FUNDAMENTALS OF DATABASE
PROJECT PART-4
HETERO PHARMACEUTICALS DATABASE

Group – 3

Team Members:

- Sreekar Thanda
- Shravan Kumar Nuka
- Manideep Renikindi
- Varsha Umannagari
- Rama Venkat Sumith Thota

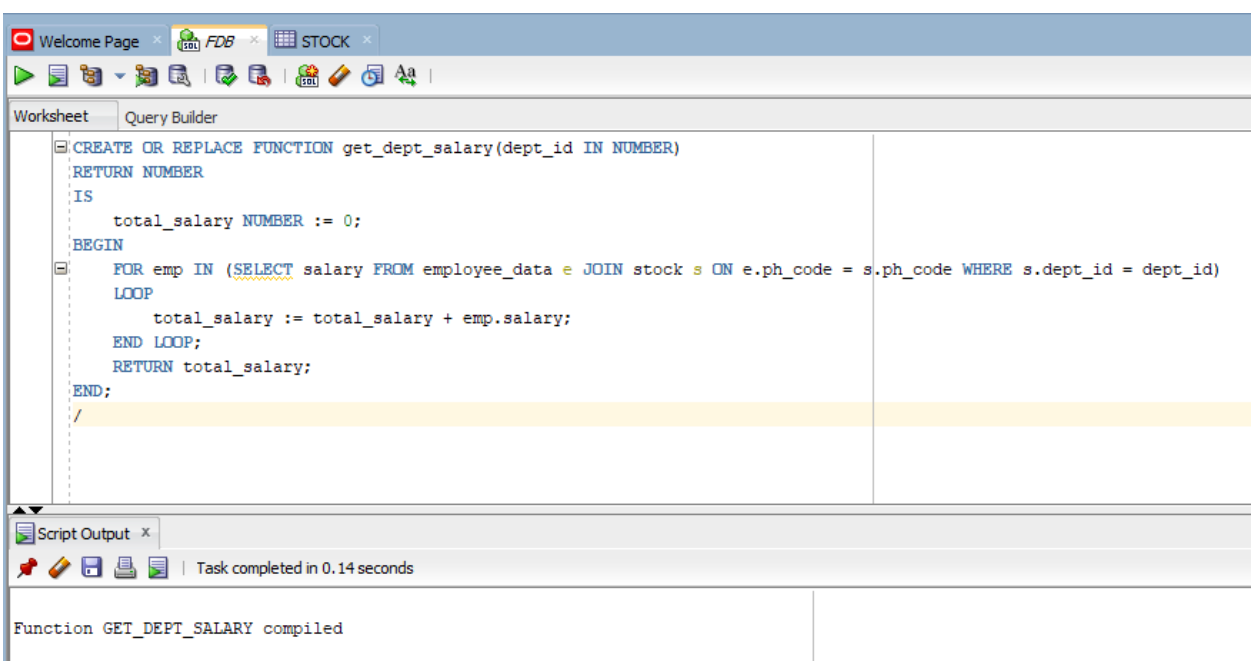
Note: There are no new assumptions for this part. Hence, as per given instructions by the grader previously, the description is not added here.

Stored Functions:

A stored function in SQL is a type of user-defined function that is stored in a database and can be called from within SQL queries and other database objects. Stored functions allow you to encapsulate a set of SQL statements into a single reusable unit, which can be called multiple times from different parts of an application or database.

The following are the 5 stored functions in the project, and they are as follows:

1) Stored Function to calculate the total salary of all employees in given department.



The above screenshot shows the function created and compiled, where the function name get_dept_salary returns the amount paid to employees of a given department in the form of salaries.

