```
--Task 19 Creating Sequences
CREATE SEQUENCE customer_seq START WITH 1 INCREMENT BY 1;
CREATE SEQUENCE book_seq START WITH 1 INCREMENT BY 1;
CREATE SEQUENCE sale_seq START WITH 1 INCREMENT BY 1;
CREATE SEQUENCE sale_item_seq START WITH 1 INCREMENT BY 1;
CREATE SEQUENCE employee_seq START WITH 1 INCREMENT BY 1;
CREATE SEQUENCE report_seq START WITH 1 INCREMENT BY 1;
CREATE SEQUENCE supplier_seq START WITH 1 INCREMENT BY 1;
CREATE SEQUENCE purchase_order_seq START WITH 1 INCREMENT BY 1;
CREATE SEQUENCE purchase_order_item_seq START WITH 1 INCREMENT BY 1;
CREATE SEQUENCE return_seq START WITH 1 INCREMENT BY 1;
CREATE SEQUENCE employee_schedule_seq START WITH 1 INCREMENT BY 1;
CREATE SEQUENCE promotion_seq START WITH 1 INCREMENT BY 1;
CREATE SEQUENCE author_seq START WITH 1 INCREMENT BY 1;
CREATE SEQUENCE genre_seq START WITH 1 INCREMENT BY 1;
CREATE SEQUENCE book_category_seq START WITH 1 INCREMENT BY 1;
CREATE SEQUENCE customer_purchase_history_seq START WITH 1 INCREMENT BY 1;
CREATE SEQUENCE employee_access_log_seq START WITH 1 INCREMENT BY 1;
CREATE SEQUENCE customer_preferences_seq START WITH 1 INCREMENT BY 1;
CREATE SEQUENCE employee_training_seq START WITH 1 INCREMENT BY 1;
CREATE SEQUENCE book_shelf_seq START WITH 1 INCREMENT BY 1;
CREATE SEQUENCE sales_promotion_log_seq START WITH 1 INCREMENT BY 1;
CREATE SEQUENCE customer_feedback_seq START WITH 1 INCREMENT BY 1;
--Task 3 and 17, Unique contsraint for email included
-- CUSTOMER TABLE
CREATE TABLE CUSTOMER (
 CustomerID NUMBER(10, 0) DEFAULT customer_seq.NEXTVAl PRIMARY KEY,
 FirstName VARCHAR2(50) NOT NULL,
 LastName VARCHAR2(50) NOT NULL,
 Email VARCHAR2(100) NOT NULL UNIQUE,
 Phone VARCHAR2(20),
 Address VARCHAR2(255),
 City VARCHAR2(50),
 State VARCHAR2(50),
 ZipCode VARCHAR2(20)
);
-- BOOK TABLE
CREATE TABLE BOOK (
 BookID NUMBER(10, 0) DEFAULT book_seq.NEXTVAl PRIMARY KEY,
 Title VARCHAR2(255) NOT NULL,
 Author VARCHAR2(100) NOT NULL,
 Genre VARCHAR2(50),
 Price DECIMAL(10, 2) NOT NULL,
 QuantityInStock NUMBER(10, 0) NOT NULL
);
-- EMPLOYEE TABLE
CREATE TABLE EMPLOYEE (
 EmployeeID NUMBER(10, 0) DEFAULT employee_seq.NEXTVAl PRIMARY KEY,
 FirstName VARCHAR2(50) NOT NULL,
 LastName VARCHAR2(50) NOT NULL,
 Email VARCHAR2(100) NOT NULL UNIQUE,
 Role VARCHAR2(50) NOT NULL,
 AccessLevel VARCHAR2(20) NOT NULL
);
```

```
-- SALE TABLE
CREATE TABLE SALE (
 SaleID NUMBER(10, 0) DEFAULT sale_seq.NEXTVAl PRIMARY KEY,
 CustomerID NUMBER(10, 0) REFERENCES CUSTOMER(CustomerID),
 EmployeeID NUMBER(10, 0) REFERENCES EMPLOYEE(EmployeeID),
 SaleDate DATE NOT NULL
);
-- SALE_ITEM TABLE
CREATE TABLE SALE_ITEM (
 SaleItemID NUMBER(10, 0) DEFAULT sale_item_seq.NEXTVAl PRIMARY KEY,
 SaleID NUMBER(10, 0) REFERENCES SALE(SaleID),
 BookID NUMBER(10, 0) REFERENCES BOOK(BookID),
 QuantitySold NUMBER(10, 0) NOT NULL
);
-- REPORT TABLE
CREATE TABLE REPORT (
 ReportID NUMBER(10, 0) DEFAULT report_seq.NEXTVAl PRIMARY KEY,
 EmployeeID NUMBER(10, 0) REFERENCES EMPLOYEE(EmployeeID),
 ReportDate DATE NOT NULL,
 ReportType VARCHAR2(50) NOT NULL,
 ReportDetails CLOB
);
-- SUPPLIER TABLE
CREATE TABLE SUPPLIER (
 SupplierID NUMBER(10, 0) DEFAULT supplier_seq.NEXTVAl PRIMARY KEY,
 SupplierName VARCHAR2(255) NOT NULL,
 ContactPerson VARCHAR2(255),
 Email VARCHAR2(100),
 Phone VARCHAR2(20)
);
-- PURCHASE_ORDER TABLE
CREATE TABLE PURCHASE_ORDER (
 OrderID NUMBER(10, 0) DEFAULT purchase_order_seq.NEXTVAl PRIMARY KEY,
 SupplierID NUMBER(10, 0) REFERENCES SUPPLIER(SupplierID),
 OrderDate DATE NOT NULL,
 ExpectedDeliveryDate DATE NOT NULL
);
-- PURCHASE_ORDER_ITEM TABLE
CREATE TABLE PURCHASE_ORDER_ITEM (
 ItemID NUMBER(10, 0) DEFAULT purchase_order_item_seq.NEXTVAl PRIMARY KEY,
 OrderID NUMBER(10, 0) REFERENCES PURCHASE_ORDER(OrderID),
 BookID NUMBER(10, 0) REFERENCES BOOK(BookID),
 QuantityOrdered NUMBER(10, 0) NOT NULL,
 UnitPrice DECIMAL(10, 2) NOT NULL
);
-- RETURN TABLE
CREATE TABLE RETURN (
 ReturnID NUMBER(10, 0) DEFAULT return_seq.NEXTVAl PRIMARY KEY,
 SaleID NUMBER(10, 0) REFERENCES SALE(SaleID),
 ReturnDate DATE NOT NULL,
 Reason VARCHAR2(255),
 RefundAmount DECIMAL(10, 2)
);
-- EMPLOYEE_SCHEDULE TABLE
```

```
CREATE TABLE EMPLOYEE_SCHEDULE (
