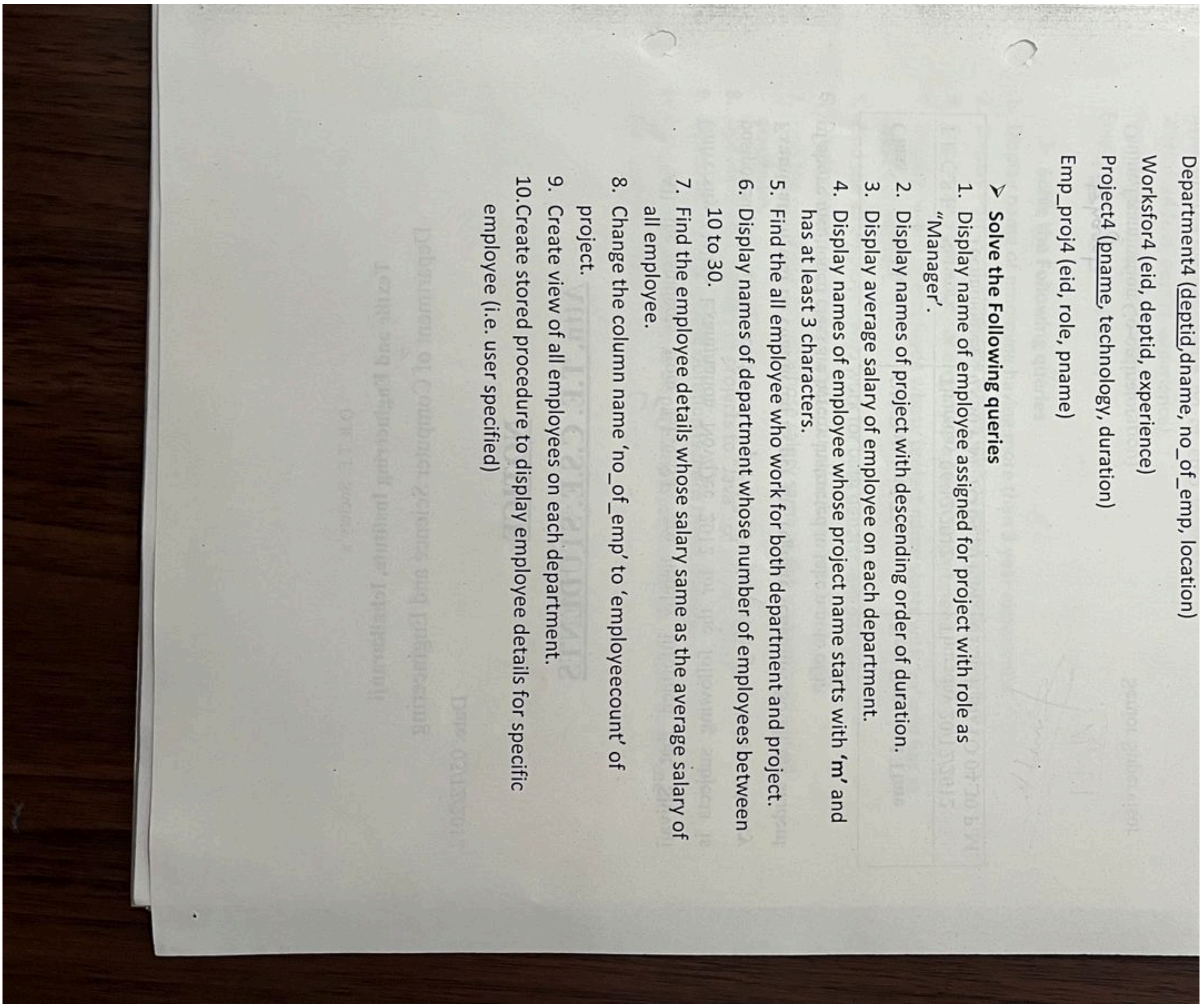
answers.md



Department4 (deptid, dname, no_of_emp, location)
Worksfor4 (eid, deptid, experience)
Project4 (pname, technology, duration)
Emp_proj4 (eid, role, pname)

➢ **Solve the Following queries**

1. Display name of employee assigned for project with role as "Manager".
2. Display names of project with descending order of duration.
3. Display average salary of employee on each department.
4. Display names of employee whose project name starts with **'m'** and has at least 3 characters.
5. Find the all employee who work for both department and project.
6. Display names of department whose number of employees between 10 to 30.
7. Find the employee details whose salary same as the average salary of all employee.
8. Change the column name 'no_of_emp' to 'employeecount' of project.
9. Create view of all employees on each department.
10. Create stored procedure to display employee details for specific employee (i.e. user specified)

Here are the Oracle SQL queries for each of your requests:

1. **Display the name of the employee assigned for a project with the role "Manager":**

```
SELECT e.employee_name
FROM employees e
JOIN project_assignments pa ON e.employee_id = pa.employee_id
WHERE pa.role = 'Manager';
```

2. **Display the names of projects in descending order of their duration:**

```
SELECT project_name
FROM projects
ORDER BY duration DESC;
```

3. **Display the average salary of employees in each department:**

```
SELECT department_id, AVG(salary) AS average_salary
FROM employees
GROUP BY department_id;
```

4. **Display the names of employees whose project name starts with 'm' and has at least 3 characters:**

```
SELECT e.employee_name
FROM employees e
JOIN projects p ON e.project_id = p.project_id
WHERE LOWER(SUBSTR(p.project_name, 1, 1)) = 'm' AND LENGTH(p.project_name) >= 3;
```

5. **Find all employees who work for both a department and a project:**

```
SELECT e.employee_name
FROM employees e
JOIN departments d ON e.department_id = d.department_id
JOIN projects p ON e.project_id = p.project_id;
```

6. **Display the names of departments whose number of employees is between 10 and 30:**

```
SELECT d.department_name
FROM departments d
JOIN employees e ON d.department_id = e.department_id
GROUP BY d.department_name
HAVING COUNT(e.employee_id) BETWEEN 10 AND 30;
```

7. **Find the employee details whose salary is the same as the average salary of all employees:**

```
SELECT *
FROM employees
WHERE salary = (SELECT AVG(salary) FROM employees);
```

8. **Change the column name 'no_of_emp' to 'employeecount' in the 'Project4' table:**

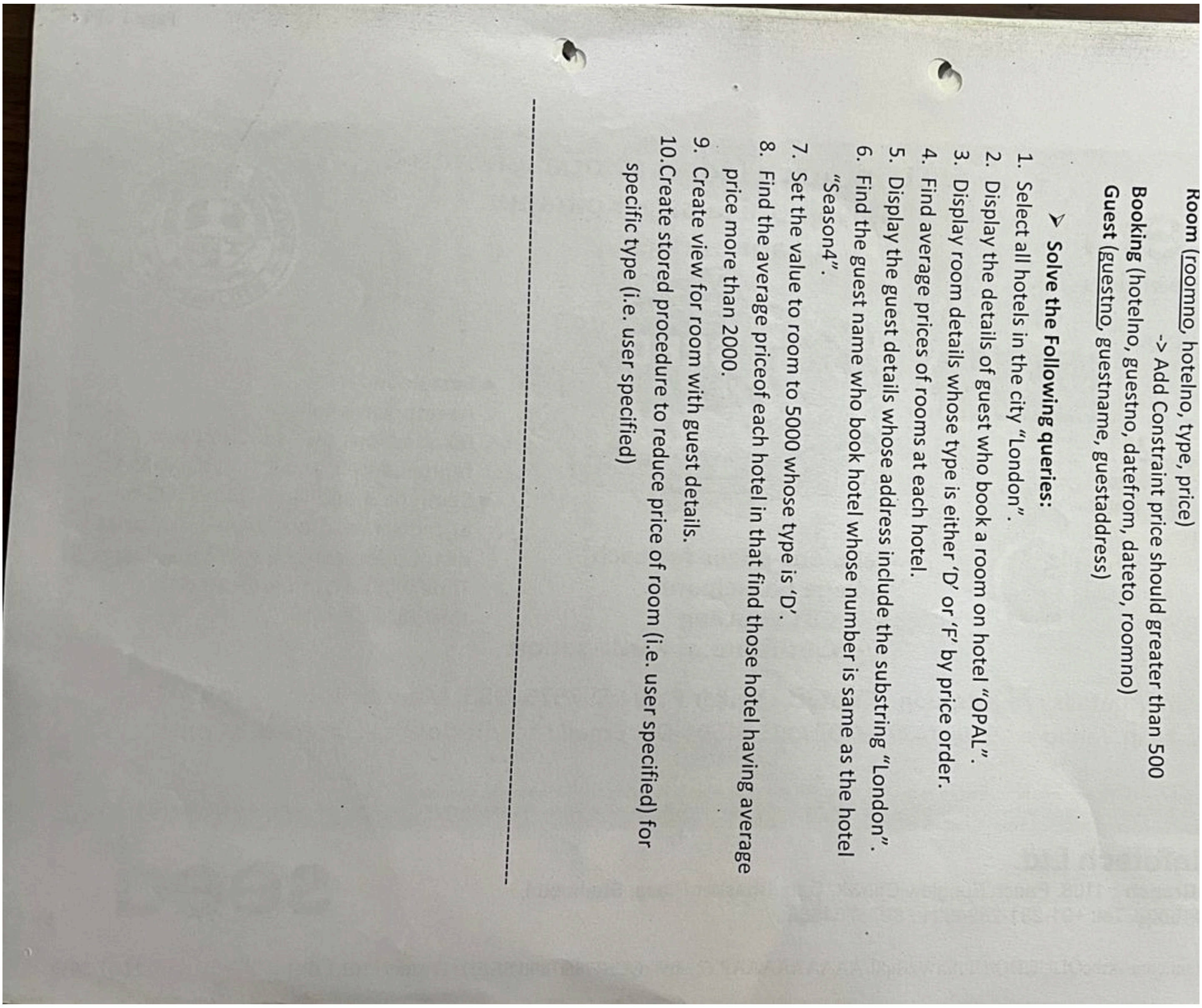
```
ALTER TABLE Project4 RENAME COLUMN no_of_emp TO employeecount;
```

9. **Create a view of all employees in each department:**

```
CREATE VIEW department_employees AS
SELECT e.employee_name, d.department_name
FROM employees e
JOIN departments d ON e.department_id = d.department_id;
```

10. **Create a stored procedure to display employee details for a specific employee (user-specified):**

```
CREATE OR REPLACE PROCEDURE GetEmployeeDetails(p_employee_id IN NUMBER)
IS
BEGIN
    SELECT *
    INTO :employee_details
    FROM employees
    WHERE employee_id = p_employee_id;
EXCEPTION
    WHEN NO_DATA_FOUND THEN
        DBMS_OUTPUT.PUT_LINE('No employee found with ID: ' || p_employee_id);
END;
```



Room (roomno, hotelno, type, price)

-> Add Constraint price should greater than 500

Booking (hotelno, guestno, datefrom, dateto, roomno)

Guest (guestno, guestname, guestaddress)

➢ **Solve the Following queries:**

1. Select all hotels in the city "London".
2. Display the details of guest who book a room on hotel "OPAL".
3. Display room details whose type is either 'D' or 'F' by price order.
4. Find average prices of rooms at each hotel.
5. Display the guest details whose address include the substring "London".
6. Find the guest name who book hotel whose number is same as the hotel "Season4".
7. Set the value to room to 5000 whose type is 'D'.
8. Find the average priceof each hotel in that find those hotel having average price more than 2000.
9. Create view for room with guest details.
10. Create stored procedure to reduce price of room (i.e. user specified) for specific type (i.e. user specified)

Here are the Oracle SQL queries for your requests:

1. **Select all hotels in the city "London":**

```
SELECT *
FROM hotels
WHERE city = 'London';
```

2. **Display the details of guests who booked a room in the hotel "OPAL":**

```
SELECT g.*
FROM guests g
JOIN bookings b ON g.guest_id = b.guest_id
JOIN hotels h ON b.hotel_id = h.hotel_id
WHERE h.hotel_name = 'OPAL';
```

3. **Display room details whose type is either 'D' or 'F' by price order:**

```
SELECT *
FROM rooms
WHERE room_type IN ('D', 'F')
ORDER BY price;
```

4. **Find the average prices of rooms at each hotel:**

```
SELECT hotel_id, AVG(price) AS average_price
FROM rooms
GROUP BY hotel_id;
```

5. **Display the guest details whose address includes the substring "London":**

```
SELECT *
FROM guests
WHERE LOWER(address) LIKE '%london%';
```

6. **Find the guest name who booked a hotel whose number is the same as the hotel "Season4":**

```
SELECT g.guest_name
FROM guests g
JOIN bookings b ON g.guest_id = b.guest_id
JOIN hotels h1 ON b.hotel_id = h1.hotel_id
WHERE h1.hotel_number = (SELECT h2.hotel_number FROM hotels h2 WHERE h2.hotel_name = 'Season4');
```

7. **Set the value of the room to 5000 whose type is 'D':**

```
UPDATE rooms
SET price = 5000
WHERE room_type = 'D';
```

8. **Find the average price of each hotel, and then find those hotels with an average price more than 2000:**

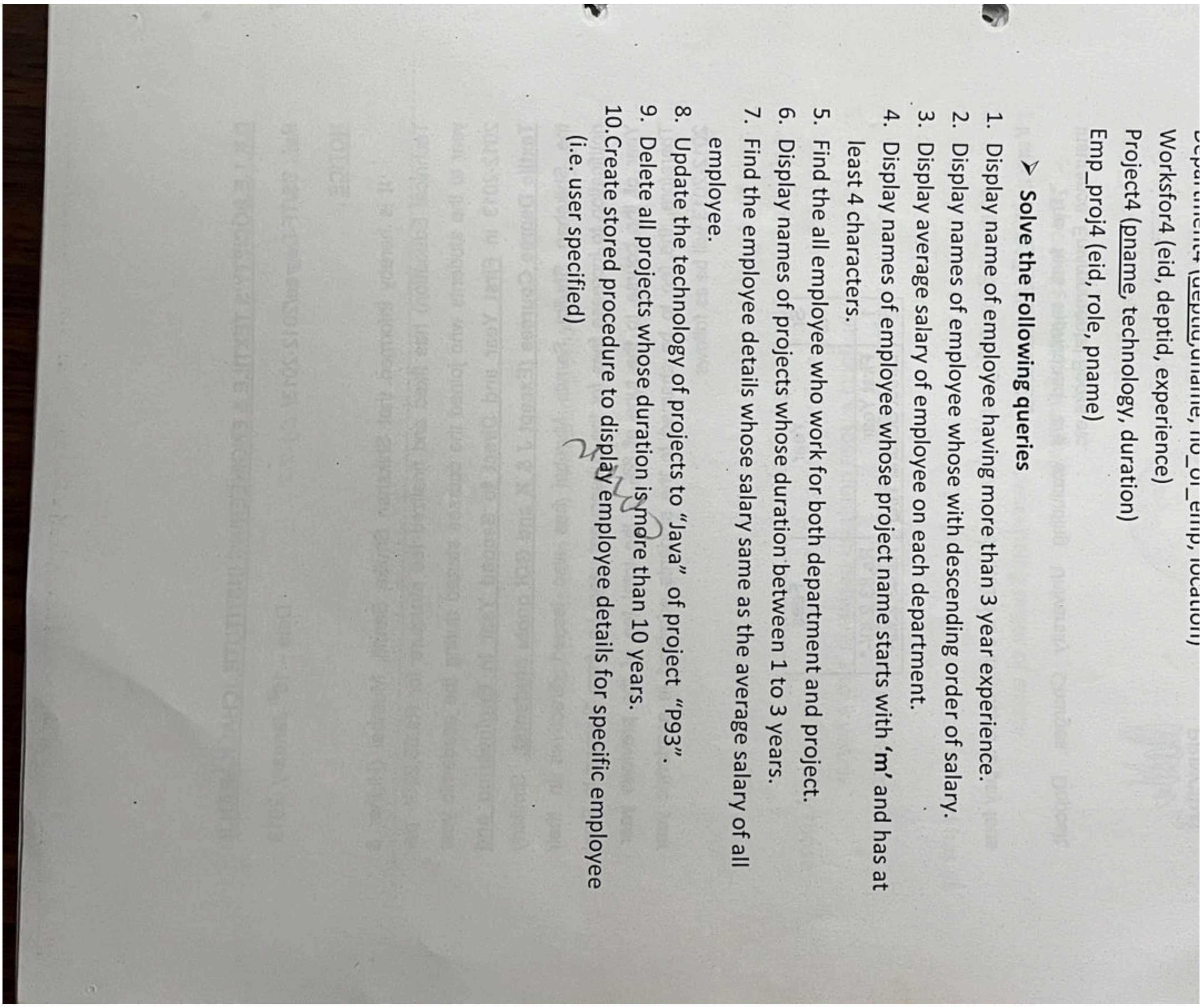
```
SELECT hotel_id, AVG(price) AS average_price
FROM rooms
GROUP BY hotel_id
HAVING AVG(price) > 2000;
```

