AMERICAN INTERNATIONAL UNIVERSITY–BANGLADESH (AIUB)

Summer 2024-25

Final Term Project Doccumentation

BOOK STORE MANAGEMENT SYSTEM

PROJECT REPORT
Advanced Database Management System
SECTION : A

NAME	ID	CONTRIBUTION
Mohammad Rafsan Yeasir	22-46065-1	30
MD.Fardin	21-45603-3	20
Asikqur Rahman	22-48971-3	20
Md. Asaduzzaman Rony	22-49057-3	30

Contents:

Based on Mid Term Feedback solution

Scenario updated

A bookstore deals with multiple operations, including book inventory, customer purchases, employee management, and payment handling. Without a proper system, manual processes cause stock mismatches, difficulties in tracking sales, and delays in customer service.

The **Bookstore Management System** solves these issues by storing and managing data about:

- **Books** (title, author, price, stock, category, publisher, supplier)
- **Customers** (personal details and order history)
- **Employees** (who handle orders)
- **Orders and OrderItems** (which track purchased books, quantity, subtotal)
- **Payments** (amount, method, date linked with orders)

When a customer places an order, it records the order date, books purchased, employee handling the sale, and total price. Stock levels are automatically updated. Payments are then recorded with method and date. This ensures smooth inventory management, accurate order handling, and reliable reporting for decision-making.

Corrected Normalization (Updated)

Belongs to (Books – Category)

UNF:
(BookID, Title, Author, Price, Stock, CategoryID, CategoryName)

1NF:
(BookID, Title, Author, Price, Stock, CategoryID, CategoryName)

2NF:

1. (BookID, Title, Author, Price, Stock, CategoryID)
2. (CategoryID, CategoryName)

3NF:

- Already in 3NF (no transitive dependency).

Final Tables:

1. Books(BookID, Title, Author, Price, Stock, CategoryID)
2. Category(CategoryID, CategoryName)

Publishes (Books – Publisher)

UNF:
(BookID, Title, Author, Price, Stock, PublisherID, PublisherName, Address, Phone, Email)

1NF:
(BookID, Title, Author, Price, Stock, PublisherID, PublisherName, Address, Phone, Email)

2NF:

1. (BookID, Title, Author, Price, Stock, PublisherID)
2. (PublisherID, PublisherName, Address, Phone, Email)

3NF:

- No transitive dependency, already in 3NF.

Final Tables:

1. Books(BookID, Title, Author, Price, Stock, PublisherID)
2. Publisher(PublisherID, PublisherName, Address, Phone, Email)

Supplies (Books – Supplier)

UNF:
(BookID, Title, Author, Price, Stock, SupplierID, SupplierName, ContactPerson, Phone, Email, Address)

1NF:
(BookID, Title, Author, Price, Stock, SupplierID, SupplierName, ContactPerson, Phone, Email, Address)

2NF:

1. (BookID, Title, Author, Price, Stock, SupplierID)

2. (SupplierID, SupplierName, ContactPerson, Phone, Email, Address)

3NF:

- No transitive dependency.

Final Tables:

1. Books(BookID, Title, Author, Price, Stock, SupplierID)
2. Supplier(SupplierID, SupplierName, ContactPerson, Phone, Email, Address)

Appears in (Books – OrderItem)

UNF:
(BookID, Title, Author, Price, Stock, OrderItemID, Quantity, SubTotal)

1NF:
(BookID, Title, Author, Price, Stock, OrderItemID, Quantity, SubTotal)

2NF:

1. (OrderItemID, BookID, Quantity, SubTotal)
2. (BookID, Title, Author, Price, Stock)

3NF:

- Book details moved to Books table, Order details remain separate.

Final Tables:

1. Books(BookID, Title, Author, Price, Stock)
2. OrderItem(OrderItemID, BookID, Quantity, SubTotal)

Contains (Orders – OrderItem)

UNF:
(OrderID, OrderDate, TotalAmount, OrderItemID, Quantity, SubTotal)

1NF:
(OrderID, OrderDate, TotalAmount, OrderItemID, Quantity, SubTotal)

2NF:

1. (OrderID, OrderDate, TotalAmount)

2. (OrderItemID, Quantity, SubTotal)

3NF:

- Already in 3NF.

Final Tables:

1. Orders(OrderID, OrderDate, TotalAmount)
2. OrderItem(OrderItemID, OrderID, Quantity, SubTotal)

Handles (Orders – Employee)

UNF:
(OrderID, OrderDate, TotalAmount, EmployeeID, EmployeeName, Role, Email, Phone)

1NF:
(OrderID, OrderDate, TotalAmount, EmployeeID, EmployeeName, Role, Email, Phone)

2NF:

1. (OrderID, OrderDate, TotalAmount, EmployeeID)
2. (EmployeeID, EmployeeName, Role, Email, Phone)

3NF:

- Already in 3NF.

Final Tables:

1. Orders(OrderID, OrderDate, TotalAmount, EmployeeID)
2. Employee(EmployeeID, EmployeeName, Role, Email, Phone)

Places (Orders – Customer)

UNF:
(OrderID, OrderDate, TotalAmount, CustomerID, CustomerName, Email, Phone, Address)

1NF:
(OrderID, OrderDate, TotalAmount, CustomerID, CustomerName, Email, Phone, Address)

2NF:

1. (OrderID, OrderDate, TotalAmount, CustomerID)

2. (CustomerID, CustomerName, Email, Phone, Address)

3NF:

- Already in 3NF.

Final Tables:

1. Orders(OrderID, OrderDate, TotalAmount, CustomerID)
2. Customer(CustomerID, CustomerName, Email, Phone, Address)

Paid (Orders – Payment)

UNF:
(OrderID, OrderDate, TotalAmount, PaymentID, Amount, PaymentDate, Method)

1NF:
(OrderID, OrderDate, TotalAmount, PaymentID, Amount, PaymentDate, Method)

2NF:

1. (OrderID, OrderDate, TotalAmount)
2. (PaymentID, Amount, PaymentDate, Method, OrderID)

3NF:

- Already in 3NF.

