Course: Advance Database Management System

Sec: B

Project Name: Helping Hand Management System

Group: 08

Group Members

Name	ID	Contribution	Topic
Tahmid Al Rafid Siddique	20-42086-1	25%	Query Writing, ER Diagram, Data Insertion
Istyak Ahmed	21-45658-3	25%	Diagrams, Interface Design, Scenario Description, Database Connection
Mohammed Adnan Babu	21-45454-3	25%	Normalization, Finalization, Conclusion
Sadakatul Ferdous Pranto	21-45623-3	25%	Schema Diagram, Table Creation, Project Proposal

Table of Contents

Project Updates

- Cardinality issue with scenario description has been attempted to solve.
- Ternary from ER Diagram was removed.
- Slight Changes were made to the Normalization and Table Creation.
- Schema Diagram was updated.

Introduction

Our project revolves around building a Helping Hand Management System (HHMS) to streamline the recruitment process for domestic helpers. The HHMS connects homeowners with verified helpers, simplifying tasks such as profile browsing, job matching, and scheduling. It also supports agencies with profile management and contract tracking. Our aim is to provide an intuitive platform that enhances efficiency and trust, improving the experience for both employers and workers.

Project Proposal

We propose the development of a Home Service Management System (HSMS) to address the operational inefficiencies faced by service agencies in managing interactions between homeowners, helpers, and agency staff. The HSMS aims to streamline workflows, enhance communication, and improve overall service quality. By automating manual tasks and centralizing critical data, the system will create a seamless experience for both service providers and recipients.

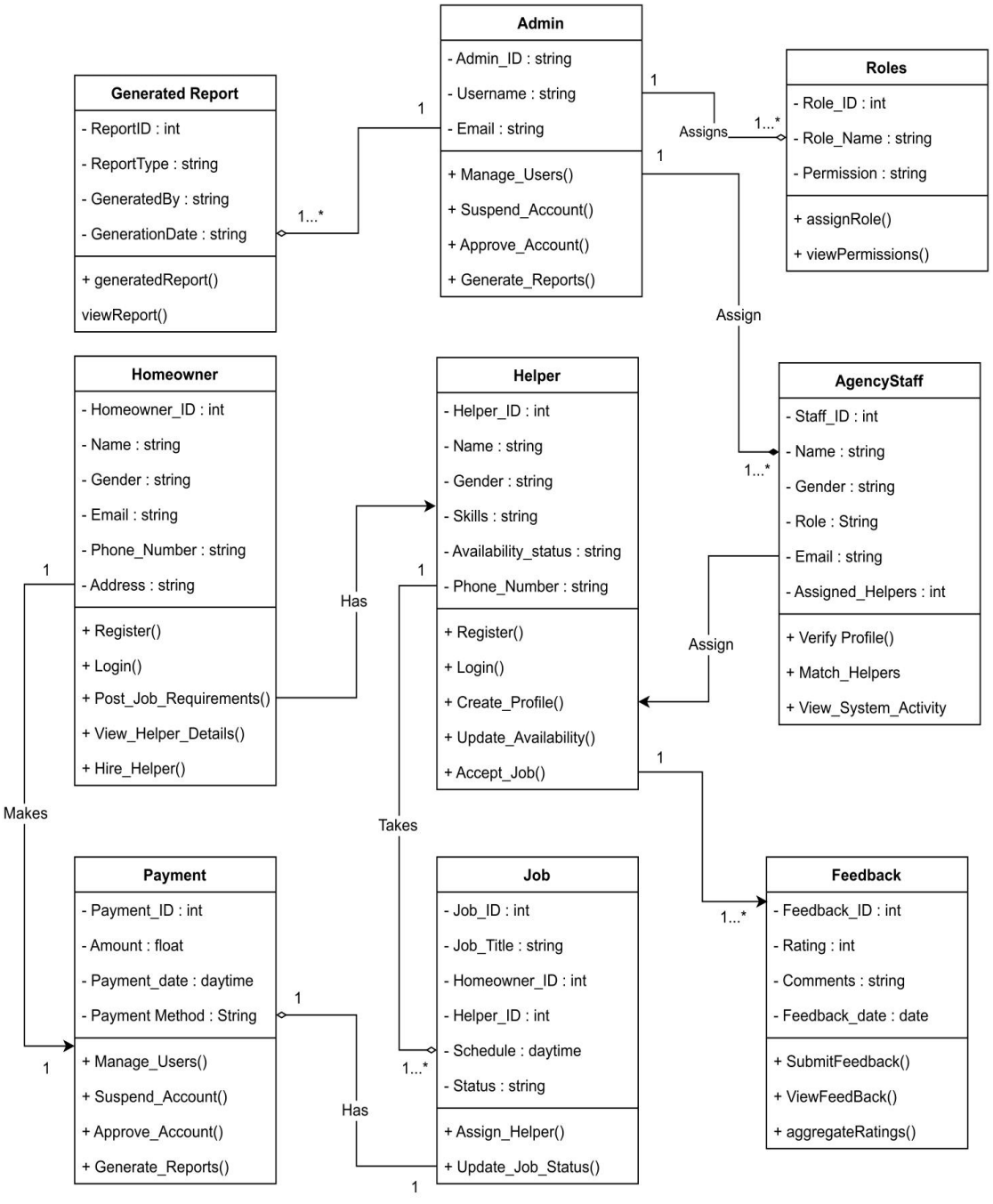
The primary objective of the HSMS is to simplify management tasks and ensure operational transparency while maintaining a user-friendly interface. This system will encompass essential functionalities, such as job management, where homeowners can post and schedule jobs while tracking their status. It will also include tools for user management, ensuring that accurate and secure records are maintained for homeowners, helpers, and administrative staff. Payment management will allow the recording and monitoring of transactions, while feedback and communication modules will facilitate service improvements and timely notifications. Additionally, the system will generate detailed reports to assist agencies in making informed decisions.

To achieve these goals, we will adopt an iterative development approach that emphasizes stakeholder engagement at every stage. We will begin with thorough requirements gathering to align with the specific needs of users. Following this, we will design a robust database structure and implement a scalable, secure system tailored to handle complex workflows efficiently. The implementation phase will focus on developing all modules cohesively, followed by extensive testing to ensure functionality, reliability, and user satisfaction. Finally, we will deploy the system and provide comprehensive training and documentation to ensure smooth adoption by all stakeholders.

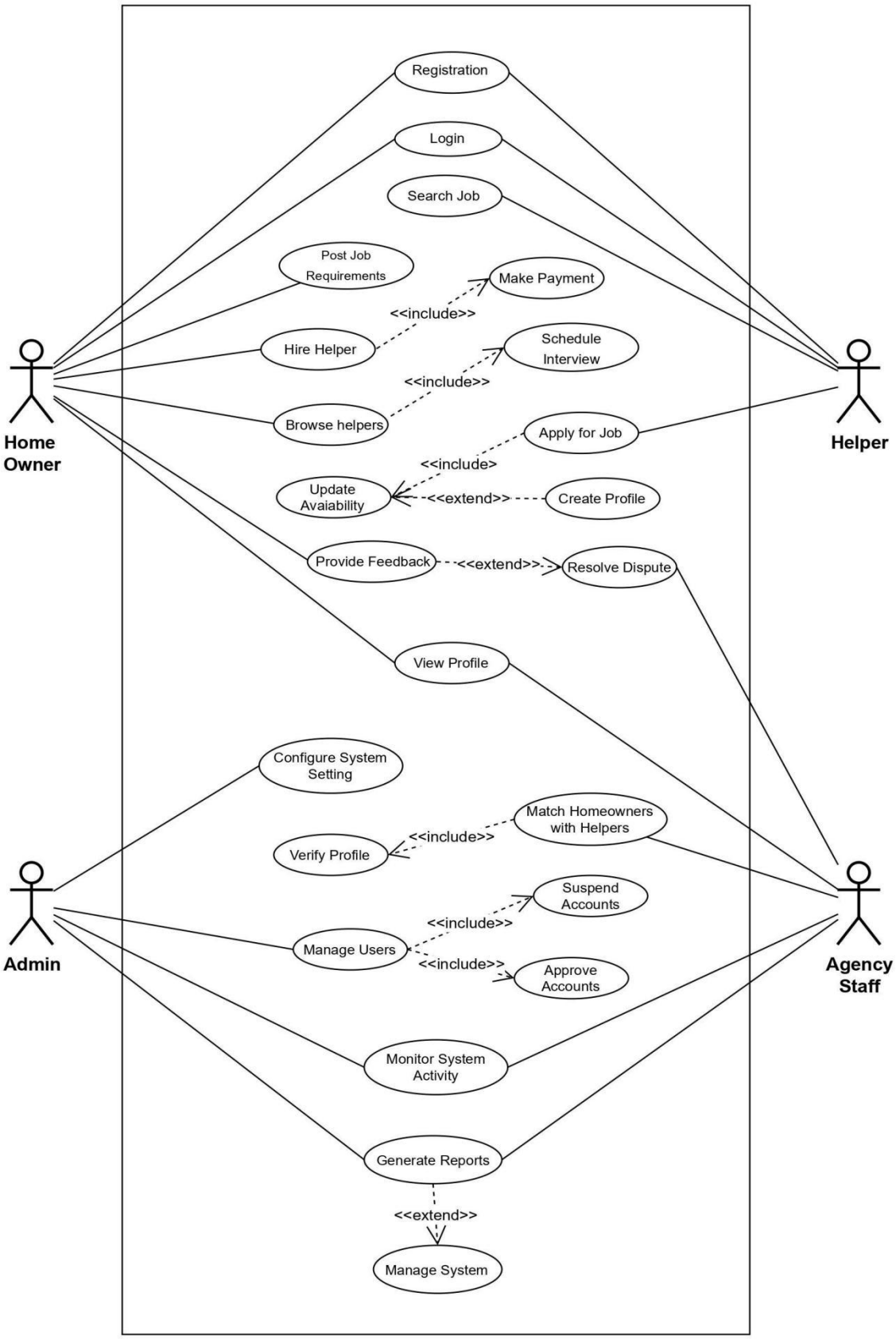
This project will deliver a fully functional Home Service Management System, accompanied by user documentation, a deployment-ready platform, and support for ongoing maintenance. By revolutionizing how service agencies operate, our HSMS will foster automation, improve communication, and significantly enhance the user experience. With a committed team and a clear roadmap, we are confident in delivering a high-quality solution that meets the demands of all involved parties.

Diagrams

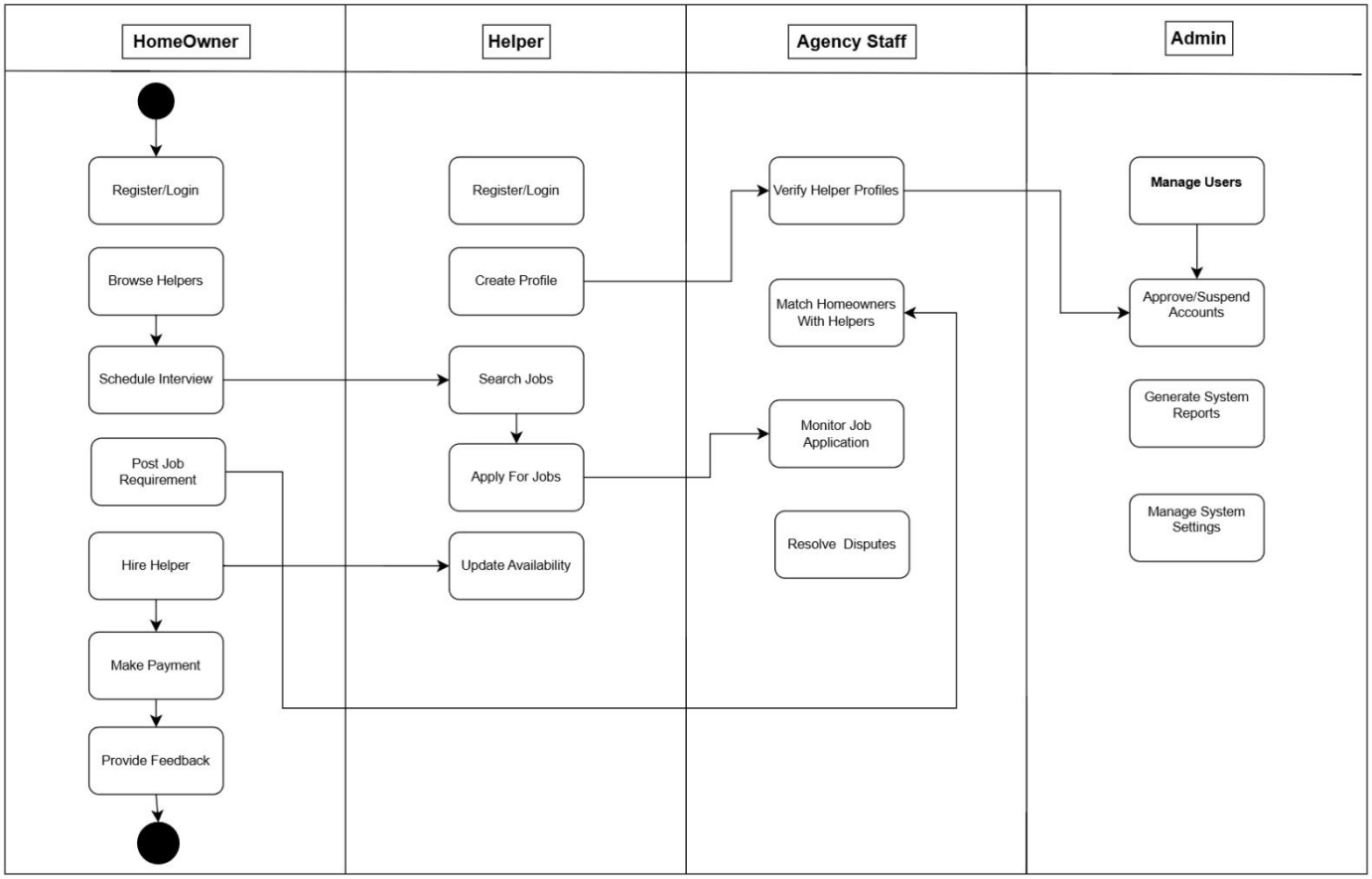
Class Diagram



Use Case Diagram

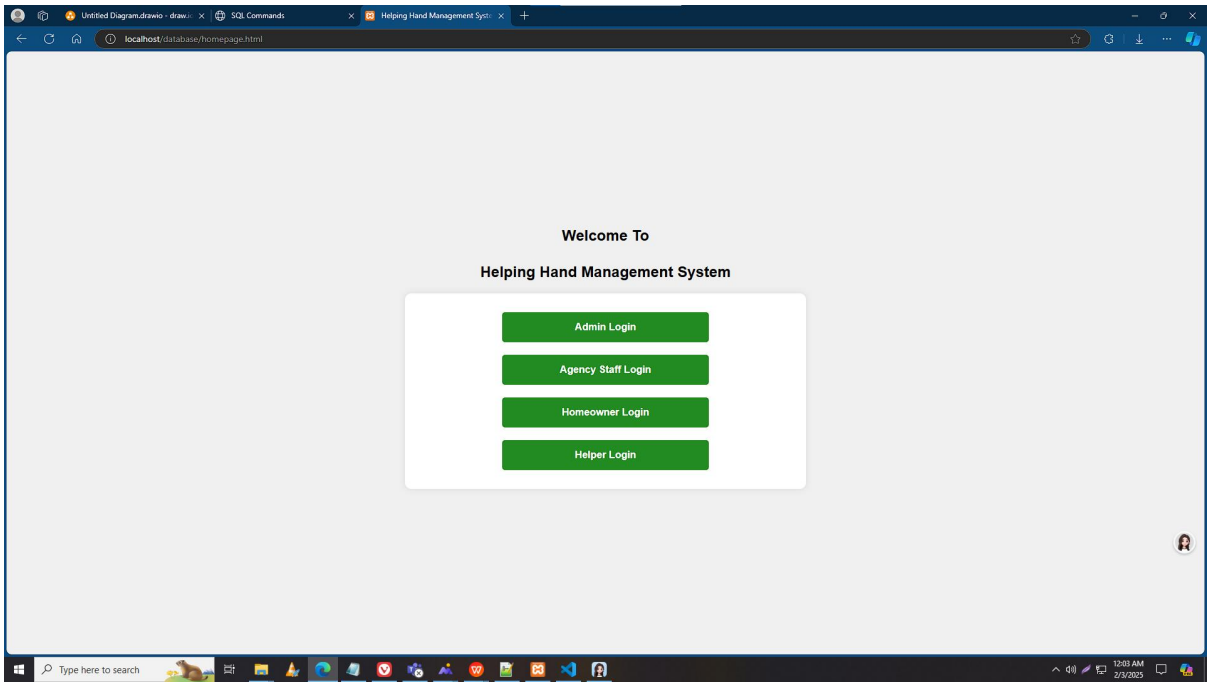


Activity Diagram

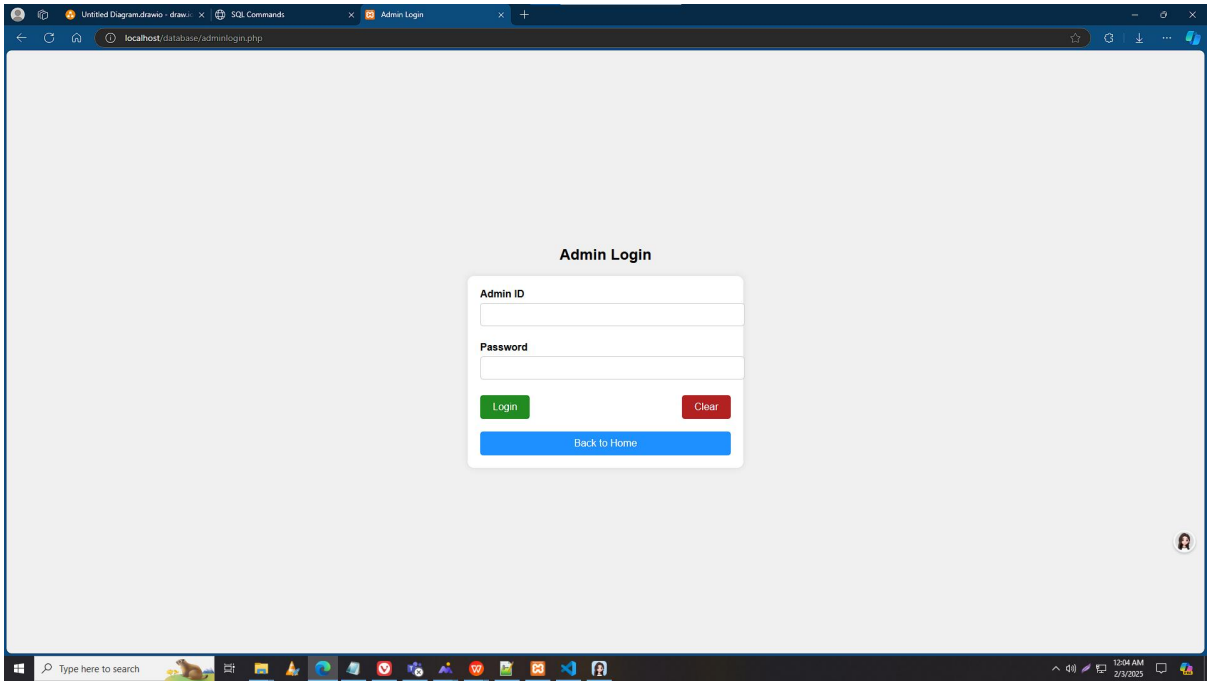


Interface Design

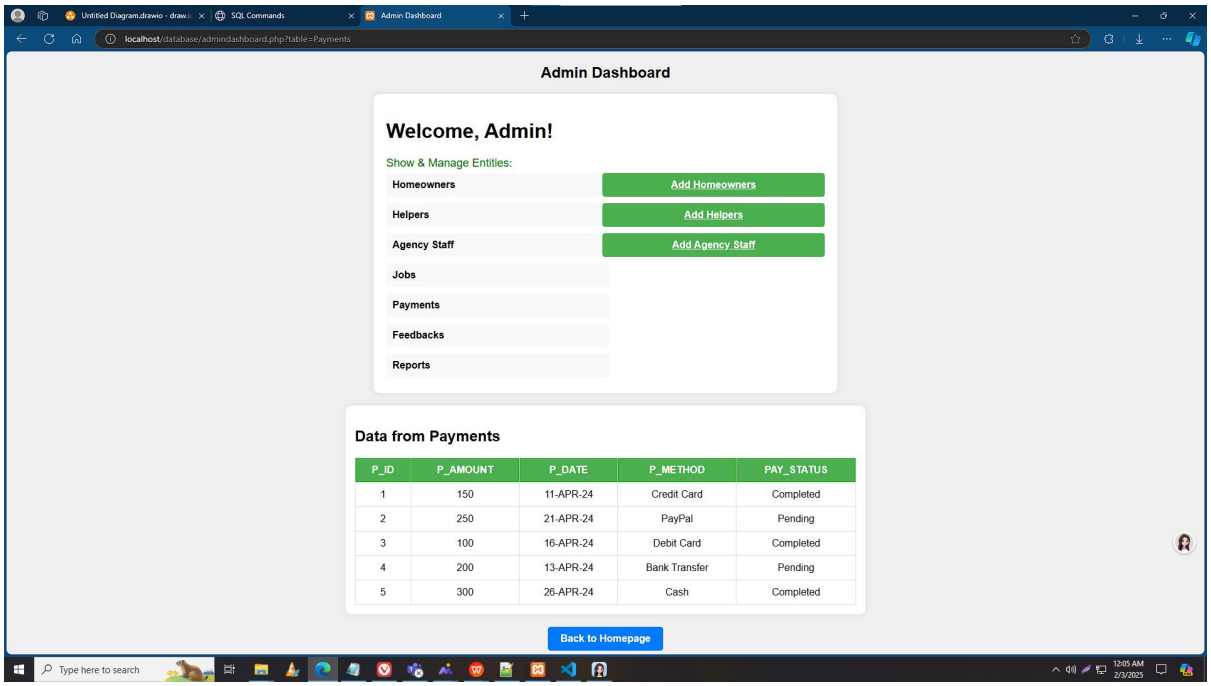
Project Homepage



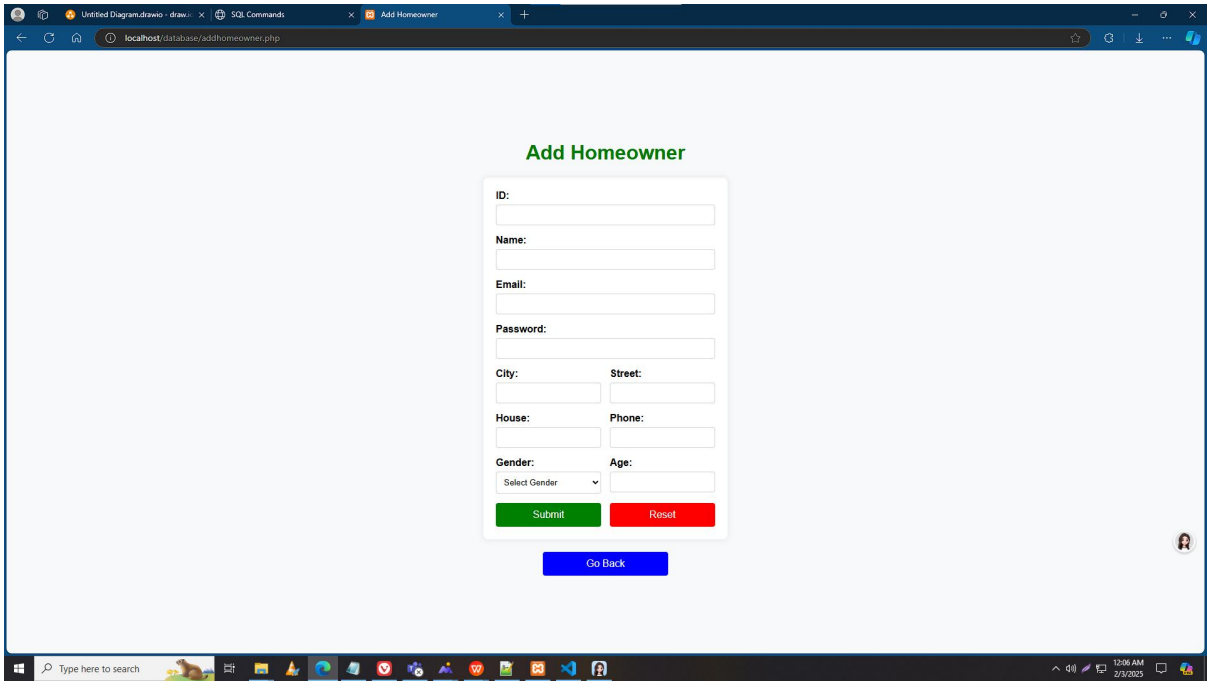
Admin Login Page



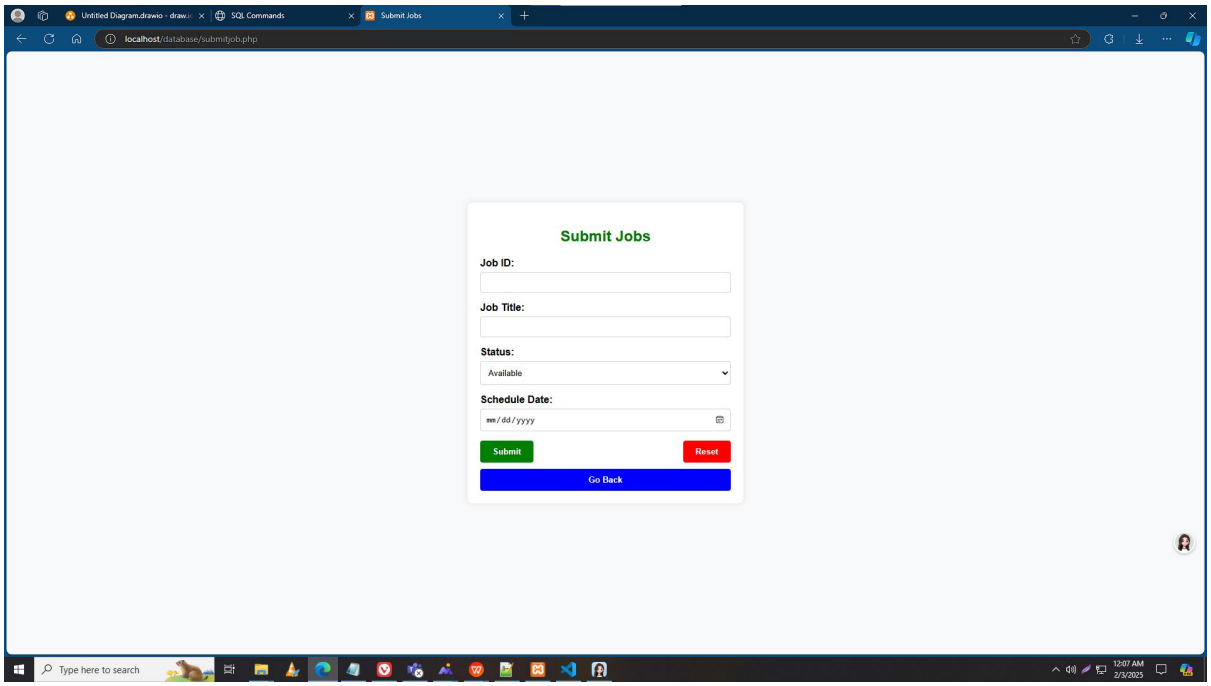
Admin Dashboard & Data Visualization



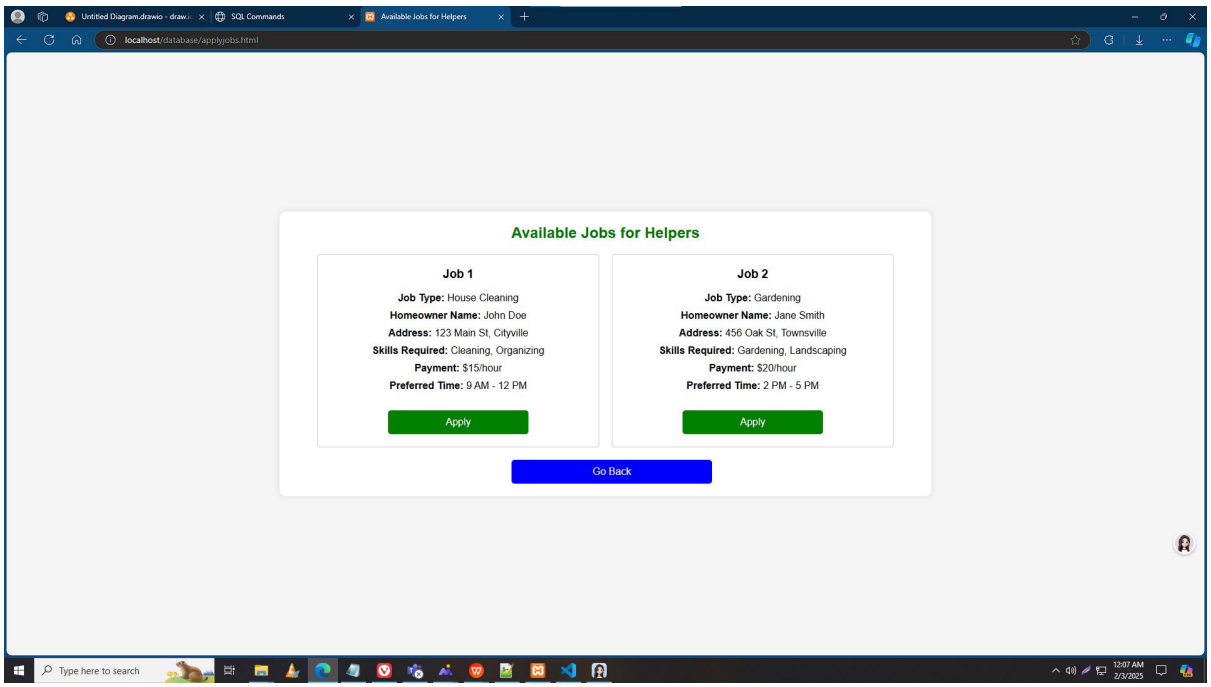
Add Homeowners



Job Submission Form



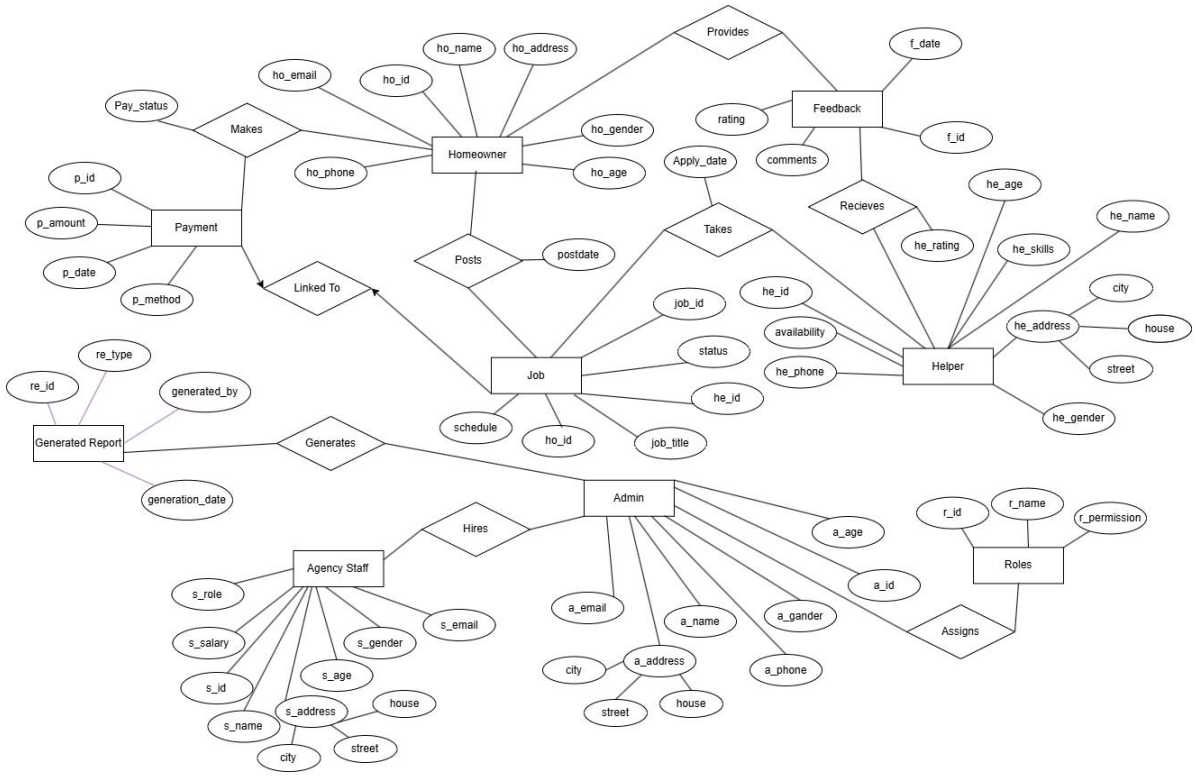
Helper Job Application



Scenario Description

The Homeowner-Helper Management System allows homeowners to efficiently hire helpers for various jobs. Each homeowner registers with a unique ID and maintains a profile that includes personal details such as name, age, gender, email, phone number, and address. Homeowners can post multiple job listings, each identified by a unique job ID, containing details like job title, schedule, status, and posting date. Helpers also register with unique IDs and provide information such as name, age, gender, skills, address, phone number, and availability. Helpers can apply for different jobs, and each job can receive applications from multiple helpers. Homeowners have the ability to provide feedback on the helpers they hire. This feedback includes a rating, comments, application date, and feedback date. Since a homeowner can hire different helpers for various jobs, they can provide multiple feedback entries, and each helper can receive feedback from different homeowners. The system tracks and updates the helper's overall rating based on these reviews. Payments for jobs are securely processed, with each payment linked to a specific job and homeowner. Homeowners can make multiple payments, each recorded with details such as payment ID, amount, date, method, and status. The system ensures transparency in financial transactions by keeping a clear record of all payments associated with job postings. To maintain system oversight, agency staff are responsible for generating reports that provide insights into activities within the platform. Reports contain details such as report ID, generation date, and type. Administrators oversee the entire system, managing users, job postings, and financial transactions. Each admin has a profile containing an ID, contact details, demographic information, and assigned roles. Roles define different permissions within the system, and an admin can be assigned multiple roles while each role can be assigned to different admins. By ensuring a well-structured job posting, application, payment, and feedback system, the platform promotes efficient hiring, secure transactions, and effective administrative management, offering a seamless experience for both homeowners and helpers.

