



**AMERICAN INTERNATIONAL UNIVERSITY BANGLADESH-(AIUB)**

**Department of Computer Science & Engineering**

**Course: ADVANCE DATABASE MANAGEMENT SYSTEM**

**SUMMER 2024-2025**

**Section : A**

**Final Term Project**

**Project Report On : MALL MANAGEMENT SYSTEM**

**Supervised By**

**Faculty: JUENA AHMED NOSHIN**

**Submitted By**

**Project Member:**

<b>NAME</b>	<b>ID</b>	<b>Contribution Percentage</b>
Sumaiya Tasnim	23-50014-1	<b>100%</b>

**Submission Deadline : 16-09-2025**

## TABLE OF CONTENTS

No	CHAPTERS	PAGES
1	Introduction	01
2	Project Proposal	01-02
3	User Interface Planning	03-09
4	Scenerio Description	09-10
5	ER Diagram	10
6	Normalization	11-17
7	Schema Diagram	18
8	Table Creation Using SQL	19-28
9	Data Insertion	29-35
10	Query Writing Using PL/SQL	37
11	Basic PL/SQL	37-52
12	Advance PL/SQL (With Exception Handling)	52-78
13	Relational Algebra	79
14	User Interface Design To Code Implementation	80-
15	Oracle 10g Database Connection Process	
16	Conclusion	

# “Mall Management System”

## 1.Introduction

The ***Mall Management System*** is a database-driven application designed to streamline the daily operations of a shopping mall. From managing shops and shopkeepers to handling customer data, inventory, billing, and maintenance records, the system provides a centralized and structured way to organize all essential information. It helps reduce manual errors, saves time, and ensures that operations run smoothly and efficiently. Malls typically generate and handle large amounts of data every day. Without a proper system in place, this can quickly become overwhelming. The Mall Management System addresses this challenge by offering an easy-to-use interface backed by a well-designed database that ensures secure data storage, quick access, and consistent updates. This project highlights the practical use of sound database design principles and demonstrates how technology can simplify complex real-world processes, improve coordination, and support better management decisions.

## 2.Project Proposal

### Objectives

- To create a system that helps manage mall activities in an organized way.
- To reduce manual work and avoid mistakes in storing and updating data.
- To keep all shop, customer, and inventory details in one place for easy access.
- To make billing and reporting faster and more accurate.
- To support mall staff in managing shops, customers, and stock more smoothly.

### Problem Statement

Managing a shopping mall involves handling a lot of information like shop details, customer records, inventory, and billing. Doing all of this manually can take a lot of time and may lead to mistakes or missing data. It also becomes hard to find or update information quickly. Because of this, mall operations can slow down and become less efficient. A proper system is needed to store all data in one place, reduce manual work, and help the mall run more smoothly.

## Methodology

We will develop the project through a clear step-by-step approach focused on database design and implementation:

**Requirement Analysis:** We will study the mall management needs to identify what data should be stored and managed.

**Database Design:** We will create an Entity-Relationship (ER) diagram to organize data and define relationships between entities such as shops, customers, and inventory.

**Database Implementation:** We will build the database using Oracle 10g, creating tables, keys, and constraints to maintain data integrity.

**Query Development:** We will write SQL queries, stored procedures, and triggers to handle data operations like inserting, updating, deleting, and generating reports.

**Documentation:** We will prepare clear documentation detailing the database schema, queries, and usage instructions.

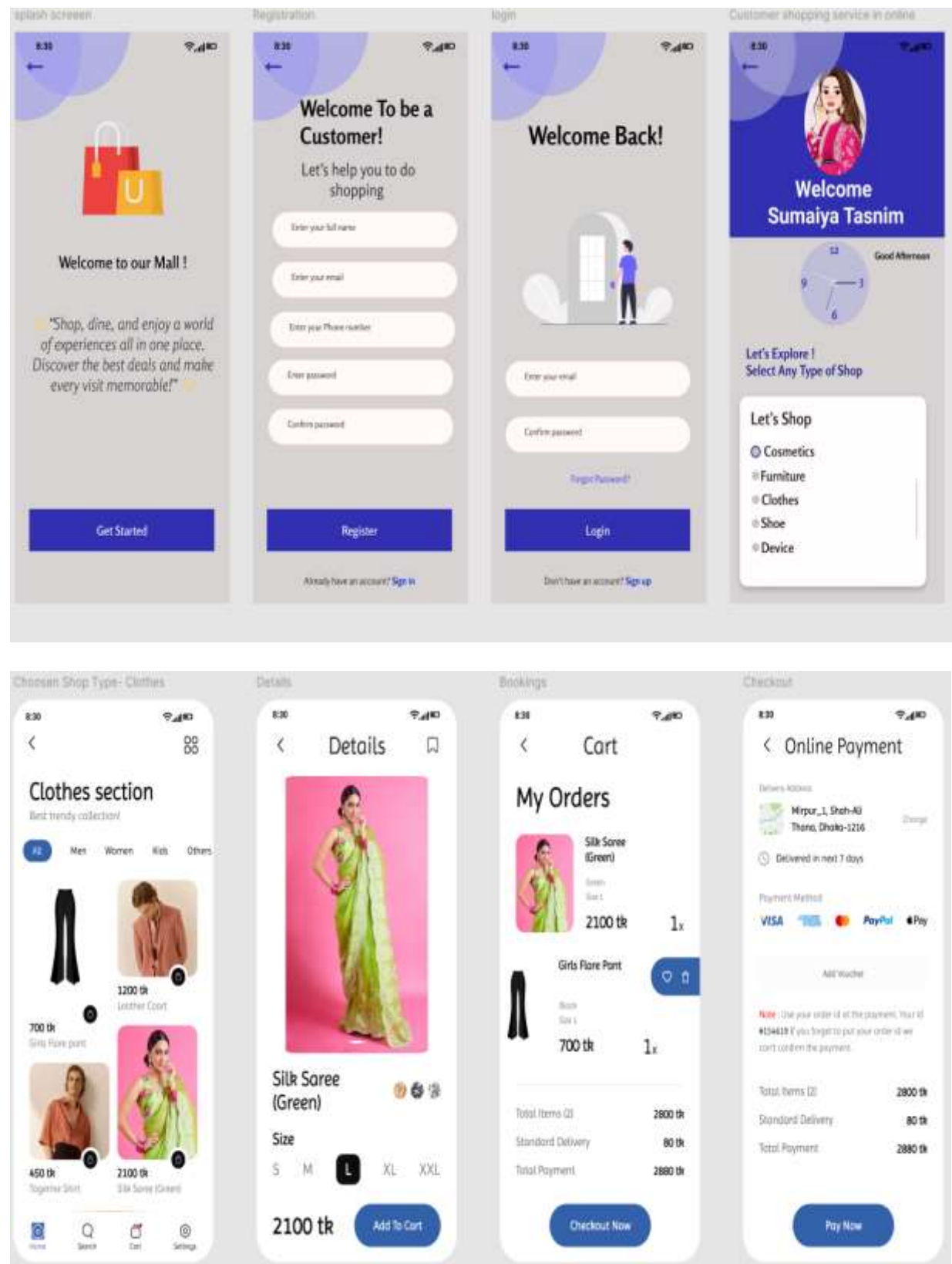
## System Features

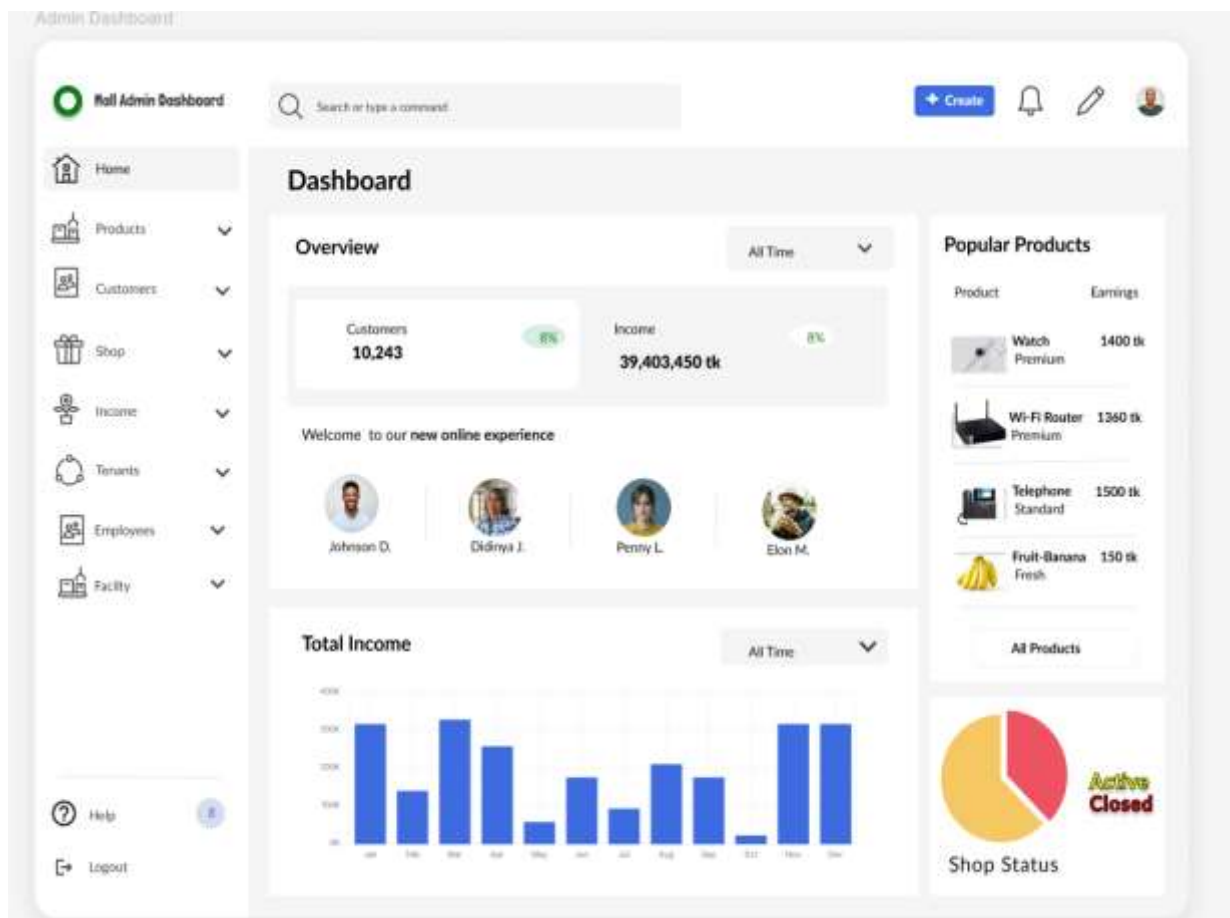
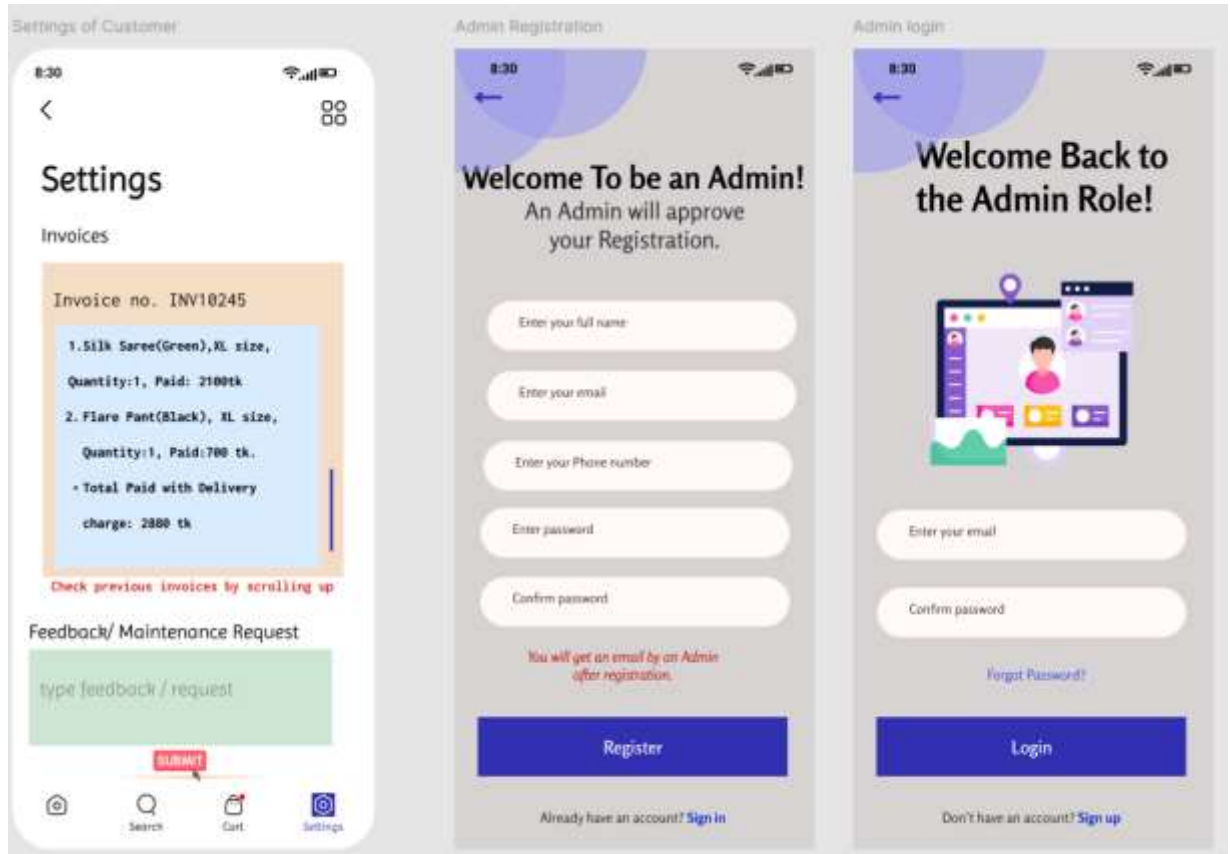
1. Add, update, and delete shop details easily.
2. Manage shopkeeper and employee information.
3. Store and access customer details and purchase history.
4. Keep track of inventory and notify when stock is low.
5. Generate bills and invoices automatically.
6. Create reports on sales, inventory, and maintenance.
7. Handle maintenance requests and track their status.

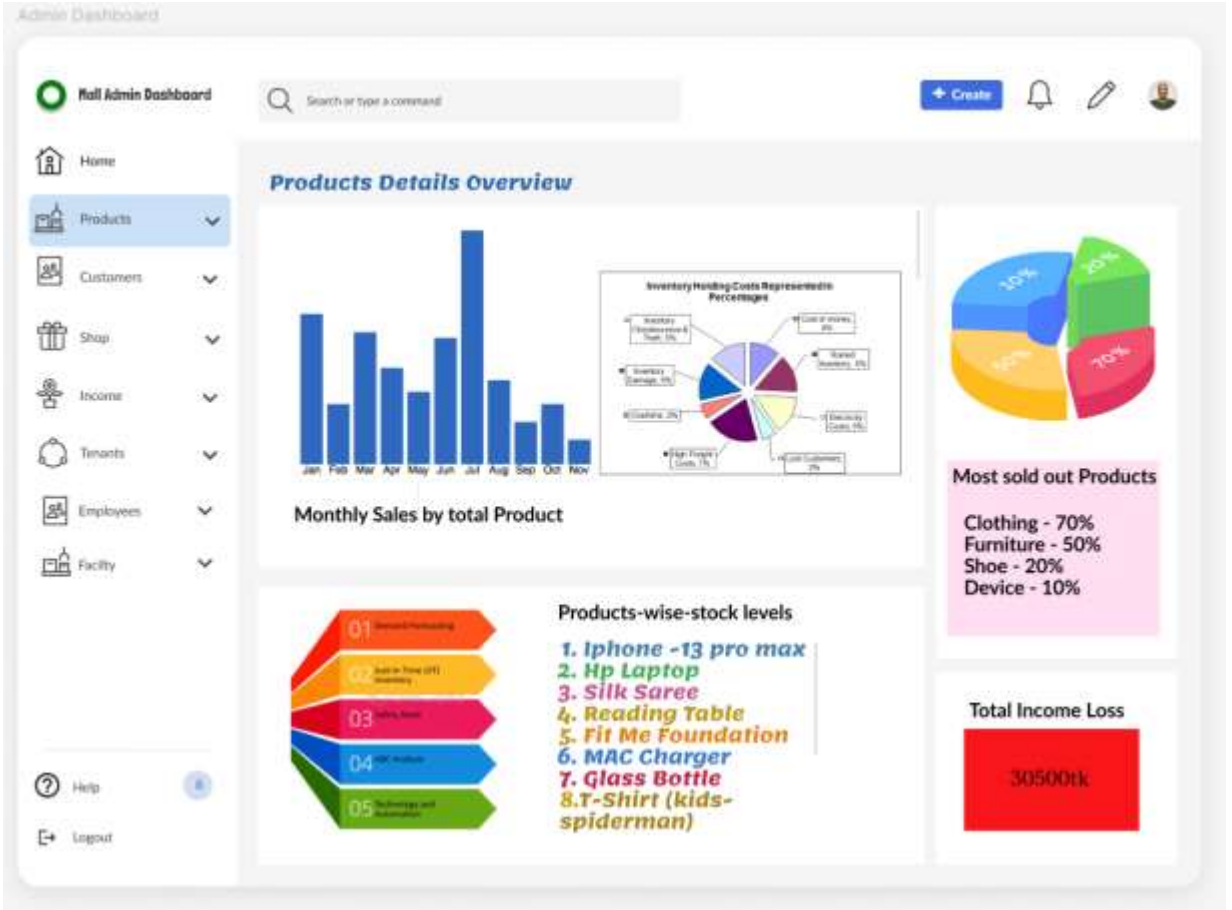
## Expected Outcome

We expect to develop a reliable and efficient database system that centralizes all mall management data in one place. This system will allow us to store, update, and retrieve information quickly and accurately using SQL operations. By automating tasks like billing, inventory tracking, and report generation, we will reduce manual errors and save time. The final outcome will be a well-organized database that supports mall administrators and customers in managing daily operations smoothly and effectively.

### 3. User Interface Planning







Mall Admin Dashboard

Home

Products

Customers

Shop

Income

Tenants

Employees

Facility

Help

Logout

Search or type a command

Create

Manage Customer Details

Customer_ID	Customer_name	Customer_email	Visit_date	Feedback	Customer_phone
A21	Mahmud Hasan	mahmud.hasan@example.com	2025-08-10	Very satisfied	01711220033
G23	Laila Akter	laila.akter@example.com	2025-08-11	Good service	01822334455
T16	Imran Hossain	imran.hossain@example.com	2025-08-12	Average experience	01566778844
B09	Nusrat Jahan	nusrat.jahan@example.com	2025-08-13	Excellent	01944556677
K31	Abdullah Al Mamun	abdullah.mamun@example.com	2025-08-14	Needs improvement	01688990011
R45	Sharmin Sultana	sharmin.sultana@example.com	2025-08-15	Very friendly staff	01799887766
M55	Farid Ahmed	farid.ahmed@example.com	2025-08-16	Quick service	01877665544
Z12	Rukhsana Begum	rukhsana.begum@example.com	2025-08-17	Excellent support	01733445522
P88	Kamrul Hasan	kamrul.hasan@example.com	2025-08-18	Satisfied	01522334455
D07	Anika Chowdhury	anika.chowdhury@example.com	2025-08-19	Could be better	01911223344

Edit

Update

Total Customers

11203

ADD Customer



Admin Dashboard

Mall Admin Dashboard

Home

Products

Customers

Shop

Income

Tenants

Employees

Facility

Help

Logout

Search or type a command

+ Create

Manage Shop Details

Shop_ID	Shop_name	Shop_type	Shop_floor	Shop_status	Rents_by_Tenant_ID
S101	Fashion Point	Clothing	1st Floor	Open	5
S102	Tech World	Electronics	2nd Floor	Open	33
S103	Fresh Mart	Grocery	Ground	Open	2
S104	Book Haven	Bookstore	3rd Floor	Closed	37
S105	Style & Shine	Salon	1st Floor	Open	10
S106	Tasty Bites	Food Court	2nd Floor	Under Renov	76
S107	Urban Decor	Home & Living	Ground	Open	18
S108	Coffee Corner	Cafe	1st Floor	Open	55
S109	Gadget Galaxy	Electronics	3rd Floor	Closed	21

S301	Gadget Hub	Electronics	1st Floor	Open	192
S302	Sweet Tooth	Bakery	Ground	Open	190
S303	Green Leaf	Grocery	2nd Floor	Under Renov	191
S304	Trendy Styles	Clothing	3rd Floor	Open	3

+

Edit

Update

Total Shops

200

ADD Shop

Admin Dashboard

Mall Admin Dashboard

Home

Products

Customers

Shop

Income

Bookings

Payments

Tenants

Employees

Facility

Help

Logout

Search or type a command

+ Create

Bookings Details

Booking_ID	Customer_ID	Lease_start_date	Lease_end_date	Shop_type	Product_name	Booking_payment_status
B001	A21	2025-08-01	2026-07-31	Clothing	Men's T-Shirt	Paid
B002	G23	2025-08-05	2026-08-04	Electronics	Smartphone	Unpaid
B003	I16	2025-08-10	2026-08-09	Grocery	Rice Skg	Paid
B004	B09	2025-08-12	2026-08-11	Food Court	Sandwich Combo	Paid
B005	K31	2025-08-15	2026-08-14	Clothing	Women's Dress	Unpaid
B006	B45	2025-08-17	2026-08-16	Electronics	Laptop	Paid
B007	M55	2025-08-18	2026-08-17	Grocery	Cooking Oil 1L	Paid
B008	Z12	2025-08-20	2026-08-19	Food Court	Coffee Latte	Unpaid
B009	P88	2025-08-22	2026-08-21	Clothing	Kids Jacket	Paid
B010	D07	2025-08-23	2026-08-22	Electronics	Headphones	Paid
B011	A21	2025-08-25	2026-08-24	Grocery	Fresh Vegetables Pack	Paid
B012	G23	2025-08-27	2026-08-26	Food Court	Pizza Slice	Unpaid
B013	I16	2025-08-28	2026-08-27	Clothing	Casual Shoes	Paid
B014	B09	2025-08-30	2026-08-29	Electronics	Smartwatch	Paid
B015	K31	2025-09-01	2026-08-31	Grocery	Dairy Milk 1L	Unpaid
B016	B45	2025-09-03	2026-09-02	Food Court	Sandwich Combo	Paid
B017	M55	2025-09-05	2026-09-04	Clothing	Men's T-Shirt	Paid

+

Edit

Update

Total Bookings

207

ADD Booking



Admin Dashboard

Null Admin Dashboard

Home

Products

Customers

Shop

Income

Bookings

Payments

Tenants

Employees

Facility

Help

Logout

Search or type a command

Create

Payments Details

Payment_ID	Customer_ID	Product_name	Amount (tk)	Payment_method	Payment_date
P001	A21	Men's T-Shirt	1500	Cash	2025-08-01
P002	G23	Smartphone	45000	Credit Card	2025-08-05
P003	T16	Rice Skg	600	Mobile Payment	2025-08-10
P004	B09	Sandwich Combo	350	Cash	2025-08-12
P005	K31	Women's Dress	2000	Credit Card	2025-08-15
P006	R45	Laptop	65000	Bank Transfer	2025-08-17
P007	M55	Cooking Oil 1L	400	Mobile Payment	2025-08-18
P008	Z12	Coffee Latte	300	Cash	2025-08-20
P009	P88	Kids' Jacket	1800	Credit Card	2025-08-22
P011	A21	Fresh Vegetables Pack	750	Cash	2025-08-25
P012	G23	Pizza Slice	500	Credit Card	2025-08-27
P013	T16	Casual Shoes	1200	Mobile Payment	2025-08-28
P014	B09	Smartwatch	8000	Bank Transfer	2025-08-30
P015	K31	Dairy Milk 1L	150	Cash	2025-09-01
P016	R45	Sandwich Combo	350	Mobile Payment	2025-09-03
P017	M55	Men's T-Shirt	1500	Credit Card	2025-09-05
P018	Z12	Laptop	65000	Bank Transfer	2025-09-07
P019	P88	Rice Skg	600	Cash	2025-09-09

Edit

Update

Total Payments completed

1302 Customers

Total Payments remaining

150 Customers

Admin Dashboard

Null Admin Dashboard

Home

Products

Customers

Shop

Income

Tenants

Employees

Facility

Help

Logout

Search or type a command

Create

Manage Tenant Details

Tenant_ID	Tenant_name	Tenant_phone	Tenant_ID_proof	Tenant_email
1	Rafiah Uddin	01711223344	NID-1234567890	rafiah.uddin@example.com
2	Karen Hassan	01899687766	Passport-BA12345	karen.hassan@example.com
3	Sumaya Akter	01622334455	NID-2233445566	sumaya.akter@example.com
4	Tarek Ahmed	01933445566	DrivingLicense-OL78909	tarek.ahmed@example.com
5	Fatima Sultan	01755667788	NID-9988776655	fatima.sultan@example.co
6	Jamil Chowdhury	01566778899	Passport-CX56789	jamil.c Chowdhury@example.co
190	Nadia Rahman	01777888900	NID-4455667788	nadia.rahman@example.com
191	Rakibul Islam	01811224400	Passport-DH56789	rakibul.islam@example.com
192	Shabnam Chowdhury	01922113344	DrivingLicense-OL33344	shabnam.chowdhury@example

+

Edit

Update

Total Tenants

189

ADD Tenant

Admin Dashboard

Mall Admin Dashboard

Home

Products

Customers

Shop

Income

Tenants

Employees

Facility

Help

Logout

Search or type a command

Create

Manage Employees Details

Employee_ID	Employee_name	Emp_salary	Employee_role	Emp_shift_time	Employee_phone
E0001	Saiful Islam	30000	Manager	9 AM – 5 PM	01711223344
E0002	Jannatul Ferdous	22000	Receptionist	10 AM – 6 PM	01822334455
E0003	Habib Rahman	25000	Accountant	9 AM – 5 PM	01533445566
E0004	Shaila Akter	18000	Housekeeping	8 AM – 4 PM	01944556677
E0005	Rashed Khan	27000	Supervisor	2 PM – 10 PM	01655667788
E0006	Nargis Sultana	20000	Security Guard	10 PM – 6 AM	01766778899

E3781	Tarek Mahmud	26000	Front Desk Officer	9 AM – 5 PM	01888990011
E3782	Shirin Akhter	19000	Cleaner	7 AM – 3 PM	01733446655
E3783	Kamal Uddin	28000	Technician	11 AM – 7 PM	01566779922
E3784	Farzana Yasmin	23000	HR Assistant	9 AM – 5 PM	01911224477

Edit

Update

Total Employees

3780

ADD Employee

Admin Dashboard

Mall Admin Dashboard

Home

Products

Customers

Shop

Income

Tenants

Employees

Facility

Help

Logout

Search or type a command

Create

Manage Facility Details

Facility_ID	Facility_name	Availability_status	Hired_Employee_name	Hired_Employee_role
F001	Escalator	Available	Kamal Uddin	Technician
F002	Elevator	Under Maintenance	Shirin Akhter	Technician
F003	Fire Safety System	Available	Rashed Khan	Safety Officer
F004	CCTV Cameras	Available	Nargis Sultana	Security Officer
F005	PA System	Available	Tarek Mahmud	Technician
F006	HVAC System	Under Maintenance	Farzana Yasmin	Technician
F007	Parking Lot	Available	Habib Rahman	Supervisor
F008	Food Court Seating	Available	Shaila Akter	Housekeeping
F009	Water Supply	Available	Saiful Islam	Maintenance Staff

F026	Gym	Available	Kamal Uddin	Trainer
------	-----	-----------	-------------	---------