Final Tables:

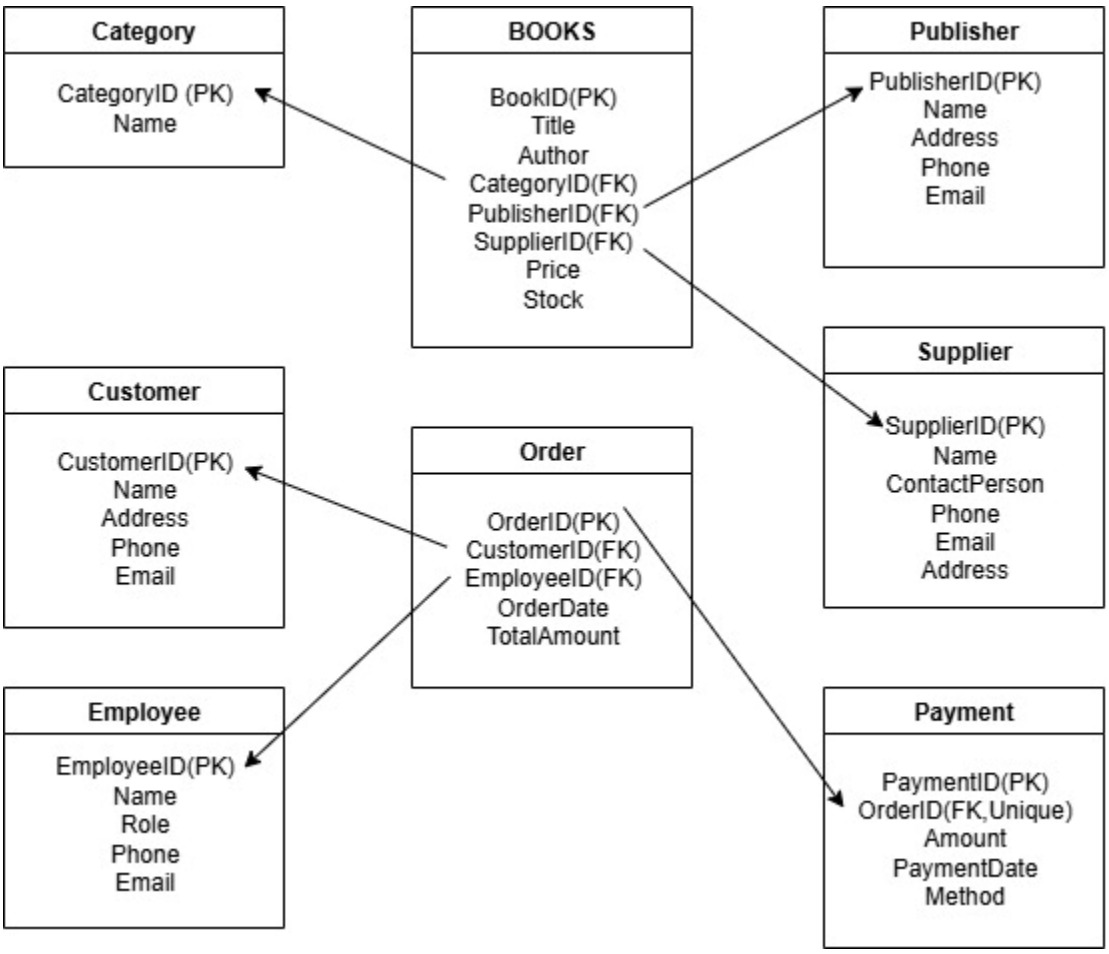
1. Orders(OrderID, OrderDate, TotalAmount)
2. Payment(PaymentID, OrderID, Amount, PaymentDate, Method)

Final Normalized Tables (Consolidated)

1. Books(BookID, Title, Author, Price, Stock, CategoryID FK, PublisherID FK, SupplierID FK)
2. Category(CategoryID, CategoryName)
3. Publisher(PublisherID, PublisherName, Address, Phone, Email)
4. Supplier(SupplierID, SupplierName, ContactPerson, Phone, Email, Address)
5. OrderItem(OrderItemID, OrderID FK, BookID FK, Quantity, SubTotal)

6. Orders(OrderID, OrderDate, TotalAmount, CustomerID FK, EmployeeID FK)
7. Payment(PaymentID, OrderID FK, Amount, PaymentDate, Method)
8. Customer(CustomerID, CustomerName, Email, Phone, Address)
9. Employee(EmployeeID, EmployeeName, Role, Email, Phone)

Schema Diagram: (updated)