To attain this, the stock table is joined with the employee table. Now to calculate the salaries, a loop is used. This loop adds all the salaries of each employee, and it iterates till the last row of the given specified department.

```
set serveroutput on;
DECLARE
    dept_id NUMBER := 1;
    total_salary NUMBER;
BEGIN
    total_salary := get_dept_salary(dept_id);
    DBMS_OUTPUT.PUT_LINE('Total salary for department ' || dept_id || ': ' || total_salary);
END;
/
```

Script Output

Task completed in 0.077 seconds

```
Total salary for department 1: 18600


PL/SQL procedure successfully completed.
```

After, Creating and compiling the function, it must be called to implement the function, So, Anonymous PL/SQL is used to call the function, such that the dept_id is given as 1.

The function is thus called and had returned the output ‘Total salary for department 1 is 18600’.

2) Function to check if a given patient has valid insurance to claim:

```
--2)
CREATE OR REPLACE FUNCTION check_insurance_claim(patient_id IN VARCHAR2)
RETURN BOOLEAN
IS
    valid_claim insurance_claim.validity%TYPE;
    CURSOR c_insurance_claims IS
        SELECT validity
        FROM insurance_claim
        WHERE patient_id = patient_id;
BEGIN
    FOR claim IN c_insurance_claims LOOP
        valid_claim := claim.validity;
        IF valid_claim = 'valid' THEN
            RETURN TRUE;
        END IF;
    END LOOP;

    RETURN FALSE;
END;
/
```

Script Output | Task completed in 0.037 seconds

```
Function CHECK_INSURANCE_CLAIM compiled
```

The above function "check_insurance_claim" is created and compiled to check if the patient's insurance claim is valid or not. patient_id is given as the input argument to the function. Cursor is used here, which holds the validity into validity variable till if endif loop is executed and finished." For" loop is used for iteration and if endif condition loops are used to find if the given condition is valid or not.

```
set serveroutput on;
DECLARE
    is_valid BOOLEAN;
BEGIN
    is_valid := check_insurance_claim('2021');
    IF is_valid = TRUE THEN
        DBMS_OUTPUT.PUT_LINE('Insurance claim is valid.');
    ELSE
        DBMS_OUTPUT.PUT_LINE('Insurance claim is not valid.');
    END IF;
END;
/
```

Script Output | Task completed in 0.07 seconds

```
Insurance claim is valid.


PL/SQL procedure successfully completed.
```

The anonymous PL/SQL block is used to call the function check_insurance_claim of the

patient_id “2021”, If the condition specified is true (given patient_id could claim a valid insurance) then function returns as Insurance claim is valid.

3) Function to display the patient’s age from the given patient’s name.

```
CREATE OR REPLACE FUNCTION get_patient_details(p_name IN patient.patient_name%TYPE)
RETURN VARCHAR2
IS
  patient_dob DATE;
  patient_age NUMBER;
BEGIN
  SELECT date_of_birth INTO patient_dob FROM patient WHERE patient_name = p_name;
  patient_age := TRUNC(MONTHS_BETWEEN(SYSDATE, patient_dob)/12);
  RETURN p_name || ' is ' || patient_age || ' years old.';
EXCEPTION
  WHEN NO_DATA_FOUND THEN
    RETURN 'Patient ' || p_name || ' does not exist in the database.';
END;
/
```

Script Output | Task completed in 0.073 seconds

```
Function GET_PATIENT_DETAILS compiled
```

The above function named as “get_patient_details” returns the patient age for the given patient. Select query is used in the function to select the date of birth and derive the age in years from date of birth. If the patient’s name is not existed in table, Function returns as Patient name does not exist in the database.

```
set serveroutput on;
SELECT get_patient_details('Sreekar') AS patient_details FROM DUAL;
SELECT get_patient_details('John') AS patient_details FROM DUAL;
```

Script Output | Query Result | Query Result 1

SQL | All Rows Fetched: 1 in 0.009 seconds

	PATIENT_DETAILS
1	Patient Sreekar does not exist in the database.

Given the patient’s name as Sreekar which does not exist in the database then the function is returned as above.

Script Output | Query Result | Query Result 1

SQL | All Rows Fetched: 1 in 0.004 seconds

	PATIENT_DETAILS
1	John is 25 years old.

Given the patient's name as John which exist in the database then the function calculated the age of the patient from patient's date of birth and so function returned the calculated age till today in years.

