SAMPLE QUESTIONS FOR INTERNAL2

JOINS

1. find the salesperson and customer who reside in the same city. Return Salesman, cust_name and city.

   ```
   SELECT s.salesman, c.cust_name, c.city
   FROM salesman s
   JOIN customer c ON s.city = c.city;
   ```

2. From the following tables write a SQL query to find those orders where the order amount exists between 500 and 2000. Return ord_no, purch_amt, cust_name, city.

   ```
   SELECT o.ord_no, o.purch_amt, c.cust_name, c.city
   FROM orders o
   JOIN customer c ON o.cust_id = c.cust_id
   WHERE o.purch_amt BETWEEN 500 AND 2000;
   ```

3. write a SQL query to find salespeople who received commissions of more than 12 percent from the company. Return Customer Name, customer city, Salesman, commission.

   ```
   SELECT c.cust_name, c.city, s.salesman, s.commission
   FROM salesman s
   JOIN customer c ON s.salesman_id = c.salesman_id
   WHERE s.commission > 12;
   ```

4. Write a SQL statement to join the tables salesman, customer and orders so that the same column of each table appears once and only the relational rows are returned.

   ```
   SELECT s.salesman, c.cust_name, o.ord_no, o.purch_amt
   FROM salesman s
   JOIN customer c ON s.salesman_id = c.salesman_id
   JOIN orders o ON c.cust_id = o.cust_id;
   ```

5. write a SQL query to display the customer name, customer city, grade, salesman, salesman city. The results should be sorted by ascending customer_id.

   ```
   SELECT c.cust_name, c.city, c.grade, s.salesman, s.city AS salesman_city
   FROM customer c
   JOIN salesman s ON c.salesman_id = s.salesman_id
   ORDER BY c.customer_id ASC;
   ```

VIEWS

1. Create a view supplier_view with columns sid, sname, pid. See the contents of the view created.

```
CREATE VIEW supplier_view AS
SELECT sid, sname, pid FROM supplier;

SELECT * FROM supplier_view;
```

2. Create a complex view on customers and orders and check the base tables after updation

```
CREATE VIEW customer_orders_view AS
SELECT c.cust_name, o.ord_no, o.purch_amt
FROM customer c
JOIN orders o ON c.cust_id = o.cust_id;

UPDATE customer_orders_view
SET purch_amt = 1500
WHERE ord_no = 1;

SELECT * FROM orders;
```

3. create a view for displaying a sname whose sid=4?

```
CREATE VIEW sname_view AS
SELECT sname FROM supplier WHERE sid = 4;
```

4. Create and Update a view for Passenger table as ticket_no,ppno,gender.

```
CREATE VIEW passenger_view AS
SELECT ticket_no, ppno, gender FROM passenger;

UPDATE passenger_view
SET gender = 'F'
WHERE ticket_no = 123;
```

5. create a view for displaying sname having cid=3?

```
CREATE VIEW supplier_name_view AS
SELECT sname FROM supplier WHERE cid = 3;
```

PL/SQL, Functions,Procedures

1. Write a PL/SQL code to check given number is Armstrong or not.

```
DECLARE
  num NUMBER := &num;
  temp NUMBER;
  sum NUMBER := 0;
  r NUMBER;
BEGIN
  temp := num;
  WHILE temp > 0 LOOP
    r := MOD(temp, 10);
    sum := sum + r*r*r;
    temp := temp / 10;
  END LOOP;
  IF sum = num THEN
    DBMS_OUTPUT.PUT_LINE(num || ' is an Armstrong number');
  ELSE
    DBMS_OUTPUT.PUT_LINE(num || ' is not an Armstrong number');
  END IF;
END;
```

2. Write a PL/SQL code to implement calculator program

```
DECLARE
  a NUMBER := &a;
  b NUMBER := &b;
  result NUMBER;
BEGIN
  result := a + b;
  DBMS_OUTPUT.PUT_LINE('Addition: ' || result);
  result := a - b;
  DBMS_OUTPUT.PUT_LINE('Subtraction: ' || result);
  result := a * b;
  DBMS_OUTPUT.PUT_LINE('Multiplication: ' || result);
  result := a / b;
  DBMS_OUTPUT.PUT_LINE('Division: ' || result);
END;
```

3. Write query PL/SQL procedure to find factorial of a number

```
CREATE OR REPLACE PROCEDURE factorial(num IN NUMBER, fact OUT NUMBER)
AS
BEGIN
  fact := 1;
  FOR i IN 1..num LOOP
    fact := fact * i;
  END LOOP;
END;
```

4. Write a function to find cube of a number passed as an argument

```
CREATE OR REPLACE FUNCTION cube_number(num IN NUMBER) RETURN
NUMBER IS
  result NUMBER;
BEGIN
  result := num * num * num;
  RETURN result;
END;
```

5. Write a function to find perfect number

```
CREATE OR REPLACE FUNCTION is_perfect(num IN NUMBER) RETURN
VARCHAR2 IS
  sum NUMBER := 0;
BEGIN
  FOR i IN 1..num-1 LOOP
    IF MOD(num, i) = 0 THEN
      sum := sum + i;
    END IF;
  END LOOP;
  IF sum = num THEN
    RETURN 'Perfect';
  ELSE
    RETURN 'Not Perfect';
  END IF;
END;
```