 ScheduleID NUMBER(10, 0) DEFAULT employee_schedule_seq.NEXTVAl PRIMARY
KEY,
 EmployeeID NUMBER(10, 0) REFERENCES EMPLOYEE(EmployeeID),
 ShiftDate DATE NOT NULL,
 ShiftStartTime TIMESTAMP,
 ShiftEndTime TIMESTAMP
);
-- PROMOTION TABLE
CREATE TABLE PROMOTION (
 PromotionID NUMBER(10, 0) DEFAULT promotion_seq.NEXTVAl PRIMARY KEY,
 BookID NUMBER(10, 0) REFERENCES BOOK(BookID),
 StartDate DATE NOT NULL,
 EndDate DATE NOT NULL,
 DiscountPercentage DECIMAL(5, 2) NOT NULL
);
-- AUTHOR TABLE
CREATE TABLE AUTHOR (
 AuthorID NUMBER(10, 0) DEFAULT author_seq.NEXTVAl PRIMARY KEY,
 AuthorName VARCHAR2(255) NOT NULL
);
-- GENRE TABLE
CREATE TABLE GENRE (
 GenreID NUMBER(10, 0) DEFAULT genre_seq.NEXTVAl PRIMARY KEY,
 GenreName VARCHAR2(255) NOT NULL
);
-- BOOK_CATEGORY TABLE
CREATE TABLE BOOK_CATEGORY (
 CategoryID NUMBER(10, 0) DEFAULT book_category_seq.NEXTVAl PRIMARY KEY,
 BookID NUMBER(10, 0) REFERENCES BOOK(BookID),
 GenreID NUMBER(10, 0) REFERENCES GENRE(GenreID)
);
-- CUSTOMER_PURCHASE_HISTORY TABLE
CREATE TABLE CUSTOMER_PURCHASE_HISTORY (
 HistoryID NUMBER(10, 0) DEFAULT customer_purchase_history_seq.NEXTVAl
PRIMARY
KEY,
 CustomerID NUMBER(10, 0) REFERENCES CUSTOMER(CustomerID),
 PurchaseDate DATE NOT NULL,
 BookID NUMBER(10, 0) REFERENCES BOOK(BookID),
 QuantityPurchased NUMBER(10, 0) NOT NULL
);
-- EMPLOYEE_ACCESS_LOG TABLE
CREATE TABLE EMPLOYEE_ACCESS_LOG (
 LogID NUMBER(10, 0) DEFAULT employee_access_log_seq.NEXTVAl PRIMARY KEY,
 EmployeeID NUMBER(10, 0) REFERENCES EMPLOYEE(EmployeeID),
 LogDate DATE NOT NULL,
 LogType VARCHAR2(255) NOT NULL,
 LogDetails VARCHAR2(255)
);
-- CUSTOMER_PREFERENCES TABLE
CREATE TABLE CUSTOMER_PREFERENCES (
 PreferenceID NUMBER(10, 0) DEFAULT customer_preferences_seq.NEXTVAl
PRIMARY
```

```
KEY,
 CustomerID NUMBER(10, 0) REFERENCES CUSTOMER(CustomerID),
 PreferredGenreID NUMBER(10, 0) REFERENCES GENRE(GenreID)
);
-- EMPLOYEE_TRAINING TABLE
CREATE TABLE EMPLOYEE_TRAINING (
 TrainingID NUMBER(10, 0) DEFAULT employee_training_seq.NEXTVAl PRIMARY
KEY,
 EmployeeID NUMBER(10, 0) REFERENCES EMPLOYEE(EmployeeID),
 TrainingDate DATE NOT NULL,
 TrainingTopic VARCHAR2(255) NOT NULL,
 Trainer VARCHAR2(255)
);
-- BOOK_SHELF TABLE
CREATE TABLE BOOK_SHELF (
 ShelfID NUMBER(10, 0) DEFAULT book_shelf_seq.NEXTVAl PRIMARY KEY,
 ShelfNumber VARCHAR2(255) NOT NULL,
 Capacity NUMBER(10, 0) NOT NULL
);
-- SALES_PROMOTION_LOG TABLE
CREATE TABLE SALES_PROMOTION_LOG (
 LogID NUMBER(10, 0) DEFAULT sales_promotion_log_seq.NEXTVAl PRIMARY KEY,
 SaleID NUMBER(10, 0) REFERENCES SALE(SaleID),
 PromotionID NUMBER(10, 0) REFERENCES PROMOTION(PromotionID),
 LogDate DATE NOT NULL,
 LogDetails VARCHAR2(255)
);
-- CUSTOMER_FEEDBACK TABLE
CREATE TABLE CUSTOMER_FEEDBACK (
 FeedbackID NUMBER(10, 0) DEFAULT customer_feedback_seq.NEXTVAl PRIMARY
KEY,
 CustomerID NUMBER(10, 0) REFERENCES CUSTOMER(CustomerID),
 FeedbackDate DATE NOT NULL,
 FeedbackText CLOB
);
-- Sample data for CUSTOMER table
INSERT INTO CUSTOMER VALUES (customer_seq.NEXTVAL, 'John', 'Doe',
'john.doe@email.com', '1234567890', '123 Main St', 'Cityville', 'CA',
'12345');
INSERT INTO CUSTOMER VALUES (customer_seq.NEXTVAL, 'Jane', 'Smith',
'jane.smith@email.com', '9876543210', '456 Oak St', 'Towndale', 'NY',
'54321');
INSERT INTO CUSTOMER VALUES (customer_seq.NEXTVAL, 'Bob', 'Johnson',
'bob.johnson@email.com', '5551234567', '789 Pine St', 'Villagetown', 'TX',
'67890');
INSERT INTO CUSTOMER VALUES (customer_seq.NEXTVAL, 'Alice', 'Williams',
'alice.williams@email.com', '9998887777', '101 Elm St', 'Hamletville',
'FL',
'11223');
INSERT INTO CUSTOMER VALUES (customer_seq.NEXTVAL, 'Charlie', 'Brown',
'charlie.brown@email.com', '1112223333', '202 Birch St', 'Valleyville',
'AZ',
'33445');
-- Sample data for BOOK table
```

```
INSERT INTO BOOK VALUES (book_seq.NEXTVAL, 'The Great Gatsby', 'F. Scott
Fitzgerald', 'Fiction', 15.99, 100);
INSERT INTO BOOK VALUES (book_seq.NEXTVAL, 'To Kill a Mockingbird',
'Harper Lee',
'Fiction', 12.99, 75);
INSERT INTO BOOK VALUES (book_seq.NEXTVAL, '1984', 'George Orwell',
'Dystopian',
10.99, 120);
INSERT INTO BOOK VALUES (book_seq.NEXTVAL, 'The Catcher in the Rye', 'J.D.