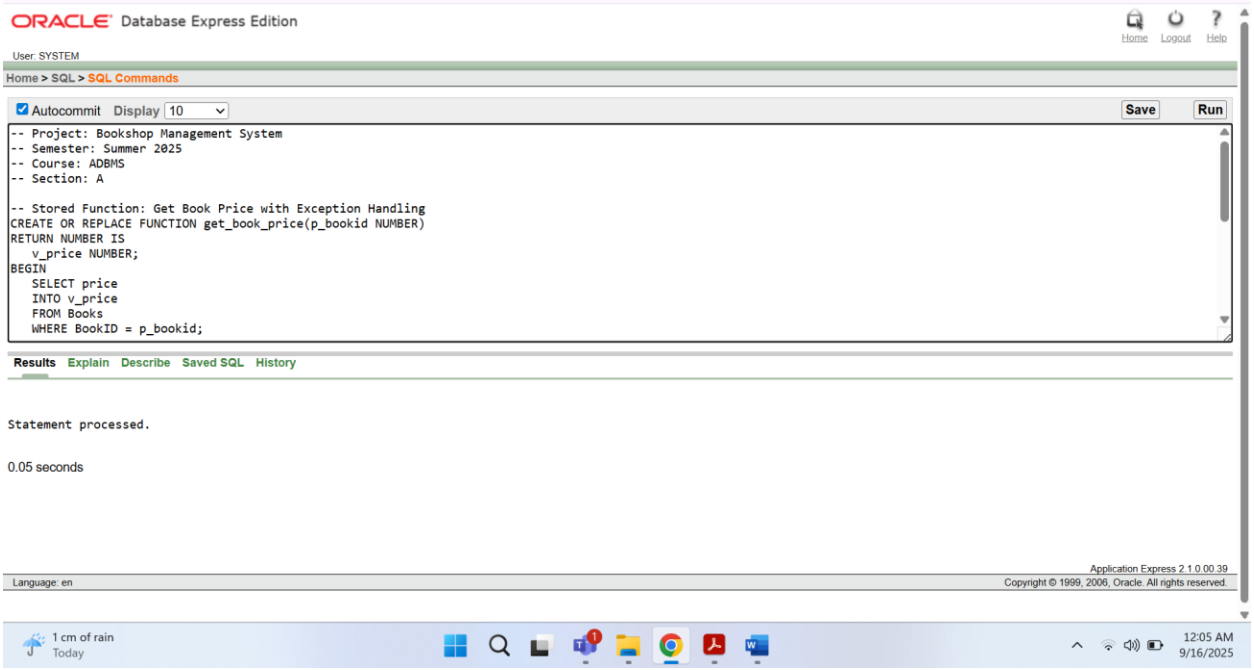


Advance PL/SQL With Exception Handling

Stored function:

```
-- Project: Bookshop Management System
-- Semester: Summer 2025
-- Course: ADBMS
-- Section: A

-- Stored Function: Get Book Price with Exception Handling
CREATE OR REPLACE FUNCTION get_book_price(p_bookid NUMBER)
RETURN NUMBER IS
  v_price NUMBER;
BEGIN
  SELECT price
  INTO v_price
  FROM Books
  WHERE BookID = p_bookid;

  RETURN v_price;

EXCEPTION
  WHEN NO_DATA_FOUND THEN
    DBMS_OUTPUT.PUT_LINE('No book found with ID: ' || p_bookid);
    RETURN NULL;  -- or you could RETURN -1 as an indicator
  WHEN TOO_MANY_ROWS THEN
    DBMS_OUTPUT.PUT_LINE('Multiple books found with ID: ' || p_bookid);
    RETURN NULL;
  WHEN OTHERS THEN
```

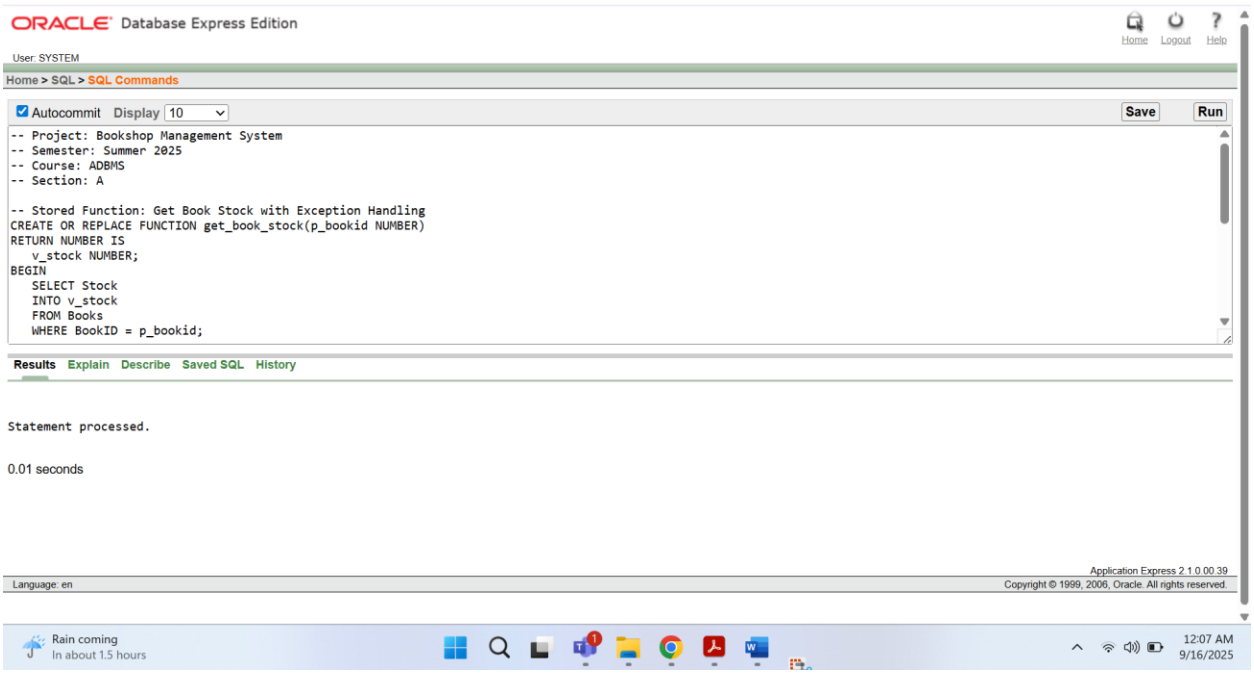
```
    DBMS_OUTPUT.PUT_LINE('Unexpected error: ' || SQLERRM);
    RETURN NULL;
END;
/
```



Stored function:

```
-- Project: Bookshop Management System
-- Semester: Summer 2025
-- Course: ADBMS
-- Section: A

-- Stored Function: Get Book Stock with Exception Handling
CREATE OR REPLACE FUNCTION get_book_stock(p_bookid NUMBER)
RETURN NUMBER IS
  v_stock NUMBER;
BEGIN
  SELECT Stock
```

```
  INTO v_stock
  FROM Books
  WHERE BookID = p_bookid;

  RETURN v_stock;

EXCEPTION
  WHEN NO_DATA_FOUND THEN
    DBMS_OUTPUT.PUT_LINE('No book found with ID: ' || p_bookid);
    RETURN -1;  -- return -1 to indicate missing book
  WHEN TOO_MANY_ROWS THEN
    DBMS_OUTPUT.PUT_LINE('Data error: multiple books found with ID: ' || p_bookid);
    RETURN -2;  -- return -2 to indicate data issue
  WHEN OTHERS THEN
    DBMS_OUTPUT.PUT_LINE('Unexpected error: ' || SQLERRM);
    RETURN -99; -- generic error code
END;
/
```



stored procedure :

-- Stored Procedure: Calculate Total Sales

-- Project: Bookshop Management System

-- Semester: Summer 2025

-- Course: ADBMS

-- Section: A

```
CREATE OR REPLACE PROCEDURE get_total_sales(p_total OUT NUMBER)
IS
BEGIN
    SELECT NVL(SUM(TotalAmount), 0)
    INTO p_total
    FROM Orders;

EXCEPTION
```

```
    WHEN NO_DATA_FOUND THEN
        DBMS_OUTPUT.PUT_LINE('No sales data found.');
        p_total := 0;
    WHEN OTHERS THEN
        DBMS_OUTPUT.PUT_LINE('Unexpected error: ' || SQLERRM);
        p_total := -1; -- error indicator
END;
/
```