ER Diagram


Provides
f_date
ho_name
ho_address
ho_email
ho_id
Pay_status
Makes
Feedback
rating
f_id
ho_gender
Homeowner
comments
Apply_date
ho_phone
ho_age
p_id
Recieves
he_age
p_amount
Payment
Takes
he_name
he_rating
he_skills
Posts
postdate
p_date
Linked To
p_method
job_id
he_id
city
availability
he_address
house
status
Helper
re_type
Job
he_phone
street
re_id
generated_by
he_id
schedule
ho_id
job_title
he_gender
Generated Report
Generates
generation_date
Admin
r_id
r_name
r_permission
Hires
a_age
Roles
Agency Staff
a_id
s_role
a_email
a_name
a_gander
s_email
Assigns
s_salary
s_gender
city
a_address
a_phone
s_age
s_id
street
house
s_address
house
s_name
street
city

Normalization

Relation: Posts

UNF:

ho_id, ho_name, ho_email, ho_address, ho_phone, ho_gender, ho_age, job_id, job_title, status, schedule, postdate

1NF:

ho_id, ho_name, ho_email, ho_address, ho_phone, ho_gender, ho_age, job_id, job_title, status, schedule, postdate

2NF

1. ho_id, ho_name, ho_email, ho_address, ho_phone, ho_gender, ho_age
2. job_id, ho_id, job_title, status, schedule, postdate

3NF

1. ho_id, ho_name, ho_email, ho_address, ho_phone, ho_gender, ho_age
2. job_id, ho_id, job_title, status, schedule, postdate
3. ho_id, job_id

Relation: Linked To

UNF:

job_id, job_title, status, schedule, ho_id, he_id, postdate, p_id, p_amount, p_date, p_method, pay_status

1NF:

job_id, job_title, status, schedule, ho_id, he_id, postdate, p_id, p_amount, p_date, p_method, pay_status

2NF:

1. job_id, job_title, status, schedule, ho_id, he_id, postdate
2. p_id, job_id, p_amount, p_date, p_method, pay_status

3NF:

1. job_id, job_title, status, schedule, ho_id, he_id, postdate
2. p_id, job_id, p_amount, p_date, p_method, pay_status
3. job_id, p_id

Relation: Makes

UNF:

ho_id, ho_name, ho_address (city, street, house), ho_gender, ho_phone, p_id, p_amount, p_date, p_method, pay_status

1NF:

ho_id, ho_name, city, street, house, ho_gender, ho_phone, p_id, p_amount, p_date, p_method, pay_status

2NF:

ho_id, ho_name, city, street, house, ho_gender, ho_phone

p_id, p_amount, p_date, p_method, pay_status, ho_id

3NF:

ho_id, ho_name, city, street, house, ho_gender, ho_phone

p_id, p_amount, p_date, p_method, pay_status

ho_id, p_id

Relation: Takes

UNF:

job_id, status, job_title, schedule, he_id, availability, he_name, he_address (city, street, house), he_gender, he_phone

1NF:

job_id, status, job_title, schedule, he_id, availability, he_name, city, street, house, he_gender, he_phone

2NF:

1. he_id, he_name, city, street, house, he_gender, he_phone, availability
2. job_id, status, job_title, schedule, he_id

3NF:

1. he_id, he_name, city, street, house, he_gender, he_phone
2. he_id, availability
3. job_id, status, job_title, schedule, he_id

Relation: Provides

UNF:

ho_id, ho_name, ho_address (city, street, house), ho_gender, ho_phone, f_id, rating, comments, apply_date

1NF:

ho_id, ho_name, city, street, house, ho_gender, ho_phone, f_id, rating, comments, apply_date

2NF:

1. ho_id, ho_name, city, street, house, ho_gender, ho_phone
2. f_id, rating, comments, apply_date, ho_id

3NF:

1. ho_id, ho_name, city, street, house, ho_gender, ho_phone
2. f_id, rating, comments, apply_date
3. ho_id, f_id

Relation: Receives

UNF:

he_id, he_name, he_address (city, street, house), he_gender, he_phone, f_id, f_date, rating, comments

1NF:

he_id, he_name, city, street, house, he_gender, he_phone, f_id, f_date, rating, comments

2NF:

1. he_id, he_name, city, street, house, he_gender, he_phone
2. f_id, f_date, rating, comments, he_id

3NF:

1. he_id, he_name, city, street, house, he_gender, he_phone
2. f_id, f_date, rating, comments
3. he_id, f_id

Relation: Assigns

UNF:

s_id, s_name, s_address (city, street, house), s_gender, s_email, r_id, r_name, r_permission

1NF:

s_id, s_name, city, street, house, s_gender, s_email, r_id, r_name, r_permission

2NF:

1. s_id, s_name, city, street, house, s_gender, s_email
2. r_id, r_name, r_permission, s_id

3NF:

1. s_id, s_name, city, street, house, s_gender, s_email
2. r_id, r_name, r_permission
3. s_id, r_id

Relation: Generates

UNF:

s_id, s_name, s_address (city, street, house), s_gender, s_email, re_id, re_type, generated_by, generation_date

1NF:

s_id, s_name, city, street, house, s_gender, s_email, re_id, re_type, generated_by, generation_date

2NF:

1. s_id, s_name, city, street, house, s_gender, s_email
2. re_id, re_type, generated_by, generation_date, s_id

3NF:

1. s_id, s_name, city, street, house, s_gender, s_email
2. re_id, re_type, generated_by, generation_date
3. s_id, re_id

Relation: Hires

UNF:

a_id, a_name, a_email, a_phone, a_address (city, street, house), a_age, a_gender, s_id, s_name, s_email, s_role, s_salary, s_age, s_gender, s_address (city, street, house)

1NF:

a_id, a_name, a_email, a_phone, a_address (city, street, house), a_age, a_gender, s_id, s_name, s_email, s_role, s_salary, s_age, s_gender, s_address (city, street, house)

2NF:

a_id, a_name, a_email, a_phone, a_street, a_house, a_city, a_age, a_gender
s_id, s_name, s_email, s_role, s_salary, s_age, s_gender, s_street, s_house, s_city

3NF:

a_id, a_name, a_email, a_phone, a_age, a_gender, address_id
s_id, s_name, s_email, s_role, s_salary, s_age, s_gender, address_id
a_id, s_id

Finalization

Tables:

1. ho_id, ho_name, ho_email, ho_address (city, street, house), ho_phone, ho_gender, ho_age
2. job_id, ho_id, job_title, status, schedule, postdate
3. ho_id, job_id
4. p_id, p_amount, p_date, p_method, pay_status
5. job_id, p_id
6. he_id, he_name, he_address (city, street, house), he_gender, he_phone
7. he_id, availability
8. f_id, rating, comments, apply_date, f_date
9. ho_id, f_id
10. he_id, f_id
11. r_id, r_name, r_permission
12. a_id, a_name, a_address (city, street, house), a_gender, a_phone, a_email, a_age
13. a_id, r_id
14. s_id, s_name, s_address (city, street, house), s_gender, s_email, s_salary, s_role, s_age
15. re_id, re_type, generated_by, generation_date
16. s_id, re_id
17. a_id, s_id

SchemaDiagram

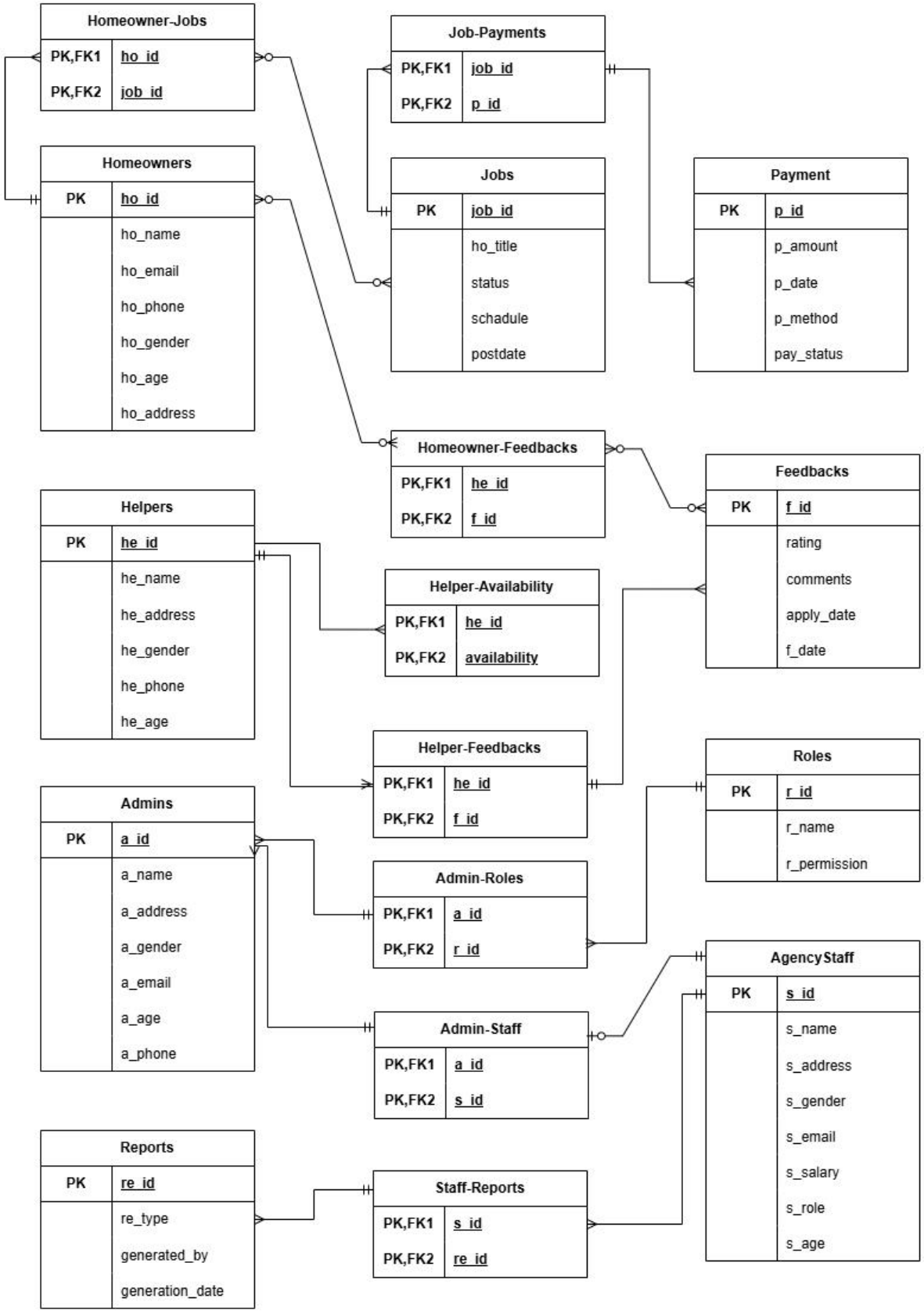


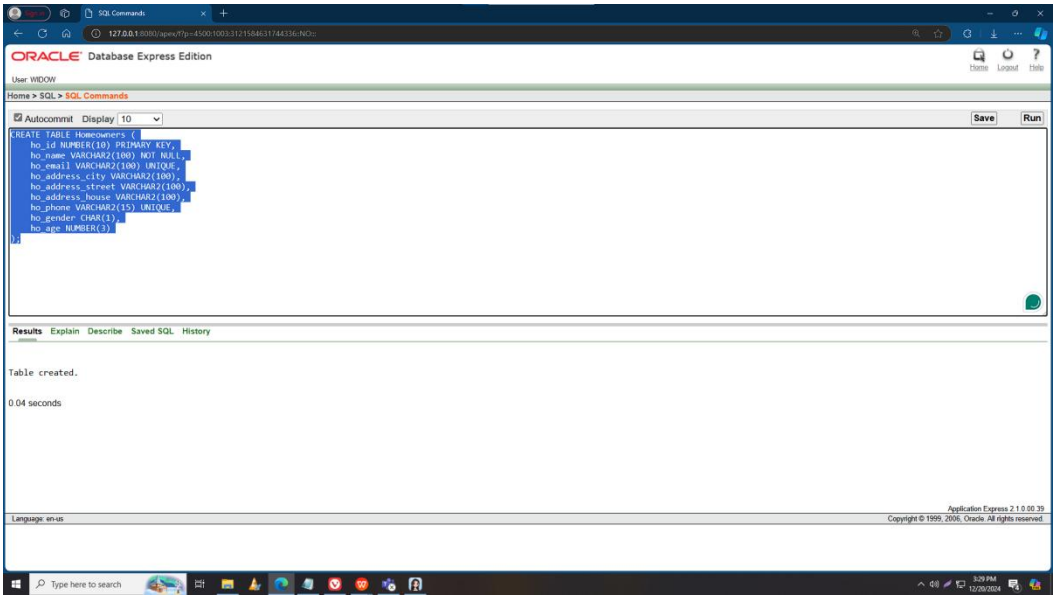
Table Creation

Table: Homeowners

Sequence: CREATE SEQUENCE homeowners_seq START WITH 10001 INCREMENT BY 1 NOCACHE NOCYCLE;

Table Query:

```
CREATE TABLE Homeowners (
    ho_id NUMBER(10) PRIMARY KEY,
    ho_name VARCHAR2(100) NOT NULL,
    ho_email VARCHAR2(100) UNIQUE,
    ho_password VARCHAR2(255) NOT NULL,
    ho_address_city VARCHAR2(100),
    ho_address_street VARCHAR2(100),
    ho_address_house VARCHAR2(100),
    ho_phone VARCHAR2(15) UNIQUE,
    ho_gender CHAR(1),
    ho_age NUMBER(3)
);
```



Index: CREATE INDEX homeowners_idx_email ON Homeowners(ho_id, ho_email);

Table: Jobs

Sequence: CREATE SEQUENCE jobs_seq START WITH 20001 INCREMENT BY 1 NOCACHE NOCYCLE;

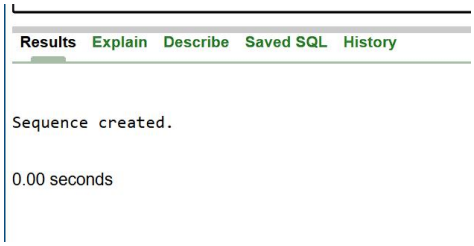


Table Query:

```
CREATE TABLE Jobs (
    job_id NUMBER(10) PRIMARY KEY,
    ho_id NUMBER(10),
    job_title VARCHAR2(100),
    status VARCHAR2(50),
    schedule DATE,
    postdate DATE,
FOREIGN KEY (ho_id) REFERENCES Homeowners(ho_id));
```

Index: CREATE INDEX jobs_idx_title ON Jobs(job_id, job_title);

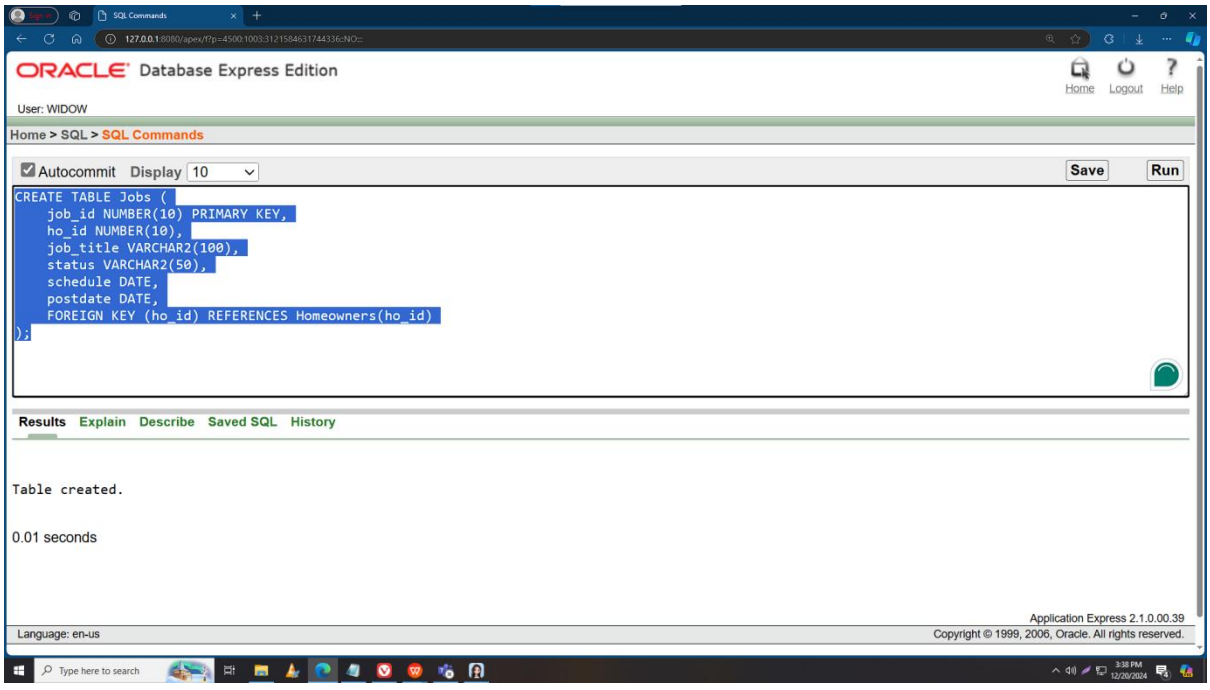


Table: Homeowner-Jobs

Table Query:

```
CREATE TABLE HomeownerJobs (
    ho_id NUMBER(10),
    job_id NUMBER(10),
    PRIMARY KEY (ho_id, job_id),
    FOREIGN KEY (ho_id) REFERENCES Homeowners(ho_id),
    FOREIGN KEY (job_id) REFERENCES Jobs(job_id)
);
```

Index: CREATE INDEX homeownerjobs_idx ON HomeownerJobs(ho_id, job_id);

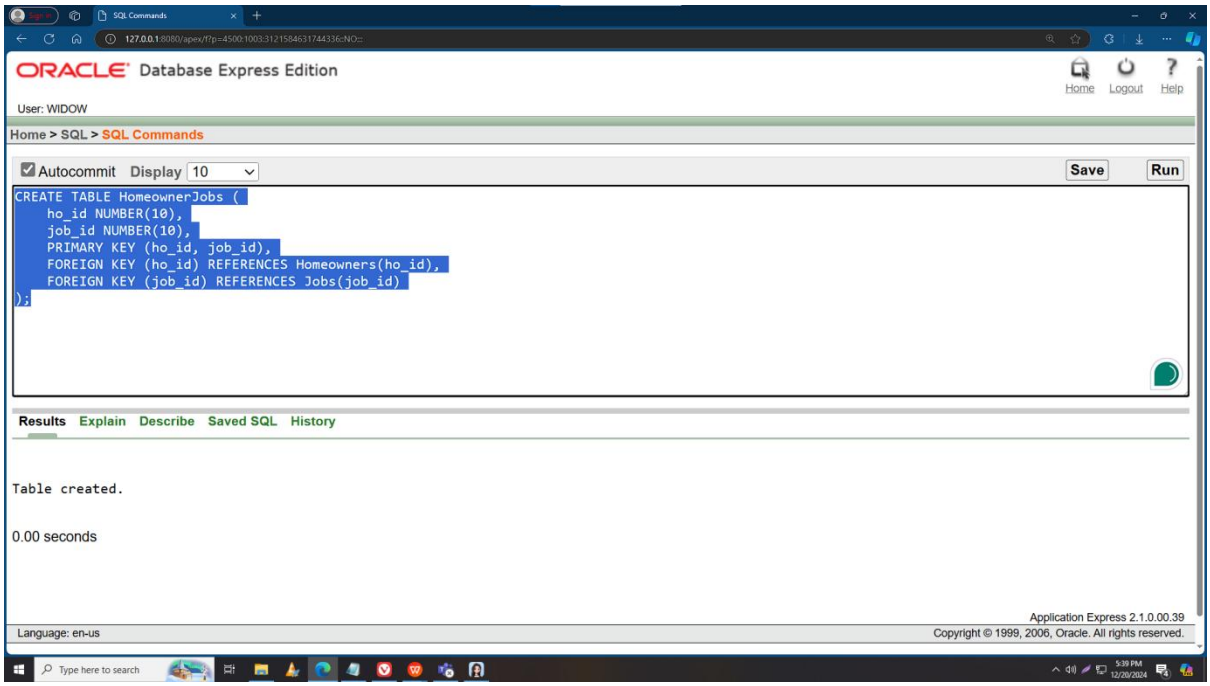


Table: Payments

Seuqence: CREATE SEQUENCE payments_seq START WITH 30001 INCREMENT BY 1 NOCACHE NOCYCLE;

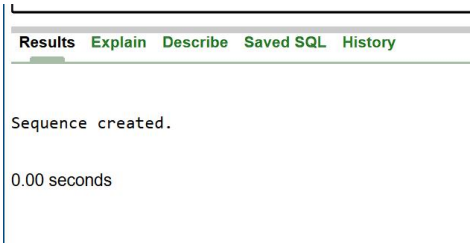
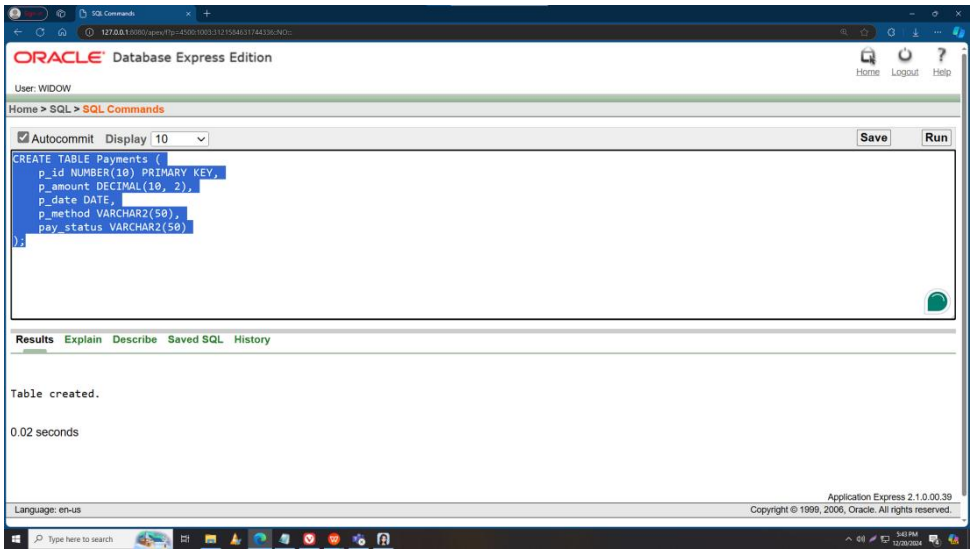


Table Query:

```
CREATE TABLE Payments (
    p_id NUMBER(10) PRIMARY KEY,
    p_amount DECIMAL(10, 2),
    p_date DATE,
    p_method VARCHAR2(50),
    pay_status VARCHAR2(50)
);
```



Index: CREATE INDEX payments_idx_status ON Payments(p_id, pay_status);

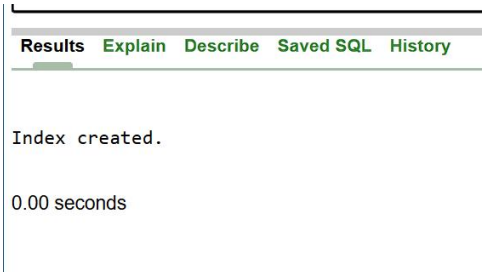
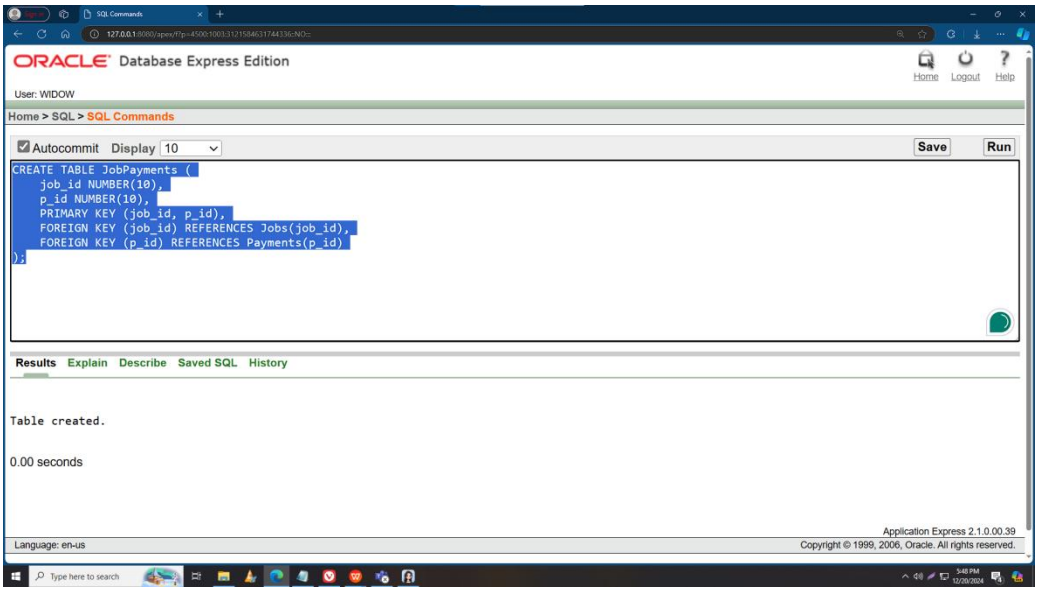


Table: Job-Payments

Table Query:

```
CREATE TABLE JobPayments (
    job_id NUMBER(10),
    p_id NUMBER(10),
    PRIMARY KEY (job_id, p_id),
    FOREIGN KEY (job_id) REFERENCES Jobs(job_id),
    FOREIGN KEY (p_id) REFERENCES Payments(p_id));
```



Index: CREATE INDEX jobpayments_idx ON JobPayments(job_id, p_id);

Table: Helpers

Sequence: CREATE SEQUENCE helpers_seq START WITH 40001 INCREMENT BY 1 NOCACHE NOCYCLE;

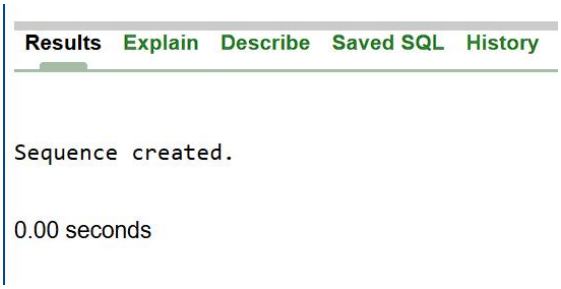
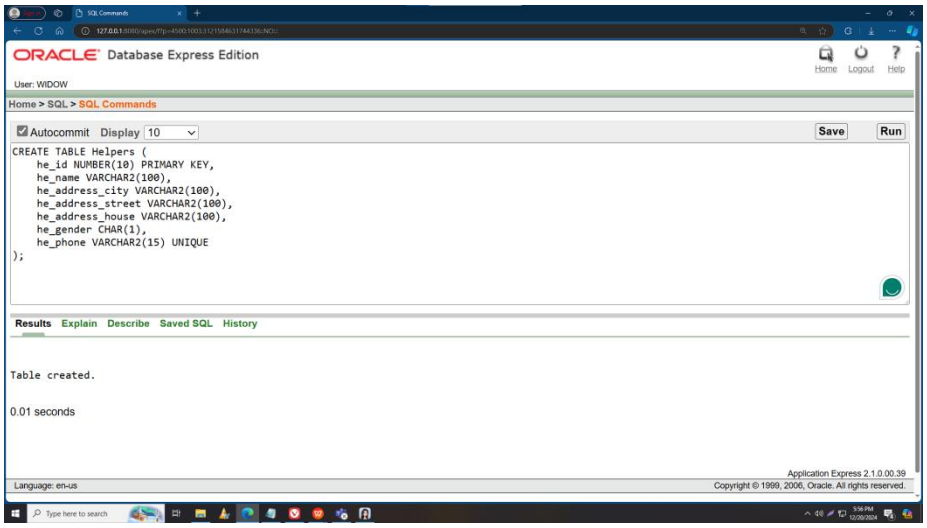


Table Query:

```
CREATE TABLE Helpers (
    he_id NUMBER(10) PRIMARY KEY,
    he_name VARCHAR2(100),
    he_password VARCHAR2(255) NOT NULL,
    he_address_city VARCHAR2(100),
    he_address_street VARCHAR2(100),
    he_address_house VARCHAR2(100),
    he_gender CHAR(1),
    he_phone VARCHAR2(15) UNIQUE
);
```



Index: CREATE INDEX helpers_idx_phone ON Helpers(he_id, he_phone);

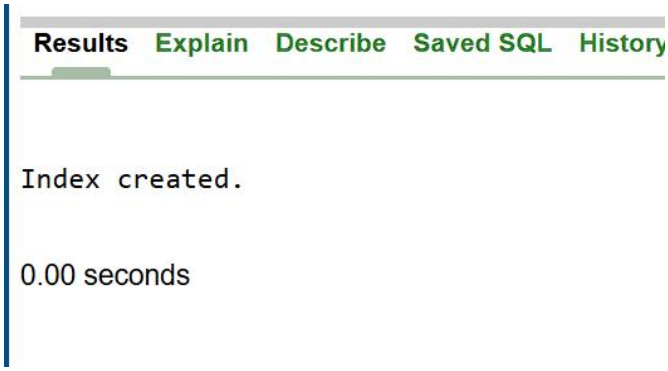
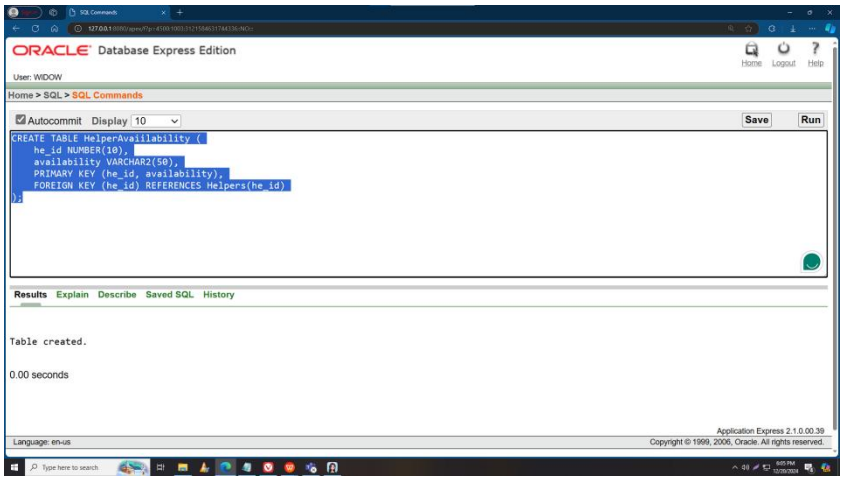


Table: helper-Availability

Table Query:

```
CREATE TABLE Helper_Availability (
    he_id NUMBER(10),
    availability VARCHAR2(50),
    PRIMARY KEY (he_id, availability),
    FOREIGN KEY (he_id) REFERENCES Helpers(he_id)
);
```



Index: CREATE INDEX helperavailability_idx ON HelperAvailability(he_id, availability);

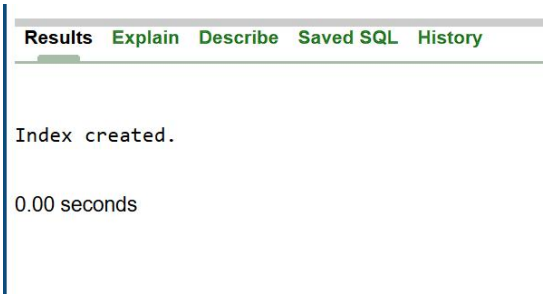


Table: Feedbacks

Sequence:

CREATE SEQUENCE feedbacks_seq START WITH 50001 INCREMENT BY 1 NOCACHE NOCYCLE;