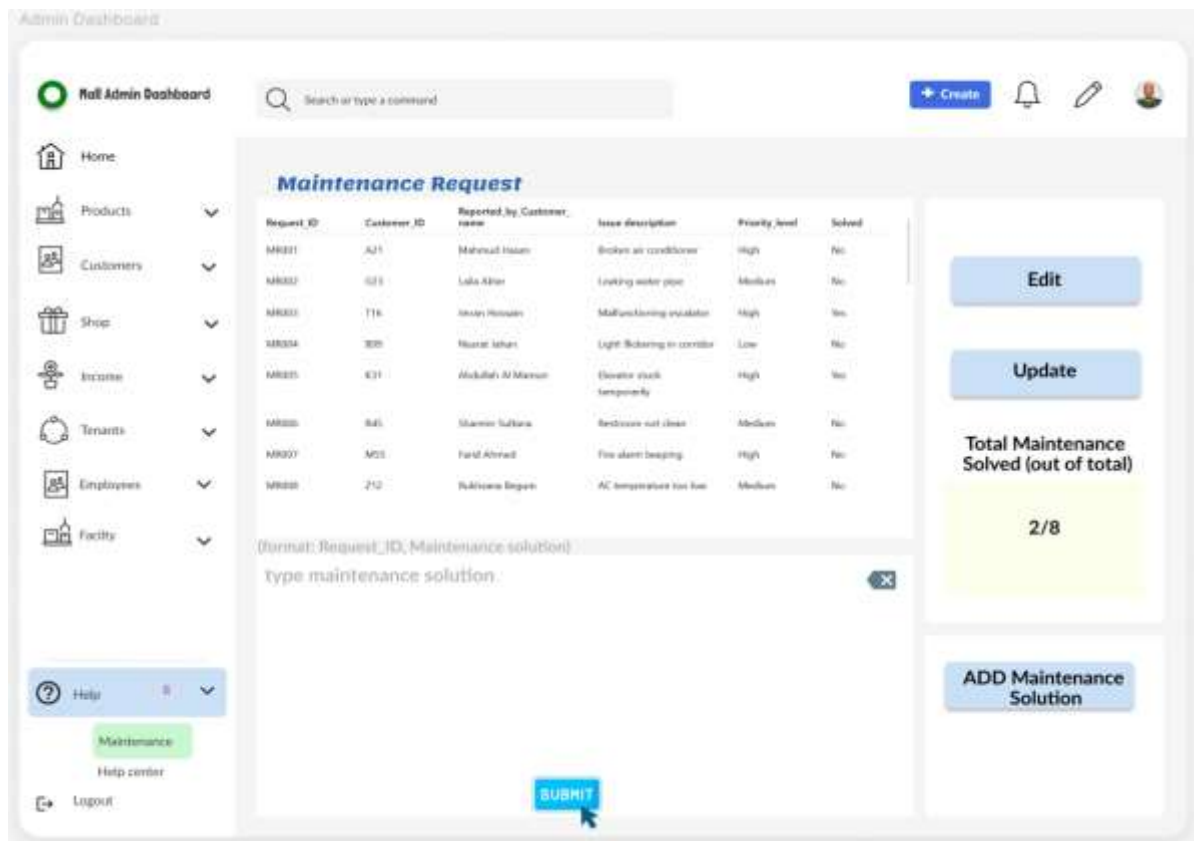
Edit

Update

Total Facilities

25

ADD Facility

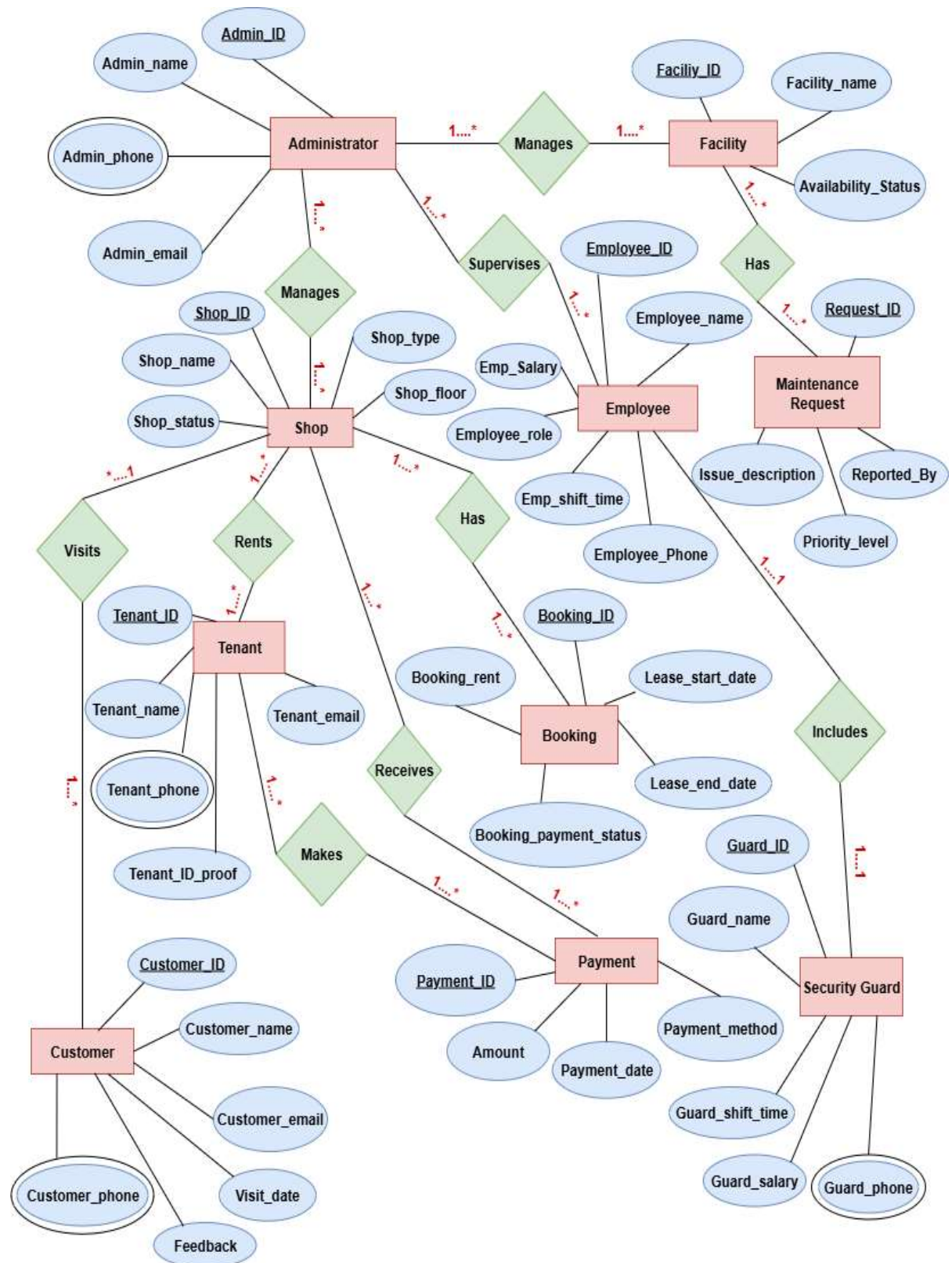


## 4.Scenario Description

In a Mall Management System, one administrator manages many shops, facilities and supervises employees. Each administrator has a unique identification number, name, phone number, and email. A shop has a unique identification number, name, type, floor, and current status. Each shop is rented by exactly one tenant. One tenant may rent one or more shops. A tenant is defined by a unique tenant ID, name, phone number, email and ID proof. A shop must have a booking record. One shop can have one or more bookings, but a booking is related to one shop only. A booking is defined by a booking ID, rent, lease start and end dates, and payment status. A payment is made by a tenant for a shop. One payment is linked to exactly one tenant and one shop. Payment is identified by a unique payment ID, amount, date and method. A shop can have many customers. Each customer has a unique identification number, name, phone number, email, visit date, and feedback. A customer can have more than one phone number and email. Each employee works under the administrator and may include cleaners, support staff, and other roles. An employee has a unique identification number, name, role, shift time, salary and phone number. The mall provides various facilities like elevators, parking, and restrooms. Each facility has a unique identification number, name, and availability status. Facilities may be associated with one or more maintenance requests. A maintenance request is defined by a request ID, facility ID, issue description, reported by, priority level. Security guards are also part of the employee group, but are maintained separately. Each security guard has a unique identification number, name, shift time, phone number

and salary. The system offers a centralized platform for efficient administration, ensuring that all operations—from shop allocation to facility monitoring—run seamlessly to enhance service quality and overall mall management.

## 5.ER Diagram



## 6.Normalization

### Manages (Administrator-Shop)

#### UNF

Manages (Admin\_ID, Admin\_name, Admin\_phone, Admin\_email, Shop\_ID, Shop\_name, Shop\_status, Shop\_type, Shop\_floor)

#### 1NF

Admin\_phone is a multi valued attribute.

1. Admin\_ID, Admin\_name, Admin\_phone, Admin\_email, Shop\_ID, Shop\_name, Shop\_status, Shop\_type, Shop\_floor

#### 2NF

1. Admin\_ID, Admin\_name, Admin\_phone, Admin\_email
2. Shop\_ID, Shop\_name, Shop\_status, Shop\_type, Shop\_floor

#### 3NF

There is no transitive dependency. Relation already in 3NF.

1. Admin\_ID, Admin\_name, Admin\_phone, Admin\_email
2. Shop\_ID, Shop\_name, Shop\_status, Shop\_type, Shop\_floor

#### Table Creation

1. Admin\_ID, Admin\_name, Admin\_phone, Admin\_email
2. Shop\_ID, Shop\_name, Shop\_status, Shop\_type, Shop\_floor, **A\_ID**

### Manages (Administrator-Facility)

#### UNF

Manages (Admin\_ID, Admin\_name, Admin\_phone, Admin\_email, Facility\_ID, Facility\_name, Availability\_Status)

#### 1NF

Admin\_phone is a multivalued attribute.

1. Admin\_ID, Admin\_name, Admin\_phone, Admin\_email, Facility\_ID, Facility\_name, Availability\_Status

#### 2NF

1. Admin\_ID, Admin\_name, Admin\_phone, Admin\_email
2. Facility\_ID, Facility\_name, Availability\_Status

### **3NF**

There is no transitive dependency. Relation already in 3NF.

1. Admin\_ID, Admin\_name, Admin\_phone, Admin\_email
2. Facility\_ID, Facility\_name, Availability\_Status

### **Table Creation**

1. Admin\_ID, Admin\_name, Admin\_phone, Admin\_email
2. Facility\_ID, Facility\_name, Availability\_Status, **A\_ID**

## **Supervises (Administrator-Employee)**

### **UNF**

Supervises (Admin\_ID, Admin\_name, Admin\_phone, Admin\_email, Employee\_ID, Employee\_name, Emp\_salary, Employee\_role, Employee\_phone, Emp\_shift\_time)

### **1NF**

Admin\_phone and Employee\_phone are multi valued attributes.

- 1 Admin\_ID, Admin\_name, Admin\_phone, Admin\_email

2. Employee\_ID, Employee\_name, Emp\_salary, Employee\_role, Employee\_phone, Emp\_shift\_time

### **2NF**

1. Admin\_ID, Admin\_name, Admin\_phone, Admin\_email

2. Employee\_ID, Employee\_name, Emp\_salary, Employee\_role, Employee\_phone, Emp\_shift\_time

### **3NF**

There is no transitive dependency. Relation already in 3NF.

1. Admin\_ID, Admin\_name, Admin\_phone, Admin\_email
2. Employee\_ID, Employee\_name, Emp\_salary, Employee\_role, Employee\_phone, Emp\_shift\_time

### **Table Creation**

1. Admin\_ID, Admin\_name, Admin\_phone, Admin\_email
2. Employee\_ID, Employee\_name, Emp\_salary, Employee\_role, Employee\_phone, Emp\_shift\_time, **A\_ID**

## **Has (Facility - Maintenance Request)**

### **UNF**

Has (Facility\_ID, Facility\_name, Availability\_Status, Request\_ID, Issue\_description, Reported\_by, Priority\_level)

### **1NF**

There is no multi valued attribute. Relation already in 1NF.

1. Facility\_ID, Facility\_name, Availability\_Status
2. Request\_ID, Issue\_description, Reported\_by, Priority\_level

### **2NF**

1. Facility\_ID, Facility\_name, Availability\_Status
2. Request\_ID, Issue\_description, Reported\_by, Priority\_level

### **3NF**

There is no transitive dependency. Relation already in 3NF.

1. Facility\_ID, Facility\_name, Availability\_Status
2. Request\_ID, Issue\_description, Reported\_by, Priority\_level

### **Table Creation**

1. Facility\_ID, Facility\_name, Availability\_Status
2. Request\_ID, Issue\_description, Reported\_by, Priority\_level, **F\_ID**

## **Rents (Tenant-Shop)**

### **UNF**

Rents (Tenant\_ID, Tenant\_name, Tenant\_email, Tenant\_phone, Tenant\_ID\_proof, Shop\_ID, Shop\_name, Shop\_status, Shop\_type, Shop\_floor)

### **1NF**

Tenant\_phone is multi valued attribute.

1. Tenant\_ID, Tenant\_name, Tenant\_email, Tenant\_phone, Tenant\_ID\_proof
2. Shop\_ID, Shop\_name, Shop\_status, Shop\_type, Shop\_floor

### **2NF**

1. Tenant\_ID, Tenant\_name, Tenant\_email, Tenant\_phone, Tenant\_ID\_proof
2. Shop\_ID, Shop\_name, Shop\_status, Shop\_type, Shop\_floor

### **3NF**

There is no transitive dependency. Relation already in 3NF.

1. Tenant\_ID, Tenant\_name, Tenant\_email, Tenant\_phone, Tenant\_ID\_proof
2. Shop\_ID, Shop\_name, Shop\_status, Shop\_type, Shop\_floor

### **Table Creation**

1. Tenant\_ID, Tenant\_name, Tenant\_email, Tenant\_phone, Tenant\_ID\_proof
2. Shop\_ID, Shop\_name, Shop\_status, Shop\_type, Shop\_floor, **T\_ID**



## Makes (Tenant - Payment)

### UNF

Makes (Tenant\_ID, Tenant\_name, Tenant\_phone, Tenant\_email, Tenant\_ID\_proof, Payment\_ID, Amount, Payment\_date, Payment\_method)

### 1NF

Tenant\_phone is a multi valued attribute.

1. Tenant\_ID, Tenant\_name, Tenant\_phone, Tenant\_email, Tenant\_ID\_proof, Payment\_ID, Amount, Payment\_date, Payment\_method

### 2NF

1. Tenant\_ID, Tenant\_name, Tenant\_phone, Tenant\_email, Tenant\_ID\_proof

2. Payment\_ID, Amount, Payment\_date, Payment\_method

### 3NF

There is no transitive dependency. Relation already in 3NF.

1. Tenant\_ID, Tenant\_name, Tenant\_phone, Tenant\_email, Tenant\_ID\_proof

2. Payment\_ID, Amount, Payment\_date, Payment\_method

### Table Creation

1. Tenant\_ID, Tenant\_name, Tenant\_phone, Tenant\_email, Tenant\_ID\_proof

2. Payment\_ID, Amount, Payment\_date, Payment\_method, **T\_ID**

## Receives (Shop - Payment)

### UNF

Receives (Shop\_ID, Shop\_name, Shop\_status, Shop\_type, Shop\_floor, Payment\_ID, Amount, Payment\_date, Payment\_method)

### 1NF

There is no multivalued attribute. Relation is already in 1NF.

1. Shop\_ID, Shop\_name, Shop\_status, Shop\_type, Shop\_floor, Payment\_ID, Amount, Payment\_date, Payment\_method

### 2NF

1. Shop\_ID, Shop\_name, Shop\_status, Shop\_type, Shop\_floor

2. Payment\_ID, Amount, Payment\_date, Payment\_method

### 3NF

There is no transitive dependency. Relation already in 3NF.

1. Shop\_ID, Shop\_name, Shop\_status, Shop\_type, Shop\_floor

2. Payment\_ID, Amount, Payment\_date, Payment\_method

#### Table Creation

1. Shop\_ID, Shop\_name, Shop\_status, Shop\_type, Shop\_floor
2. Payment\_ID, Amount, Payment\_date, Payment\_method, **S\_ID**

## Has (Shop - Booking)

#### UNF

Has (Shop\_ID, Shop\_name, Shop\_status, Shop\_type, Shop\_floor, Booking\_ID, Booking\_rent, Lease\_start\_date, Lease\_end\_date, Booking\_payment\_status)

#### 1NF

There is no multivalued attribute. Relation is already in 1NF.

1. Shop\_ID, Shop\_name, Shop\_status, Shop\_type, Shop\_floor, Booking\_ID, Booking\_rent, Lease\_start\_date, Lease\_end\_date, Booking\_payment\_status

#### 2NF

1. Shop\_ID, Shop\_name, Shop\_status, Shop\_type, Shop\_floor
2. Booking\_ID, Booking\_rent, Lease\_start\_date, Lease\_end\_date, Booking\_payment\_status

#### 3NF

There is no transitive dependency. Relation already in 3NF.

1. Shop\_ID, Shop\_name, Shop\_status, Shop\_type, Shop\_floor
2. Booking\_ID, Booking\_rent, Lease\_start\_date, Lease\_end\_date, Booking\_payment\_status

#### Table Creation

1. Shop\_ID, Shop\_name, Shop\_status, Shop\_type, Shop\_floor
2. Booking\_ID, Booking\_rent, Lease\_start\_date, Lease\_end\_date, Booking\_payment\_status, **S\_ID**

## Visits (Shop - Customer)

#### UNF

Visits (Shop\_ID, Shop\_name, Shop\_status, Shop\_type, Shop\_floor, Customer\_ID, Customer\_name, Customer\_email, Customer\_phone, Visit\_date, Feedback)

#### 1NF

Customer\_phone is a multi valued attribute.

1. Shop\_ID, Shop\_name, Shop\_status, Shop\_type, Shop\_floor, Customer\_ID, Customer\_name, Customer\_email, Customer\_phone, Visit\_date, Feedback

## **2NF**

1. Shop\_ID, Shop\_name, Shop\_status, Shop\_type, Shop\_floor
2. Customer\_ID, Customer\_name, Customer\_email, Customer\_phone, Visit\_date, Feedback

## **3NF**

There is no transitive dependency. Relation already in 3NF.

1. Shop\_ID, Shop\_name, Shop\_status, Shop\_type, Shop\_floor
2. Customer\_ID, Customer\_name, Customer\_email, Customer\_phone, Visit\_date, Feedback

## **Table Creation**

1. Shop\_ID, Shop\_name, Shop\_status, Shop\_type, Shop\_floor
2. Customer\_ID, Customer\_name, Customer\_email, Customer\_phone, Visit\_date, Feedback, S\_ID

## **Includes (Employee – Security Guard)**

## **UNF**

Includes (Employee\_ID, Employee\_name, Emp\_salary, Employee\_role, Emp\_shift\_time, Employee\_phone, Guard\_ID, Guard\_name, Guard\_phone, Guard\_shift\_time, Guard\_salary)

## **1NF**

Employee\_phone and Guard\_phone are multi valued attributes.

1. Employee\_ID, Employee\_name, Emp\_salary, Employee\_role, Emp\_shift\_time, Employee\_phone, Guard\_ID, Guard\_name, Guard\_phone, Guard\_shift\_time, Guard\_salary

## **2NF**

1. Employee\_ID, Employee\_name, Emp\_salary, Employee\_role, Emp\_shift\_time, Employee\_phone
2. Guard\_ID, Guard\_name, Guard\_phone, Guard\_shift\_time, Guard\_salary

## **3NF**

There is no transitive dependency. Relation already in 3NF.

1. Employee\_ID, Employee\_name, Emp\_salary, Employee\_role, Emp\_shift\_time, Employee\_phone
2. Guard\_ID, Guard\_name, Guard\_phone, Guard\_shift\_time, Guard\_salary, Employee\_ID

## **Table Creation**

1. Employee\_ID, Employee\_name, Emp\_salary, Employee\_role, Emp\_shift\_time, Employee\_phone
2. Guard\_ID, Guard\_name, Guard\_phone, Guard\_shift\_time, Guard\_salary, E\_ID

## Temporary Tables

1. Admin\_ID, Admin\_name, Admin\_phone, Admin\_email
2. Shop\_ID, Shop\_name, Shop\_status, Shop\_type, Shop\_floor, **A\_ID**
3. ~~Admin\_ID, Admin\_name, Admin\_phone, Admin\_email~~
4. Facility\_ID, Facility\_name, Availability\_Status, **A\_ID**
5. ~~Admin\_ID, Admin\_name, Admin\_phone, Admin\_email~~
6. Employee\_ID, Employee\_name, Emp\_salary, Employee\_role, Employee\_phone, Emp\_shift\_time, **A\_ID**
7. ~~Facility\_ID, Facility\_name, Availability\_Status~~
8. Request\_ID, Issue\_description, Reported\_by, Priority\_level, **F\_ID**
9. Tenant\_ID, Tenant\_name, Tenant\_email, Tenant\_phone, Tenant\_ID\_proof
10. ~~Shop\_ID, Shop\_name, Shop\_status, Shop\_type, Shop\_floor, **T\_ID**~~
11. ~~Tenant\_ID, Tenant\_name, Tenant\_phone, Tenant\_email, Tenant\_ID\_proof~~
12. Payment\_ID, Amount, Payment\_date, Payment\_method, **T\_ID**
13. ~~Shop\_ID, Shop\_name, Shop\_status, Shop\_type, Shop\_floor~~
14. ~~Payment\_ID, Amount, Payment\_date, Payment\_method, **S\_ID**~~
15. ~~Shop\_ID, Shop\_name, Shop\_status, Shop\_type, Shop\_floor~~
16. Booking\_ID, Booking\_rent, Lease\_start\_date, Lease\_end\_date, Booking\_payment\_status, **S\_ID**
17. ~~Shop\_ID, Shop\_name, Shop\_status, Shop\_type, Shop\_floor~~
18. Customer\_ID, Customer\_name, Customer\_email, Customer\_phone, Visit\_date, Feedback, **S\_ID**
19. ~~Employee\_ID, Employee\_name, Emp\_salary, Employee\_role, Emp\_shift\_time, Employee\_phone~~
20. Guard\_ID, Guard\_name, Guard\_phone, Guard\_shift\_time, Guard\_salary, **E\_ID**

## Final Tables

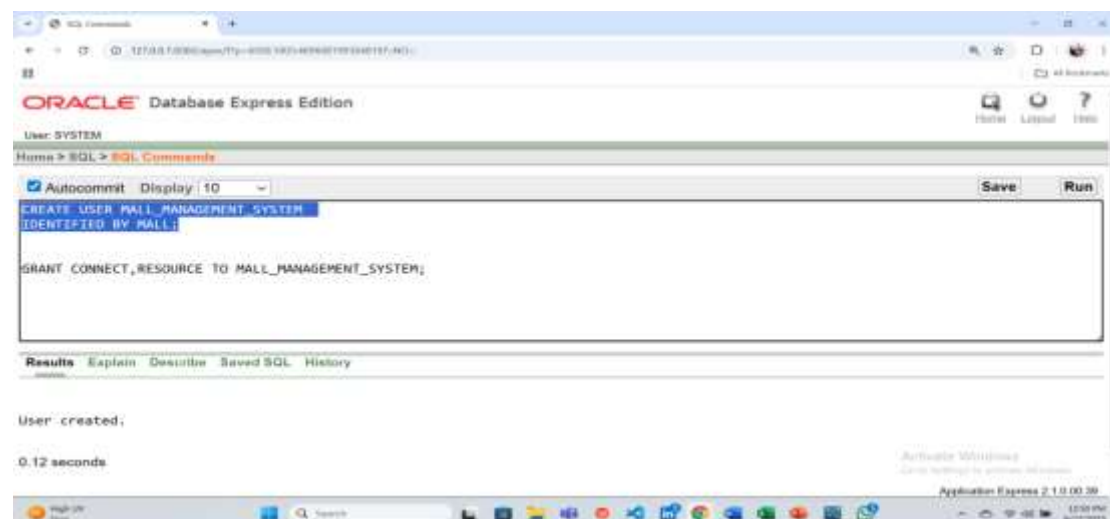
1. Admin\_ID, Admin\_name, Admin\_phone, Admin\_email
2. Shop\_ID, Shop\_name, Shop\_status, Shop\_type, Shop\_floor, **A\_ID**
3. Facility\_ID, Facility\_name, Availability\_Status, **A\_ID**
4. Employee\_ID, Employee\_name, Emp\_salary, Employee\_role, Employee\_phone, Emp\_shift\_time, **A\_ID**
5. Request\_ID, Issue\_description, Reported\_by, Priority\_level, **F\_ID**
6. Tenant\_ID, Tenant\_name, Tenant\_email, Tenant\_phone, Tenant\_ID\_proof
7. Payment\_ID, Amount, Payment\_date, Payment\_method, **T\_ID**
8. Booking\_ID, Booking\_rent, Lease\_start\_date, Lease\_end\_date, Booking\_payment\_status, **S\_ID**
9. Customer\_ID, Customer\_name, Customer\_email, Customer\_phone, Visit\_date, Feedback, **S\_ID**
10. Guard\_ID, Guard\_name, Guard\_phone, Guard\_shift\_time, Guard\_salary, **E\_ID**

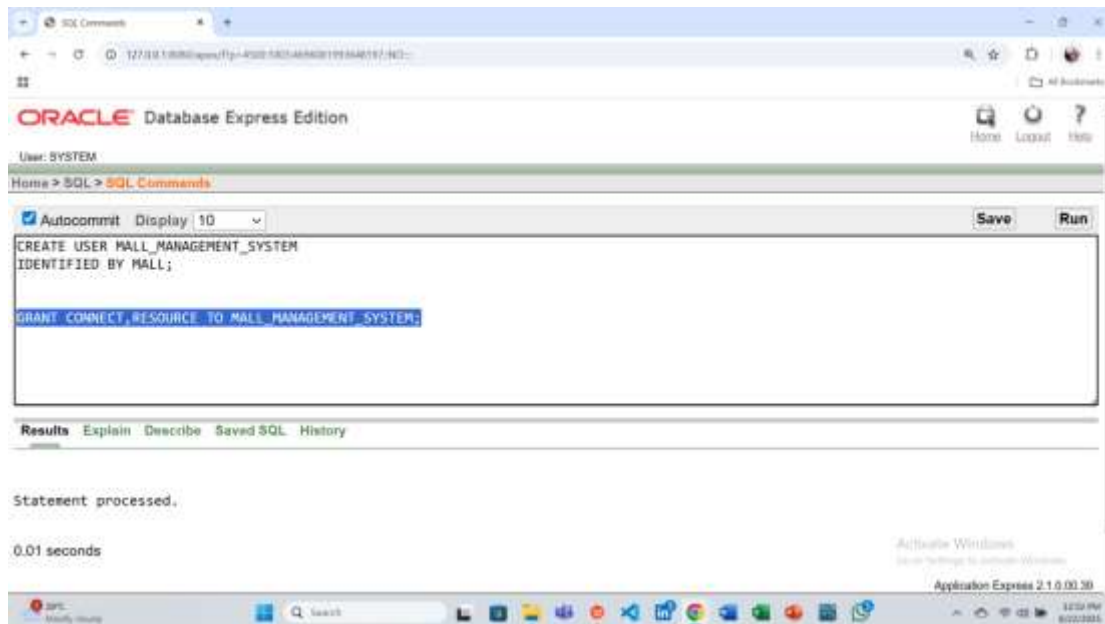
## 7.Schema Diagram



## 8.Table Creation Using SQL

Created A separate Database/Schema for Mall management system:





## Creating all Final Tables:

-- 1. Admin

CREATE TABLE Admin (

Admin\_ID INT PRIMARY KEY,

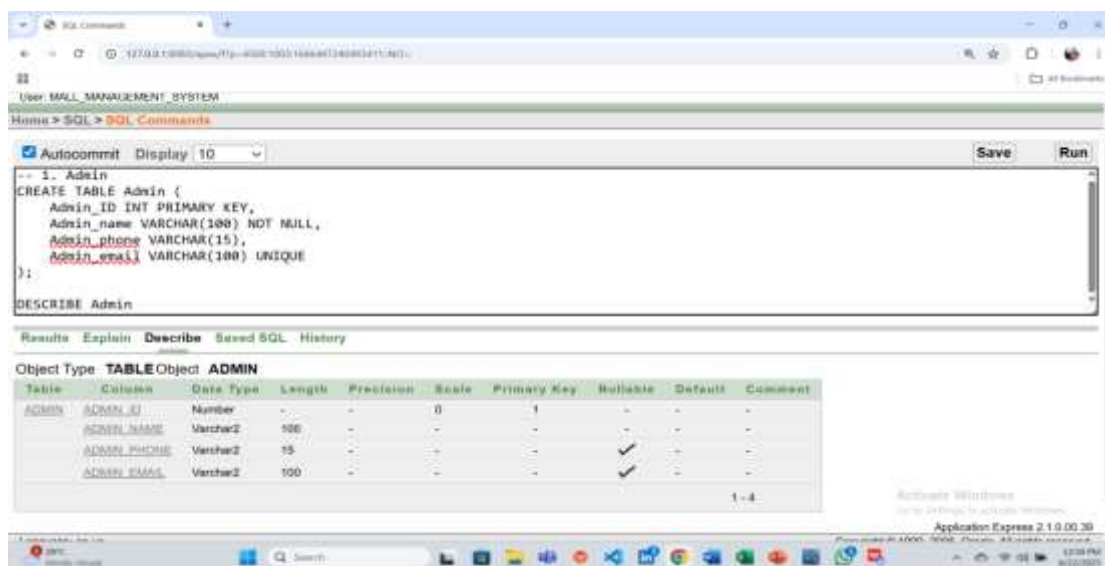
Admin\_name VARCHAR(100) NOT NULL,

Admin\_phone VARCHAR(15),

Admin\_email VARCHAR(100) UNIQUE

);

DESCRIBE Admin



-- 2. Shop

CREATE TABLE Shop (

Shop\_ID INT PRIMARY KEY,

Shop\_name VARCHAR(100) NOT NULL,

Shop\_status VARCHAR(20),

Shop\_type VARCHAR(50),

Shop\_floor INT,

A\_ID INT,

FOREIGN KEY (A\_ID) REFERENCES Admin(Admin\_ID)

);

DESCRIBE Shop

The screenshot shows a SQL Command window with the following SQL code:

```
-- 2. Shop
CREATE TABLE Shop (
  Shop_ID INT PRIMARY KEY,
  Shop_name VARCHAR(100) NOT NULL,
  Shop_status VARCHAR(20),
  Shop_type VARCHAR(50),
  Shop_floor INT,
  A_ID INT,
  FOREIGN KEY (A_ID) REFERENCES Admin(Admin_ID)
);
DESCRIBE Shop
```

Below the code, the 'Results' tab displays the table structure for 'SHOP'.