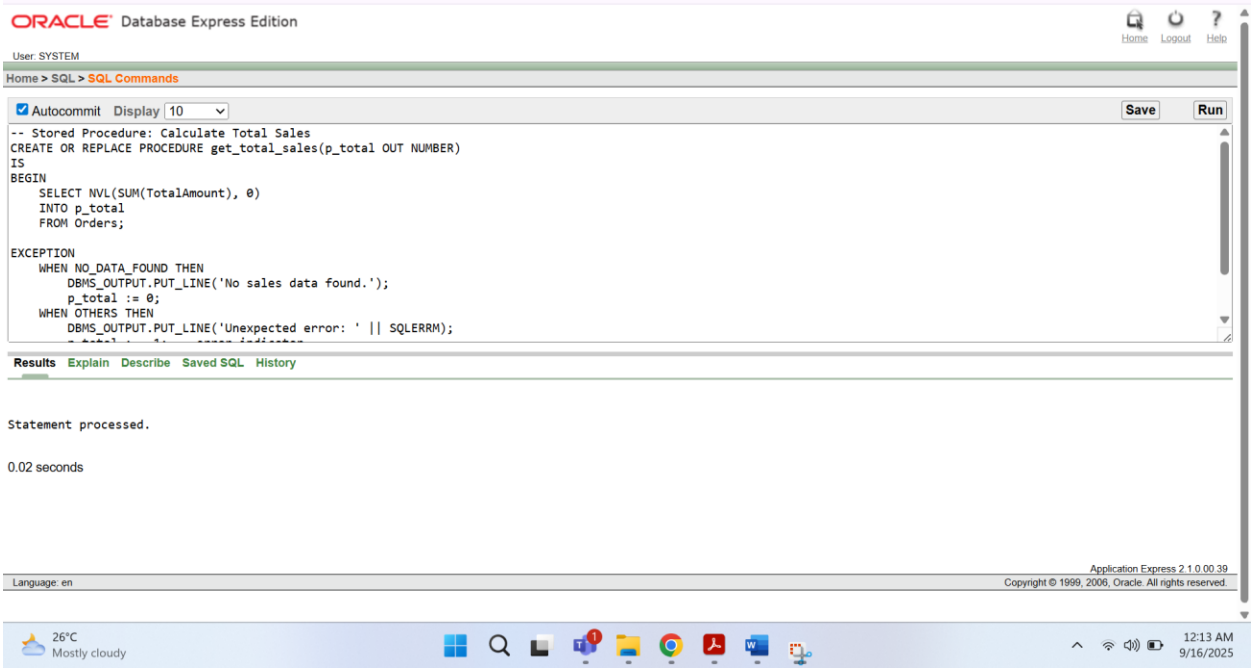
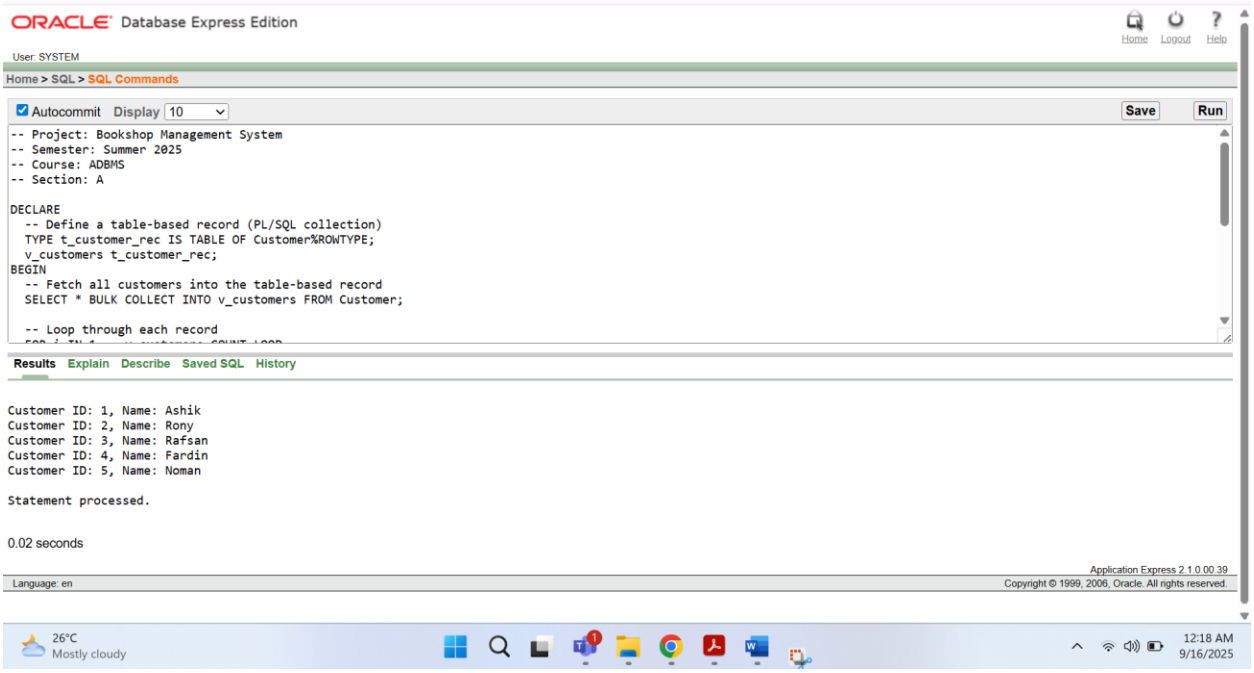


Table-based record :

```
-- Project: Bookshop Management System
-- Semester: Summer 2025
-- Course: ADBMS
-- Section: A
```

```
DECLARE
  -- Define a table-based record (PL/SQL collection)
  TYPE t_customer_rec IS TABLE OF Customer%ROWTYPE;
  v_customers t_customer_rec;
BEGIN
  -- Fetch all customers into the table-based record
  SELECT * BULK COLLECT INTO v_customers FROM Customer;

  -- Loop through each record
  FOR i IN 1 .. v_customers.COUNT LOOP
    DBMS_OUTPUT.PUT_LINE('Customer ID: ' || v_customers(i).CustomerID ||
                         ', Name: ' || v_customers(i).Name);
  END LOOP;

EXCEPTION
  WHEN NO_DATA_FOUND THEN
    DBMS_OUTPUT.PUT_LINE('No customers found.');
  WHEN TOO_MANY_ROWS THEN
    DBMS_OUTPUT.PUT_LINE('More rows than expected.');
  WHEN OTHERS THEN
    DBMS_OUTPUT.PUT_LINE('Error: ' || SQLERRM);
END;
/
```