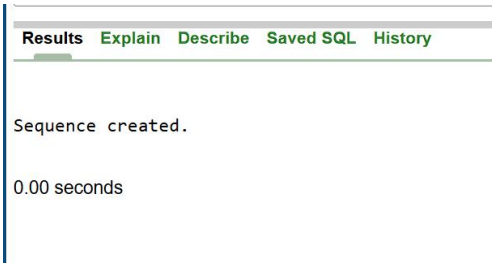
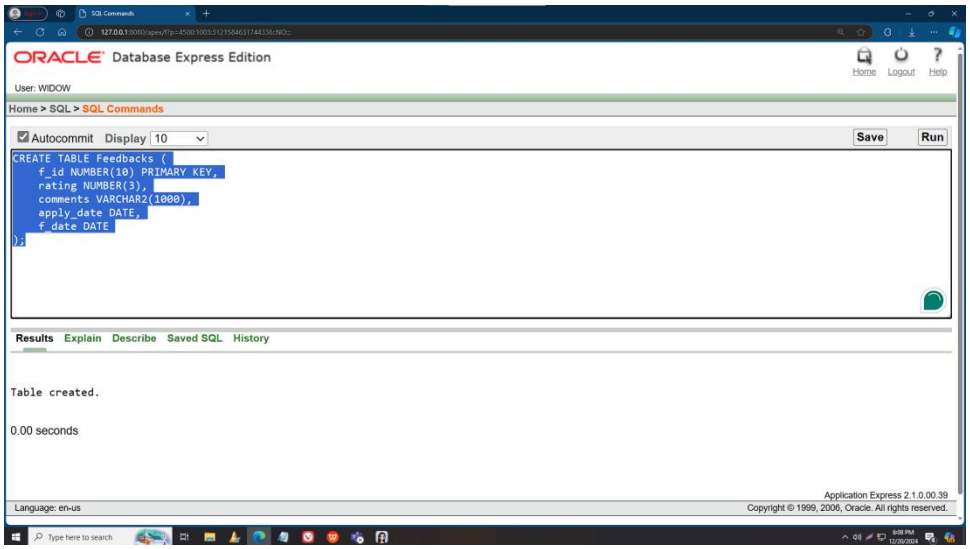


Table Query:

```
CREATE TABLE Feedbacks (
    f_id NUMBER(10) PRIMARY KEY,
    rating NUMBER(3),
    comments VARCHAR2(1000),
    apply_date DATE,
f_date DATE);
```



Index: CREATE INDEX feedbacks_idx_rating ON Feedbacks(f_id, rating);

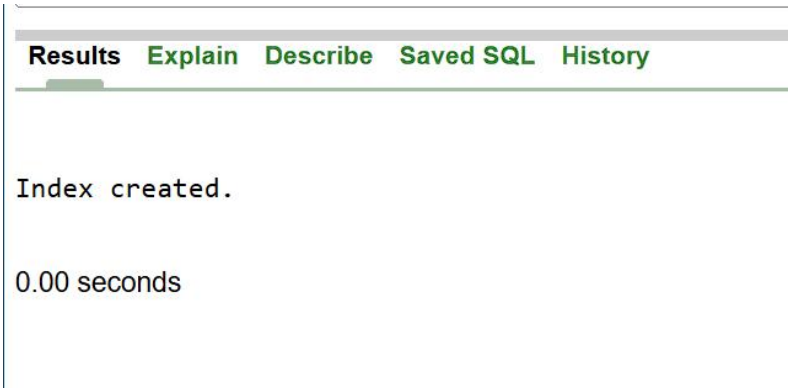
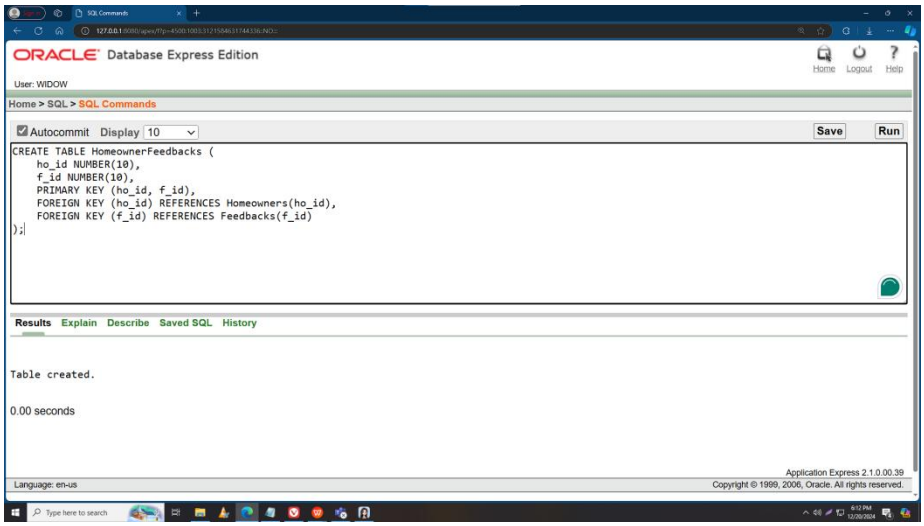


Table: Homeowner-Feedbacks

Table Query:

```
CREATE TABLE HomeownerFeedbacks (
    ho_id NUMBER(10),
    f_id NUMBER(10),
    PRIMARY KEY (ho_id, f_id),
FOREIGN KEY (ho_id) REFERENCES Homeowners(ho_id),
    FOREIGN KEY (f_id) REFERENCES Feedbacks(f_id)
);
```



Index: CREATE INDEX homeownerfeedbacks_idx ON HomeownerFeedbacks(ho_id, f_id);

Table: Helper-Feedback

Table Query:

```
CREATE TABLE HelperFeedbacks (
    he_id NUMBER(10),
    f_id NUMBER(10),
    PRIMARY KEY (he_id, f_id),
    FOREIGN KEY (he_id) REFERENCES Helpers(he_id),
    FOREIGN KEY (f_id) REFERENCES Feedbacks(f_id)
);
```

Index: CREATE INDEX helperfeedbacks_idx ON HelperFeedbacks(he_id, f_id);

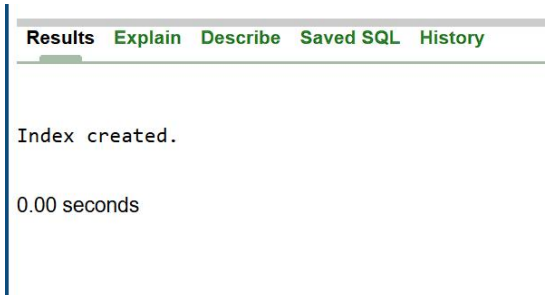


Table: Roles

Sequence:

```
CREATE SEQUENCE roles_seq START WITH 60001 INCREMENT BY 1
NOCACHE NOCYCLE;
```

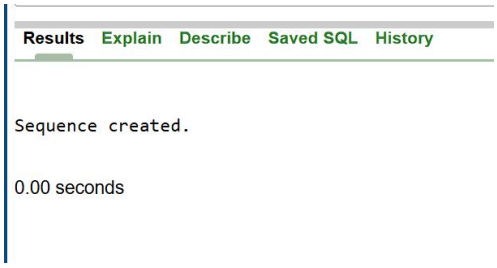
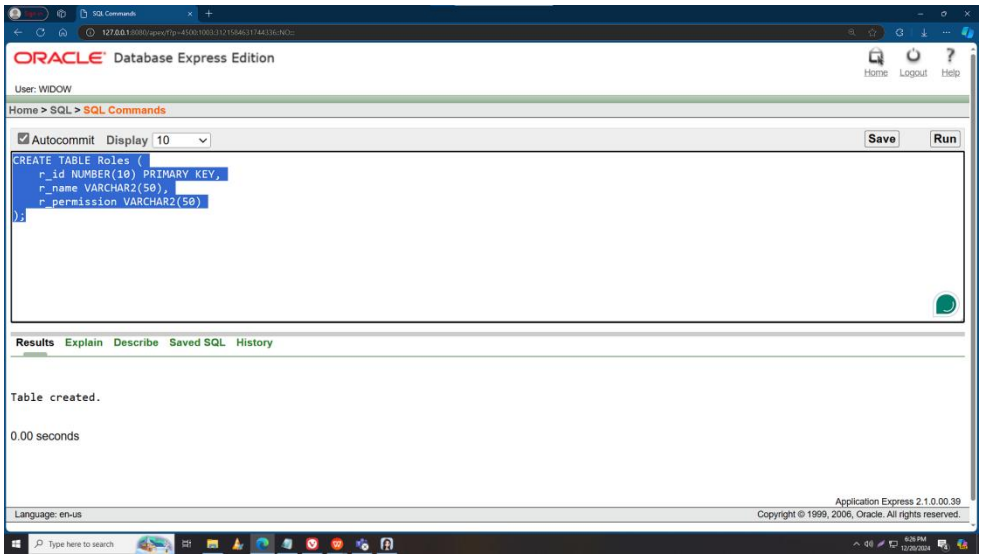


Table Query:

```
CREATE TABLE Roles (
    r_id NUMBER(10) PRIMARY KEY,
    r_name VARCHAR2(50),
    r_permission VARCHAR2(50)
);
```



Index: CREATE INDEX roles_idx_name ON Roles(r_id, r_name);

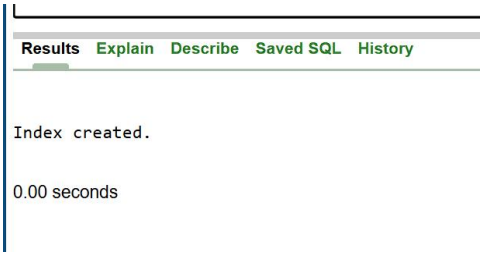


Table: Admin

Sequence:

CREATE SEQUENCE admins_seq START WITH 70001 INCREMENT BY 1

NOCACHE NOCYCLE;

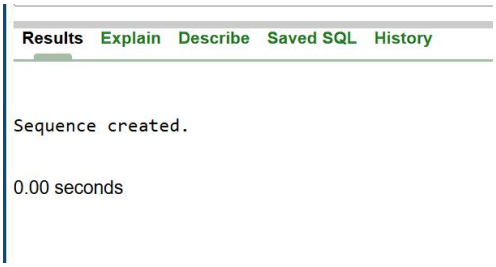
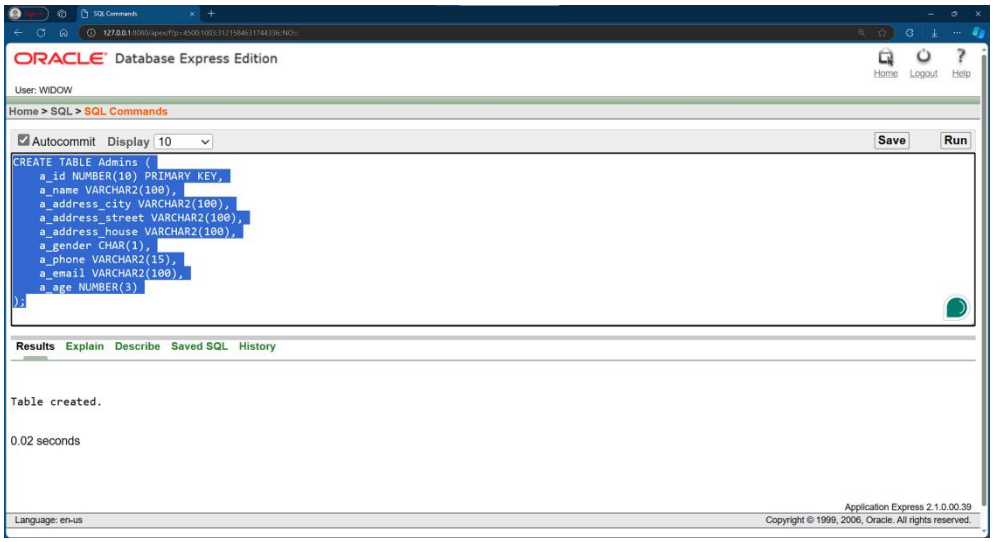


Table Query:

```
CREATE TABLE Admins (
    a_id NUMBER(10) PRIMARY KEY,
    a_name VARCHAR2(100),
    a_password VARCHAR2(255) NOT NULL,
    a_address_city VARCHAR2(100),
    a_address_street VARCHAR2(100),
    a_address_house VARCHAR2(100),
    a_gender CHAR(1),
    a_phone VARCHAR2(15) UNIQUE,
    a_email VARCHAR2(100) UNIQUE,
    a_age NUMBER(3)
);
```



Index: CREATE INDEX admins_idx_email ON Admins(a_id, a_email);

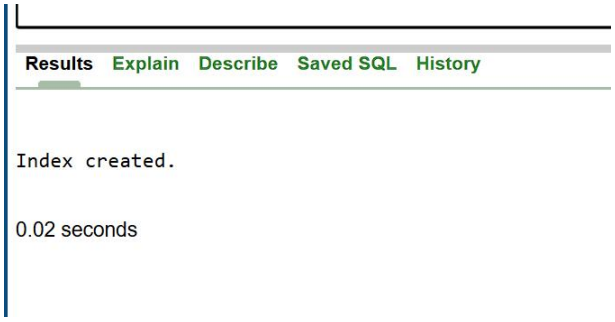
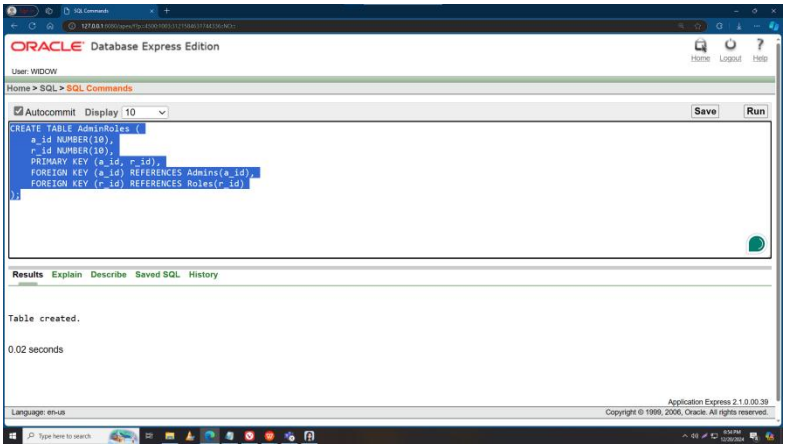


Table: Admin-Roles

Table Query:

```
CREATE TABLE AdminRoles (
    a_id NUMBER(10),
    r_id NUMBER(10),
    PRIMARY KEY (a_id, r_id),
    FOREIGN KEY (a_id) REFERENCES Admins(a_id),
FOREIGN KEY (r_id) REFERENCES Roles(r_id));
```



Index: CREATE INDEX adminroles_idx ON AdminRoles(a_id, r_id);

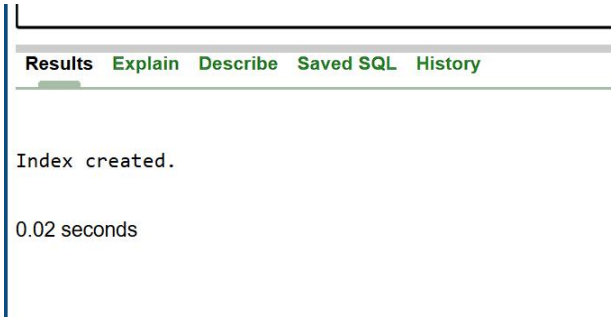
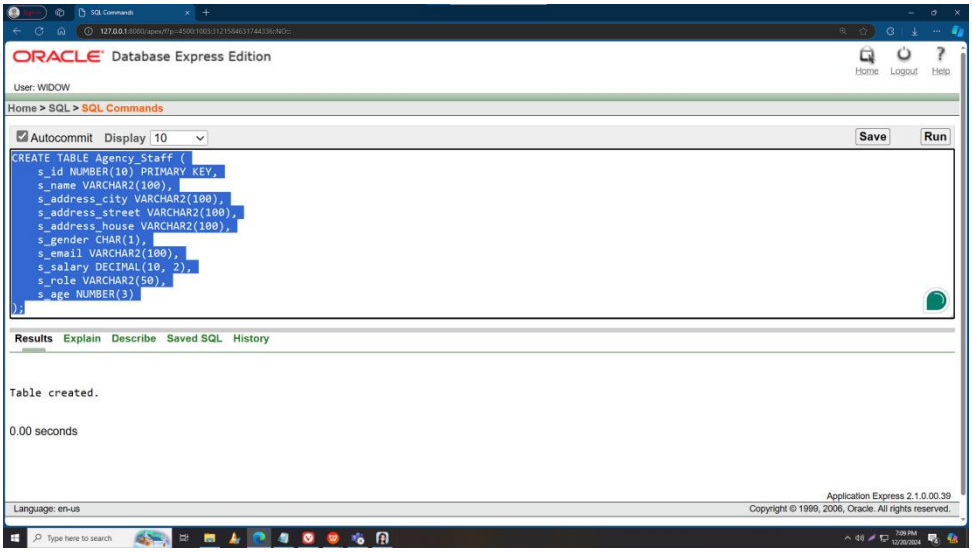


Table: Agency Staff

Sequence: CREATE SEQUENCE staff_seq START WITH 80001 INCREMENT BY 1 NOCACHE NOCYCLE;

Table Query:

```
CREATE TABLE Agency_Staff (
    s_id NUMBER(10) PRIMARY KEY,
    s_name VARCHAR2(100),
    s_password VARCHAR2(255) NOT NULL,
    s_address_city VARCHAR2(100),
    s_address_street VARCHAR2(100),
    s_address_house VARCHAR2(100),
    s_gender CHAR(1),
    s_email VARCHAR2(100) UNIQUE,
    s_salary DECIMAL(10, 2),
    s_role VARCHAR2(50),
    s_age NUMBER(3)
);
```



Index: CREATE INDEX staff_idx_email ON AgencyStaff(s_id, s_email);

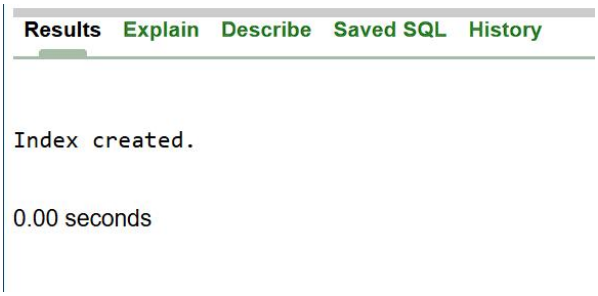


Table: Reports

Sequence: CREATE SEQUENCE reports_seq START WITH 90001 INCREMENT BY 1 NOCACHE NOCYCLE;

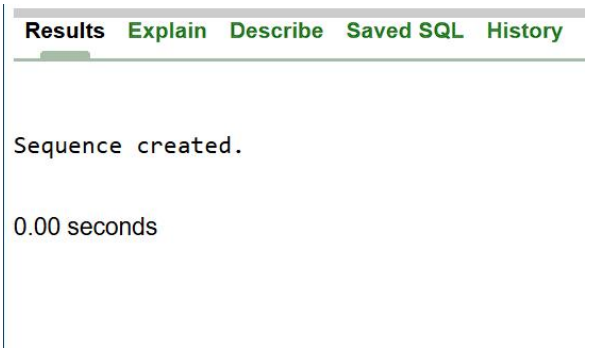


Table Query:

```
CREATE TABLE Reports (
    re_id NUMBER(10) PRIMARY KEY,
    re_type VARCHAR2(50),
    generated_by VARCHAR2(100),
    generation_date DATE
);
```

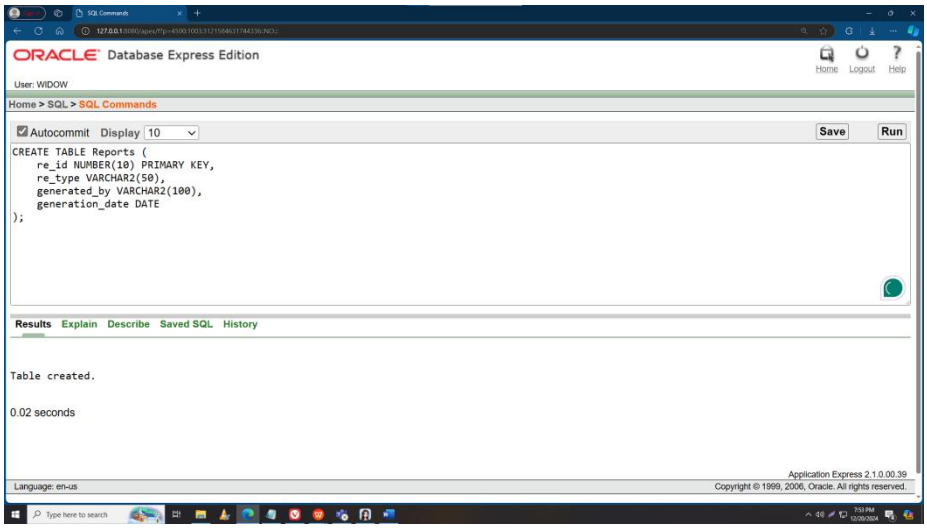


Table Index:

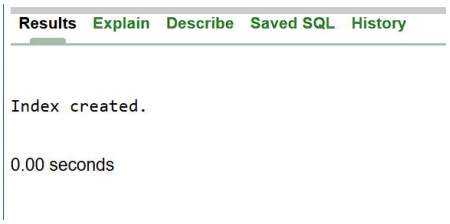


Table: Staff Reports

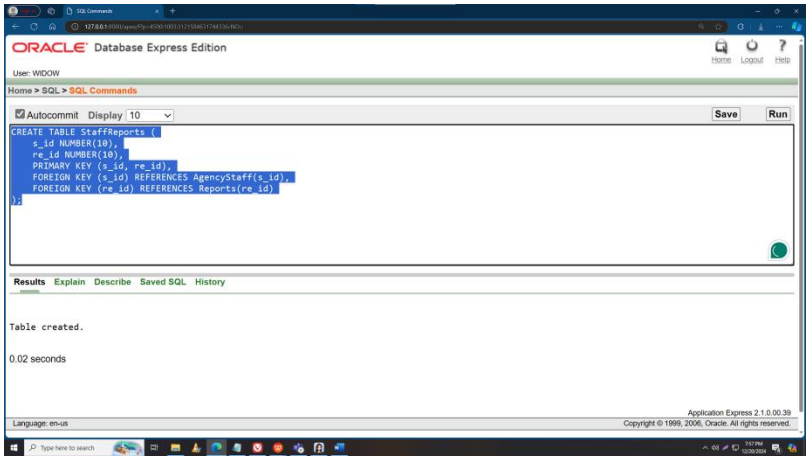
Table Query:

```
CREATE TABLE StaffReports (
    s_id NUMBER(10),
    re_id NUMBER(10),
    PRIMARY KEY (s_id, re_id),
```

FOREIGN KEY (s_id) REFERENCES AgencyStaff(s_id),

FOREIGN KEY (re_id) REFERENCES Reports(re_id)

);



Index: CREATE INDEX staffreports_idx ON StaffReports(s_id, re_id);

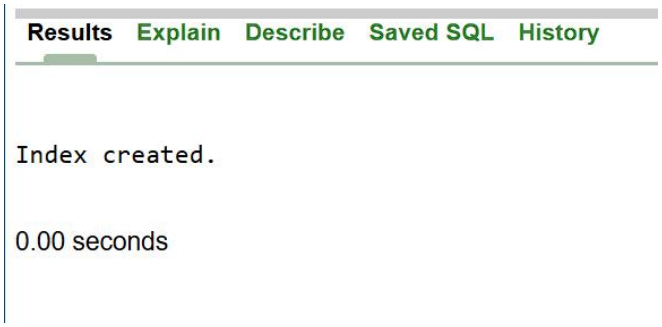


Table: Stuff-Admin

Sequence:

CREATE SEQUENCE admin_seq START WITH 1 INCREMENT BY 1;

CREATE SEQUENCE staff_seq START WITH 1 INCREMENT BY 1;

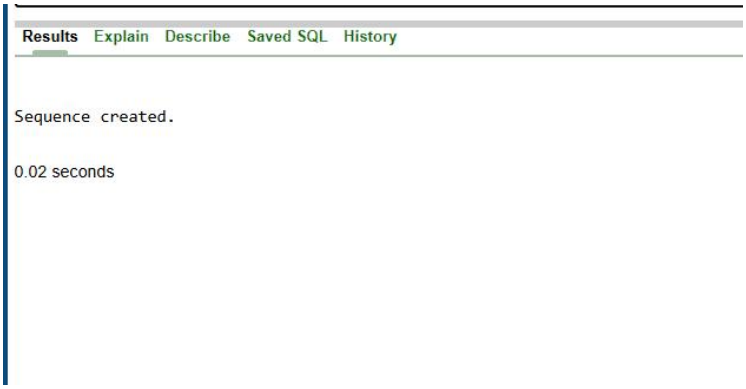
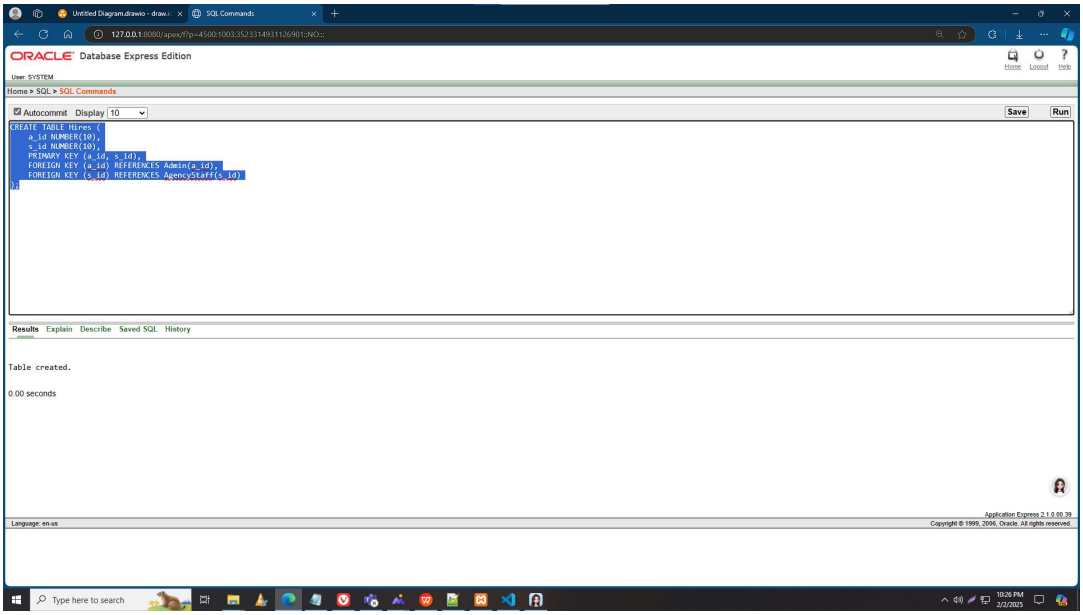


Table Creation:

```
CREATE TABLE Hires (
    a_id NUMBER(10),
    s_id NUMBER(10),
    PRIMARY KEY (a_id, s_id),
    FOREIGN KEY (a_id) REFERENCES Admin(a_id),
    FOREIGN KEY (s_id) REFERENCES AgencyStaff(s_id)
);
```



Data Insertion

Table: Homeowners

INSERT INTO Homeowners (ho_id, ho_name, ho_email, ho_password, ho_address_city, ho_address_street, ho_address_house, ho_phone, ho_gender, ho_age)

VALUES (1, 'John Doe', 'john.doe@example.com', 'pass123', 'New York', '5th Avenue', 'Apt 12B', '1234567890', 'M', 35);

INSERT INTO Homeowners (ho_id, ho_name, ho_email, ho_password, ho_address_city, ho_address_street, ho_address_house, ho_phone, ho_gender, ho_age)

VALUES (2, 'Jane Smith', 'jane.smith@example.com', 'pass123', 'Los Angeles', 'Sunset Blvd', 'House 24', '2345678901', 'F', 29);

INSERT INTO Homeowners (ho_id, ho_name, ho_email, ho_password, ho_address_city, ho_address_street, ho_address_house, ho_phone, ho_gender, ho_age)

VALUES (3, 'Robert Brown', 'robert.brown@example.com', 'pass123', 'Chicago', 'Lake Shore Dr', 'Unit 7C', '3456789012', 'M', 42);

INSERT INTO Homeowners (ho_id, ho_name, ho_email, ho_password, ho_address_city, ho_address_street, ho_address_house, ho_phone, ho_gender, ho_age)

VALUES (4, 'Emily Johnson', 'emily.johnson@example.com', 'pass123', 'Houston', 'Main St', 'Suite 10A', '4567890123', 'F', 31);

INSERT INTO Homeowners (ho_id, ho_name, ho_email, ho_password, ho_address_city, ho_address_street, ho_address_house, ho_phone, ho_gender, ho_age)

VALUES (5, 'Michael Davis', 'michael.davis@example.com', 'pass123', 'San Francisco', 'Market St', 'Loft 5D', '5678901234', 'M', 38);

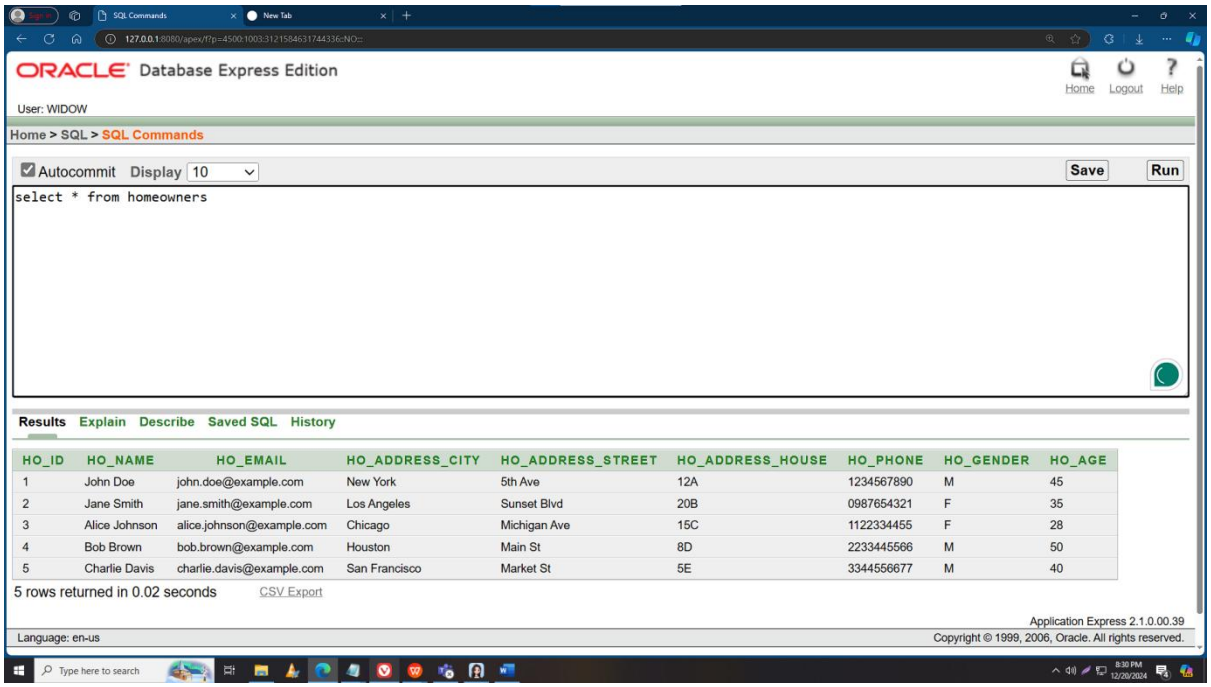


Table: Jobs

INSERT INTO Jobs (job_id, ho_id, job_title, status, schedule, postdate) VALUES (1, 1, 'Lawn Mowing', 'Completed', TO_DATE('2024-04-10', 'YYYY-MM-DD'), TO_DATE('2024-04-01', 'YYYY-MM-DD'));

INSERT INTO Jobs (job_id, ho_id, job_title, status, schedule, postdate) VALUES (2, 2, 'Roof Repair', 'Pending', TO_DATE('2024-04-20', 'YYYY-MM-DD'), TO_DATE('2024-04-10', 'YYYY-MM-DD'));

INSERT INTO Jobs (job_id, ho_id, job_title, status, schedule, postdate) VALUES (3, 3, 'Painting', 'In Progress', TO_DATE('2024-04-15', 'YYYY-MM-DD'), TO_DATE('2024-04-05', 'YYYY-MM-DD'));

INSERT INTO Jobs (job_id, ho_id, job_title, status, schedule, postdate) VALUES (4, 4, 'Electrical Fix', 'Completed', TO_DATE('2024-04-12', 'YYYY-MM-DD'), TO_DATE('2024-04-02', 'YYYY-MM-DD'));

INSERT INTO Jobs (job_id, ho_id, job_title, status, schedule, postdate) VALUES (5, 5, 'Plumbing', 'Pending', TO_DATE('2024-04-25', 'YYYY-MM-DD'), TO_DATE('2024-04-15', 'YYYY-MM-DD'));

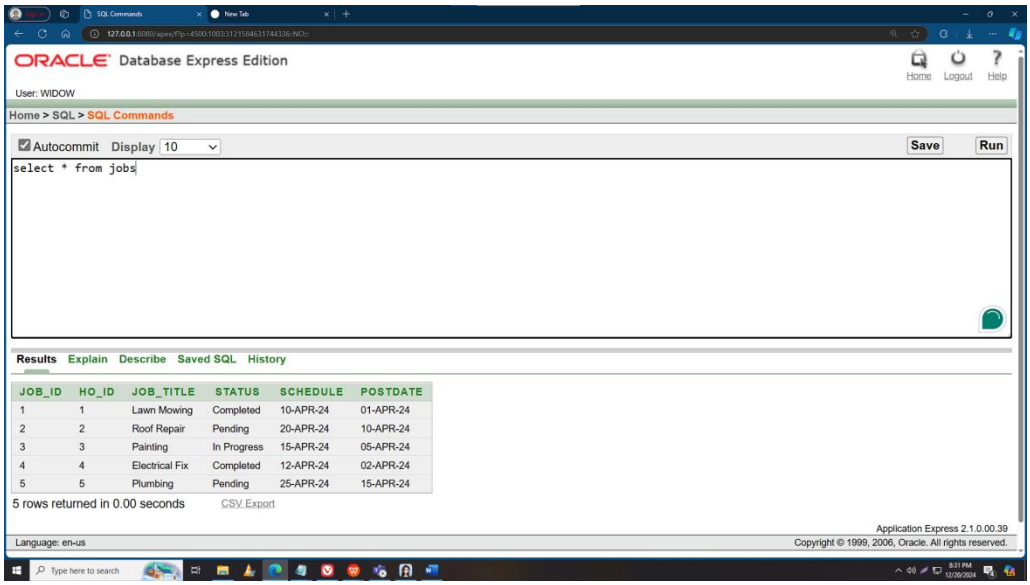


Table: Homeowner - Jobs

INSERT INTO HomeownerJobs (ho_id, job_id) VALUES (1, 1);