Salinger', 'Coming of Age', 11.99, 90);
INSERT INTO BOOK VALUES (book_seq.NEXTVAL, 'Pride and Prejudice', 'Jane
Austen',
'Romance', 14.99, 80);
-- Sample data for EMPLOYEE table
INSERT INTO EMPLOYEE VALUES (employee_seq.NEXTVAL, 'Manager', 'One',
'manager.one@email.com', 'Manager', 'Full Access');
INSERT INTO EMPLOYEE VALUES (employee_seq.NEXTVAL, 'Staff', 'Two',
'staff.two@email.com', 'Staff', 'Limited Access');
INSERT INTO EMPLOYEE VALUES (employee_seq.NEXTVAL, 'Admin', 'Three',
'admin.three@email.com', 'Administrator', 'Full Access');
INSERT INTO EMPLOYEE VALUES (employee_seq.NEXTVAL, 'Clerk', 'Four',
'clerk.four@email.com', 'Clerk', 'Limited Access');
INSERT INTO EMPLOYEE VALUES (employee_seq.NEXTVAL, 'Supervisor', 'Five',
'supervisor.five@email.com', 'Supervisor', 'Limited Access');
-- Sample data for SALE table
INSERT INTO SALE VALUES (sale_seq.NEXTVAL, 1, 1, TO_DATE('2023-01-28',
'YYYY-MM-DD'));
INSERT INTO SALE VALUES (sale_seq.NEXTVAL, 2, 2, TO_DATE('2023-01-29',
'YYYY-MM-DD'));
INSERT INTO SALE VALUES (sale_seq.NEXTVAL, 3, 3, TO_DATE('2023-01-30',
'YYYY-MM-DD'));
INSERT INTO SALE VALUES (sale_seq.NEXTVAL, 4, 4, TO_DATE('2023-01-31',
'YYYY-MM-DD'));
INSERT INTO SALE VALUES (sale_seq.NEXTVAL, 5, 5, TO_DATE('2023-02-01',
'YYYY-MM-DD'));
-- Sample data for SALE_ITEM table
INSERT INTO SALE_ITEM VALUES (sale_item_seq.NEXTVAL, 1, 1, 2);
INSERT INTO SALE_ITEM VALUES (sale_item_seq.NEXTVAL, 2, 2, 3);
INSERT INTO SALE_ITEM VALUES (sale_item_seq.NEXTVAL, 3, 3, 1);
INSERT INTO SALE_ITEM VALUES (sale_item_seq.NEXTVAL, 4, 4, 4);
INSERT INTO SALE_ITEM VALUES (sale_item_seq.NEXTVAL, 5, 5, 2);
-- Sample data for REPORT table
INSERT INTO REPORT VALUES (report_seq.NEXTVAL, 1, TO_DATE('2023-02-01',
'YYYY-MM-DD'), 'Monthly Sales', 'Detailed sales report for February.');
INSERT INTO REPORT VALUES (report_seq.NEXTVAL, 2, TO_DATE('2023-02-02',
'YYYY-MM-DD'), 'Inventory Status', 'Current stock levels and reordering
recommendations.');
INSERT INTO REPORT VALUES (report_seq.NEXTVAL, 3, TO_DATE('2023-02-03',
'YYYY-MM-DD'), 'Employee Performance', 'Evaluation of staff performance
for the month.');
INSERT INTO REPORT VALUES (report_seq.NEXTVAL, 4, TO_DATE('2023-02-04',
'YYYY-MM-DD'), 'Customer Feedback', 'Summary of customer feedback and
suggestions.');
```

```
INSERT INTO REPORT VALUES (report_seq.NEXTVAL, 5, TO_DATE('2023-02-05',
'YYYY-MM-DD'), 'Promotion Impact', 'Analysis of the effectiveness of
recent promotions.');
-- Sample data for SUPPLIER table
INSERT INTO SUPPLIER VALUES (supplier_seq.NEXTVAL, 'Book Supplier Inc.',
'John
Supplier', 'john.supplier@email.com', '555-1234');
INSERT INTO SUPPLIER VALUES (supplier_seq.NEXTVAL, 'Paperbacks R Us',
'Jane
Paperback', 'jane.paperback@email.com', '555-5678');
INSERT INTO SUPPLIER VALUES (supplier_seq.NEXTVAL, 'Literary
Distributors', 'Bob
Distributor', 'bob.distributor@email.com', '555-9101');
INSERT INTO SUPPLIER VALUES (supplier_seq.NEXTVAL, 'Wholesale Books Co.',
'Alice
Wholesale', 'alice.wholesale@email.com', '555-1122');
INSERT INTO SUPPLIER VALUES (supplier_seq.NEXTVAL, 'Book World Suppliers',
'Charlie
Book', 'charlie.book@email.com', '555-3344');
-- Sample data for PURCHASE_ORDER table
INSERT INTO PURCHASE_ORDER VALUES (purchase_order_seq.NEXTVAL, 1,
TO_DATE('2023-02-
01', 'YYYY-MM-DD'), TO_DATE('2023-02-15', 'YYYY-MM-DD'));
INSERT INTO PURCHASE_ORDER VALUES (purchase_order_seq.NEXTVAL, 2,
TO_DATE('2023-02-
02', 'YYYY-MM-DD'), TO_DATE('2023-02-16', 'YYYY-MM-DD'));
INSERT INTO PURCHASE_ORDER VALUES (purchase_order_seq.NEXTVAL, 3,
TO_DATE('2023-02-
03', 'YYYY-MM-DD'), TO_DATE('2023-02-17', 'YYYY-MM-DD'));
INSERT INTO PURCHASE_ORDER VALUES (purchase_order_seq.NEXTVAL, 4,
TO_DATE('2023-02-
04', 'YYYY-MM-DD'), TO_DATE('2023-02-18', 'YYYY-MM-DD'));
INSERT INTO PURCHASE_ORDER VALUES (purchase_order_seq.NEXTVAL, 5,
TO_DATE('2023-02-
05', 'YYYY-MM-DD'), TO_DATE('2023-02-19', 'YYYY-MM-DD'));
-- Sample data for PURCHASE_ORDER_ITEM table
INSERT INTO PURCHASE_ORDER_ITEM VALUES (purchase_order_item_seq.NEXTVAL,
1, 1, 50,
14.50);
INSERT INTO PURCHASE_ORDER_ITEM VALUES (purchase_order_item_seq.NEXTVAL,
2, 2, 40,
11.75);
INSERT INTO PURCHASE_ORDER_ITEM VALUES (purchase_order_item_seq.NEXTVAL,
3, 3, 30,
9.99);
INSERT INTO PURCHASE_ORDER_ITEM VALUES (purchase_order_item_seq.NEXTVAL,
4, 4, 20,
8.25);
INSERT INTO PURCHASE_ORDER_ITEM VALUES (purchase_order_item_seq.NEXTVAL,
5, 5, 10,
13.00);
-- Sample data for RETURN table
INSERT INTO RETURN VALUES (return_seq.NEXTVAL, 1, TO_DATE('2023-02-05',
'YYYY-MM-DD'), 'Defective product', 5.00);
```

```
INSERT INTO RETURN VALUES (return_seq.NEXTVAL, 2, TO_DATE('2023-02-06',
'YYYY-MM-DD'), 'Changed mind', 8.50);
INSERT INTO RETURN VALUES (return_seq.NEXTVAL, 3, TO_DATE('2023-02-07',
'YYYY-MM-DD'), 'Not as expected', 12.25);
INSERT INTO RETURN VALUES (return_seq.NEXTVAL, 4, TO_DATE('2023-02-08',
'YYYY-MM-DD'), 'Ordered wrong item', 15.75);
INSERT INTO RETURN VALUES (return_seq.NEXTVAL, 5, TO_DATE('2023-02-09',