6. Using procedures find he sum of digits of a number

```
CREATE OR REPLACE PROCEDURE sum_of_digits(num IN NUMBER, sum OUT
NUMBER) AS
  digit NUMBER;
BEGIN
  sum := 0;
  WHILE num > 0 LOOP
    digit := MOD(num, 10);
    sum := sum + digit;
    num := num / 10;
  END LOOP;
END;
```

7. Using functions find gcd of 2 numbers

```
CREATE OR REPLACE FUNCTION gcd(a IN NUMBER, b IN NUMBER) RETURN
NUMBER IS
BEGIN
  IF b = 0 THEN
    RETURN a;
  ELSE
    RETURN gcd(b, MOD(a, b));
  END IF;
```

```
END;
```

8. Write a PL/SQL code to check given number is even or not.

```
CREATE OR REPLACE PROCEDURE check_even(num IN NUMBER) AS
BEGIN
  IF MOD(num, 2) = 0 THEN
    DBMS_OUTPUT.PUT_LINE(num || ' is even');
  ELSE
    DBMS_OUTPUT.PUT_LINE(num || ' is odd');
  END IF;
END;
```

9. Write a procedure to find reverse of a number

```
CREATE OR REPLACE PROCEDURE reverse_number(num IN NUMBER, reversed OUT
NUMBER) AS
  temp NUMBER := num;
  remainder NUMBER;
BEGIN
  reversed := 0;
  WHILE temp > 0 LOOP
    remainder := MOD(temp, 10);
    reversed := (reversed * 10) + remainder;
    temp := temp / 10;
  END LOOP;
END;
```

10. Write a function to check a number is prime or not

```
CREATE OR REPLACE FUNCTION is_prime(num IN NUMBER) RETURN
VARCHAR2 IS
  i NUMBER;
BEGIN
  IF num <= 1 THEN
    RETURN 'Not Prime';
  END IF;
  FOR i IN 2..SQRT(num) LOOP
    IF MOD(num, i) = 0 THEN
      RETURN 'Not Prime';
    END IF;
  END LOOP;
  RETURN 'Prime';
END;
```

CURSORS (can be asked as explicit or implicit)

1. Write a cursor to find name, id and age of employees whose name starts with letter ‘P’.

```
DECLARE
  CURSOR emp_cursor IS
    SELECT emp_name, emp_id, age
    FROM employees
    WHERE emp_name LIKE 'P%';  -- Names starting with 'P'
  emp_row emp_cursor%ROWTYPE;
BEGIN
  OPEN emp_cursor;
  LOOP
    FETCH emp_cursor INTO emp_row;
    EXIT WHEN emp_cursor%NOTFOUND;  -- Exit when no more rows are found
    DBMS_OUTPUT.PUT_LINE('Name: ' || emp_row.emp_name || ', ID: ' ||
emp_row.emp_id || ', Age: ' || emp_row.age);
  END LOOP;
  CLOSE emp_cursor;
END;
```

```
DECLARE
  emp_name employees.emp_name%TYPE;
  emp_id employees.emp_id%TYPE;
  age employees.age%TYPE;
BEGIN
  -- Implicit cursor for selecting one employee whose name starts with 'P'
  SELECT emp_name, emp_id, age
  INTO emp_name, emp_id, age
  FROM employees
  WHERE emp_name LIKE 'P%' AND ROWNUM = 1;  -- Fetch one row only

  DBMS_OUTPUT.PUT_LINE('Name: ' || emp_name || ', ID: ' || emp_id || ', Age: ' || age);
END;
```

2. Write a cursor to find names of passengers who travel on RED BUS

```
DECLARE
  CURSOR red_bus_cursor IS
    SELECT passenger_name
    FROM passengers
    WHERE bus_name = 'RED BUS';
  pass_row red_bus_cursor%ROWTYPE;
BEGIN
  OPEN red_bus_cursor;
  LOOP
    FETCH red_bus_cursor INTO pass_row;
    EXIT WHEN red_bus_cursor%NOTFOUND;
    DBMS_OUTPUT.PUT_LINE('Passenger: ' || pass_row.passenger_name);
  END LOOP;
  CLOSE red_bus_cursor;
```

```
END;


DECLARE
  passenger_name passengers.passenger_name%TYPE;
BEGIN
  -- Implicit cursor for selecting one passenger traveling on 'RED BUS'
  SELECT passenger_name INTO passenger_name
  FROM passengers
  WHERE bus_name = 'RED BUS' AND ROWNUM = 1;

  DBMS_OUTPUT.PUT_LINE('Passenger: ' || passenger_name);
END;
```