9. **Create a view for rooms with guest details:**

```
CREATE VIEW room_guest_details AS
SELECT r.room_id, r.room_type, r.price, g.guest_name, g.address
FROM rooms r
JOIN bookings b ON r.room_id = b.room_id
JOIN guests g ON b.guest_id = g.guest_id;
```

10. **Create a stored procedure to reduce the price of a room (user-specified) for a specific type (user-specified):**

```
CREATE OR REPLACE PROCEDURE ReduceRoomPrice(
    p_room_id IN NUMBER,
    p_room_type IN VARCHAR2,
    p_reduction_amount IN NUMBER
)
IS
BEGIN
    UPDATE rooms
    SET price = price - p_reduction_amount
    WHERE room_id = p_room_id
    AND room_type = p_room_type;

    IF SQL%ROWCOUNT = 0 THEN
        DBMS_OUTPUT.PUT_LINE('No room found with the specified criteria.');
    ELSE
        DBMS_OUTPUT.PUT_LINE('Price reduced successfully.');
    END IF;
END;
```



Department4 (deptid,dname, no_of_emp, location)
Worksfor4 (eid, deptid, experience)
Project4 (pname, technology, duration)
Emp_proj4 (eid, role, pname)

➢ **Solve the Following queries**

1. Display name of employee having more than 3 year experience.
2. Display names of employee whose with descending order of salary.
3. Display average salary of employee on each department.
4. Display names of employee whose project name starts with **'m'** and has at least 4 characters.
5. Find the all employee who work for both department and project.
6. Display names of projects whose duration between 1 to 3 years.
7. Find the employee details whose salary same as the average salary of all employee.
8. Update the technology of projects to "Java" of project "P93".
9. Delete all projects whose duration is more than 10 years.
10. Create stored procedure to display employee details for specific employee (i.e. user specified)

Here are the Oracle SQL queries for your requests:

1. **Display names of employees with more than 3 years of experience:**

   ```
   SELECT employee_name
   FROM employees
   WHERE experience_years > 3;
   ```

2. **Display names of employees in descending order of salary:**

   ```
   SELECT employee_name
   FROM employees
   ORDER BY salary DESC;
   ```

3. **Display average salary of employees in each department:**

   ```
   SELECT department_id, AVG(salary) AS average_salary
   FROM employees
   GROUP BY department_id;
   ```

4. **Display names of employees whose project name starts with 'm' and has at least 4 characters:**

   ```
   SELECT e.employee_name
   FROM employees e
   JOIN projects p ON e.project_id = p.project_id
   WHERE LOWER(SUBSTR(p.project_name, 1, 1)) = 'm' AND LENGTH(p.project_name) >= 4;
   ```

5. **Find all employees who work for both a department and a project:**

```
SELECT e.employee_name
FROM employees e
JOIN departments d ON e.department_id = d.department_id
JOIN projects p ON e.project_id = p.project_id;
```

6. **Display names of projects whose duration is between 1 to 3 years:**

   ```
   SELECT project_name
   FROM projects
   WHERE duration BETWEEN 1 AND 3;
   ```

7. **Find the employee details whose salary is the same as the average salary of all employees:**

   ```
   SELECT *
   FROM employees
   WHERE salary = (SELECT AVG(salary) FROM employees);
   ```

8. **Update the technology of project "P93" to "Java":**

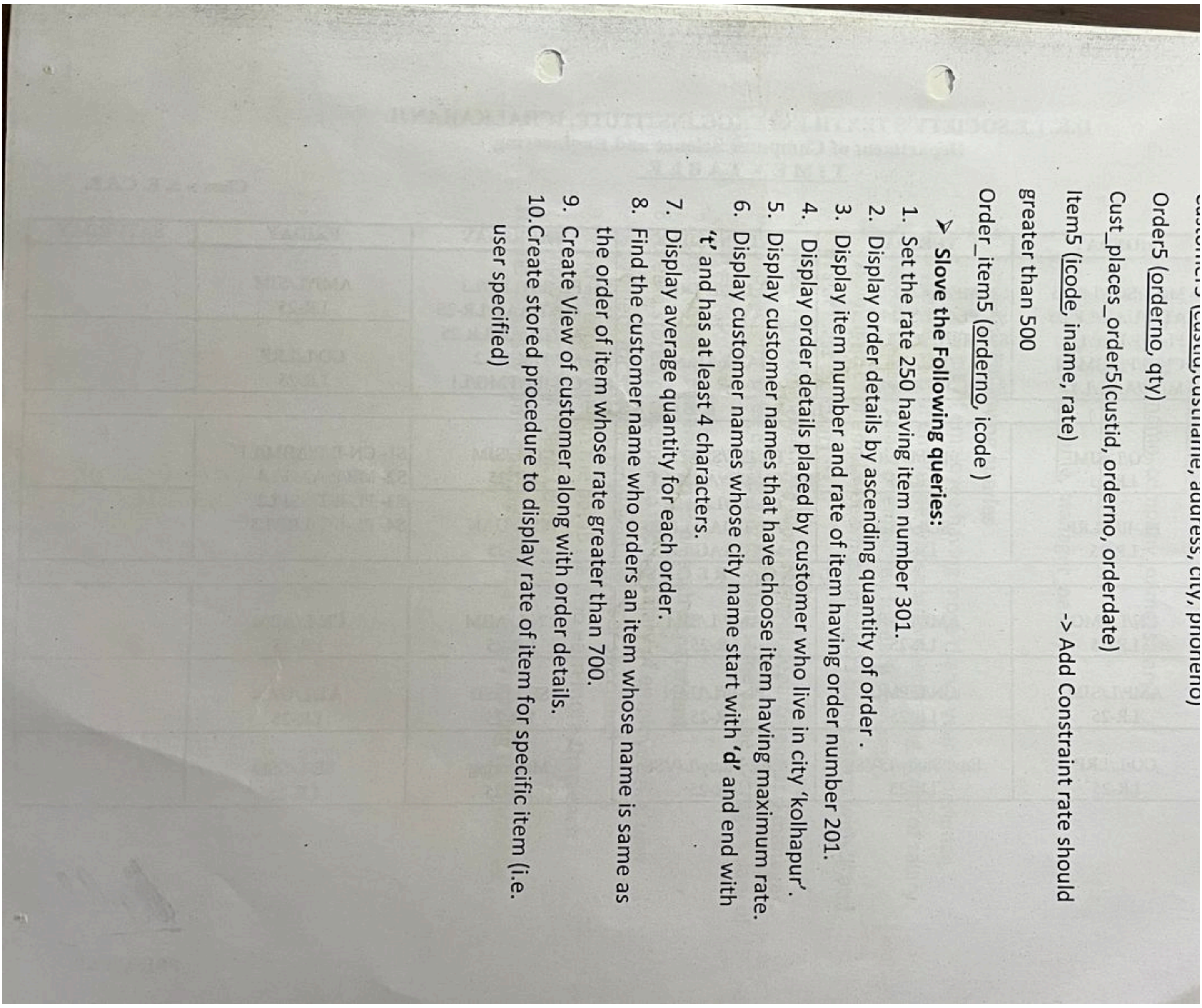
   ```
   UPDATE projects
   SET technology = 'Java'
   WHERE project_id = 'P93';
   ```

9. **Delete all projects whose duration is more than 10 years:**

   ```
   DELETE FROM projects
   WHERE duration > 10;
   ```

10. **Create a stored procedure to display employee details for a specific employee (user-specified):**

    ```
    CREATE OR REPLACE PROCEDURE GetEmployeeDetails(p_employee_id IN NUMBER)
    IS
    BEGIN
        SELECT *
        INTO :employee_details
        FROM employees
        WHERE employee_id = p_employee_id;
    EXCEPTION
        WHEN NO_DATA_FOUND THEN
            DBMS_OUTPUT.PUT_LINE('No employee found with ID: ' || p_employee_id);
    END;
    ```



(custid,custname, address, city, phoneno)

Order5 (orderno, qty)

Cust_places_order5(custid, orderno, orderdate)

Item5 (icode, iname, rate) -> Add Constraint rate should greater than 500

Order_item5 (orderno, icode)

➢ **Slove the Following queries:**

1. Set the rate 250 having item number 301.
2. Display order details by ascending quantity of order .
3. Display item number and rate of item having order number 201.
4. Display order details placed by customer who live in city ‘kolhapur’.
5. Display customer names that have choose item having maximum rate.
6. Display customer names whose city name start with **‘d’** and end with **‘t’** and has at least 4 characters.
7. Display average quantity for each order.
8. Find the customer name who orders an item whose name is same as the order of item whose rate greater than 700.
9. Create View of customer along with order details.
10. Create stored procedure to display rate of item for specific item (i.e. user specified)

Here are the Oracle SQL queries for each of your requests:

1. **Set the rate to 250 for item number 301:**

   ```
   UPDATE items
   SET rate = 250
   WHERE item_number = 301;
   ```

2. **Display order details by ascending quantity of order:**

   ```
   SELECT *
   FROM orders
   ORDER BY quantity ASC;
   ```

3. **Display item number and rate of item having order number 201:**

   ```
   SELECT i.item_number, i.rate
   FROM items i
   JOIN orders o ON i.item_number = o.item_number
   WHERE o.order_number = 201;
   ```

4. **Display order details placed by a customer who lives in city 'Kolhapur':**

   ```
   SELECT o.*
   FROM orders o
   JOIN customers c ON o.customer_id = c.customer_id
   WHERE c.city = 'Kolhapur';
   ```

5. **Display customer names that have chosen items with the maximum rate:**

```
SELECT DISTINCT c.customer_name
FROM customers c
JOIN orders o ON c.customer_id = o.customer_id
JOIN items i ON o.item_number = i.item_number
WHERE i.rate = (SELECT MAX(rate) FROM items);
```

6. **Display customer names whose city name starts with 'd' and ends with 't' and has at least 4 characters:**

```
SELECT customer_name
FROM customers
WHERE LOWER(city) LIKE 'd%' AND LOWER(city) LIKE '%t' AND LENGTH(city) >= 4;
```

7. **Display average quantity for each order:**

```
SELECT order_number, AVG(quantity) AS avg_quantity
FROM orders
GROUP BY order_number;
```

8. **Find the customer name who orders an item whose name is the same as the name of an item with a rate greater than 700:**

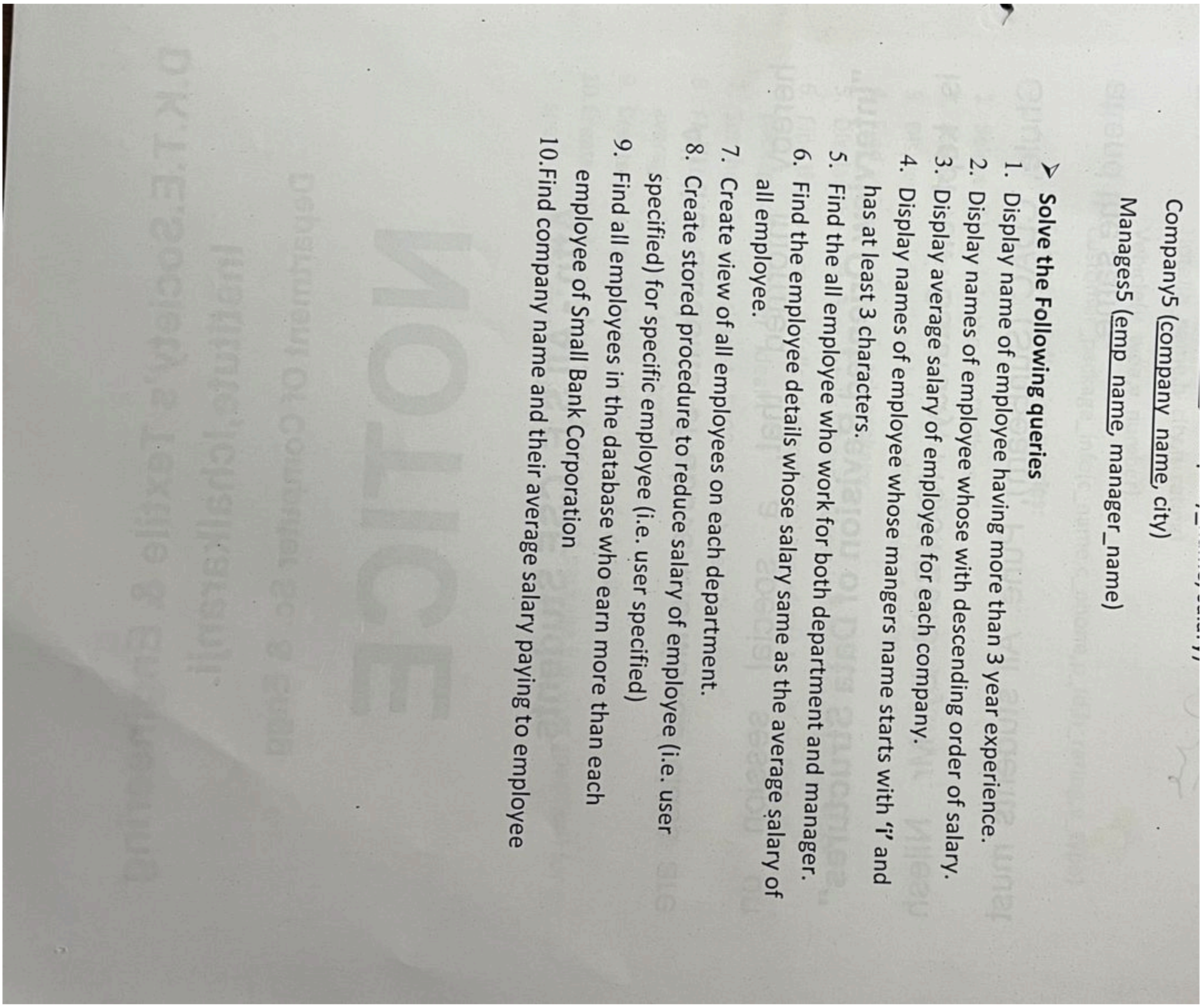
```
SELECT DISTINCT c.customer_name
FROM customers c
JOIN orders o ON c.customer_id = o.customer_id
JOIN items i1 ON o.item_number = i1.item_number
WHERE i1.item_name = (SELECT i2.item_name
                      FROM items i2
                      WHERE i2.rate > 700);
```

9. **Create a view of customer along with order details:**

```
CREATE VIEW customer_order_details AS
SELECT c.customer_name, o.order_number, o.quantity, i.item_name, i.rate
FROM customers c
JOIN orders o ON c.customer_id = o.customer_id
JOIN items i ON o.item_number = i.item_number;
```

