_total_new);

  SELECT SUM(total_price)
  INTO v_total_price
  FROM Orders_Total
  WHERE student_id = :OLD.student_id
```

```
    AND order_date >= ADD_MONTHS(TRUNC(SYSDATE, 'YEAR'), -12);

-- check if v_total_price is empty or not.
-- if we try to use empty -> v_total_new becomes empty and the trigger
does not work
-- if v_total_price not empty, we can use it
  IF v_total_price > 0 THEN
  -- subtract deleted order price
    v_total_new := v_total_price - :OLD.total_price;
  ELSE
  -- if no matches, just 0
    v_total_new := 0;
  END IF;

-- check in terminal
  DBMS_OUTPUT.PUT_LINE('initial total that we are deleting: ' ||
:OLD.total_price);
  DBMS_OUTPUT.PUT_LINE(' after, v_total_new= ' || v_total_new);

  -- set discount
  IF v_total_new > 2000 THEN
    UPDATE Student
    SET discount = 0.20
    WHERE student_id = :OLD.student_id;
  ELSIF v_total_new > 1000 THEN
    UPDATE Student
    SET discount = 0.10
    WHERE student_id = :OLD.student_id;
  ELSE
    UPDATE Student
    SET discount = 0
    WHERE student_id = :OLD.student_id;
  END IF;

-- delete from Orders_Total
  DELETE FROM Orders_Total WHERE order_id = :OLD.order_id;

END;
/


COMMIT;
```

```
SET AUTOCOMMIT ON
```

Triggers Used Description:

1. **check_credit_card**

   - This trigger ensures that a valid credit card number is entered into the Orders table **before insertion** on the Orders table.
   - It will check whether the payment is 'Credit Card'. If it is 'Credit Card', it will check whether the card number is valid.
   - If the card number is null, it will trigger an application error with a message indicating that the credit card number is invalid. Thus, the order will not be placed.

2. **check_book_amount**

   - This trigger ensures that there is enough quantity of books available in the *Book* table **before inserting** a new record into the Orders_Book table.
   - A local variable called *book_amount* is declared with NUMBER type.
   - The *amount* attribute from Book table, showing the stock of the specified book, will be stored it in the *book_amount* attribute, and compares it with *book_amount* (amount of books ordered) from Orders_Book.
   - If the *:NEW.book_amount* exceeds the quantity of the book in the Book table, it will trigger an application error with the message 'Not enough books'.

3. **update_amount Trigger**

   - This trigger updates the *amount* attribute from the Book table (stock amount of specified book) **after insertion** on Orders_Book table.
   - A local variable called *v_book_amount* with NUMBER type is created to store the *book_amount* (amount of books ordered) from Orders_Book table.
   - The *amount* attribute from Book table for specified *book_id* will be updated.

4. **add_book_amount**

   - This trigger updates the amounts of books in the Book table **after deletion** on the Orders_Book table.
   - A local variable called *v_book_amount* with NUMBER type is created to store the *book_amount* (amount of books from deleted order) from Orders_Book table.
   - The *amount* attribute from Book table for specified *book_id* will be updated.

5. **update_student_discount**

   - This trigger updates the student discount **after inserting** a new order has been

in Orders table.
- Local variables called *v_total_price* and *v_total_new* with DECIMAL(10,2) type are created.
- *v_total_price* stores the total price paid by student.
- *v_total_new* stores the sum of *v_total_price* and new order's total price.
- Check *v_total_new* if its bigger than 2000 or 1000. If *v_total_new* is bigger than 2000, then set discount to 0.2. Else if *v_total_new* is bigger than 1000 but smaller than 2000, set discount to 0.1. If *v_total_new* is less than 1000, set discount to 0.

6. **update_student_discount_del**
   - This trigger updates the student discount **after deletion** of a orderfrom Orders table.
   - Local variables called *v_total_price* and *v_total_new* with DECIMAL(10,2) type are created.
   - *v_total_price* stores the total price paid by student.
   - *v_total_new* stores *v_total_price* minus the deleted order's total price.
   - Check *v_total_new* if its bigger than 2000 or 1000. If *v_total_new* is bigger than 2000, then set discount to 0.2. Else if *v_total_new* is bigger than 1000 but smaller than 2000, set discount to 0.1. If *v_total_new* is less than 1000, set discount to 0.

c. BookOrder.java

```
import java.util.*;

public class BookOrder {
    public int bookId;
    public int bookAmount;

    public BookOrder(int bookId, int bookAmount) {
        this.bookId = bookId;
        this.bookAmount = bookAmount;
    }

    public int getBookId() {
        return bookId;
    }

    public int getBookAmount() {
        return bookAmount;
    }
```

```
    // New method to get the amount of a book order based on its ID
    public static int getBAmount(ArrayList<BookOrder> orders, int
bookId) {
        int amount = 0;
        for (BookOrder order : orders) {
            if (order.getBookId() == bookId) {
                amount = order.getBookAmount();
                break;
            }
        }
        return amount;
    }
}
```

d. UniversityBookShop.java

```
import java.awt.GridLayout;
import java.util.*;
import java.awt.TextField;
import java.awt.event.ComponentAdapter;
import java.awt.event.ComponentEvent;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;
import java.util.Scanner;
import java.util.concurrent.TimeUnit;

import javax.swing.*;

import java.util.Properties;

import com.jcraft.jsch.JSch;
import com.jcraft.jsch.JSchException;
import com.jcraft.jsch.Session;

