SAMPLE QUESTIONS FOR INTERNAL2

JOINS

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

VIEWS

1. Create a view supplier_view with columns sid, sname, pid. See the contents of the view created.
2. Create a complex view on customers and orders and check the base tables after updation
3. create a view for displaying a sname whose sid=4?
4. Create and Update a view for Passenger table as ticket_no,ppno,gender.
5. create a view for displaying sname having cid=3?

PL/SQL, Functions,Procedures

1. Write a PL/SQL code to check given number is Armstrong or not.
2. Write a PL/SQL code to implement calculator program
3. Write query PL/SQL procedure to find factorial of a number
4. Write a function to find cube of a number passed as an argument
5. Write a function to find perfect number
6. Using procedures find he sum of digits of a number
7. Using functions find gcd of 2 numbers

8. Write a PL/SQL code to check given number is even or not.
9. Write a procedure to find reverse of a number
10. Write a function to check a number is prime or not

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’.
2. Write a cursor to find names of passengers who travel on RED BUS
3. Write a cursor to display names of faculty who teach “Java Programming”.
4. Write a cursor to List the employees along with their Experience and Daily Salary.
5. Write a cursor to list names of doctors whose salary is greater than doctor “Jhon”.

Triggers

1. Write a trigger to check age validity of a customer using row level triggers.
   (Age should not be less than 20)
2. Create a Trigger for one instance of student table it will update another table while inserting values.
3. Create a row level after trigger on customer table.
4. Create a statement level trigger in employee table.
5. Create an after trigger to update rows in book relation

Here are the SQL and PL/SQL solutions for the given questions:

JOINS

1. **Find the salesperson and customer who reside in the same city. Return Salesman, cust_name, and city:**

```sql
SELECT s.name AS Salesman, c.cust_name, c.city
FROM salesman s
JOIN customer c ON s.city = c.city;
```

2. **Find those orders where the order amount exists between 500 and 2000. Return ord_no, purch_amt, cust_name, city:**

```sql
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. **Find salespeople who received commissions of more than 12 percent. Return Customer Name, customer city, Salesman, commission:**

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

4. **Join the tables salesman, customer, and orders so that the same column of each table appears once and only the relational rows are returned:**

```sql
SELECT s.name AS 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. **Display the customer name, customer city, grade, salesman, salesman city. Sorted by ascending customer_id:**

```sql
SELECT c.cust_name, c.city, c.grade, s.name AS Salesman, s.city AS Salesman_City
FROM customer c
JOIN salesman s ON c.salesman_id = s.salesman_id
ORDER BY c.cust_id ASC;
```

VIEWS

1. **Create a view `supplier_view` with columns sid, sname, pid:**

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

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

```sql
CREATE VIEW customer_orders AS
```

```sql
SELECT c.cust_name, c.city, o.ord_no, o.purch_amt
FROM customer c
JOIN orders o ON c.cust_id = o.cust_id;

-- To check base tables after update
SELECT * FROM customer;
SELECT * FROM orders;
```

3. **Create a view for displaying `sname` where `sid = 4`:**

```sql
CREATE VIEW supplier_sid4 AS
SELECT sname
FROM supplier
WHERE sid = 4;
```

4. **Create and update a view for Passenger table as `ticket_no`, `ppno`, `gender`:**

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

-- Update the view
UPDATE passenger_view
SET gender = 'M'
WHERE ppno = 'P1234';
```

5. **Create a view for displaying `sname` having `cid = 3`:**

```sql
CREATE VIEW supplier_cid3 AS
SELECT sname
FROM supplier
WHERE cid = 3;
```

PL/SQL, Functions, Procedures

1. **PL/SQL code to check if a given number is Armstrong or not:**

```plsql
DECLARE
  num NUMBER := &input_num;
  sum NUMBER := 0;
  digit NUMBER;
  temp NUMBER;
BEGIN
  temp := num;
```

```
  WHILE temp > 0 LOOP
    digit := MOD(temp, 10);
    sum := sum + POWER(digit, 3);
    temp := FLOOR(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. **PL/SQL code to implement a calculator program:**

```plsql
DECLARE
  num1 NUMBER := &num1;
  num2 NUMBER := &num2;
  operation CHAR(1) := '&operation'; -- (+, -, *, /)
  result NUMBER;
BEGIN
  CASE operation
    WHEN '+' THEN result := num1 + num2;
    WHEN '-' THEN result := num1 - num2;
    WHEN '*' THEN result := num1 * num2;
    WHEN '/' THEN result := num1 / num2;
    ELSE DBMS_OUTPUT.PUT_LINE('Invalid operation');
  END CASE;
  DBMS_OUTPUT.PUT_LINE('Result: ' || result);
END;
```

3. **Procedure to find factorial of a number:**

```plsql
CREATE OR REPLACE PROCEDURE find_factorial(n IN NUMBER, fact OUT NUMBER) IS
BEGIN
  fact := 1;
  FOR i IN 1..n LOOP
    fact := fact * i;
  END LOOP;
END;
```