10. **Create a stored procedure to display the rate of an item for a specific item (user-specified):**

```
CREATE OR REPLACE PROCEDURE GetItemRate(p_item_number IN NUMBER)
IS
v_rate NUMBER;
BEGIN
    SELECT rate
    INTO v_rate
    FROM items
    WHERE item_number = p_item_number;

    DBMS_OUTPUT.PUT_LINE('The rate of item ' || p_item_number || ' is: ' || v_rate);
EXCEPTION
    WHEN NO_DATA_FOUND THEN
        DBMS_OUTPUT.PUT_LINE('No item found with item number: ' || p_item_number);
END;
```



Company5 (company_name, city)

Manages5 (emp_name, manager_name)

➢ **Solve the Following queries**

1. Display name of employee having more than 3 year experience.
2. Display names of employee whose with descending order of salary.
3. Display average salary of employee for each company.
4. Display names of employee whose mangers name starts with **'i'** and has at least 3 characters.
5. Find the all employee who work for both department and manager.
6. Find the employee details whose salary same as the average salary of all employee.
7. Create view of all employees on each department.
8. Create stored procedure to reduce salary of employee (i.e. user specified) for specific employee (i.e. user specified)
9. Find all employees in the database who earn more than each employee of Small Bank Corporation
10. Find company name and their average salary paying to employee

Here are the Oracle SQL queries for your requests:

1. **Display names of employees with more than 3 years of experience:**

```
SELECT employee_name
FROM employees
WHERE experience_years > 3;
```

2. **Display names of employees in descending order of salary:**

```
SELECT employee_name
FROM employees
ORDER BY salary DESC;
```

3. **Display average salary of employees for each company:**

```
SELECT company_name, AVG(salary) AS average_salary
FROM employees
GROUP BY company_name;
```

4. **Display names of employees whose managers' names start with 'i' and have at least 3 characters:**

```
SELECT e.employee_name
FROM employees e
JOIN employees m ON e.manager_id = m.employee_id
WHERE LOWER(SUBSTR(m.employee_name, 1, 1)) = 'i' AND LENGTH(m.employee_name) >= 3;
```

5. **Find all employees who work for both a department and a manager:**

```
SELECT e.employee_name
FROM employees e
JOIN departments d ON e.department_id = d.department_id
JOIN employees m ON e.manager_id = m.employee_id;
```

6. **Find the employee details whose salary is the same as the average salary of all employees:**

```
SELECT *
FROM employees
WHERE salary = (SELECT AVG(salary) FROM employees);
```

7. **Create a view of all employees in each department:**

```
CREATE VIEW department_employees AS
SELECT e.employee_name, d.department_name
FROM employees e
JOIN departments d ON e.department_id = d.department_id;
```

8. **Create a stored procedure to reduce the salary of an employee (user-specified) for a specific employee (user-specified):**

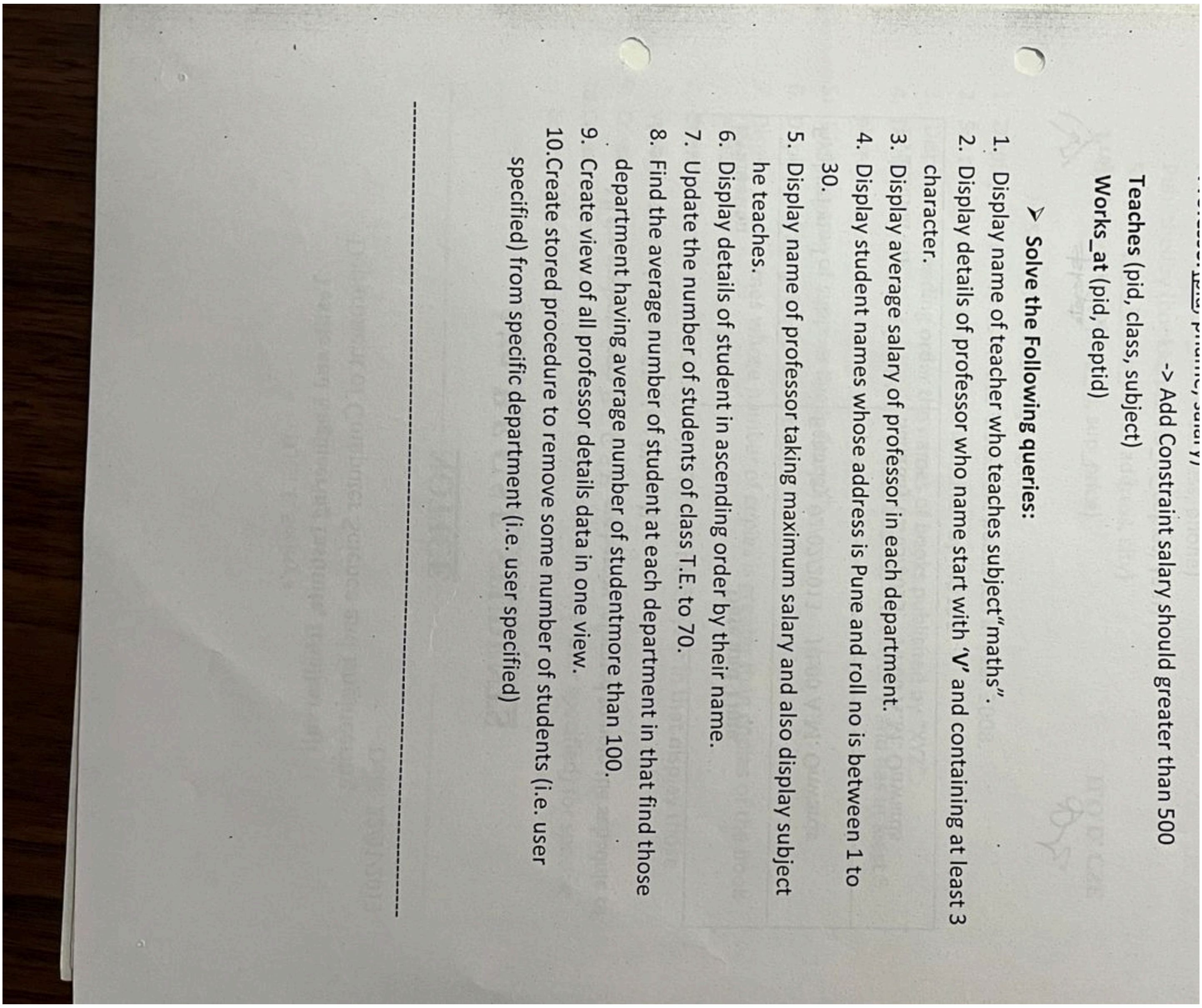
```
CREATE OR REPLACE PROCEDURE ReduceEmployeeSalary(
    p_employee_id IN NUMBER,
    p_reduction_amount IN NUMBER
)
IS
BEGIN
    UPDATE employees
    SET salary = salary - p_reduction_amount
    WHERE employee_id = p_employee_id;

    IF SQL%ROWCOUNT = 0 THEN
        DBMS_OUTPUT.PUT_LINE('No employee found with ID: ' || p_employee_id);
    ELSE
        DBMS_OUTPUT.PUT_LINE('Salary reduced successfully.');
    END IF;
END;
```

9. **Find all employees in the database who earn more than each employee of Small Bank Corporation:**

```
SELECT employee_name
FROM employees e
WHERE e.salary > ALL (SELECT salary FROM employees WHERE company_name = 'Small Bank Corporation');
```

10. **Find the company name and their average salary paying to employees:**

```
SELECT company_name, AVG(salary) AS average_salary
FROM employees
GROUP BY company_name;
```



->Add Constraint salary should greater than 500

Teaches (pid, class, subject)

Works_at (pid, deptid)

➢ Solve the Following queries:

1. Display name of teacher who teaches subject"maths".
2. Display details of professor who name start with **'V'** and containing at least 3 character.
3. Display average salary of professor in each department.
4. Display student names whose address is Pune and roll no is between 1 to 30.
5. Display name of professor taking maximum salary and also display subject he teaches.
6. Display details of student in ascending order by their name.
7. Update the number of students of class T.E. to 70.
8. Find the average number of student at each department in that find those department having average number of studentmore than 100.
9. Create view of all professor details data in one view.
10. Create stored procedure to remove some number of students (i.e. user specified) from specific department (i.e. user specified)

Here are the Oracle SQL queries for your requests:

1. **Display the name of the professor who teaches the subject "Maths":**

```
SELECT professor_name
FROM professors
WHERE subject = 'Maths';
```

2. **Display details of the professor whose name starts with 'V' and contains at least 3 characters:**

```
SELECT *
FROM professors
WHERE LOWER(SUBSTR(professor_name, 1, 1)) = 'v' AND LENGTH(professor_name) >= 3;
```

3. **Display the average salary of professors in each department:**

```
SELECT department_id, AVG(salary) AS average_salary
FROM professors
GROUP BY department_id;
```

4. **Display student names whose address is Pune and roll number is between 1 and 30:**

```
SELECT student_name
FROM students
WHERE address = 'Pune' AND roll_number BETWEEN 1 AND 30;
```

5. **Display the name of the professor taking maximum salary and also display the subject he teaches:**

```
SELECT professor_name, subject
FROM professors
```

```
WHERE salary = (SELECT MAX(salary) FROM professors);
```

6. **Display details of students in ascending order by their name:**

```
SELECT *
FROM students
ORDER BY student_name ASC;
```

7. **Update the number of students in class T.E. to 70:**

```
UPDATE classes
SET number_of_students = 70
WHERE class_name = 'T.E.';
```

8. **Find the average number of students in each department. Then, find those departments having an average number of students more than 100:**

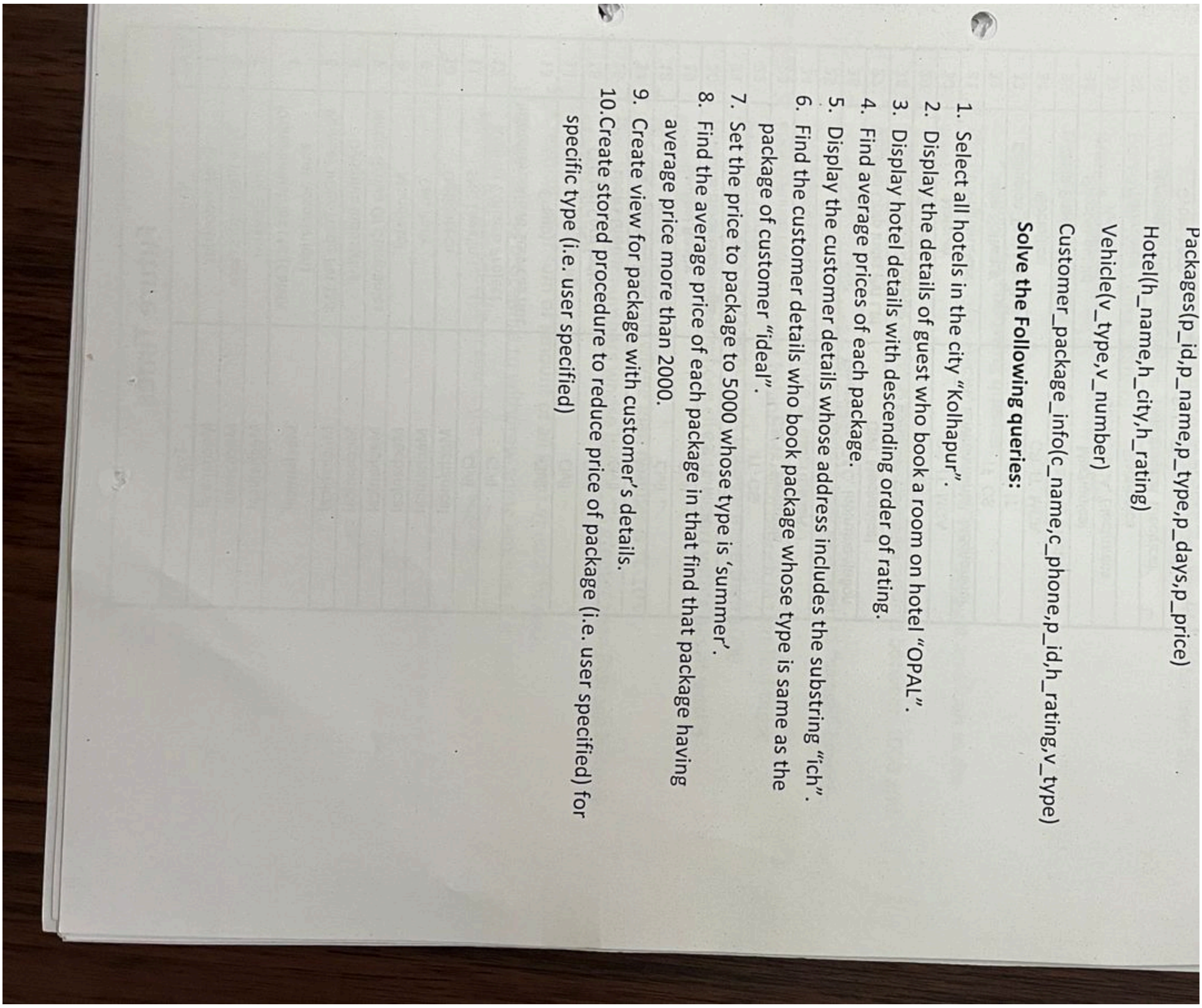
```
SELECT department_id, AVG(number_of_students) AS avg_students
FROM classes
GROUP BY department_id
HAVING AVG(number_of_students) > 100;
```

9. **Create a view of all professor details in one view:**

```
CREATE VIEW professor_details_view AS
SELECT *
FROM professors;
```

10. **Create a stored procedure to remove some number of students (user-specified) from a specific department (user-specified):**

```
CREATE OR REPLACE PROCEDURE RemoveStudents(
    p_department_id IN NUMBER,
    p_remove_count IN NUMBER
)
IS
BEGIN
    UPDATE classes
    SET number_of_students = number_of_students - p_remove_count
    WHERE department_id = p_department_id;

    IF SQL%ROWCOUNT = 0 THEN
        DBMS_OUTPUT.PUT_LINE('No classes found for department: ' || p_department_id);
    ELSE
        DBMS_OUTPUT.PUT_LINE(p_remove_count || ' students removed from department: ' ||
p_department_id);
    END IF;
END;
```



Packages(p_id,p_name,p_type,p_days,p_price)

Hotel(h_name,h_city,h_rating)

Vehicle(v_type,v_number)

Customer_package_info(c_name,c_phone,p_id,h_rating,v_type)

Solve the Following queries:

1. Select all hotels in the city "Kolhapur".
2. Display the details of guest who book a room on hotel "OPAL".
3. Display hotel details with descending order of rating.
4. Find average prices of each package.
5. Display the customer details whose address includes the substring "ich".
6. Find the customer details who book package whose type is same as the package of customer "ideal".
7. Set the price to package to 5000 whose type is 'summer'.
8. Find the average price of each package in that find that package having average price more than 2000.
9. Create view for package with customer's details.
10. Create stored procedure to reduce price of package (i.e. user specified) for specific type (i.e. user specified)

Here are the Oracle SQL queries for your requests:

1. **Select all hotels in the city "Kolhapur":**

   ```
   SELECT *
   FROM hotels
   WHERE city = 'Kolhapur';
   ```

2. **Display the details of guests who booked a room in hotel "OPAL":**

   ```
   SELECT g.*
   FROM guests g
   JOIN bookings b ON g.guest_id = b.guest_id
   JOIN hotels h ON b.hotel_id = h.hotel_id
   WHERE h.hotel_name = 'OPAL';
   ```

3. **Display hotel details with descending order of rating:**

   ```
   SELECT *
   FROM hotels
   ORDER BY rating DESC;
   ```

4. **Find average prices of each package:**

   ```
   SELECT package_type, AVG(price) AS avg_price
   FROM packages
   GROUP BY package_type;
   ```