Table	Column	Data Type	Length	Precision	Scale	Primary Key	Nullable	Default	Comment
SHOP	SHOP_ID	Number			0	✓	✓		
SHOP	SHOP_NAME	Varchar2	100				✓		
SHOP	SHOP_STATUS	Varchar2	20				✓		
SHOP	SHOP_TYPE	Varchar2	50				✓		
SHOP	SHOP_FLOOR	Number			0		✓		
SHOP	A_ID	Number			0		✓		

The window also shows the 'Autocommit' checkbox checked and the 'Run' button.

-- 3. Facility

CREATE TABLE Facility (

Facility\_ID INT PRIMARY KEY,

Facility\_name VARCHAR(100),

Availability\_Status VARCHAR(20),

A\_ID INT,

FOREIGN KEY (A\_ID) REFERENCES Admin(Admin\_ID)



);

DESCRIBE Facility

The screenshot shows the SQL Developer interface. The SQL Command window contains the following SQL code:

```
-- 3. Facility
CREATE TABLE Facility (
  Facility_ID INT PRIMARY KEY,
  Facility_Name VARCHAR(100),
  Availability_Status VARCHAR(20),
  A_ID INT,
  FOREIGN KEY (A_ID) REFERENCES Admin(Admin_ID)
);
DESCRIBE Facility;
```

The Results window displays the table structure for 'FACILITY':

Column	Table	Column	Data Type	Length	Precision	Scale	Primary Key	Nullable	Default	Comment
FACILITY_ID	FACILITY	FACILITY_ID	NUMBER	4		0	✓	✓		
FACILITY_NAME	FACILITY	FACILITY_NAME	VARCHAR2	100				✓		
AVAILABILITY_STATUS	FACILITY	AVAILABILITY_STATUS	VARCHAR2	20				✓		
A_ID	FACILITY	A_ID	NUMBER	4		0		✓		

-- 4. Employee

CREATE TABLE Employee (

Employee\_ID INT PRIMARY KEY,

Employee\_name VARCHAR(100),

Emp\_salary DECIMAL(10,2),

Employee\_role VARCHAR(50),

Employee\_phone VARCHAR(15),

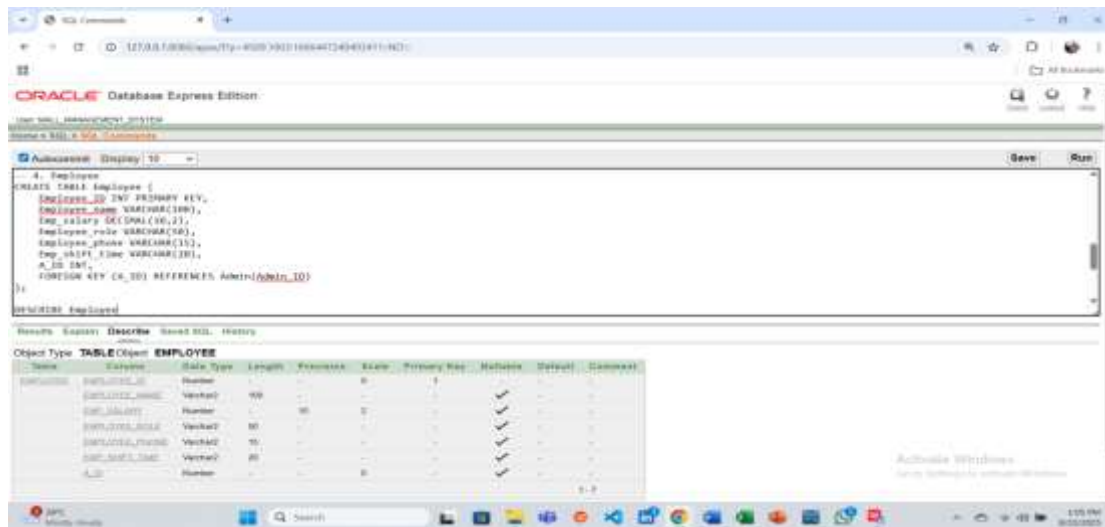
Emp\_shift\_time VARCHAR(20),

A\_ID INT,

FOREIGN KEY (A\_ID) REFERENCES Admin(Admin\_ID)

);

DESCRIBE Employee



-- 5. Request

CREATE TABLE Request (

Request\_ID INT PRIMARY KEY,

Issue\_description VARCHAR(255),

Reported\_by VARCHAR(100),

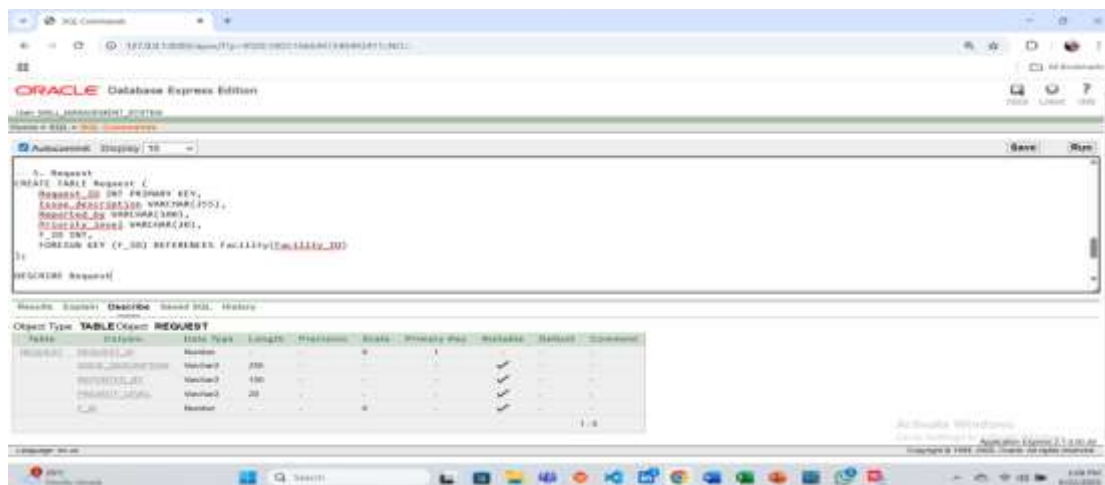
Priority\_level VARCHAR(20),

F\_ID INT,

FOREIGN KEY (F\_ID) REFERENCES Facility(Facility\_ID)

);

DESCRIBE Request



-- 6. Tenant

CREATE TABLE Tenant (

Tenant\_ID INT PRIMARY KEY,

Tenant\_name VARCHAR(100),

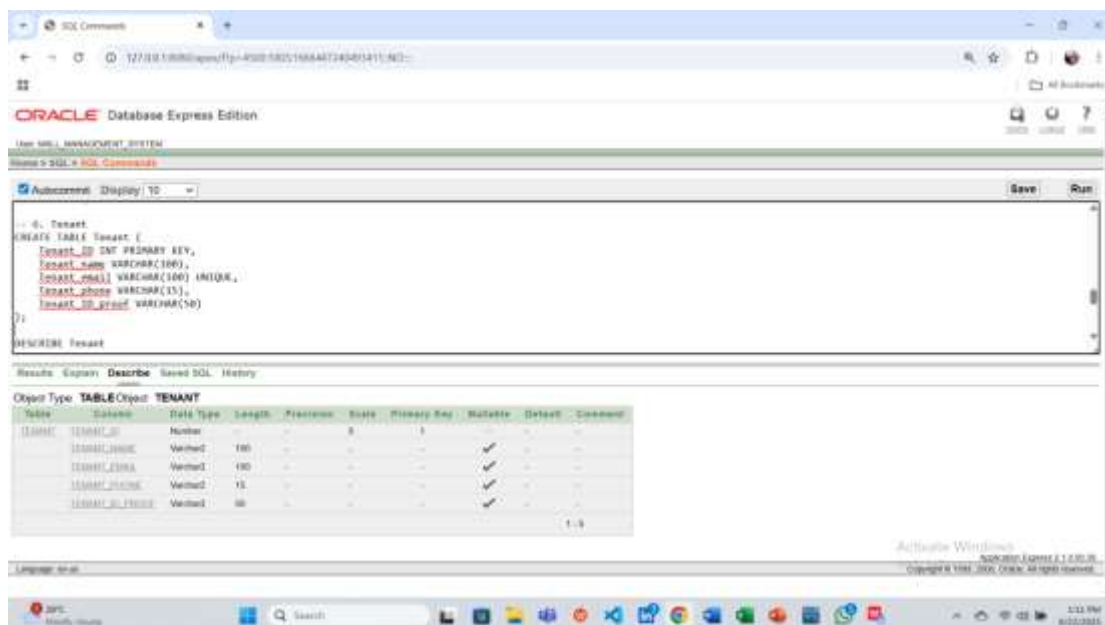
Tenant\_email VARCHAR(100) UNIQUE,

Tenant\_phone VARCHAR(15),

Tenant\_ID\_proof VARCHAR(50)

);

DESCRIBE Tenant



-- 7. Payment

CREATE TABLE Payment (

Payment\_ID INT PRIMARY KEY,

Amount DECIMAL(10,2),

Payment\_date DATE,

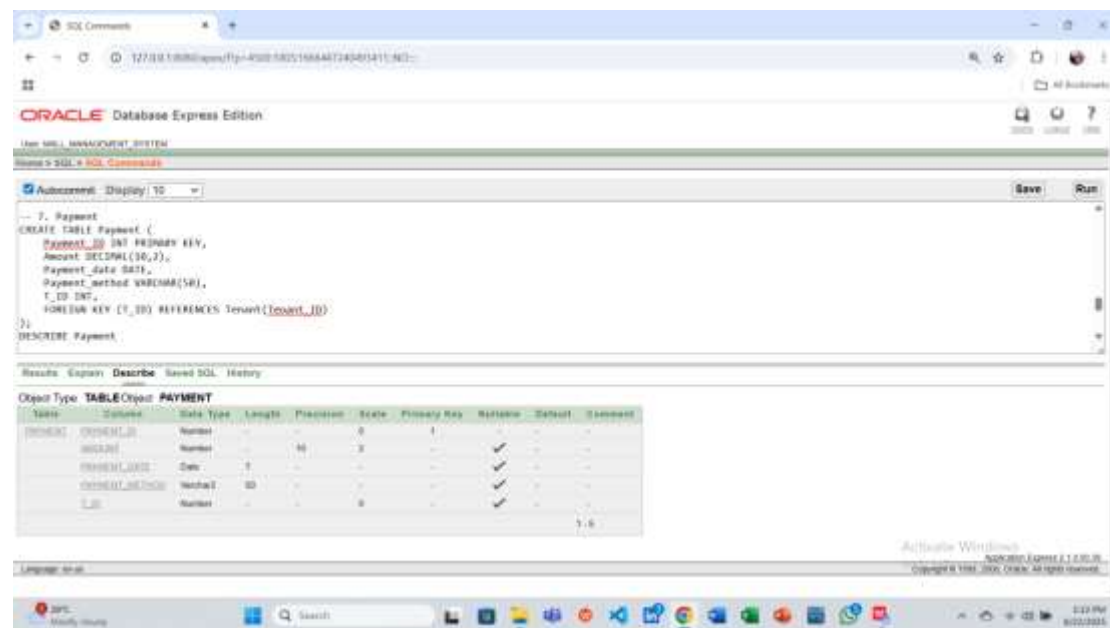
Payment\_method VARCHAR(50),

T\_ID INT,

FOREIGN KEY (T\_ID) REFERENCES Tenant(Tenant\_ID)

);

## DESCRIBE Payment



-- 8. Booking

CREATE TABLE Booking (

Booking\_ID INT PRIMARY KEY,

Booking\_rent DECIMAL(10,2),

Lease\_start\_date DATE,

Lease\_end\_date DATE,

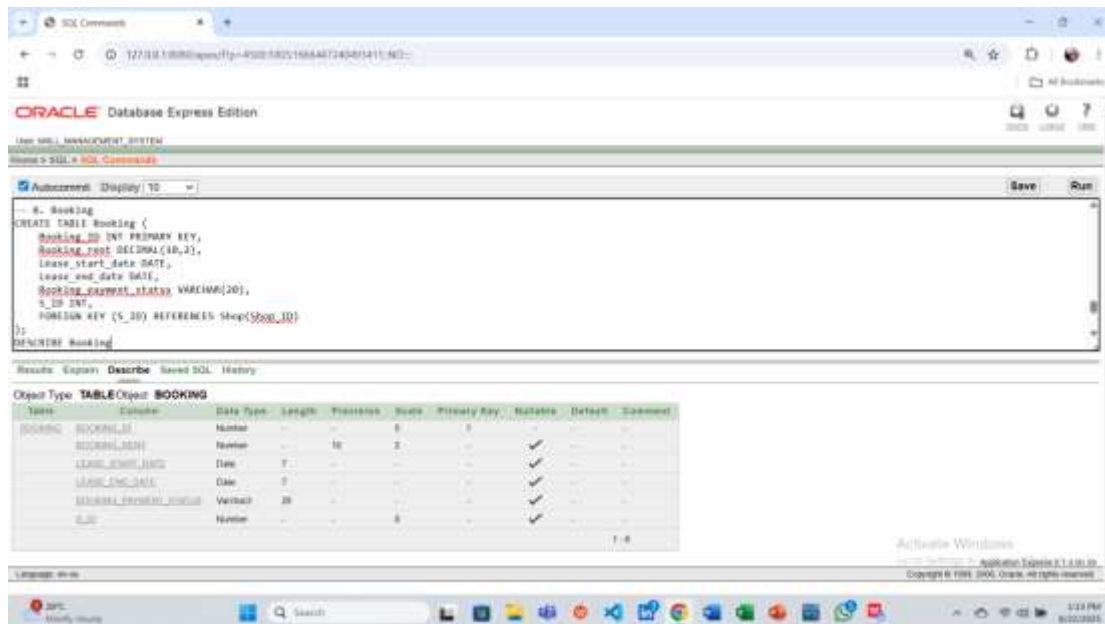
Booking\_payment\_status VARCHAR(20),

S\_ID INT,

FOREIGN KEY (S\_ID) REFERENCES Shop(Shop\_ID)

);

DESCRIBE Booking



-- 9. Customer

CREATE TABLE Customer (

Customer\_ID INT PRIMARY KEY,

Customer\_name VARCHAR(100),

Customer\_email VARCHAR(100),

Customer\_phone VARCHAR(15),

Visit\_date DATE,

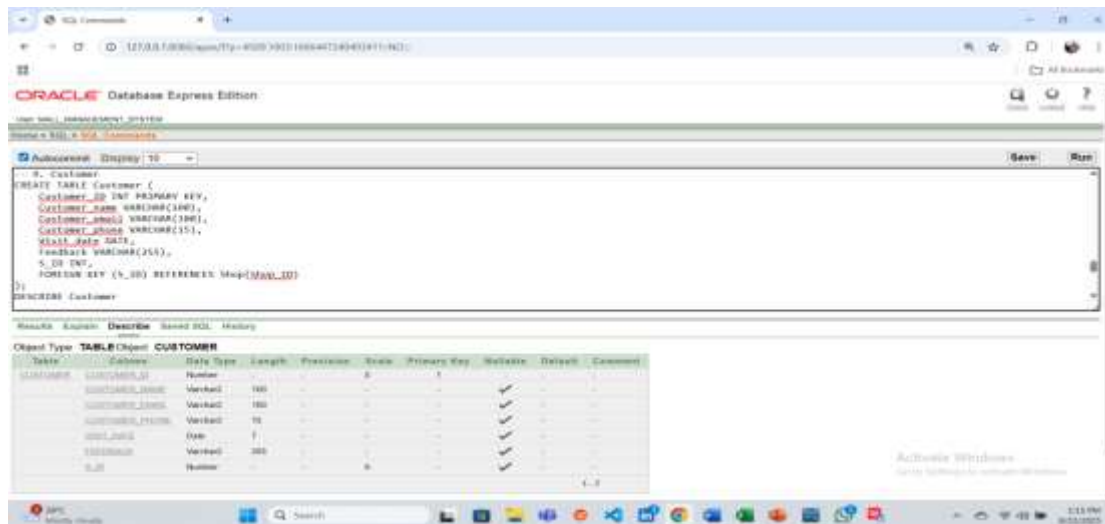
Feedback VARCHAR(255),

S\_ID INT,

FOREIGN KEY (S\_ID) REFERENCES Shop(Shop\_ID)

);

DESCRIBE Customer



-- 10. Guard

CREATE TABLE Guard (

Guard\_ID INT PRIMARY KEY,

Guard\_name VARCHAR(100),

Guard\_phone VARCHAR(15),

Guard\_shift\_time VARCHAR(20),

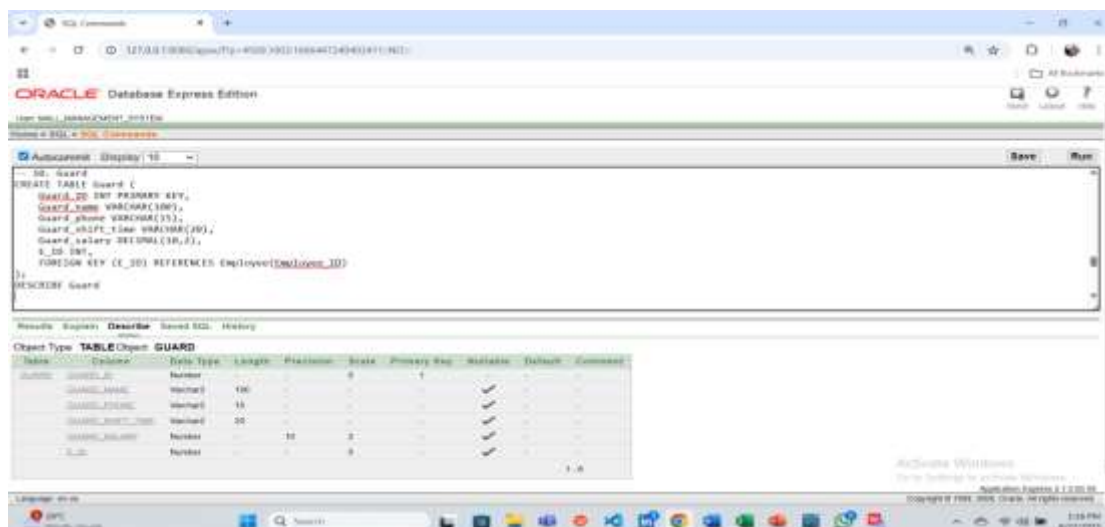
Guard\_salary DECIMAL(10,2),

E\_ID INT,

FOREIGN KEY (E\_ID) REFERENCES Employee(Employee\_ID)

);

DESCRIBE Guard



## 9.Data Insertion

-- Admin

```
INSERT INTO Admin VALUES (1, 'John Smith', '01710000001', 'john.smith@example.com');
```

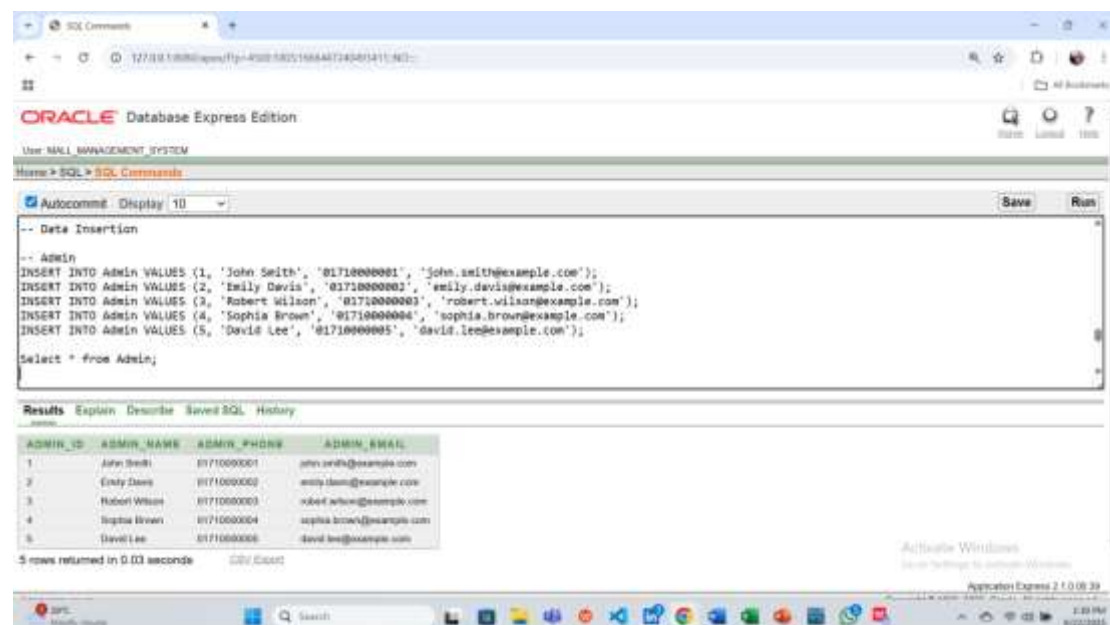
```
INSERT INTO Admin VALUES (2, 'Emily Davis', '01710000002', 'emily.davis@example.com');
```

```
INSERT INTO Admin VALUES (3, 'Robert Wilson', '01710000003', 'robert.wilson@example.com');
```

```
INSERT INTO Admin VALUES (4, 'Sophia Brown', '01710000004', 'sophia.brown@example.com');
```

```
INSERT INTO Admin VALUES (5, 'David Lee', '01710000005', 'david.lee@example.com');
```

```
Select * from Admin;
```



-- Shop

```
INSERT INTO Shop VALUES (101, 'Fashion Hub', 'Open', 'Clothing', 1, 1);
```

```
INSERT INTO Shop VALUES (102, 'Tech World', 'Open', 'Electronics', 2, 2);
```

```
INSERT INTO Shop VALUES (103, 'Book Corner', 'Closed', 'Books', 1, 3);
```

```
INSERT INTO Shop VALUES (104, 'Grocery Mart', 'Open', 'Groceries', 1, 4);
```

```
INSERT INTO Shop VALUES (105, 'Sports Zone', 'Open', 'Sports', 3, 5);
```

```
select * from Shop;
```



The screenshot shows the Oracle Database Express Edition interface. The SQL Command window contains the following code:

```
-- Shop
INSERT INTO Shop VALUES (101, 'Fashion Hub', 'Open', 'Clothing', 1, 1);
INSERT INTO Shop VALUES (102, 'Tech World', 'Open', 'Electronics', 2, 2);
INSERT INTO Shop VALUES (103, 'Book Corner', 'Closed', 'Books', 1, 3);
INSERT INTO Shop VALUES (104, 'Grocery Mart', 'Open', 'Groceries', 1, 4);
INSERT INTO Shop VALUES (105, 'Sports Zone', 'Open', 'Sports', 3, 5);

select * from Shop;
```

The Results window displays the following table:

SHOP_ID	SHOP_NAME	SHOP_STATUS	SHOP_TYPE	SHOP_FLOOR	A_ID
101	Fashion Hub	Open	Clothing	1	1
102	Tech World	Open	Electronics	2	2
103	Book Corner	Closed	Books	1	3
104	Grocery Mart	Open	Groceries	1	4
105	Sports Zone	Open	Sports	3	5

5 rows returned in 0.02 seconds

-- Facility

INSERT INTO Facility VALUES (201, 'Parking Lot', 'Available', 1);

INSERT INTO Facility VALUES (202, 'Food Court', 'Available', 2);

INSERT INTO Facility VALUES (203, 'Play Area', 'Under Maintenance', 3);

INSERT INTO Facility VALUES (204, 'Cinema Hall', 'Available', 4);

INSERT INTO Facility VALUES (205, 'ATM Booth', 'Available', 5);

select \* from Facility;

The screenshot shows the Oracle Database Express Edition interface. The SQL Command window contains the following code:

```
-- Facility
INSERT INTO Facility VALUES (201, 'Parking Lot', 'Available', 1);
INSERT INTO Facility VALUES (202, 'Food Court', 'Available', 2);
INSERT INTO Facility VALUES (203, 'Play Area', 'Under Maintenance', 3);
INSERT INTO Facility VALUES (204, 'Cinema Hall', 'Available', 4);
INSERT INTO Facility VALUES (205, 'ATM Booth', 'Available', 5);

select * from Facility;
```

The Results window displays the following table:

FACILITY_ID	FACILITY_NAME	AVAILABILITY STATUS	A_ID
201	Parking Lot	Available	1
202	Food Court	Available	2
203	Play Area	Under Maintenance	3
204	Cinema Hall	Available	4
205	ATM Booth	Available	5

5 rows returned in 0.03 seconds

-- Employee

```
INSERT INTO Employee VALUES (301, 'Alex Johnson', 25000, 'Manager', '01710000011', 'Morning', 1);
```

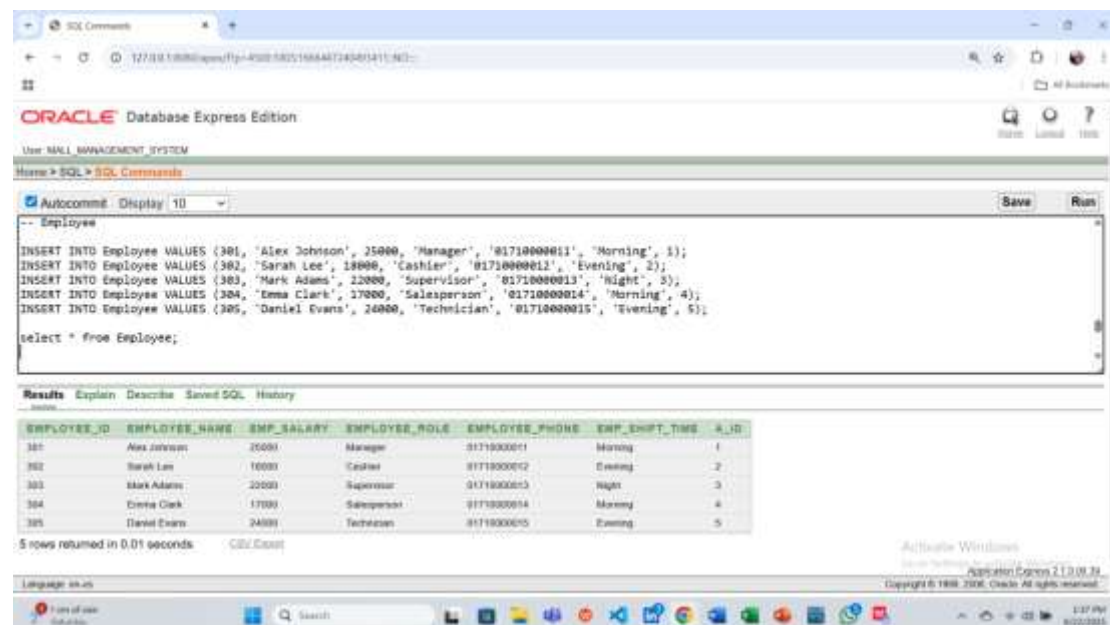
```
INSERT INTO Employee VALUES (302, 'Sarah Lee', 18000, 'Cashier', '01710000012', 'Evening', 2);
```

```
INSERT INTO Employee VALUES (303, 'Mark Adams', 22000, 'Supervisor', '01710000013', 'Night', 3);
```

```
INSERT INTO Employee VALUES (304, 'Emma Clark', 17000, 'Salesperson', '01710000014', 'Morning', 4);
```

```
INSERT INTO Employee VALUES (305, 'Daniel Evans', 24000, 'Technician', '01710000015', 'Evening', 5);
```

```
select * from Employee;
```