INSERT INTO HomeownerJobs (ho_id, job_id) VALUES (2, 2);

INSERT INTO HomeownerJobs (ho_id, job_id) VALUES (3, 3);

INSERT INTO HomeownerJobs (ho_id, job_id) VALUES (4, 4);

INSERT INTO HomeownerJobs (ho_id, job_id) VALUES (5, 5);

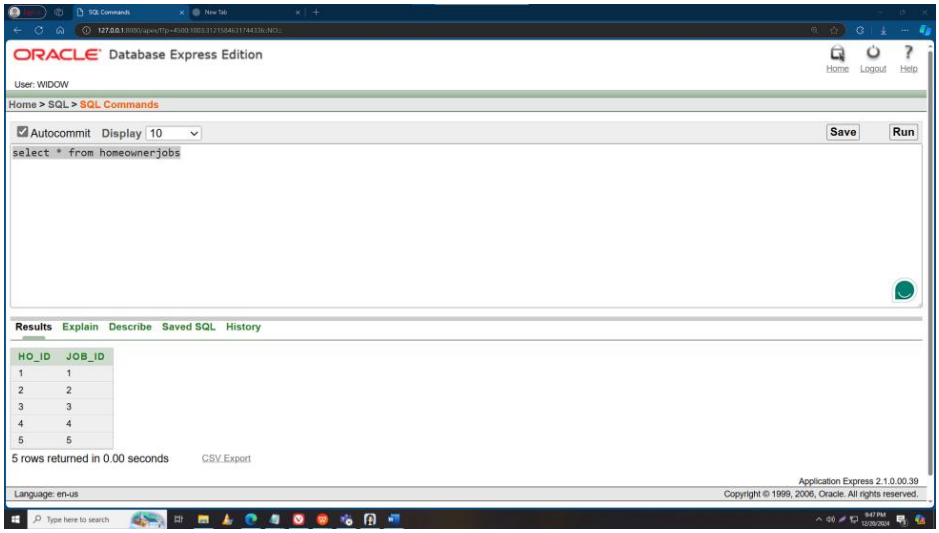


Table: Payments

INSERT INTO Payments (p_id, p_amount, p_date, p_method, pay_status) VALUES (1, 150.00, TO_DATE('2024-04-11', 'YYYY-MM-DD'), 'Credit Card', 'Completed');

INSERT INTO Payments (p_id, p_amount, p_date, p_method, pay_status) VALUES (2, 250.00, TO_DATE('2024-04-21', 'YYYY-MM-DD'), 'PayPal', 'Pending');

INSERT INTO Payments (p_id, p_amount, p_date, p_method, pay_status) VALUES (3, 100.00, TO_DATE('2024-04-16', 'YYYY-MM-DD'), 'Debit Card', 'Completed');

INSERT INTO Payments (p_id, p_amount, p_date, p_method, pay_status) VALUES (4, 200.00, TO_DATE('2024-04-13', 'YYYY-MM-DD'), 'Bank Transfer', 'Pending');

INSERT INTO Payments (p_id, p_amount, p_date, p_method, pay_status) VALUES (5, 300.00, TO_DATE('2024-04-26', 'YYYY-MM-DD'), 'Cash', 'Completed');

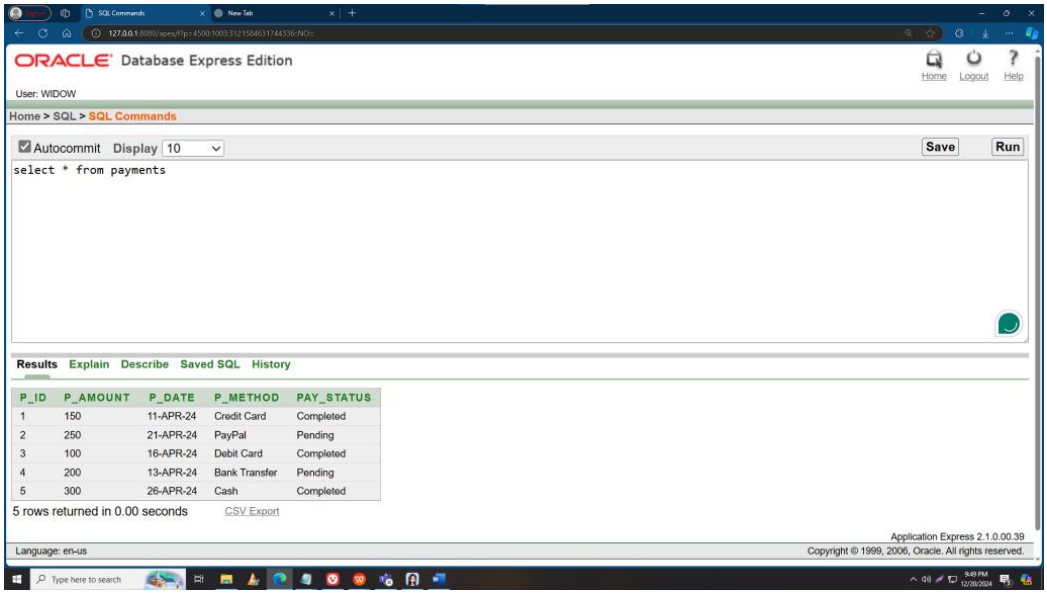


Table: Job-Payments

INSERT INTO JobPayments (job_id, p_id) VALUES (1, 1);

INSERT INTO JobPayments (job_id, p_id) VALUES (2, 2);

INSERT INTO JobPayments (job_id, p_id) VALUES (3, 3);

INSERT INTO JobPayments (job_id, p_id) VALUES (4, 4);

INSERT INTO JobPayments (job_id, p_id) VALUES (5, 5);

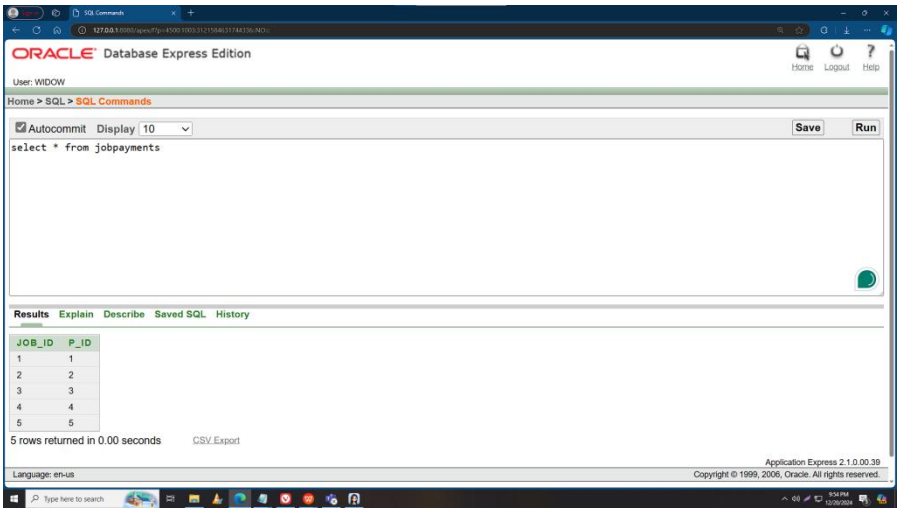


Table: Helpers

INSERT INTO Helpers (he_id, he_name, he_password, he_address_city, he_address_street, he_address_house, he_gender, he_phone)

VALUES (1, 'Alice Green', 'pass123', 'New York', 'Madison Ave', 'Apt 3C', 'F', '1234567890');

INSERT INTO Helpers (he_id, he_name, he_password, he_address_city, he_address_street, he_address_house, he_gender, he_phone)

VALUES (2, 'Brian Adams', 'pass123', 'Los Angeles', 'Hollywood Blvd', 'House 14', 'M', '2345678901');

INSERT INTO Helpers (he_id, he_name, he_password, he_address_city, he_address_street, he_address_house, he_gender, he_phone)

VALUES (3, 'Sophia Carter', 'pass123', 'Chicago', 'Michigan Ave', 'Unit 5B', 'F', '3456789012');

INSERT INTO Helpers (he_id, he_name, he_password, he_address_city, he_address_street, he_address_house, he_gender, he_phone)

VALUES (4, 'Daniel White', 'pass123', 'Houston', 'Broadway St', 'Suite 22A', 'M', '4567890123');

INSERT INTO Helpers (he_id, he_name, he_password, he_address_city, he_address_street, he_address_house, he_gender, he_phone)

VALUES (5, 'Emma Wilson', 'pass123', 'San Francisco', 'Van Ness Ave', 'Loft 7D', 'F', '5678901234');

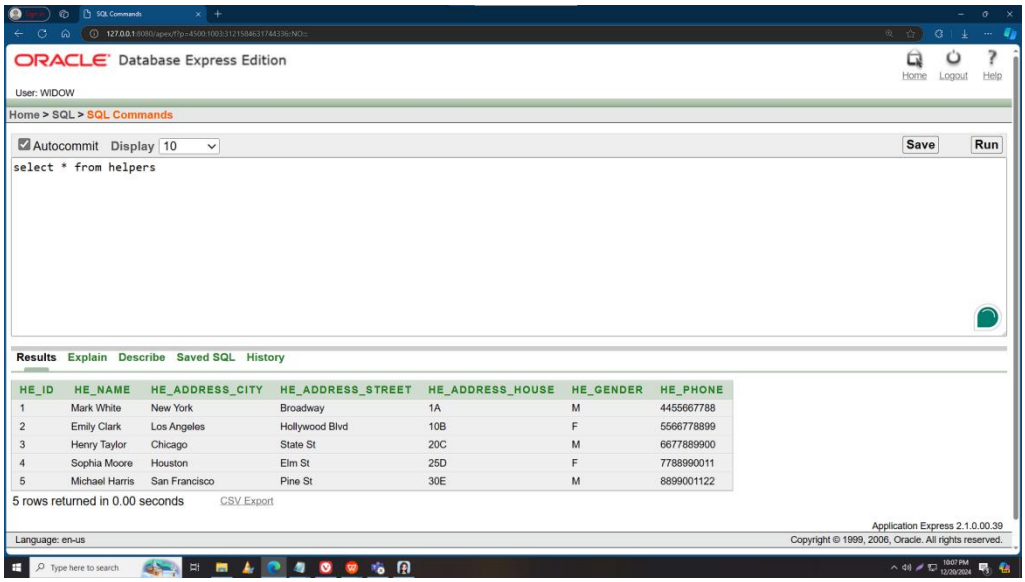


Table: Helper-Availability

INSERT INTO Helper_Availability (he_id, availability) VALUES (1, 'Morning');

INSERT INTO Helper_Availability (he_id, availability) VALUES (2, 'Afternoon');

INSERT INTO Helper_Availability (he_id, availability) VALUES (3, 'Evening');

INSERT INTO Helper_Availability (he_id, availability) VALUES (4, 'Morning');

INSERT INTO Helper_Availability (he_id, availability) VALUES (5, 'Afternoon');

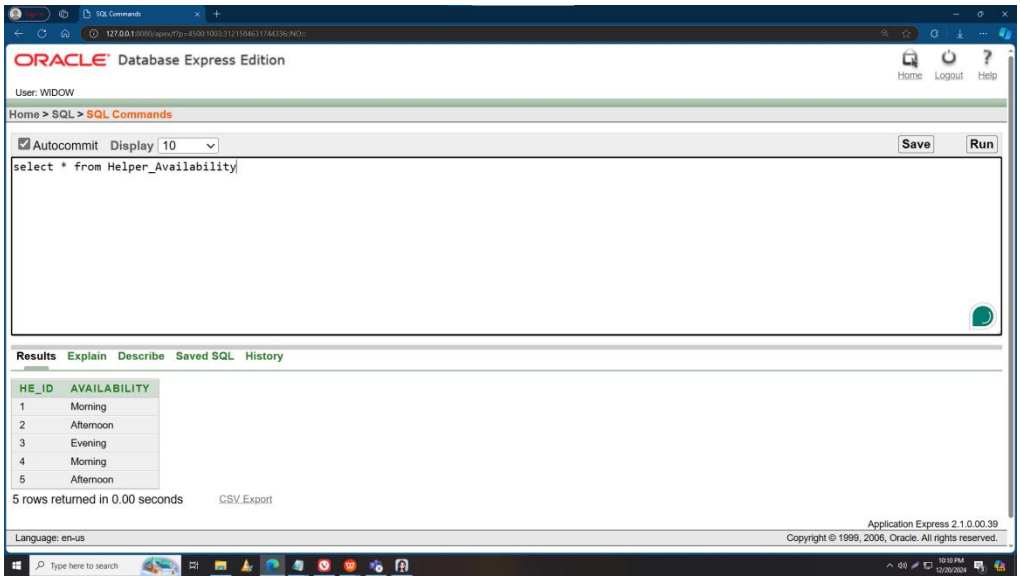


Table: Feedbacks

INSERT INTO Feedbacks (f_id, rating, comments, apply_date, f_date) VALUES (1, 5, 'Excellent job!', TO_DATE('2024-04-10', 'YYYY-MM-DD'), TO_DATE('2024-04-11', 'YYYY-MM-DD'));

INSERT INTO Feedbacks (f_id, rating, comments, apply_date, f_date) VALUES (2, 4, 'Good work!', TO_DATE('2024-04-20', 'YYYY-MM-DD'), TO_DATE('2024-04-21', 'YYYY-MM-DD'));

INSERT INTO Feedbacks (f_id, rating, comments, apply_date, f_date) VALUES (3, 3, 'Satisfactory', TO_DATE('2024-04-15', 'YYYY-MM-DD'), TO_DATE('2024-04-16', 'YYYY-MM-DD'));

INSERT INTO Feedbacks (f_id, rating, comments, apply_date, f_date) VALUES (4, 2, 'Needs improvement', TO_DATE('2024-04-12', 'YYYY-MM-DD'), TO_DATE('2024-04-13', 'YYYY-MM-DD'));

INSERT INTO Feedbacks (f_id, rating, comments, apply_date, f_date) VALUES (5, 1, 'Poor service', TO_DATE('2024-04-25', 'YYYY-MM-DD'), TO_DATE('2024-04-26', 'YYYY-MM-DD'));

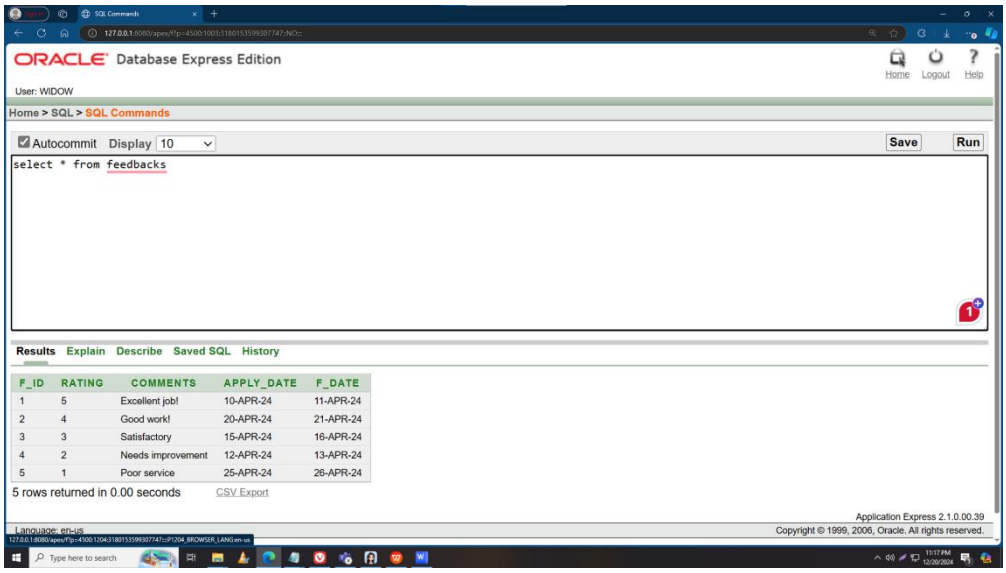


Table: Homeowner-Feedbacks

INSERT INTO HomeownerFeedbacks (ho_id, f_id) VALUES (1, 1);

INSERT INTO HomeownerFeedbacks (ho_id, f_id) VALUES (2, 2);

INSERT INTO HomeownerFeedbacks (ho_id, f_id) VALUES (3, 3);

INSERT INTO HomeownerFeedbacks (ho_id, f_id) VALUES (4, 4);

INSERT INTO HomeownerFeedbacks (ho_id, f_id) VALUES (5, 5);

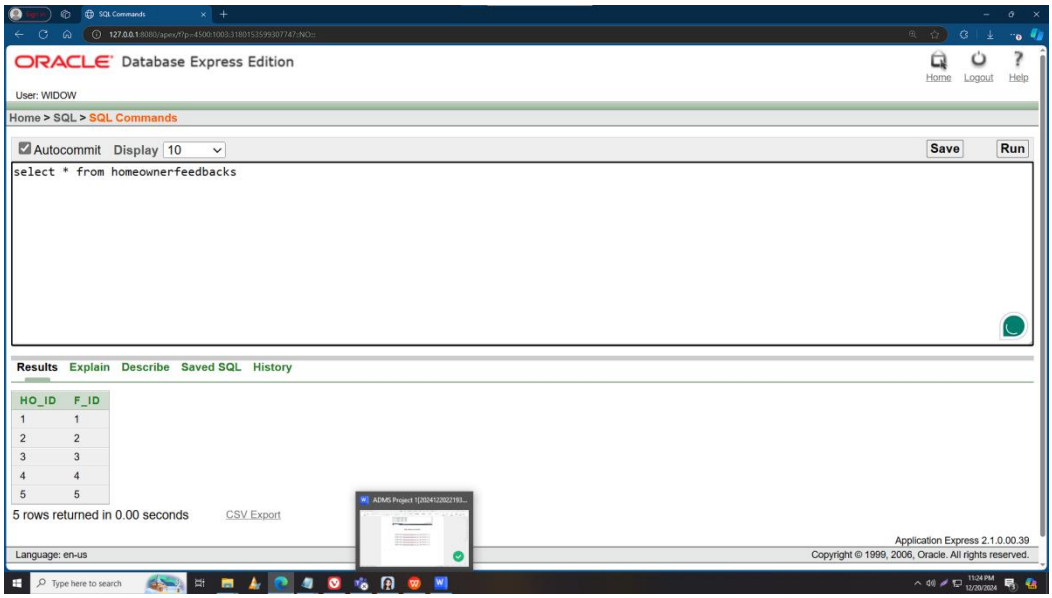


Table: Helper-Feedbacks

INSERT INTO HelperFeedbacks (he_id, f_id) VALUES (1, 1);

INSERT INTO HelperFeedbacks (he_id, f_id) VALUES (2, 2);

INSERT INTO HelperFeedbacks (he_id, f_id) VALUES (3, 3);

INSERT INTO HelperFeedbacks (he_id, f_id) VALUES (4, 4);

INSERT INTO HelperFeedbacks (he_id, f_id) VALUES (5, 5);

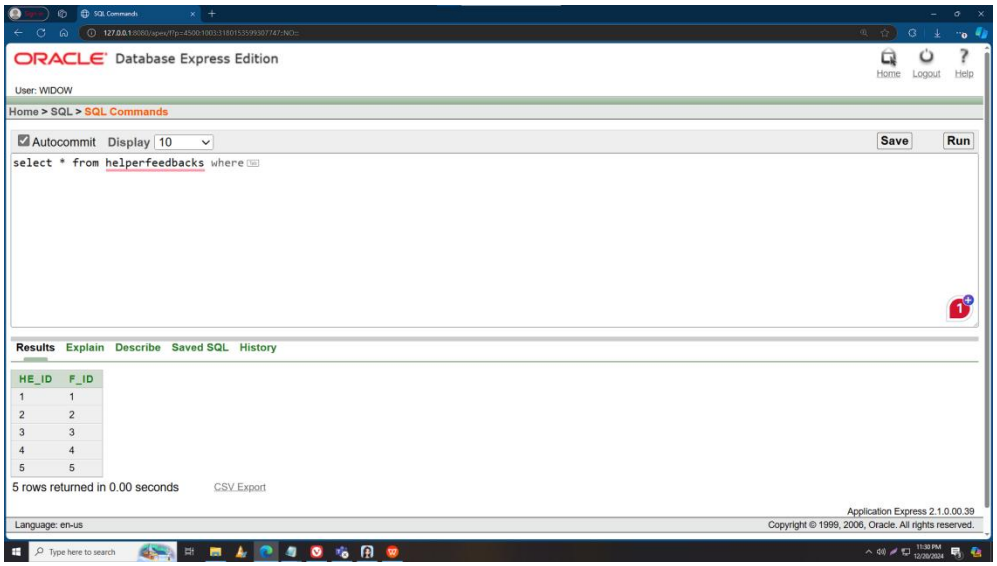


Table: Roles

INSERT INTO Roles (r_id, r_name, r_permission) VALUES (1, 'Manager', 'Full');

INSERT INTO Roles (r_id, r_name, r_permission) VALUES (2, 'Assistant', 'Partial');

INSERT INTO Roles (r_id, r_name, r_permission) VALUES (3, 'Clerk', 'Read-Only');

INSERT INTO Roles (r_id, r_name, r_permission) VALUES (4, 'Supervisor', 'Full');

INSERT INTO Roles (r_id, r_name, r_permission) VALUES (5, 'Admin', 'Full');

\

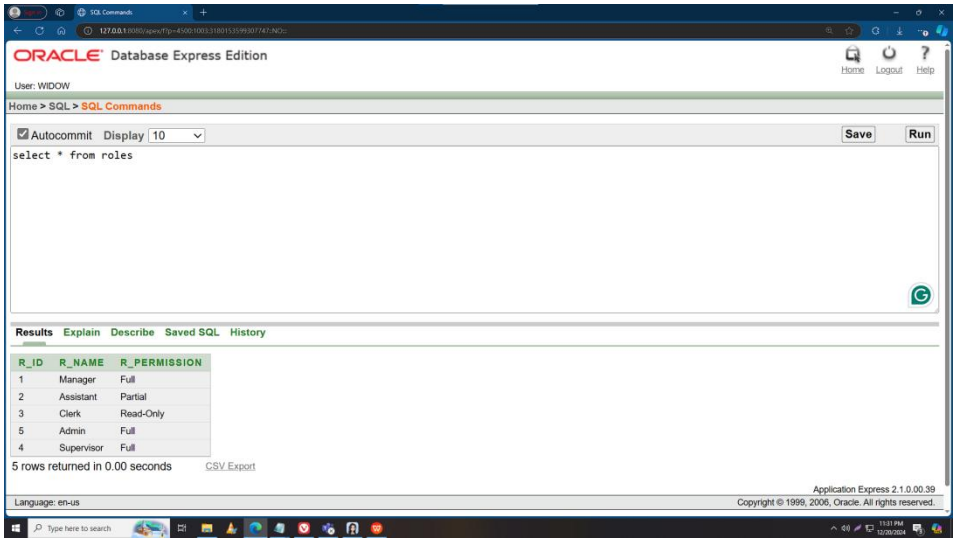


Table: Admins

INSERT INTO Admins (a_id, a_name, a_password, a_address_city, a_address_street, a_address_house, a_gender, a_phone, a_email, a_age)

VALUES (1, 'Oliver King', 'pass123', 'New York', 'Lexington Ave', 'Apt 10A', 'M', '1234509876', 'oliver.king@example.com', 40);

INSERT INTO Admins (a_id, a_name, a_password, a_address_city, a_address_street, a_address_house, a_gender, a_phone, a_email, a_age)

VALUES (2, 'Sophia Martinez', 'pass123', 'Los Angeles', 'Beverly Blvd', 'House 27', 'F', '2345608765', 'sophia.martinez@example.com', 35);

INSERT INTO Admins (a_id, a_name, a_password, a_address_city, a_address_street, a_address_house, a_gender, a_phone, a_email, a_age)

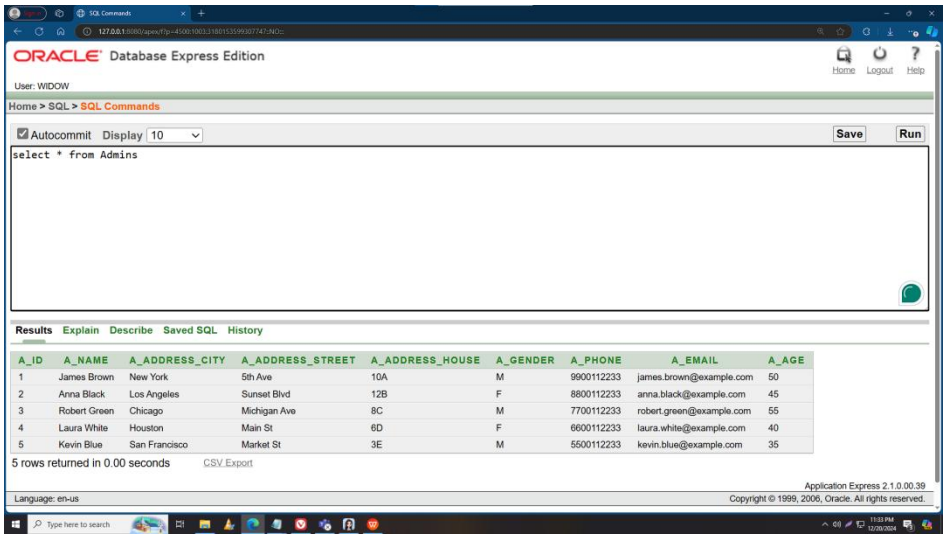
VALUES (3, 'Ethan Wright', 'pass123', 'Chicago', 'State St', 'Unit 9C', 'M', '3456707654', 'ethan.wright@example.com', 45);

INSERT INTO Admins (a_id, a_name, a_password, a_address_city, a_address_street, a_address_house, a_gender, a_phone, a_email, a_age)

VALUES (4, 'Ava Johnson', 'pass123', 'Houston', 'Westheimer Rd', 'Suite 18B', 'F', '4567806543', 'ava.johnson@example.com', 38);

INSERT INTO Admins (a_id, a_name, a_password, a_address_city, a_address_street, a_address_house, a_gender, a_phone, a_email, a_age)

VALUES (5, 'William Brown', 'pass123', 'San Francisco', 'Mission St', 'Loft 3D', 'M', '5678905432', 'william.brown@example.com', 50);



Tables: Admin-Roles

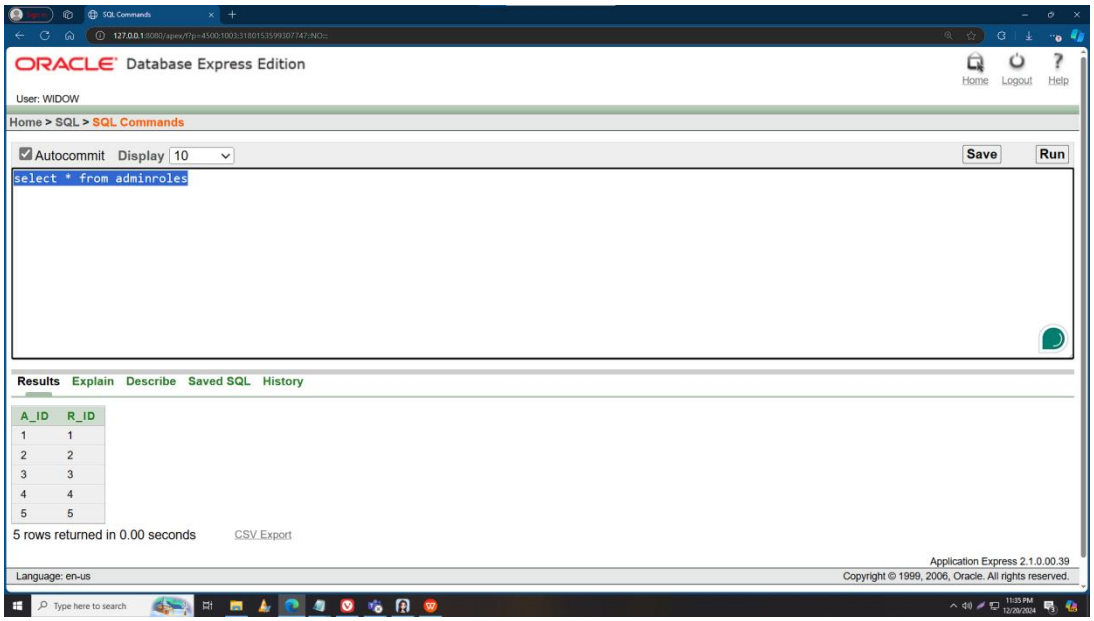
INSERT INTO AdminRoles (a_id, r_id) VALUES (1, 1);

INSERT INTO AdminRoles (a_id, r_id) VALUES (2, 2);

INSERT INTO AdminRoles (a_id, r_id) VALUES (3, 3);

INSERT INTO AdminRoles (a_id, r_id) VALUES (4, 4);

INSERT INTO AdminRoles (a_id, r_id) VALUES (5, 5);



Tables: Agency-Staff

INSERT INTO Agency_Staff (s_id, s_name, s_password, s_address_city, s_address_street, s_address_house, s_gender, s_email, s_salary, s_role, s_age)

VALUES (1, 'Liam Carter', 'pass123', 'New York', 'Wall Street', 'Apt 8A', 'M', 'liam.carter@example.com', 55000.00, 'Manager', 40);

INSERT INTO Agency_Staff (s_id, s_name, s_password, s_address_city, s_address_street, s_address_house, s_gender, s_email, s_salary, s_role, s_age)

VALUES (2, 'Emma Green', 'pass123', 'Los Angeles', 'Sunset Blvd', 'House 32', 'F', 'emma.green@example.com', 48000.00, 'Coordinator', 35);

INSERT INTO Agency_Staff (s_id, s_name, s_password, s_address_city, s_address_street, s_address_house, s_gender, s_email, s_salary, s_role, s_age)

VALUES (3, 'Noah Mitchell', 'pass123', 'Chicago', 'Lake Shore Dr', 'Unit 11C', 'M', 'noah.mitchell@example.com', 60000.00, 'Supervisor', 45);

INSERT INTO Agency_Staff (s_id, s_name, s_password, s_address_city, s_address_street, s_address_house, s_gender, s_email, s_salary, s_role, s_age)

VALUES (4, 'Olivia Adams', 'pass123', 'Houston', 'Main St', 'Suite 5B', 'F', 'olivia.adams@example.com', 52000.00, 'HR Specialist', 38);

INSERT INTO Agency_Staff (s_id, s_name, s_password, s_address_city, s_address_street, s_address_house, s_gender, s_email, s_salary, s_role, s_age)

VALUES (5, 'William Scott', 'pass123', 'San Francisco', 'Broadway St', 'Loft 2D', 'M', 'william.scott@example.com', 58000.00, 'Accountant', 50);

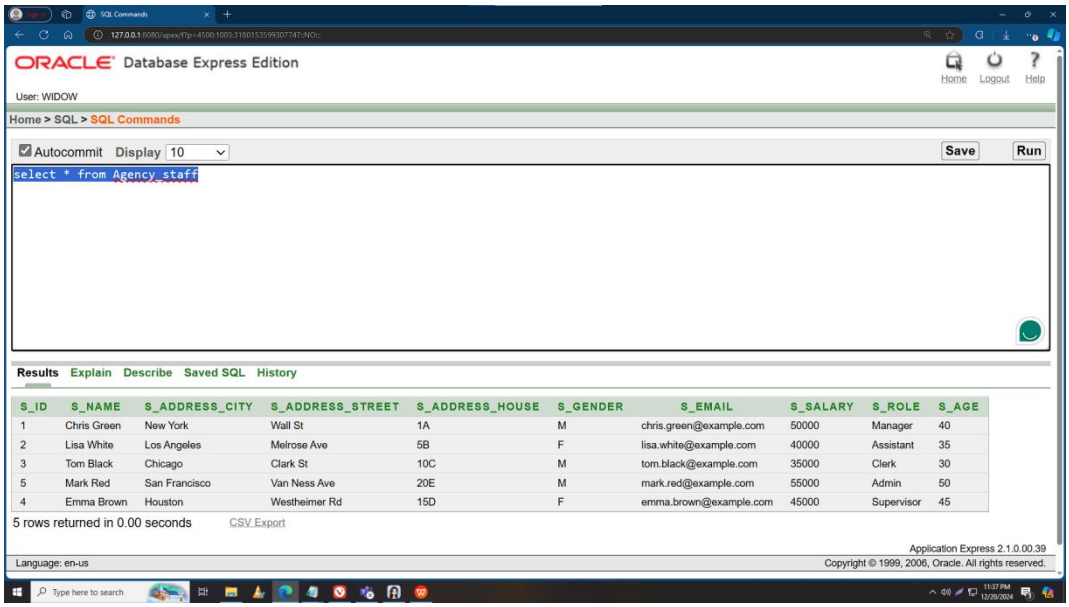


Table: Reports

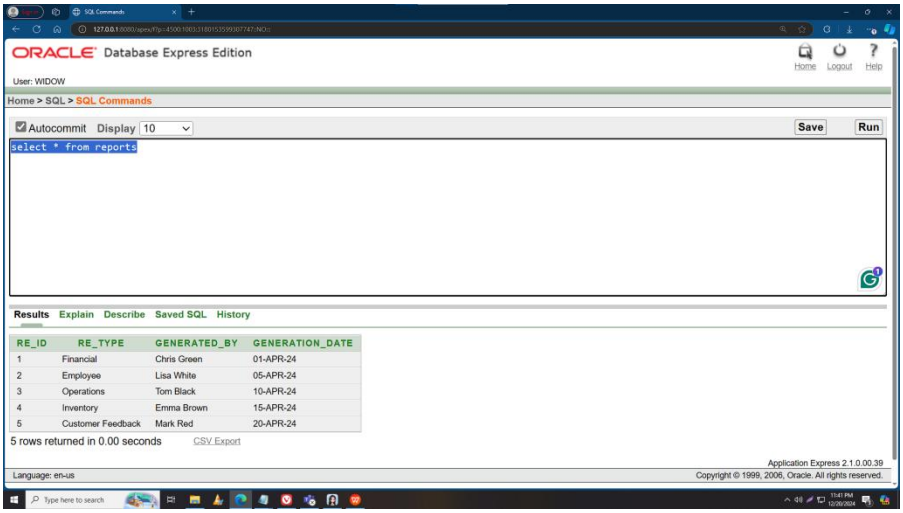
INSERT INTO Reports (re_id, re_type, generated_by, generation_date) VALUES (1, 'Financial', 'Chris Green', TO_DATE('2024-04-01', 'YYYY-MM-DD'));