5. **Display the customer details whose address includes the substring "ich":**

```
SELECT *
FROM customers
WHERE LOWER(address) LIKE '%ich%';
```

6. **Find the customer details who booked packages of the same type as the package of customer "ideal":**

```
SELECT c.*
FROM customers c
JOIN bookings b ON c.customer_id = b.customer_id
JOIN packages p ON b.package_id = p.package_id
WHERE p.package_type = (SELECT p2.package_type
                        FROM customers c2
                        JOIN bookings b2 ON c2.customer_id = b2.customer_id
                        JOIN packages p2 ON b2.package_id = p2.package_id
                        WHERE c2.customer_name = 'ideal');
```

7. **Set the price of a package to 5000 whose type is "summer":**

```
UPDATE packages
SET price = 5000
WHERE package_type = 'summer';
```

8. **Find the average price of each package and then find those packages having an average price more than 2000:**

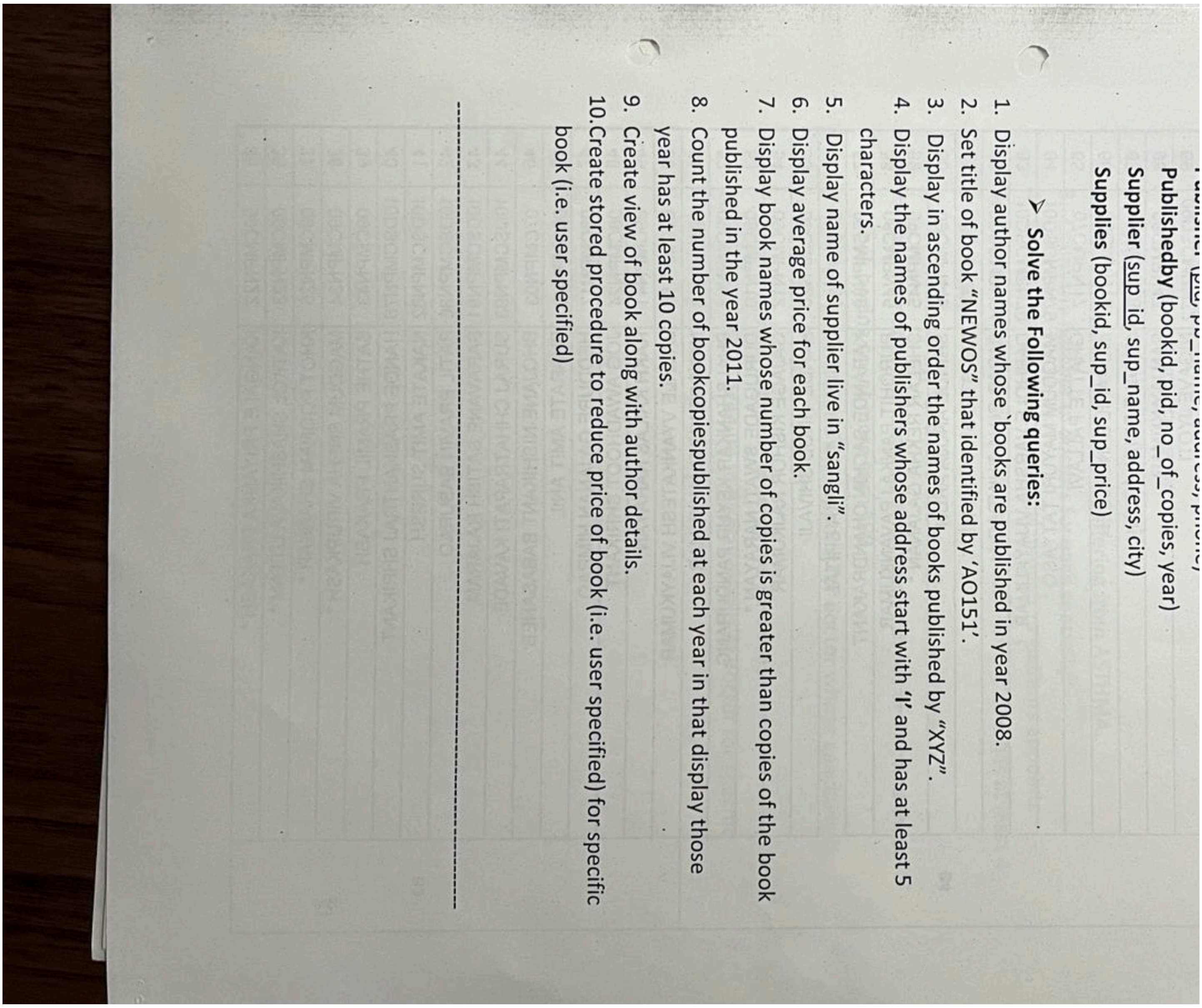
```
SELECT package_type, AVG(price) AS avg_price
FROM packages
GROUP BY package_type
HAVING AVG(price) > 2000;
```

9. **Create a view for packages with customer details:**

```
CREATE VIEW package_customer_details AS
SELECT p.package_type, p.price, c.customer_name, c.address
FROM packages p
JOIN bookings b ON p.package_id = b.package_id
JOIN customers c ON b.customer_id = c.customer_id;
```

10. **Create a stored procedure to reduce the price of a package (user-specified) for a specific type (user-specified):**

```
CREATE OR REPLACE PROCEDURE ReducePackagePrice(
    p_package_type IN VARCHAR2,
    p_reduction_amount IN NUMBER
)
IS
BEGIN
    UPDATE packages
    SET price = price - p_reduction_amount
    WHERE package_type = p_package_type;

    IF SQL%ROWCOUNT = 0 THEN
        DBMS_OUTPUT.PUT_LINE('No package found with the specified type: ' || p_package_type);
    ELSE
        DBMS_OUTPUT.PUT_LINE('Price reduced for package type: ' || p_package_type);
    END IF;
END;
```



Publishedby (bookid, pid, no_of_copies, year)

Supplier (sup_id, sup_name, address, city)

Supplies (bookid, sup_id, sup_price)

➢ **Solve the Following queries:**

1. Display author names whose `books are published in year 2008.
2. Set title of book "NEWOS" that identified by 'AO151'.
3. Display in ascending order the names of books published by "XYZ".
4. Display the names of publishers whose address start with '**I**' and has at least 5 characters.
5. Display name of supplier live in "sangli".
6. Display average price for each book.
7. Display book names whose number of copies is greater than copies of the book published in the year 2011.
8. Count the number of bookcopiespublished at each year in that display those year has at least 10 copies.
9. Create view of book along with author details.
10. Create stored procedure to reduce price of book (i.e. user specified) for specific book (i.e. user specified)

Here are the Oracle SQL queries for your requests:

1. **Display author names whose books were published in the year 2008:**

   ```
   SELECT a.author_name
   FROM authors a
   JOIN books b ON a.author_id = b.author_id
   WHERE b.publication_year = 2008;
   ```

2. **Set the title of the book "NEWOS" identified by "A0151":**

   ```
   UPDATE books
   SET title = 'NEWOS'
   WHERE book_id = 'A0151';
   ```

3. **Display in ascending order the names of books published by "XYZ":**

   ```
   SELECT b.title
   FROM books b
   JOIN publishers p ON b.publisher_id = p.publisher_id
   WHERE p.publisher_name = 'XYZ'
   ORDER BY b.title ASC;
   ```

4. **Display the names of publishers whose addresses start with 'I' and have at least 5 characters:**

   ```
   SELECT publisher_name
   FROM publishers
   WHERE LOWER(address) LIKE 'i%' AND LENGTH(address) >= 5;
   ```

5. **Display the name of the supplier living in "Sangli":**

```
SELECT supplier_name
FROM suppliers
WHERE city = 'Sangli';
```

6. **Display the average price for each book:**

```
SELECT title, AVG(price) AS avg_price
FROM books
GROUP BY title;
```

7. **Display book names whose number of copies is greater than the number of copies of the book published in the year 2011:**

```
SELECT title
FROM books
WHERE number_of_copies > (SELECT MAX(number_of_copies)
                          FROM books
                          WHERE publication_year = 2011);
```

8. **Count the number of book copies published each year. Then, display those years that have at least 10 copies:**

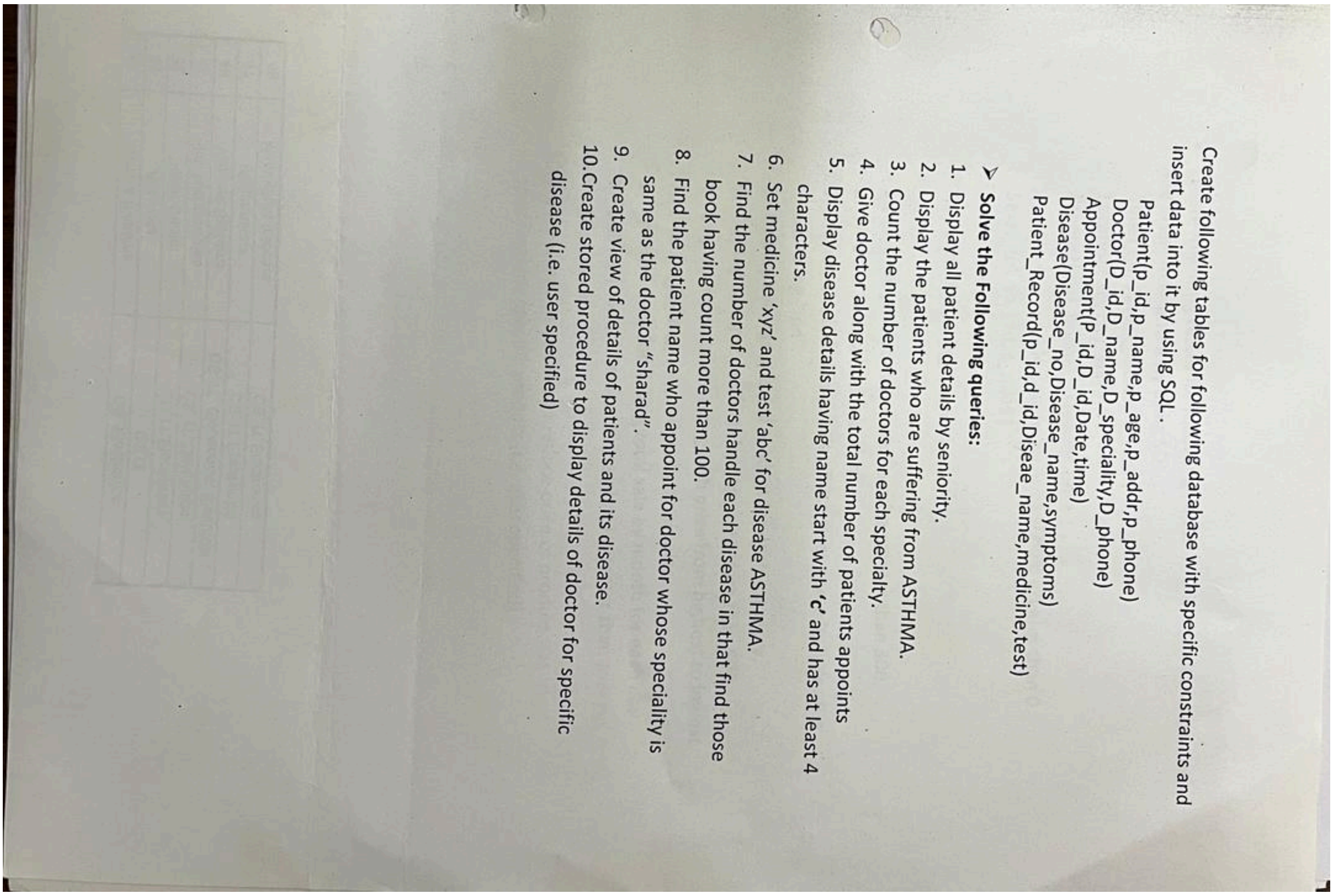
```
SELECT publication_year, SUM(number_of_copies) AS total_copies
FROM books
GROUP BY publication_year
HAVING SUM(number_of_copies) >= 10;
```

9. **Create a view of books along with author details:**

```
CREATE VIEW book_author_details AS
SELECT b.title, b.publication_year, a.author_name, a.author_bio
FROM books b
JOIN authors a ON b.author_id = a.author_id;
```

10. **Create a stored procedure to reduce the price of a book (user-specified) for a specific book (user-specified):**

```
CREATE OR REPLACE PROCEDURE ReduceBookPrice(
    p_book_id IN VARCHAR2,
    p_reduction_amount IN NUMBER
)
IS
BEGIN
    UPDATE books
    SET price = price - p_reduction_amount
    WHERE book_id = p_book_id;

    IF SQL%ROWCOUNT = 0 THEN
        DBMS_OUTPUT.PUT_LINE('No book found with ID: ' || p_book_id);
    ELSE
        DBMS_OUTPUT.PUT_LINE('Price reduced for book ID: ' || p_book_id);
    END IF;
END;
```



Create following tables for following database with specific constraints and insert data into it by using SQL.

Patient(p_id,p_name,p_age,p_addr,p_phone)
Doctor(D_id,D_name,D_speciality,D_phone)
Appointment(P_id,D_id,Date,time)
Disease(Disease_no,Disease_name,symptoms)
Patient_Record(p_id,d_id,Diseae_name,medicine,test)

➢ **Solve the Following queries:**

1. Display all patient details by seniority.
2. Display the patients who are suffering from ASTHMA.
3. Count the number of doctors for each specialty.
4. Give doctor along with the total number of patients appoints
5. Display disease details having name start with **'c'** and has at least 4 characters.
6. Set medicine 'xyz' and test 'abc' for disease ASTHMA.
7. Find the number of doctors handle each disease in that find those book having count more than 100.
8. Find the patient name who appoint for doctor whose speciality is same as the doctor "sharad".
9. Create view of details of patients and its disease.
10. Create stored procedure to display details of doctor for specific disease (i.e. user specified)

Here are the Oracle SQL queries for your requests:

1. **Display all patient details by seniority (assuming seniority is based on age):**

   ```
   SELECT *
   FROM patients
   ORDER BY age DESC;
   ```

2. **Display the patients who are suffering from ASTHMA:**

   ```
   SELECT *
   FROM patients
   WHERE disease = 'ASTHMA';
   ```

3. **Count the number of doctors for each specialty:**

   ```
   SELECT specialty, COUNT(doctor_id) AS doctor_count
   FROM doctors
   GROUP BY specialty;
   ```

4. **Give doctor along with the total number of patients' appointments:**

   ```
   SELECT d.doctor_name, COUNT(a.appointment_id) AS total_appointments
   FROM doctors d
   JOIN appointments a ON d.doctor_id = a.doctor_id
   GROUP BY d.doctor_name;
   ```

5. **Display disease details having a name that starts with 'c' and has at least 4 characters:**

   ```
   SELECT *
   FROM diseases
   WHERE LOWER(disease_name) LIKE 'c%' AND LENGTH(disease_name) >= 4;
   ```

6. **Set medicine 'xyz' and test 'abc' for disease ASTHMA:**

   ```
   UPDATE diseases
   SET medicine = 'xyz', test = 'abc'
   WHERE disease_name = 'ASTHMA';
   ```

7. **Find the number of doctors handling each disease. Then, find those diseases having a count more than 100:**

   ```
   SELECT d.disease_name, COUNT(doc.doctor_id) AS doctor_count
   FROM diseases d
   JOIN doctor_disease_mapping ddm ON d.disease_id = ddm.disease_id
   JOIN doctors doc ON ddm.doctor_id = doc.doctor_id
   GROUP BY d.disease_name
   HAVING COUNT(doc.doctor_id) > 100;
   ```

8. **Find the patient name who appoints a doctor whose specialty is the same as the doctor "sharad":**