/**
 *
 * This is a University BookShop manager that supports:
```

```
 * (1) 🔍 Search Order by OrderID: This option allows the manager to
search for
 * a
 * specific order by its unique order ID.
 * (2) 🔍👩‍🎓 Search Order by StudentID: This option allows the manager
to
 * search for all orders placed by a specific student, identified by
their
 * student ID.
 * (3) 🔍📚 Update Order for Student: This option allows the manager to
update
 * an existing order for a specific student, such as modifying the
delivery date
 * or
 * adding/removing books.
 * (4) 🛍️ Place an Order: This option allows the student to place a new
order
 * specifying the books and delivery date.
 * Books in one order can have different delivery dates.
 * Delivery date for every book ranges between 3-14 days.
 *
 * A student can place an order if the following conditions are met:
 * § No books in the order are out of stock.
 * § The student does not have any outstanding orders (all books
ordered earlier
 * had been delivered).
 * After an order is confirmed, the total price of the order should be
 * calculated automatically based on the book prices and the current
discount
 * level.
 * § If payment method is credit card, credit card no is required.
 *
 * (5) 🗑️ Cancel an Order: This option allows the manager to cancel an
existing
 * order, provided that no books from the order have been delivered.
 * A student can cancel an order if the following conditions are met:
 * § None of the books in the order has been delivered.
 * § The order was made in the recent 7 days.
 * (6) 📚 Show All Books: This option displays a list of all available
books in
 * the
 * inventory.
```

```
 * (7) 📋 Show All Orders: This option displays a list of all orders
placed in
 * the
 * system, along with their details such as the order ID, student ID,
and
 * delivery date.
 *
 * With these options, the University BookShop manager can effectively
manage
 * the book inventory, process orders, and provide timely and efficient
service
 * to its customers.
 */

public class UniversityBookshop {

    Scanner in = null;
    Connection conn = null;
    // Database Host
    final String databaseHost = "orasrv1.comp.hkbu.edu.hk";
    // Database Port
    final int databasePort = 1521;
    // Database name
    final String database = "pdborcl.orasrv1.comp.hkbu.edu.hk";
    final String proxyHost = "faith.comp.hkbu.edu.hk";
    final int proxyPort = 22;
    final String forwardHost = "localhost";
    int forwardPort;
    Session proxySession = null;
    boolean noException = true;

    // JDBC connecting host
    String jdbcHost;
    // JDBC connecting port
    int jdbcPort;

    String[] options = {
            "- Search Order by OrderID",
            "- Search Order by StudentID", // changed to student emoji
            "- Update Order for Student",
            "- Place an Order",
            "- Cancel an Order",
            "- Show All Books",
```

```
            "- Show All Orders",
            "- Check Discount for StudentID",
            "Exit"
    };

    /**
     * Get YES or NO. Do not change this function.
     *
     * @return boolean
     */
    boolean getYESorNO(String message) {
        JPanel panel = new JPanel();
        panel.add(new JLabel(message));
        JOptionPane pane = new JOptionPane(panel,
JOptionPane.QUESTION_MESSAGE, JOptionPane.YES_NO_OPTION);
        JDialog dialog = pane.createDialog(null, "Question");
        dialog.setVisible(true);
        boolean result = JOptionPane.YES_OPTION == (int)
pane.getValue();
        dialog.dispose();
        return result;
    }

    /**
     * Get username & password. Do not change this function.
     *
     * @return username & password
     */
    String[] getUsernamePassword(String title) {
        JPanel panel = new JPanel();
        final TextField usernameField = new TextField();
        final JPasswordField passwordField = new JPasswordField();
        panel.setLayout(new GridLayout(2, 2));
        panel.add(new JLabel("Username"));
        panel.add(usernameField);
        panel.add(new JLabel("Password"));
        panel.add(passwordField);
        JOptionPane pane = new JOptionPane(panel,
JOptionPane.QUESTION_MESSAGE, JOptionPane.OK_CANCEL_OPTION) {
            public static final long serialVersionUID = 1L;

            @Override
            public void selectInitialValue() {
```

```
                usernameField.requestFocusInWindow();
            }
        };
        JDialog dialog = pane.createDialog(null, title);
        dialog.setVisible(true);
        dialog.dispose();
        return new String[] { usernameField.getText(), new
String(passwordField.getPassword()) };
    }

    /*
     * Login the proxy. Do not change this function.
     *
     * @return boolean
     */
    public boolean loginProxy() {
        if (getYESorNO("Using ssh tunnel or not?")) { // if using ssh
tunnel
            String[] namePwd = getUsernamePassword("Login cs lab
computer");
            String sshUser = namePwd[0];
            String sshPwd = namePwd[1];
            try {
                proxySession = new JSch().getSession(sshUser,
proxyHost, proxyPort);
                proxySession.setPassword(sshPwd);
                Properties config = new Properties();
                config.put("StrictHostKeyChecking", "no");
                proxySession.setConfig(config);
                proxySession.connect();
                proxySession.setPortForwardingL(forwardHost, 0,
databaseHost, databasePort);
                forwardPort =
Integer.parseInt(proxySession.getPortForwardingL()[0].split(":")[0]);
            } catch (JSchException e) {
                e.printStackTrace();
                return false;
            }
            jdbcHost = forwardHost;
            jdbcPort = forwardPort;
        } else {
            jdbcHost = databaseHost;
            jdbcPort = databasePort;
```

```
        }
        return true;
    }

    /**
     * Login the oracle system. Change for your own credentials.
     *
     * @return boolean
     */
    public boolean loginDB() {
        String username = "f1204276";// Replace to your username
        String password = "f1204276";// Replace to your password

        /* Do not change the code below */
        if (username.equalsIgnoreCase("e1234567") ||
password.equalsIgnoreCase("e1234567")) {
            String[] namePwd = getUsernamePassword("Login sqlplus");
            username = namePwd[0];
            password = namePwd[1];
        }
        String URL = "jdbc:oracle:thin:@" + jdbcHost + ":" + jdbcPort +
"/" + database;

        try {
            System.out.println("Logging " + URL + " ...");
            conn = DriverManager.getConnection(URL, username,
password);
            return true;
        } catch (SQLException e) {
            e.printStackTrace();
            return false;
        }
    }

    /**
     * Show the options. If you want to add one more option, put into
the
     * options array above.
     */
    public void showOptions() {
        System.out.println("Please choose following option:");
        for (int i = 0; i < options.length; ++i) {
            System.out.println("(" + (i + 1) + ") " + options[i]);
```