INSERT INTO Reports (re_id, re_type, generated_by, generation_date) VALUES (2, 'Employee', 'Lisa White', TO_DATE('2024-04-05', 'YYYY-MM-DD'));

INSERT INTO Reports (re_id, re_type, generated_by, generation_date) VALUES (3, 'Operations', 'Tom Black', TO_DATE('2024-04-10', 'YYYY-MM-DD'));

INSERT INTO Reports (re_id, re_type, generated_by, generation_date) VALUES (4, 'Inventory', 'Emma Brown', TO_DATE('2024-04-15', 'YYYY-MM-DD'));

INSERT INTO Reports (re_id, re_type, generated_by, generation_date) VALUES (5, 'Customer Feedback', 'Mark Red', TO_DATE('2024-04-20', 'YYYY-MM-DD'));



Reports: Staff Report

INSERT INTO StaffReports (s_id, re_id) VALUES (1, 1);

INSERT INTO StaffReports (s_id, re_id) VALUES (2, 2);

INSERT INTO StaffReports (s_id, re_id) VALUES (3, 3);

INSERT INTO StaffReports (s_id, re_id) VALUES (4, 4);

INSERT INTO StaffReports (s_id, re_id) VALUES (5, 5);

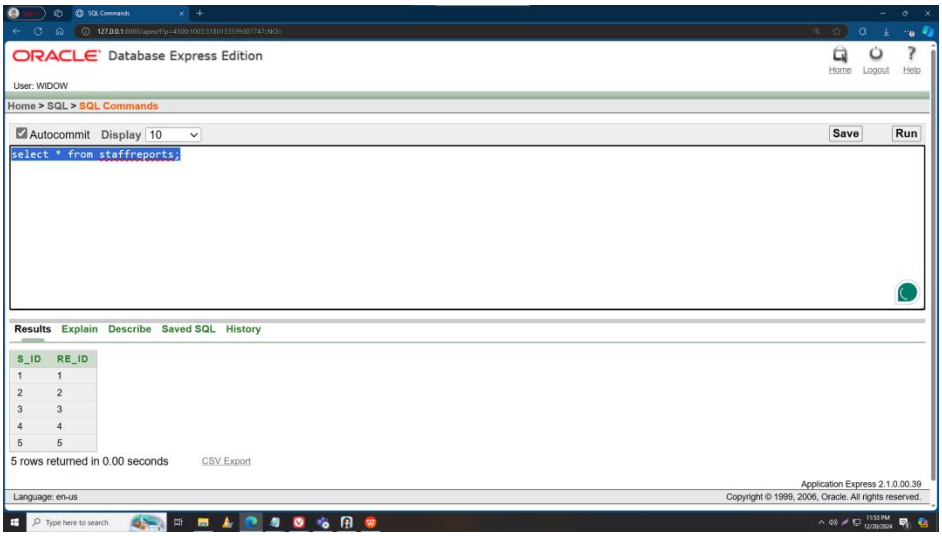


Table: Staff-Admin

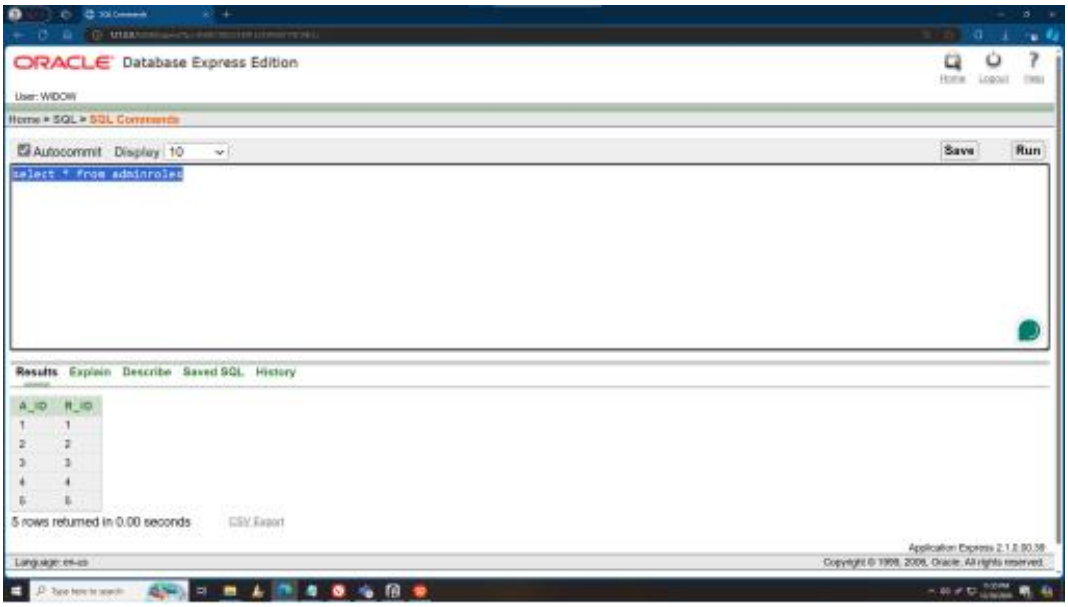
INSERT INTO Hires (a_id, s_id) VALUES (1, 101);

INSERT INTO Hires (a_id, s_id) VALUES (2, 102);

INSERT INTO Hires (a_id, s_id) VALUES (3, 103);

INSERT INTO Hires (a_id, s_id) VALUES (4, 104);

INSERT INTO Hires (a_id, s_id) VALUES (5, 105);

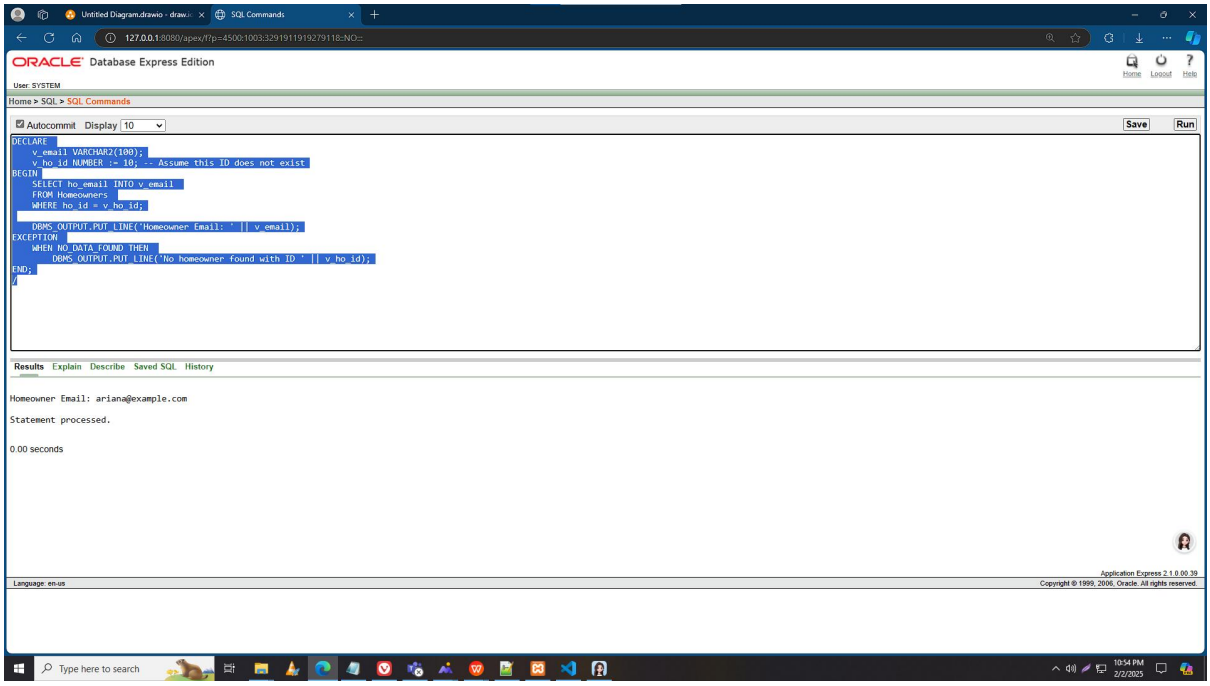


Basic Query Writing

● Exception Handling

1. Write a PL/SQL block that retrieves a homeowner's email based on a given homeowner ID. If the ID does not exist, handle the NO_DATA_FOUND exception by displaying a message.

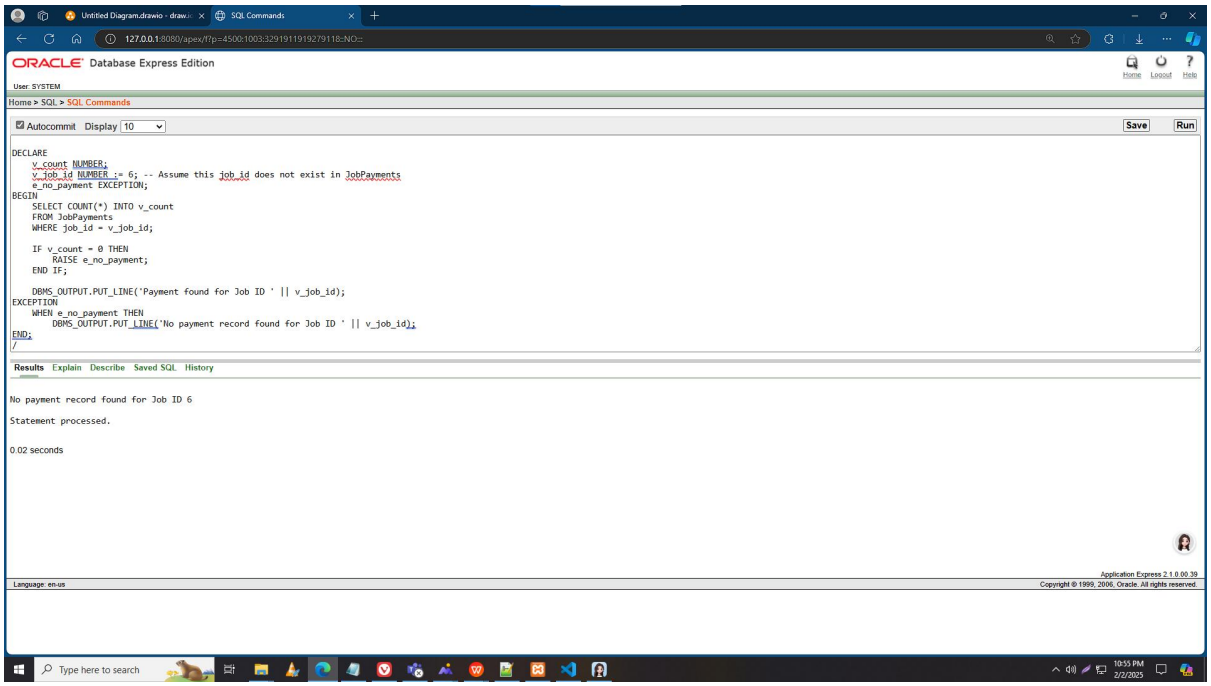
```
DECLARE
    v_email VARCHAR2(100);
    v_ho_id NUMBER := 10; -- Assume this ID does not exist
BEGIN
    SELECT ho_email INTO v_email
    FROM Homeowners
    WHERE ho_id = v_ho_id;

    DBMS_OUTPUT.PUT_LINE('Homeowner Email: ' || v_email);
EXCEPTION
    WHEN NO_DATA_FOUND THEN
        DBMS_OUTPUT.PUT_LINE('No homeowner found with ID ' ||
v_ho_id);
END;
/
```



2. Create a PL/SQL block that checks if a payment exists for a given job. If not, raise a user-defined exception.

```
DECLARE
    v_count NUMBER;
    v_job_id NUMBER := 6; -- Assume this job_id does not exist in JobPayments
    e_no_payment EXCEPTION;
BEGIN
    SELECT COUNT(*) INTO v_count
    FROM JobPayments
    WHERE job_id = v_job_id;

    IF v_count = 0 THEN
        RAISE e_no_payment;
    END IF;

    DBMS_OUTPUT.PUT_LINE('Payment found for Job ID ' || v_job_id);
```

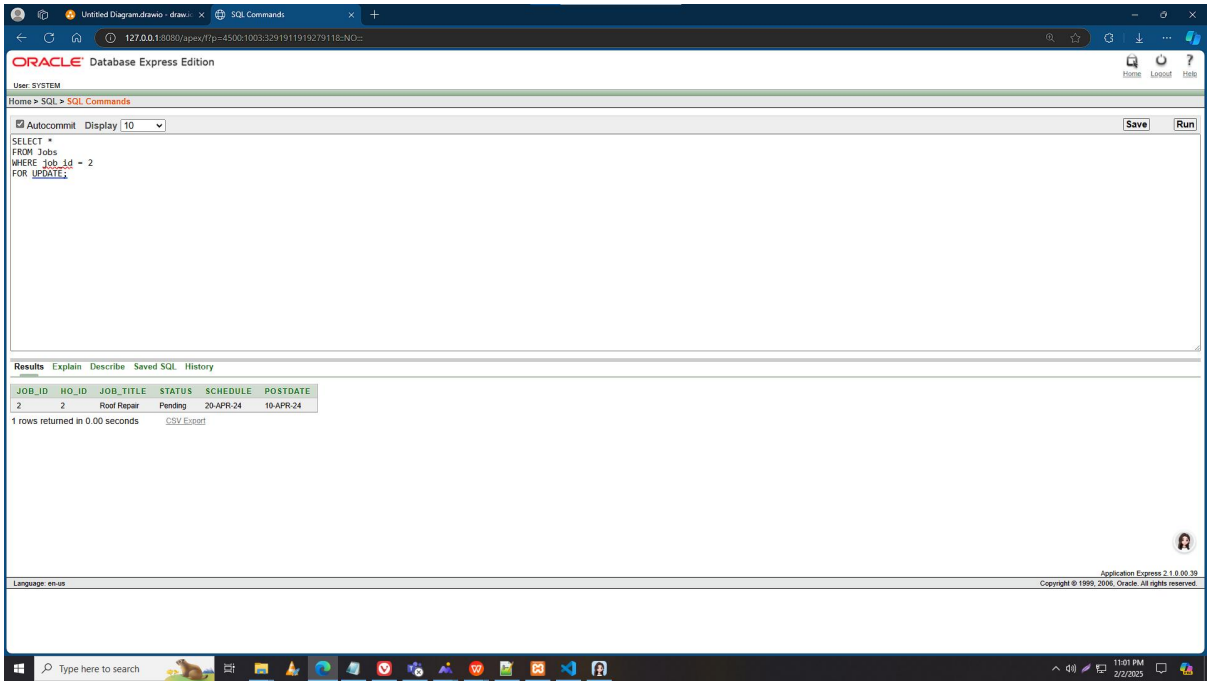
```
        EXCEPTION
            WHEN e_no_payment THEN
                DBMS_OUTPUT.PUT_LINE('No payment record found for Job ID
' || v_job_id);
        END;
/
```



- **Implicit Locking**

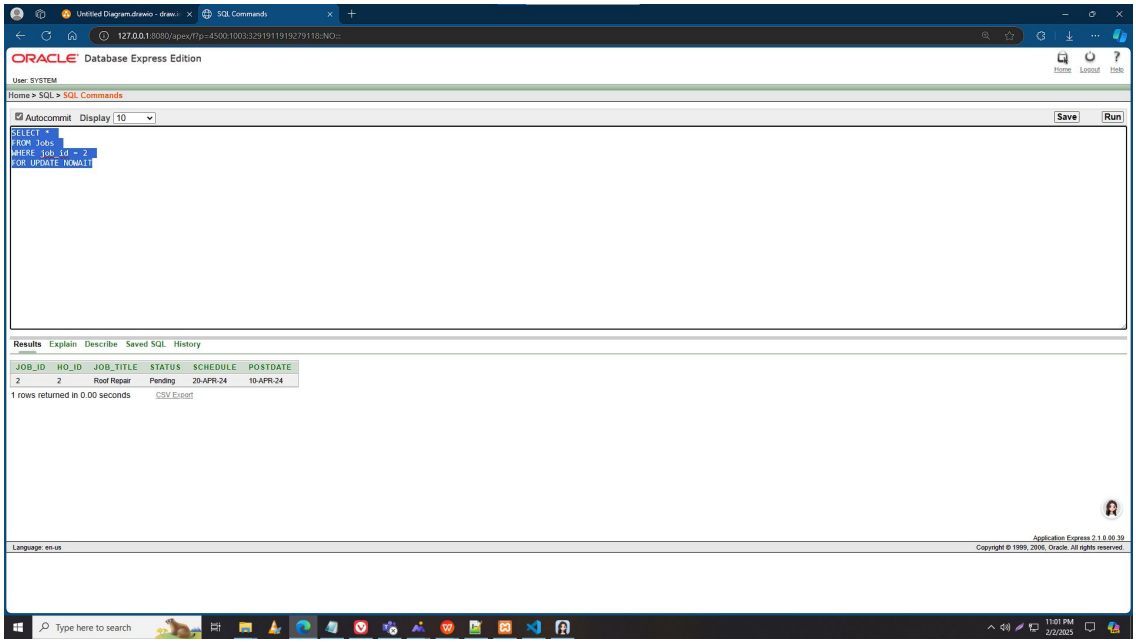
1. Write a query that selects a job record for update and prevents other sessions from modifying it.

```
SELECT *
FROM Jobs
WHERE job_id = 2
FOR UPDATE;
```



2. Write a query that tries to lock a Jobs row but fails immediately if another session holds a lock.

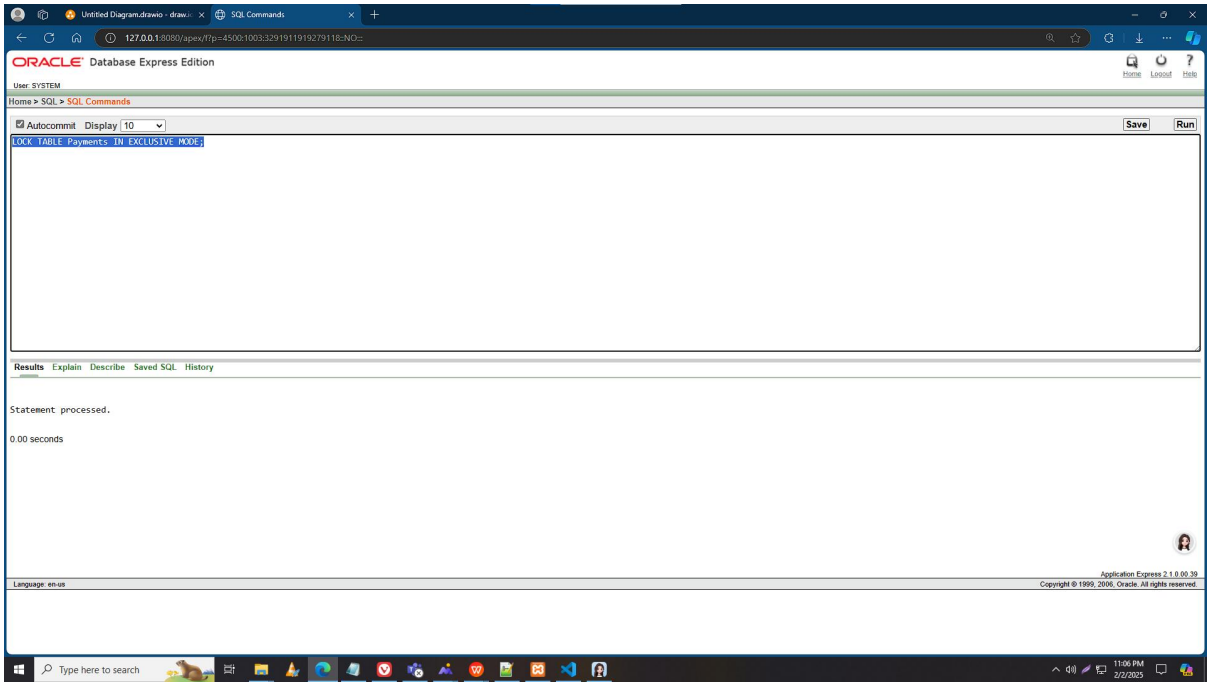
```
SELECT *
FROM Jobs
WHERE job_id = 2
FOR UPDATE NOWAIT
```



● Explicit Locking

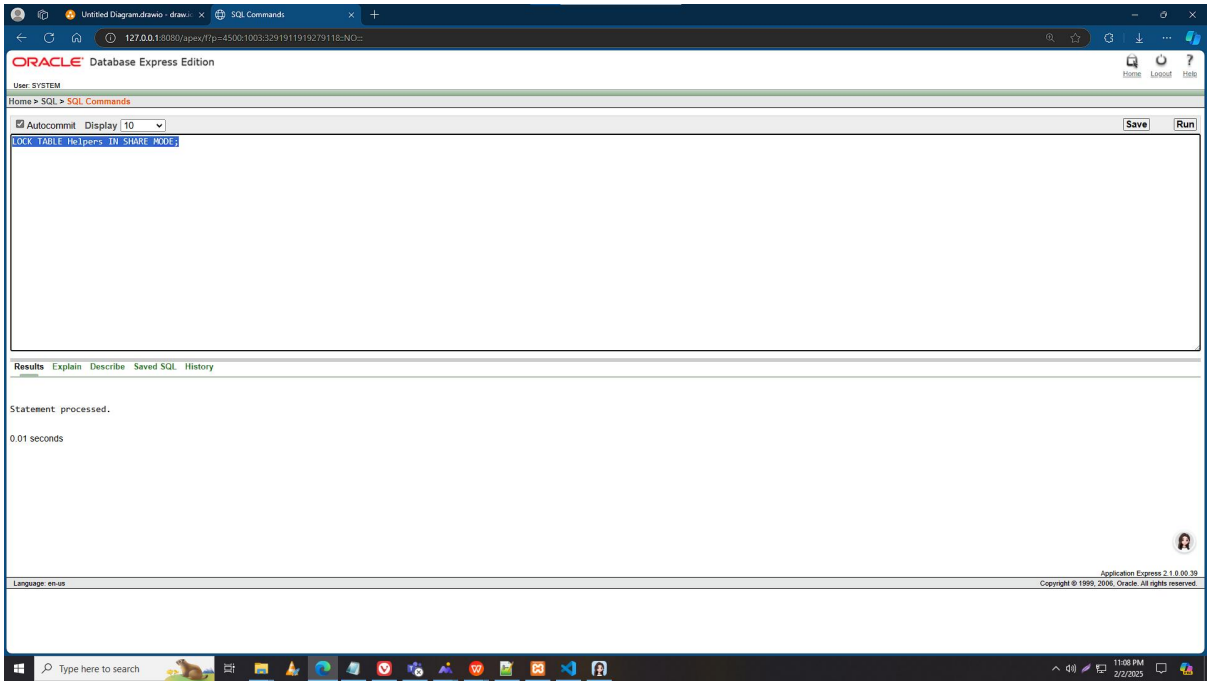
1. Write a query to lock the entire Payments table so only the current session can modify it.

LOCK TABLE Payments IN EXCLUSIVE MODE;



2. Write a query to allow only SELECT queries on the Helpers table but prevent updates or deletions.

LOCK TABLE Helpers IN SHARE MODE;



● Relational Algebra

1. Express a relational algebra query to find all homeowners from New York.

$$\sigma_{ho\_address\_city='NewYork'}(Homeowners)$$

2. Retrieve job titles where the job status is 'Pending'.

$$\pi_{job\_title}(\sigma_{status='Pending'}(Jobs))$$

3. Find homeowners who have both posted a job and made a payment.

$$\pi_{ho\_id}(Jobs) \cap \pi_{ho\_id}(JobPayments \bowtie Payments)$$

4. Retrieve the names of homeowners who have provided feedback.

$$\pi_{ho\_name}(Homeowners \bowtie HomeownerFeedbacks)$$

5. Find the total amount paid for each job using an aggregation operation.

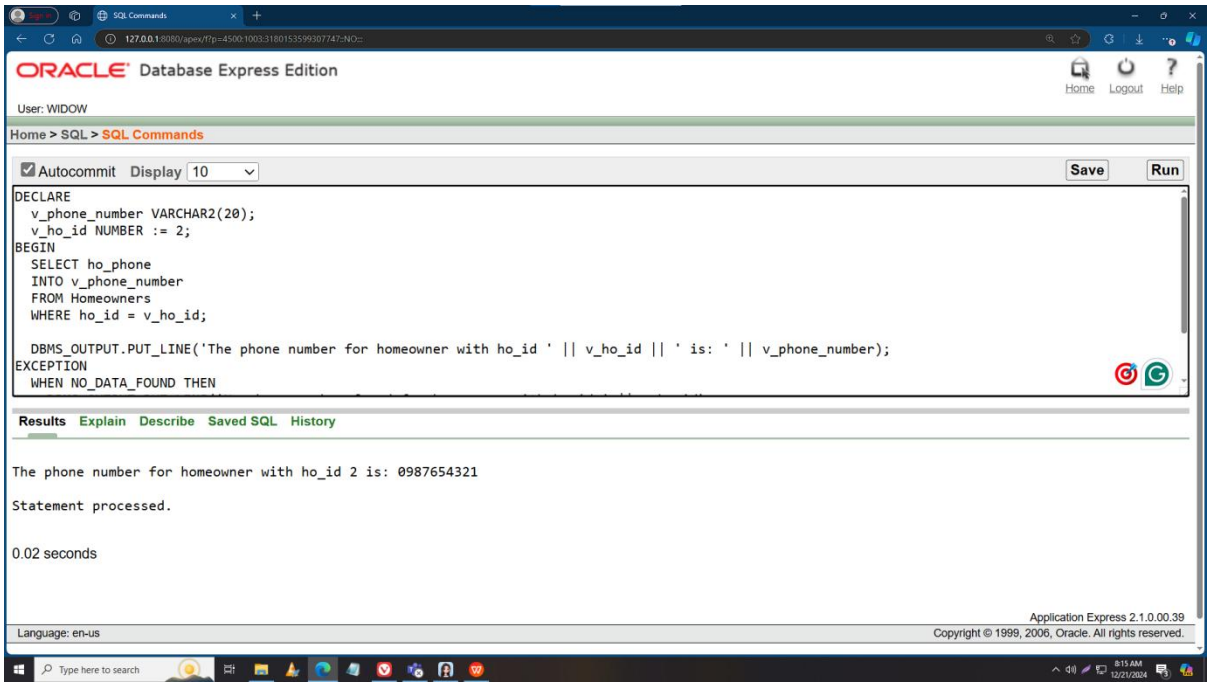
JobPayments⋈ PaymentsGjob_id,SUM(p_amount) (JobPayments⋈ Payments)

(Midterm Part)

● Variables

1. Write a PL/SQL block to retrieve the phone number of a specific homeowner (ho_id = 2) and display it. Use a PL/SQL variable to store the phone number.

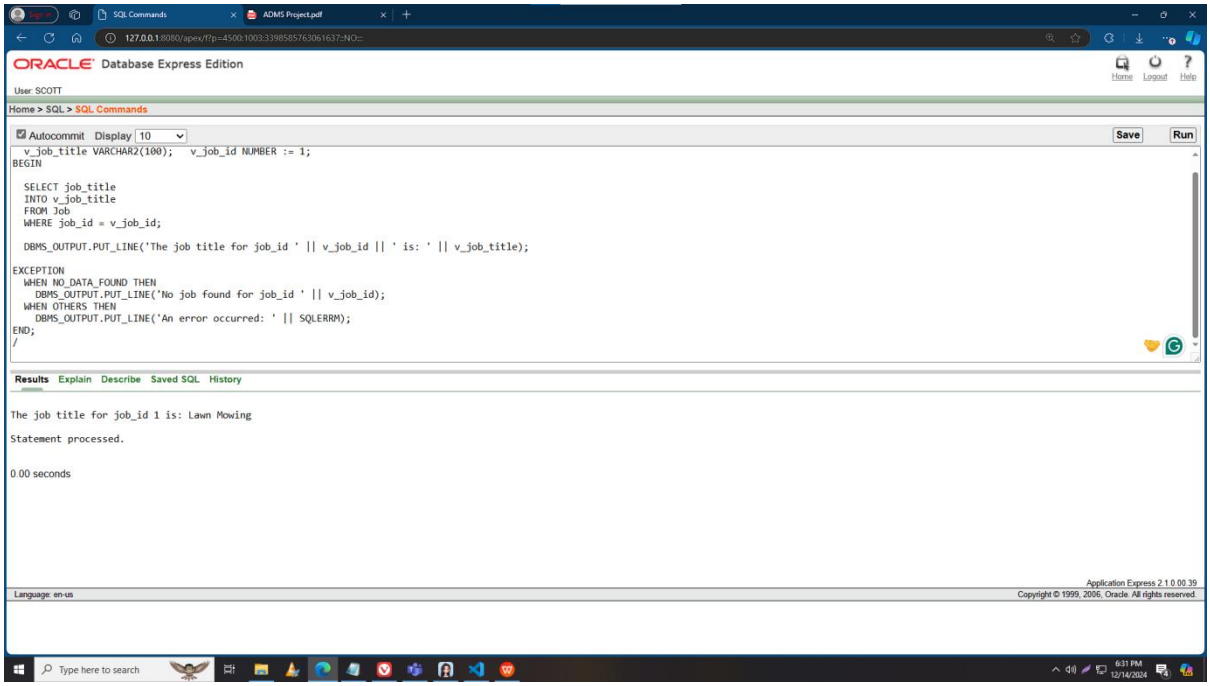
```
DECLARE
   v_phone_number VARCHAR2(20);
   v_ho_id NUMBER := 2;
BEGIN
   SELECT ho_phone
   INTO v_phone_number
   FROM Homeowner
   WHERE ho_id = v_ho_id;

   DBMS_OUTPUT.PUT_LINE('The phone number for homeowner with ho_id ' ||
v_ho_id || ' is: ' || v_phone_number);
EXCEPTION
   WHEN NO_DATA_FOUND THEN
      DBMS_OUTPUT.PUT_LINE('No phone number found for homeowner with
ho_id ' || v_ho_id);
   WHEN OTHERS THEN
      DBMS_OUTPUT.PUT_LINE('An error occurred: ' || SQLERRM);
END;
```



2. Write a PL/SQL block to retrieve the job title for a specific job (job_id = 1) and display it. Use a PL/SQL variable to store the job title.

```
DECLARE
   v_job_title VARCHAR2(100);      v_job_id NUMBER := 1;
BEGIN

   SELECT job_title
   INTO v_job_title
   FROM Job
   WHERE job_id = v_job_id;

   DBMS_OUTPUT.PUT_LINE('The job title for job_id ' || v_job_id || ' is: ' || v_job_title);

EXCEPTION
   WHEN NO_DATA_FOUND THEN
```

```
    DBMS_OUTPUT.PUT_LINE('No job found for job_id ' || v_job_id);
  WHEN OTHERS THEN
    DBMS_OUTPUT.PUT_LINE('An error occurred: ' || SQLERRM);
END;
```