The screenshot shows the Oracle Database Express Edition interface. The SQL Command window contains the following commands:

```
-- Employee
INSERT INTO Employee VALUES (301, 'Alex Johnson', 25000, 'Manager', '01710000011', 'Morning', 1);
INSERT INTO Employee VALUES (302, 'Sarah Lee', 18000, 'Cashier', '01710000012', 'Evening', 2);
INSERT INTO Employee VALUES (303, 'Mark Adams', 22000, 'Supervisor', '01710000013', 'Night', 3);
INSERT INTO Employee VALUES (304, 'Emma Clark', 17000, 'Salesperson', '01710000014', 'Morning', 4);
INSERT INTO Employee VALUES (305, 'Daniel Evans', 24000, 'Technician', '01710000015', 'Evening', 5);

select * from Employee;
```

The Results window displays the following data:

EMPLOYEE_ID	EMPLOYEE_NAME	EMP_SALARY	EMPLOYEE_ROLE	EMPLOYEE_PHONE	EMP_SHIFT_TIME	A_ID
301	Alex Johnson	25000	Manager	01710000011	Morning	1
302	Sarah Lee	18000	Cashier	01710000012	Evening	2
303	Mark Adams	22000	Supervisor	01710000013	Night	3
304	Emma Clark	17000	Salesperson	01710000014	Morning	4
305	Daniel Evans	24000	Technician	01710000015	Evening	5

5 rows returned in 0.01 seconds. CPU: 0.00%

-- Request

```
INSERT INTO Request VALUES (401, 'Broken AC in Food Court', 'Manager', 'High', 202);
```

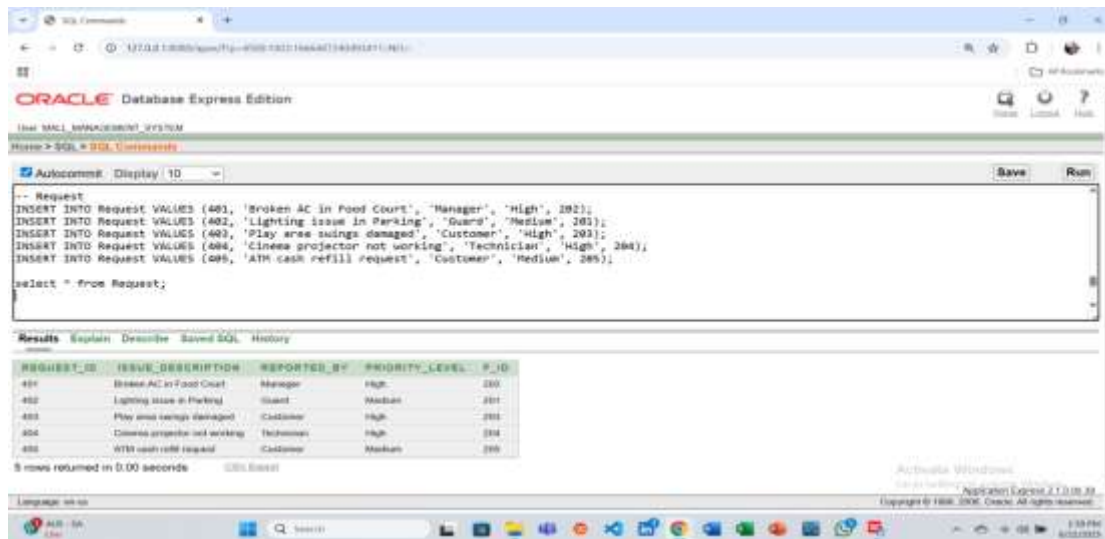
```
INSERT INTO Request VALUES (402, 'Lighting issue in Parking', 'Guard', 'Medium', 201);
```

```
INSERT INTO Request VALUES (403, 'Play area swings damaged', 'Customer', 'High', 203);
```

```
INSERT INTO Request VALUES (404, 'Cinema projector not working', 'Technician', 'High', 204);
```

```
INSERT INTO Request VALUES (405, 'ATM cash refill request', 'Customer', 'Medium', 205);
```

```
select * from Request;
```



-- Tenant

INSERT INTO Tenant VALUES (501, 'Michael Brown', 'michael.brown@example.com', '01710000021', 'NID123456');

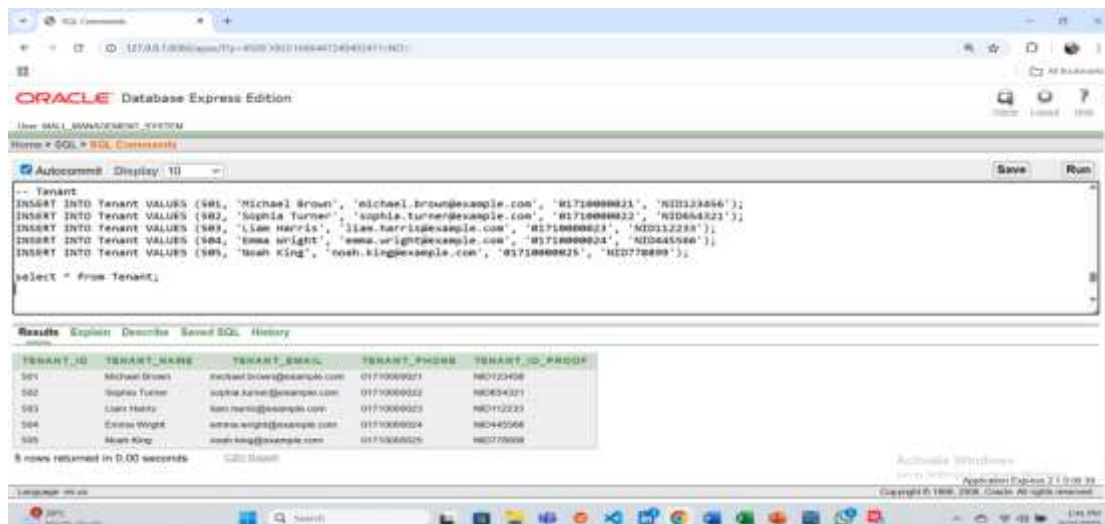
INSERT INTO Tenant VALUES (502, 'Sophia Turner', 'sophia.turner@example.com', '01710000022', 'NID654321');

INSERT INTO Tenant VALUES (503, 'Liam Harris', 'liam.harris@example.com', '01710000023', 'NID112233');

INSERT INTO Tenant VALUES (504, 'Emma Wright', 'emma.wright@example.com', '01710000024', 'NID445566');

INSERT INTO Tenant VALUES (505, 'Noah King', 'noah.king@example.com', '01710000025', 'NID778899');

select \* from Tenant;



-- Payment

```
INSERT INTO Payment VALUES (601, 15000, TO_DATE('2025-08-01','YYYY-MM-DD'),  
'Cash', 501);
```

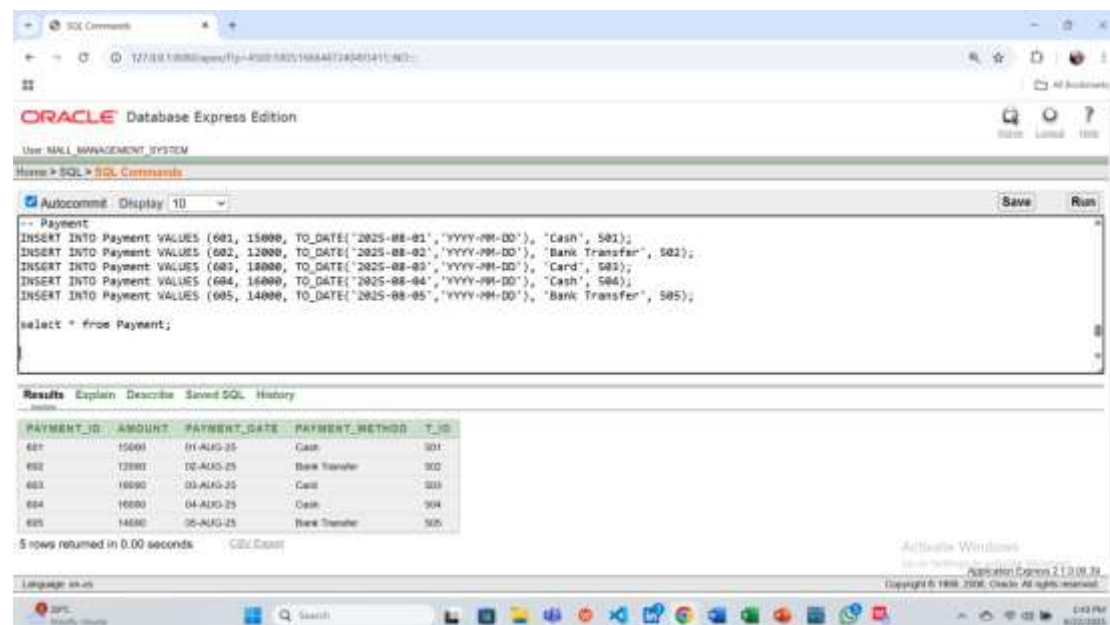
```
INSERT INTO Payment VALUES (602, 12000, TO_DATE('2025-08-02','YYYY-MM-DD'),  
'Bank Transfer', 502);
```

```
INSERT INTO Payment VALUES (603, 18000, TO_DATE('2025-08-03','YYYY-MM-DD'),  
'Card', 503);
```

```
INSERT INTO Payment VALUES (604, 16000, TO_DATE('2025-08-04','YYYY-MM-DD'),  
'Cash', 504);
```

```
INSERT INTO Payment VALUES (605, 14000, TO_DATE('2025-08-05','YYYY-MM-DD'),  
'Bank Transfer', 505);
```

```
select * from Payment;
```



-- Booking

```
INSERT INTO Booking VALUES
```

```
(701, 20000, TO_DATE('2025-08-01','YYYY-MM-DD'), TO_DATE('2026-07-31','YYYY-MM-DD'), 'Paid', 101);
```

```
INSERT INTO Booking VALUES
```

```
(702, 25000, TO_DATE('2025-08-02','YYYY-MM-DD'), TO_DATE('2026-08-01','YYYY-MM-DD'), 'Unpaid', 102);
```

```
INSERT INTO Booking VALUES
```

(703, 18000, TO\_DATE('2025-08-03','YYYY-MM-DD'), TO\_DATE('2026-08-02','YYYY-MM-DD'), 'Paid', 103);

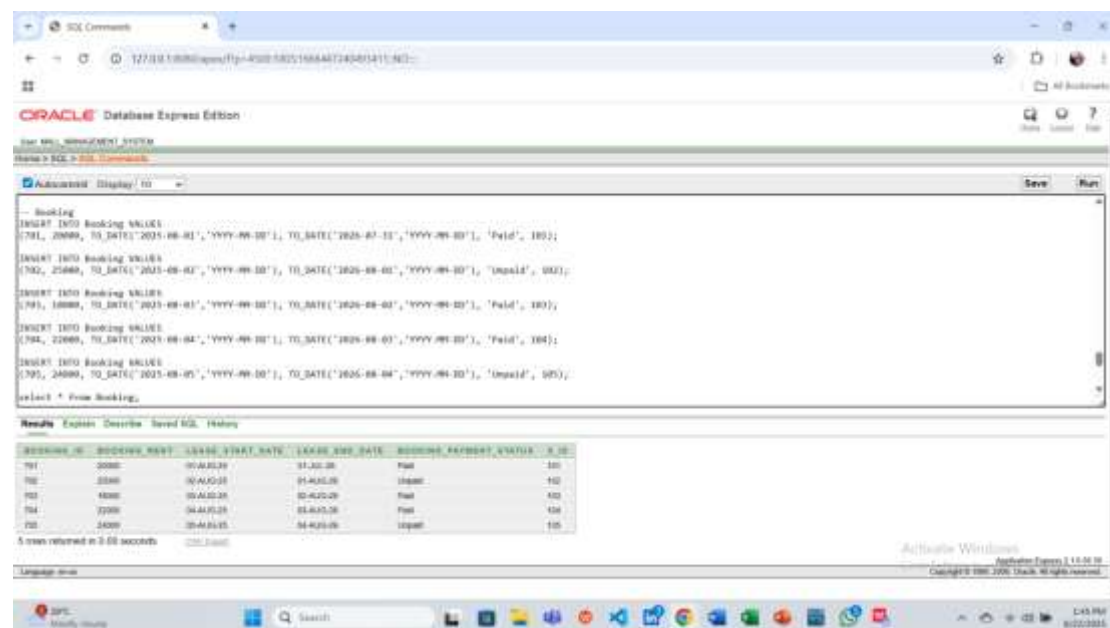
INSERT INTO Booking VALUES

(704, 22000, TO\_DATE('2025-08-04','YYYY-MM-DD'), TO\_DATE('2026-08-03','YYYY-MM-DD'), 'Paid', 104);

INSERT INTO Booking VALUES

(705, 24000, TO\_DATE('2025-08-05','YYYY-MM-DD'), TO\_DATE('2026-08-04','YYYY-MM-DD'), 'Unpaid', 105);

select \* from Booking;



-- Customer

INSERT INTO Customer VALUES (801, 'Liam Wilson', 'liam.wilson@example.com', '01710000031', TO\_DATE('2025-08-01','YYYY-MM-DD'), 'Great service', 101);

INSERT INTO Customer VALUES (802, 'Olivia White', 'olivia.white@example.com', '01710000032', TO\_DATE('2025-08-02','YYYY-MM-DD'), 'Nice products', 102);

INSERT INTO Customer VALUES (803, 'Noah Thompson', 'noah.thompson@example.com', '01710000033', TO\_DATE('2025-08-03','YYYY-MM-DD'), 'Loved the store', 103);

INSERT INTO Customer VALUES (804, 'Emma Martinez', 'emma.martinez@example.com', '01710000034', TO\_DATE('2025-08-04','YYYY-MM-DD'), 'Helpful staff', 104);

INSERT INTO Customer VALUES (805, 'Lucas Robinson', 'lucas.robinson@example.com', '01710000035', TO\_DATE('2025-08-05','YYYY-MM-DD'), 'Affordable prices', 105);

select \* from Customer;

The screenshot shows the Oracle Database Express Edition interface. The SQL Command window contains the following code:

```
-- Customer
INSERT INTO Customer VALUES (801, 'Liam Wilson', 'liam.wilson@example.com', '01710000011', TO_DATE('2025-08-01', 'YYYY-MM-DD'), 'Great service', 301);
INSERT INTO Customer VALUES (802, 'Olivia White', 'olivia.white@example.com', '01710000012', TO_DATE('2025-08-02', 'YYYY-MM-DD'), 'Nice products', 302);
INSERT INTO Customer VALUES (803, 'Noah Thompson', 'noah.thompson@example.com', '01710000013', TO_DATE('2025-08-03', 'YYYY-MM-DD'), 'Loved the store', 303);
INSERT INTO Customer VALUES (804, 'Emma Martinez', 'emma.martinez@example.com', '01710000014', TO_DATE('2025-08-04', 'YYYY-MM-DD'), 'Helpful staff', 304);
INSERT INTO Customer VALUES (805, 'Lucas Robinson', 'lucas.robinson@example.com', '01710000015', TO_DATE('2025-08-05', 'YYYY-MM-DD'), 'Affordable prices', 305);

select * from Customer;
```

The Results window displays the following data:

CUSTOMER_ID	CUSTOMER_NAME	CUSTOMER_EMAIL	CUSTOMER_PHONE	VISIT_DATE	FEEDBACK	R_ID
801	Liam Wilson	liam.wilson@example.com	01710000011	01-AUG-25	Great service	301
802	Olivia White	olivia.white@example.com	01710000012	02-AUG-25	Nice products	302
803	Noah Thompson	noah.thompson@example.com	01710000013	03-AUG-25	Loved the store	303
804	Emma Martinez	emma.martinez@example.com	01710000014	04-AUG-25	Helpful staff	304
805	Lucas Robinson	lucas.robinson@example.com	01710000015	05-AUG-25	Affordable prices	305

5 rows returned in 0.00 seconds

-- Guard

INSERT INTO Guard VALUES (901, 'David Miller', '01710000041', 'Night', 15000, 301);

INSERT INTO Guard VALUES (902, 'Emma Scott', '01710000042', 'Day', 14000, 302);

INSERT INTO Guard VALUES (903, 'Liam Parker', '01710000043', 'Night', 15500, 303);

INSERT INTO Guard VALUES (904, 'Sophia Hall', '01710000044', 'Day', 14500, 304);

INSERT INTO Guard VALUES (905, 'Ethan Lewis', '01710000045', 'Night', 16000, 305);

select \* from Guard;

The screenshot shows the Oracle Database Express Edition interface. The SQL Command window contains the following code:

```
-- Guard
INSERT INTO Guard VALUES (901, 'David Miller', '01710000041', 'Night', 15000, 301);
INSERT INTO Guard VALUES (902, 'Emma Scott', '01710000042', 'Day', 14000, 302);
INSERT INTO Guard VALUES (903, 'Liam Parker', '01710000043', 'Night', 15500, 303);
INSERT INTO Guard VALUES (904, 'Sophia Hall', '01710000044', 'Day', 14500, 304);
INSERT INTO Guard VALUES (905, 'Ethan Lewis', '01710000045', 'Night', 16000, 305);

select * from Guard;
```

The Results window displays the following data:

GUARD_ID	GUARD_NAME	GUARD_PHONE	GUARD_SHIFT_TIME	GUARD_SALARY	R_ID
901	David Miller	01710000041	Night	15000	301
902	Emma Scott	01710000042	Day	14000	302
903	Liam Parker	01710000043	Night	15500	303
904	Sophia Hall	01710000044	Day	14500	304
905	Ethan Lewis	01710000045	Night	16000	305

5 rows returned in 0.00 seconds



## 10. Query Writing Using PL/SQL

### 11. Basic PL/SQL

#### - Using 2 variables

**Question 1: Display the salary of Employee\_ID = 301 using a variable.**

**Answer:**

DECLARE

    v\_emp\_salary NUMBER(10,2);

BEGIN

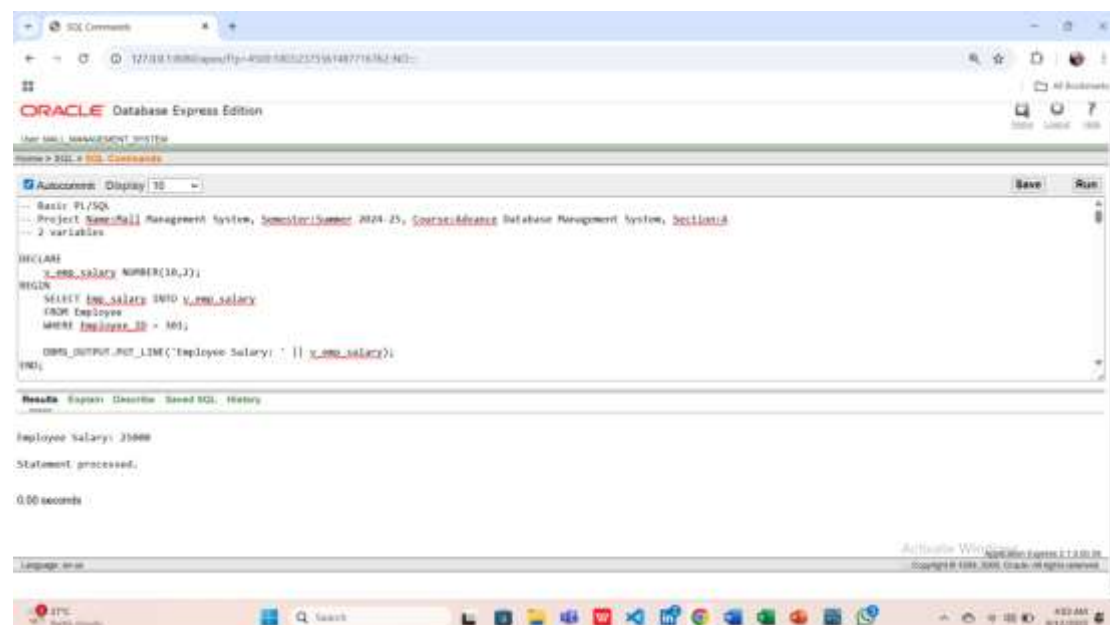
    SELECT Emp\_salary INTO v\_emp\_salary

    FROM Employee

    WHERE Employee\_ID = 301;

    DBMS\_OUTPUT.PUT\_LINE('Employee Salary: ' || v\_emp\_salary);

END;



**Question 2: Display the total salary of guards reporting to Employee\_ID = 301 using a variable.**

**Answer:**

DECLARE



```
v_guard_salary NUMBER(10,2);
```

```
BEGIN
```

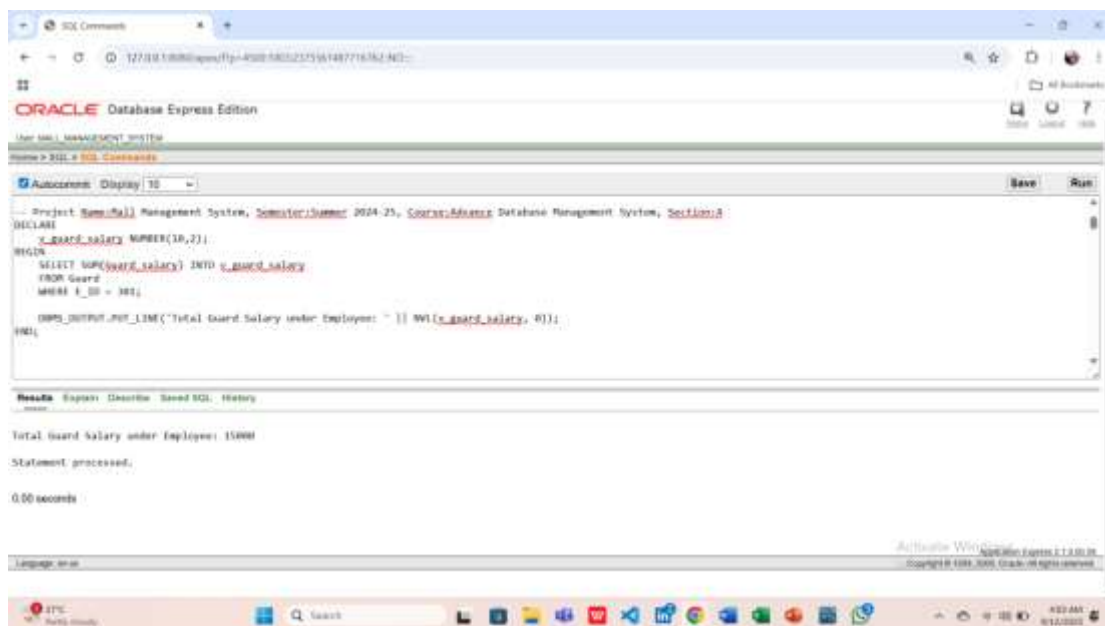
```
SELECT SUM(Guard_salary) INTO v_guard_salary
```

```
FROM Guard
```

```
WHERE E_ID = 301;
```

```
DBMS_OUTPUT.PUT_LINE('Total Guard Salary under Employee: ' || NVL(v_guard_salary, 0));
```

```
END;
```



## -Using 2 Operators

### **Question 1: Calculate Balance using Subtraction Operator**

**Answer:**

```
DECLARE
```

```
v_payment_amount NUMBER(10,2);
```

```
v_booking_rent NUMBER(10,2);
```

```
v_balance NUMBER(10,2);
```

```
BEGIN
```

```
SELECT Amount INTO v_payment_amount
```

FROM Payment

WHERE T\_ID = 501;

SELECT Booking\_rent INTO v\_booking\_rent

FROM Booking

WHERE S\_ID = 101;

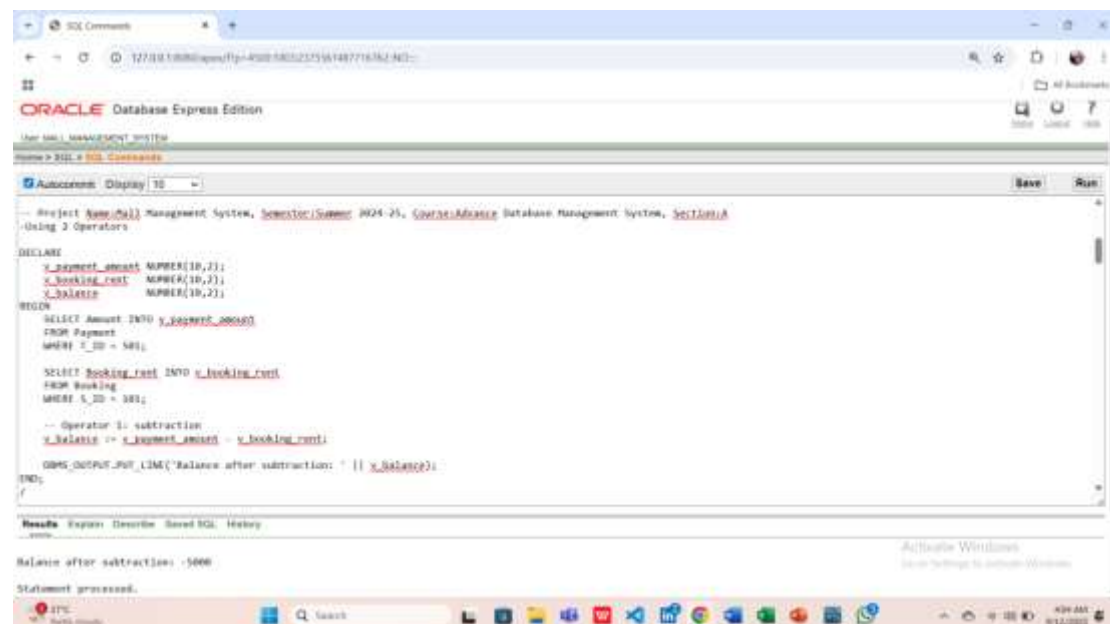
-- Operator 1: subtraction

v\_balance := v\_payment\_amount - v\_booking\_rent;

DBMS\_OUTPUT.PUT\_LINE('Balance after subtraction: ' || v\_balance);

END;

/



## Question 2: Check if Balance is Positive or Negative using Comparison Operator

### Answer:

DECLARE

v\_payment\_amount NUMBER(10,2);

v\_booking\_rent NUMBER(10,2);

v\_balance NUMBER(10,2);

BEGIN

SELECT Amount INTO v\_payment\_amount

```

FROM Payment

WHERE T_ID = 501;

SELECT Booking_rent INTO v_booking_rent

FROM Booking

WHERE S_ID = 101;

v_balance := v_payment_amount - v_booking_rent;

-- Operator 2: comparison

IF v_balance >= 0 THEN

    DBMS_OUTPUT.PUT_LINE('Balance is Positive: ' || v_balance);

ELSE

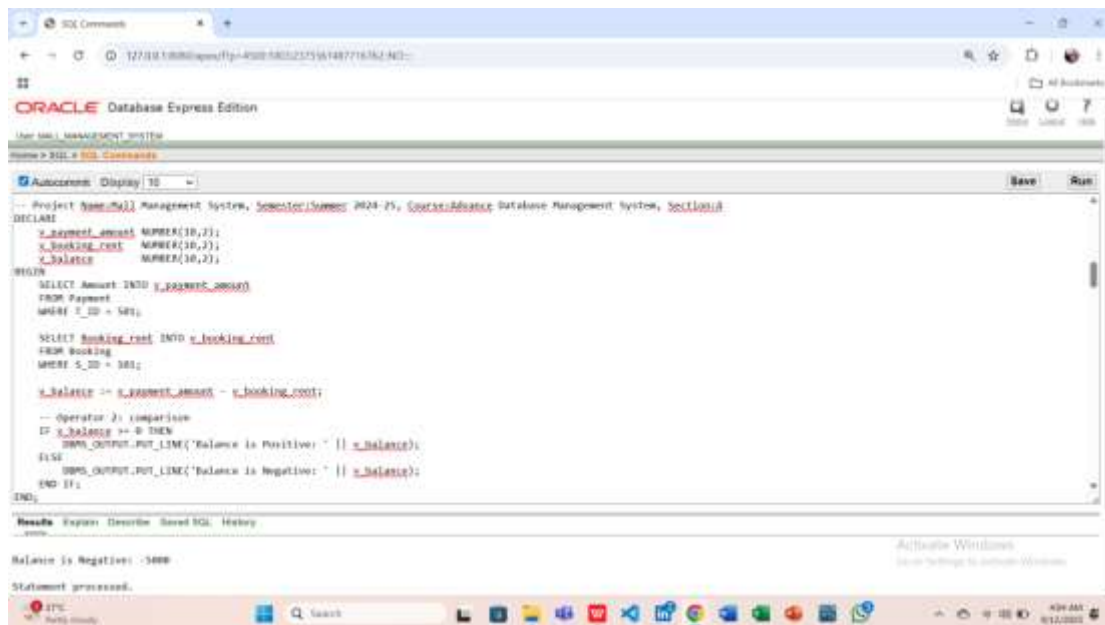
    DBMS_OUTPUT.PUT_LINE('Balance is Negative: ' || v_balance);

END IF;

END;

/