'YYYY-MM-DD'), 'Duplicate order', 6.50);
-- Sample data for EMPLOYEE_SCHEDULE table
INSERT INTO EMPLOYEE_SCHEDULE VALUES (employee_schedule_seq.NEXTVAL, 1,
TO_DATE('2023-02-01', 'YYYY-MM-DD'), TO_TIMESTAMP('08:00:00',
'HH24:MI:SS'),
TO_TIMESTAMP('16:00:00', 'HH24:MI:SS'));
INSERT INTO EMPLOYEE_SCHEDULE VALUES (employee_schedule_seq.NEXTVAL, 2,
TO_DATE('2023-02-02', 'YYYY-MM-DD'), TO_TIMESTAMP('09:00:00',
'HH24:MI:SS'),
TO_TIMESTAMP('17:00:00', 'HH24:MI:SS'));
INSERT INTO EMPLOYEE_SCHEDULE VALUES (employee_schedule_seq.NEXTVAL, 3,
TO_DATE('2023-02-03', 'YYYY-MM-DD'), TO_TIMESTAMP('10:00:00',
'HH24:MI:SS'),
TO_TIMESTAMP('18:00:00', 'HH24:MI:SS'));
INSERT INTO EMPLOYEE_SCHEDULE VALUES (employee_schedule_seq.NEXTVAL, 4,
TO_DATE('2023-02-04', 'YYYY-MM-DD'), TO_TIMESTAMP('11:00:00',
'HH24:MI:SS'),
TO_TIMESTAMP('19:00:00', 'HH24:MI:SS'));
INSERT INTO EMPLOYEE_SCHEDULE VALUES (employee_schedule_seq.NEXTVAL, 5,
TO_DATE('2023-02-05', 'YYYY-MM-DD'), TO_TIMESTAMP('12:00:00',
'HH24:MI:SS'),
TO_TIMESTAMP('20:00:00', 'HH24:MI:SS'));
-- Sample data for PROMOTION table
INSERT INTO PROMOTION VALUES (promotion_seq.NEXTVAL, 1, TO_DATE('2023-02-
01',
'YYYY-MM-DD'), TO_DATE('2023-02-14', 'YYYY-MM-DD'), 10.00);
INSERT INTO PROMOTION VALUES (promotion_seq.NEXTVAL, 2, TO_DATE('2023-02-
02',
'YYYY-MM-DD'), TO_DATE('2023-02-15', 'YYYY-MM-DD'), 15.00);
INSERT INTO PROMOTION VALUES (promotion_seq.NEXTVAL, 3, TO_DATE('2023-02-
03',
'YYYY-MM-DD'), TO_DATE('2023-02-16', 'YYYY-MM-DD'), 12.50);
INSERT INTO PROMOTION VALUES (promotion_seq.NEXTVAL, 4, TO_DATE('2023-02-
04',
'YYYY-MM-DD'), TO_DATE('2023-02-17', 'YYYY-MM-DD'), 8.00);
INSERT INTO PROMOTION VALUES (promotion_seq.NEXTVAL, 5, TO_DATE('2023-02-
05',
'YYYY-MM-DD'), TO_DATE('2023-02-18', 'YYYY-MM-DD'), 20.00);
-- Sample data for AUTHOR table
INSERT INTO AUTHOR VALUES (author_seq.NEXTVAL, 'William Shakespeare');
INSERT INTO AUTHOR VALUES (author_seq.NEXTVAL, 'J.K. Rowling');
INSERT INTO AUTHOR VALUES (author_seq.NEXTVAL, 'Agatha Christie');
INSERT INTO AUTHOR VALUES (author_seq.NEXTVAL, 'George R.R. Martin');
INSERT INTO AUTHOR VALUES (author_seq.NEXTVAL, 'Dan Brown');
-- Sample data for GENRE table
INSERT INTO GENRE VALUES (genre_seq.NEXTVAL, 'Mystery');
INSERT INTO GENRE VALUES (genre_seq.NEXTVAL, 'Fantasy');
```

```
INSERT INTO GENRE VALUES (genre_seq.NEXTVAL, 'Science Fiction');
INSERT INTO GENRE VALUES (genre_seq.NEXTVAL, 'Romance');
INSERT INTO GENRE VALUES (genre_seq.NEXTVAL, 'Thriller');
-- Sample data for BOOK_CATEGORY table
INSERT INTO BOOK_CATEGORY VALUES (book_category_seq.NEXTVAL, 1, 1);
INSERT INTO BOOK_CATEGORY VALUES (book_category_seq.NEXTVAL, 2, 2);
INSERT INTO BOOK_CATEGORY VALUES (book_category_seq.NEXTVAL, 3, 3);
INSERT INTO BOOK_CATEGORY VALUES (book_category_seq.NEXTVAL, 4, 4);
INSERT INTO BOOK_CATEGORY VALUES (book_category_seq.NEXTVAL, 5, 5);
-- Sample data for CUSTOMER_PURCHASE_HISTORY table
INSERT INTO CUSTOMER_PURCHASE_HISTORY VALUES
(customer_purchase_history_seq.NEXTVAL, 1, TO_DATE('2023-02-01', 'YYYY-MM-
DD'), 1,
2);
INSERT INTO CUSTOMER_PURCHASE_HISTORY VALUES
(customer_purchase_history_seq.NEXTVAL, 2, TO_DATE('2023-02-02', 'YYYY-MM-
DD'), 2,
3);
INSERT INTO CUSTOMER_PURCHASE_HISTORY VALUES
(customer_purchase_history_seq.NEXTVAL, 3, TO_DATE('2023-02-03', 'YYYY-MM-
DD'), 3,
1)  ;
INSERT INTO CUSTOMER_PURCHASE_HISTORY VALUES
(customer_purchase_history_seq.NEXTVAL, 4, TO_DATE('2023-02-04', 'YYYY-MM-
DD'), 4,
4);
INSERT INTO CUSTOMER_PURCHASE_HISTORY VALUES
(customer_purchase_history_seq.NEXTVAL, 5, TO_DATE('2023-02-05', 'YYYY-MM-
DD'), 5,
2)  ;
-- Sample data for EMPLOYEE_ACCESS_LOG table
INSERT INTO EMPLOYEE_ACCESS_LOG VALUES (employee_access_log_seq.NEXTVAL,
1,
TO_DATE('2023-02-01', 'YYYY-MM-DD'), 'Login', 'Successful login');
INSERT INTO EMPLOYEE_ACCESS_LOG VALUES (employee_access_log_seq.NEXTVAL,
2,
TO_DATE('2023-02-02', 'YYYY-MM-DD'), 'Logout', 'User logged out');
INSERT INTO EMPLOYEE_ACCESS_LOG VALUES (employee_access_log_seq.NEXTVAL,
3,
TO_DATE('2023-02-03', 'YYYY-MM-DD'), 'Data Update', 'Updated customer
records');
INSERT INTO EMPLOYEE_ACCESS_LOG VALUES (employee_access_log_seq.NEXTVAL,
4,
TO_DATE('2023-02-04', 'YYYY-MM-DD'), 'Report Access', 'Viewed monthly
sales
report');
INSERT INTO EMPLOYEE_ACCESS_LOG VALUES (employee_access_log_seq.NEXTVAL,
5,
TO_DATE('2023-02-05', 'YYYY-MM-DD'), 'Error', 'Access denied, invalid
credentials');
-- Sample data for CUSTOMER_PREFERENCES table
INSERT INTO CUSTOMER_PREFERENCES VALUES (customer_preferences_seq.NEXTVAL,
1, 1);
```

```
INSERT INTO CUSTOMER_PREFERENCES VALUES (customer_preferences_seq.NEXTVAL,
2, 2);
INSERT INTO CUSTOMER_PREFERENCES VALUES (customer_preferences_seq.NEXTVAL,
3, 3);
INSERT INTO CUSTOMER_PREFERENCES VALUES (customer_preferences_seq.NEXTVAL,
4, 4);
INSERT INTO CUSTOMER_PREFERENCES VALUES (customer_preferences_seq.NEXTVAL,
5, 5);
-- Sample data for EMPLOYEE_TRAINING table
INSERT INTO EMPLOYEE_TRAINING VALUES (employee_training_seq.NEXTVAL, 1,
TO_DATE('2023-02-01', 'YYYY-MM-DD'), 'Customer Service Training', 'John