```
        }
    }

    /**
     * Run the manager
     */
    public void run() {

        // update every time program starts
        update();
        while (noException) {
            showOptions();
            String line = in.nextLine();
            if (line.equalsIgnoreCase("exit"))
                return;
            int choice = -1;
            try {
                choice = Integer.parseInt(line);
            } catch (Exception e) {
                System.out.println("This option is not available");
                continue;
            }
            if (!(choice >= 1 && choice <= options.length)) {
                System.out.println("This option is not available");
                continue;
            }
            if (choice == 1) {
                orderSearchbyID();
            } else if (choice == 2) {
                orderSearchforStudent("Search");
            } else if (choice == 3) {
                orderSearchforStudent("Update");
            } else if (choice == 4) {
                placeOrder();
            } else if (choice == 5) {
                cancelOrder();
            } else if (choice == 6) {
                displayBooks();
            } else if (choice == 7) {
                listAllOrders();
            } else if (choice == 8) {
                discountSearchforStudent();
            } else if (options[choice - 1].equalsIgnoreCase("exit")) {
```

```
                break;
            }
        }
    }
    // ============================Main
    // functions==================================

    /*
     * Search order by order_id driver
     */
    public void orderSearchbyID() {
        System.out.println("Please input order_id or -1 for exit:");

        int order_id = Integer.parseInt(in.nextLine());

        if (order_id == -1)
            return;

        // Call main function to find order.
        orderSearchbyID(order_id);
    }

    /**
     * Given order_id, find All info about the order.
     */
    public void orderSearchbyID(int order_id) {

        try {
            /**
             * Create the statement and sql
             */
            Statement stm = conn.createStatement();

            String sql = "SELECT * FROM Orders WHERE order_id = " +
order_id;

            //// System.out.println(sql);

            ResultSet rs = stm.executeQuery(sql);

            boolean exists = rs.next();
            if (!exists) {
                // if order does not exist, return
```

```
                System.out.println("No such order");
                return;
            }

            String[] heads = { "order_id", "student_id",
"payment_method", "order_date", "total_price",
                    "card_no", "order_delivered" };

            // if order exist, give order's information
            while (exists) {
                for (int i = 0; i < 7; i++) {
                    String result = "";
                    switch (heads[i]) {
                        // order_id and student_id format is integer
                        case "order_id":
                        case "student_id":
                            result =
Integer.toString(rs.getInt(heads[i]));
                            break;
                        // order_date format is DATE
                        case "order_date":
                            result = rs.getDate(heads[i]).toString();
                            break;
                        // total_price format is double
                        case "total_price":
                            result = String.format("%.2f",
rs.getBigDecimal(heads[i]));
                            break;
                        // payment_method and order_delivered format is
String
                        case "payment_method":
                        case "order_delivered":
                            result = rs.getString(heads[i]);
                            break;
                        // card_no format can be null
                        case "card_no":
                            // if it is null, return N/A
                            if (rs.getString(heads[i]) == null) {
                                result = "N/A";
                            } else {
                                // else, return String format
                                result = rs.getString(heads[i]);
                            }
```

```
                            break;
                    }
                    System.out.println("   " + heads[i] + " : " +
result);
                }
                exists = rs.next();
            }

            // Give all information of the books in order table
            System.out.println("\n===============Books List:
==================\n");

            // Call function to view all books for particular order_id
            displayBooksInOrder(order_id);
            rs.close();
            stm.close();
        } catch (SQLException e) {
            e.printStackTrace();
            // noException = false;
        }

    }

    /**
     * Show all Books in the Order.
     */
    public void displayBooksInOrder(int order_id) {

        try {
            Statement stm = conn.createStatement();

            String sql = "SELECT * FROM Orders_Book WHERE order_id = "
+ order_id;

            //// System.out.println(sql);

            ResultSet rs = stm.executeQuery(sql);

            boolean exists = rs.next();

            // if order_id does not exist, return
            if (!exists) {
                System.out.println("No such order");
```

```
                return;
            }

            String[] heads = { "order_id", "book_id", "book_amount",
"delivery_date" };

            // if order_id exists, show books in orders_book table for
specific order_id
            // e.g., if order_id: 2001 has 2 different books, shows the
2 different books.
            while (exists) {
                System.out.println("================Book Info:
====================");
                for (int i = 0; i < 4; i++) {
                    String result = "";
                    switch (heads[i]) {
                        case "delivery_date":
                            result = rs.getDate(heads[i]).toString();
                            break;
                        default:
                            result =
Integer.toString(rs.getInt(heads[i]));
                            break;
                    }
                    System.out.println("  " + heads[i] + " : " +
result);
                }

                exists = rs.next();
            }
            rs.close();
            stm.close();

        } catch (SQLException e) {
            e.printStackTrace();
            // noException = false;
        }
    }

    /*
     * Given student_id, find all orders for the student driver.
     */
    public void orderSearchforStudent(String choice) {
```

```
        int student_id = askForStudentId();

        if (student_id == -1) {
            System.out.println("No valid student ID was entered.
Exiting the order search process.");
            return;
        }
        // call main function
        orderSearchbyStudentID(student_id, choice);

    }

    /*
     * Given student_id, find student's discount rate.
     */
    public void discountSearchforStudent() {
        // Asks for student_id
        int student_id = askForStudentId();

        if (student_id == -1) {
            System.out.println("No valid student ID was entered.
Exiting the order search process.");
            return;
        }

        // Prints student's discount rate
        getDiscount(student_id);

    }

    /*
     * Given student_id, find all orders for the student.
     */
    public void orderSearchbyStudentID(int student_id, String choice) {
        try {
            Statement stm = conn.createStatement();
            String sql = "SELECT order_id FROM Orders WHERE Student_id
=" + student_id;
            ResultSet rs = stm.executeQuery(sql);

            boolean exists = rs.next();
            // if order_id does not exist, return
            if (!exists) {
```

```
                System.out.println("No such order");
                return;
            }

            System.out.println("Lets us " + choice + " the order(s)
now.");

            // if exists
            while (exists) {

                switch (choice) {
                    // search for the order
                    case "Search":
                        orderSearchbyID(rs.getInt(1));
                        break;
                    // from buyers point of view, if book is delivered,
can update manually
                    case "Update":
                        updateOrder(rs.getInt(1));
                        break;
                    default:
                        break;
                }
                // orderSearchbyID(rs.getInt(1));