```



### -Using 2 single-row function

**Question 1: Display the total payment made by a tenant.**

**Answer:**

DECLARE

v\_payment\_amount NUMBER(10,2);

BEGIN

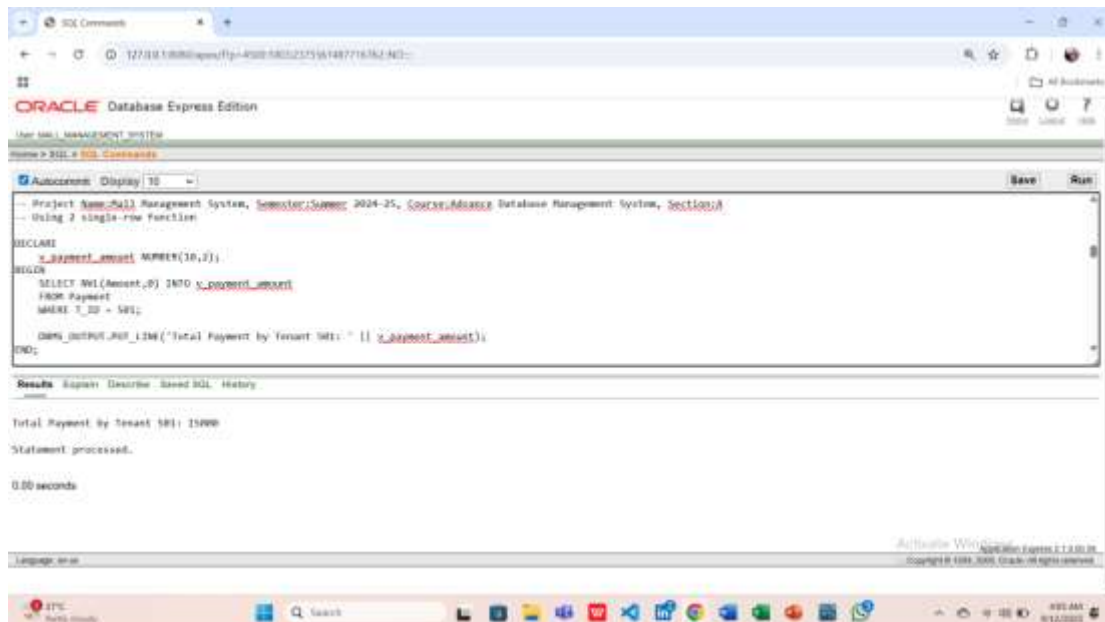
SELECT NVL(Amount,0) INTO v\_payment\_amount

FROM Payment

WHERE T\_ID = 501;

DBMS\_OUTPUT.PUT\_LINE('Total Payment by Tenant 501: ' || v\_payment\_amount);

END;



**Question 2: Display the booking rent for a shop.**

**Answer:**

DECLARE

v\_booking\_rent NUMBER(10,2);

BEGIN

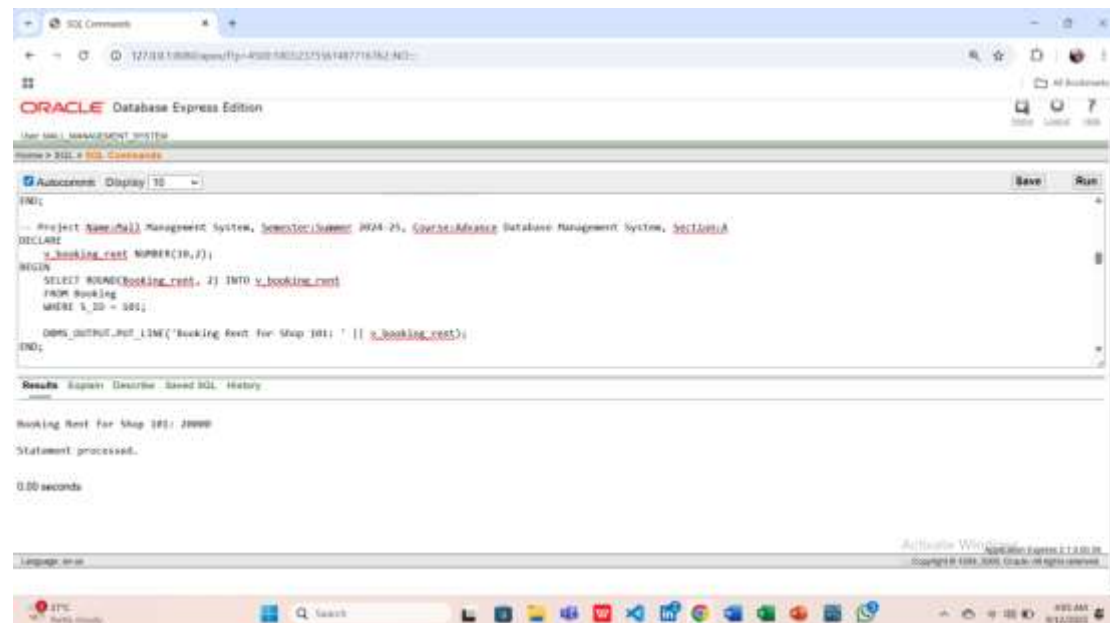
SELECT ROUND(Booking\_rent, 2) INTO v\_booking\_rent

FROM Booking

WHERE S\_ID = 101;

```
DBMS_OUTPUT.PUT_LINE('Booking Rent for Shop 101: ' || v_booking_rent);
```

```
END;
```



## - Using 2 group function

### **Question 1: Find the Total Payment Amount**

**Answer:**

```
DECLARE
```

```
    v_total NUMBER(10,2);
```

```
BEGIN
```

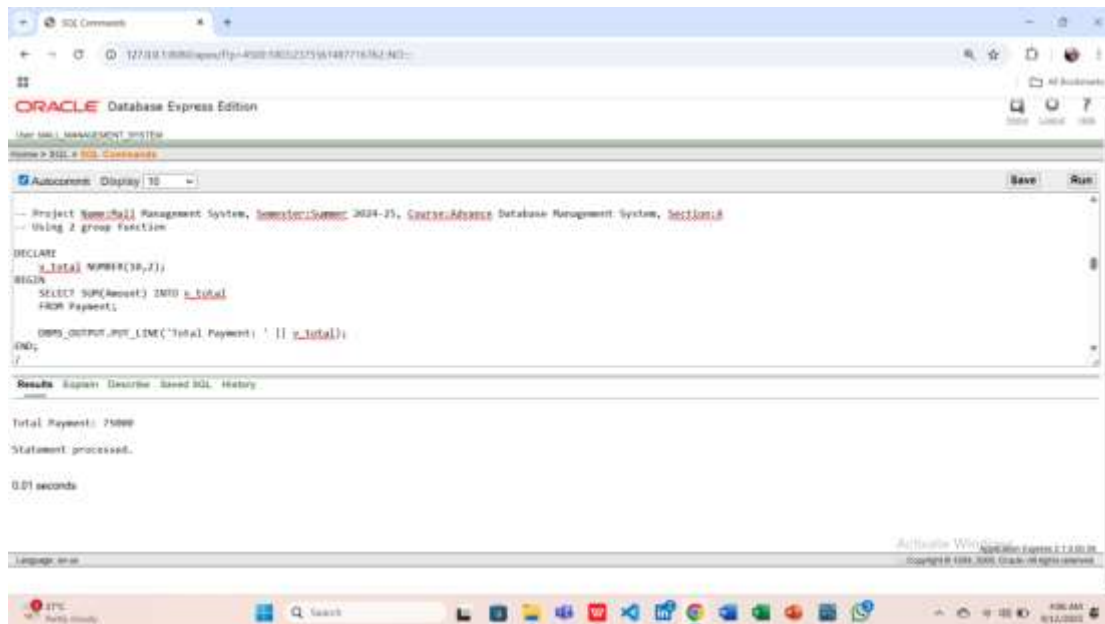
```
    SELECT SUM(Amount) INTO v_total
```

```
    FROM Payment;
```

```
    DBMS_OUTPUT.PUT_LINE('Total Payment: ' || v_total);
```

```
END;
```

```
/
```



## Question 2: Find the Average Payment Amount

### Answer:

DECLARE

v\_avg NUMBER(10,2);

BEGIN

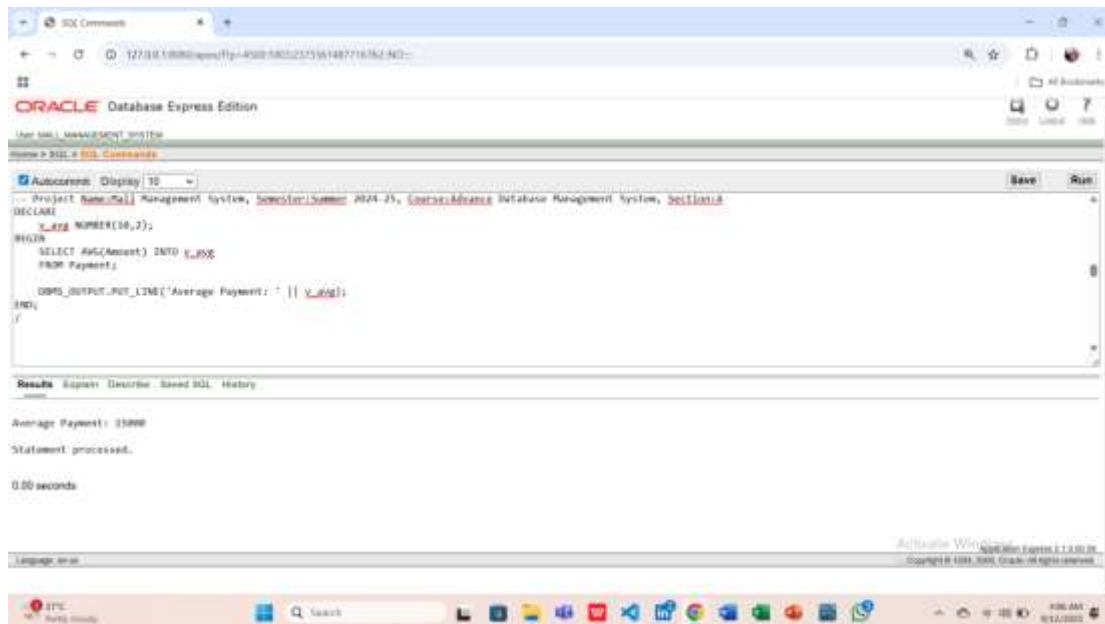
SELECT AVG(Amount) INTO v\_avg

FROM Payment;

DBMS\_OUTPUT.PUT\_LINE('Average Payment: ' || v\_avg);

END;

/



## -Using 2 loop

### **Question 1: Display Shop IDs from 101 to 105 (Numeric FOR Loop)**

**Answer:**

DECLARE

BEGIN

-- Loop 1: Numeric FOR loop

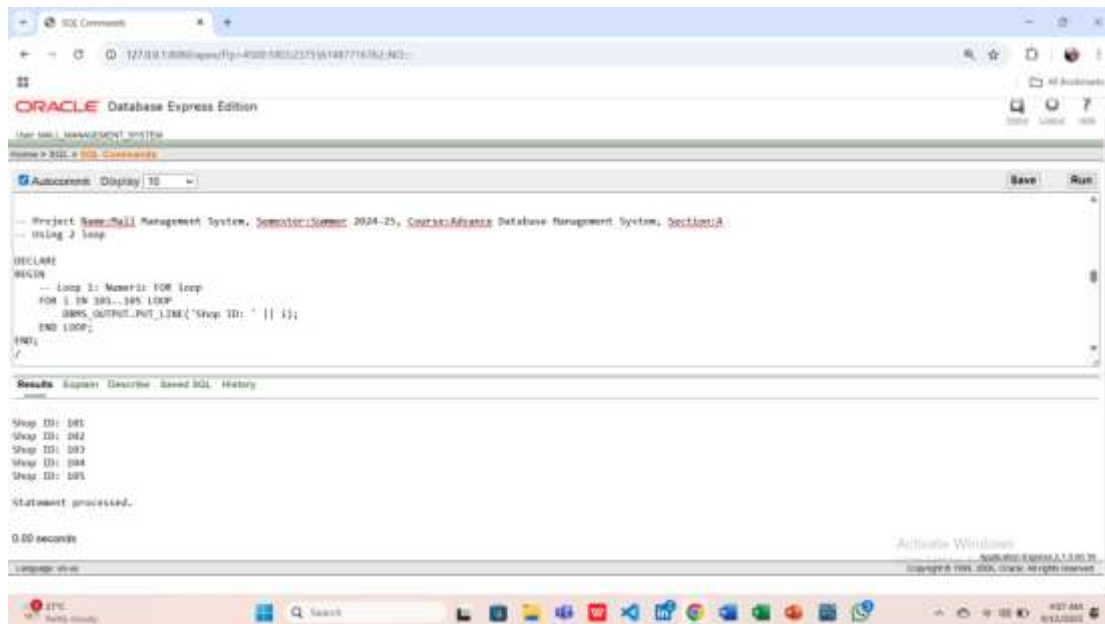
FOR i IN 101..105 LOOP

    DBMS\_OUTPUT.PUT\_LINE('Shop ID: ' || i);

END LOOP;

END;

/



## Question 2: Display Shop IDs from 201 to 205 (WHILE Loop)

**Answer:**

DECLARE

    i NUMBER := 201;

BEGIN

    -- Loop 2: WHILE loop

    WHILE i <= 205 LOOP

        DBMS\_OUTPUT.PUT\_LINE('Shop ID: ' || i);

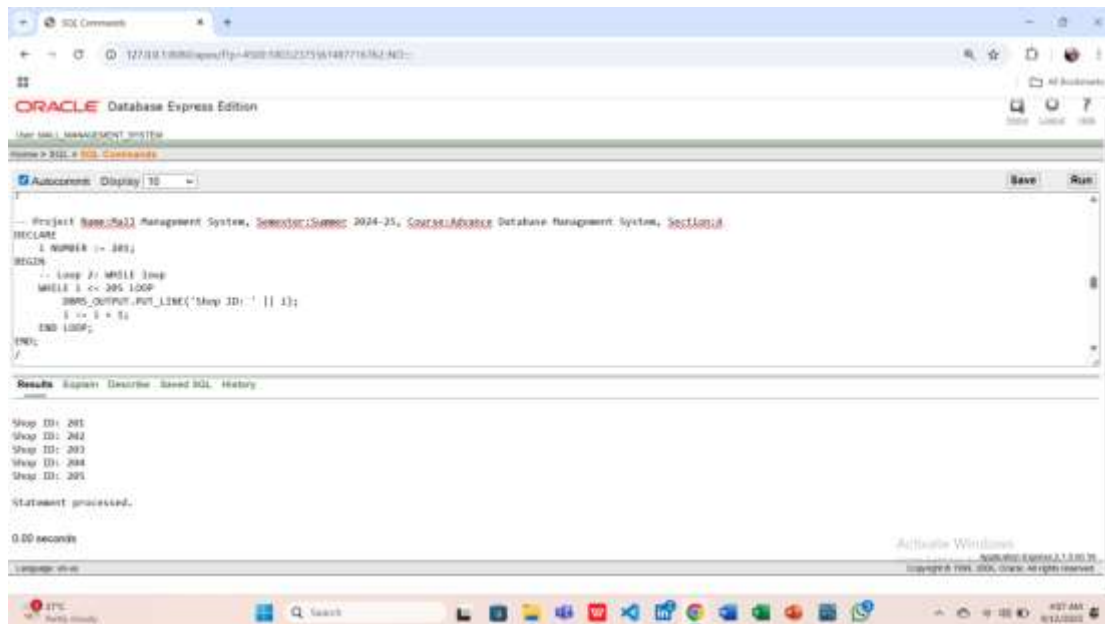
        i := i + 1;

    END LOOP;

END;

/





## -2 conditional statements

### **Question 1: Check Booking Payment Status using IF–ELSE**

#### **Answer:**

DECLARE

    v\_status Booking.Booking\_payment\_status%TYPE;

BEGIN

    SELECT Booking\_payment\_status INTO v\_status

    FROM Booking

    WHERE Booking\_ID = 702;

    -- Conditional statement 1: IF–ELSE

    IF v\_status = 'Paid' THEN

        DBMS\_OUTPUT.PUT\_LINE('Booking is Paid');

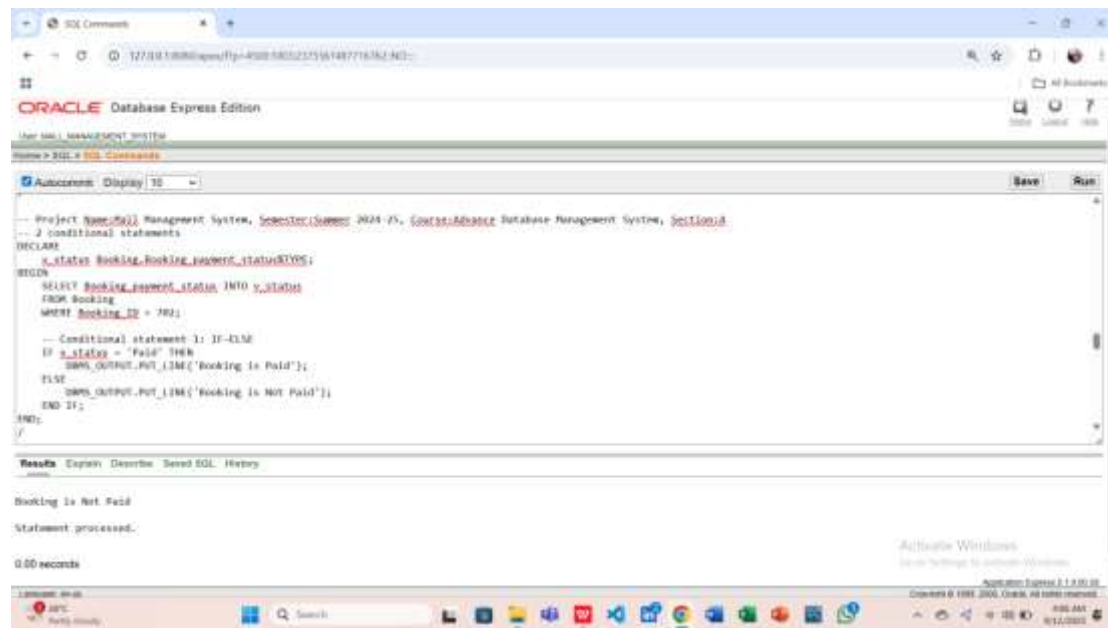
    ELSE

        DBMS\_OUTPUT.PUT\_LINE('Booking is Not Paid');

    END IF;

END;

/



## Question 2: Check Booking Payment Status using CASE

### Answer:

DECLARE

v\_status Booking.Booking\_payment\_status%TYPE;

BEGIN

SELECT Booking\_payment\_status INTO v\_status

FROM Booking

WHERE Booking\_ID = 702;

-- Conditional statement 2: CASE

CASE v\_status

WHEN 'Paid' THEN

DBMS\_OUTPUT.PUT\_LINE('Status = Paid');

WHEN 'Unpaid' THEN

DBMS\_OUTPUT.PUT\_LINE('Status = Unpaid');

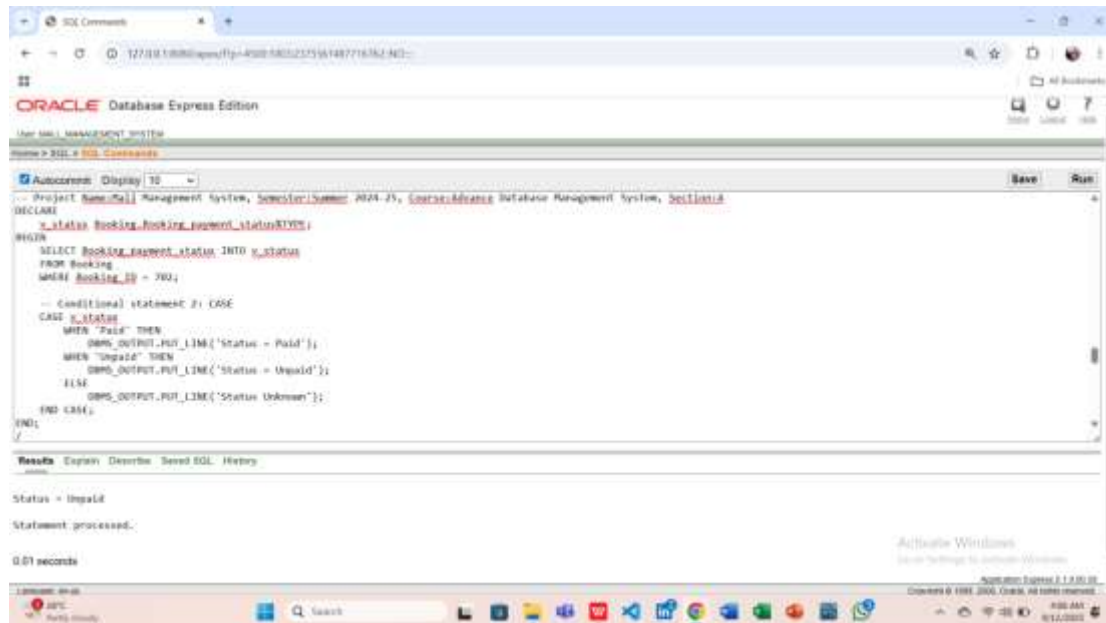
ELSE

DBMS\_OUTPUT.PUT\_LINE('Status Unknown');

END CASE;

END;

/



## -2 subquery

**Question 1: Find Employee Name with Maximum Salary (Subquery in WHERE clause)**

**Answer:**

DECLARE

v\_emp\_name Employee.Employee\_name%TYPE;

BEGIN

SELECT Employee\_name INTO v\_emp\_name

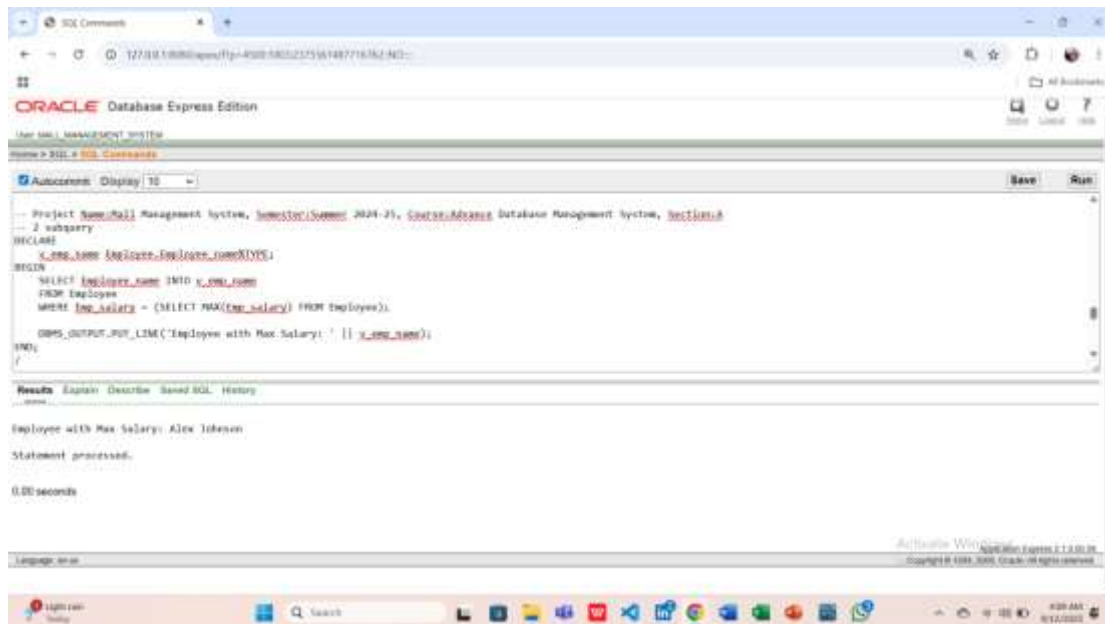
FROM Employee

WHERE Emp\_salary = (SELECT MAX(Emp\_salary) FROM Employee);

DBMS\_OUTPUT.PUT\_LINE('Employee with Max Salary: ' || v\_emp\_name);

END;

/



**Question 2: Find Shop Managed by Admin 'Emily Davis' (Subquery in WHERE clause)**

**Answer:**

DECLARE

    v\_shop\_name Shop.Shop\_name%TYPE;

BEGIN

    SELECT Shop\_name INTO v\_shop\_name

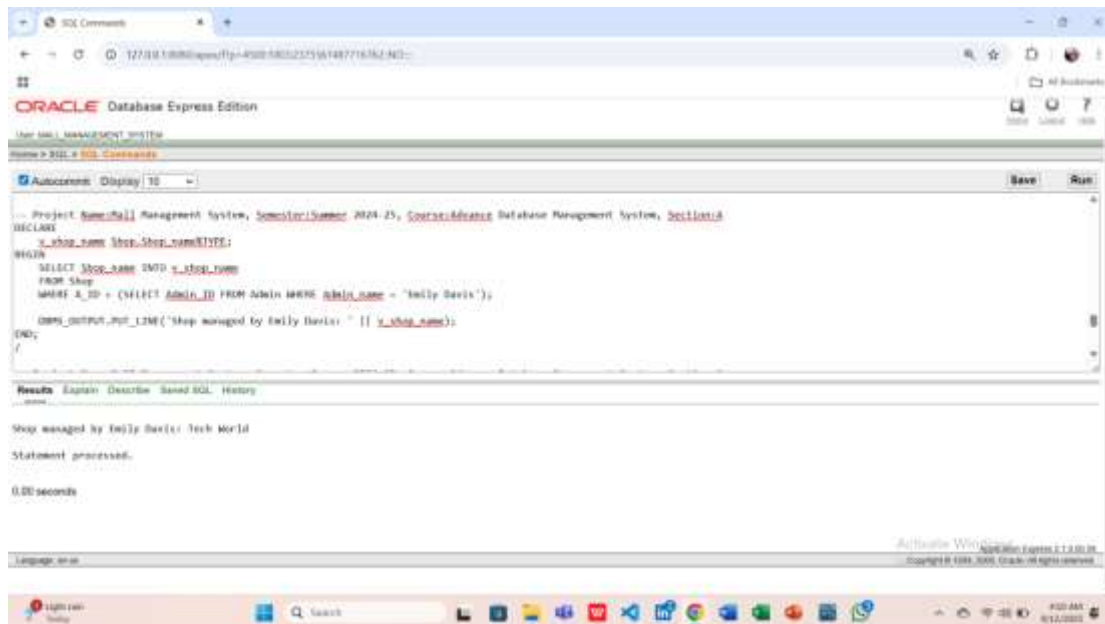
    FROM Shop

    WHERE A\_ID = (SELECT Admin\_ID FROM Admin WHERE Admin\_name = 'Emily Davis');

    DBMS\_OUTPUT.PUT\_LINE('Shop managed by Emily Davis: ' || v\_shop\_name);

END;

/



## -2 joining

### Question 1: Display One Employee and Guard Name (JOIN)

#### Answer:

DECLARE

    v\_emp\_name Employee.Employee\_name%TYPE;

    v\_guard\_name Guard.Guard\_name%TYPE;

BEGIN

    SELECT E.Employee\_name, G.Guard\_name

    INTO v\_emp\_name, v\_guard\_name

    FROM Employee E

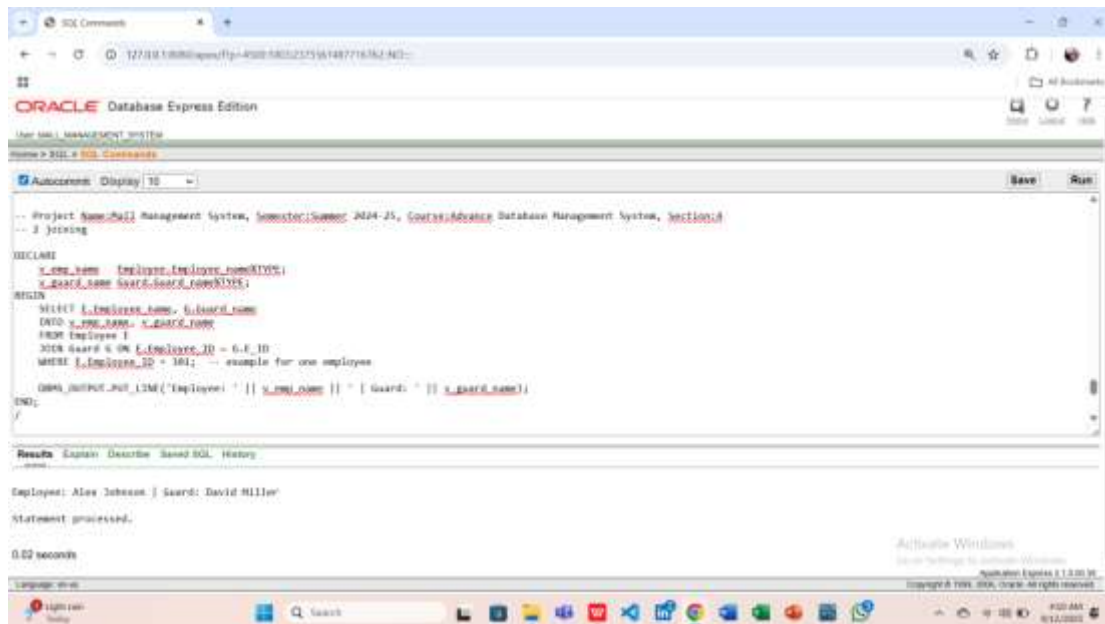
    JOIN Guard G ON E.Employee\_ID = G.E\_ID