3. Write a cursor to display names of faculty who teach “Java Programming”.

```
DECLARE
  CURSOR java_faculty_cursor IS
    SELECT faculty_name
    FROM faculty
    WHERE subject = 'Java Programming';
  fac_row java_faculty_cursor%ROWTYPE;
BEGIN
  OPEN java_faculty_cursor;
  LOOP
    FETCH java_faculty_cursor INTO fac_row;
    EXIT WHEN java_faculty_cursor%NOTFOUND;
    DBMS_OUTPUT.PUT_LINE('Faculty: ' || fac_row.faculty_name);
  END LOOP;
  CLOSE java_faculty_cursor;
END;


DECLARE
  faculty_name faculty.faculty_name%TYPE;
BEGIN
  -- Implicit cursor for selecting one faculty teaching 'Java Programming'
  SELECT faculty_name INTO faculty_name
  FROM faculty
  WHERE subject = 'Java Programming' AND ROWNUM = 1;

  DBMS_OUTPUT.PUT_LINE('Faculty: ' || faculty_name);
END;
```

4. Write a cursor to List the employees along with their Experience and Daily Salary.

```
DECLARE
  CURSOR emp_exp_salary_cursor IS
    SELECT emp_name, experience, salary/30 AS daily_salary
    FROM employees;
  emp_row emp_exp_salary_cursor%ROWTYPE;
BEGIN
```

```
  OPEN emp_exp_salary_cursor;
  LOOP
    FETCH emp_exp_salary_cursor INTO emp_row;
    EXIT WHEN emp_exp_salary_cursor%NOTFOUND;
    DBMS_OUTPUT.PUT_LINE('Name: ' || emp_row.emp_name || ', Experience: ' ||
emp_row.experience || ', Daily Salary: ' || emp_row.daily_salary);
  END LOOP;
  CLOSE emp_exp_salary_cursor;
END;


DECLARE
  emp_name employees.emp_name%TYPE;
  experience employees.experience%TYPE;
  daily_salary employees.salary%TYPE;
BEGIN
  -- Implicit cursor for selecting one employee's experience and daily salary
  SELECT emp_name, experience, salary/30 INTO emp_name, experience, daily_salary
  FROM employees
  WHERE ROWNUM = 1;

  DBMS_OUTPUT.PUT_LINE('Name: ' || emp_name || ', Experience: ' || experience || ',
Daily Salary: ' || daily_salary);
END;
```

5. Write a cursor to list names of doctors whose salary is greater than doctor “Jhon”.

```
DECLARE
  CURSOR doctor_cursor IS
    SELECT doctor_name
    FROM doctors
    WHERE salary > (SELECT salary FROM doctors WHERE doctor_name = 'Jhon');
  doc_row doctor_cursor%ROWTYPE;
BEGIN
  OPEN doctor_cursor;
  LOOP
    FETCH doctor_cursor INTO doc_row;
    EXIT WHEN doctor_cursor%NOTFOUND;
    DBMS_OUTPUT.PUT_LINE('Doctor: ' || doc_row.doctor_name);
  END LOOP;
  CLOSE doctor_cursor;
END;


DECLARE
  doctor_name doctors.doctor_name%TYPE;
BEGIN
  -- Implicit cursor for selecting one doctor whose salary is greater than Jhon's salary
  SELECT doctor_name INTO doctor_name
  FROM doctors
  WHERE salary > (SELECT salary FROM doctors WHERE doctor_name = 'Jhon') AND
ROWNUM = 1;
```

```
    DBMS_OUTPUT.PUT_LINE('Doctor: ' || doctor_name);
END;
```

Triggers

1. Write a trigger to check age validity of a customer using row level triggers.
   (Age should not be less than 20)

   ```
   CREATE OR REPLACE TRIGGER age_check_trigger

   BEFORE INSERT OR UPDATE ON customer

   FOR EACH ROW

   BEGIN

     IF :NEW.age < 20 THEN

       RAISE_APPLICATION_ERROR(-20001, 'Age must be at least 20.');

     END IF;

   END;
   ```

2. Create a Trigger for one instance of student table it will update another table while inserting values.

   ```
   CREATE OR REPLACE TRIGGER student_update_trigger

   AFTER INSERT ON student

   FOR EACH ROW

   BEGIN

     UPDATE other_table

     SET related_column = :NEW.student_column

     WHERE condition_column = :NEW.student_id;

   END;
   ```

3. Create a row level after trigger on customer table.

   ```
   CREATE OR REPLACE TRIGGER log_customer_insert

   AFTER INSERT ON customer

   FOR EACH ROW

   BEGIN
   ```

```
    INSERT INTO customer_log (cust_id, log_date)
    VALUES (:NEW.cust_id, SYSDATE);
END;
```

4. Create a statement level trigger in employee table.

```
CREATE OR REPLACE TRIGGER log_employee_update
AFTER UPDATE ON employee
BEGIN
    INSERT INTO employee_log (log_message, log_date)
    VALUES ('An update occurred on employee table', SYSDATE);
END;
```

5. Create an after trigger to update rows in book relation

```
CREATE OR REPLACE TRIGGER update_book_rows
AFTER UPDATE ON book
FOR EACH ROW
BEGIN
    UPDATE book
    SET last_modified = SYSDATE
    WHERE book_id = :OLD.book_id;
END;
```