4) PL/SQL function to display the prescription details for the specified patient and doctor.

```
CREATE OR REPLACE FUNCTION get_prescriptions(
    p_patient_name IN VARCHAR2,
    p_doctor_name IN VARCHAR2
)
RETURN SYS_REFCURSOR
IS
    cur SYS_REFCURSOR;
BEGIN
    OPEN cur FOR
        SELECT pr.prescription_id, pr.p_date,pr.patient_id,pr.doctor_id
        FROM prescription pr
        JOIN patient p ON pr.patient_id = p.patient_id
        JOIN Doctor d ON pr.doctor_id = d.doctor_id
        WHERE p.patient_name = p_patient_name
        AND d.doctor_name = p_doctor_name;
    RETURN cur;
EXCEPTION
    WHEN NO_DATA_FOUND THEN
        DBMS_OUTPUT.PUT_LINE('No prescriptions found for this patient and doctor combination.');
        RETURN NULL;
END;
/
```

Script Output

Task completed in 0.069 seconds

```
PL/SQL procedure successfully completed.
```

The function named "get_prescriptions" will fetch the prescription details of the specified patient name and doctor name. For this, Cursor is used and join operations on tables are used such that to match the patient_id and doctor_id details for the given patient_name and doctor_name details, Also if there are no prescriptions for the specified patient_name and doctor_name then the output returns as "No prescriptions found for this patient and doctor combination".

```
set serveroutput on;
VAR cur REFCURSOR
EXEC :cur := get_prescriptions('John','John');
PRINT cur
```

Script Output

Task completed in 0.069 seconds

```
PRESCRIPTION_ID      P_DATE    PATIENT_ID           DOCTOR_ID
-------------------- --------- -------------------- ----------
301                  03-JUN-23 2012                        700
```

The above is executed called using cursor. The patient name is specified as John and doctor name is also specified as John, The prescription details in the database where doctor John has treated Patient John are shown in above screenshot.

5) Create a function using PL/SQL to find the total bill amount for a particular prescription.

```
CREATE OR REPLACE FUNCTION get_total_bill_amount(prescription_id IN VARCHAR2)
RETURN NUMBER
IS
    total_bill_amount NUMBER := 0;
BEGIN
    FOR purchase_rec IN (SELECT trans_id FROM purchase WHERE prescription_id = prescription_id)
    LOOP
        SELECT bill_amount INTO total_bill_amount
        FROM bills
        WHERE trans_id = purchase_rec.trans_id;

        total_bill_amount := total_bill_amount + total_bill_amount;
    END LOOP;

    RETURN total_bill_amount;
END;
/
```

Script Output Query Result

Task completed in 0.055 seconds

```
Function GET_TOTAL_BILL_AMOUNT compiled
```

The function named “get_total_bill_amount” is created and compiled, it takes input “prescription_id” and returns the total bill amount by the patient purchased in the pharmacy.

```
SELECT get_total_bill_amount('234534') FROM dual;
```

Script Output | Query Result

SQL | All Rows Fetched: 1 in 0.005 seconds

	GET_TOTAL_BILL_AMOUNT('234534')
1	1130

To call the function, select function_name('inputargument') from dual is used. So, "select get_total_bill_amount('234534') from dual;" "234534" is the prescription_id which takes the input. The query call the function and once function is executed with statements in it, returns the output for the given preciption_id(234534) as 1130.

Stored Procedures:

A stored procedure in SQL is a group of pre-written SQL statements that are stored in a database and can be executed as a single unit. Stored procedures can be called from within SQL queries or other database objects, taking input parameters and returning output values.