Trainer');
INSERT INTO EMPLOYEE_TRAINING VALUES (employee_training_seq.NEXTVAL, 2,
TO_DATE('2023-02-02', 'YYYY-MM-DD'), 'Inventory Management', 'Jane
Trainer');
INSERT INTO EMPLOYEE_TRAINING VALUES (employee_training_seq.NEXTVAL, 3,
TO_DATE('2023-02-03', 'YYYY-MM-DD'), 'Point of Sale System', 'Bob
Trainer');
INSERT INTO EMPLOYEE_TRAINING VALUES (employee_training_seq.NEXTVAL, 4,
TO_DATE('2023-02-04', 'YYYY-MM-DD'), 'Sales Techniques', 'Alice Trainer');
INSERT INTO EMPLOYEE_TRAINING VALUES (employee_training_seq.NEXTVAL, 5,
TO_DATE('2023-02-05', 'YYYY-MM-DD'), 'Security Protocols', 'Chris
Trainer');
-- Sample data for BOOK_SHELF table
INSERT INTO BOOK_SHELF VALUES (book_shelf_seq.NEXTVAL, 'A-1', 100);
INSERT INTO BOOK_SHELF VALUES (book_shelf_seq.NEXTVAL, 'B-2', 150);
INSERT INTO BOOK_SHELF VALUES (book_shelf_seq.NEXTVAL, 'C-3', 120);
INSERT INTO BOOK_SHELF VALUES (book_shelf_seq.NEXTVAL, 'D-4', 200);
INSERT INTO BOOK_SHELF VALUES (book_shelf_seq.NEXTVAL, 'E-5', 80);
-- Sample data for SALES_PROMOTION_LOG table
INSERT INTO SALES_PROMOTION_LOG VALUES (sales_promotion_log_seq.NEXTVAL,
1, 1,
TO_DATE('2023-02-01', 'YYYY-MM-DD'), 'Promotion applied successfully');
INSERT INTO SALES_PROMOTION_LOG VALUES (sales_promotion_log_seq.NEXTVAL,
2, 2,
TO_DATE('2023-02-02', 'YYYY-MM-DD'), 'Promotion code redeemed');
INSERT INTO SALES_PROMOTION_LOG VALUES (sales_promotion_log_seq.NEXTVAL,
3, 3,
TO_DATE('2023-02-03', 'YYYY-MM-DD'), 'Customer notified about upcoming
promotion');
INSERT INTO SALES_PROMOTION_LOG VALUES (sales_promotion_log_seq.NEXTVAL,
4, 4,
TO_DATE('2023-02-04', 'YYYY-MM-DD'), 'Promotion details updated');
INSERT INTO SALES_PROMOTION_LOG VALUES (sales_promotion_log_seq.NEXTVAL,
5, 5,
TO_DATE('2023-02-05', 'YYYY-MM-DD'), 'Promotion expired, no longer
applicable');
-- Sample data for CUSTOMER_FEEDBACK table
INSERT INTO CUSTOMER_FEEDBACK VALUES (customer_feedback_seq.NEXTVAL, 1,
TO_DATE('2023-02-01', 'YYYY-MM-DD'), 'Great service, loved the book
recommendations!');
INSERT INTO CUSTOMER_FEEDBACK VALUES (customer_feedback_seq.NEXTVAL, 2,
TO_DATE('2023-02-02', 'YYYY-MM-DD'), 'Book selection could be improved');
INSERT INTO CUSTOMER_FEEDBACK VALUES (customer_feedback_seq.NEXTVAL, 3,
```

```
TO_DATE('2023-02-03', 'YYYY-MM-DD'), 'Fast and efficient checkout
process');
INSERT INTO CUSTOMER_FEEDBACK VALUES (customer_feedback_seq.NEXTVAL, 4,
TO_DATE('2023-02-04', 'YYYY-MM-DD'), 'Friendly staff, will visit again!');
INSERT INTO CUSTOMER_FEEDBACK VALUES (customer_feedback_seq.NEXTVAL, 5,
TO_DATE('2023-02-05', 'YYYY-MM-DD'), 'Clean and well-organized
bookstore');
--Views
--Views with String function (Task 4 and 5) Concat and Substr
--The vw_customer_names view combines the FirstName and LastName columns
to present
--a full name for each customer. Additionally, it extracts the username
from the
--email address by using the SUBSTR function, providing a convenient
snapshot of
--customer names and corresponding usernames for various applications that
require
--this information.
CREATE VIEW vw_customer_names AS
SELECT CustomerID,
 FirstName || ' ' || LastName AS Full_Name,
 SUBSTR(Email, 1, INSTR(Email, '@') - 1) AS Username
FROM CUSTOMER;
--View with Number function (Task 4 and 6) Multiplication and Round
--The vw_book_prices view provides a concise summary of book information,
including
--the original price and a discounted price calculated at a 10% reduction.
This view
--assists in quickly assessing pricing details for effective decision-
making and
--promotional strategies within the bookstore management system.
CREATE VIEW vw_book_prices AS
SELECT BookID,
 Title,
 Price,
 ROUND(Price * 0.9, 2) AS Discounted_Price
FROM BOOK;
--Creating index Task18
CREATE INDEX idx_sale_date ON SALE (SaleDate);
--View with Date function (Task 4 and 7) Months_Between and Next_Day
--The vw_recent_sales view provides a concise summary of recent sales,
indicating
--the sale ID, sale date, and the number of months elapsed since each
sale.
--Additionally, it includes the date of the next Monday after each sale,
offering a
--convenient snapshot for tracking recent sales trends and planning future
business
--activities.
CREATE VIEW vw_recent_sales AS
SELECT SaleID,
 SaleDate,
 MONTHS_BETWEEN(SYSDATE, SaleDate) AS Months_Ago,
 NEXT_DAY(SaleDate, 'MONDAY') AS Next_Monday
```

```
FROM SALE;
--This view simplifies tracking and managing employee status, providing a
quick
--overview of active and inactive employees based on their roles.
--View with DECODE function (Task 4 and 8)
CREATE VIEW vw_employee_status AS
SELECT EmployeeID,
 FirstName,
 LastName,
 DECODE(Role, 'Manager', 'Active', 'Administrator', 'Active', 'Inactive')
AS
Status
FROM EMPLOYEE;
--View with Group By, Having, and Inner Join ( Task 4,9,11)
--This view provides a list of books that have high sales quantities,
helping to
--identify popular or best-selling books. Adjust the HAVING condition
according to
--your specific requirements.