Explicit Cursor :

-- Project: Bookshop Management System

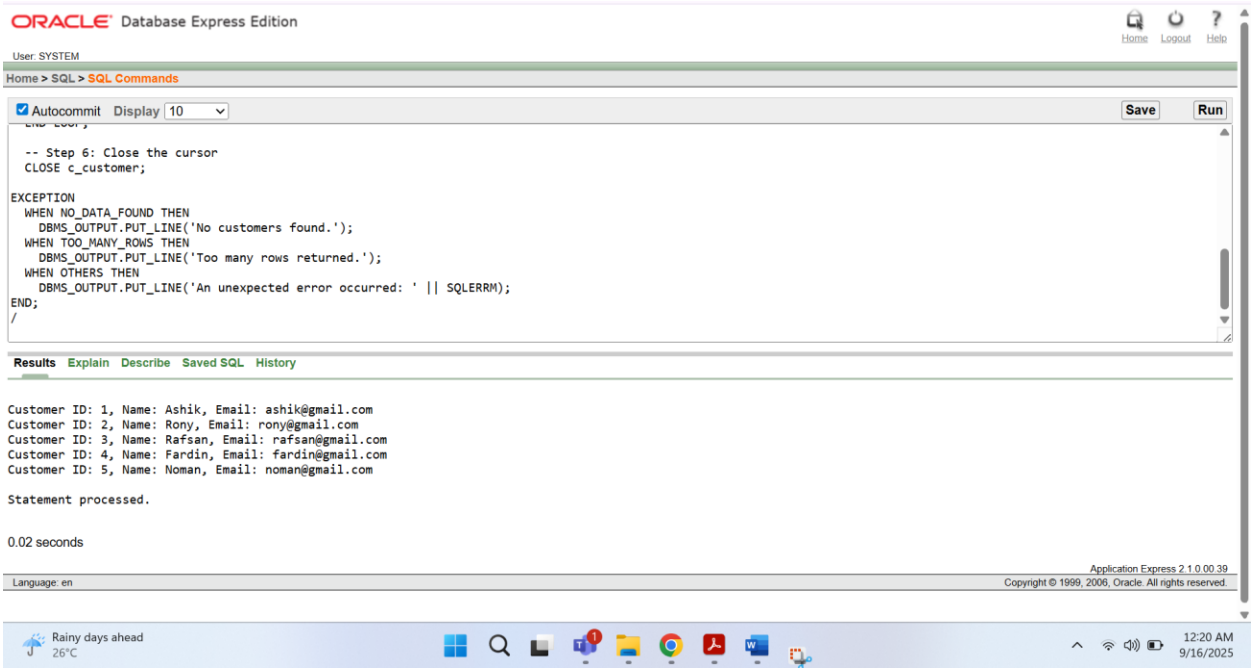
-- Semester: Summer 2025

-- Course: ADBMS

-- Section: A

```
DECLARE
  -- Step 1: Declare the explicit cursor
  CURSOR c_customer IS
    SELECT CustomerID, Name, Email FROM Customer;

  -- Step 2: Declare a record to hold each row fetched
  v_customer c_customer%ROWTYPE;
BEGIN
```

```
  -- Step 3: Open the cursor
  OPEN c_customer;

  -- Step 4: Fetch rows in a loop
  LOOP
    FETCH c_customer INTO v_customer;
    EXIT WHEN c_customer%NOTFOUND;  -- Exit loop when no more rows

    -- Step 5: Process each row
    DBMS_OUTPUT.PUT_LINE('Customer ID: ' || v_customer.CustomerID ||
                         ', Name: ' || v_customer.Name ||
                         ', Email: ' || v_customer.Email);
  END LOOP;

  -- Step 6: Close the cursor
  CLOSE c_customer;

EXCEPTION
  WHEN NO_DATA_FOUND THEN
    DBMS_OUTPUT.PUT_LINE('No customers found.');
  WHEN TOO_MANY_ROWS THEN
    DBMS_OUTPUT.PUT_LINE('Too many rows returned.');
  WHEN OTHERS THEN
    DBMS_OUTPUT.PUT_LINE('An unexpected error occurred: ' || SQLERRM);
END;
/
```



Cursor-based record :