System.out.println("===========================================");
                exists = rs.next();

            }
            rs.close();
            stm.close();
        } catch (SQLException e1) {
            e1.printStackTrace();
            // noException = false;
        }
    }

    /*
     * Given student_id, find all orders for the student that are not
delivered.
     */
    public boolean outstandingOrderSearchbyStudentID(int student_id) {
        try {
```

```
            Statement stm = conn.createStatement();
            String sql = "SELECT order_id FROM Orders WHERE student_id
= " + student_id
                    + " AND order_delivered = 'pending'";

            ResultSet rs = stm.executeQuery(sql);

            boolean exists = rs.next();
            // if order does not exist, return
            if (!exists) {
                System.out.println("No such order");
                return false;
            }

            // if order exist
            while (exists) {
                // show outstanding orders
                orderSearchbyID(rs.getInt(1));

System.out.println("=============================================");
                exists = rs.next();

            }
            rs.close();
            stm.close();

            return true;
        } catch (SQLException e1) {
            e1.printStackTrace();
            // noException = false;
            return false;

        }
    }

    /**
     * List all Orders in the database.
     */
    public void listAllOrders() {
        System.out.println("All orders in the database now:");
        try {
            Statement stm = conn.createStatement();
            String sql = "SELECT order_id FROM Orders";
```

```
            // //System.out.println(sql);

            ResultSet rs = stm.executeQuery(sql);

            while (rs.next()) {
                // call function to find the order based on order_id
                orderSearchbyID(rs.getInt(1));

System.out.println("=============================================");

            }
            rs.close();
            stm.close();
        } catch (SQLException e) {
            e.printStackTrace();
            // noException = false;
        }
    }

    /**
     * Show all Books in the database.
     */
    public void displayBooks() {

        try {
            /**
             * Create the statement and sql
             */
            Statement stm = conn.createStatement();

            String sql = "SELECT book_id FROM Book";

            //// System.out.println(sql);

            ResultSet rs = stm.executeQuery(sql);
            // show the books one by one
            while (rs.next()) {

                diplayBook(rs.getInt(1));

System.out.println("📚=============================================📚");
                //
            }
```

```
            rs.close();
            stm.close();
        } catch (SQLException e) {
            e.printStackTrace();
            // noException = false;
        }
    }

    /*
     * Given book_id, display all info about the book.
     */
    public void diplayBook(int book_id) {
        try {
            Statement stm = conn.createStatement();

            String sql = "SELECT * FROM Book WHERE book_id = " +
book_id;

            ResultSet rs = stm.executeQuery(sql);

            while (rs.next()) {

                String[] heads = { "book_id", "title", "author",
"price", "amount" };

                for (int i = 0; i < 5; ++i) {
                    try {
                        // Print the relevant data
                        System.out.println(heads[i] + " : " +
rs.getString(i + 1));
                    } catch (SQLException e) {
                        e.printStackTrace();
                    }
                }
            }
            rs.close();
            stm.close();
        } catch (SQLException e) {
            e.printStackTrace();
            // noException = false;

        }
    }
```

```
    /////////////////////////// Helper Functions
//////////////////////////
    /*
     * Ask to enter a student ID.
     */
    public int askForStudentId() {
        while (true) {
            // prompt the user for a student ID
            System.out.println("Please enter a student ID:");

            String line = in.nextLine();
            int student_id = Integer.parseInt(line);

            // check if the student ID exists in the database
            if (checkStudentId(student_id)) {
                System.out.println("Student ID " + student_id + "
exists in the database.");
                return student_id; // exit the method and return the
valid student ID
            } else {
                // if student ID does not exist
                System.out.println("Student ID " + student_id + " does
not exist in the database.");

                // prompt the user to enter a new student ID or exit
the program
                System.out.println("Press 'N' to escape or press any
other key to enter a new student ID");
                line = in.nextLine();

                if (line.equalsIgnoreCase("N")) {
                    // exit the method without returning a valid
student ID
                    return -1; // or any other invalid value to
indicate no valid student ID was entered
                }
            }
        }

    }

    /*
```

```
     * Ask to enter a student ID.
     */
    public int askForOrderId() {
        while (true) {
            // prompt the user for a student ID
            System.out.println("Please enter a Order ID:");

            String line = in.nextLine();
            int order_id = Integer.parseInt(line);

            // check if the order ID exists in the database
            if (checkOrder(order_id)) {
                System.out.println("Order ID " + order_id + " exists in
the database.");
                return order_id; // exit the method and return the
valid student ID
            } else {
                // If order ID does not exist
                System.out.println("Order ID " + order_id + " does not
exist in the database.");

                // prompt the user to enter a new student ID or exit
the program
                System.out.println("Press 'N' to escape or press any
other key to enter a new order ID");
                line = in.nextLine();

                if (line.equalsIgnoreCase("N")) {
                    // exit the method without returning a valid order
ID
                    return -1; // or any other invalid value to
indicate no valid order ID was entered
                }
            }
        }

    }

    /*
     * Check if the student ID exists in the database.
     */
    public boolean checkStudentId(int student_id) {
        try {
```

```
            Statement stm = conn.createStatement();

            // check if student_id exists
            String sql = "SELECT * FROM Student WHERE Student_id = " +
student_id;

            //// System.out.println(sql);

            ResultSet result = stm.executeQuery(sql);

            boolean exists = result.next();

            if (!exists) { // if Student ID does not exist
                System.out.println("No such Student ID exists in the
database.");
            }

            stm.close();
            result.close();

            return exists;
        } catch (SQLException e) {
            // If there is any exception, return false
            e.printStackTrace();
            return false;
        }
    }

    /*
     * Check if all the books in the order are delivered
     * True - no pending orders
     * False - there are pending orders
     */
    public boolean allDelivered(int order_id) {
        try {
            Statement stm = conn.createStatement();

            // check if order_id exists and the delivery date has not
been passed for some
            // books
            String sql = "SELECT * FROM Orders_Book WHERE delivery_date
> SYSDATE AND order_id = " + order_id;
```

```
            // System.out.println(sql);