CREATE VIEW vw_high_sales_books AS
SELECT
 B.BookID,
 B.Title,
 SUM(SI.QuantitySold) AS TotalQuantitySold
FROM
 BOOK B
Inner JOIN
 SALE_ITEM SI ON B.BookID = SI.BookID
GROUP BY
 B.BookID, B.Title
HAVING
 SUM(SI.QuantitySold) >2; -- adjust the threshold as needed
--View with Outer Join (Task 4,11)
--The creation of this view makes sense in scenarios where you want to
analyze
--customer data, including those who haven't made any purchases recently.
It provides
--a comprehensive overview of all customers and their associated sales
data, helping
--you identify patterns and target specific customer segments for
promotions or
--engagement.
CREATE VIEW vw_sales_and_customers AS
SELECT s.SaleID,
 s.SaleDate,
 c.CustomerID,
 c.FirstName,
 c.LastName
FROM CUSTOMER c
LEFT OUTER JOIN SALE s ON c.CustomerID = s.CustomerID;
--Creating index Task18
--View with SubQuery ( Task 4 and 11)
--This view is useful for identifying and managing high-value customers
based on
```

```
--their purchasing behavior.

CREATE VIEW vw_high_value_customers AS
SELECT
 c.CustomerID,
 FirstName || ' ' || LastName AS Full_Name,
 c.Email,
 total_amount_spent.TotalAmountSpent
FROM
 CUSTOMER c
JOIN
 (
 SELECT
 s.CustomerID,
 SUM(si.QuantitySold * b.Price) AS TotalAmountSpent
 FROM
 SALE s
 JOIN
 SALE_ITEM si ON s.SaleID = si.SaleID
 JOIN
 BOOK b ON si.BookID = b.BookID
 GROUP BY
 s.CustomerID
 HAVING
 SUM(si.QuantitySold * b.Price) > 10 -- Adjust the threshold as needed
 ) total_amount_spent ON c.CustomerID = total_amount_spent.CustomerID;
--Task 12 Demonstarte View Point
-- Assume a new batch of books is received, and you want to update the
stock.If you
--enter the wrong quanitity, you can rollback using the savepoint created.
To be used
--before any updatation to records.
SAVEPOINT start_update;
UPDATE BOOK SET QuantityInStock = QuantityInStock + 10 WHERE Genre =
'Fiction';
Select * from BOOK ;
ROLLBACK TO start_update;
Select * from BOOK ;
--Task 13 Insert and Delete Through a View
-- Create a view that selects highly rated books
CREATE VIEW vw_highly_cost_books AS
SELECT BookID, Title, Author, Genre , Price, Quantityinstock
FROM BOOK
WHERE Price >= 14;
-- Insert a new highly costed book through the view
INSERT INTO vw_highly_cost_books (BookID, Title, Author, Genre, Price,
Quantityinstock)
VALUES (101, 'New Bestseller', 'Author X', 'Fiction',25.00,135);
-- Delete a highly cost book through the view
DELETE FROM vw_highly_cost_books
WHERE BookID = 101;
--Creating index Task18
CREATE INDEX idx_genre_book ON BOOK (Genre);
--Task 14 Update with Embedded Select
```

```
-- Update the price of a book based on its genre
UPDATE BOOK
SET Price = Price * 1.1
WHERE Genre IN (
 SELECT DISTINCT Genre
 FROM BOOK
 WHERE Price < 13
);
--Creating index Task18
CREATE INDEX idx_quantity_in_stock ON BOOK (QuantityInStock);
CREATE INDEX idx_price ON BOOK (Price);
--Task 15 Delete rows using more than one conditions in the where clause
-- Delete books that are out of stock and have a low rating
INSERT INTO BOOK VALUES (book_seq.NEXTVAL, 'Harry Potter', 'J. K.
Rowling', 'Sci-Fi', 14.99, 120);
DELETE FROM BOOK
WHERE Price >14 AND Quantityinstock> 100;
--Task 16 Create a Table from Another Table
--This statement creates the CUSTOMER_FEEDBACK_ARCHIVE table and populates
it with
--feedback entries received before February 3, 2023.
CREATE TABLE CUSTOMER_FEEDBACK_ARCHIVE AS
SELECT * FROM CUSTOMER_FEEDBACK
WHERE FeedbackDate < TO_DATE('2023-02-03', 'YYYY-MM-DD');
--Task 17 unique constraints and check constraints wherever possible
-- Add a new column 'Age' to the 'EMPLOYEE' table
ALTER TABLE EMPLOYEE
ADD (Age NUMBER(3, 0) CHECK (Age >= 18 AND Age <= 100));
-- Add check constraints to the PURCHASE_ORDER_ITEM table
ALTER TABLE PURCHASE_ORDER_ITEM
ADD CONSTRAINT CHK_PURCHASE_ORDER_ITEM_QUANTITY CHECK (QuantityOrdered >=
0);
-- Add check constraints to the CUSTOMER_PURCHASE_HISTORY table
ALTER TABLE CUSTOMER_PURCHASE_HISTORY
ADD CONSTRAINT CHK_CUSTOMER_PURCHASE_HISTORY_QUANTITY CHECK
(QuantityPurchased >= 0);


--FINAL LAB
--TASK 1,2,7
--The procedure "ProcessBookData" is designed to analyze and display
essential information about available books and recorded sales, providing
insights into inventory and sales activities.
CREATE OR REPLACE PROCEDURE ProcessBookData IS
    v_total_books NUMBER;
    v_total_sales NUMBER;
BEGIN
    -- Calculate total number of books in stock
    SELECT COUNT(*) INTO v_total_books FROM BOOK;

    -- Calculate total number of sales
    SELECT COUNT(*) INTO v_total_sales FROM SALE;

    -- Check if there are books in stock
```

```
    IF v_total_books > 0 THEN
        DBMS_OUTPUT.PUT_LINE('Total number of books in stock: ' ||
v_total_books);
    ELSE
        DBMS_OUTPUT.PUT_LINE('No books in stock.');
    END IF;

    -- Check if there are sales recorded
    IF v_total_sales > 0 THEN
        DBMS_OUTPUT.PUT_LINE('Total number of sales recorded: ' ||
v_total_sales);
    ELSE
        DBMS_OUTPUT.PUT_LINE('No sales recorded.');
    END IF;

    -- Loop through each book and display details
    FOR book_rec IN (SELECT * FROM BOOK) LOOP
        DBMS_OUTPUT.PUT_LINE('Book ID: ' || book_rec.BookID || ', Title: '
|| book_rec.Title || ', Author: ' || book_rec.Author || ', Price: ' ||
book_rec.Price);
    END LOOP;

    -- Loop through each sale and display details
    FOR sale_rec IN (SELECT * FROM SALE) LOOP
        DBMS_OUTPUT.PUT_LINE('Sale ID: ' || sale_rec.SaleID || ', Customer
ID: ' || sale_rec.CustomerID || ', Employee ID: ' || sale_rec.EmployeeID
|| ', Sale Date: ' || sale_rec.SaleDate);
    END LOOP;
END;
/


BEGIN
    ProcessBookData;
END;

--TASK 3,6,7,8
--This function calculates the average price of books within a specified
genre by summing up their prices and dividing by the count of books.