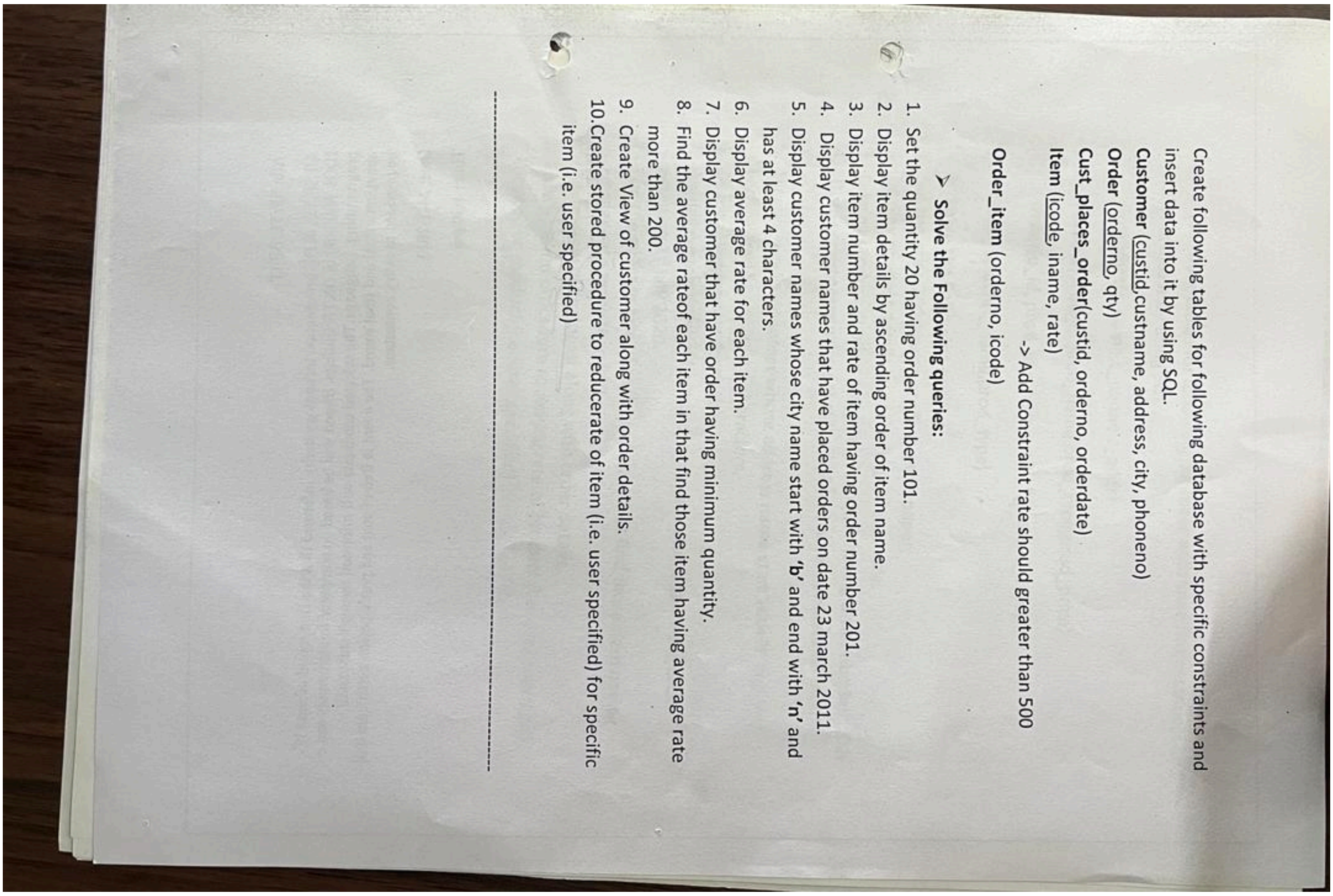
   ```
   SELECT p.patient_name
   FROM patients p
   JOIN appointments a ON p.patient_id = a.patient_id
   JOIN doctors d ON a.doctor_id = d.doctor_id
   WHERE d.specialty = (SELECT specialty
                        FROM doctors
                        WHERE doctor_name = 'sharad');
   ```

9. **Create a view of details of patients and their diseases:**

   ```
   CREATE VIEW patient_disease_details AS
   SELECT p.patient_name, p.age, p.address, d.disease_name, d.medicine, d.test
   FROM patients p
   JOIN diseases d ON p.disease_id = d.disease_id;
   ```

10. **Create a stored procedure to display details of a doctor for a specific disease (user-specified):**

    ```
    CREATE OR REPLACE PROCEDURE GetDoctorDetailsForDisease(
        p_disease_name IN VARCHAR2
    )
    IS
    BEGIN
        SELECT d.doctor_name, d.specialty, d.experience_years
        FROM doctors d
        JOIN doctor_disease_mapping ddm ON d.doctor_id = ddm.doctor_id
        JOIN diseases dis ON ddm.disease_id = dis.disease_id
        WHERE dis.disease_name = p_disease_name;

        IF SQL%ROWCOUNT = 0 THEN
            DBMS_OUTPUT.PUT_LINE('No doctor found for the disease: ' || p_disease_name);
        END IF;
    END;
    ```



Create following tables for following database with specific constraints and insert data into it by using SQL.

Customer (custid,custname, address, city, phoneno)
Order (orderno, qty)
Cust_places_order(custid, orderno, orderdate)
Item (icode, iname, rate)
-> Add Constraint rate should greater than 500
Order_item (orderno, icode)

➢ **Solve the Following queries:**

1. Set the quantity 20 having order number 101.
2. Display item details by ascending order of item name.
3. Display item number and rate of item having order number 201.
4. Display customer names that have placed orders on date 23 march 2011.
5. Display customer names whose city name start with **'b'** and end with **'n'** and has at least 4 characters.
6. Display average rate for each item.
7. Display customer that have order having minimum quantity.
8. Find the average rateof each item in that find those item having average rate more than 200.
9. Create View of customer along with order details.
10. Create stored procedure to reducerate of item (i.e. user specified) for specific item (i.e. user specified)

Here are the Oracle SQL queries for your requests:

1. **Set the quantity to 20 for order number 101:**

   ```
   UPDATE orders
   SET quantity = 20
   WHERE order_number = 101;
   ```

2. **Display item details by ascending order of item name:**

   ```
   SELECT *
   FROM items
   ORDER BY item_name ASC;
   ```

3. **Display item number and rate of item having order number 201:**

   ```
   SELECT i.item_number, i.rate
   FROM items i
   JOIN orders o ON i.item_number = o.item_number
   WHERE o.order_number = 201;
   ```

4. **Display customer names that have placed orders on date 23 March 2011:**

   ```
   SELECT c.customer_name
   FROM customers c
   JOIN orders o ON c.customer_id = o.customer_id
   WHERE o.order_date = TO_DATE('23-MAR-2011', 'DD-MON-YYYY');
   ```

5. **Display customer names whose city name starts with 'b' and ends with 'n' and has at least 4 characters:**

   ```
   SELECT customer_name
   FROM customers
   WHERE LOWER(city) LIKE 'b%n' AND LENGTH(city) >= 4;
   ```

6. **Display average rate for each item:**

   ```
   SELECT item_name, AVG(rate) AS avg_rate
   FROM items
   GROUP BY item_name;
   ```

7. **Display customers that have orders with the minimum quantity:**

```
SELECT c.customer_name
FROM customers c
JOIN orders o ON c.customer_id = o.customer_id
WHERE o.quantity = (SELECT MIN(quantity) FROM orders);
```

8. **Find the average rate of each item and then find those items having an average rate more than 200:**

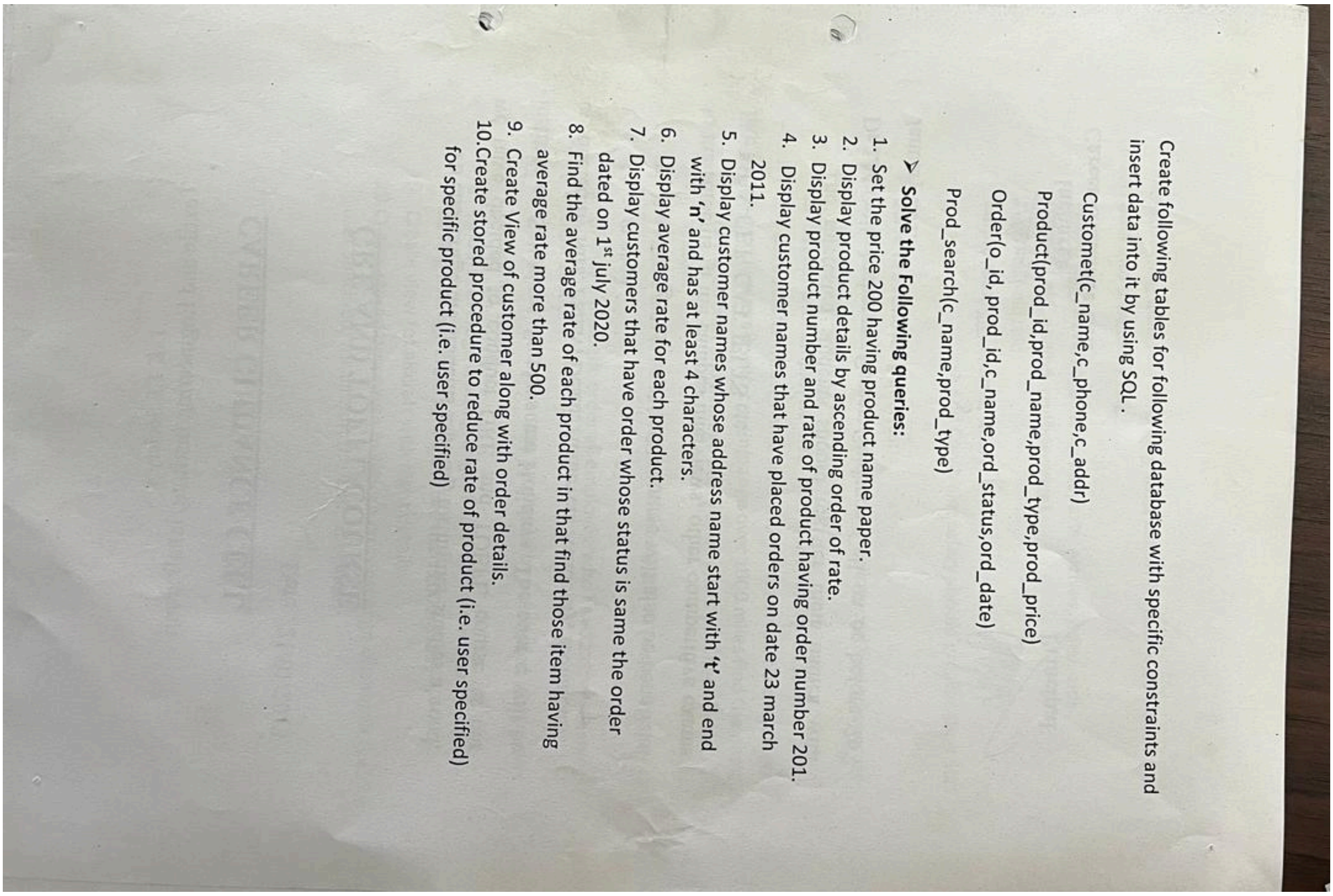
```
SELECT item_name, AVG(rate) AS avg_rate
FROM items
GROUP BY item_name
HAVING AVG(rate) > 200;
```

9. **Create a view of customer along with order details:**

```
CREATE VIEW customer_order_details AS
SELECT c.customer_name, o.order_number, o.order_date, o.quantity, i.item_name, i.rate
FROM customers c
JOIN orders o ON c.customer_id = o.customer_id
JOIN items i ON o.item_number = i.item_number;
```

10. **Create a stored procedure to reduce the rate of an item (user-specified) for a specific item (user-specified):**

```
CREATE OR REPLACE PROCEDURE ReduceItemRate(
    p_item_number IN VARCHAR2,
    p_reduction_amount IN NUMBER
)
IS
BEGIN
    UPDATE items
    SET rate = rate - p_reduction_amount
    WHERE item_number = p_item_number;

    IF SQL%ROWCOUNT = 0 THEN
        DBMS_OUTPUT.PUT_LINE('No item found with number: ' || p_item_number);
    ELSE
        DBMS_OUTPUT.PUT_LINE('Rate reduced for item number: ' || p_item_number);
    END IF;
END;
```



Create following tables for following database with specific constraints and insert data into it by using SQL.

Customet(c_name,c_phone,c_addr)
Product(prod_id,prod_name,prod_type,prod_price)
Order(o_id, prod_id,c_name,ord_status,ord_date)
Prod_search(c_name,prod_type)

➢ **Solve the Following queries:**

1. Set the price 200 having product name paper.
2. Display product details by ascending order of rate.
3. Display product number and rate of product having order number 201.
4. Display customer names that have placed orders on date 23 march 2011.
5. Display customer names whose address name start with **'t'** and end with **'n'** and has at least 4 characters.
6. Display average rate for each product.
7. Display customers that have order whose status is same the order dated on 1st july 2020.
8. Find the average rate of each product in that find those item having average rate more than 500.
9. Create View of customer along with order details.
10. Create stored procedure to reduce rate of product (i.e. user specified) for specific product (i.e. user specified)

Here are the Oracle SQL queries for your requests:

1. **Set the price to 200 for the product named "paper":**

   ```
   UPDATE products
   SET price = 200
   WHERE product_name = 'paper';
   ```

2. **Display product details by ascending order of rate:**

   ```
   SELECT *
   FROM products
   ORDER BY rate ASC;
   ```

3. **Display product number and rate of the product having order number 201:**

   ```
   SELECT p.product_number, p.rate
   FROM products p
   JOIN orders o ON p.product_number = o.product_number
   WHERE o.order_number = 201;
   ```

4. **Display customer names that have placed orders on date 23 March 2011:**

   ```
   SELECT c.customer_name
   FROM customers c
   JOIN orders o ON c.customer_id = o.customer_id
   WHERE o.order_date = TO_DATE('23-MAR-2011', 'DD-MON-YYYY');
   ```

5. **Display customer names whose address names start with 't' and end with 'n' and have at least 4 characters:**

   ```
   SELECT customer_name
   FROM customers
   WHERE LOWER(address) LIKE 't%n' AND LENGTH(address) >= 4;
   ```

6. **Display average rate for each product:**

   ```
   SELECT product_name, AVG(rate) AS avg_rate
   FROM products
   GROUP BY product_name;
   ```

7. **Display customers that have orders whose status is the same as the order dated on 1 July 2020:**

```
SELECT DISTINCT c.customer_name
FROM customers c
JOIN orders o ON c.customer_id = o.customer_id
WHERE o.status = (SELECT status
                  FROM orders
                  WHERE order_date = TO_DATE('01-JUL-2020', 'DD-MON-YYYY'));
```

8. **Find the average rate of each product and then find those products having an average rate more than 500:**