    WHERE E.Employee\_ID = 301; -- example for one employee

    DBMS\_OUTPUT.PUT\_LINE('Employee: ' || v\_emp\_name || ' | Guard: ' || v\_guard\_name);

END;

/



## Question 2: Display One Shop and Admin Name (JOIN)

### Answer:

DECLARE

    v\_shop\_name Shop.Shop\_name%TYPE;

    v\_admin\_name Admin.Admin\_name%TYPE;

BEGIN

    SELECT S.Shop\_name, A.Admin\_name

    INTO v\_shop\_name, v\_admin\_name

    FROM Shop S

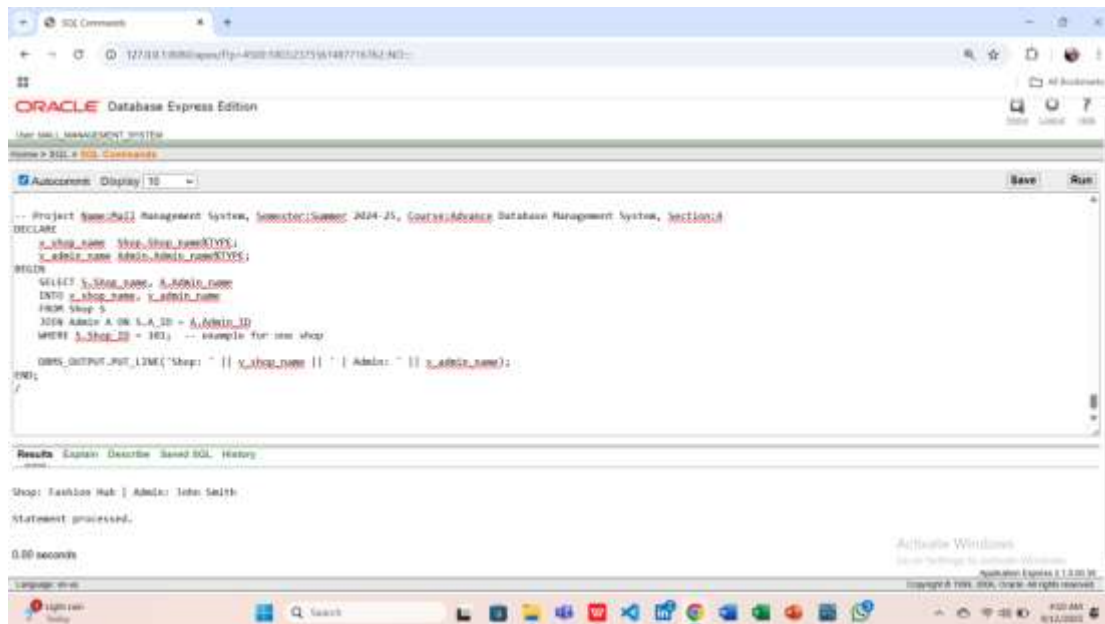
    JOIN Admin A ON S.A\_ID = A.Admin\_ID

    WHERE S.Shop\_ID = 101; -- example for one shop

    DBMS\_OUTPUT.PUT\_LINE('Shop: ' || v\_shop\_name || ' | Admin: ' || v\_admin\_name);

END;

/



## 12. Advance PL/SQL (with Exception Handling)

### -2 stored function

**Question-1: Write a stored function to calculate total salary of an Employee including Guards under them.**

**Answer:**

CREATE OR REPLACE FUNCTION Total\_Salary(emp\_id IN NUMBER)

RETURN NUMBER

IS

v\_emp\_salary NUMBER(10,2);

v\_guard\_salary NUMBER(10,2);

v\_total NUMBER(10,2);

BEGIN

-- Get employee salary

SELECT Emp\_salary

INTO v\_emp\_salary

FROM Employee

```

WHERE Employee_ID = emp_id;

-- Get guard salary under this employee

SELECT NVL(SUM(Guard_salary),0)

INTO v_guard_salary

FROM Guard

WHERE E_ID = emp_id;

-- Calculate total

v_total := v_emp_salary + v_guard_salary;

RETURN v_total;

EXCEPTION

WHEN NO_DATA_FOUND THEN

    DBMS_OUTPUT.PUT_LINE('Error: Employee not found for ID = ' || emp_id);

    RETURN 0; -- returning 0 if no employee exists

WHEN OTHERS THEN

    DBMS_OUTPUT.PUT_LINE('Unexpected error: ' || SQLERRM);

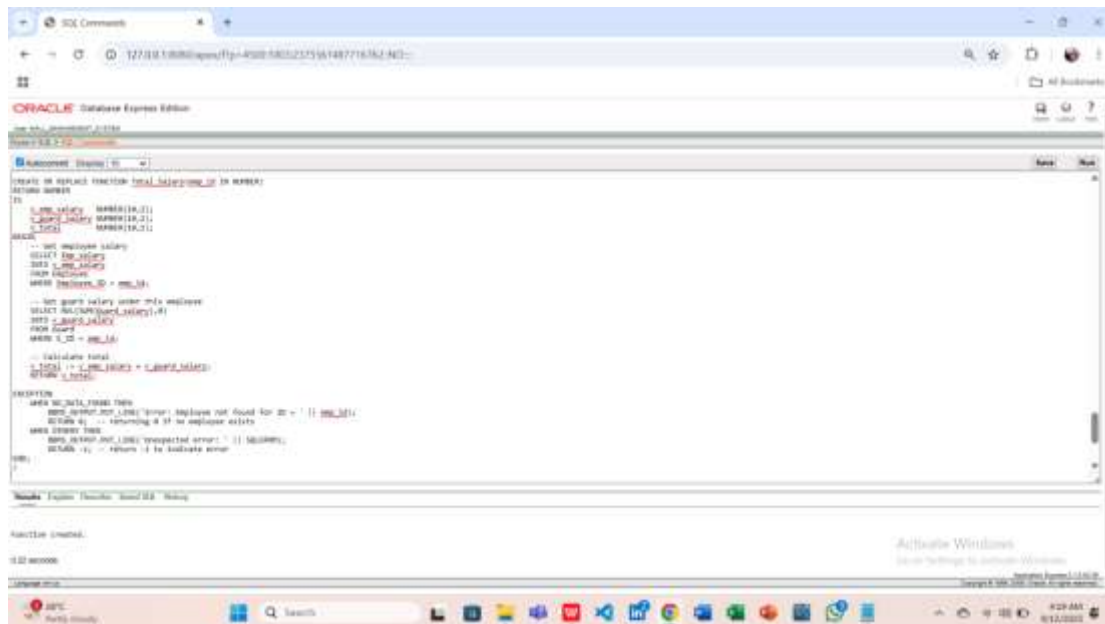
    RETURN -1; -- return -1 to indicate error

END;

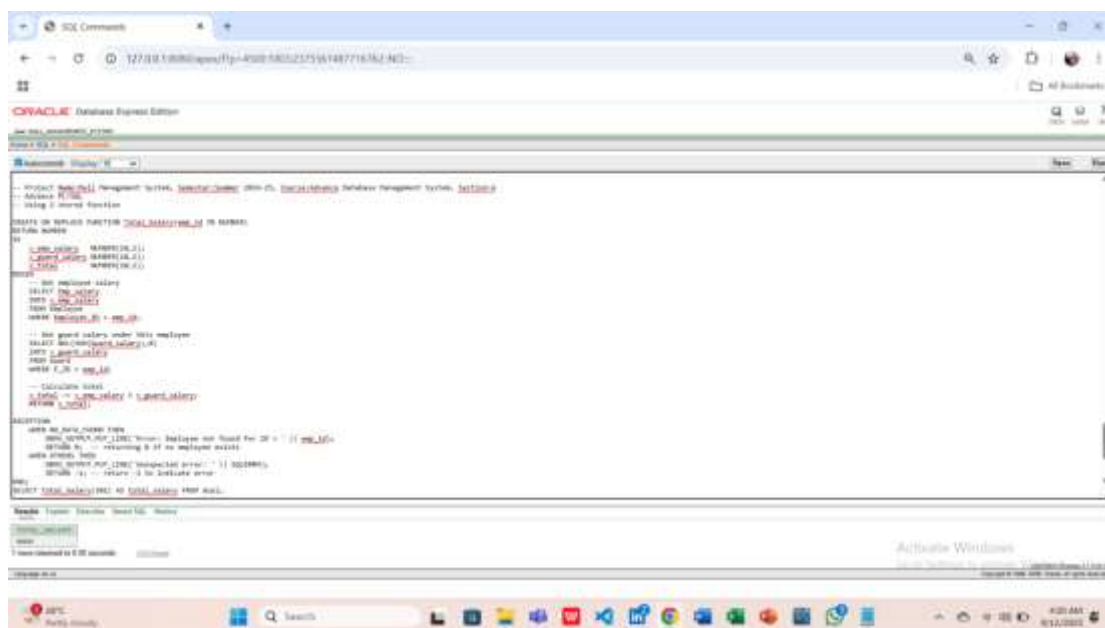
/

```





SELECT Total\_Salary(301) AS total\_salary FROM dual;



**Question-2: Write a stored function to calculate total payment made by a tenant.**

**Answer:**

CREATE OR REPLACE FUNCTION Tenant\_Total\_Payment(t\_id IN NUMBER)

RETURN NUMBER

IS

v\_total NUMBER(10,2);

BEGIN

-- Calculate total payments made by tenant

SELECT NVL(SUM(Amount),0)

INTO v\_total

FROM Payment

WHERE T\_ID = t\_id;

RETURN v\_total;

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

DBMS\_OUTPUT.PUT\_LINE('Error: No payment records found for Tenant ID = ' || t\_id);

RETURN 0; -- return 0 if no rows found

WHEN OTHERS THEN

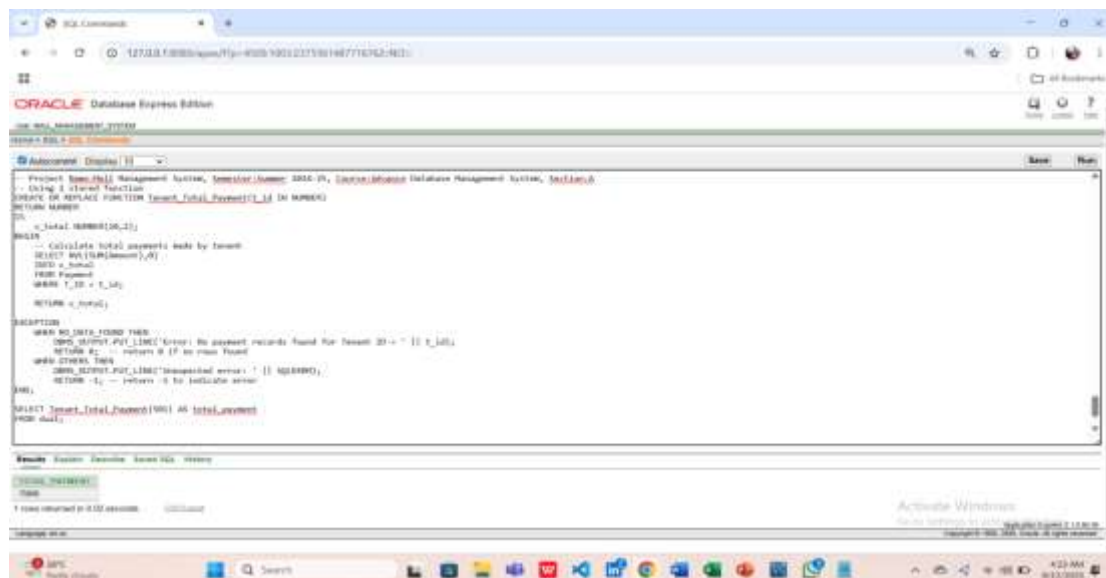
DBMS\_OUTPUT.PUT\_LINE('Unexpected error: ' || SQLERRM);

RETURN -1; -- return -1 to indicate error

END;

SELECT Tenant\_Total\_Payment(501) AS total\_payment

FROM dual;



```
-- Calculate total payments made by tenant
SELECT NVL(SUM(Amount),0)
INTO v_total
FROM Payment
WHERE T_ID = t_id;

RETURN v_total;

EXCEPTION
WHEN NO_DATA_FOUND THEN
    DBMS_OUTPUT.PUT_LINE('Error: No payment records found for Tenant ID = ' || t_id);
    RETURN 0; -- return 0 if no rows found
WHEN OTHERS THEN
    DBMS_OUTPUT.PUT_LINE('Unexpected error: ' || SQLERRM);
    RETURN -1; -- return -1 to indicate error
END;

SELECT Tenant_Total_Payment(501) AS total_payment
FROM dual;
```

Results: Execution Summary: Success SQL: History

1 row returned in 0.002 seconds

## -2 stored procedure

**Question-1: Write a stored procedure to display all Shops and their Admin names.**

**Answer:**

```
CREATE OR REPLACE PROCEDURE Show_Shops_Admins
```

```
IS
```

```
BEGIN
```

```
-- Loop through all shops and their admins
```

```
FOR rec IN (SELECT S.Shop_name, A.Admin_name
```

```
FROM Shop S
```

```
JOIN Admin A ON S.A_ID = A.Admin_ID) LOOP
```

```
DBMS_OUTPUT.PUT_LINE('Shop: ' || rec.Shop_name || ' | Admin: ' || rec.Admin_name);
```

```
END LOOP;
```

```
EXCEPTION
```

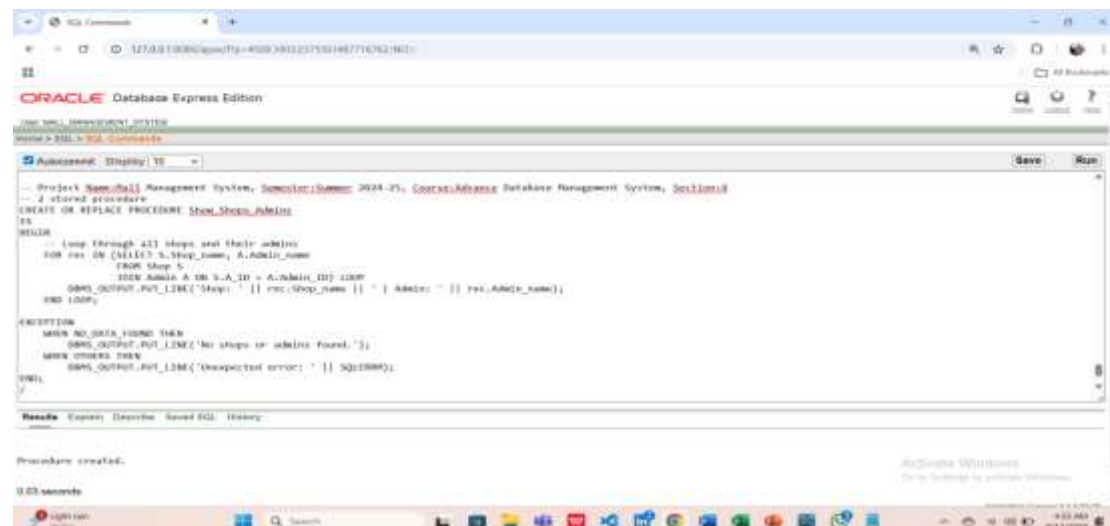
```
WHEN NO_DATA_FOUND THEN
```

```
DBMS_OUTPUT.PUT_LINE('No shops or admins found.');
```

```
WHEN OTHERS THEN
```

```
DBMS_OUTPUT.PUT_LINE('Unexpected error: ' || SQLERRM);
```

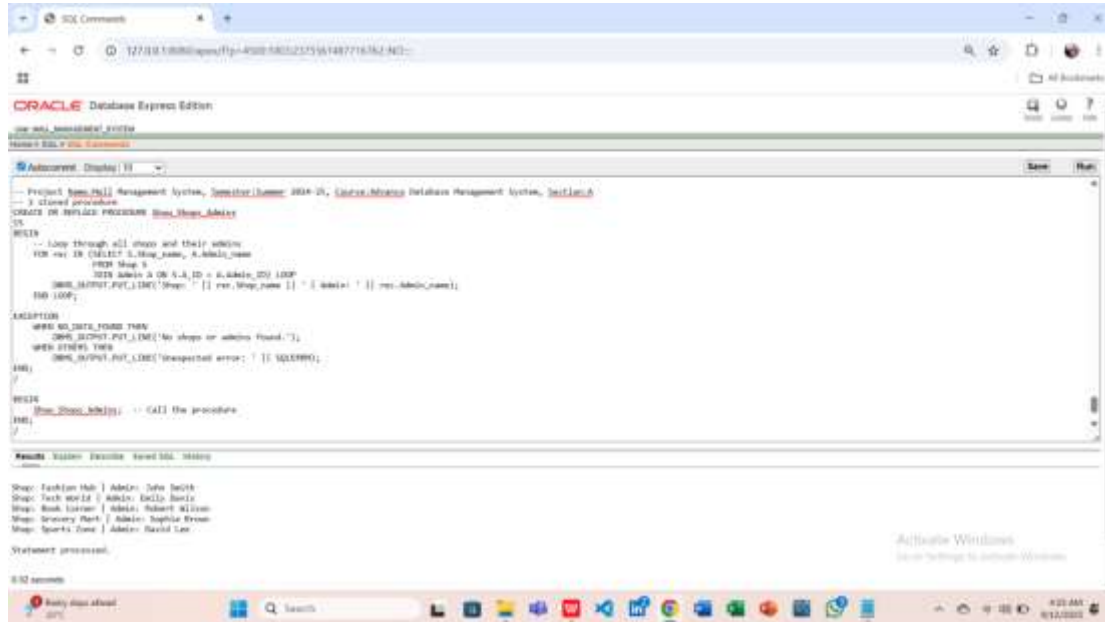
```
END;
```



BEGIN

Show\_Shops\_Admins; -- Call the procedure

END;



**Question-2: Write a stored procedure to display Employee and total Guard salary under them.**

**Answer:**

CREATE OR REPLACE PROCEDURE Show\_Emp\_Guard\_Salary(emp\_id IN NUMBER)

IS

v\_total\_salary NUMBER(10,2);

BEGIN

-- Call the Total\_Salary function

v\_total\_salary := Total\_Salary(emp\_id);

DBMS\_OUTPUT.PUT\_LINE('Employee ID: ' || emp\_id ||

' | Total Salary (including guards): ' || v\_total\_salary);

EXCEPTION

WHEN OTHERS THEN

```

DBMS_OUTPUT.PUT_LINE('Error calculating total salary for Employee ID ' || emp_id ||

        ' ' || SQLERRM);

END;

BEGIN

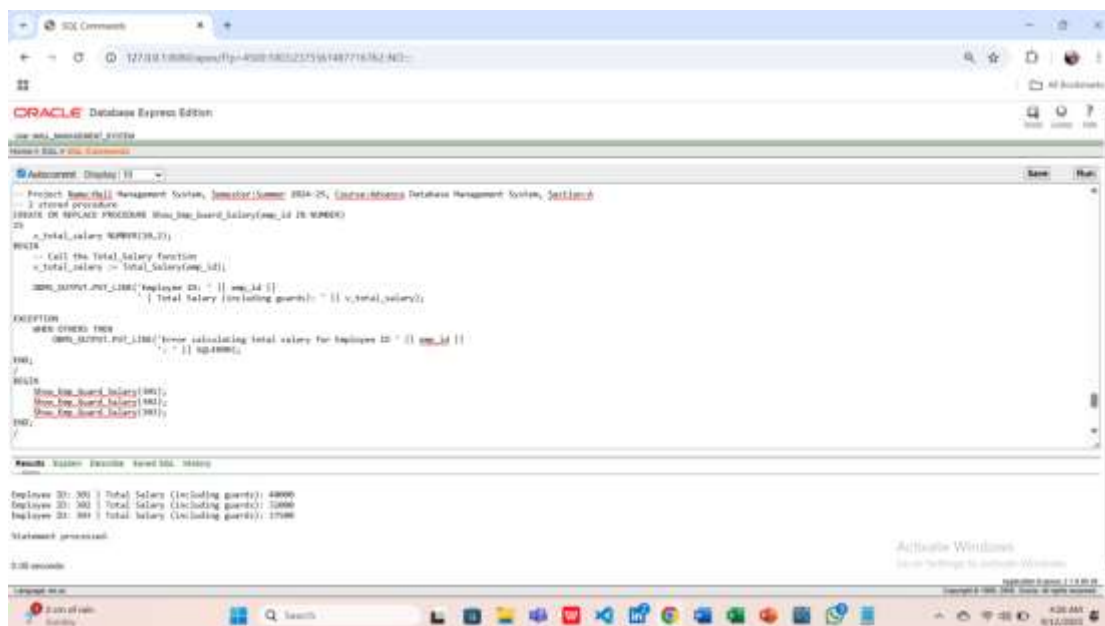
    Show_Emp_Guard_Salary(301);

    Show_Emp_Guard_Salary(302);

    Show_Emp_Guard_Salary(303);

END;

```



## -2 table-based record

**Question-1: Display all Employees using a table-based record type.**

**Answer:**

```

DECLARE

    TYPE emp_table_type IS TABLE OF Employee%ROWTYPE;

    emp_table emp_table_type;

BEGIN

    -- Fetch all employees into a collection

    SELECT * BULK COLLECT INTO emp_table FROM Employee;

```

-- Loop through the collection

FOR i IN 1..emp\_table.COUNT LOOP

```
    DBMS_OUTPUT.PUT_LINE('Employee Name: ' || emp_table(i).Employee_name ||  
        ' | Salary: ' || emp_table(i).Emp_salary);
```

END LOOP;

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

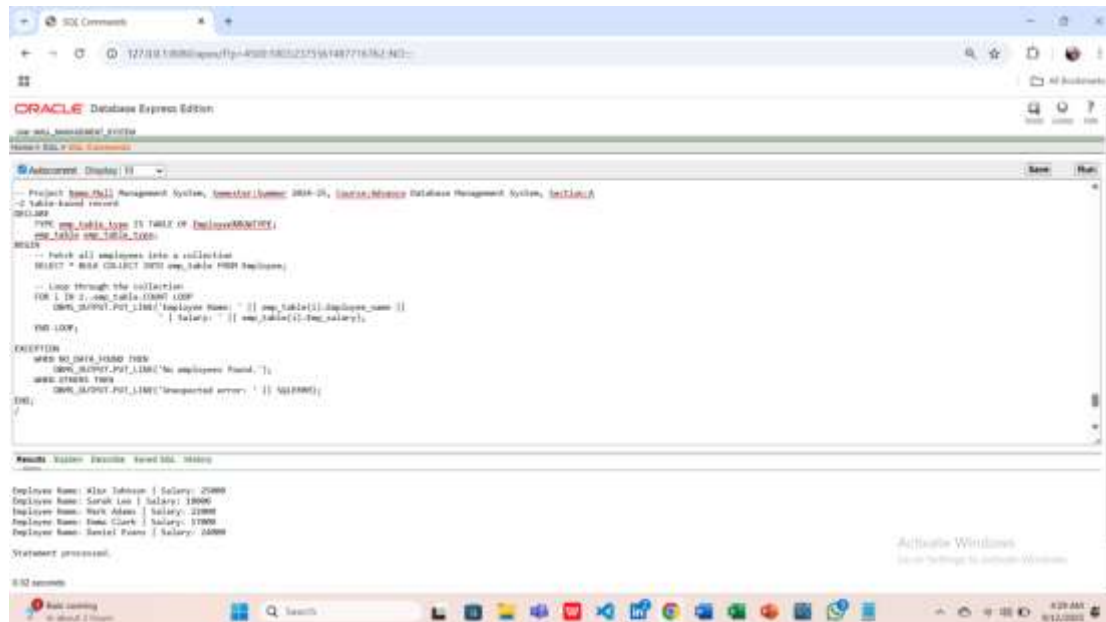
```
    DBMS_OUTPUT.PUT_LINE('No employees found.');
```

WHEN OTHERS THEN

```
    DBMS_OUTPUT.PUT_LINE('Unexpected error: ' || SQLERRM);
```

END;

/



**Question 2: Display all Tenants using a table-based record type.**

**Answer:**

DECLARE

```
    TYPE tenant_table_type IS TABLE OF Tenant%ROWTYPE;
```

```
tenant_table tenant_table_type;
```

```
BEGIN
```

```
-- Fetch all tenants into a collection
```

```
SELECT * BULK COLLECT INTO tenant_table FROM Tenant;
```

```
-- Loop through the collection
```

```
FOR i IN 1..tenant_table.COUNT LOOP
```

```
    DBMS_OUTPUT.PUT_LINE('Tenant Name: ' || tenant_table(i).Tenant_name ||  
        ' | Email: ' || tenant_table(i).Tenant_email);
```

```
END LOOP;
```

```
EXCEPTION
```

```
WHEN NO_DATA_FOUND THEN
```

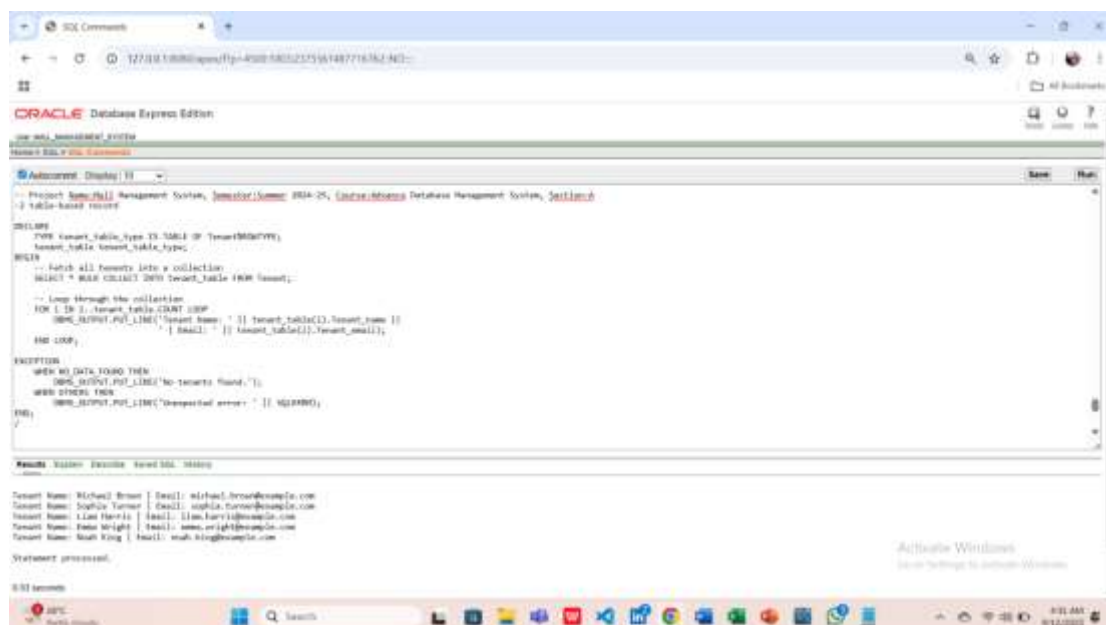
```
    DBMS_OUTPUT.PUT_LINE('No tenants found.');
```

```
WHEN OTHERS THEN
```

```
    DBMS_OUTPUT.PUT_LINE('Unexpected error: ' || SQLERRM);
```

```
END;
```

```
/
```



The screenshot shows the Oracle Database Express Edition SQL Command window. The SQL script is pasted into the command area and has been executed. The results are displayed in a table below the command area.

```
SQL Command
127.0.0.1:8080/express/zip-4000-58052375961487716762-NC...
ORACLE Database Express Edition
SQL Command
-- Fetch all tenants into a collection
SELECT * BULK COLLECT INTO tenant_table FROM Tenant;
-- Loop through the collection
FOR i IN 1..tenant_table.COUNT LOOP
    DBMS_OUTPUT.PUT_LINE('Tenant Name: ' || tenant_table(i).Tenant_name ||
        ' | Email: ' || tenant_table(i).Tenant_email);
END LOOP;
EXCEPTION
WHEN NO_DATA_FOUND THEN
    DBMS_OUTPUT.PUT_LINE('No tenants found.');
```