1) Create a procedure to get the full details of an employee.

```
CREATE OR REPLACE PROCEDURE employee_details (
    search_name IN VARCHAR2
) AS
    emp_id NUMBER;
    emp_name VARCHAR2(30);
    emp_designation VARCHAR2(50);
    emp_gender VARCHAR2(8);
    emp_contact VARCHAR2(12);
    emp_salary NUMBER;
BEGIN
    SELECT emp_id, emp_name, gender, emp_contact, designation,salary
    INTO emp_id, emp_name, emp_gender, emp_contact, emp_designation,emp_salary
    FROM employee_data
    WHERE emp_name LIKE '%' || search_name || '%';
    DBMS_OUTPUT.PUT_LINE('Employee ID: ' || emp_id);
    DBMS_OUTPUT.PUT_LINE('Employee Name: ' || emp_name);
    DBMS_OUTPUT.PUT_LINE('Gender: ' || emp_gender);
    DBMS_OUTPUT.PUT_LINE('Contact_number: ' || emp_contact);
    DBMS_OUTPUT.PUT_LINE('Designation: ' || emp_designation);
    DBMS_OUTPUT.PUT_LINE('Salary: ' || emp_salary);
END;
/
```

Script Output

Task completed in 0.072 seconds

```
Procedure EMPLOYEE_DETAILS compiled
```

The stored procedure, employee_details is created and compiled, these procedures find the fetches the employee details of the given input employee name similar to name Arun. This

can be used at a point of time when the employee's full name is not known but only a part of employee name is known. This function prints the entire employee details whose name matches with given name, therefore, function returns employee_id, Employee_full_name, gender,contact, designation and salary of given employee name.

```
set serveroutput on;
EXECUTE EMPLOYEE_DETAILS('Arun');
```

Script Output

Task completed in 0.081 seconds

```
Employee ID: 201
Employee Name: Arun
Gender: male
Contact_number: 922-2922-222
Designation: Manager
Salary: 6600


PL/SQL procedure successfully completed.
```

Here, Considered the partial known name of employee is Arun, and executed the procedures with input as" Arun" and so all the details of employee whose has name as Arun prints out.

2) Create a procedure to display the summary of Inventory.

```
--2)
CREATE OR REPLACE PROCEDURE INVENTORY_STOCK_SUMMARY
AS
   item_name inventory.item_name%TYPE;
   total_quantity NUMBER;
BEGIN
   FOR c IN (SELECT i.item_name, SUM(i.presently_available_quantity) AS total_quantity
             FROM inventory i
             JOIN stock s ON i.item_id = s.item_id
             GROUP BY i.item_name
             ORDER BY total_quantity)
   LOOP
      DBMS_OUTPUT.PUT_LINE('Item: ' || c.item_name || ', Total Quantity: ' || c.total_quantity);
   END LOOP;
END;
/
```

Script Output

Task completed in 0.054 seconds

```
Procedure INVENTORY_STOCK_SUMMARY compiled
```

This Procedure named “Inventory_stock_summary” is created and compiled which displays all the medicines and the quantity present in each medicine. For this, the stock table is joined upon inventory table matching item_id and the resulted joined table is grouped on item_name and is sorted ascendingly on total_quantity present. This procedure is used for all tuples present in the inventory table.

```
EXECUTE INVENTORY_STOCK_SUMMARY;
```

Script Output

Task completed in 0.074 seconds

```
Item: Paracetamol, Total Quantity: 42
Item: cetirizine, Total Quantity: 58
Item: Acetaminophen, Total Quantity: 15000
Item: Tylenol, Total Quantity: 15000
Item: Diltiazem, Total Quantity: 15000
Item: Brinzolamide, Total Quantity: 15000
Item: Naproxen, Total Quantity: 15000
Item: Dolo, Total Quantity: 30000


PL/SQL procedure successfully completed.
```

The procedure is executed and seen that the procedure has displayed all the medicines and the available total quantity.