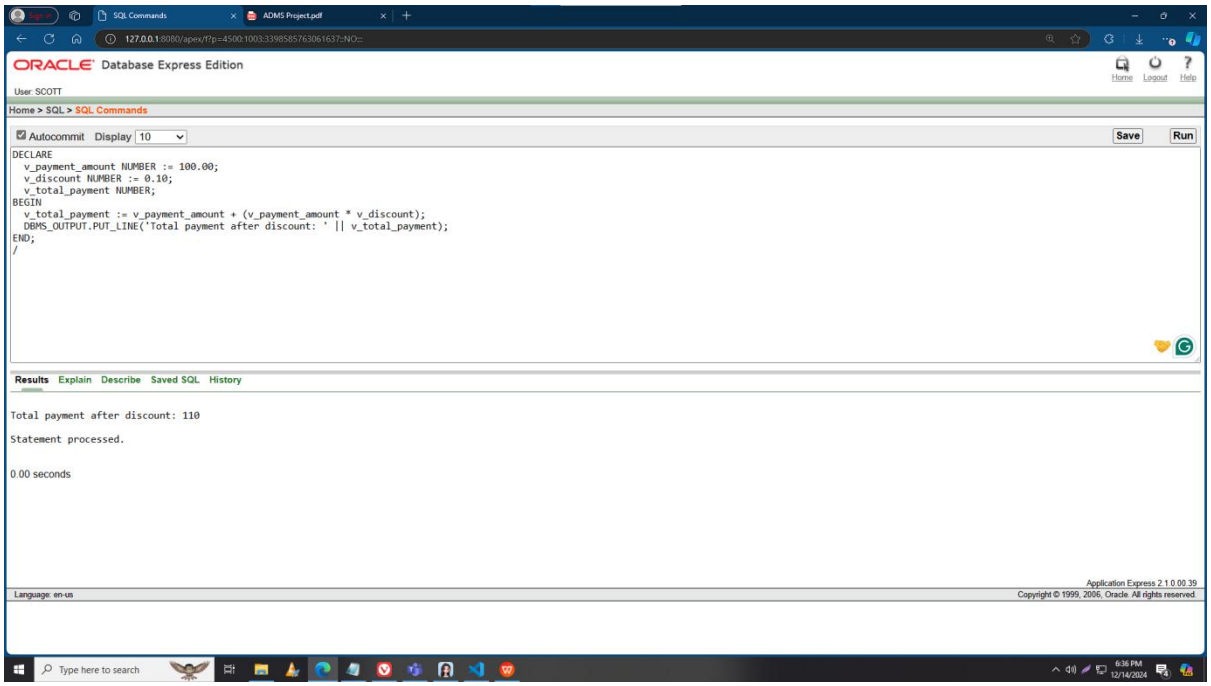


● Opeartors

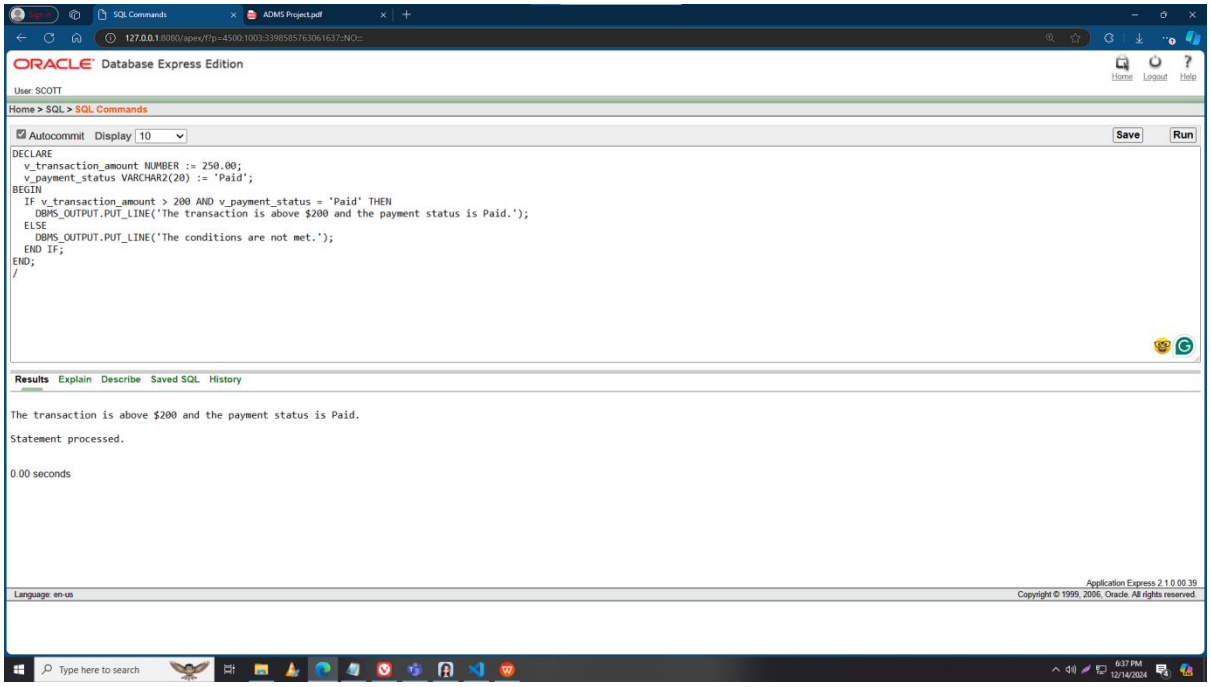
1. Write a PL/SQL block that uses arithmetic operators to calculate the total payment for a specific homeowner. The total payment should be the sum of the payment amount and a 10% discount. Use variables to store the payment amount and the discount.

```
DECLARE
  v_payment_amount NUMBER := 100.00;
  v_discount NUMBER := 0.10;
  v_total_payment NUMBER;
BEGIN
  v_total_payment := v_payment_amount + (v_payment_amount * v_discount);
```

```
    DBMS_OUTPUT.PUT_LINE('Total payment after discount: ' || v_total_payment);
END;
```



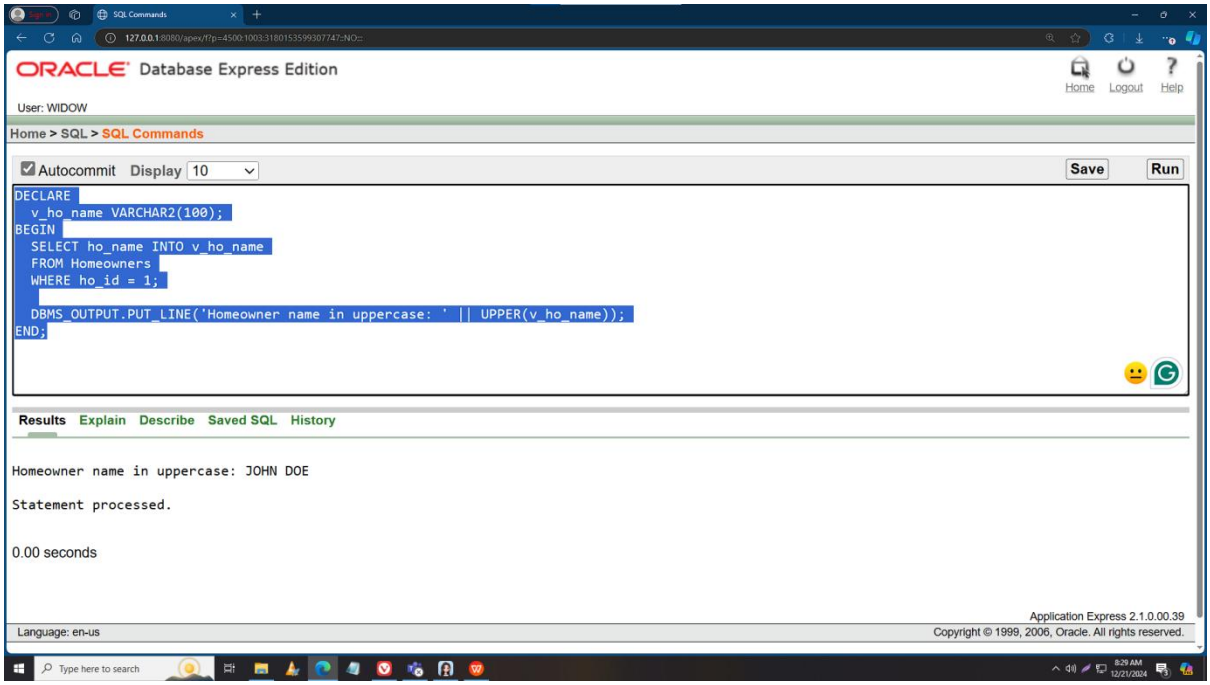
2. Write a PL/SQL block that uses logical operators to check if a homeowner's transaction (transaction_id = 1) is above $200 and the payment status is 'Paid'. Display a message indicating whether both conditions are true.



● Single_Row Function

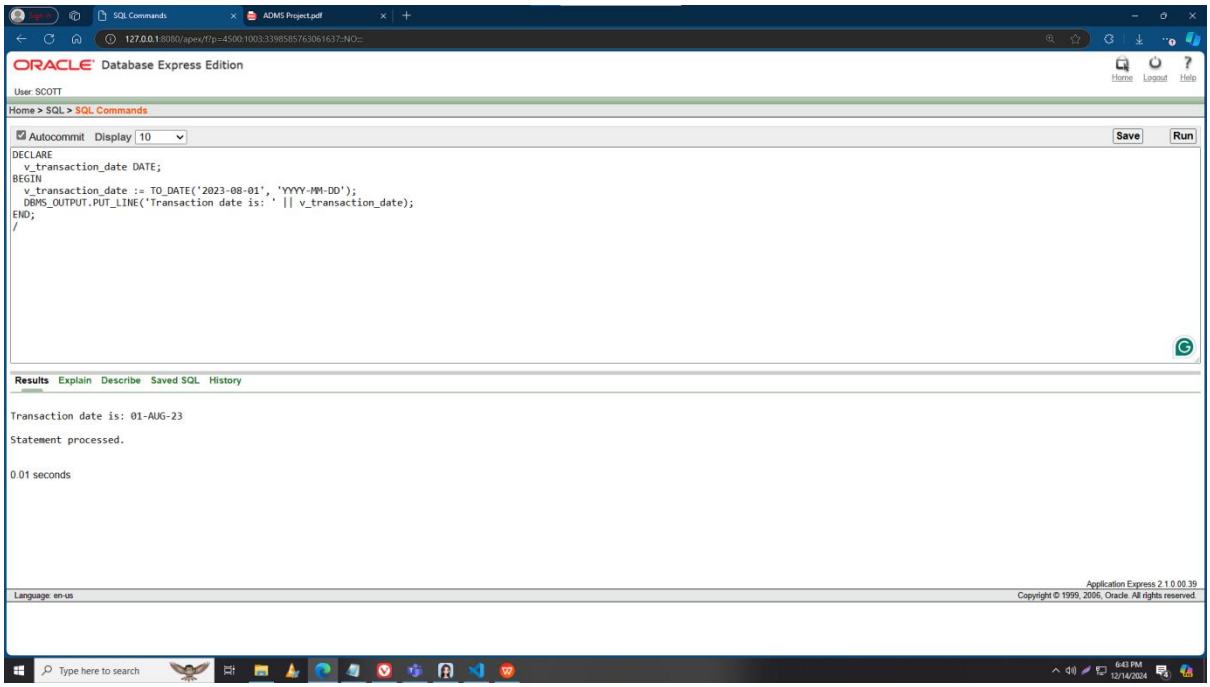
1. Write a PL/SQL block that uses the UPPER() function to convert the homeowner's name (ho_name) for a given ho_id = 101 to uppercase and displays the result.

```
DECLARE
   v_ho_name VARCHAR2(100);
BEGIN
   SELECT ho_name INTO v_ho_name
   FROM Homeowners
   WHERE ho_id = 1;

   DBMS_OUTPUT.PUT_LINE('Homeowner name in uppercase: ' || UPPER(v_ho_name));
END;
```



2. Write a PL/SQL block that uses the TO_DATE() function to convert a string ('2023-08-01') into a date format and displays the result.

```
DECLARE
   v_transaction_date DATE;
BEGIN
   v_transaction_date := TO_DATE('2023-08-01', 'YYYY-MM-DD');
   DBMS_OUTPUT.PUT_LINE('Transaction date is: ' || v_transaction_date);
END;
```

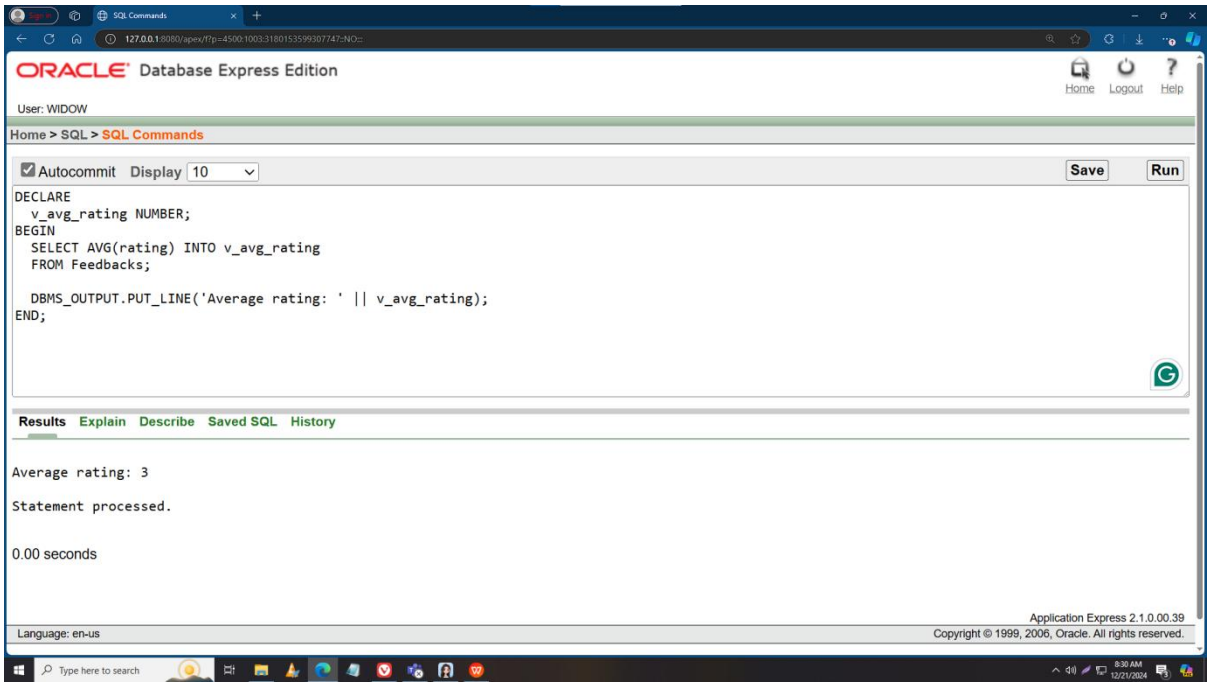


● Group Function

1. **Write a PL/SQL block that calculates the average rating from the Feedback table and displays the result.**

```
DECLARE
   v_avg_rating NUMBER;
```

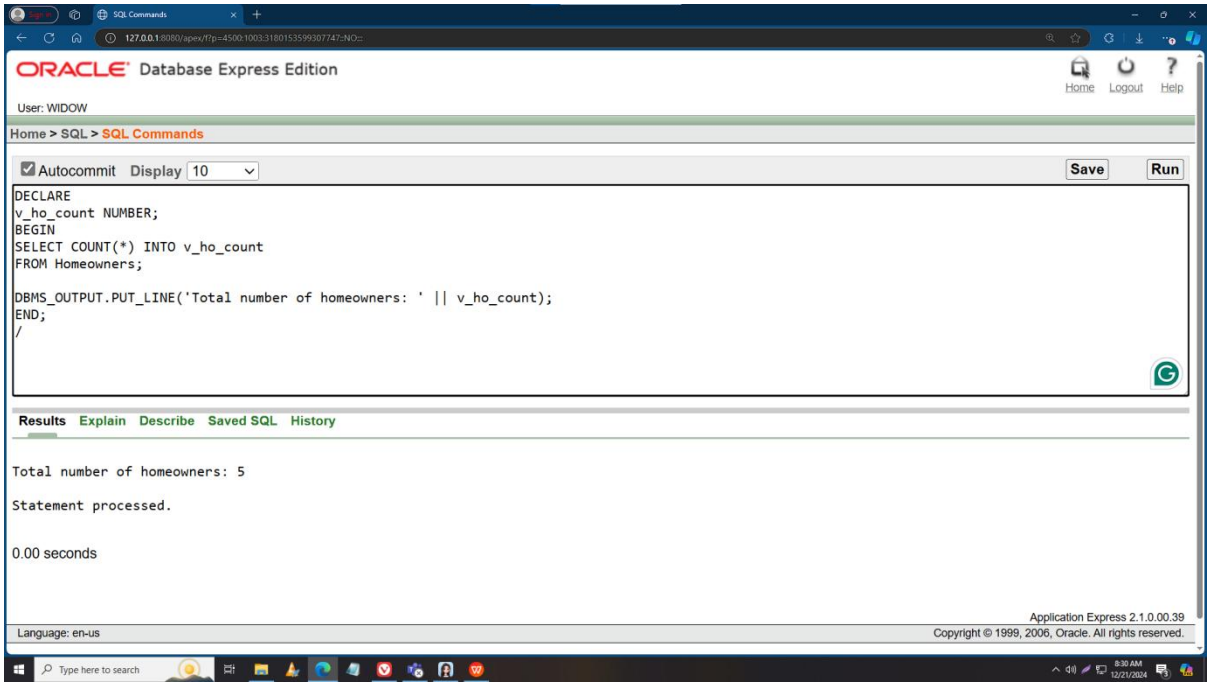
```
BEGIN
    SELECT AVG(rating) INTO v_avg_rating
    FROM Feedbacks;

    DBMS_OUTPUT.PUT_LINE('Average rating: ' || v_avg_rating);
END;
```



2. Write a PL/SQL block that counts the number of Homeowner entries in the Homeowner table and displays the result.

```
DECLARE
v_ho_count NUMBER;
BEGIN
SELECT COUNT(*) INTO v_ho_count
FROM Homeowners;
```

```
DBMS_OUTPUT.PUT_LINE('Total number of homeowners: ' || v_ho_count);

END;

/
```

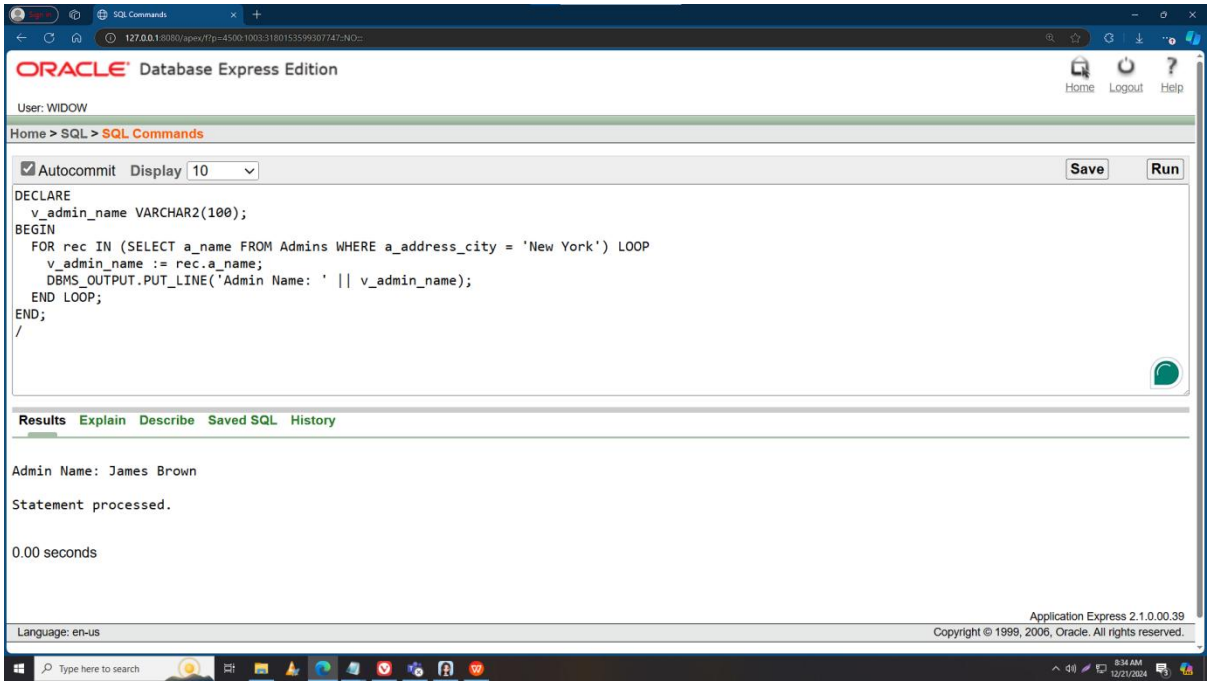


Loops

1. Write a PL/SQL block that uses a FOR loop to display the names of all admins from the Admins table who live in "New York".

```
 DECLARE
   v_admin_name VARCHAR2(100);
BEGIN
   FOR rec IN (SELECT a_name FROM Admins WHERE a_address_city = 'New York')
LOOP
      v_admin_name := rec.a_name;
      DBMS_OUTPUT.PUT_LINE('Admin Name: ' || v_admin_name);
   END LOOP;
END;
```

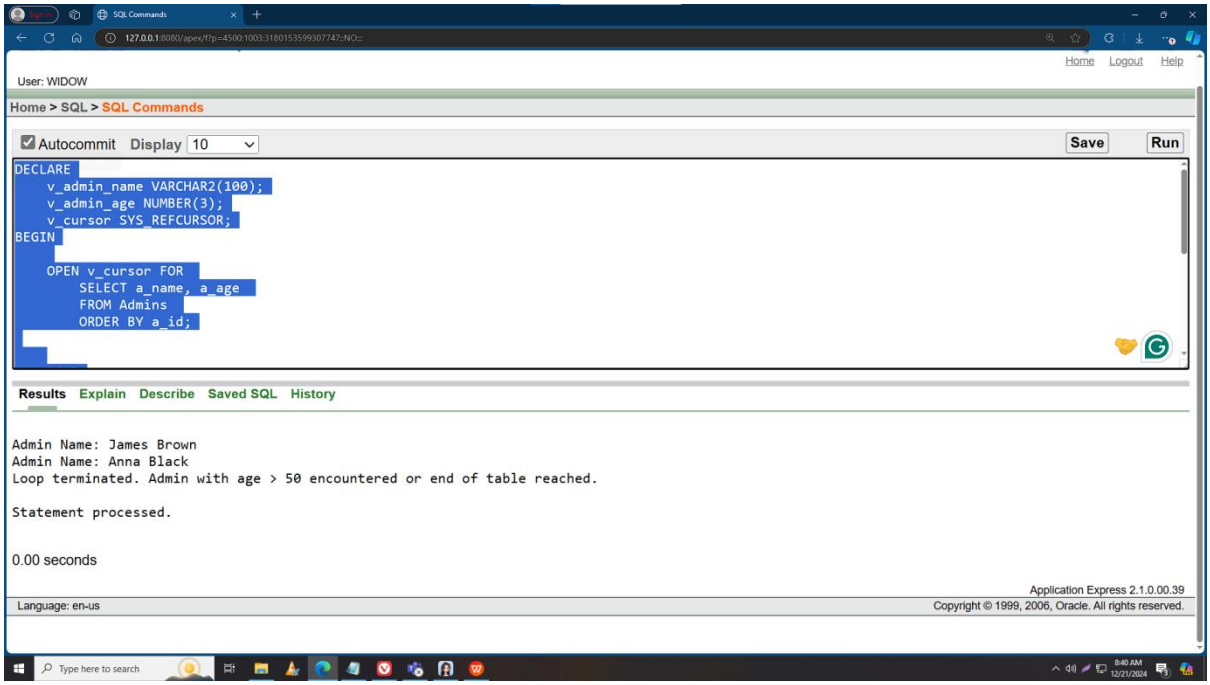
/



2. Write a PL/SQL block that uses a WHILE loop to display the names of admins from the Admins table until an admin with an age greater than 50 is found.

```
DECLARE
    v_admin_name VARCHAR2(100);
    v_admin_age NUMBER(3);
    v_cursor SYS_REFCURSOR;
BEGIN

    OPEN v_cursor FOR
        SELECT a_name, a_age
        FROM Admins
        ORDER BY a_id;


    LOOP
        FETCH v_cursor INTO v_admin_name, v_admin_age;
```

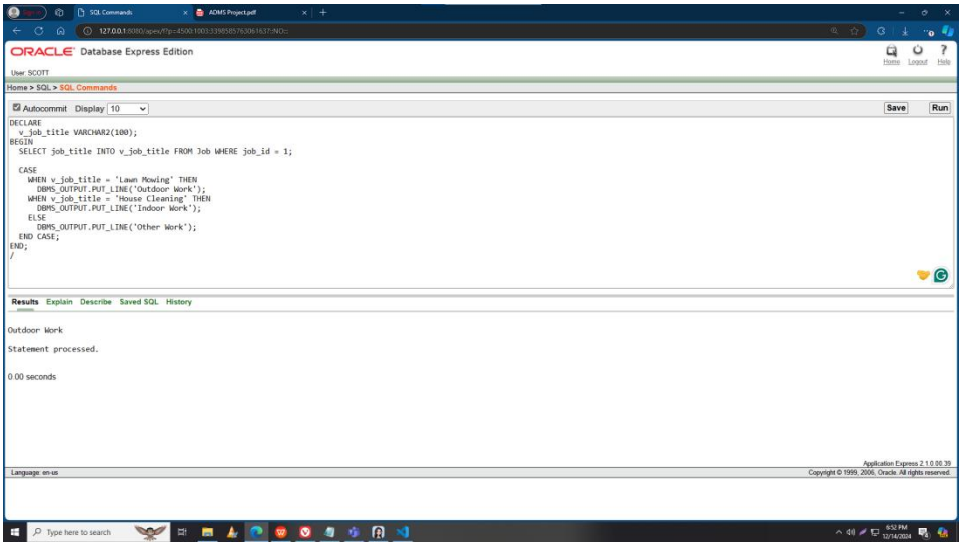
```
        EXIT WHEN v_cursor%NOTFOUND OR v_admin_age > 50;

        DBMS_OUTPUT.PUT_LINE('Admin Name: ' || v_admin_name);
    END LOOP;

    CLOSE v_cursor;

    DBMS_OUTPUT.PUT_LINE('Loop terminated. Admin with age > 50 encountered or end of table reached.');
END;
/
```



● Conditional Statement

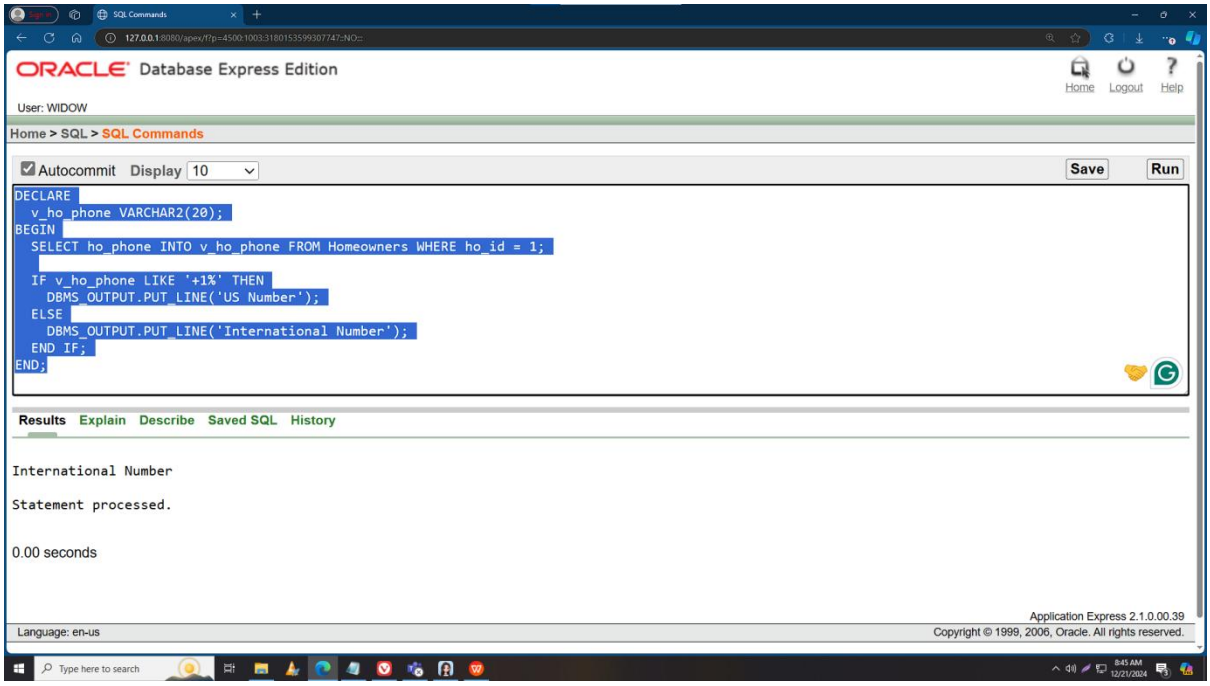
3. Write a PL/SQL block that uses a CASE statement to check the job_title in the Job table. If the job_title is 'Lawn Mowing', display "Outdoor Work". If the job_title is 'House Cleaning', display "Indoor Work". Otherwise, display "Other Work".

```
DECLARE
   v_job_title VARCHAR2(100);
BEGIN
   SELECT job_title INTO v_job_title FROM Job WHERE job_id = 1;

   CASE
      WHEN v_job_title = 'Lawn Mowing' THEN
         DBMS_OUTPUT.PUT_LINE('Outdoor Work');
      WHEN v_job_title = 'House Cleaning' THEN
         DBMS_OUTPUT.PUT_LINE('Indoor Work');
      ELSE
         DBMS_OUTPUT.PUT_LINE('Other Work');
   END CASE;
END;
```



1. Write a PL/SQL block that uses an IF-ELSE statement to check if a homeowner's ho_phone number starts with "+1". If it does, display "US Number", otherwise display "International Number".

```
DECLARE
   v_ho_phone VARCHAR2(20);
BEGIN
   SELECT ho_phone INTO v_ho_phone FROM Homeowners WHERE ho_id = 1;

   IF v_ho_phone LIKE '+1%' THEN
      DBMS_OUTPUT.PUT_LINE('US Number');
   ELSE
      DBMS_OUTPUT.PUT_LINE('International Number');
   END IF;
END;
```

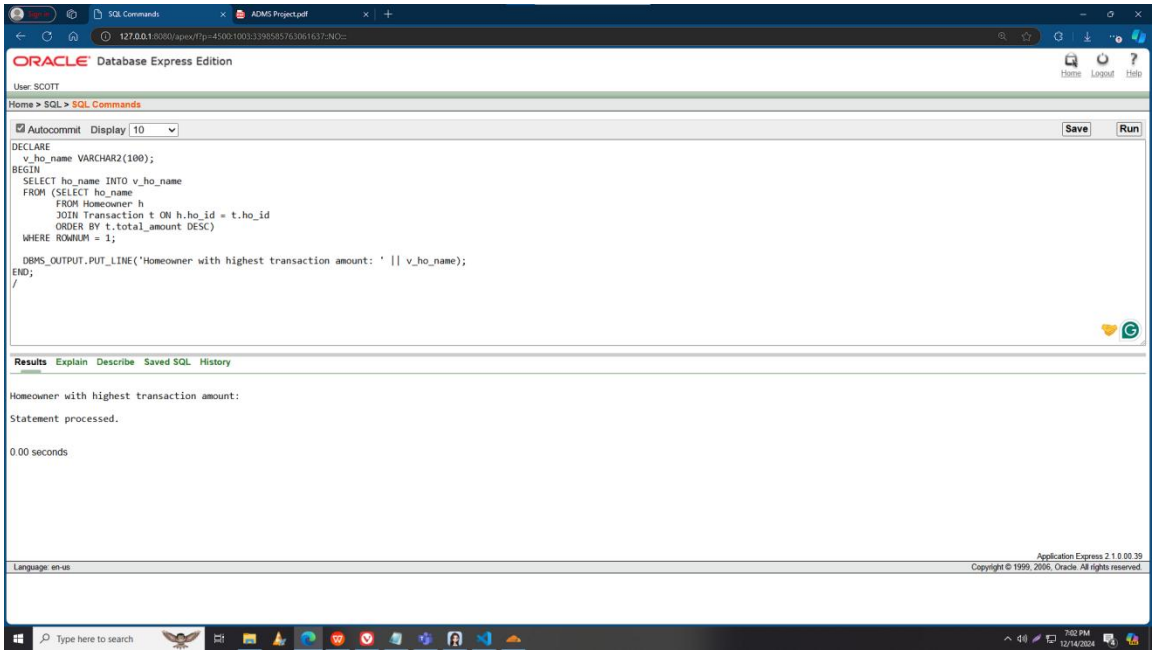


● Subquaeries

1. Write a PL/SQL block that uses a subquery to find the ho_name of the homeowner who has the highest total_amount in the Transaction table.

```
DECLARE
   v_ho_name VARCHAR2(100);
BEGIN
   SELECT ho_name INTO v_ho_name
   FROM (SELECT ho_name
            FROM Homeowner h
            JOIN Transaction t ON h.ho_id = t.ho_id
```