Tenant Name	Email
Michael Brown	michael.brown@example.com
Sophia Taylor	sophia.taylor@example.com
Liam Davis	liam.davis@example.com
Elena Wright	elena.wright@example.com
Noah King	noah.king@example.com

Statement processed.  
0:01 seconds

## -2 explicit cursor

### **Question 1: Display Shop names and types using an explicit cursor.**

#### **Answer:**

```
DECLARE

    CURSOR shop_cur IS SELECT Shop_name, Shop_type FROM Shop;

    v_name Shop.Shop_name%TYPE;

    v_type Shop.Shop_type%TYPE;

BEGIN

    OPEN shop_cur;

    LOOP

        FETCH shop_cur INTO v_name, v_type;

        EXIT WHEN shop_cur%NOTFOUND;

        DBMS_OUTPUT.PUT_LINE('Shop: ' || v_name || ' | Type: ' || v_type);

    END LOOP;

    CLOSE shop_cur;

EXCEPTION

    WHEN NO_DATA_FOUND THEN

        DBMS_OUTPUT.PUT_LINE('No shops found.');
```

```
    WHEN OTHERS THEN

        DBMS_OUTPUT.PUT_LINE('Unexpected error: ' || SQLERRM);

        -- Ensure cursor is closed if an exception occurs

        IF shop_cur%ISOPEN THEN

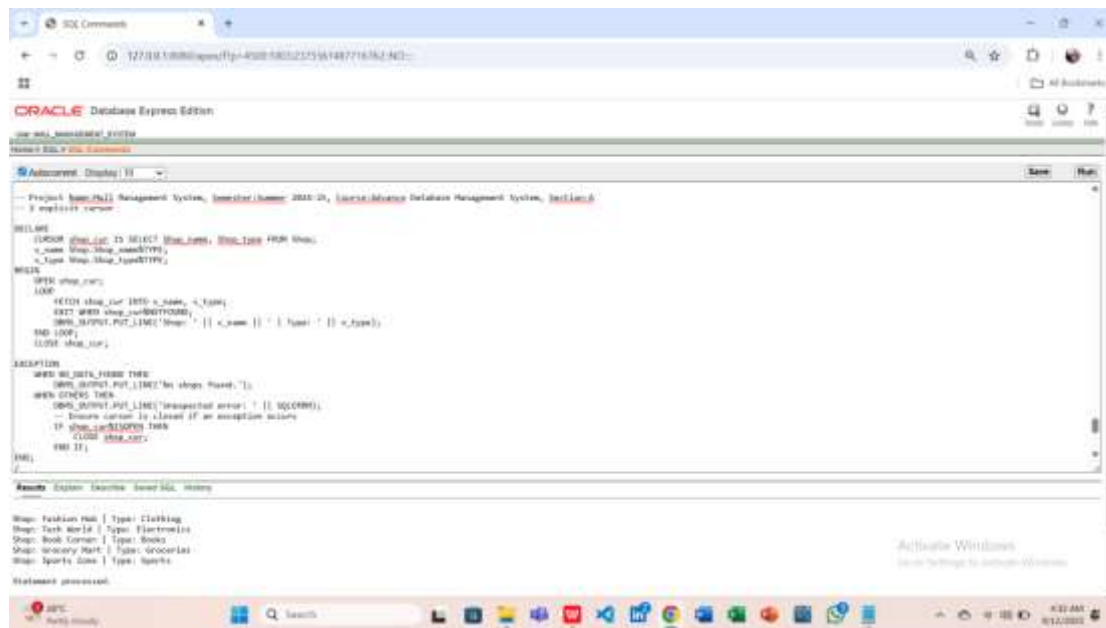
            CLOSE shop_cur;

        END IF;

END;
```



/



**Question -2: Display Customer names and their Shop names using an explicit cursor.**

**Answer:**

DECLARE

CURSOR cust\_cur IS

SELECT C.Customer\_name, S.Shop\_name

FROM Customer C

JOIN Shop S ON C.S\_ID = S.Shop\_ID;

v\_cust Customer.Customer\_name%TYPE;

v\_shop Shop.Shop\_name%TYPE;

BEGIN

OPEN cust\_cur;

LOOP

FETCH cust\_cur INTO v\_cust, v\_shop;

EXIT WHEN cust\_cur%NOTFOUND;

DBMS\_OUTPUT.PUT\_LINE('Customer: ' || v\_cust || ' | Shop: ' || v\_shop);

END LOOP;

CLOSE cust\_cur;

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

DBMS\_OUTPUT.PUT\_LINE('No customers or shops found.');

WHEN OTHERS THEN

DBMS\_OUTPUT.PUT\_LINE('Unexpected error: ' || SQLERRM);

-- Ensure cursor is closed if an exception occurs

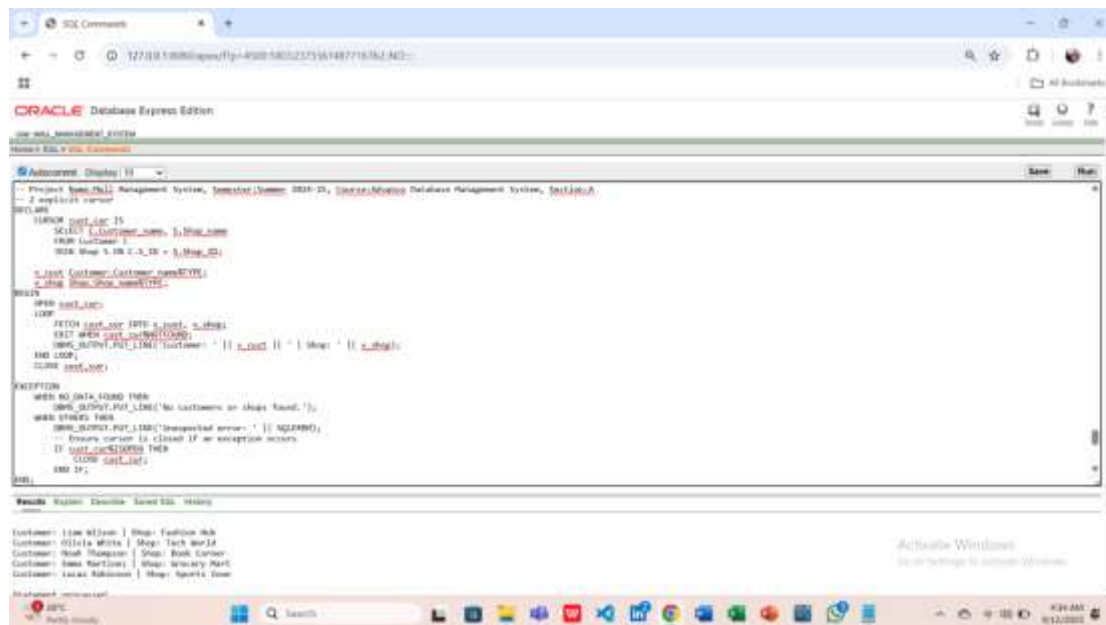
IF cust\_cur%ISOPEN THEN

CLOSE cust\_cur;

END IF;

END;

/



## -2 cursor-based record

**Question 1:Display Employee and salary using a cursor-based record.**

**Answer:**

```

DECLARE

CURSOR emp_cur IS

    SELECT Employee_name, Emp_salary FROM Employee;

    emp_rec emp_cur%ROWTYPE;

BEGIN

    OPEN emp_cur;

    LOOP

        FETCH emp_cur INTO emp_rec;

        EXIT WHEN emp_cur%NOTFOUND;

        DBMS_OUTPUT.PUT_LINE('Employee: ' || emp_rec.Employee_name || ' | Salary: ' ||
emp_rec.Emp_salary);

    END LOOP;

    CLOSE emp_cur;

EXCEPTION

    WHEN NO_DATA_FOUND THEN

        DBMS_OUTPUT.PUT_LINE('No employees found.');
```

```

    WHEN OTHERS THEN

        DBMS_OUTPUT.PUT_LINE('Unexpected error: ' || SQLERRM);

        -- Ensure cursor is closed if an exception occurs

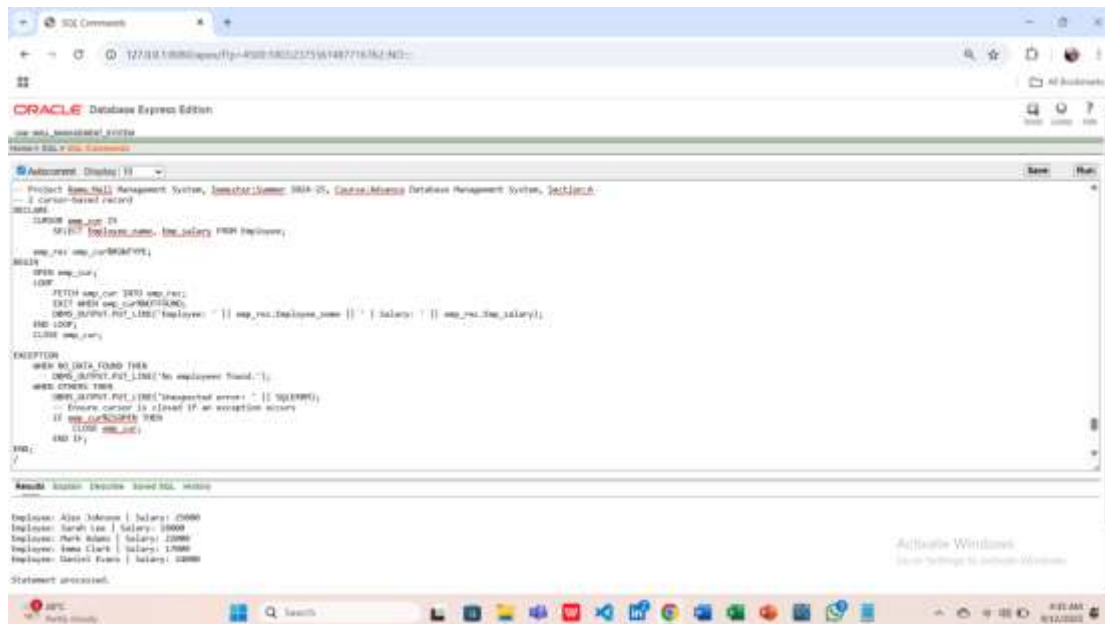
        IF emp_cur%ISOPEN THEN

            CLOSE emp_cur;

        END IF;

END;

/
```



**Question-2: Display Guards and their shift times using a cursor-based record.**

**Answer:**

DECLARE

CURSOR guard\_cur IS

SELECT Guard\_name, Guard\_shift\_time FROM Guard;

guard\_rec guard\_cur%ROWTYPE;

BEGIN

OPEN guard\_cur;

LOOP

FETCH guard\_cur INTO guard\_rec;

EXIT WHEN guard\_cur%NOTFOUND;

DBMS\_OUTPUT.PUT\_LINE('Guard: ' || guard\_rec.Guard\_name || ' | Shift: ' || guard\_rec.Guard\_shift\_time);

END LOOP;

CLOSE guard\_cur;

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

DBMS\_OUTPUT.PUT\_LINE('No guards found.');

WHEN OTHERS THEN

DBMS\_OUTPUT.PUT\_LINE('Unexpected error: ' || SQLERRM);

-- Ensure cursor is closed if an exception occurs

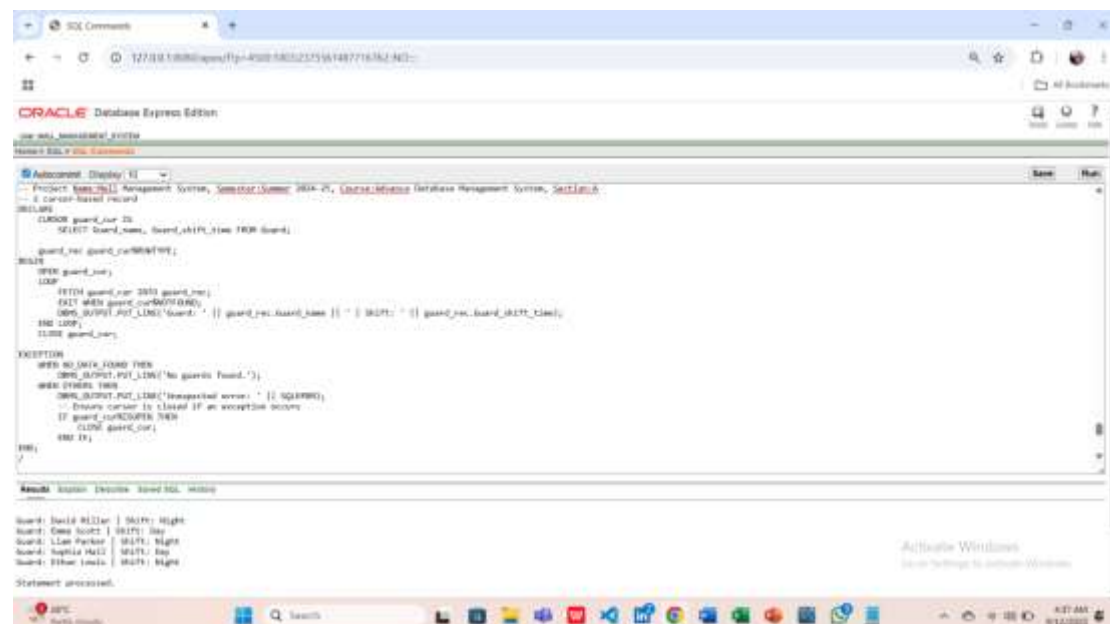
IF guard\_cur%ISOPEN THEN

CLOSE guard\_cur;

END IF;

END;

/



## -2 row level trigger

**Question-1: Create a row-level trigger to log any insert in Payment table.**

**Answer:**

CREATE OR REPLACE TRIGGER log\_payment\_insert

AFTER INSERT ON Payment

FOR EACH ROW

BEGIN

BEGIN

```
DBMS_OUTPUT.PUT_LINE('New Payment Inserted: Payment_ID = ' || :NEW.Payment_ID ||  
    ', Amount = ' || :NEW.Amount);
```

EXCEPTION

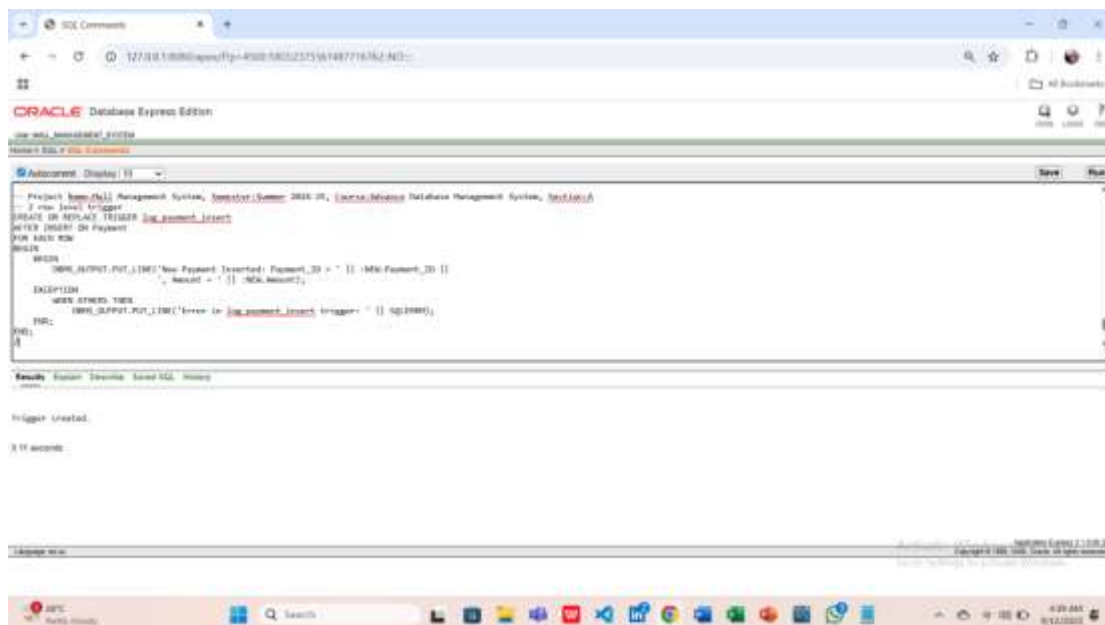
WHEN OTHERS THEN

```
DBMS_OUTPUT.PUT_LINE('Error in log_payment_insert trigger: ' || SQLERRM);
```

END;

END;

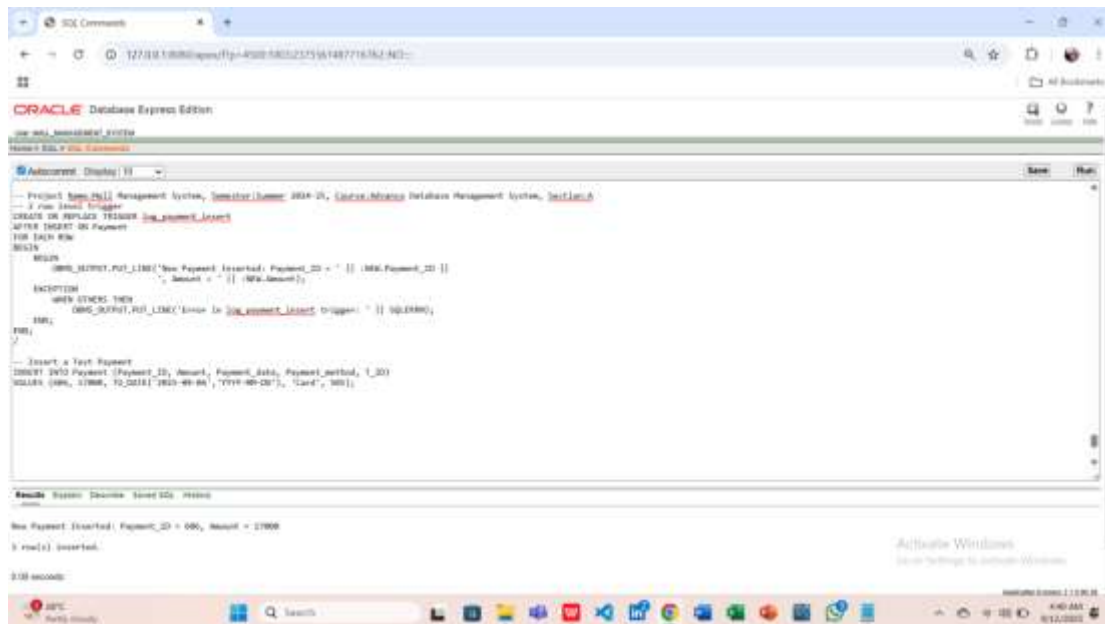
/



-- Insert a Test Payment

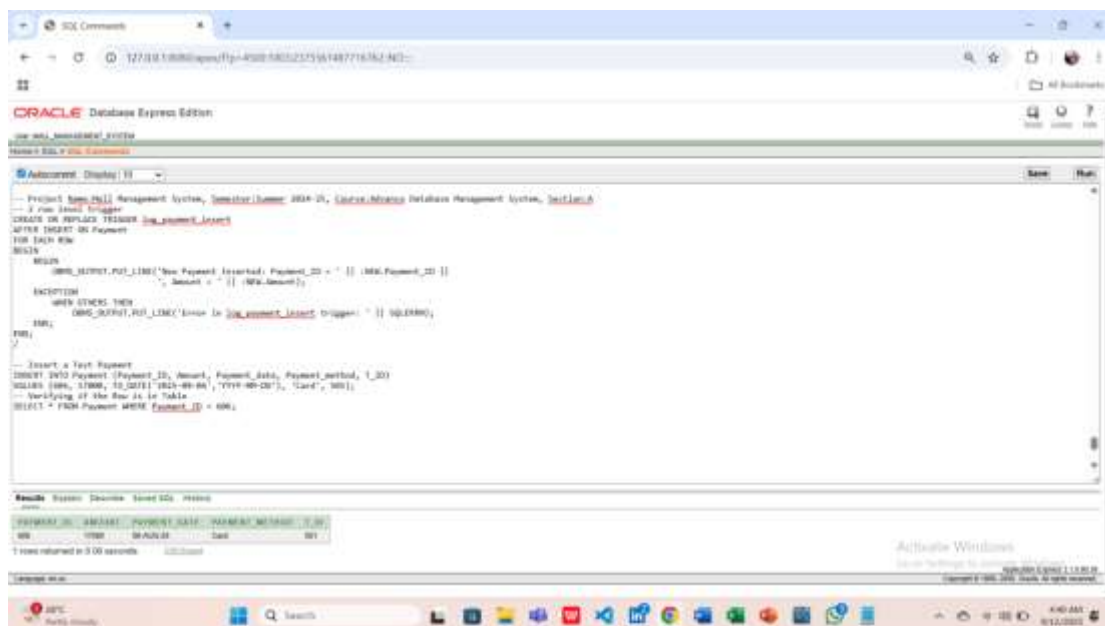
```
INSERT INTO Payment (Payment_ID, Amount, Payment_date, Payment_method, T_ID)
```

```
VALUES (606, 17000, TO_DATE('2025-08-06','YYYY-MM-DD'), 'Card', 501);
```



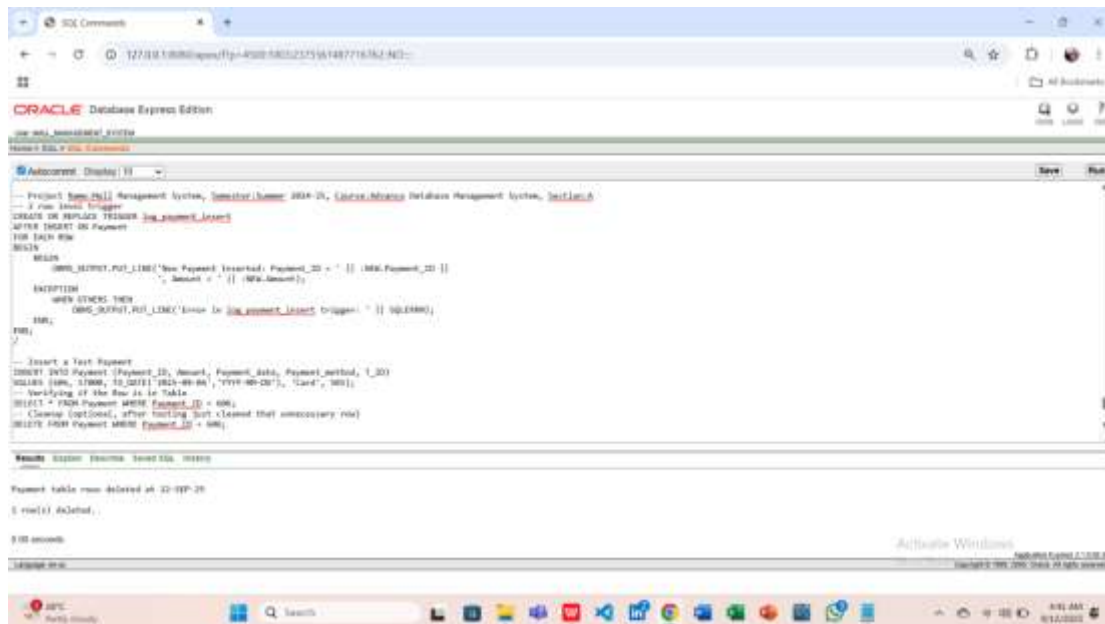
-- Verifying if the Row is in Table

SELECT \* FROM Payment WHERE Payment\_ID = 606;



-- Cleanup (optional, after testing just cleaned that unnecessary row)

DELETE FROM Payment WHERE Payment\_ID = 606;



**Question-2: Create a row-level trigger to log new Customer insertion.**

**Answer:**

CREATE OR REPLACE TRIGGER log\_customer\_insert

AFTER INSERT ON Customer

FOR EACH ROW

BEGIN

BEGIN

DBMS\_OUTPUT.PUT\_LINE('New Customer: ' || :NEW.Customer\_name ||

' | Shop ID: ' || :NEW.S\_ID);

EXCEPTION

WHEN OTHERS THEN

DBMS\_OUTPUT.PUT\_LINE('Error in log\_customer\_insert trigger: ' || SQLERRM);

END;

END;

/

-- Insert a Test Customer

INSERT INTO Customer (Customer\_ID, Customer\_name, Customer\_email, Customer\_phone, Visit\_date, Feedback, S\_ID)



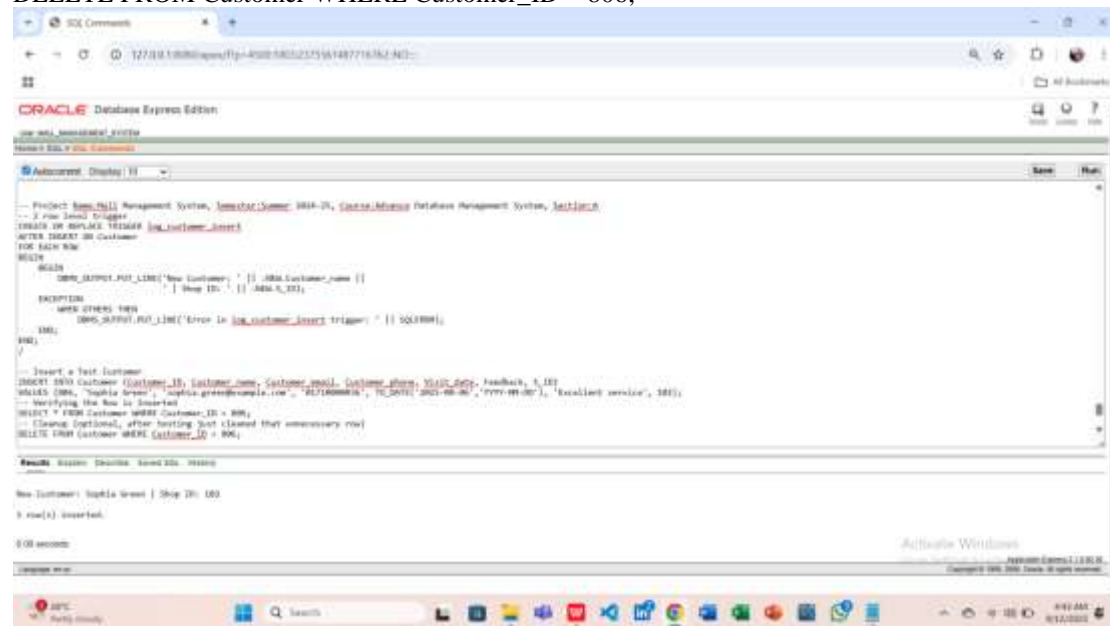
```
VALUES (806, 'Sophia Green', 'sophia.green@example.com', '01710000036', TO_DATE('2025-08-06','YYYY-MM-DD'), 'Excellent service', 102);
```

-- Verifying the Row is Inserted

```
SELECT * FROM Customer WHERE Customer_ID = 806;
```

-- Cleanup (optional, after testing just cleaned that unnecessary row)

```
DELETE FROM Customer WHERE Customer_ID = 806;
```



## -2 statement level trigger

**Question-1: Create a statement-level trigger to log any update in Shop table.**

**Answer:**

```
CREATE OR REPLACE TRIGGER log_shop_update
```

```
AFTER UPDATE ON Shop
```

```
BEGIN
```

```
    BEGIN
```

```
        DBMS_OUTPUT.PUT_LINE('Shop table updated at ' || SYSDATE);
```

```
    EXCEPTION
```