For in case, If needed to find out the least 5 available quantity medicines is shown below,

```
CREATE OR REPLACE PROCEDURE INVENTORY_STOCK_SUMMARY
AS
   item_name inventory.item_name%TYPE;
   total_quantity NUMBER;
BEGIN
   FOR c IN (SELECT i.item_name, SUM(i.presently_available_quantity) AS total_quantity
             FROM inventory i
             JOIN stock s ON i.item_id = s.item_id
             GROUP BY i.item_name
             ORDER BY total_quantity ASC
             FETCH FIRST 5 ROWS ONLY)
   LOOP
      DBMS_OUTPUT.PUT_LINE('Item: ' || c.item_name || ', Total Quantity: ' || c.total_quantity);
   END LOOP;
END;
/

EXECUTE INVENTORY_STOCK_SUMMARY;
```

Script Output

Task completed in 0.052 seconds

```
Procedure INVENTORY_STOCK_SUMMARY compiled
```

```
EXECUTE INVENTORY_STOCK_SUMMARY;
```

Script Output

Task completed in 0.052 seconds

```
Item: Paracetamol, Total Quantity: 42
Item: cetirizine, Total Quantity: 58
Item: Brinzolamide, Total Quantity: 15000
Item: Diltiazem, Total Quantity: 15000
Item: Acetaminophen, Total Quantity: 15000


PL/SQL procedure successfully completed.
```

After execution of the procedure, could see that the least 5 available medicines in the inventory.

3) Create a procedure to list the doctor names for the given specified specialization.

```
--3)
set serveroutput on;
CREATE OR REPLACE PROCEDURE list_doctors_by_specialty(
    specialty_name IN VARCHAR2
)
IS
BEGIN
  FOR doctor IN (SELECT * FROM Doctor WHERE specialization = specialty_name)
  LOOP
    DBMS_OUTPUT.PUT_LINE(doctor.doctor_name);
  END LOOP;
END;
/
```

Script Output

Task completed in 0.058 seconds

```
Procedure LIST_DOCTORS_BY_SPECIALTY compiled
```

The procedure “list_doctors_by_speciality” is created and compiled which displays the names of the doctors whose specialization is specified in input. Then specialization input is given in speciality_name variable and for loop finds each row in the doctor table till the specialization is equal to given input in speciality_name and prints that doctor name.

```
BEGIN
  list_doctors_by_specialty('cardiologist');
END;
```

Script Output

Task completed in 0.05 seconds

```
John


PL/SQL procedure successfully completed.
```

To execute the procedure, short anonymous pl/sql is used to find the doctor's name whose is specialized as a cardiologist.

4) Create a procedure to list the count of patients each doctor has treated.

```
--4)
set serveroutput on;
CREATE OR REPLACE PROCEDURE list_doctors_with_patients AS
BEGIN
  FOR r IN (SELECT doctor.doctor_id, doctor.doctor_name, COUNT(treatment.token_number) AS patient_count
            FROM doctor LEFT JOIN treatment ON doctor.doctor_id = treatment.doctor_id
            GROUP BY doctor.doctor_id, doctor.doctor_name)
  LOOP
    DBMS_OUTPUT.PUT_LINE(r.doctor_name || ': ' || r.patient_count || ' patients');
  END LOOP;
END;
/
```

Script Output

Task completed in 0.059 seconds

```
PL/SQL procedure successfully completed.


Procedure LIST_DOCTORS_WITH_PATIENTS compiled
```

The procedure "list_doctors_with_patients" is created and compiled to display the count of the patients that each doctor has treated. For this loop is used to find the patients of each doctor and count the number of patient with respect to doctor. So, the doctor table is joined with treatment table matching doctor_ids and the resultant table is grouped based on doctor_id and doctor_name.

```
execute list_doctors_with_patients();
```

Script Output

Task completed in 0.059 seconds

```
John: 1 patients
Peter: 1 patients
Cinderella: 1 patients
Alex: 1 patients
Joe: 1 patients
Mark Henry (Surgeon) : 1 patients
Elizabeth (Surgeon) : 1 patients
Ashley (Surgeon) : 1 patients
Bill Hunter: 0 patients


PL/SQL procedure successfully completed.
```

The procedure is executed as shown in above screenshot and could see the result as well.