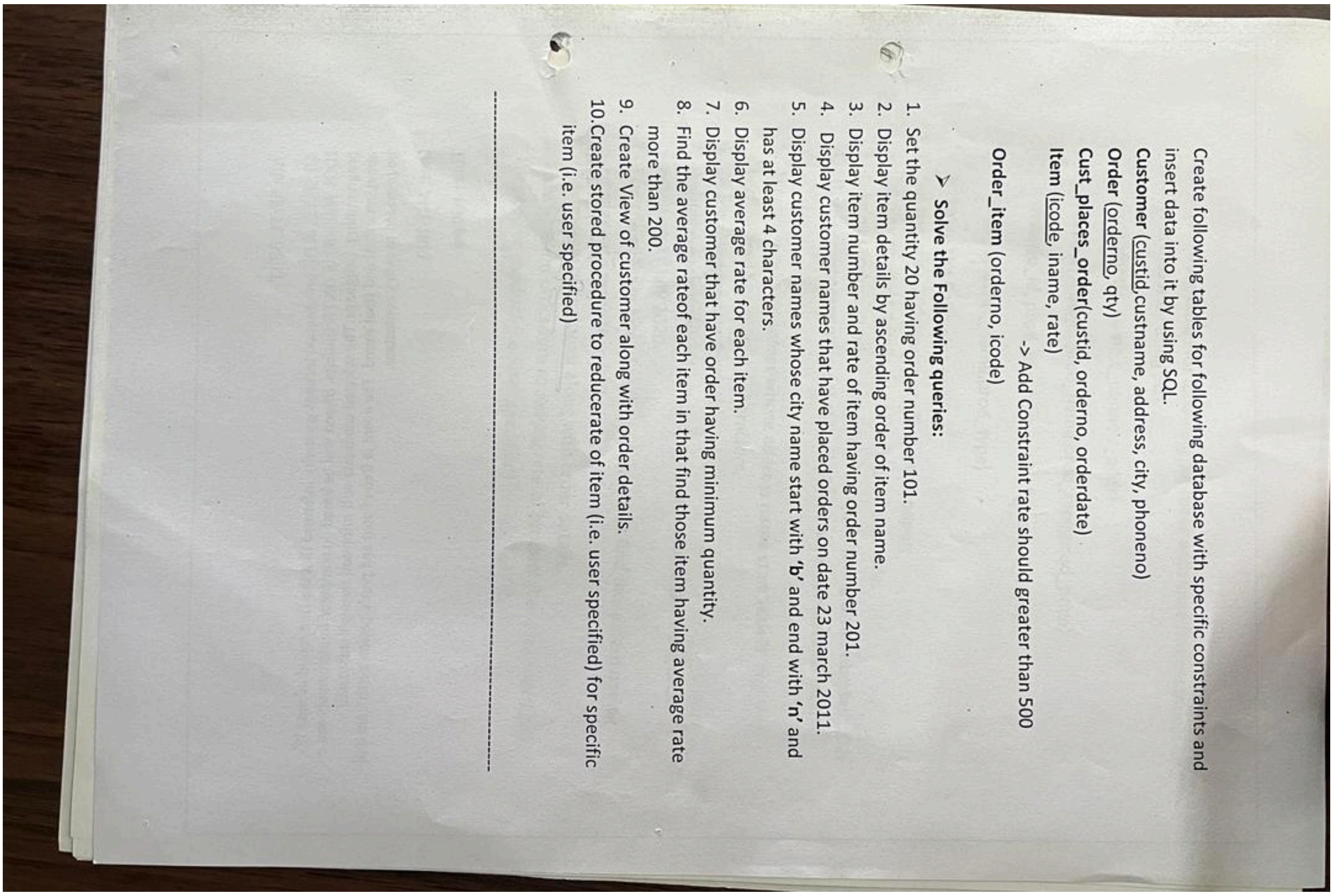
```
SELECT product_name, AVG(rate) AS avg_rate
FROM products
GROUP BY product_name
HAVING AVG(rate) > 500;
```

9. **Create a view of customer along with order details:**

```
CREATE VIEW customer_order_details AS
SELECT c.customer_name, o.order_number, o.order_date, o.status, p.product_name, p.rate
FROM customers c
JOIN orders o ON c.customer_id = o.customer_id
JOIN products p ON o.product_number = p.product_number;
```

10. **Create a stored procedure to reduce the rate of a product (user-specified) for a specific product (user-specified):**

```
CREATE OR REPLACE PROCEDURE ReduceProductRate(
    p_product_number IN VARCHAR2,
    p_reduction_amount IN NUMBER
)
IS
BEGIN
    UPDATE products
    SET rate = rate - p_reduction_amount
    WHERE product_number = p_product_number;

    IF SQL%ROWCOUNT = 0 THEN
        DBMS_OUTPUT.PUT_LINE('No product found with number: ' || p_product_number);
    ELSE
        DBMS_OUTPUT.PUT_LINE('Rate reduced for product number: ' || p_product_number);
    END IF;
END;
```



Create following tables for following database with specific constraints and insert data into it by using SQL.

Customer (custid,custname, address, city, phoneno)
Order (orderno, qty)
Cust_places_order(custid, orderno, orderdate)
Item (icode, iname, rate)
-> Add Constraint rate should greater than 500
Order_item (orderno, icode)

- **Solve the Following queries:**

1. Set the quantity 20 having order number 101.
2. Display item details by ascending order of item name.
3. Display item number and rate of item having order number 201.
4. Display customer names that have placed orders on date 23 march 2011.
5. Display customer names whose city name start with **'b'** and end with **'n'** and has at least 4 characters.
6. Display average rate for each item.
7. Display customer that have order having minimum quantity.
8. Find the average rateof each item in that find those item having average rate more than 200.
9. Create View of customer along with order details.
10. Create stored procedure to reducerate of item (i.e. user specified) for specific item (i.e. user specified)

Here are the Oracle SQL queries tailored to your requests:

1. **Set the quantity to 20 for order number 101:**

```
UPDATE orders
SET quantity = 20
WHERE order_number = 101;
```

2. **Display item details by ascending order of item name:**

```
SELECT *
FROM items
ORDER BY item_name ASC;
```

3. **Display item number and rate of item having order number 201:**

```
SELECT i.item_number, i.rate
FROM items i
JOIN orders o ON i.item_number = o.item_number
WHERE o.order_number = 201;
```

4. **Display customer names that have placed orders on date 23 March 2011:**

```
SELECT c.customer_name
FROM customers c
JOIN orders o ON c.customer_id = o.customer_id
WHERE o.order_date = TO_DATE('23-MAR-2011', 'DD-MON-YYYY');
```

5. **Display customer names whose city name starts with 'b' and ends with 'n' and has at least 4 characters:**

```
SELECT customer_name
FROM customers
WHERE LOWER(city) LIKE 'b%n' AND LENGTH(city) >= 4;
```

6. **Display average rate for each item:**

```
SELECT item_name, AVG(rate) AS avg_rate
FROM items
GROUP BY item_name;
```

7. **Display customers that have orders having minimum quantity:**

   ```
   SELECT c.customer_name
   FROM customers c
   JOIN orders o ON c.customer_id = o.customer_id
   WHERE o.quantity = (SELECT MIN(quantity) FROM orders);
   ```

8. **Find the average rate of each item and then find those items having an average rate more than 200:**

   ```
   SELECT item_name, AVG(rate) AS avg_rate
   FROM items
   GROUP BY item_name
   HAVING AVG(rate) > 200;
   ```

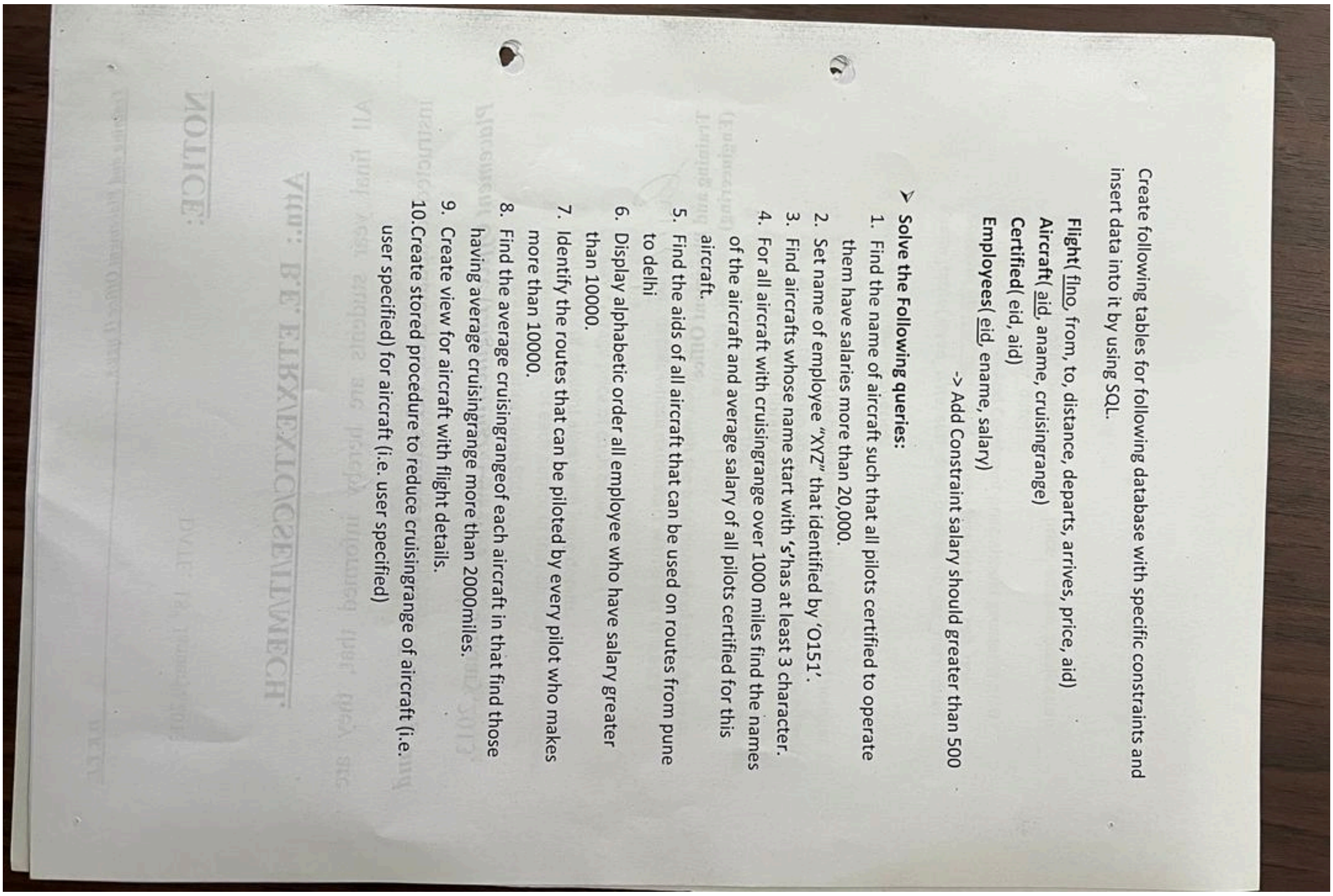
9. **Create a view of customer along with order details:**

   ```
   CREATE VIEW customer_order_details AS
   SELECT c.customer_name, o.order_number, o.order_date, o.quantity, i.item_name, i.rate
   FROM customers c
   JOIN orders o ON c.customer_id = o.customer_id
   JOIN items i ON o.item_number = i.item_number;
   ```

10. **Create a stored procedure to reduce the rate of an item (user-specified) for a specific item (user-specified):**

    ```
    CREATE OR REPLACE PROCEDURE ReduceItemRate(
        p_item_number IN VARCHAR2,
        p_reduction_amount IN NUMBER
    )
    IS
    BEGIN
        UPDATE items
        SET rate = rate - p_reduction_amount
        WHERE item_number = p_item_number;

        IF SQL%ROWCOUNT = 0 THEN
            DBMS_OUTPUT.PUT_LINE('No item found with number: ' || p_item_number);
        ELSE
            DBMS_OUTPUT.PUT_LINE('Rate reduced for item number: ' || p_item_number);
        END IF;
    END;
    ```

These queries cover the tasks you've specified, ensuring proper use of SQL syntax and logical conditions. If you need any further adjustments or explanations, feel free to ask!



Create following tables for following database with specific constraints and insert data into it by using SQL.

Flight(fino, from, to, distance, departs, arrives, price, aid)
Aircraft(aid, aname, cruisingrange)
Certified(eid, aid)
Employees(eid, ename, salary)
->Add Constraint salary should greater than 500

➢ **Solve the Following queries:**

1. Find the name of aircraft such that all pilots certified to operate them have salaries more than 20,000.
2. Set name of employee "XYZ" that identified by '0151'.
3. Find aircrafts whose name start with 's'has at least 3 character.
4. For all aircraft with cruisingrange over 1000 miles find the names of the aircraft and average salary of all pilots certified for this aircraft.
5. Find the aids of all aircraft that can be used on routes from pune to delhi
6. Display alphabetic order all employee who have salary greater than 10000.
7. Identify the routes that can be piloted by every pilot who makes more than 10000.
8. Find the average cruisingrangeof each aircraft in that find those having average cruisingrange more than 2000miles.
9. Create view for aircraft with flight details.
10. Create stored procedure to reduce cruisingrange of aircraft (i.e. user specified) for aircraft (i.e. user specified)

Here are the Oracle SQL queries to address your requests:

1. **Find the names of aircraft such that all pilots certified to operate them have salaries more than 20,000:**

   ```
   SELECT a.aircraft_name
   FROM aircraft a
   JOIN pilots p ON a.aircraft_id = p.certified_aircraft_id
   GROUP BY a.aircraft_name
   HAVING MIN(p.salary) > 20000;
   ```

2. **Set the name of employee "XYZ" identified by '0151':**

   ```
   UPDATE employees
   SET employee_name = 'XYZ'
   WHERE employee_id = '0151';
   ```

3. **Find aircrafts whose name starts with 's' and has at least 3 characters:**

   ```
   SELECT aircraft_name
   FROM aircraft
   WHERE LOWER(aircraft_name) LIKE 's%' AND LENGTH(aircraft_name) >= 3;
   ```

4. **For all aircraft with cruising range over 1000 miles, find the names of the aircraft and the average salary of all pilots certified for this aircraft:**

   ```
   SELECT a.aircraft_name, AVG(p.salary) AS average_salary
   FROM aircraft a
   JOIN pilots p ON a.aircraft_id = p.certified_aircraft_id
   WHERE a.cruising_range > 1000
   GROUP BY a.aircraft_name;
   ```

5. **Find the IDs of all aircraft that can be used on routes from Pune to Delhi:**

   ```
   SELECT DISTINCT a.aircraft_id
   FROM aircraft a
   JOIN routes r ON a.aircraft_id = r.aircraft_id
   WHERE r.source = 'Pune' AND r.destination = 'Delhi';
   ```

6. **Display in alphabetical order all employees who have salaries greater than 10,000:**

```
SELECT employee_name
FROM employees
WHERE salary > 10000
ORDER BY employee_name ASC;
```

7. **Identify the routes that can be piloted by every pilot who makes more than 10,000:**

```
SELECT r.route_id, r.source, r.destination
FROM routes r
WHERE NOT EXISTS (
    SELECT *
    FROM pilots p
    WHERE p.salary > 10000
    AND p.pilot_id NOT IN (
        SELECT rp.pilot_id
        FROM route_pilots rp
        WHERE rp.route_id = r.route_id
    )
);
```

8. **Find the average cruising range of each aircraft and then find those having an average cruising range more than 2000 miles:**

```
SELECT aircraft_name, AVG(cruising_range) AS avg_cruising_range
FROM aircraft
GROUP BY aircraft_name
HAVING AVG(cruising_range) > 2000;
```