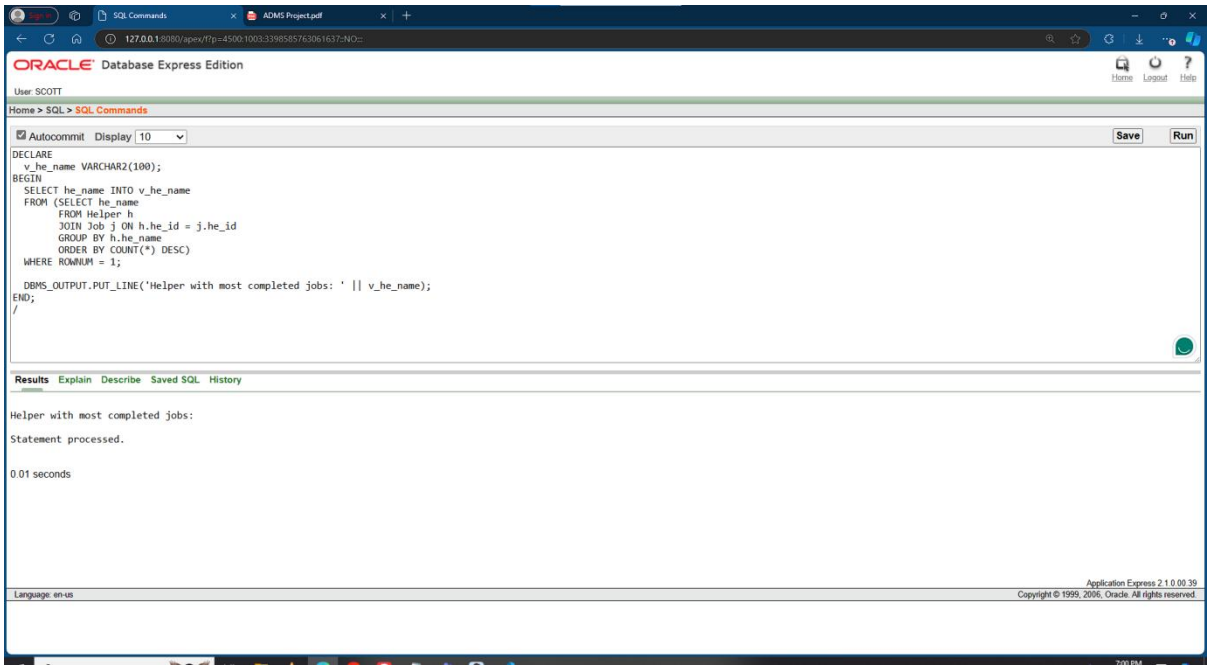
```
        ORDER BY t.total_amount DESC)
    WHERE ROWNUM = 1;


    DBMS_OUTPUT.PUT_LINE('Homeowner with highest transaction amount: ' ||
v_ho_name);
END;
/
```



2. **Write a PL/SQL block that uses a subquery to find the he_name of the helper who has completed the most number of jobs in the Job table.**

```
DECLARE
    v_he_name VARCHAR2(100);
BEGIN
    SELECT he_name INTO v_he_name
    FROM (SELECT he_name
            FROM Helper h
            JOIN Job j ON h.he_id = j.he_id
            GROUP BY h.he_name
```

```
        ORDER BY COUNT(*) DESC)
    WHERE ROWNUM = 1;

    DBMS_OUTPUT.PUT_LINE('Helper with most completed jobs: ' || v_he_name);
END;
```

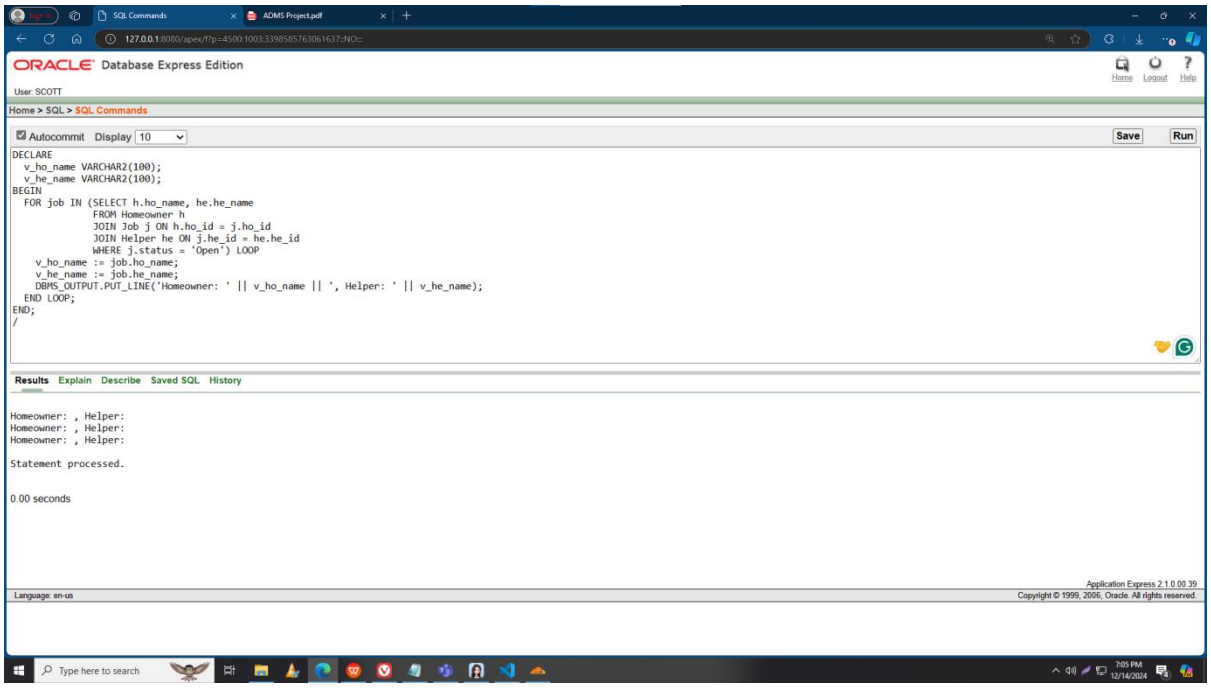


- **Joining**

1. Write a PL/SQL block to retrieve the names of homeowners and the corresponding helper names for all jobs where the job status is 'Open'.

```
DECLARE
    v_ho_name VARCHAR2(100);
    v_he_name VARCHAR2(100);
BEGIN
    FOR job IN (SELECT h.ho_name, he.he_name
```

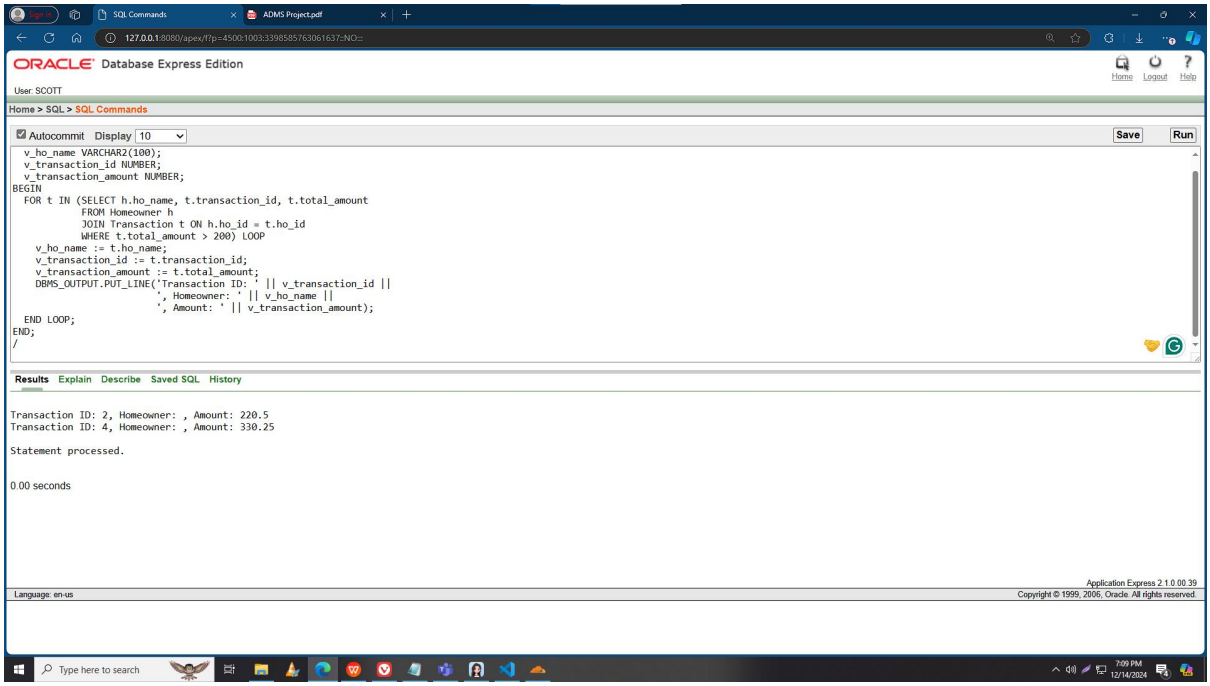
```
                FROM Homeowners h
                JOIN Job j ON h.ho_id = j.ho_id
                JOIN Helper he ON j.he_id = he.he_id
                WHERE j.status = 'Open') LOOP
        v_ho_name := job.ho_name;
        v_he_name := job.he_name;
        DBMS_OUTPUT.PUT_LINE('Homeowner: ' || v_ho_name || ', Helper: ' || v_he_name);
    END LOOP;
END;
```



2. Write a PL/SQL block to retrieve the details of all transactions along with the corresponding homeowner names, where the transaction amount is greater than 200.

```
DECLARE
    v_ho_name VARCHAR2(100);
    v_transaction_id NUMBER;
    v_transaction_amount NUMBER;
```

```
BEGIN
    FOR t IN (SELECT h.ho_name, t.transaction_id, t.total_amount
                    FROM Homeowner h
                    JOIN Transaction t ON h.ho_id = t.ho_id
                    WHERE t.total_amount > 200) LOOP
        v_ho_name := t.ho_name;
        v_transaction_id := t.transaction_id;
        v_transaction_amount := t.total_amount;
        DBMS_OUTPUT.PUT_LINE('Transaction ID: ' || v_transaction_id ||
                                        ', Homeowner: ' || v_ho_name ||
                                        ', Amount: ' || v_transaction_amount);
    END LOOP;
END;
/
```

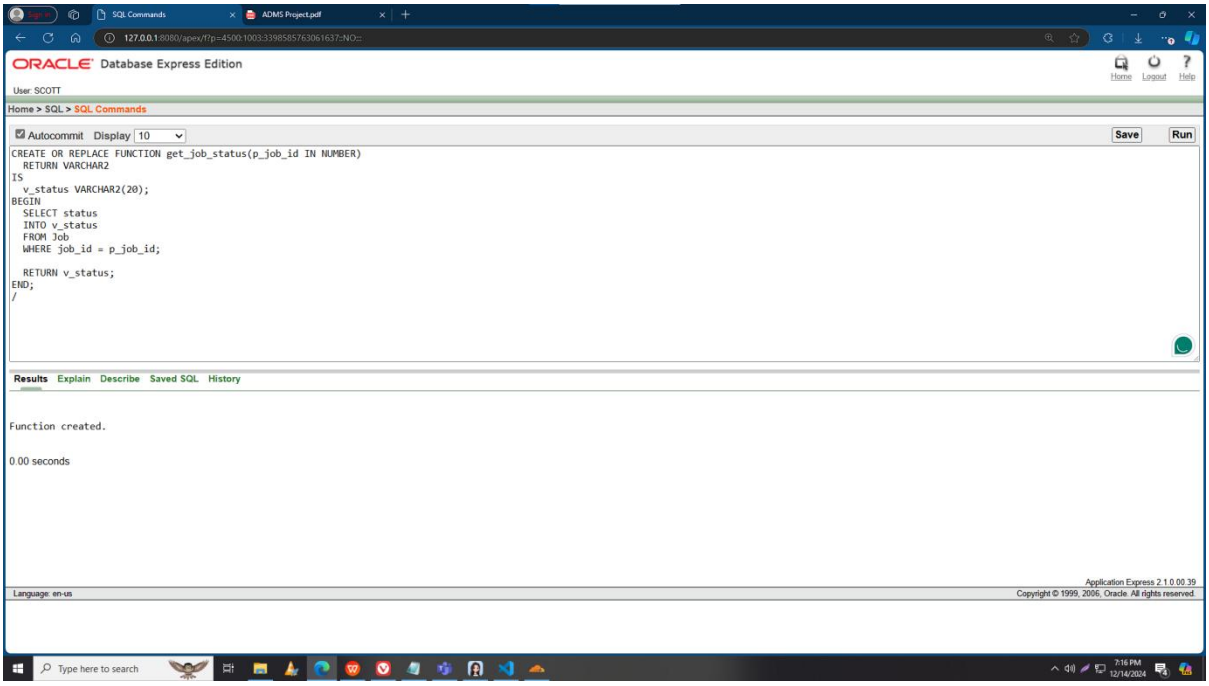


Advanced Query Writing (PL/SQL):

● Stored Function:

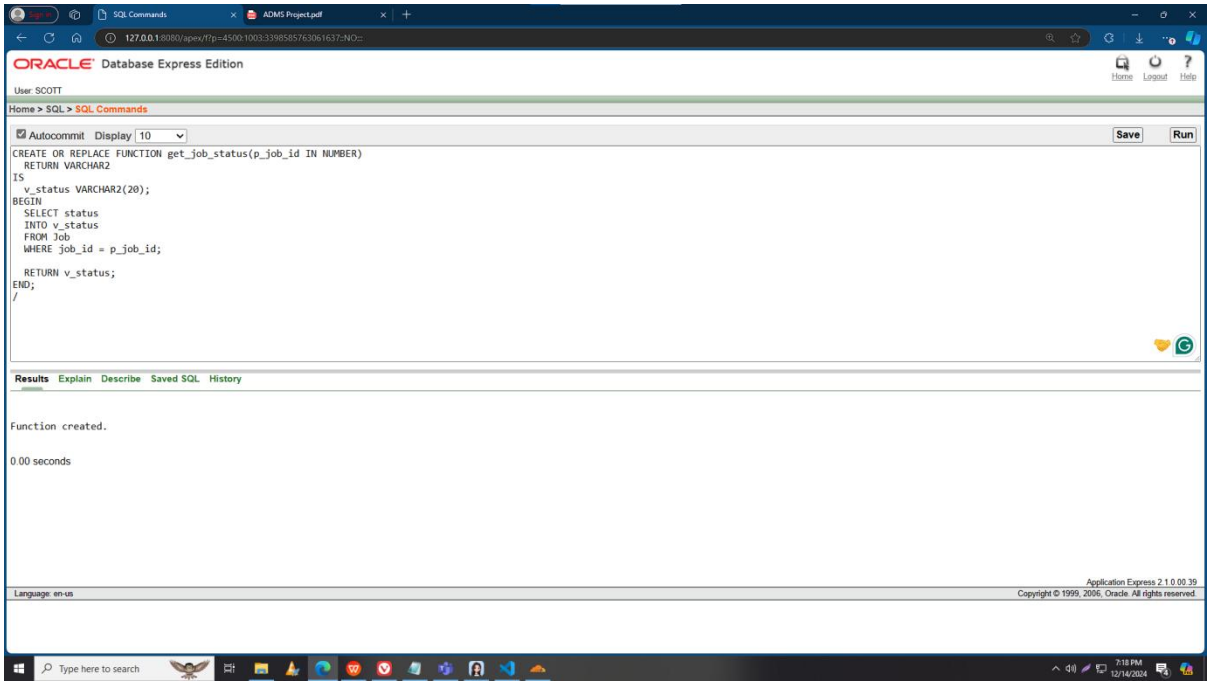
1. Write a PL/SQL stored function that takes a ho_id as an input parameter and returns the total amount of all transactions for that homeowner.

```
CREATE OR REPLACE FUNCTION get_total_transaction_amount(p_ho_id IN
NUMBER)
   RETURN NUMBER
IS
   v_total_amount NUMBER := 0;
BEGIN
   SELECT SUM(total_amount)
   INTO v_total_amount
   FROM Transaction
   WHERE ho_id = p_ho_id;

   RETURN v_total_amount;
END;
```



2. Write a PL/SQL stored function that takes a job_id as an input parameter and returns the status of the job.

```
CREATE OR REPLACE FUNCTION get_job_status(p_job_id IN NUMBER)
   RETURN VARCHAR2
IS
   v_status VARCHAR2(20);
BEGIN
   SELECT status
   INTO v_status
   FROM Job
   WHERE job_id = p_job_id;

   RETURN v_status;
END;
```

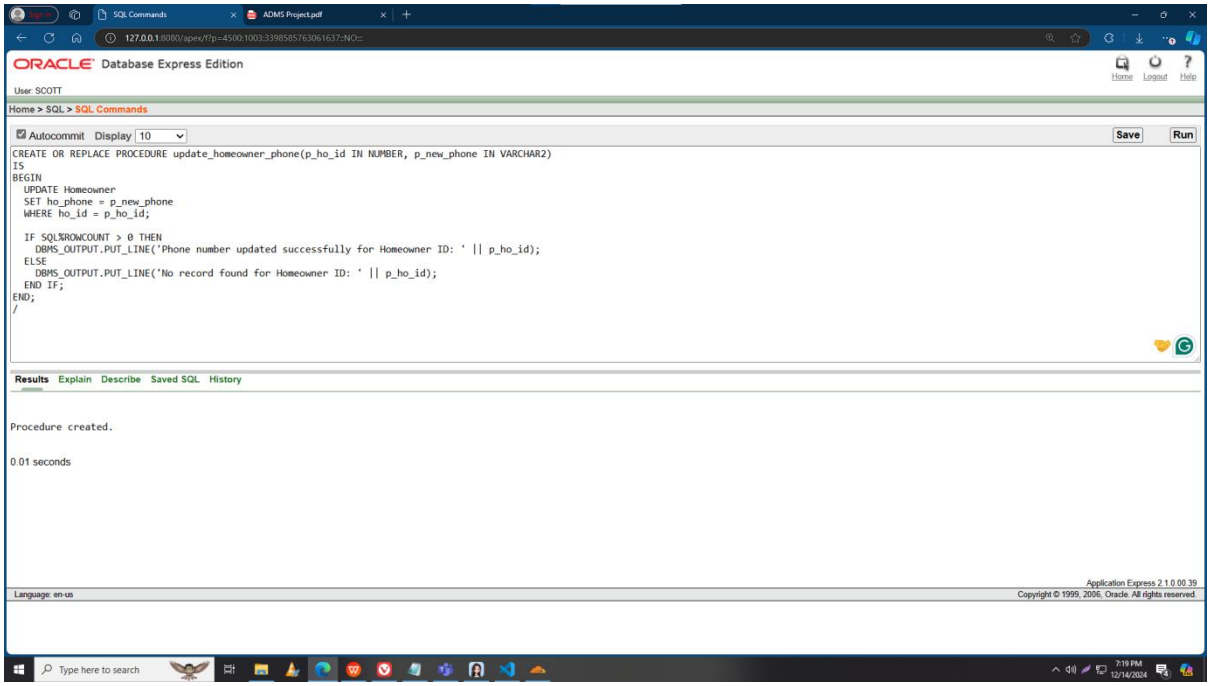


- **Stored Procedure:**

1. Write a PL/SQL stored procedure that takes a ho_id as an input parameter and updates the homeowner's phone number in the Homeowner table. The procedure should also print a message indicating whether the phone number was updated successfully or not.

```
CREATE OR REPLACE PROCEDURE update_homeowner_phone(p_ho_id IN
NUMBER, p_new_phone IN VARCHAR2)

IS

BEGIN

   UPDATE Homeowner

   SET ho_phone = p_new_phone

   WHERE ho_id = p_ho_id;


   IF SQL%ROWCOUNT > 0 THEN

      DBMS_OUTPUT.PUT_LINE('Phone number updated successfully for
Homeowner ID: ' || p_ho_id);
```

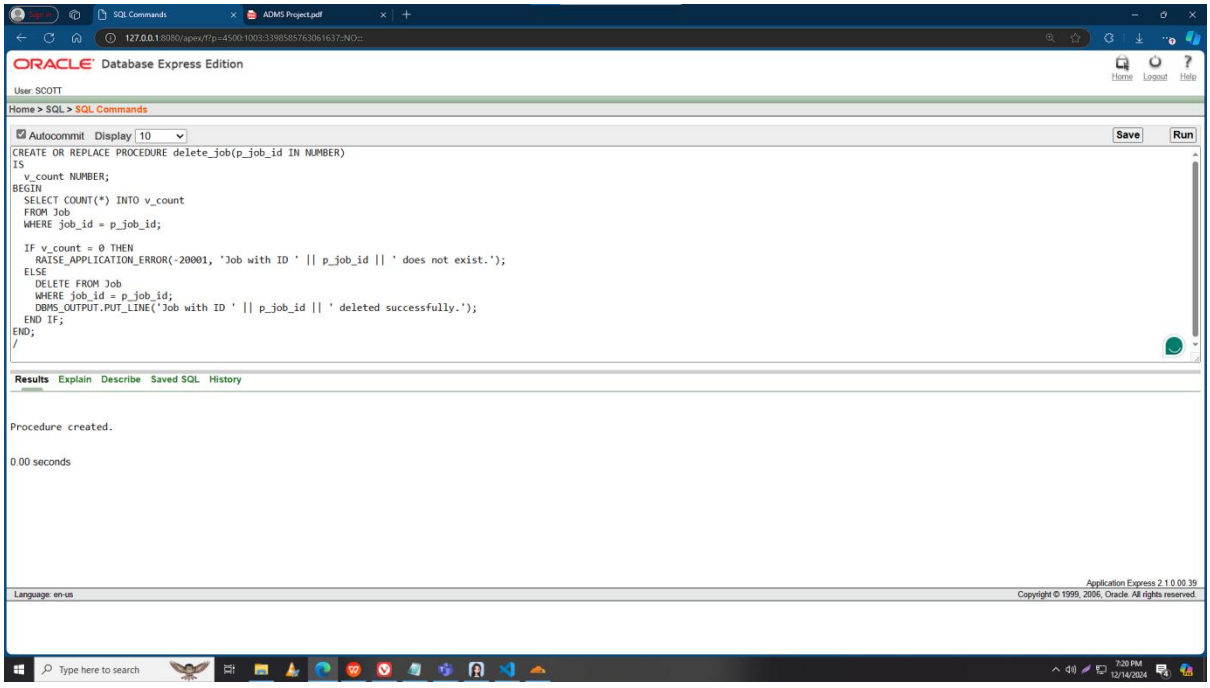
```
    ELSE
        DBMS_OUTPUT.PUT_LINE('No record found for Homeowner ID: ' || p_ho_id);
    END IF;
END;
```



2. Write a PL/SQL stored procedure that takes a job_id as an input parameter and deletes the corresponding job from the Job table. The procedure should raise an exception if no job with the given job_id exists in the table.

```
CREATE OR REPLACE PROCEDURE delete_job(p_job_id IN NUMBER)
IS
    v_count NUMBER;
BEGIN
    SELECT COUNT(*) INTO v_count
    FROM Job
    WHERE job_id = p_job_id;

    IF v_count = 0 THEN
```

```
        RAISE_APPLICATION_ERROR(-20001, 'Job with ID ' || p_job_id || ' does not
exist.');
    ELSE
        DELETE FROM Job
        WHERE job_id = p_job_id;
        DBMS_OUTPUT.PUT_LINE('Job with ID ' || p_job_id || ' deleted successfully.');
    END IF;
END;
/
```

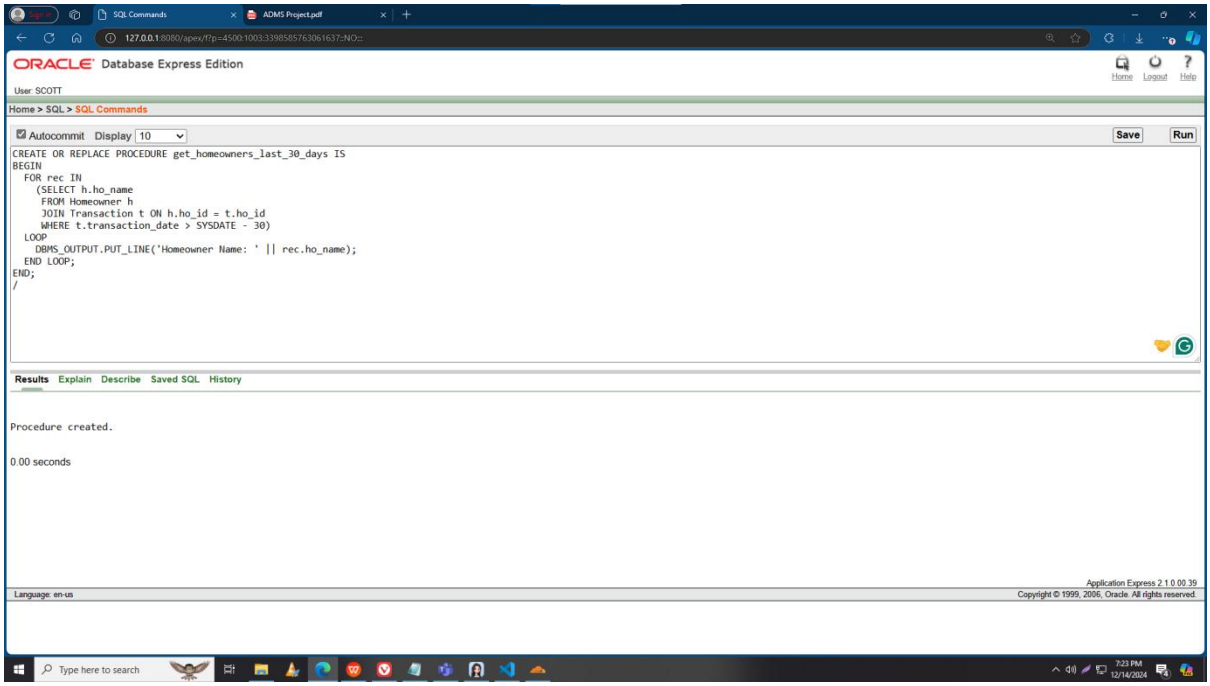


- **Time-based Record**

1. Write a PL/SQL stored procedure to retrieve the names of homeowners who have made transactions in the last 30 days. The procedure should accept no input parameters and should display the names of these homeowners.

```
CREATE OR REPLACE PROCEDURE get_homeowners_last_30_days IS
BEGIN
    FOR rec IN
```

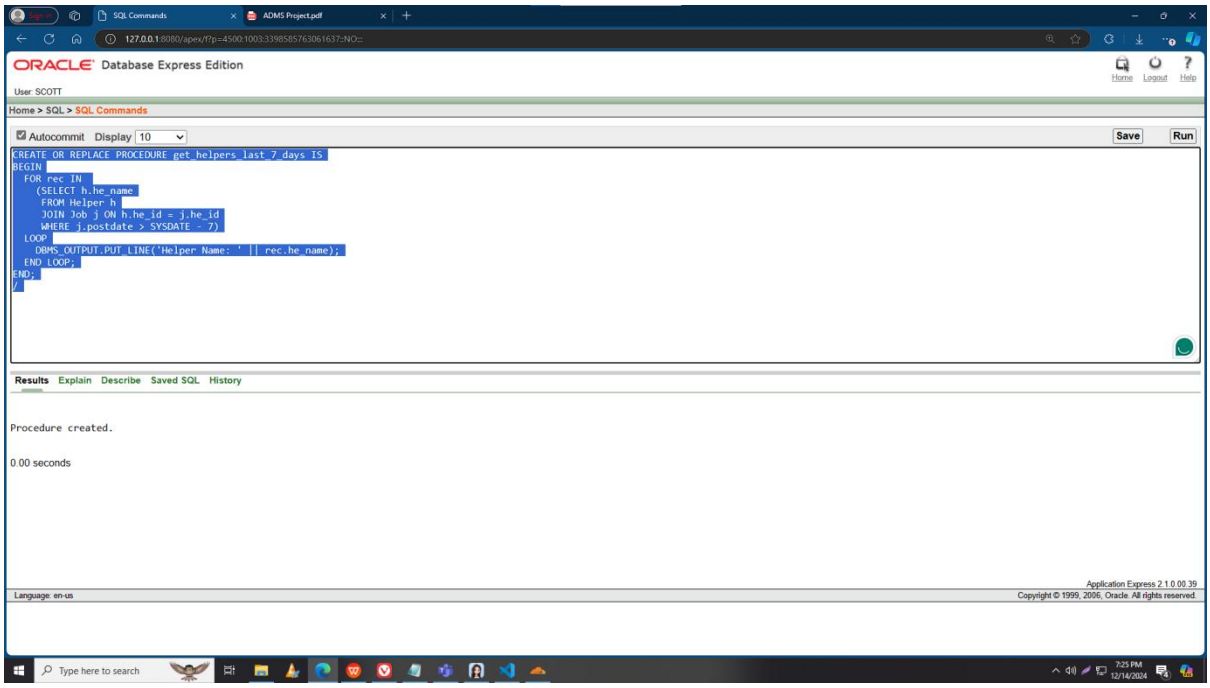
```
    (SELECT h.ho_name
     FROM Homeowner h
     JOIN Transaction t ON h.ho_id = t.ho_id
     WHERE t.transaction_date > SYSDATE - 30)
  LOOP
    DBMS_OUTPUT.PUT_LINE('Homeowner Name: ' || rec.ho_name);
  END LOOP;
END;
```



2. Write a PL/SQL stored procedure to display the names of helpers who have completed at least one job in the last 7 days. The procedure should accept no parameters and should display the names of these helpers.

```
CREATE OR REPLACE PROCEDURE get_helpers_last_7_days IS
BEGIN
  FOR rec IN
    (SELECT h.he_name
     FROM Helper h
```

```
        JOIN Job j ON h.he_id = j.he_id
        WHERE j.postdate > SYSDATE - 7)
    LOOP
      DBMS_OUTPUT.PUT_LINE('Helper Name: ' || rec.he_name);
    END LOOP;
END;
```

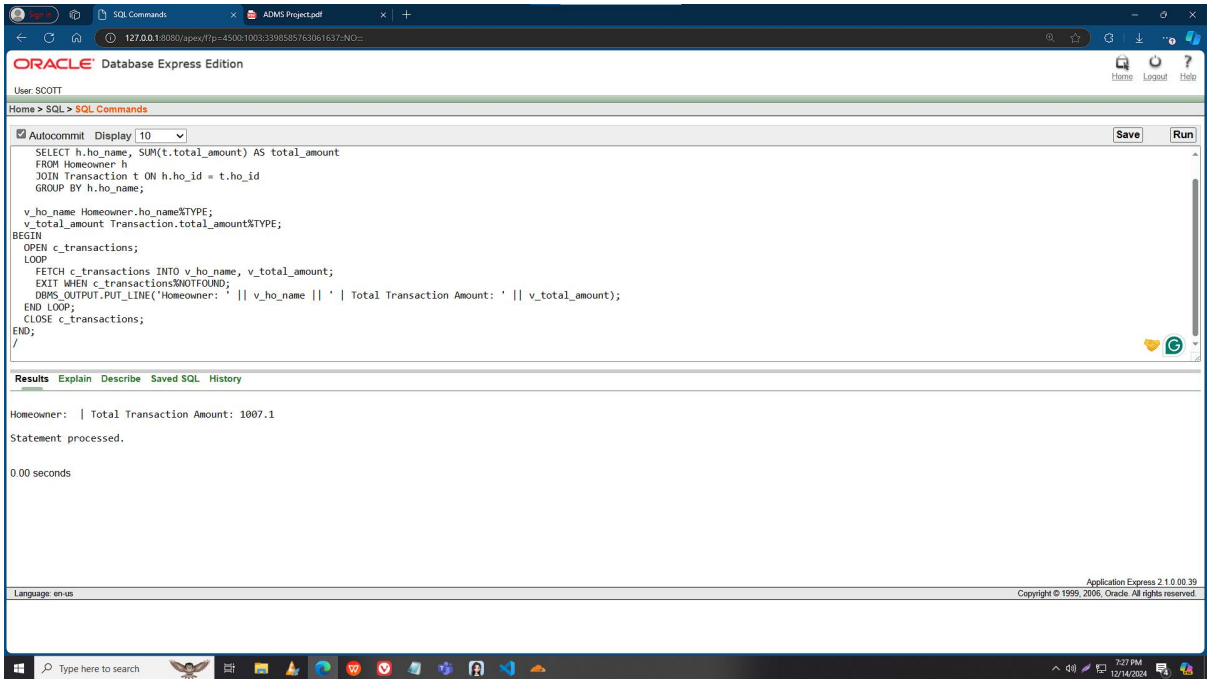


- **Explicit Cursor**

1. Write a PL/SQL block that uses an explicit cursor to retrieve the total transaction amount for each homeowner from the Transaction table. The block should output the ho_name and the corresponding total amount.