            ResultSet result = stm.executeQuery(sql);

            boolean exists = result.next();

            if (exists) { // If orders have still not been delivered
                System.out.println("There are still pending orders.");
            }

            stm.close();
            result.close();

            return !exists;
        } catch (SQLException e) {
            e.printStackTrace();
            return false;
        }
    }

    /*
     * Check if all the books in the order are NOT delivered (for
cancel order)
     * True - there are delivered orders
     * False - not orders are delivered
     */
    public boolean allNOTDelivered(int order_id) {
        try {
            Statement stm = conn.createStatement();

            // check if order_id exists and the delivery date has
already passed for some
            // books
            String sql = "SELECT * FROM Orders_Book WHERE delivery_date
<= SYSDATE AND order_id = " + order_id;

            //// System.out.println(sql);

            ResultSet result = stm.executeQuery(sql);

            boolean exists = result.next();

            if (exists) { // If some books have been delivered
```

```
                System.out.println("There are delivered orders...");
            }

            stm.close();
            result.close();

            return !exists;
        } catch (SQLException e) {
            e.printStackTrace();
            return false;
        }
    }

    /*
     * Check if all the books in the order are NOT delivered (for
cancel order)
     * True - there are delivered orders
     * False - not orders are delivered
     */
    public boolean orderDelivered(int order_id) {
        try {
            Statement stm = conn.createStatement();

            // check if order_id exists and the delivery date has
already passed for some
            // books
            String sql = "SELECT * FROM Orders WHERE order_id = " +
order_id + " AND order_delivered = 'delivered'";

            //// System.out.println(sql);

            ResultSet result = stm.executeQuery(sql);

            boolean exists = result.next();

            if (exists) { // If some books have been delivered
                System.out.println("The order is delivered...");
            }

            stm.close();
            result.close();

            return exists;
```

```
        } catch (SQLException e) {
            e.printStackTrace();
            return false;
        }
    }

    /*
     * Update order status to delivered if all delivered to delivered
     */
    public void updateOrder(int order_id) {
        try {
            Statement stm = conn.createStatement();

            if (allDelivered(order_id)) { // If all the books have been
delivered
                System.out.println("All books in order " + order_id + "
have been delivered.");
            } else { // If not all books have been delivered, return
                System.out.println("There are still pending orders.");
                return;
            }

            // Order is updated to 'delivered' if all books have been
delivered
            String sql = "UPDATE Orders SET order_delivered =
'delivered' WHERE order_id = " + order_id;

            //// System.out.println(sql);

            stm.executeUpdate(sql);

            stm.close();

            System.out.println("succeed to update order " + order_id);

        } catch (SQLException e) {
            e.printStackTrace();
            System.out.println("Could not update order " + order_id);
            // noException = false;
        }
    }

    /*
```

```
     * update all orders every time we run the program
     */
    public void update() {

        System.out.println("All orders in the database now:");
        try {
            Statement stm = conn.createStatement();
            String sql = "SELECT order_id FROM Orders";
            //// System.out.println(sql);

            ResultSet rs = stm.executeQuery(sql);

            while (rs.next()) {
                // update information about all the orders, i.e. check
whether they have been
                // delivered each time after the program is run
                updateOrder(rs.getInt(1));

System.out.println("==========================================");

            }
            rs.close();
            stm.close();
        } catch (SQLException e) {
            e.printStackTrace();
            // noException = false;
        }

    }

    /**
     * Insert data into database
     *
     * @return
     */
    public void placeOrder() {
        /**
         * A sample input is:
         */
        // Call function to get student ID
        int student_id = askForStudentId();

        if (student_id == -1) {
```

```
            // If input is -1, return
            System.out.println("No valid student ID was entered.
Exiting the order placement process.");
            return;
        }

        // check if have outstanding order
        if (outstandingOrderSearchbyStudentID(student_id)) {
            // Return if they have outstanding orders
            System.out.println("You have outstanding order. Please
wait.");
            return;
        } else { // Allow them to order if there is no outstanding
order
            System.out.println("You don't have outstanding order. You
can place order now.");
        }

        // int student_id = Integer.parseInt(line);

        int order_id = 0;
        Random rand = new Random();

        System.out.println("Assigning Order ID...");
        do {
            order_id = rand.nextInt(9001) + 1000; // Generate a random
order ID between 1000 and 10000
            System.out.println("Generated order ID: " + order_id);
        } while (checkOrder(order_id)); // Keep looping while the order
ID is already taken

        System.out.println("Order ID: " + order_id);

        System.out.println("👋 Welcome to our bookshop! Here are our
Books: ");

        // display all the books in the Books table, with their title,
quantity, amount
        // and price
        displayBooks();

        // Prompt the user to enter the number of different books to
order
```

```
        System.out.print("How many different books would you like to
order? Max is 10");

        int numBooks = 0;

        while (true) {
            System.out.print("Enter the number of books (1-10, or -1 to
exit): ");
            numBooks = Integer.parseInt(in.nextLine());

            if (numBooks == -1) {
                break;
            } else if (numBooks < 1 || numBooks > 10) {
                System.out.println("Invalid input. Please enter a
number between 1 and 10, or -1 to exit.");
            } else {
                break;
            }
        }

        // To calculate the total price of all the books ordered
        double total_price = 0;

        // Create an ArrayList to record successful book orders
        ArrayList<BookOrder> orders = new ArrayList<>();
        HashSet<Integer> addedBookIds = new HashSet<>();

        // Loop through the number of different books and prompt the
user to enter the
        // book ID and amount for each book
        for (int i = 1; i <= numBooks; i++) {
            System.out.println("Enter information for book " + i +
"....");

            // Ask for the bookId you want to order
            int bookId = askBookId();

            if (bookId == -1) { // If the input is -1, return
                System.out.println("No valid book ID was entered.
Exiting the order placement process.");
                return;
            }
```

```
            // Check if the book ID is already in the order
            if (addedBookIds.contains(bookId)) { //
                // If the bookId is already in the order, ask the user
whether they want to
                // change the amount
                System.out.println("This book is already in your order.
Do you want to change the amount? (y/n)");

                String answer = in.nextLine();
                if (answer.equalsIgnoreCase("y")) {
                    i--; // Decrease the number of books ordered