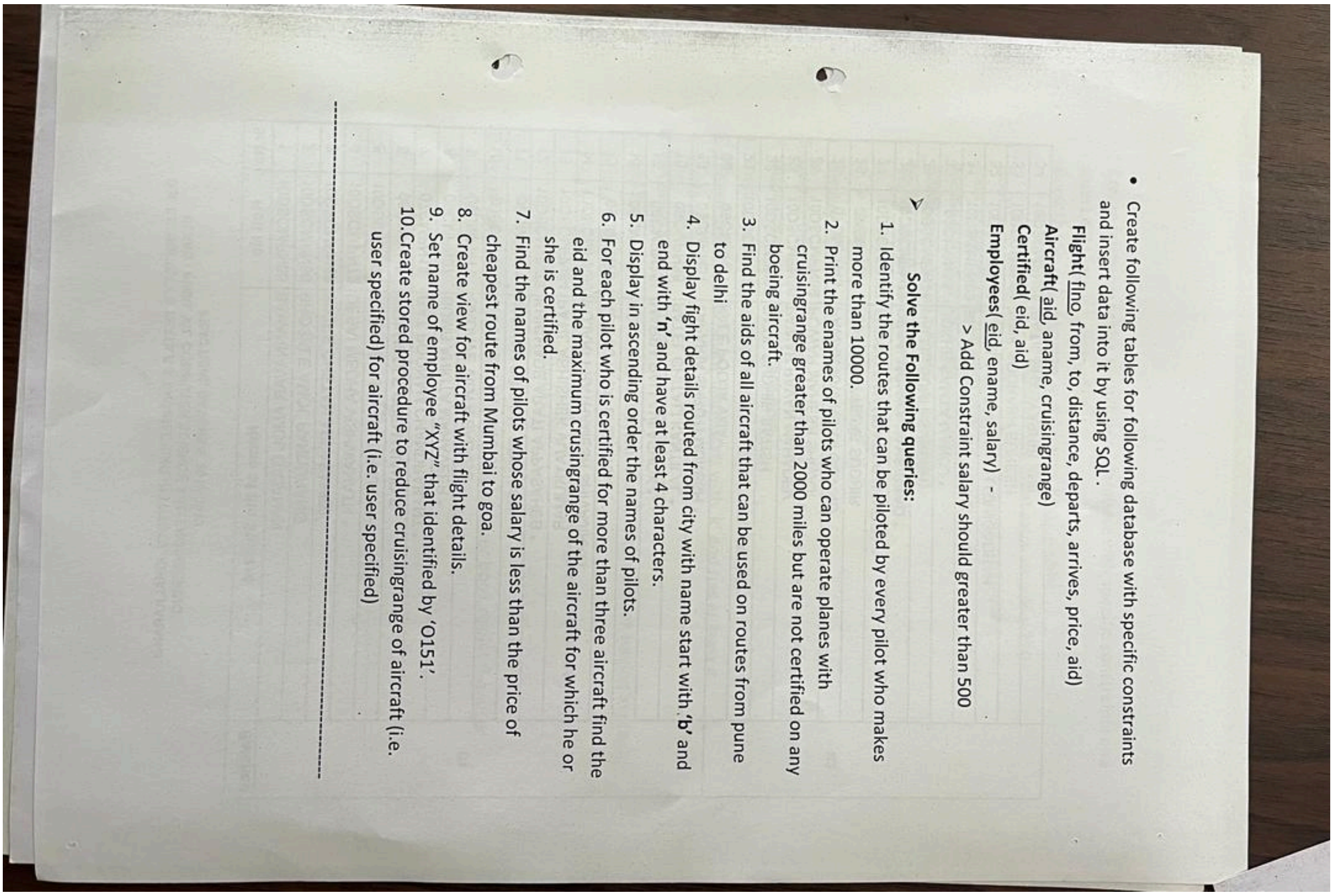
9. **Create a view for aircraft with flight details:**

```
CREATE VIEW aircraft_flight_details AS
SELECT a.aircraft_name, a.cruising_range, r.route_id, r.source, r.destination
FROM aircraft a
JOIN routes r ON a.aircraft_id = r.aircraft_id;
```

10. **Create a stored procedure to reduce the cruising range of an aircraft (user-specified) for a specific aircraft (user-specified):**

```
CREATE OR REPLACE PROCEDURE ReduceCruisingRange(
    p_aircraft_id IN VARCHAR2,
    p_reduction_amount IN NUMBER
)
IS
BEGIN
    UPDATE aircraft
    SET cruising_range = cruising_range - p_reduction_amount
    WHERE aircraft_id = p_aircraft_id;

    IF SQL%ROWCOUNT = 0 THEN
        DBMS_OUTPUT.PUT_LINE('No aircraft found with ID: ' || p_aircraft_id);
    ELSE
        DBMS_OUTPUT.PUT_LINE('Cruising range reduced for aircraft ID: ' || p_aircraft_id);
    END IF;
END;
```

These queries should meet your requirements effectively. If you need further clarification or adjustments, feel free to ask!



- Create following tables for following database with specific constraints and insert data into it by using SQL .

  Flight(fino, from, to, distance, departs, arrives, price, aid)

  Aircraft(aid, aname, cruisingrange)

  Certified(eid, aid)

  Employees(eid, ename, salary) -

  >Add Constraint salary should greater than 500

Solve the Following queries:

1. Identify the routes that can be piloted by every pilot who makes more than 10000.
2. Print the enames of pilots who can operate planes with cruisingrange greater than 2000 miles but are not certified on any boeing aircraft.
3. Find the aids of all aircraft that can be used on routes from pune to delhi
4. Display fight details routed from city with name start with **'b'** and end with **'n'** and have at least 4 characters.
5. Display in ascending order the names of pilots.
6. For each pilot who is certified for more than three aircraft find the eid and the maximum crusingrange of the aircraft for which he or she is certified.
7. Find the names of pilots whose salary is less than the price of cheapest route from Mumbai to goa.
8. Create view for aircraft with flight details.
9. Set name of employee "XYZ" that identified by 'O151'.
10. Create stored procedure to reduce cruisingrange of aircraft (i.e. user specified) for aircraft (i.e. user specified)

Here are the Oracle SQL queries tailored to your requirements:

1. **Identify the routes that can be piloted by every pilot who makes more than 10,000:**

```
SELECT r.route_id, r.source, r.destination
FROM routes r
WHERE NOT EXISTS (
    SELECT p.pilot_id
    FROM pilots p
    WHERE p.salary > 10000
    AND p.pilot_id NOT IN (
        SELECT rp.pilot_id
        FROM route_pilots rp
        WHERE rp.route_id = r.route_id
    )
);
```

2. **Print the names of pilots who can operate planes with cruising range greater than 2,000 miles but are not certified on any Boeing aircraft:**

```
SELECT DISTINCT p.pilot_name
FROM pilots p
JOIN aircraft a ON p.certified_aircraft_id = a.aircraft_id
WHERE a.cruising_range > 2000
AND p.pilot_id NOT IN (
    SELECT p2.pilot_id
    FROM pilots p2
    JOIN aircraft a2 ON p2.certified_aircraft_id = a2.aircraft_id
    WHERE a2.brand = 'Boeing'
);
```

3. **Find the IDs of all aircraft that can be used on routes from Pune to Delhi:**

```
SELECT DISTINCT a.aircraft_id
FROM aircraft a
JOIN routes r ON a.aircraft_id = r.aircraft_id
WHERE r.source = 'Pune' AND r.destination = 'Delhi';
```

4. **Display flight details routed from cities with names starting with 'b' and ending with 'n' and having at least 4 characters:**

```
SELECT r.route_id, r.source, r.destination
FROM routes r
WHERE LOWER(r.source) LIKE 'b%n' AND LENGTH(r.source) >= 4;
```

5. **Display in ascending order the names of pilots:**

```
SELECT pilot_name
FROM pilots
ORDER BY pilot_name ASC;
```

6. **For each pilot who is certified for more than three aircraft, find the eid and the maximum cruising range of the aircraft for which he or she is certified:**

```
SELECT p.pilot_id, MAX(a.cruising_range) AS max_cruising_range
FROM pilots p
JOIN aircraft a ON p.certified_aircraft_id = a.aircraft_id
GROUP BY p.pilot_id
HAVING COUNT(a.aircraft_id) > 3;
```

7. **Find the names of pilots whose salary is less than the price of the cheapest route from Mumbai to Goa:**

```
SELECT p.pilot_name
FROM pilots p
WHERE p.salary < (
    SELECT MIN(route_price)
    FROM routes
    WHERE source = 'Mumbai' AND destination = 'Goa'
);
```

8. **Create a view for aircraft with flight details:**

```
CREATE VIEW aircraft_flight_details AS
SELECT a.aircraft_name, a.cruising_range, r.route_id, r.source, r.destination
FROM aircraft a
JOIN routes r ON a.aircraft_id = r.aircraft_id;
```

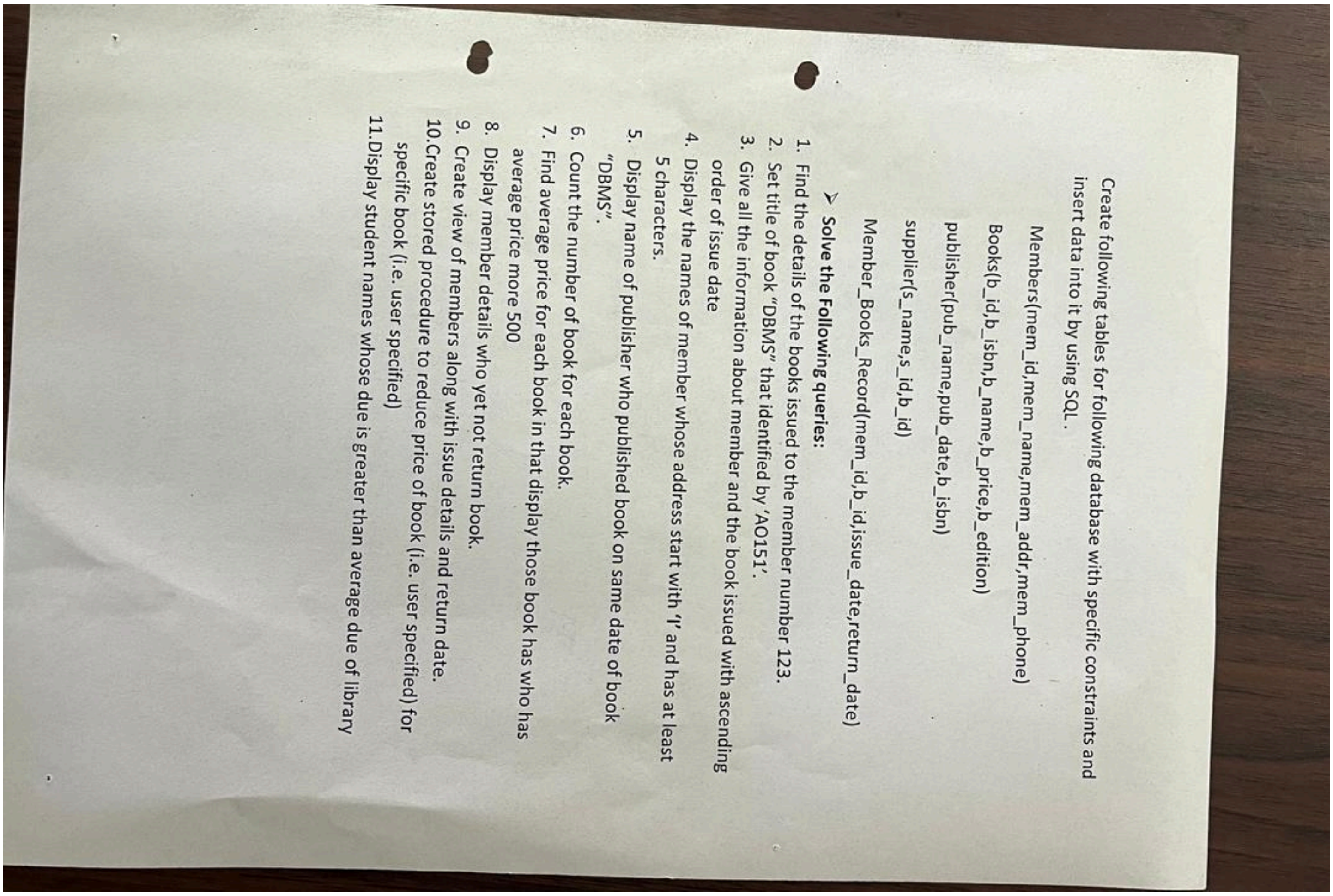
9. **Set the name of employee "XYZ" identified by '0151':**

```
UPDATE employees
SET employee_name = 'XYZ'
WHERE employee_id = '0151';
```

10. **Create a stored procedure to reduce the cruising range of an aircraft (user-specified) for a specific aircraft (user-specified):**

```
CREATE OR REPLACE PROCEDURE ReduceCruisingRange(
    p_aircraft_id IN VARCHAR2,
    p_reduction_amount IN NUMBER
)
IS
BEGIN
    UPDATE aircraft
    SET cruising_range = cruising_range - p_reduction_amount
    WHERE aircraft_id = p_aircraft_id;

    IF SQL%ROWCOUNT = 0 THEN
        DBMS_OUTPUT.PUT_LINE('No aircraft found with ID: ' || p_aircraft_id);
    ELSE
        DBMS_OUTPUT.PUT_LINE('Cruising range reduced for aircraft ID: ' || p_aircraft_id);
    END IF;
END;
```

These queries should fulfill your requirements effectively. If you need any additional modifications or explanations, feel free to ask!



Create following tables for following database with specific constraints and insert data into it by using SQL.

Members(mem_id,mem_name,mem_addr,mem_phone)

Books(b_id,b_isbn,b_name,b_price,b_edition)

publisher(pub_name,pub_date,b_isbn)

supplier(s_name,s_id,b_id)

Member_Books_Record(mem_id,b_id,issue_date,return_date)

➢ **Solve the Following queries:**

1. Find the details of the books issued to the member number 123.
2. Set title of book "DBMS" that identified by 'AO151'.
3. Give all the information about member and the book issued with ascending order of issue date
4. Display the names of member whose address start with '**I**' and has at least 5 characters.
5. Display name of publisher who published book on same date of book "DBMS".
6. Count the number of book for each book.
7. Find average price for each book in that display those book has who has average price more 500
8. Display member details who yet not return book.
9. Create view of members along with issue details and return date.
10. Create stored procedure to reduce price of book (i.e. user specified) for specific book (i.e. user specified)
11. Display student names whose due is greater than average due of library

Here are the Oracle SQL queries tailored to your requests regarding books, members, and other library-related operations:

1. **Find the details of the books issued to the member number 123:**

```
SELECT b.*
FROM books b
JOIN issues i ON b.book_id = i.book_id
WHERE i.member_id = 123;
```

2. **Set the title of the book "DBMS" identified by 'A0151':**

```
UPDATE books
SET title = 'DBMS'
WHERE book_id = 'A0151';
```

3. **Give all the information about members and the book issued with ascending order of issue date:**

```
SELECT m.*, b.*, i.issue_date
FROM members m
JOIN issues i ON m.member_id = i.member_id
JOIN books b ON i.book_id = b.book_id
ORDER BY i.issue_date ASC;
```

4. **Display the names of members whose address starts with 'l' and has at least 5 characters:**

```
SELECT member_name
FROM members
WHERE LOWER(address) LIKE 'l%' AND LENGTH(address) >= 5;
```

5. **Display the name of the publisher who published the book on the same date as the book "DBMS":**

```
SELECT p.publisher_name
FROM publishers p
JOIN books b ON p.publisher_id = b.publisher_id
WHERE b.issue_date = (SELECT issue_date FROM books WHERE title = 'DBMS');
```

6. **Count the number of books for each book:**

```
SELECT title, COUNT(*) AS book_count
FROM books
GROUP BY title;
```

7. **Find the average price for each book and display those books that have an average price more than 500:**

   ```
   SELECT title, AVG(price) AS average_price
   FROM books
   GROUP BY title
   HAVING AVG(price) > 500;
   ```

8. **Display member details who have not yet returned a book:**

   ```
   SELECT m.*
   FROM members m
   JOIN issues i ON m.member_id = i.member_id
   WHERE i.return_date IS NULL;
   ```

9. **Create a view of members along with issue details and return date:**

   ```
   CREATE VIEW member_issue_details AS
   SELECT m.member_id, m.member_name, b.title, i.issue_date, i.return_date
   FROM members m
   JOIN issues i ON m.member_id = i.member_id
   JOIN books b ON i.book_id = b.book_id;
   ```