--It aids in analyzing pricing trends within different book genres.
-- Create a package
CREATE OR REPLACE PACKAGE BookPackage AS
    -- Define a function to calculate the average price of books in a
specific genre
    FUNCTION CalculateAveragePrice(genre_id NUMBER) RETURN NUMBER;

    -- Define a procedure to display the average price of books in a
specific genre
    PROCEDURE DisplayAveragePrice(genre_id NUMBER);
END BookPackage;
/

-- Create the package body
CREATE OR REPLACE PACKAGE BODY BookPackage AS
```

```
    -- Function to calculate average price of books in a specific genre
    FUNCTION CalculateAveragePrice(genre_id NUMBER) RETURN NUMBER IS
        total_price NUMBER := 0;
        book_count NUMBER := 0;
    BEGIN
        -- Cursor to retrieve book price for the given genre
        FOR book_rec IN (SELECT Price FROM BOOK WHERE BookID IN (SELECT
BookID FROM BOOK_CATEGORY WHERE GenreID = genre_id)) LOOP
            total_price := total_price + book_rec.Price;
            book_count := book_count + 1;
        END LOOP;

        IF book_count = 0 THEN
            RETURN NULL; -- Return NULL if no books found in the given
genre
        ELSE
            RETURN total_price / book_count; -- Return the average price
        END IF;
    END CalculateAveragePrice;

    -- Procedure to display average price of books in a specific genre
    PROCEDURE DisplayAveragePrice(genre_id NUMBER) IS
        total_price NUMBER := 0; -- Declaration added here
        book_count NUMBER := 0; -- Declaration added here
        CURSOR book_cursor IS
            SELECT Price FROM BOOK WHERE BookID IN (SELECT BookID FROM
BOOK_CATEGORY WHERE GenreID = genre_id);
        average_price NUMBER;
    BEGIN
        -- Open the cursor
        OPEN book_cursor;

        -- Fetch book prices and calculate total price
        LOOP
            FETCH book_cursor INTO average_price;
            EXIT WHEN book_cursor%NOTFOUND;

            total_price := total_price + average_price;
            book_count := book_count + 1;
        END LOOP;

        -- Close the cursor
        CLOSE book_cursor;

        IF book_count = 0 THEN
            DBMS_OUTPUT.PUT_LINE('No books found in the specified
genre.');
        ELSE
            DBMS_OUTPUT.PUT_LINE('Average price of books in Genre ID ' ||
genre_id || ': ' || TO_CHAR(total_price / book_count, '99999.99'));
        END IF;
    END DisplayAveragePrice;
END BookPackage;
/
```

```
-- Execute the procedure to display the average price for a specific genre
ID (e.g., Genre ID 1)
BEGIN
    BookPackage.DisplayAveragePrice(6);
END;
/


--Task 4,6
--This function calculates the total discount amount for a given book
based on the quantity purchased,
--considering the discount percentage associated with the book. It helps
in determining the discounted price for bulk purchases of books.
CREATE OR REPLACE FUNCTION calculate_discounted_price (
    p_book_id IN NUMBER,
    p_quantity IN NUMBER
) RETURN NUMBER
IS
    v_discount_percentage PROMOTION.DiscountPercentage%TYPE;
    v_price_per_unit BOOK.Price%TYPE;
    v_total_price NUMBER;
    v_available_quantity BOOK.QuantityInStock%TYPE;
    insufficient_quantity EXCEPTION;
BEGIN
    -- Retrieve discount percentage for the given book
    SELECT DiscountPercentage INTO v_discount_percentage
    FROM PROMOTION
    WHERE BookID = p_book_id;

    -- Retrieve price per unit for the given book
    SELECT Price INTO v_price_per_unit
    FROM BOOK
    WHERE BookID = p_book_id;

    -- Retrieve available quantity in stock for the given book
    SELECT QuantityInStock INTO v_available_quantity
    FROM BOOK
    WHERE BookID = p_book_id;

    -- Check if requested quantity exceeds available quantity
    IF p_quantity > v_available_quantity THEN
        RAISE insufficient_quantity;
    END IF;

    -- Calculate total price after discount
    v_total_price := (v_price_per_unit * p_quantity) * (1 -
v_discount_percentage / 100);

    RETURN v_total_price;
EXCEPTION
    WHEN insufficient_quantity THEN
```

```
        DBMS_OUTPUT.PUT_LINE('Insufficient quantity in stock for the
requested book.');
        RETURN NULL;
    WHEN OTHERS THEN
        DBMS_OUTPUT.PUT_LINE('An error occurred: ' || SQLERRM);
        RETURN NULL;
END;
/


-- Example of executing the function to calculate discounted price for a
specific book and quantity
DECLARE
    v_book_id NUMBER := 1; -- Replace with the desired BookID
    v_quantity NUMBER := 5;   -- Replace with the desired quantity
    v_final_price NUMBER;
BEGIN
    v_final_price := calculate_discounted_price(v_book_id, v_quantity);
    DBMS_OUTPUT.PUT_LINE('Final Price after discount: ' || v_final_price);
END;
/


--Task5
--The purpose of the sale_insert_trigger trigger is to log the details of
inserted sales into the SALE_INSERT_LOG table for tracking purposes.
--Similarly, the sale_delete_trigger trigger logs the details of deleted
sales into the SALE_DELETE_LOG table for audit and tracking purposes.
-- Sequence for SALE_INSERT_LOG
CREATE SEQUENCE sale_insert_log_seq START WITH 1 INCREMENT BY 1;

-- Sequence for SALE_DELETE_LOG
CREATE SEQUENCE sale_delete_log_seq START WITH 1 INCREMENT BY 1;


-- Create SALE_INSERT_LOG table to track insertions
CREATE TABLE SALE_INSERT_LOG (
    LogID NUMBER(10, 0) PRIMARY KEY,
    SaleID NUMBER(10, 0),
    InsertionDate DATE
);

-- Create SALE_DELETE_LOG table to track deletions
CREATE TABLE SALE_DELETE_LOG (
    LogID NUMBER(10, 0) PRIMARY KEY,
    SaleID NUMBER(10, 0),
    DeletionDate DATE
);

-- Trigger to track insertions into the SALE table
CREATE OR REPLACE TRIGGER sale_insert_trigger
AFTER INSERT ON SALE
FOR EACH ROW
BEGIN
```

```
    INSERT INTO SALE_INSERT_LOG (LOGID, SaleID, InsertionDate)
    VALUES (sale_insert_log_seq.NEXTVAL, :NEW.SaleID, SYSDATE);
END;
/

-- Trigger to track deletions from the SALE table
CREATE OR REPLACE TRIGGER sale_delete_trigger
AFTER DELETE ON SALE
FOR EACH ROW
BEGIN
    INSERT INTO SALE_DELETE_LOG (LOGID, SaleID, DeletionDate)
    VALUES (sale_delete_log_seq.NEXTVAL, :OLD.SaleID, SYSDATE);
END;
/
--Statemnets to test it
INSERT INTO SALE VALUES (sale_seq.NEXTVAL, 5, 5, TO_DATE('2023-02-01',
'YYYY-MM-DD'));
DELETE FROM SALE WHERE SaleID = 7;

--Task9,4
--The purpose of the BILL object type is to calculate the total bill
amount for a list of purchased books based on their IDs and quantities, +
--and to provide a method for displaying the calculated bill amount.