-- Project: Bookshop Management System

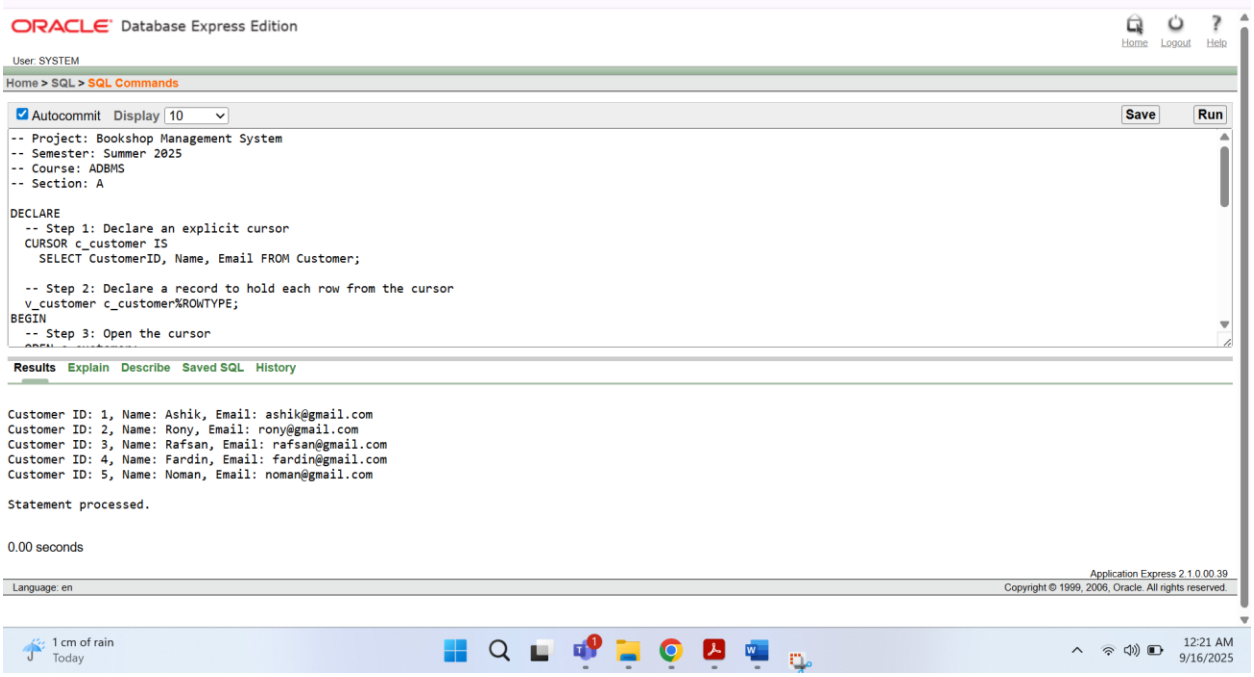
-- Semester: Summer 2025

-- Course: ADBMS

-- Section: A

```
DECLARE
  -- Step 1: Declare an explicit cursor
  CURSOR c_customer IS
    SELECT CustomerID, Name, Email FROM Customer;

  -- Step 2: Declare a record to hold each row from the cursor
  v_customer c_customer%ROWTYPE;
BEGIN
```

```
  -- Step 3: Open the cursor
  OPEN c_customer;

  -- Step 4: Fetch each row into the record and process
  LOOP
    FETCH c_customer INTO v_customer;
    EXIT WHEN c_customer%NOTFOUND;  -- Exit loop when no more rows

    -- Step 5: Display the row data
    DBMS_OUTPUT.PUT_LINE('Customer ID: ' || v_customer.CustomerID ||
                         ', Name: ' || v_customer.Name ||
                         ', Email: ' || v_customer.Email);
  END LOOP;

  -- Step 6: Close the cursor
  CLOSE c_customer;

EXCEPTION
  WHEN NO_DATA_FOUND THEN
    DBMS_OUTPUT.PUT_LINE('No customers found.');
  WHEN TOO_MANY_ROWS THEN
    DBMS_OUTPUT.PUT_LINE('Too many rows returned.');
  WHEN OTHERS THEN
    DBMS_OUTPUT.PUT_LINE('An unexpected error occurred: ' || SQLERRM);
END;
/
```



Row Level Trigger:

-- Project: Bookshop Management System

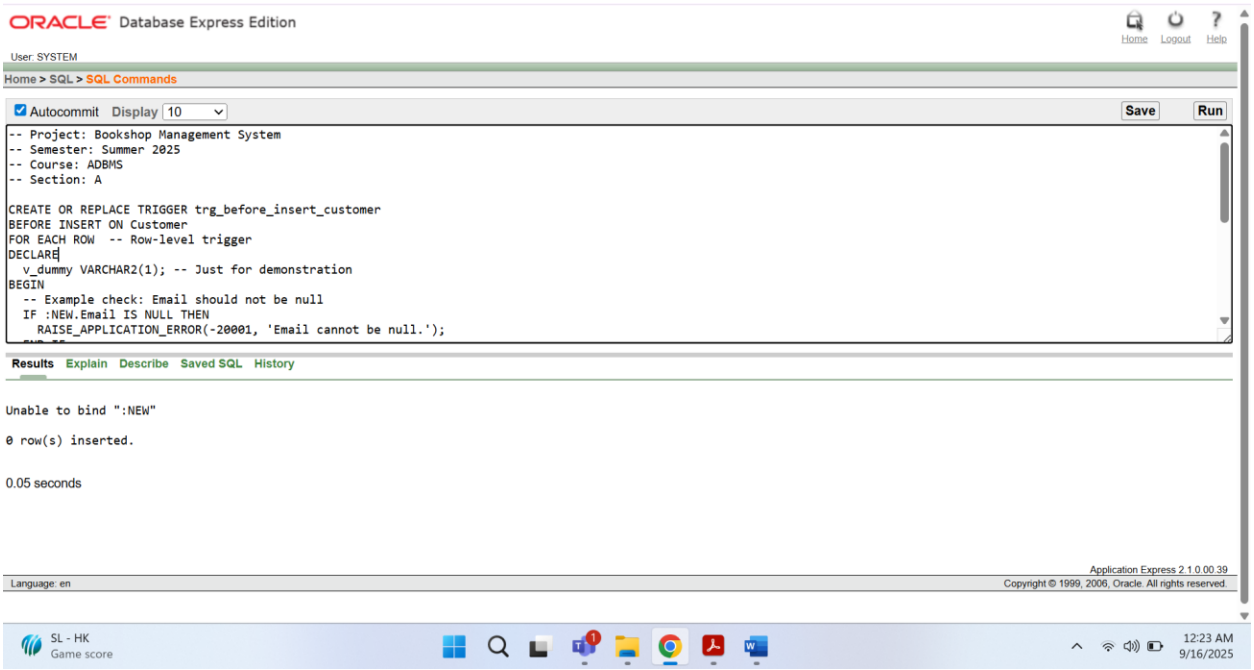
-- Semester: Summer 2025

-- Course: ADBMS

-- Section: A

```
CREATE OR REPLACE TRIGGER trg_before_insert_customer
BEFORE INSERT ON Customer
FOR EACH ROW  -- Row-level trigger
DECLARE
  v_dummy VARCHAR2(1); -- Just for demonstration
BEGIN
  -- Example check: Email should not be null
  IF :NEW.Email IS NULL THEN
    RAISE_APPLICATION_ERROR(-20001, 'Email cannot be null.');
```