                    // If y is pressed, record the original amount of
books
                    int oldAmount = 0;
                    for (int j = 0; j < orders.size(); j++) {
                        if (orders.get(j).getBookId() == bookId) {
                            oldAmount = orders.get(j).getBookAmount();
                            break;
                        }
                    }
                    // Get the updated number of books
                    System.out.print("New book amount: ");

                    int newAmount = Integer.parseInt(in.nextLine());
                    // in.nextLine(); // consume the remaining newline
character

                    // Check if there is enough stock of the book
                    if (newAmount > getAmount(bookId)) {
                        System.out.println("We don't have enough stock
of this book. Sorry!");
                        continue;
                    } else {
                        // If there are enough books, let them order
and update the amount of the books
                        // in the order
                        for (BookOrder order : orders) {
                            if (order.getBookId() == bookId) {
                                order.bookAmount = newAmount;
                                break;
                            }
                        }
```

```
                        // Get the price of the book by ID from the
database
                        double book_price = getPriceByID(bookId);

                        // Calculate the total price difference for
this book based on the changed
                        // amount
                        double book_total_price_diff = (newAmount *
book_price) - (oldAmount * book_price);

                        // Add the total price difference for this book
to the overall total price
                        total_price += book_total_price_diff;

                    }
                } else {
                    continue;
                }
            } else {

                // Check if there is enough stock of the book
                int stock = getAmount(bookId);

                // Prompt the user to enter the book amount
                System.out.print("Enter Book amount: " +
"\nAvailable: " + stock + ": )");
                int bookAmount = Integer.parseInt(in.nextLine());

                if (stock == 0) { // If there are no more books in
stock
                    System.out.println("We are out of stock for this
book. Sorry!:");
                    return;
                } else if (bookAmount > stock) {
                    // If the ordered amount is greater than the amount
in stock, all books in stock
                    // are given to the user
                    System.out.println("We don't have enough stock of
this book. Sorry! We only have " + stock
                            + " left. We will add them all...");

                    bookAmount = stock;
```

```
                }

                // Record the successful book order
                BookOrder order = new BookOrder(bookId, bookAmount);
                orders.add(order);
                addedBookIds.add(bookId); // add the book ID to the set
of added book IDs

                // Get the price of the book by ID from the database
                double book_price = getPriceByID(bookId);

                // Calculate the total price for this book
                double book_total_price = bookAmount * book_price;

                // Add the total price for this book to the overall
total price
                total_price += book_total_price;

            }

        }

        /* if any books are going to be ordered */

        if (orders.size() > 0)

        {
            // Find the discount and update total price accordingly
            double discount = getDiscount(student_id);
            total_price = total_price * (1 - discount);

            System.out.println("Your total is... " + total_price);
            // Ask if the user wants to pay
            System.out.println("Do you want to proceed with payment?
(Y/N)");
            String answer = in.nextLine();

            // Check the answer
            if (answer.equalsIgnoreCase("Y")) {
                boolean paymentSuccess = false;
                while (!paymentSuccess) { // Loop until payment has
been completed
                    // Prompt
```

```
                    String[] payment_result = getPaymentInfo();
                    String payment_method = payment_result[0];
                    String card_no = payment_result[1];

                    try {
                        // Insert to Orders
                        String insertResult = insertOrder(order_id,
student_id, total_price, payment_method, card_no);
                        // Insert to Orders_Total - helper table
                        String insertResult2 = insertOrder2(order_id,
student_id, total_price);

                        if (insertResult.equals("success") &&
insertResult2.equals("success")) { // If payment is

// successful & totall

// added successfully
                            System.out.println("Payment successful!");
                            paymentSuccess = true;
                            for (BookOrder order : orders) {
                                try { // Update order if payment is
successful
                                    InsertBook(order_id,
order.getBookId(), order.getBookAmount());
                                } catch (SQLException e) {
                                    System.out.println("Error inserting
book order details: " + e.getMessage());
                                }
                            }
                        } else if (insertResult.equals("error") ||
insertResult2.equals("error")) { // Allow them to try

// again
                            // if there is an error in
                            // payment
                            System.out.println("Invalid. Do you want to
try again? (Y/N)");
                            String response = in.nextLine();
                            if (!response.equalsIgnoreCase("Y")) { //
If input is not 'Y', return
                                return;
                            }
```

```
                        }

                    } catch (SQLException e) {
                        System.out.println("Error inserting order
details: " + e.getMessage());
                    }
                }

            } else {
                // If payment is not successfully made
                System.out.println("Payment cancelled. Order not
placed.");
            }

        } else { // If the order size is zero or less
            System.out.println("No books were added to the order.
Exiting the order placement process.");
        }
        // return;

    }

    /* Insert book to the database */
    public void InsertBook(int order_id, int bookId, int bookAmount)
throws SQLException {

        try {
            Statement stm = conn.createStatement();

            int randomInterval = 3 + (int) (Math.random() * (14 - 3 +
1)); // Generate a random integer between 3 and 14

            // Update the ordered_books table
            // Using the randomInterval, assign a random delivery date
between 3-14 days
            // since the order is made
            String sql = "INSERT INTO Orders_Book VALUES (" + order_id
+ "," + bookId + "," + bookAmount + ","
                    + "SYSDATE + INTERVAL '" + randomInterval + "'
DAY)";

            // String sql = "INSERT INTO Orders_Book VALUES (" +
order_id + "," + bookId +
```

```
            // "," + bookAmount + ","
            // + "'23-MAR-2023')";

            //// System.out.println(sql);

            stm.executeUpdate(sql);

            stm.close();

            System.out.println("succeed to insert Book" + bookId);

        } catch (SQLException e) {
            e.printStackTrace();
            System.out.println("fail to Insert Book " + bookId);
            // noException = false;
            // return "error";
        }
    }

    /*
     * Insert Order
     */
    public String insertOrder(int order_id, int student_id, double
total_price, String payment_method, String card_no)
            throws SQLException {

        try {
            Statement stm = conn.createStatement();

            String sql = "INSERT INTO Orders (order_id, student_id,
order_date, total_price, payment_method, card_no) "
                    +
                    "VALUES (" + order_id + ", " + student_id + ",
SYSDATE, " + total_price + ", '" + payment_method
                    + "', '" + card_no + "')";

            stm.executeUpdate(sql);

            stm.close();

            System.out.println("succeed to insert Order  " + order_id);

            // If order is added return success
```

```
            return "success";

        } catch (SQLException e) {
            e.printStackTrace();
            System.out.println("fail to insert order " + order_id);
            // noException = false;
            return "error";
        }
    }

    /*
     * Insert Orders_Total
     */
    public String insertOrder2(int order_id, int student_id, double
total_price)
            throws SQLException {

        try {
            Statement stm = conn.createStatement();

            // Update the orders_total table using a new order
            // The order date is set to the current date using SYSDATE.