5) Create a procedure to find the re-imbursement amount of a given patient.

```
set serveroutput on;
CREATE OR REPLACE PROCEDURE generate_reimbursement(
    p_patient_id IN VARCHAR2,
    p_insurance_id IN NUMBER,
    p_bill_amount IN NUMBER,
    p_insurance_coverage IN NUMBER,
    p_reimbursement_amount OUT NUMBER
)
AS
BEGIN
    SELECT amount * p_insurance_coverage * p_bill_amount / 100
    INTO p_reimbursement_amount
    FROM insurance
    WHERE insurance_id = p_insurance_id;

    INSERT INTO insurance_claim(patient_id, insurance_id, validity)
    VALUES(p_patient_id, p_insurance_id, 'valid');

    UPDATE bills
    SET insurance_id = p_insurance_id
    WHERE trans_id = p_patient_id;
```

Script Output

Task completed in 0.075 seconds

```
Procedure GENERATE_REIMBURSEMENT compiled
```

```
    WHERE trans_id = p_patient_id;

    COMMIT;
EXCEPTION
    WHEN NO_DATA_FOUND THEN
        p_reimbursement_amount := 0;
        ROLLBACK;
END;
/
```

Script Output

Task completed in 0.075 seconds

```
Procedure GENERATE_REIMBURSEMENT compiled
```

The procedure "generate_reimbursement" is created and compiled which takes input patient_id, insurance_id, bill_amount, insurance_coverage aas percentage. In this procedure, the select is used to find the total amount that could be reimbursed to patient as per eligibility with given parameters.

```
DECLARE
  l_reimbursement_amount NUMBER;
BEGIN
  generate_reimbursement(p_patient_id => '2017',
                         p_insurance_id => 606,
                         p_bill_amount => 100,
                         p_insurance_coverage => 80,
                         p_reimbursement_amount => l_reimbursement_amount);
  DBMS_OUTPUT.PUT_LINE('Reimbursement amount: ' || l_reimbursement_amount);
END;
/
```

Script Output

Task completed in 0.075 seconds

```
Reimbursement amount: 400000


PL/SQL procedure successfully completed.
```

To call/ execute the procedure used anonymous Pl/SQL and the input parameters are given which results out the total reimbursement amount that the patient is eligible for. In above screenshot could see that reimburse amount is maximum of 400000.

Triggers: A trigger is a special type of stored procedure that is automatically executed in response to specific events or actions occurring in a database, such as inserting, updating, or deleting data in a table.

1) Create a trigger to update the employee salary not more than 20% of the existing salary.

```
set serveroutput on;
CREATE OR REPLACE TRIGGER salary_increase_trigger
BEFORE UPDATE ON employee_data
FOR EACH ROW
DECLARE
  max_increase_pct NUMBER := 1.2; -- 20% increase
BEGIN
  IF :new.salary > :old.salary * max_increase_pct THEN
    RAISE_APPLICATION_ERROR(-20001, 'Salary increase cannot exceed 20%.');
  END IF;
END;
/
```

Script Output

Task completed in 0.053 seconds

```
Trigger SALARY_INCREASE_TRIGGER compiled
```

The trigger "salary_increase_trigger" is created and compiled. The trigger will fire before the update and check if the salary increase exceeds 20%. If it does, it will raise an exception as error and the update will fail. If the increase is less than 20%, the update will proceed normally.

In below screenshot, The specified increase in salary exceeds 20% so it returns exception saying that cannot increase salary more than 20% as error.

```
UPDATE employee_data
SET salary = salary * 1.25
WHERE emp_id = 201;
```

Script Output

Task completed in 0.051 seconds

```
Error starting at line : 279 in command -
UPDATE employee_data
SET salary = salary * 1.25
WHERE emp_id = 201
Error report -
ORA-20001: Salary increase cannot exceed 20%.
```

Here in below, The salary hike is 10% which is less than 20%, so trigger updates the employee_data table.

```
UPDATE employee_data
SET salary = salary * 1.1
WHERE emp_id = 201;
```