-- Create the BILL object type
CREATE OR REPLACE TYPE BILL AS OBJECT (
    billID NUMBER,
    bookIDs SYS.ODCINUMBERLIST,
    quantities SYS.ODCINUMBERLIST,

    -- Function to calculate the total bill amount
    MEMBER FUNCTION calculateBill RETURN NUMBER,

    -- Procedure to display the bill amount
    MEMBER PROCEDURE displayBill
);
/

-- Create the body of the BILL object type
CREATE OR REPLACE TYPE BODY BILL AS
    -- Function to calculate the total bill amount
    MEMBER FUNCTION calculateBill RETURN NUMBER IS
        totalBill NUMBER := 0;
        bookPrice NUMBER;
    BEGIN
        -- Iterate over each book ID and quantity
        FOR i IN 1..bookIDs.COUNT LOOP
            BEGIN
                -- Retrieve the price of the book from the BOOK table
                SELECT Price INTO bookPrice FROM BOOK WHERE BookID =
bookIDs(i);

                -- Calculate the subtotal for the current book
                totalBill := totalBill + (bookPrice * quantities(i));
            EXCEPTION -- SYSTEM DEFINED EXCEPTION HANDLING
```

```
                WHEN NO_DATA_FOUND THEN
                    DBMS_OUTPUT.PUT_LINE('Book with ID ' ||
TO_CHAR(bookIDs(i)) || ' not found.');
            END;
        END LOOP;

        -- Return the total bill amount
        RETURN totalBill;
    END calculateBill;

    -- Procedure to display the bill amount
    MEMBER PROCEDURE displayBill IS
        totalBill NUMBER;
    BEGIN
        -- Calculate the total bill amount
        totalBill := self.calculateBill();

        -- Display the total bill amount
        DBMS_OUTPUT.PUT_LINE('Total Bill Amount: ' || TO_CHAR(totalBill));
    END displayBill;
END;
/


-- Declare variables to hold book IDs and quantities
DECLARE
    bookIDs SYS.ODCINUMBERLIST := SYS.ODCINUMBERLIST(5, 2, 3); -- Example
book IDs
    quantities SYS.ODCINUMBERLIST := SYS.ODCINUMBERLIST(2, 1, 3); --
Example quantities
    billObj BILL; -- Declare an object of type BILL
BEGIN
    -- Initialize the BILL object with book IDs and quantities
    billObj := BILL(1, bookIDs, quantities);

    -- Display the bill amount
    billObj.displayBill();
END;
/









-- Drop views
DROP VIEW vw_customer_names;
DROP VIEW vw_book_prices;
```

```
DROP VIEW vw_recent_sales;
DROP VIEW vw_employee_status;
DROP VIEW vw_high_value_customers;
DROP VIEW vw_highly_cost_books;
Drop VIEW vw_high_sales_books;
DROP VIEW vw_sales_and_customers

-- Drop indexes
DROP INDEX idx_sale_date;
DROP INDEX idx_genre_book;
DROP INDEX idx_quantity_in_stock;
DROP INDEX idx_price;
-- Drop tables
DROP TABLE RETURN;
DROP TABLE EMPLOYEE_SCHEDULE;
DROP TABLE PROMOTION;
DROP TABLE AUTHOR;
DROP TABLE BOOK_CATEGORY;
DROP TABLE CUSTOMER_PURCHASE_HISTORY;
DROP TABLE EMPLOYEE_ACCESS_LOG;
DROP TABLE CUSTOMER_PREFERENCES;
DROP TABLE EMPLOYEE_TRAINING;
DROP TABLE BOOK_SHELF;
DROP TABLE SALES_PROMOTION_LOG;
DROP TABLE CUSTOMER_FEEDBACK;
DROP TABLE PURCHASE_ORDER_ITEM;
DROP TABLE PURCHASE_ORDER;
DROP TABLE SUPPLIER;
DROP TABLE REPORT;
DROP TABLE SALE_ITEM;
DROP TABLE SALE;
DROP TABLE BOOK;
DROP TABLE EMPLOYEE;
DROP TABLE CUSTOMER;
DROP TABLE GENRE;
DROP TABLE CUSTOMER_FEEDBACK_ARCHIVE;
DROP TABLE SALE_INSERT_LOG;
DROP TABLE SALE_DELETE_LOG;
-- Drop sequences
-- Drop sequences for CUSTOMER table
DROP SEQUENCE customer_seq;
-- Drop sequences for BOOK table
DROP SEQUENCE book_seq;
-- Drop sequences for EMPLOYEE table
DROP SEQUENCE employee_seq;
-- Drop sequences for SALE table
DROP SEQUENCE sale_seq;
-- Drop sequences for SALE_ITEM table
DROP SEQUENCE sale_item_seq;
-- Drop sequences for REPORT table
DROP SEQUENCE report_seq;
-- Drop sequences for SUPPLIER table
DROP SEQUENCE supplier_seq;
-- Drop sequences for PURCHASE_ORDER table
```

```
DROP SEQUENCE purchase_order_seq;
-- Drop sequences for PURCHASE_ORDER_ITEM table
DROP SEQUENCE purchase_order_item_seq;
-- Drop sequences for RETURN table
DROP SEQUENCE return_seq;
-- Drop sequences for EMPLOYEE_SCHEDULE table
DROP SEQUENCE employee_schedule_seq;
-- Drop sequences for PROMOTION table
DROP SEQUENCE promotion_seq;
-- Drop sequences for AUTHOR table
DROP SEQUENCE author_seq;
-- Drop sequences for GENRE table
DROP SEQUENCE genre_seq;
-- Drop sequences for BOOK_CATEGORY table
DROP SEQUENCE book_category_seq;
-- Drop sequences for CUSTOMER_PURCHASE_HISTORY table
DROP SEQUENCE customer_purchase_history_seq;
-- Drop sequences for EMPLOYEE_ACCESS_LOG table
DROP SEQUENCE employee_access_log_seq;
-- Drop sequences for CUSTOMER_PREFERENCES table
DROP SEQUENCE customer_preferences_seq;
-- Drop sequences for EMPLOYEE_TRAINING table
DROP SEQUENCE employee_training_seq;
-- Drop sequences for BOOK_SHELF table
DROP SEQUENCE book_shelf_seq;
-- Drop sequences for SALES_PROMOTION_LOG table
DROP SEQUENCE sales_promotion_log_seq;
-- Drop sequences for CUSTOMER_FEEDBACK table
DROP SEQUENCE customer_feedback_seq;
DROP SEQUENCE sale_insert_log_seq;
DROP SEQUENCE sale_delete_log_seq;
```