            String sql = "INSERT INTO Orders_Total (order_id,
student_id, order_date, total_price) " +
                    "VALUES (" + order_id + ", " + student_id + ",
SYSDATE, " + total_price + ")";

            stm.executeUpdate(sql);

            stm.close();

            // System.out.println("succeed to insert Order_table " +
order_id);

            // If order is added return success
            return "success";

        } catch (SQLException e) {
            e.printStackTrace();
            // System.out.println("fail to insert order " + order_id);
            // noException = false;
            return "error";
```

```
        }
    }

    public String cancelOrder(int order_id) {

        try {
            Statement stm = conn.createStatement();

            String sql = "DELETE FROM Orders WHERE order_id = " +
order_id;

            //// System.out.println(sql);

            stm.executeUpdate(sql);

            stm.close();

            System.out.println("succeed to delete Order " + order_id);
            return "success";

        } catch (SQLException e) {
            e.printStackTrace();
            System.out.println("fail to delete order " + order_id);
            // noException = false;
            return "error";
        }
    }

    /*
     * Cancel Order driver
     */
    public void cancelOrder() {

        /**
         * A sample input is:
         */
        int student_id = askForStudentId();
        int order_id = askForOrderId();

        if (order_id == -1) { // Return if the input is -1 for order_id
            System.out.println("No valid order ID was entered. Exiting
the order cancel process.");
            return;
```

```
        }

        System.out.println("Information about your order...");

        // Find the order information using order_id
        orderSearchbyID(order_id);

        // if order_delivered
        if (orderDelivered(order_id)) {
            return;
        }

        if (!allNOTDelivered(order_id)) { // If any of the books have
been delivered, return
            System.out.println("Some or All books in this order have
been delivered. You cannot cancel this order.");
            return;
        } else { // Continue if none of the books have been delivered
            System.out.println("None of the books in this order have
not been delivered.");
        }

        // Check how many days have passed since ordering
        if (getOrderAgeInDays(order_id) > 7) { // Return if less than 7
days passed
            System.out.println("This order is older than 7 days. You
cannot cancel this order.");
            return;
        } else { // Continue if more than 7 days passed
            System.out.println("This order is or less than 7 days
old.");
        }

        System.out.println("Do you want to cancel this order? (Y/N)");

        String answer = in.nextLine();

        if (answer.equalsIgnoreCase("Y")) {
            if (cancelOrder(order_id).equals("error")) { // If there is
any error in the cancelOrder function
                System.out.println("Error cancelling order. Start
Over...");
                return;
```

```
            } else { // If there are no errors, cancel order
                System.out.println("Order cancelled successfully!");
            }
        } else { // If user does not press 'Y'
            System.out.println("You have chosen not to cancel the
order.");
        }

    }

    /*
     * Check if the order ID exists in the database.
     */
    public boolean checkOrder(int order_id) {
        try {
            Statement stm = conn.createStatement();

            // To check if the order exists
            String sql = "SELECT * FROM Orders WHERE order_id = " +
order_id;

            // System.out.println(sql);

            ResultSet result = stm.executeQuery(sql);

            boolean exists = result.next();

            if (!exists) { // If order_id does not exist in orders
table
                System.out.println("No such Order ID exists in the
database.");
            }

            stm.close();
            result.close();

            return exists;
        } catch (SQLException e) {
            e.printStackTrace();
            return false;
        }
    }
```

```
    // To find the age of the order in days based on the order_id
    public int getOrderAgeInDays(int order_id) {
        try {
            Statement stm = conn.createStatement();

            String sql = "SELECT order_date FROM Orders WHERE order_id
= " + order_id;

            //// System.out.println(sql);

            ResultSet result = stm.executeQuery(sql);

            boolean exists = result.next();

            if (!exists) { // If order_id is not in Orders table
                System.out.println("This order does not exist.");
                return -1; // return -1 to indicate that the order does
not exist
            }

            // get the order date from the result set
            Date orderDate = result.getDate("order_date");

            // calculate the age of the order in days
            long ageInMillis = System.currentTimeMillis() -
orderDate.getTime();
            int ageInDays = (int)
TimeUnit.MILLISECONDS.toDays(ageInMillis);

            stm.close();
            result.close();

            return ageInDays;
        } catch (SQLException e) {
            e.printStackTrace();
            return -1; // return -1 to indicate an error occurred
        }
    }

    /*
     * Check if the book ID exists in the database.
     */
    public boolean checkBook(int book_id) {
```

```
        try {
            Statement stm = conn.createStatement();

            // to check if the book_id in Book table
            String sql = "SELECT * FROM Book WHERE book_id = " +
book_id;

            //// System.out.println(sql);

            ResultSet result = stm.executeQuery(sql);

            boolean exists = result.next();

            stm.close();
            result.close();

            return exists;
        } catch (SQLException e) {
            e.printStackTrace();
            return false;
        }
    }

    /*
     * Ask to enter a book ID.
     */
    public int askBookId() {
        int book_id = -1;
        boolean valid_id = false;

        while (!valid_id) { // If the ID is not valid keep looping
            System.out.print("Enter book ID (or -1 to exit): ");
            book_id = Integer.parseInt(in.nextLine());

            if (book_id == -1) {
                return -1; // Return if -1 is pressed
            } else if (checkBook(book_id)) { // If book_id exists,
valid_id is set to true so loop can be existed
                valid_id = true;
            } else {
                System.out.println("Invalid book ID.");
            }
        }
```