Script Output

Task completed in 0.031 seconds

```
1 row updated.
```

2) Create a trigger to deduct the insurance amount if the bill amount of specified patient is less than 1000.

```
CREATE OR REPLACE TRIGGER update_insurance_amount
AFTER INSERT OR UPDATE ON bills
FOR EACH ROW
DECLARE
  v_insurance_amount insurance.amount%TYPE;
BEGIN
  SELECT amount INTO v_insurance_amount
  FROM insurance
  WHERE insurance_id = :new.insurance_id;
    dbms_output.put_line('Trigger initiating');
  IF :new.bill_amount < 1000 THEN
    UPDATE insurance
    SET amount = v_insurance_amount - :new.bill_amount
    WHERE insurance_id = :new.insurance_id;
  END IF;
END;
/
```

Script Output

Task completed in 0.066 seconds

```
Trigger UPDATE_INSURANCE_AMOUNT compiled
```

The above trigger “update_insurance_amount” is created and compiled, this trigger will get triggerd on when inserting or updating a record in bills table. This trigger will update the insurance amount of the patient if the bill amount is less than 1000. The usecase here is if the bill amount is less than 1000 then the patient need not to pay any amount at the bill

counter and the bill amount is taken from the insurance and the remaining insurance amount is updated in the insurance.

```
Update bills set bill_amount=50 where trans_id='311';
```

Script Output

Task completed in 0.048 seconds

```
Trigger initiating


1 row updated.
```

The bill amount of particular transaction is updated such that the insurance amount is also updated. To make sure the triggered is trigged or not, given in trigger when trigger is about to initialize, the trigger should report that it is initializing.

3) Create a trigger to display the details of newly added doctor details in doctor table.

Worksheet | Query Builder

```
set serveroutput on;
CREATE OR REPLACE TRIGGER print_doctor_details
AFTER INSERT ON doctor
FOR EACH ROW
DECLARE
   v_doctor_name doctor.doctor_name%TYPE;
   v_specialization doctor.specialization%TYPE;
   v_doctor_id doctor.doctor_id%TYPE;
   v_contact doctor.contact%TYPE;
BEGIN
   v_doctor_name := :NEW.doctor_name;
   v_specialization := :NEW.specialization;
   v_doctor_id := :NEW.doctor_id;
   v_contact := :NEW.contact;

   DBMS_OUTPUT.PUT_LINE('New doctor record inserted:');
   DBMS_OUTPUT.PUT_LINE('Doctor Name: ' || v_doctor_name);
   DBMS_OUTPUT.PUT_LINE('Specialization: ' || v_specialization);
   DBMS_OUTPUT.PUT_LINE('Doctor ID: ' || v_doctor_id);
   DBMS_OUTPUT.PUT_LINE('Contact: ' || v_contact);
END;
/
```

Script Output

Task completed in 0.083 seconds

```
Trigger PRINT_DOCTOR_DETAILS compiled
```

The above code creates a trigger with name as “print_doctor_detail”. This trigger will get triggered whenever a new record is inserted to doctor table. The inserted data is read on

successful insertion into the respective variable names and is printed on to the console using dbms_output.put_line command.

```
INSERT INTO Doctor (doctor_name, specialization, doctor_id, contact) VALUES ('Dr. Sree', 'Neurologist', 9, '9949123094');
```

Script Output

Task completed in 0.083 seconds

```
New doctor record inserted:
Doctor Name: Dr. Vishnu
Specialization: ENT
Doctor ID: 6
Contact: 1234567899
```

The above snippet inserts a record to doctor table and then the trigger gets activated. The trigger prints the newly inserted data on the console.

Package:

A Package is a schema object that groups related functions, procedures, and other types of database objects together in a single, reusable unit. A package is created using the CREATE PACKAGE statement and consists of two parts: a package specification and a package body.