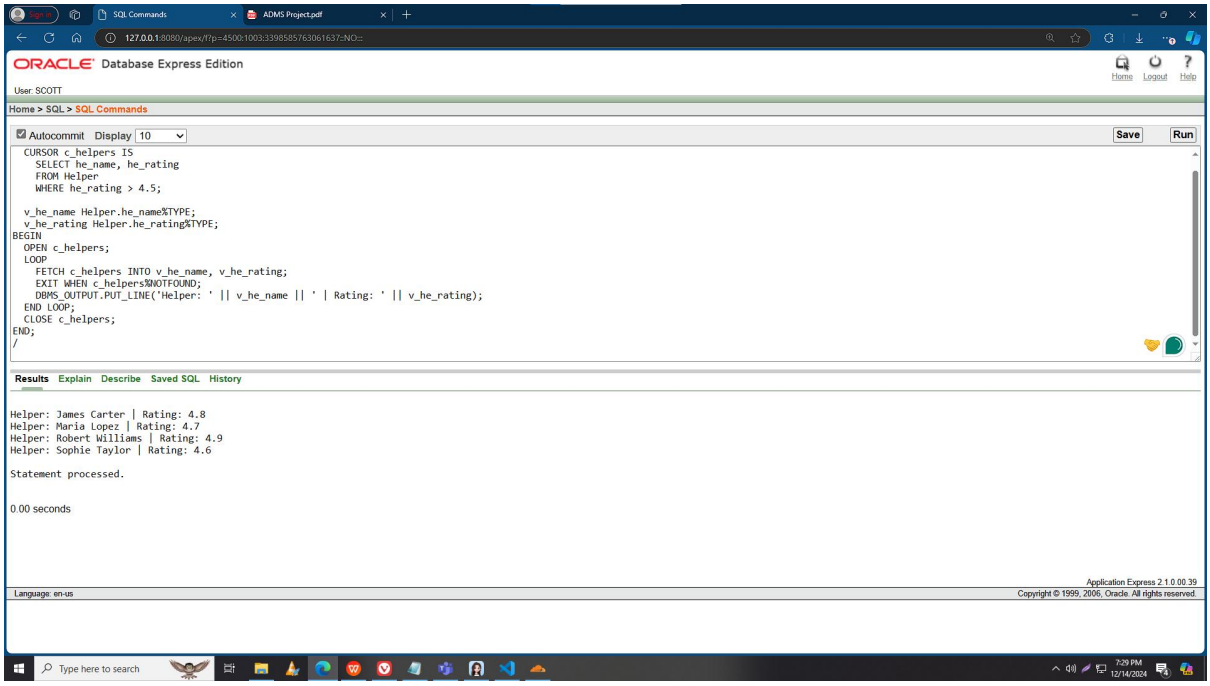
```
DECLARE
  CURSOR c_transactions IS
    SELECT h.ho_name, SUM(t.total_amount) AS total_amount
    FROM Homeowner h
```

```
    JOIN Transaction t ON h.ho_id = t.ho_id
    GROUP BY h.ho_name;


  v_ho_name Homeowner.ho_name%TYPE;
  v_total_amount Transaction.total_amount%TYPE;
BEGIN
  OPEN c_transactions;
  LOOP
    FETCH c_transactions INTO v_ho_name, v_total_amount;
    EXIT WHEN c_transactions%NOTFOUND;
    DBMS_OUTPUT.PUT_LINE('Homeowner: ' || v_ho_name || ' | Total Transaction
Amount: ' || v_total_amount);
  END LOOP;
  CLOSE c_transactions;
END;
```



2. Write a PL/SQL block that uses an explicit cursor to retrieve and display the names and ratings of helpers from the Helper table who have a rating greater than 4.5.

```
DECLARE
   CURSOR c_helpers IS
      SELECT he_name, he_rating
      FROM Helper
      WHERE he_rating > 4.5;

   v_he_name Helper.he_name%TYPE;
   v_he_rating Helper.he_rating%TYPE;
BEGIN
   OPEN c_helpers;
   LOOP
      FETCH c_helpers INTO v_he_name, v_he_rating;
      EXIT WHEN c_helpers%NOTFOUND;
      DBMS_OUTPUT.PUT_LINE('Helper: ' || v_he_name || ' | Rating: ' ||
v_he_rating);
   END LOOP;
   CLOSE c_helpers;
END;
/
```

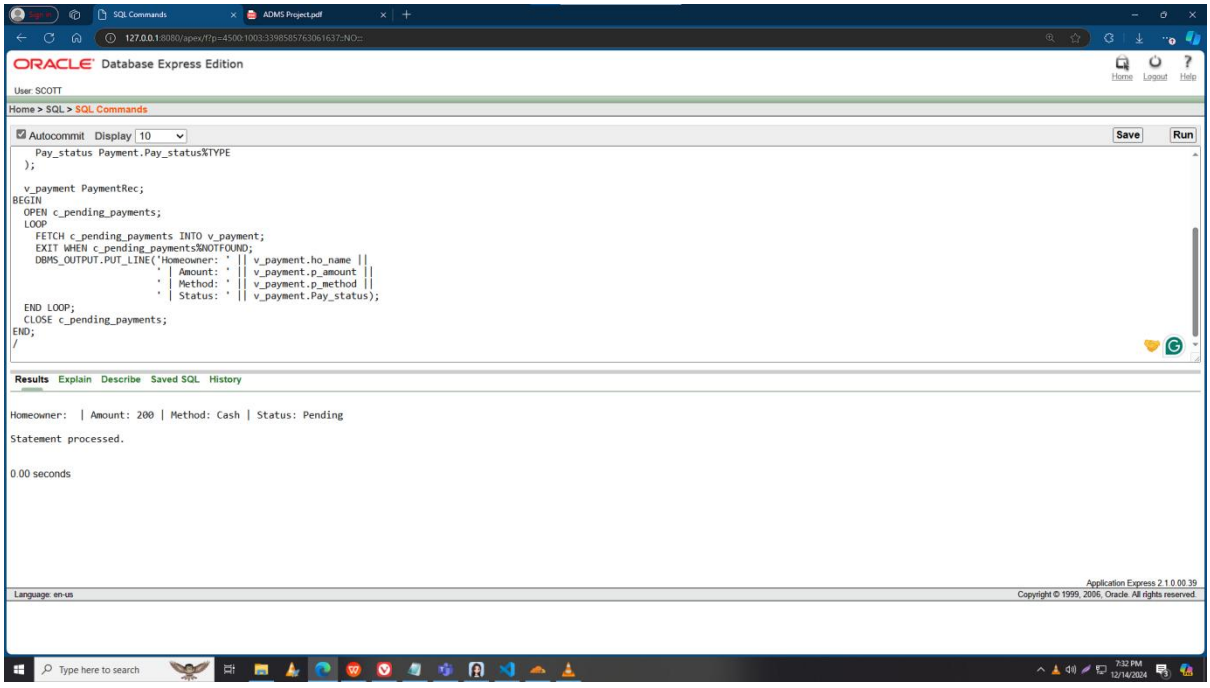


- **Cursor-Based Record**

1. Write a PL/SQL block that uses a cursor-based record to retrieve all the homeowners (ho_name) who have a pending payment status from the Payment table. The block should display the ho_name and the payment details (i.e., p_amount, p_method, and Pay_status).

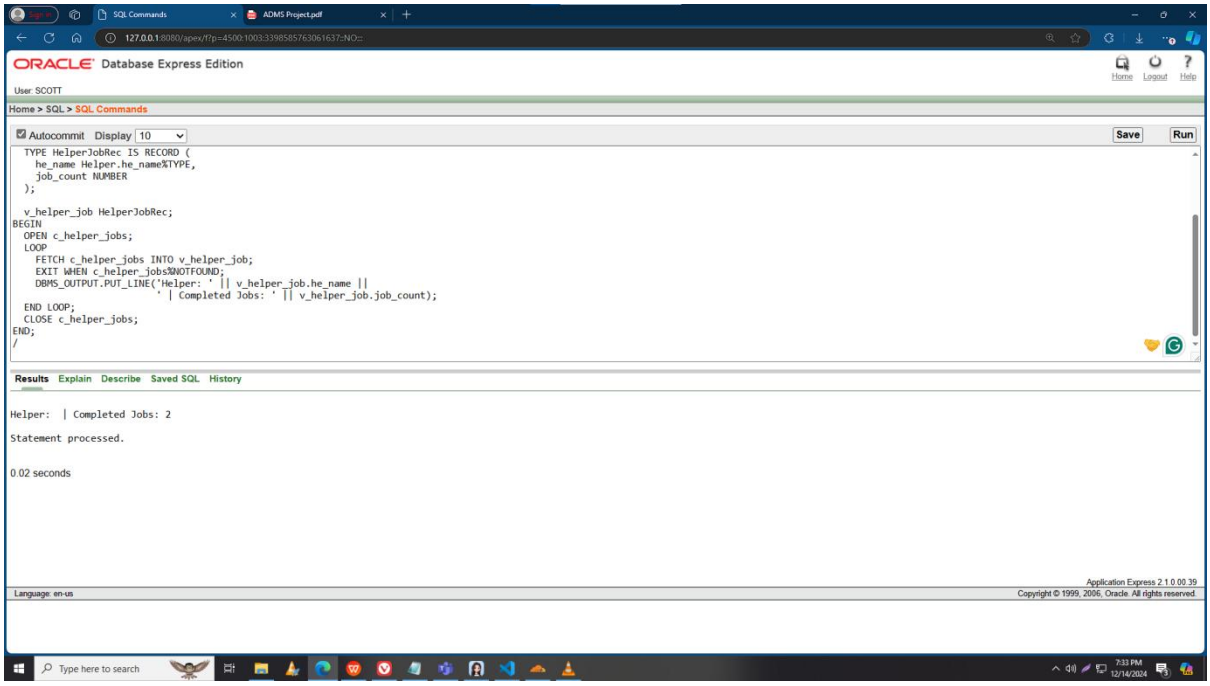
```
DECLARE
  CURSOR c_pending_payments IS
    SELECT h.ho_name, p.p_amount, p.p_method, p.Pay_status
    FROM Homeowner h
    JOIN Payment p ON h.ho_id = p.ho_id
    WHERE p.Pay_status = 'Pending';

  TYPE PaymentRec IS RECORD (
    ho_name Homeowner.ho_name%TYPE,
    p_amount Payment.p_amount%TYPE,
```

```
    p_method Payment.p_method%TYPE,
    Pay_status Payment.Pay_status%TYPE
  );

  v_payment PaymentRec;
BEGIN
  OPEN c_pending_payments;
  LOOP
    FETCH c_pending_payments INTO v_payment;
    EXIT WHEN c_pending_payments%NOTFOUND;
    DBMS_OUTPUT.PUT_LINE('Homeowner: ' || v_payment.ho_name ||
                         ' | Amount: ' || v_payment.p_amount ||
                         ' | Method: ' || v_payment.p_method ||
                         ' | Status: ' || v_payment.Pay_status);
  END LOOP;
  CLOSE c_pending_payments;
END;
```



2. Write a PL/SQL block that uses a cursor-based record to retrieve the names of all helpers (he_name) and the number of jobs they have completed from the Job table. Display the helper's name along with their job count.

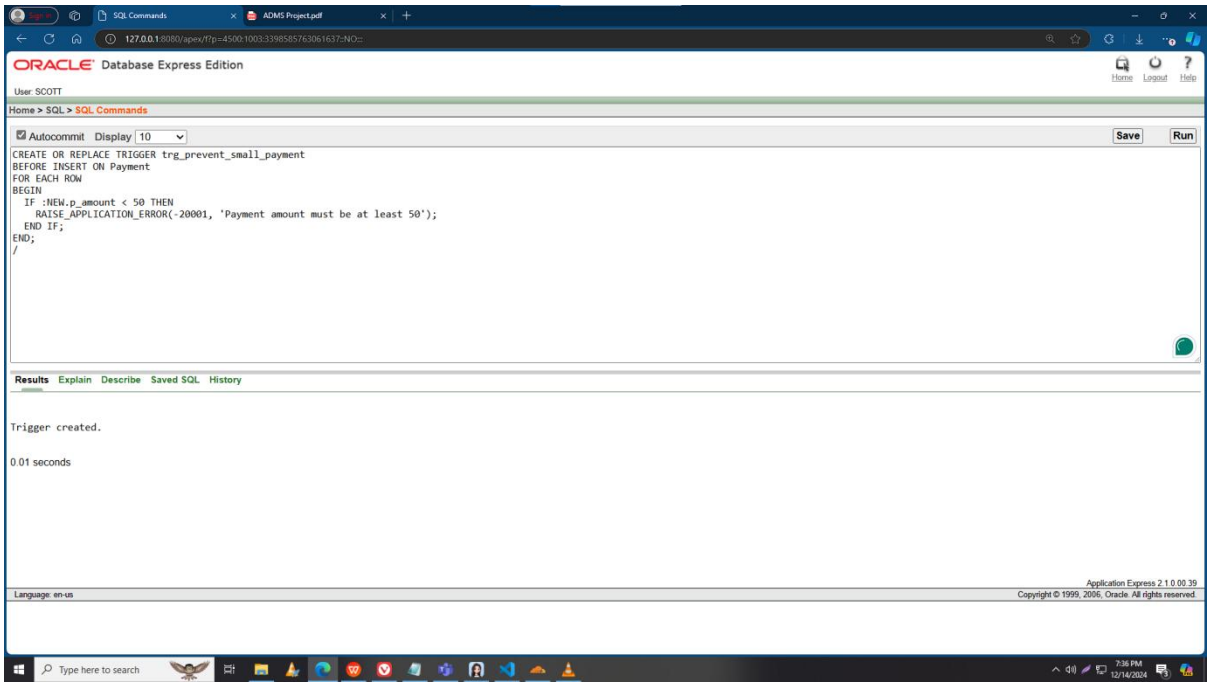
```
DECLARE
   CURSOR c_helper_jobs IS
      SELECT h.he_name, COUNT(j.job_id) AS job_count
      FROM Helper h
      JOIN Job j ON h.he_id = j.he_id
      WHERE j.status = 'Closed'
      GROUP BY h.he_name;

   TYPE HelperJobRec IS RECORD (
      he_name Helper.he_name%TYPE,
      job_count NUMBER
   );

   v_helper_job HelperJobRec;
BEGIN
   OPEN c_helper_jobs;
   LOOP
      FETCH c_helper_jobs INTO v_helper_job;
      EXIT WHEN c_helper_jobs%NOTFOUND;
      DBMS_OUTPUT.PUT_LINE('Helper: ' || v_helper_job.he_name ||
                           ' | Completed Jobs: ' || v_helper_job.job_count);
   END LOOP;
   CLOSE c_helper_jobs;
END;
/
```



- **Row-Level Trigger**

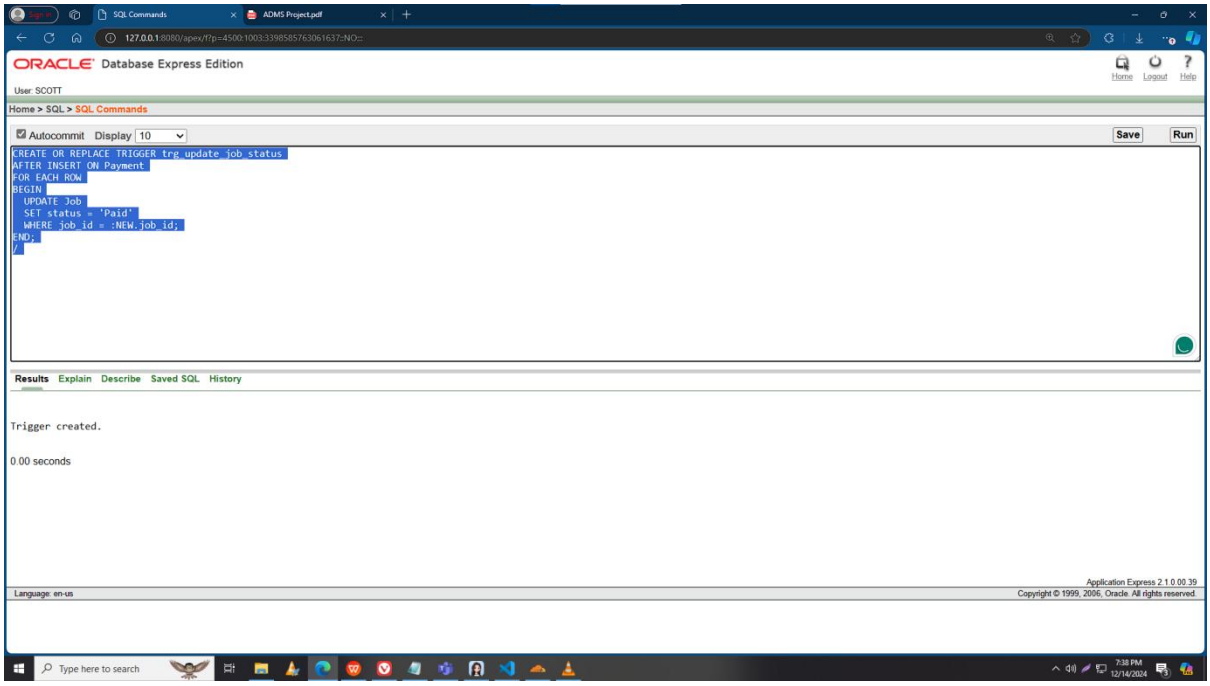
1. Create a row-level trigger on the Payment table that will prevent a payment from being inserted if the payment amount (p_amount) is less than 50. The trigger should raise an error with a message: "Payment amount must be at least 50."

```
CREATE OR REPLACE TRIGGER trg_prevent_small_payment
BEFORE INSERT ON Payment
FOR EACH ROW
BEGIN
    IF :NEW.p_amount < 50 THEN
        RAISE_APPLICATION_ERROR(-20001, 'Payment amount must be at least 50');
    END IF;
END;
```



2. **Create a row-level trigger on the Payment table that will automatically update the status of the associated Job to 'Paid' when a payment is inserted into the Payment table. The trigger should fire after an insert operation on the Payment table.**

```
CREATE OR REPLACE TRIGGER trg_update_job_status
AFTER INSERT ON Payment
FOR EACH ROW
BEGIN
    UPDATE Job
    SET status = 'Paid'
    WHERE job_id = :NEW.job_id;
END;
/
```



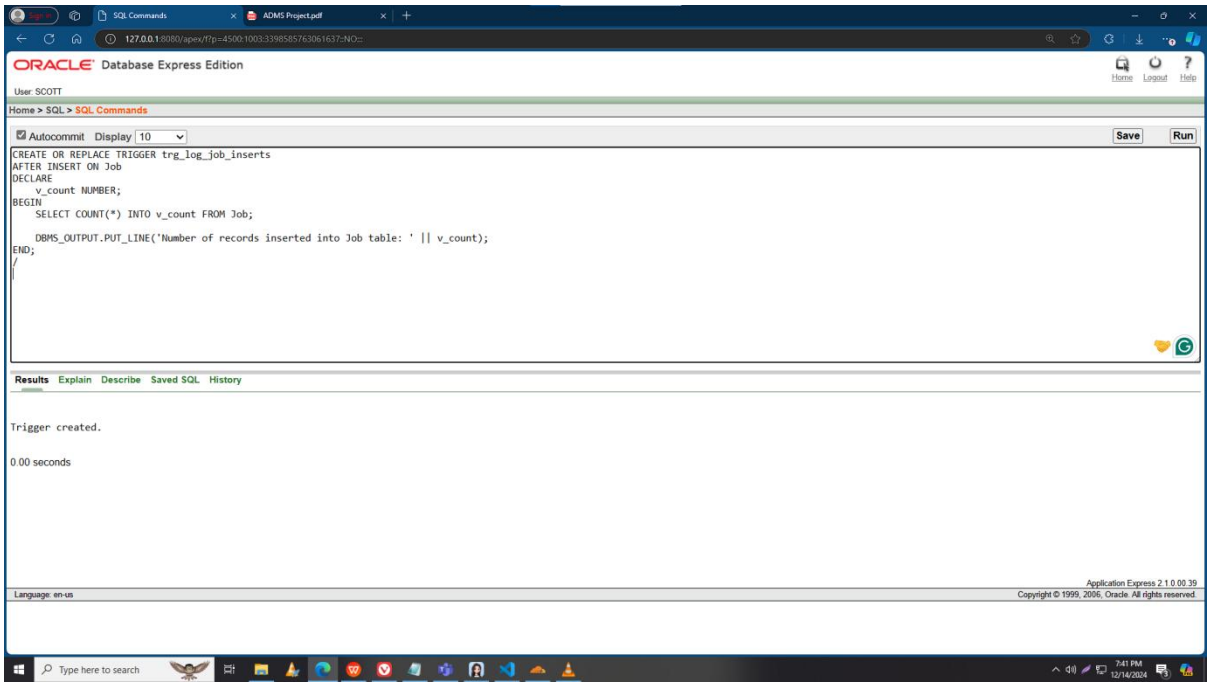
- **Statement level Trigger:**

1. Create a statement-level trigger that tracks every insert operation on the Job table. The trigger should output a message to the console that states: "Number of records inserted into Job table: X", where X is the number of rows inserted. You do not need to create a separate table to log this information.

```
CREATE OR REPLACE TRIGGER trg_log_job_inserts
AFTER INSERT ON Job
DECLARE
    v_count NUMBER;
BEGIN
    SELECT COUNT(*) INTO v_count FROM Job;
```

DBMS_OUTPUT.PUT_LINE('Number of records inserted into Job table: ' || v_count);

END;



2. Create a statement-level trigger that tracks every update operation on the Payment table. The trigger should output a message to the console stating: "Update operation performed on Payment table." The message should also include the number of records updated in the Payment table.

CREATE OR REPLACE TRIGGER trg_log_payment_updates

AFTER UPDATE ON Payment

DECLARE

v_count NUMBER;

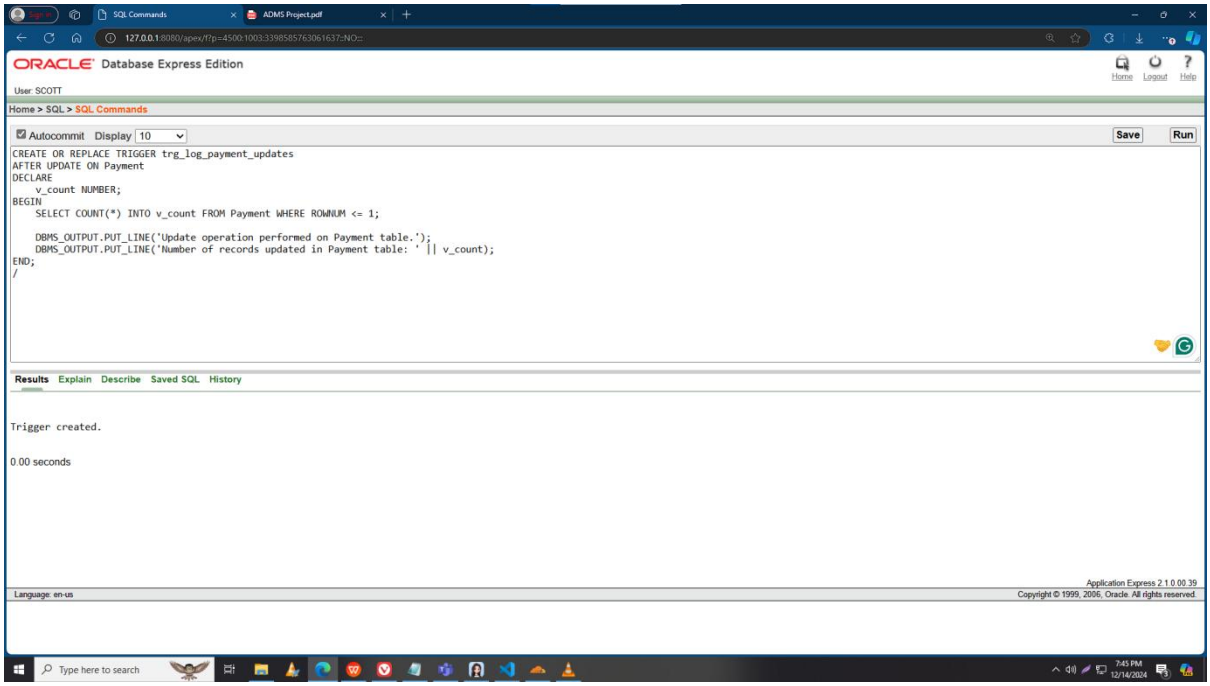
BEGIN

SELECT COUNT(*) INTO v_count FROM Payment WHERE ROWNUM <= 1;

DBMS_OUTPUT.PUT_LINE('Update operation performed on Payment table.');

DBMS_OUTPUT.PUT_LINE('Number of records updated in Payment table: ' || v_count);

END;



- **Package**

1. Create a package to retrieve homeowner details.

CREATE OR REPLACE PACKAGE homeowner_pkg IS

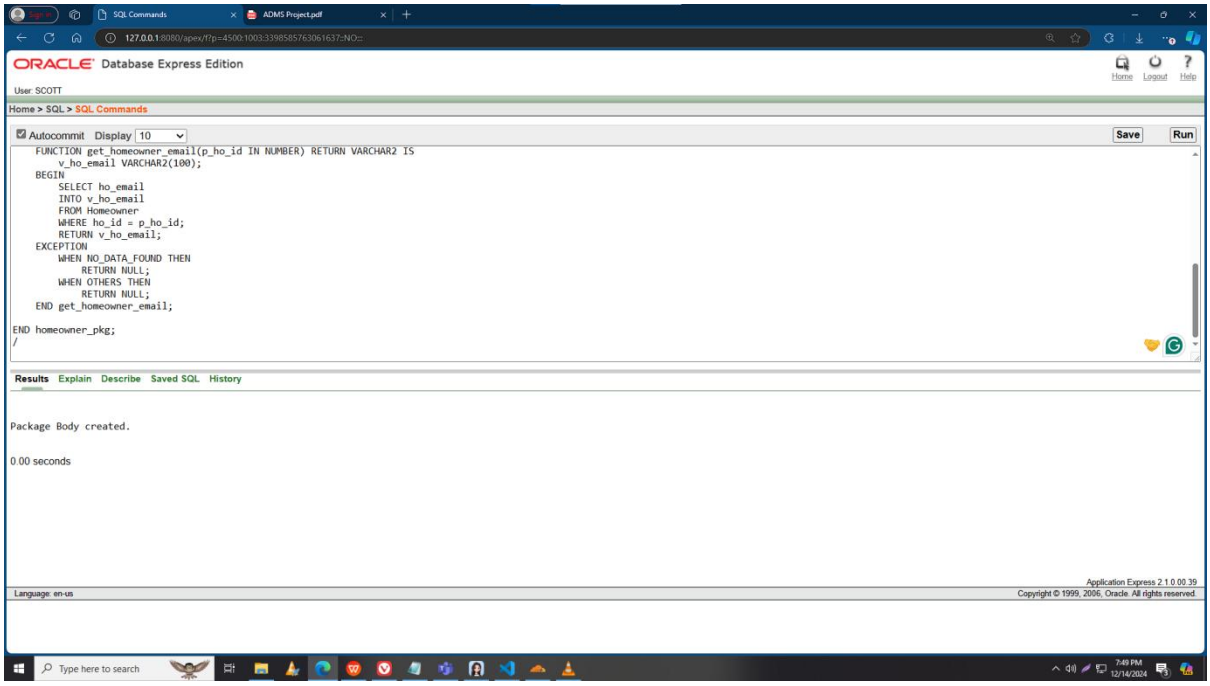
PROCEDURE get_homeowner_details(p_ho_id IN NUMBER);

FUNCTION get_homeowner_email(p_ho_id IN NUMBER) RETURN VARCHAR2;

END homeowner_pkg;

```
CREATE OR REPLACE PACKAGE BODY homeowner_pkg IS

    PROCEDURE get_homeowner_details(p_ho_id IN NUMBER)
IS
        v_ho_name       VARCHAR2(100);
        v_ho_address VARCHAR2(255);
        v_ho_email     VARCHAR2(100);
        v_ho_phone     VARCHAR2(20);
    BEGIN
        SELECT ho_name, ho_address, ho_email, ho_phone
        INTO v_ho_name, v_ho_address, v_ho_email, v_ho_phone
        FROM Homeowner
        WHERE ho_id = p_ho_id;

        DBMS_OUTPUT.PUT_LINE('Homeowner Details:');
        DBMS_OUTPUT.PUT_LINE('Name: ' || v_ho_name);
        DBMS_OUTPUT.PUT_LINE('Address: ' || v_ho_address);
        DBMS_OUTPUT.PUT_LINE('Email: ' || v_ho_email);
        DBMS_OUTPUT.PUT_LINE('Phone: ' || v_ho_phone);
    EXCEPTION
        WHEN NO_DATA_FOUND THEN
            DBMS_OUTPUT.PUT_LINE('No homeowner found
with ID ' || p_ho_id);
```

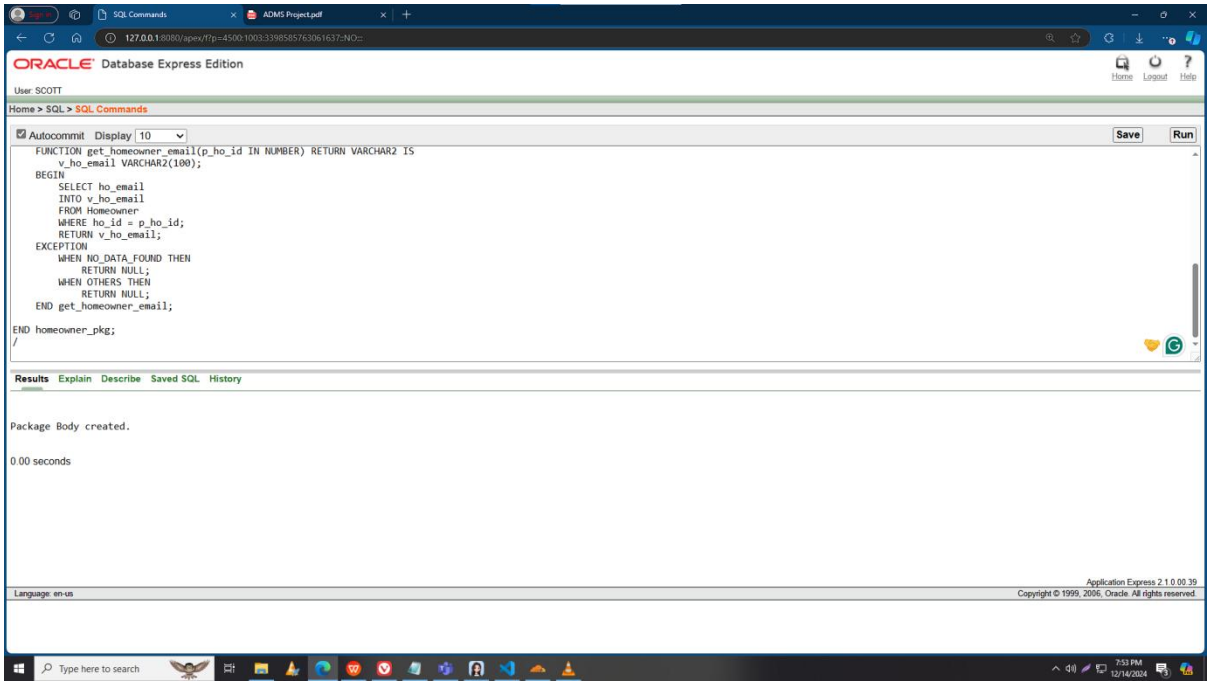
```
        WHEN OTHERS THEN
            DBMS_OUTPUT.PUT_LINE('An error occurred.');
    END get_homeowner_details;

    FUNCTION get_homeowner_email(p_ho_id IN NUMBER)
RETURN VARCHAR2 IS
        v_ho_email VARCHAR2(100);
    BEGIN
        SELECT ho_email
        INTO v_ho_email
        FROM Homeowner
        WHERE ho_id = p_ho_id;
        RETURN v_ho_email;
    EXCEPTION
        WHEN NO_DATA_FOUND THEN
            RETURN NULL;
        WHEN OTHERS THEN
            RETURN NULL;
    END get_homeowner_email;
END homeowner_pkg;
```



2. Create a package to calculate helper performance.

```
CREATE OR REPLACE PACKAGE helper_performance_pkg IS
    PROCEDURE update_performance(p_he_id IN NUMBER, p_rating IN NUMBER);
    FUNCTION get_average_rating(p_he_id IN NUMBER) RETURN NUMBER;
END helper_performance_pkg;

CREATE OR REPLACE PACKAGE BODY helper_performance_pkg IS

    PROCEDURE update_performance(p_he_id IN NUMBER, p_rating IN NUMBER) IS
    BEGIN
        UPDATE Helper
        SET he_rating = p_rating
        WHERE he_id = p_he_id;

        COMMIT;
        DBMS_OUTPUT.PUT_LINE('Helper performance updated for ID ' || p_he_id);
```

```
    EXCEPTION
        WHEN NO_DATA_FOUND THEN
            DBMS_OUTPUT.PUT_LINE('No helper found with ID ' || p_he_id);
        WHEN OTHERS THEN
            DBMS_OUTPUT.PUT_LINE('An error occurred while updating
performance.');
    END update_performance;

    FUNCTION get_average_rating(p_he_id IN NUMBER) RETURN NUMBER IS
        v_avg_rating NUMBER;
    BEGIN
        SELECT AVG(he_rating)
        INTO v_avg_rating
        FROM Helper
        WHERE he_id = p_he_id;

        RETURN v_avg_rating;
    EXCEPTION
        WHEN NO_DATA_FOUND THEN
            RETURN NULL;
        WHEN OTHERS THEN
            RETURN NULL;
    END get_average_rating;
END helper_performance_pkg;
```


SQL Commands
ADMS Project.pdf
127.0.0.1:8080/apex/f?p=4500:1003:33985857630616 37::NO::
ORACLE Database Express Edition
Home
Logout
Help
User: SCOTT
Home > SQL > SQL Commands
Autocommit
Display 10
Save
Run
FUNCTION get_homeowner_email(p_ho_id IN NUMBER) RETURN VARCHAR2 IS
v_ho_email VARCHAR2(100);
BEGIN
SELECT ho_email
INTO v_ho_email
FROM Homeowner
WHERE ho_id = p_ho_id;
RETURN v_ho_email;
EXCEPTION
WHEN NO_DATA_FOUND THEN
RETURN NULL;
WHEN OTHERS THEN
RETURN NULL;
END get_homeowner_email;
END homeowner_pkg;
/
Results
Explain
Describe
Saved SQL
History
Package Body created.
0.00 seconds
Application Express 2.1.0.00.39
Language: en-us
Copyright © 1999, 2006, Oracle. All rights reserved.
Type here to search
7:53 PM
12/14/2024

Conclusion

Developed a comprehensive database schema for an helping hand agency. Created sample data and executed SQL queries for data retrieval. Explored relational algebra for database query formulation.

Our planned future work:

- Improve data analysis capabilities with complex SQL queries.
- Develop a user-friendly interface for easier database interaction.
- Enhance security measures for data protection.
- Optimize database performance for scalability.