```
        return book_id;
    }

    /*
     * Ask to enter a book amount.
     */
    int getAmount(int book_id) {

        int result = 0;

        try {

            Statement stm = conn.createStatement();

            // To find the amount of books based on book_id from Book
table
            String sql = "SELECT amount FROM Book WHERE book_id = " +
book_id;

            ResultSet rs = stm.executeQuery(sql);
            if (!rs.next())
                return 0;
            String[] heads = { "📖 amount" };
            for (int i = 0; i < 1; ++i) {
                try {
                    result = rs.getInt(i + 1);
                    System.out.println(heads[i] + " : ");
                    System.out.println(result);
                } catch (SQLException e) {
                    e.printStackTrace();
                }
            }
        } catch (SQLException e1) {
            e1.printStackTrace();
            // noException = false;
        }

        return result;
    }

    /*
     * Get the discount of a student.
```

```
     */
    double getDiscount(int student_id) {

        double result = 0;
        System.out.print("Discount for " + student_id);
        try {

            Statement stm = conn.createStatement();

            // To find the discount for each student from Student table
            String sql = "SELECT discount FROM Student WHERE student_id
= " + student_id;
            ResultSet rs = stm.executeQuery(sql);
            if (!rs.next())
                return 0;
            String[] heads = { "discount" };
            for (int i = 0; i < 1; ++i) {
                try {
                    result = rs.getDouble(i + 1);
                    System.out.println(heads[i] + " : ");
                    System.out.println(result);
                } catch (SQLException e) {
                    e.printStackTrace();
                }
            }
        } catch (SQLException e1) {
            e1.printStackTrace();
            // noException = false;
        }

        return result;
    }

    /*
     * Get the price of a book.
     */
    double getPriceByID(int book_id) {

        double result = 0;

        try {

            Statement stm = conn.createStatement();
```

```
            // To find the price of each book from Book table
            String sql = "SELECT price FROM Book WHERE book_id = " +
book_id;
            ResultSet rs = stm.executeQuery(sql);
            if (!rs.next())
                return 0;
            String[] heads = { "- book_price" };
            for (int i = 0; i < 1; ++i) {
                try {
                    result = rs.getDouble(i + 1);
                    System.out.println(heads[i] + " : ");
                    System.out.println(result);
                } catch (SQLException e) {
                    e.printStackTrace();
                }
            }
        } catch (SQLException e1) {
            e1.printStackTrace();
            // noException = false;
        }

        return result;

    }

    /*
     * Get the payment information.
     */
    public String[] getPaymentInfo() {
        String[] result = new String[2];
        // Define payment options
        String[] paymentOptions = { "Apple Pay", "AliPay", "Credit
Card" };

        // Print payment options
        System.out.println("Please choose a payment method:");
        for (int i = 0; i < paymentOptions.length; i++) {
            System.out.println((i + 1) + ". " + paymentOptions[i]);
        }

        // Read user input for payment method
        int paymentOption = Integer.parseInt(in.nextLine());
```

```
        String paymentMethod;

        // Assign payment method based on user input
        switch (paymentOption) {
            case 1:
                paymentMethod = "Apple Pay";
                break;
            case 2:
                paymentMethod = "AliPay";
                break;
            case 3:
                paymentMethod = "Credit Card";

                // !! If payment method is credit card, prompt for card
number
                break;
            default:
                // If anything other than 1,2,3 is the input

                System.out.println("Invalid payment option selected.");
                System.out.println("Do you want to select a valid
payment option? (Y/N)");
                String answer = in.next();
                if (answer.equalsIgnoreCase("Y")) {
                    return getPaymentInfo(); // recursively call the
method to get a valid payment option
                } else {
                    return null; // user chose not to enter a valid
payment option
                }
        }

        String cardNumber = null; // Initially set cardNumber to null.
It is updated only when Credit Card is
                                  // selected

        if (paymentMethod.equalsIgnoreCase("Credit Card")) {
            // If payment method is credit card, prompt for card number
            cardNumber = getCreditCard();
            if (cardNumber == null) {
                System.out.println("This will result in error... " +
cardNumber);
```

```
            } else {
                // Process credit card payment with the card number
                System.out.println("Processing credit card payment with
card number: " + cardNumber);
            }
        } else {
            // Process non-credit card payment
            System.out.println("Processing " + paymentMethod + "
payment.");
        }

        result[0] = paymentMethod;
        result[1] = cardNumber;

        return result; // return the chosen payment method
    }

    /* ask for credit card info */
    public String getCreditCard() {
        while (true) {
            System.out.print("Please enter your credit card number (16
digits), or enter \"-1\" to exit: ");
            String input = in.nextLine();
            if (input.equals("-1")) {
                return null;
            }
            String creditCardNumber = input.replaceAll("\\s+", "");
            if (creditCardNumber.matches("\\d{16}")) { // There has to
be 16 digits for the cardNumber
                return creditCardNumber;
            } else { // It is invalid if there are not 16 digits
                System.out.println("Invalid credit card number. Please
try again.");
            }
        }
    }

    /**
     * Close the manager. Do not change this function.
     */
    public void close() {
        System.out.println("Thanks for using this manager! Bye...");
        try {
```

```
            if (conn != null)
                conn.close();
            if (proxySession != null) {
                proxySession.disconnect();
            }
            in.close();
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }

    /**
     * Constructor of University Bookshop.
     */
    public UniversityBookshop() {
        System.out.println("Welcome to use this manager!");
        in = new Scanner(System.in);
    }

    /**
     * Main function
     *
     * @param args
     */
    public static void main(String[] args) {
        UniversityBookshop manager = new UniversityBookshop();
        if (!manager.loginProxy()) {
            System.out.println("Login proxy failed, please re-examine
your username and password!");
            return;
        }
        if (!manager.loginDB()) {
            System.out.println("Login database failed, please
re-examine your username and password!");
            return;
        }
        System.out.println("Login succeed!");
        try {
            manager.run();
        } finally {
            manager.close();
        }
    }
```

```
}
```