1. Create a package to update the payroll table when the salary of employees is updated.

Package "update_salary" is created which updated the salary of employee in payroll when salary in employee is updated.

```
CREATE OR REPLACE PACKAGE payroll_management_pkg AS
  PROCEDURE update_salary(
    p_emp_id IN employee_data.emp_id%TYPE,
    p_new_salary IN employee_data.salary%TYPE
  );
  FUNCTION get_payroll_salary(
    p_emp_id IN payroll.empid%TYPE
  ) RETURN payroll.payroll_salary%TYPE;
END payroll_management_pkg;
/

CREATE OR REPLACE PACKAGE BODY payroll_management_pkg AS
  PROCEDURE update_salary(
    p_emp_id IN employee_data.emp_id%TYPE,
    p_new_salary IN employee_data.salary%TYPE
  ) IS
  BEGIN
    UPDATE employee_data
    SET salary = p_new_salary
```

Script Output

Task completed in 0.024 seconds

```
Package PAYROLL_MANAGEMENT_PKG compiled


Package Body PAYROLL_MANAGEMENT_PKG compiled
```

In above screenshot, Package “update_salary” is initialized which declares a procedure and function. The Procedure takes input arguments of employee_id and salary. The function takes input employee_id and returns the payroll of that employee.

The package body is created with procedure and function, procedure initially, updates the given salary in the employee_data table and then updates the salary in payroll.

```
    SET salary = p_new_salary
    WHERE emp_id = p_emp_id;

    UPDATE payroll
    SET payroll_salary = p_new_salary
    WHERE empid = p_emp_id;

    COMMIT;
  END;

  FUNCTION get_payroll_salary(
    p_emp_id IN payroll.empid%TYPE
  ) RETURN payroll.payroll_salary%TYPE IS
    v_payroll_salary payroll.payroll_salary%TYPE;
  BEGIN
    SELECT payroll_salary
    INTO v_payroll_salary
    FROM payroll
    WHERE empid = p_emp_id;
```

Script Output

Task completed in 0.024 seconds

```
Package PAYROLL_MANAGEMENT_PKG compiled


Package Body PAYROLL_MANAGEMENT_PKG compiled
```

Once, the procedure is committed, the function is called which displays the payroll salary of specified employee_id.

```
    RETURN v_payroll_salary;
  END;
END payroll_management_pkg;
/
```

Script Output

Task completed in 0.024 seconds

```
Package PAYROLL_MANAGEMENT_PKG compiled


Package Body PAYROLL_MANAGEMENT_PKG compiled
```

After the package and package body is compiled, then package is executed by giving the inputs as emp_id and salary.

As for the package process, the procedure must be executed and then the function must be called.

Here, when package is executed, the procedure is successfully completed which means the tables are updated. But as there is a already a trigger created on employee details before, when there is a change in employee_data table then the trigger got fired so, there displayed a output “I got triggered” in the console as shown in below screenshot.

```
execute payroll_management_pkg.update_salary('206',500);
```

Script Output

Task completed in 0.033 seconds

```
I got triggered


PL/SQL procedure successfully completed.
```

After the procedure is executed, the function has to be called to display the updated payroll salary of specified employee_id as shown in below screenshot.

```
declare
  emp_id Varchar2(5);
begin
  emp_id := '206';
  dbms_output.put_line('The Employee with id = ' || emp_id || ', salary details are ' || payroll_management_pkg.get_payroll_salary(emp_id));
end;
/
```

Script Output

Task completed in 0.075 seconds

```
The Employee with id = 206, salary details are 500


PL/SQL procedure successfully completed.
```

Individual contribution: -

- I have created the function to find the total bill amount for a particular prescription (function 5) and executed it.
- I have created the stored procedure to find the re-imbursement amount of a given patient (procedure 5) and executed it.
- I have created 2 triggers.
  - Create a trigger to update the employee salary not more than 20% of the existing salary (trigger 1) and executed it.
  - Create a trigger to deduct the insurance amount if the bill amount of specified patient is less than 1000 (trigger 2) and executed it.
- Created the package to update the payroll table when the salary of employees is updated.
- I have distributed the work to teammates, monitored and guided them with their respective distributed work.
- I have cross verified my teammates' works and their tasks and their answers.
- As a team, we have collaborated with each other to complete this project Part 4.