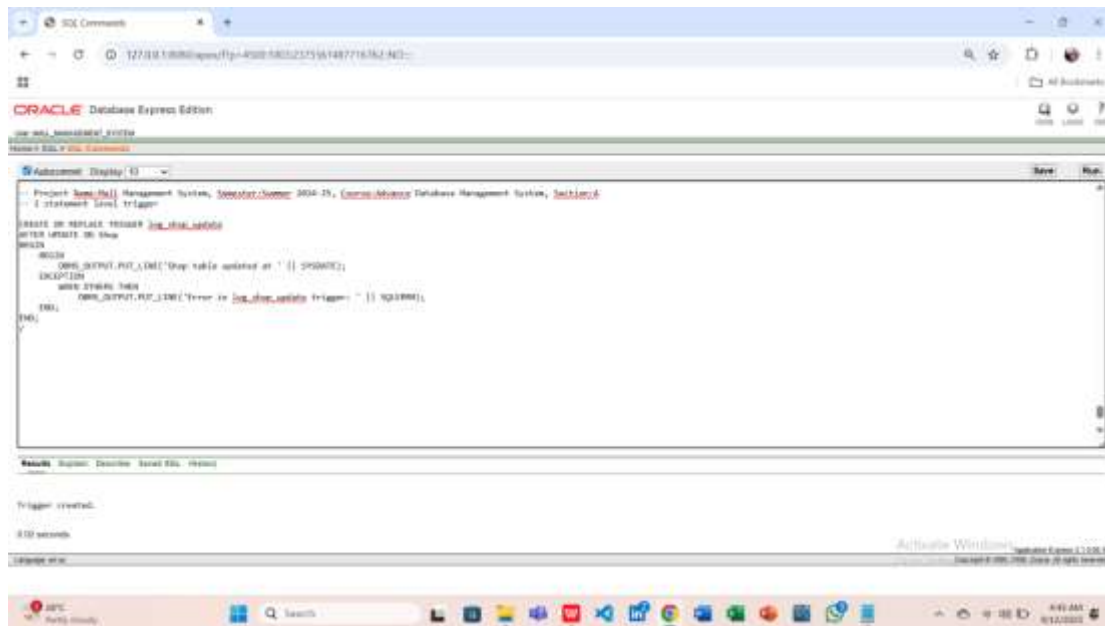
```
        WHEN OTHERS THEN
```

```
            DBMS_OUTPUT.PUT_LINE('Error in log_shop_update trigger: ' || SQLERRM);
```

```
    END;
```

END;

/

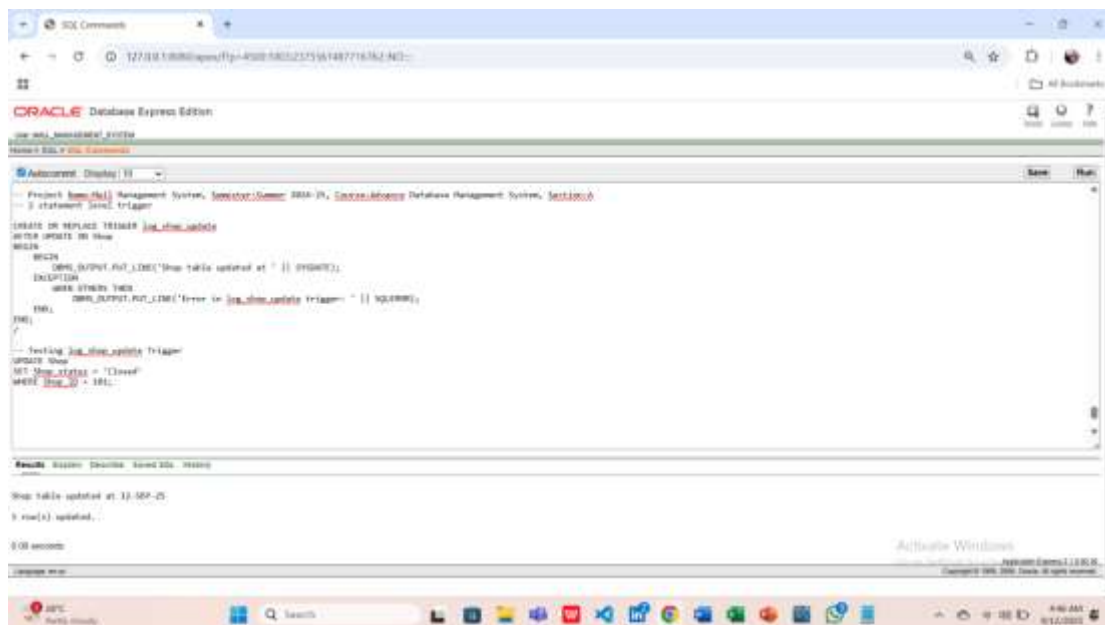


-- Testing log\_shop\_update Trigger

UPDATE Shop

SET Shop\_status = 'Closed'

WHERE Shop\_ID = 101;

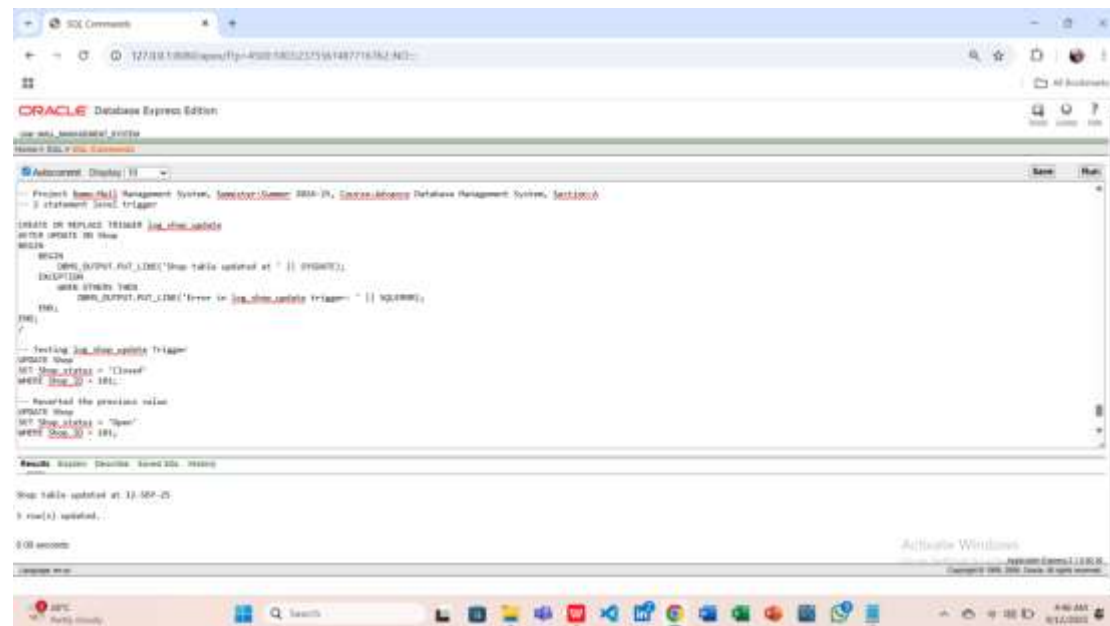


-- Reverted the previous value

UPDATE Shop

SET Shop\_status = 'Open'

WHERE Shop\_ID = 101;



**Question-2: Create a statement-level trigger to log any delete from Payment table.**

**Answer:**

CREATE OR REPLACE TRIGGER log\_payment\_delete

AFTER DELETE ON Payment

BEGIN

BEGIN

DBMS\_OUTPUT.PUT\_LINE('Payment table rows deleted at ' || SYSDATE);

EXCEPTION

WHEN OTHERS THEN

DBMS\_OUTPUT.PUT\_LINE('Error in log\_payment\_delete trigger: ' || SQLERRM);

END;

END;

-- Testing the Trigger, Inserting a sample row

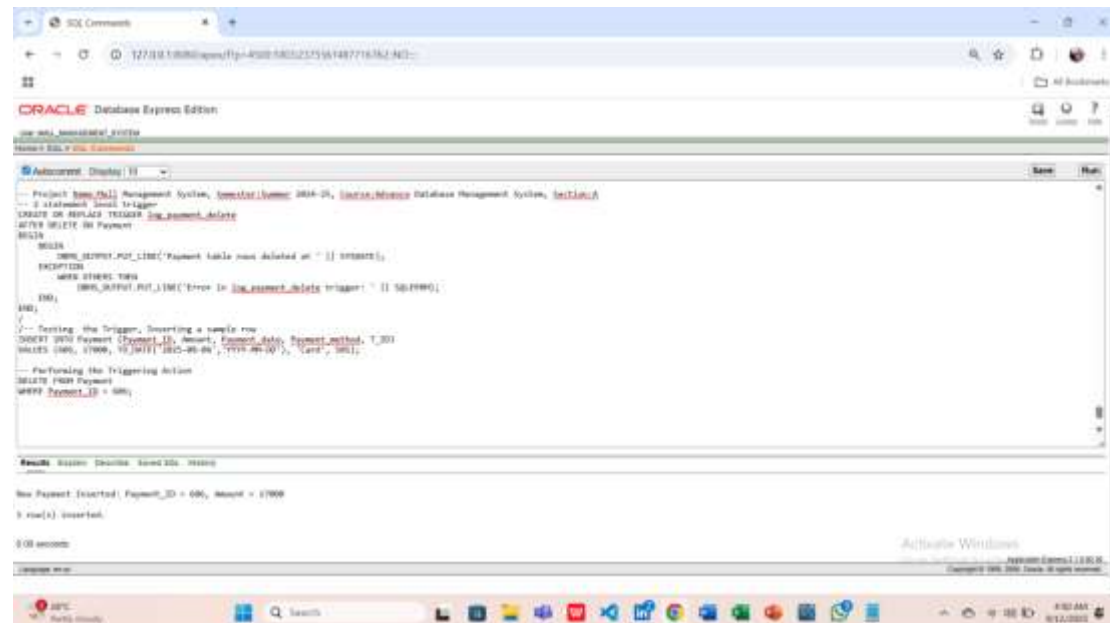
INSERT INTO Payment (Payment\_ID, Amount, Payment\_date, Payment\_method, T\_ID)

VALUES (606, 17000, TO\_DATE('2025-08-06','YYYY-MM-DD'), 'Card', 501);

-- Performing the Triggering Action

DELETE FROM Payment

WHERE Payment\_ID = 606;



## -2 package

**Question-1: Create a package to handle Employee salary queries.**

**Answer:**

-- Package Specification

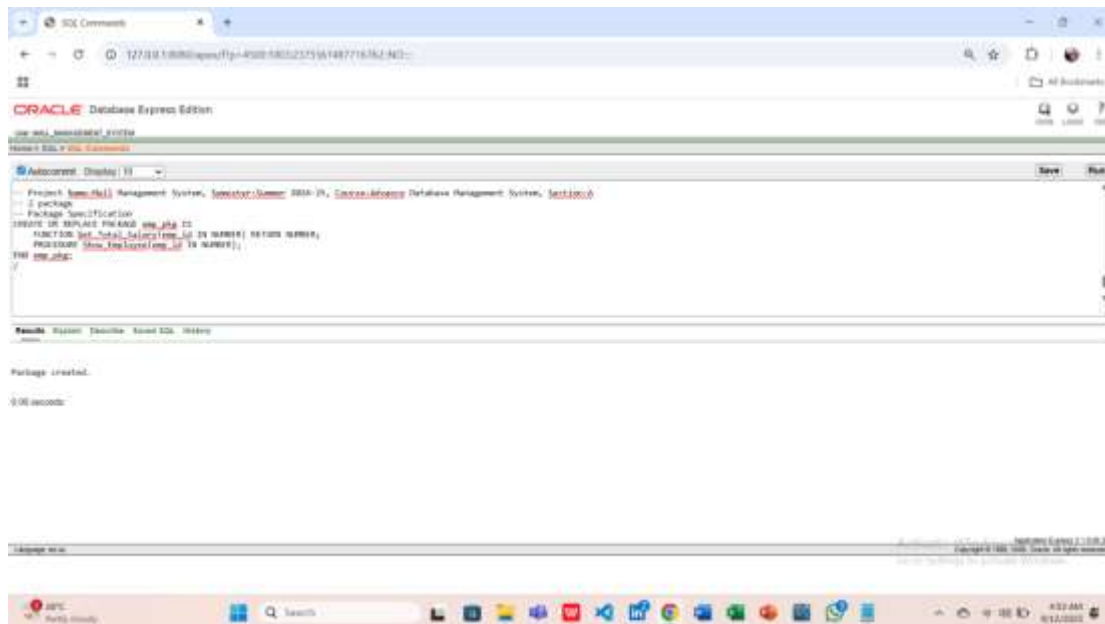
CREATE OR REPLACE PACKAGE emp\_pkg IS

    FUNCTION Get\_Total\_Salary(emp\_id IN NUMBER) RETURN NUMBER;

    PROCEDURE Show\_Employee(emp\_id IN NUMBER);

END emp\_pkg;

/



-- Package Body

CREATE OR REPLACE PACKAGE BODY emp\_pkg IS

-- Function to get total salary

FUNCTION Get\_Total\_Salary(emp\_id IN NUMBER) RETURN NUMBER IS

    v\_salary NUMBER(10,2);

BEGIN

    SELECT Emp\_salary INTO v\_salary

    FROM Employee

    WHERE Employee\_ID = emp\_id;

    RETURN v\_salary;

EXCEPTION

    WHEN NO\_DATA\_FOUND THEN

        DBMS\_OUTPUT.PUT\_LINE('Employee not found for ID = ' || emp\_id);

        RETURN 0;

    WHEN OTHERS THEN

        DBMS\_OUTPUT.PUT\_LINE('Unexpected error in Get\_Total\_Salary: ' || SQLERRM);

        RETURN -1;

```
END Get_Total_Salary;
```

```
-- Procedure to display employee info
```

```
PROCEDURE Show_Employee(emp_id IN NUMBER) IS
```

```
    v_name Employee.Employee_name%TYPE;
```

```
    v_salary NUMBER(10,2);
```

```
BEGIN
```

```
    SELECT Employee_name, Emp_salary INTO v_name, v_salary
```

```
    FROM Employee
```

```
    WHERE Employee_ID = emp_id;
```

```
    DBMS_OUTPUT.PUT_LINE('Employee: ' || v_name || ' | Salary: ' || v_salary);
```

```
EXCEPTION
```

```
    WHEN NO_DATA_FOUND THEN
```

```
        DBMS_OUTPUT.PUT_LINE('Employee not found for ID = ' || emp_id);
```

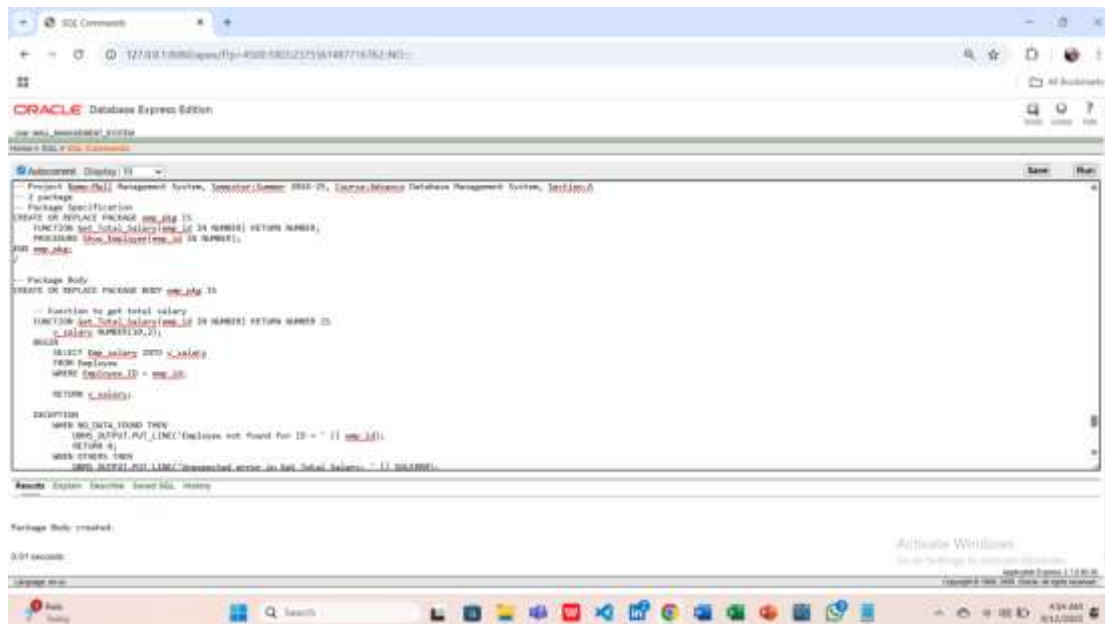
```
    WHEN OTHERS THEN
```

```
        DBMS_OUTPUT.PUT_LINE('Unexpected error in Show_Employee: ' || SQLERRM);
```

```
END Show_Employee;
```

```
END emp_pkg;
```

```
/
```

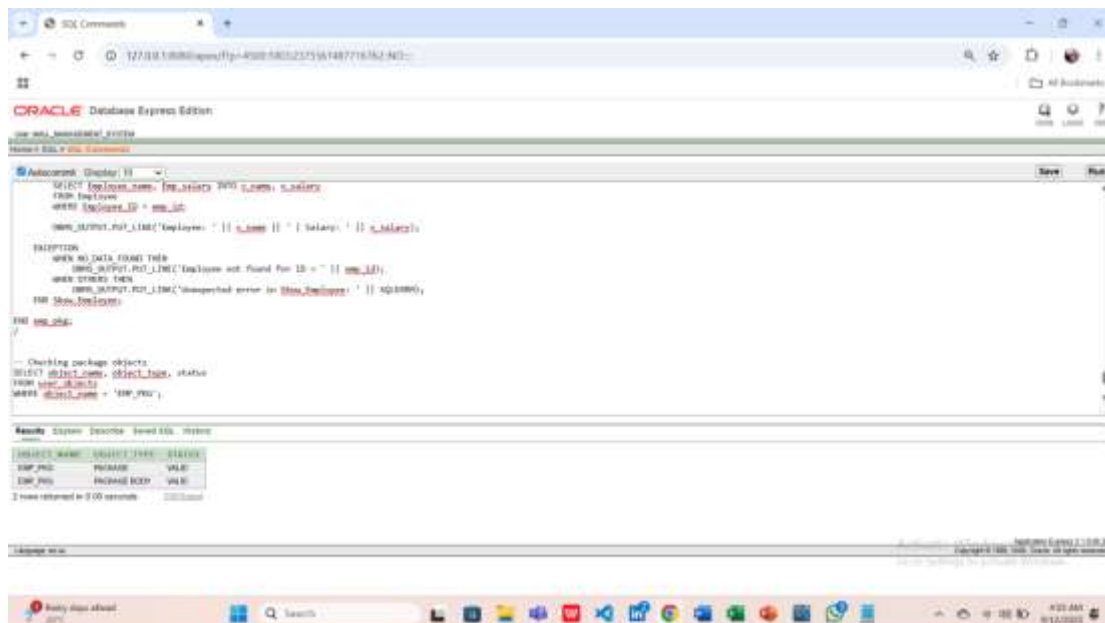


-- Checking package objects

SELECT object\_name, object\_type, status

FROM user\_objects

WHERE object\_name = 'EMP\_PKG';



## Question-2: Create a package to handle Tenant payment queries.

### Answer:

-- Package Specification

CREATE OR REPLACE PACKAGE tenant\_pkg IS

FUNCTION Total\_Payment(t\_id IN NUMBER) RETURN NUMBER;

PROCEDURE Show\_Tenant(t\_id IN NUMBER);

```

END tenant_pkg;

/

-- Package Body

CREATE OR REPLACE PACKAGE BODY tenant_pkg IS

    -- Function to calculate total payment

    FUNCTION Total_Payment(t_id IN NUMBER) RETURN NUMBER IS

        v_total NUMBER(10,2);

    BEGIN

        SELECT NVL(SUM(Amount),0) INTO v_total
        FROM Payment
        WHERE T_ID = t_id;

        RETURN v_total;

    EXCEPTION

        WHEN NO_DATA_FOUND THEN

            DBMS_OUTPUT.PUT_LINE('No payments found for Tenant ID = ' || t_id);

            RETURN 0;

        WHEN OTHERS THEN

            DBMS_OUTPUT.PUT_LINE('Unexpected error in Total_Payment: ' || SQLERRM);

            RETURN -1;

    END Total_Payment;

    -- Procedure to show tenant details

    PROCEDURE Show_Tenant(t_id IN NUMBER) IS

        v_name Tenant.Tenant_name%TYPE;

        v_total NUMBER(10,2);

    BEGIN

        SELECT Tenant_name INTO v_name
        FROM Tenant
        WHERE Tenant_ID = t_id;

        v_total := Total_Payment(t_id);

```



```
DBMS_OUTPUT.PUT_LINE('Tenant: ' || v_name || ' | Total Payment: ' || v_total);
```

EXCEPTION

```
WHEN NO_DATA_FOUND THEN
```

```
    DBMS_OUTPUT.PUT_LINE('Tenant not found for ID = ' || t_id);
```

```
WHEN OTHERS THEN
```

```
    DBMS_OUTPUT.PUT_LINE('Unexpected error in Show_Tenant: ' || SQLERRM);
```

```
END Show_Tenant;
```

```
END tenant_pkg;
```

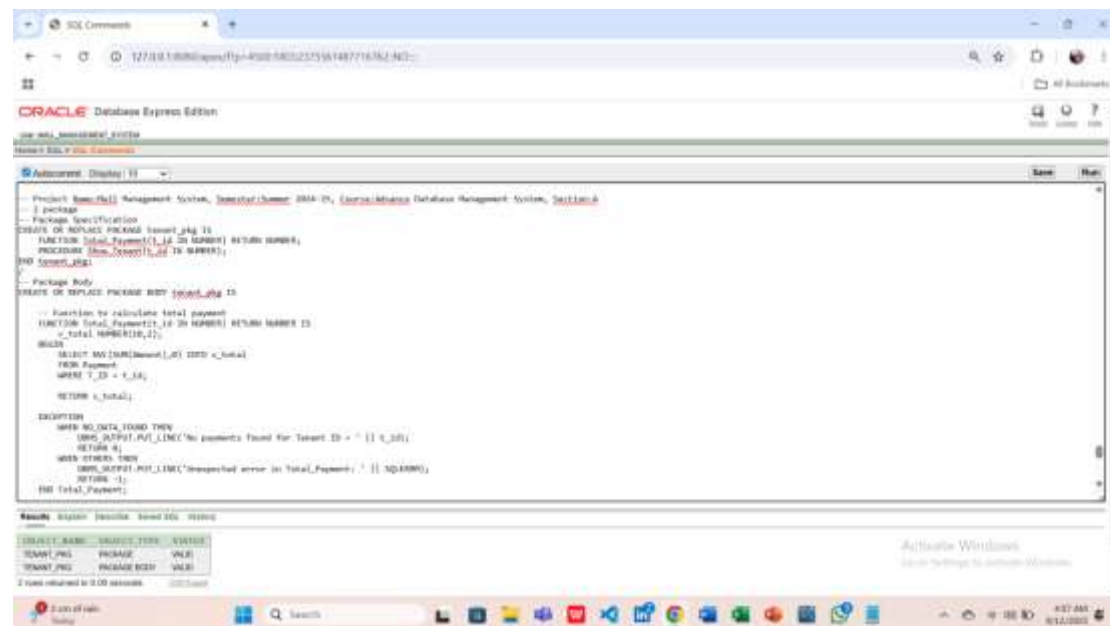
```
/
```

```
-- Checking package objects
```

```
SELECT object_name, object_type, status
```

```
FROM user_objects
```

```
WHERE object_name = 'TENANT_PKG';
```



## 13. Relational Algebra

**Question 1: List the names and salaries of all Employees.**

**Answer :**  $\pi_{\text{Employee\_name}, \text{Emp\_salary}}(\text{Employee})$

**Question 2: Find the Guard working under Employee\_ID = 301.**

**Answer :**  $\pi_{\text{Guard\_name}}(\sigma_{\text{E\_ID}=301}(\text{Guard}))$

**Question 3: List all Shops and the name of the Admin managing them.**

**Answer :**  $\pi_{\text{Shop\_name}, \text{Admin\_name}}(\text{Shop} \bowtie_{\text{Shop.A\_ID}=\text{Admin.Admin\_ID}} \text{Admin})$

**Question 4: Find the total payment made by Tenant\_ID = 501.**

**Answer :**  $\gamma_{\text{SUM}(\text{Amount}) \rightarrow \text{Total\_Payment}}(\sigma_{\text{T\_ID}=501}(\text{Payment}))$

**Question 5: List all Tenants who have made payments greater than 15000.**

**Answer :**

$\pi_{\text{Tenant\_name}}(\sigma_{\text{Amount}>15000}(\text{Payment} \bowtie_{\text{Payment.T\_ID}=\text{Tenant.Tenant\_ID}} \text{Tenant}))$

## **14. User Interface Design To Code Implementation (Screenshots)**

## 15. Oracle 10g Database Connection Process

Based on the midterm UI designs, we implemented the corresponding user interfaces in **C# programming language**.

The following procedure was followed to connect Oracle 10g Database using C# programming language:

### 1. Oracle 10g Installation and Configuration

First, we ensured that Oracle 10g Database was properly installed and running (It was already ensured in mid-term, as we created database & also table creation, data insertion were done). We checked the listener service and noted the host name, port number (default 1521), and database SID which would be required for the connection.

### 2. Preparing the Development Environment

Next, we prepared the development environment in Visual Studio. A new C# Console Application project was created where we would implement the connection code.

### 3. Adding Oracle Data Access Library

After creating the project, we added the required Oracle Data Access library. For this, the **Oracle.ManagedDataAccess.dll** file was included in the project references. This library is necessary for establishing communication between the C# application and the Oracle database.

### 4. Constructing the Connection String

Then we constructed the database connection string. In the connection string, we specified the database username, password, and the data source, which contained the host, port, and Oracle SID.

Example Format:

User Id=**MALL\_MANGEMENT\_SYSTEM**; Password=**mall**; Data Source=localhost:1521/orcl;

### 5. Writing the Connection Code

After preparing the connection string, we wrote C# code to connect with the Oracle 10g database. Inside the code, we created an OracleConnection object using the connection string and opened the connection inside a try-catch block to handle any possible errors.

## 6. Executing a Sample SQL Query

Once the connection was successful, we executed a sample SQL query. For verification, a simple query was executed on the Oracle system table dual to fetch the current system date. The returned result confirmed that the connection was working correctly.

## 7. Testing the Process

Finally, we tested the entire process. The project was built and executed. On successful execution, a confirmation message displayed that the application was connected to the Oracle 10g database along with the current database date.

## 8. Troubleshooting and Error Handling

In case of errors, we reviewed common issues. For example, if the Oracle listener was not running, we started it using the Oracle tools. If the credentials were incorrect, we corrected them in the connection string. If the Oracle assembly was not found, we ensured that the Oracle Data Provider for .NET was properly installed.

This way, we successfully connected a C# application with the Oracle 10g Database and verified the connection by executing a query.

## Issues Faced During Oracle 10g Database Connection

**Listener Not Running** : Sometimes the Oracle listener service was not active, which caused connection failures. We had to manually start it using Oracle tools.

**Invalid Credentials** : Connection attempts failed when incorrect usernames or passwords were entered. Updating the connection string with correct credentials solved this.

**Missing Oracle Assembly** : The project could not run if the Oracle Data Access library (Oracle.ManagedDataAccess.dll or Oracle.DataAccess.dll) was not properly installed or referenced.

**Port or Network Issues**: The default Oracle port (1521) was sometimes blocked by firewall or network restrictions, which prevented the C# application from connecting.

**Configuration Mismatches** : If the SID or service name in the connection string did not match the Oracle database configuration, the connection could not be established.

**Outdated Oracle 10g Version** : Since Oracle 10g is an older version, compatibility issues were faced with newer development tools and drivers. Some modern libraries do not fully support Oracle 10g, which required extra configuration and adjustments.

## 16. Conclusion

The Mall Management System successfully demonstrates how database-driven applications can streamline and organize mall operations. By integrating core functions such as shop and tenant management, customer tracking, inventory monitoring, billing, employee supervision, and facility maintenance, the system reduces manual errors, enhances efficiency, and ensures secure and consistent data handling. The use of proper database design principles, normalization, and PL/SQL queries ensures data integrity and reliability, while the centralized system provides quick access to information, supporting effective decision-making for mall administrators and shopkeepers. Overall, the project highlights the potential of database systems to improve real-world operations by saving time, minimizing errors, and providing structured insights.

### Future Work

Although the system meets its primary objectives, there are several areas where it can be improved and expanded in the future:

1. **Web-Based User Interface** – Develop a web or mobile application connected to the database to allow mall staff, tenants, and customers to interact with the system in real time.
2. **Automation & Notifications** – Add features such as automated reminders for rent payments, stock replenishment alerts, and maintenance scheduling.
3. **Analytics & Reporting** – Integrate advanced reporting dashboards and data visualization tools for sales forecasting, customer trends, and performance analysis.
4. **Security Enhancements** – Implement stronger authentication, role-based access control, and encryption to protect sensitive tenant and customer data.
5. **Integration with IoT Devices** – Connect the system with sensors (e.g., for energy monitoring, security cameras, or foot traffic tracking) to enable smart mall management.
6. **Scalability Improvements** – Optimize the database for handling larger malls with thousands of shops, tenants, and customers without performance issues.
7. **Cloud Deployment** – Move the system to a cloud-based environment for better accessibility, reliability, and scalability.

By addressing these improvements, the Mall Management System can evolve into a fully functional, intelligent, and scalable solution capable of meeting the growing demands of modern shopping complexes.