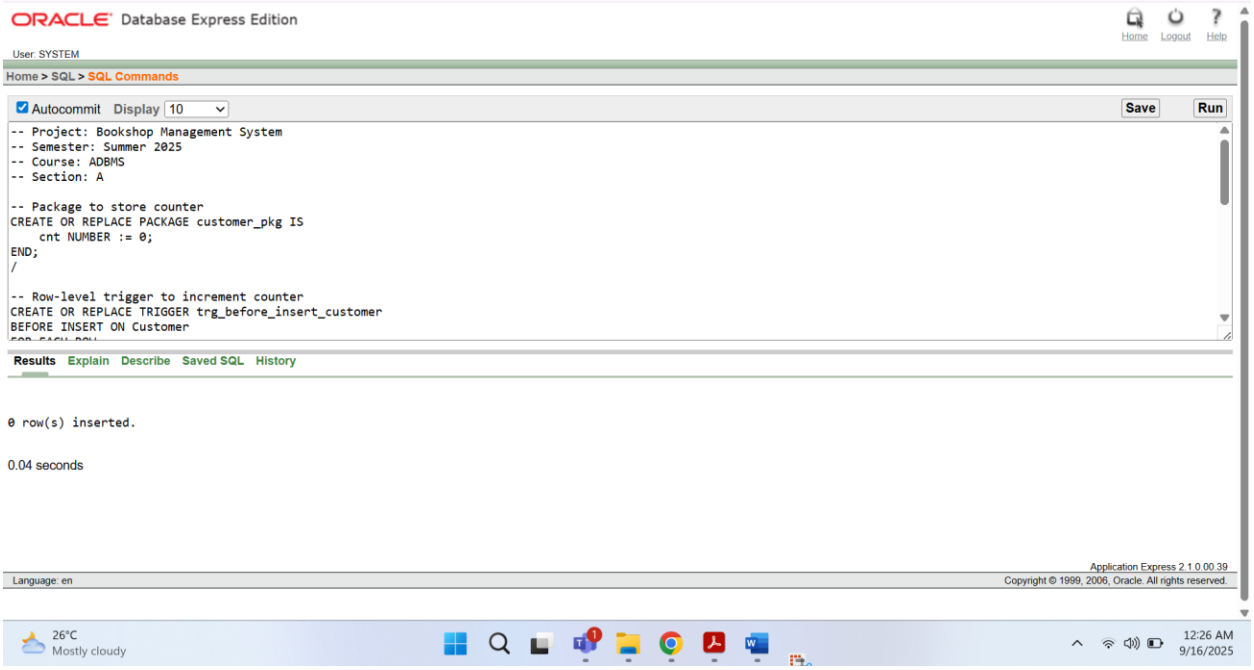
```
  END IF;

  -- Example: Assign a default value if Name is null
  IF :NEW.Name IS NULL THEN
    :NEW.Name := 'Unknown';
  END IF;

EXCEPTION
  WHEN OTHERS THEN
    -- Handle any unexpected error
    DBMS_OUTPUT.PUT_LINE('Error in trigger: ' || SQLERRM);
    -- Optionally, re-raise the error to stop the DML
    RAISE;
END;
/
```



Statement level trigger:

```
-- Project: Bookshop Management System
-- Semester: Summer 2025
-- Course: ADBMS
-- Section: A

-- Package to store counter
CREATE OR REPLACE PACKAGE customer_pkg IS
   cnt NUMBER := 0;
END;
/

-- Row-level trigger to increment counter
CREATE OR REPLACE TRIGGER trg_before_insert_customer
BEFORE INSERT ON Customer
FOR EACH ROW
BEGIN
   customer_pkg.cnt := customer_pkg.cnt + 1;
END;
/

-- Statement-level trigger to log count
CREATE OR REPLACE TRIGGER trg_after_insert_customer
AFTER INSERT ON Customer
DECLARE
BEGIN
   INSERT INTO Customer_Audit(Audit_Date, Total_Customers)
```

```
    VALUES (SYSDATE, customer_pkg.cnt);


    -- Reset counter
    customer_pkg.cnt := 0;
EXCEPTION
    WHEN OTHERS THEN
        DBMS_OUTPUT.PUT_LINE('Error in statement-level trigger: ' || SQLERRM);
        RAISE;
END;
/
```



Package:

--Package Specification

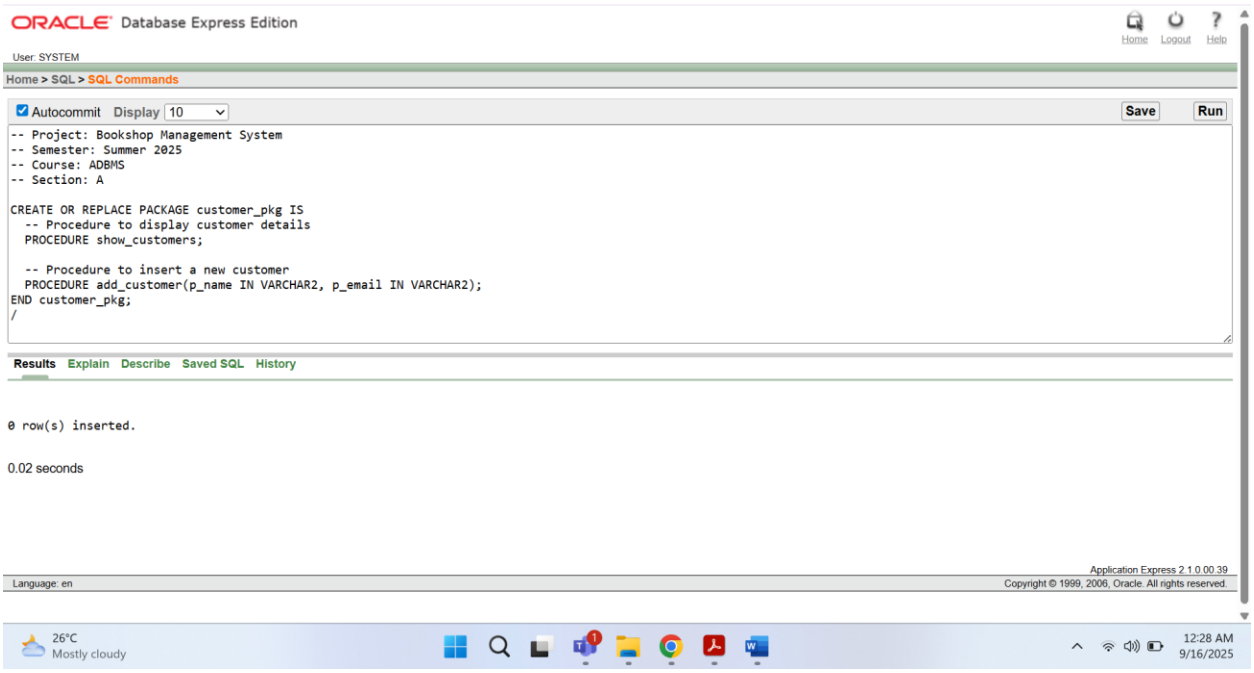
-- Project: Bookshop Management System

-- Semester: Summer 2025

-- Course: ADBMS

-- Section: A

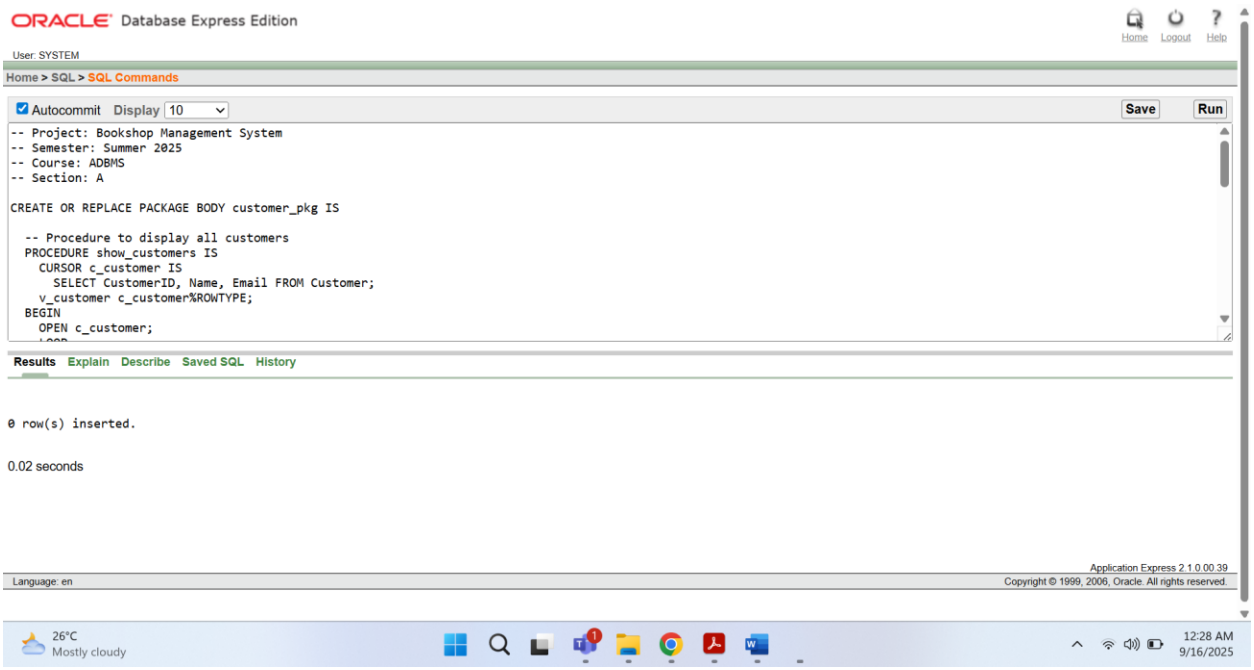
```
CREATE OR REPLACE PACKAGE customer_pkg IS
  -- Procedure to display customer details
  PROCEDURE show_customers;

  -- Procedure to insert a new customer
  PROCEDURE add_customer(p_name IN VARCHAR2, p_email IN VARCHAR2);
END customer_pkg;
/
```




```
--BODY
-- Project: Bookshop Management System
-- Semester: Summer 2025
-- Course: ADBMS
-- Section: A
```

```
CREATE OR REPLACE PACKAGE BODY customer_pkg IS

  -- Procedure to display all customers
  PROCEDURE show_customers IS
    CURSOR c_customer IS
      SELECT CustomerID, Name, Email FROM Customer;
    v_customer c_customer%ROWTYPE;
  BEGIN
    OPEN c_customer;
    LOOP
      FETCH c_customer INTO v_customer;
      EXIT WHEN c_customer%NOTFOUND;

      DBMS_OUTPUT.PUT_LINE('Customer ID: ' || v_customer.CustomerID ||
                           ', Name: ' || v_customer.Name ||
                           ', Email: ' || v_customer.Email);
    END LOOP;
    CLOSE c_customer;

  EXCEPTION
    WHEN NO_DATA_FOUND THEN
      DBMS_OUTPUT.PUT_LINE('No customers found.');
    WHEN OTHERS THEN
      DBMS_OUTPUT.PUT_LINE('Error in show_customers: ' || SQLERRM);
  END show_customers;
```

```
  -- Procedure to add a new customer
  PROCEDURE add_customer(p_name IN VARCHAR2, p_email IN VARCHAR2) IS
  BEGIN
    INSERT INTO Customer(Name, Email)
    VALUES (p_name, p_email);

    DBMS_OUTPUT.PUT_LINE('Customer added successfully!');

  EXCEPTION
    WHEN DUP_VAL_ON_INDEX THEN
      DBMS_OUTPUT.PUT_LINE('Error: Customer with this email already exists.');
    WHEN VALUE_ERROR THEN
      DBMS_OUTPUT.PUT_LINE('Error: Invalid data provided.');
    WHEN OTHERS THEN
      DBMS_OUTPUT.PUT_LINE('Error in add_customer: ' || SQLERRM);
  END add_customer;

END customer_pkg;
/
```



Relational Algebra Questions and Answers:

1. Question:
Find the names of all customers who have purchased a book priced over 500.

Relations:

- Customer(CustID, Name)
- Purchase(CustID, BookID)
- Book(BookID, Title, Price)

Answer:
π_Name (Customer ⋈ (σ_Price>500 Book ⋈ Purchase))

2. Question:
List the titles of books that have not been purchased by any customer.

Relations:

- Book(BookID, Title)
- Purchase(CustID, BookID)

Answer:
π_Title(Book) - π_Title(Book ⋈ Purchase)

3. Question:
Find the IDs of customers who purchased all books in the category “Fiction”.

Relations:

- Customer(CustID, Name)
- Purchase(CustID, BookID)
- Book(BookID, Title, Category)

Answer:
π_CustID(Customer) ÷ π_BookID(σ_Category='Fiction' Book)

4. Question:
Retrieve the names of customers who purchased both book with BookID = 101 and BookID = 102.

Relations:

- Customer(CustID, Name)
- Purchase(CustID, BookID)

Answer:
π_Name (Customer ⋈ (π_CustID(σ_BookID=101 Purchase) ∩ π_CustID(σ_BookID=102 Purchase)))

5. Question:
Find the titles and prices of books purchased by customer named “Alice”.

Relations:

- Customer(CustID, Name)
- Purchase(CustID, BookID)
- Book(BookID, Title, Price)

Answer:
π_Title, Price (σ_Name='Alice' Customer ⋈ Purchase ⋈ Book)