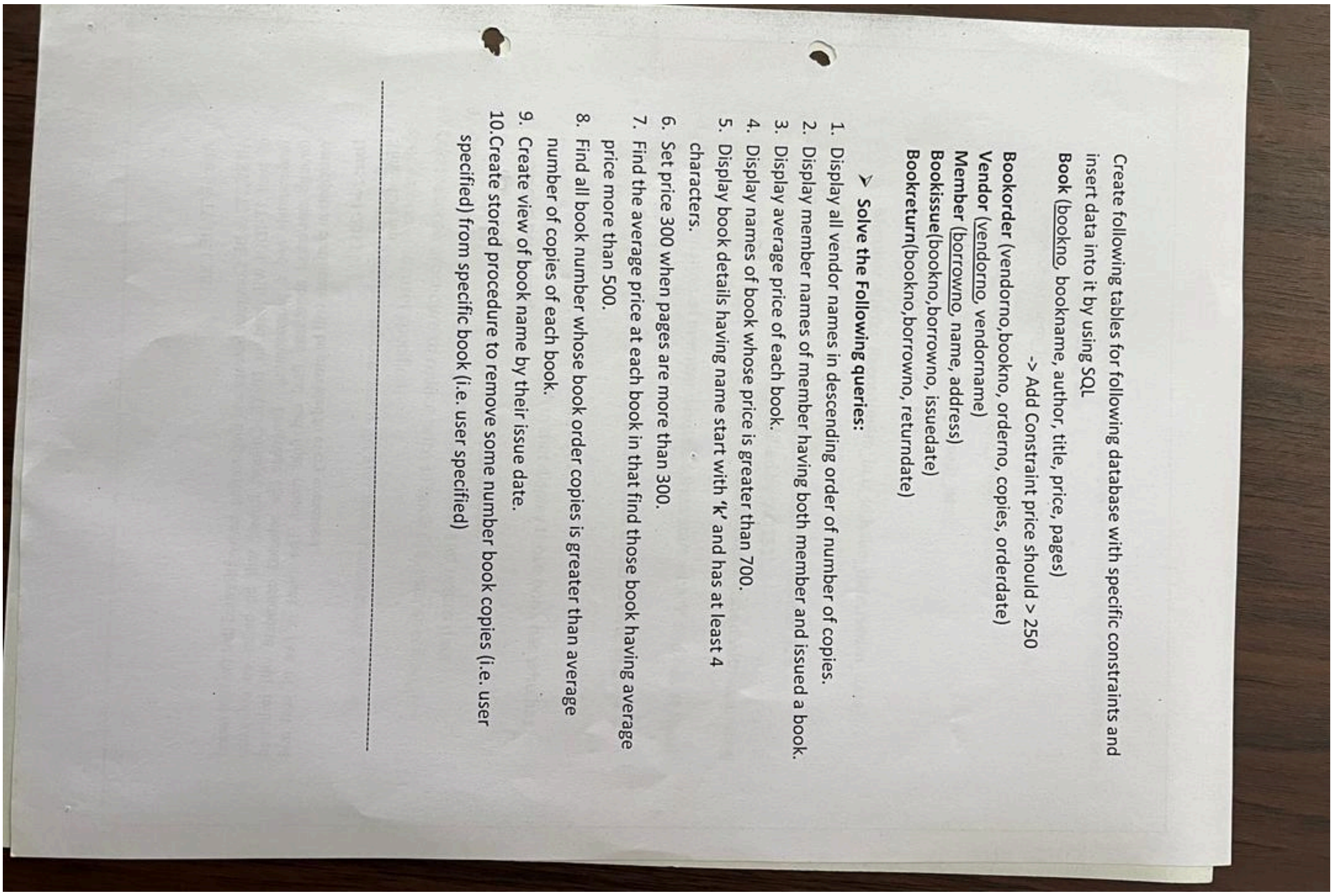
10. **Create a stored procedure to reduce the price of a book (user-specified) for a specific book (user-specified):**

    ```
    CREATE OR REPLACE PROCEDURE ReduceBookPrice(
        p_book_id IN VARCHAR2,
        p_reduction_amount IN NUMBER
    )
    IS
    BEGIN
        UPDATE books
        SET price = price - p_reduction_amount
        WHERE book_id = p_book_id;

        IF SQL%ROWCOUNT = 0 THEN
            DBMS_OUTPUT.PUT_LINE('No book found with ID: ' || p_book_id);
        ELSE
            DBMS_OUTPUT.PUT_LINE('Price reduced for book ID: ' || p_book_id);
        END IF;
    END;
    ```

11. **Display student names whose due is greater than the average due of the library:**

    ```
    SELECT student_name
    FROM students
    WHERE due > (SELECT AVG(due) FROM students);
    ```

These queries should effectively cover your requirements. If you need any further adjustments or clarifications, feel free to ask!



Create following tables for following database with specific constraints and insert data into it by using SQL

Book (bookno, bookname, author, title, price, pages)

->Add Constraint price should > 250

Bookorder (vendorno,bookno, orderno, copies, orderdate)

Vendor (vendorno, vendorname)

Member (borrowno, name, address)

Bookissue(bookno,borrowno, issuedate)

Bookreturn(bookno,borrowno, returndate)

➢ **Solve the Following queries:**

1. Display all vendor names in descending order of number of copies.
2. Display member names of member having both member and issued a book.
3. Display average price of each book.
4. Display names of book whose price is greater than 700.
5. Display book details having name start with **'k'** and has at least 4 characters.
6. Set price 300 when pages are more than 300.
7. Find the average price at each book in that find those book having average price more than 500.
8. Find all book number whose book order copies is greater than average number of copies of each book.
9. Create view of book name by their issue date.
10. Create stored procedure to remove some number book copies (i.e. user specified) from specific book (i.e. user specified)

Here are the Oracle SQL queries tailored to your requirements regarding vendors, members, books, and other related operations:

1. **Display all vendor names in descending order of the number of copies:**

```
SELECT v.vendor_name
FROM vendors v
JOIN books b ON v.vendor_id = b.vendor_id
GROUP BY v.vendor_name
ORDER BY SUM(b.copies) DESC;
```

2. **Display member names of members having both membership and issued a book:**

```
SELECT DISTINCT m.member_name
FROM members m
JOIN issues i ON m.member_id = i.member_id;
```

3. **Display average price of each book:**

```
SELECT title, AVG(price) AS average_price
FROM books
GROUP BY title;
```

4. **Display names of books whose price is greater than 700:**

```
SELECT title
FROM books
WHERE price > 700;
```

5. **Display book details having names starting with 'k' and having at least 4 characters:**

```
SELECT *
FROM books
WHERE LOWER(title) LIKE 'k%' AND LENGTH(title) >= 4;
```

6. **Set price to 300 when pages are more than 300:**

```
UPDATE books
SET price = 300
WHERE pages > 300;
```

7. **Find the average price of each book and find those books having an average price more than 500:**

   ```
   SELECT title, AVG(price) AS average_price
   FROM books
   GROUP BY title
   HAVING AVG(price) > 500;
   ```

8. **Find all book numbers whose book order copies are greater than the average number of copies of each book:**

   ```
   SELECT book_id
   FROM books
   WHERE copies > (SELECT AVG(copies) FROM books);
   ```

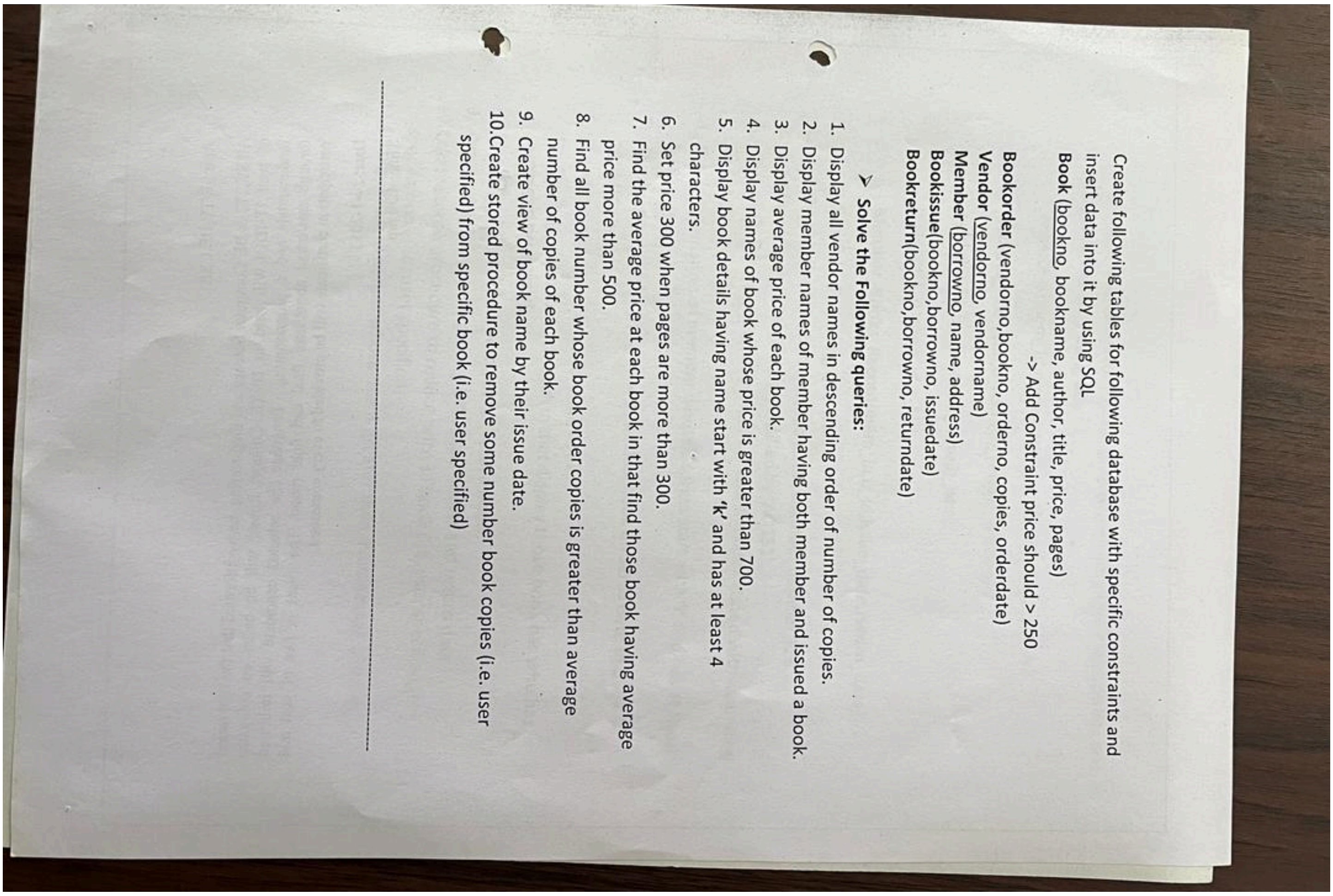
9. **Create a view of book names by their issue date:**

   ```
   CREATE VIEW book_issue_dates AS
   SELECT b.title, i.issue_date
   FROM books b
   JOIN issues i ON b.book_id = i.book_id
   ORDER BY i.issue_date;
   ```

10. **Create a stored procedure to remove some number of book copies (user-specified) from a specific book (user-specified):**

    ```
    CREATE OR REPLACE PROCEDURE RemoveBookCopies(
        p_book_id IN VARCHAR2,
        p_remove_count IN NUMBER
    )
    IS
    BEGIN
        UPDATE books
        SET copies = copies - p_remove_count
        WHERE book_id = p_book_id AND copies >= p_remove_count;

        IF SQL%ROWCOUNT = 0 THEN
            DBMS_OUTPUT.PUT_LINE('No book found with ID: ' || p_book_id || ' or insufficient copies to remove.');
        ELSE
            DBMS_OUTPUT.PUT_LINE('Removed ' || p_remove_count || ' copies from book ID: ' || p_book_id);
        END IF;
    END;
    ```

These queries should effectively meet your requirements. If you have any further questions or need additional modifications, feel free to ask!



Create following tables for following database with specific constraints and insert data into it by using SQL

Book (bookno, bookname, author, title, price, pages)

->Add Constraint price should > 250

Bookorder (vendorno,bookno, orderno, copies, orderdate)

Vendor (vendorno, vendorname)

Member (borrowno, name, address)

Bookissue(bookno,borrowno, issuedate)

Bookreturn(bookno,borrowno, returndate)

➢ **Solve the Following queries:**

1. Display all vendor names in descending order of number of copies.
2. Display member names of member having both member and issued a book.
3. Display average price of each book.
4. Display names of book whose price is greater than 700.
5. Display book details having name start with **'k'** and has at least 4 characters.
6. Set price 300 when pages are more than 300.
7. Find the average price at each book in that find those book having average price more than 500.
8. Find all book number whose book order copies is greater than average number of copies of each book.
9. Create view of book name by their issue date.
10. Create stored procedure to remove some number book copies (i.e. user specified) from specific book (i.e. user specified)

Here are the Oracle SQL queries to fulfill your requirements regarding vendors, members, books, and related operations:

1. **Display all vendor names in descending order of the number of copies:**

   ```
   SELECT v.vendor_name
   FROM vendors v
   JOIN books b ON v.vendor_id = b.vendor_id
   GROUP BY v.vendor_name
   ORDER BY SUM(b.copies) DESC;
   ```

2. **Display member names of members having both membership and issued a book:**

   ```
   SELECT DISTINCT m.member_name
   FROM members m
   JOIN issues i ON m.member_id = i.member_id;
   ```

3. **Display the average price of each book:**

   ```
   SELECT title, AVG(price) AS average_price
   FROM books
   GROUP BY title;
   ```

4. **Display names of books whose price is greater than 700:**

   ```
   SELECT title
   FROM books
   WHERE price > 700;
   ```

5. **Display book details having names starting with 'k' and having at least 4 characters:**

   ```
   SELECT *
   FROM books
   WHERE LOWER(title) LIKE 'k%' AND LENGTH(title) >= 4;
   ```

6. **Set price to 300 when pages are more than 300:**

   ```
   UPDATE books
   SET price = 300
   WHERE pages > 300;
   ```

7. **Find the average price of each book and find those books having an average price more than 500:**

   ```
   SELECT title, AVG(price) AS average_price
   FROM books
   GROUP BY title
   HAVING AVG(price) > 500;
   ```

8. **Find all book numbers whose book order copies are greater than the average number of copies of each book:**

   ```
   SELECT book_id
   FROM books
   WHERE copies > (SELECT AVG(copies) FROM books);
   ```

9. **Create a view of book names by their issue date:**

   ```
   CREATE VIEW book_issue_dates AS
   SELECT b.title, i.issue_date
   FROM books b
   JOIN issues i ON b.book_id = i.book_id
   ORDER BY i.issue_date;
   ```

10. **Create a stored procedure to remove some number of book copies (user-specified) from a specific book (user-specified):**

    ```
    CREATE OR REPLACE PROCEDURE RemoveBookCopies(
        p_book_id IN VARCHAR2,
        p_remove_count IN NUMBER
    )
    IS
    BEGIN
        UPDATE books
        SET copies = copies - p_remove_count
        WHERE book_id = p_book_id AND copies >= p_remove_count;

        IF SQL%ROWCOUNT = 0 THEN
            DBMS_OUTPUT.PUT_LINE('No book found with ID: ' || p_book_id || ' or insufficient copies to remove.');
        ELSE
            DBMS_OUTPUT.PUT_LINE('Removed ' || p_remove_count || ' copies from book ID: ' || p_book_id);
        END IF;
    END;
    ```

These queries should fulfill your requests accurately. If you have any further questions or need additional adjustments, feel free to ask!