4. **Function to find the cube of a number:**

```plsql
CREATE OR REPLACE FUNCTION cube(n IN NUMBER) RETURN NUMBER IS
BEGIN
  RETURN n * n * n;
END;
```

```

5. **Function to find if a number is perfect:**

```plsql
CREATE OR REPLACE FUNCTION is_perfect(n IN NUMBER) RETURN BOOLEAN IS
  sum NUMBER := 0;
BEGIN
  FOR i IN 1..n/2 LOOP
    IF MOD(n, i) = 0 THEN
      sum := sum + i;
    END IF;
  END LOOP;
  RETURN sum = n;
END;
```

6. **Procedure to find the sum of digits of a number:**

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

7. **Function to find GCD of two numbers:**

```plsql
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. **PL/SQL code to check if a given number is even or not:**

```plsql
DECLARE
  num NUMBER := &input_num;
```
```

```
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. **Procedure to find the reverse of a number:**

```plsql
CREATE OR REPLACE PROCEDURE reverse_number(num IN NUMBER, rev OUT NUMBER)
IS
  digit NUMBER;
BEGIN
  rev := 0;
  WHILE num > 0 LOOP
    digit := MOD(num, 10);
    rev := rev * 10 + digit;
    num := FLOOR(num / 10);
  END LOOP;
END;
```

10. **Function to check if a number is prime or not:**

```plsql
CREATE OR REPLACE FUNCTION is_prime(n IN NUMBER) RETURN BOOLEAN IS
BEGIN
  IF n <= 1 THEN
    RETURN FALSE;
  END IF;
  FOR i IN 2..FLOOR(SQRT(n)) LOOP
    IF MOD(n, i) = 0 THEN
      RETURN FALSE;
    END IF;
  END LOOP;
  RETURN TRUE;
END;
```

CURSORS

1. **Cursor to find name, id, and age of employees whose name starts with letter 'P':**

```plsql
DECLARE
  CURSOR emp_cursor IS
    SELECT name, id, age FROM employees WHERE name LIKE 'P%';
BEGIN
  FOR rec IN emp_cursor LOOP
```

```
    DBMS_OUTPUT.PUT_LINE('Name: ' || rec.name || ', ID: ' || rec.id || ', Age: ' || rec.age);
  END LOOP;
END;
```

2. **Cursor to find names of passengers who travel on RED BUS:**

```plsql
DECLARE
  CURSOR pass_cursor IS
    SELECT name FROM passengers WHERE bus_name = 'RED BUS';
BEGIN
  FOR rec IN pass_cursor LOOP
    DBMS_OUTPUT.PUT_LINE('Passenger Name: ' || rec.name);
  END LOOP;
END;
```

3. **Cursor to display names of faculty who teach "Java Programming":**

```plsql
DECLARE
  CURSOR faculty_cursor IS
    SELECT name FROM faculty WHERE subject = 'Java Programming';
BEGIN
  FOR rec IN faculty_cursor LOOP
    DBMS_OUTPUT.PUT_LINE('Faculty Name: ' || rec.name);
  END LOOP;
END;
```

4. **Cursor to list the employees along with their Experience and Daily Salary:**

```plsql
DECLARE
  CURSOR emp_cursor IS
    SELECT name, experience, salary/30 AS daily_salary FROM employees;
BEGIN
  FOR rec IN emp_cursor LOOP
    DBMS_OUTPUT.PUT_LINE('Name: ' || rec.name || ', Experience: ' || rec.experience || ', Daily
Salary: ' || rec.daily_salary);
  END LOOP;
END;
```

5. **Cursor to list names of doctors whose salary is greater than doctor "John":**

```plsql
DECLARE
  john_salary NUMBER;
  CURSOR doc_cursor IS
    SELECT name FROM doctors WHERE salary >
```

```
 john_salary;
BEGIN
  SELECT salary INTO john_salary FROM doctors WHERE name = 'John';
  FOR rec IN doc_cursor LOOP
    DBMS_OUTPUT.PUT_LINE('Doctor Name: ' || rec.name);
  END LOOP;
END;
```

TRIGGERS

1. **Trigger to check age validity of a customer (age should not be less than 20):**

```plsql
CREATE OR REPLACE TRIGGER check_age
BEFORE INSERT OR UPDATE ON customer
FOR EACH ROW
BEGIN
  IF :NEW.age < 20 THEN
    RAISE_APPLICATION_ERROR(-20001, 'Customer age cannot be less than 20.');
  END IF;
END;
```

2. **Trigger to update another table while inserting values in student table:**

```plsql
CREATE OR REPLACE TRIGGER update_another_table
AFTER INSERT ON student
FOR EACH ROW
BEGIN
  INSERT INTO another_table (student_id, student_name)
  VALUES (:NEW.student_id, :NEW.student_name);
END;
```

3. **Row-level after trigger on customer table:**

```plsql
CREATE OR REPLACE TRIGGER after_customer_insert
AFTER INSERT ON customer
FOR EACH ROW
BEGIN
  DBMS_OUTPUT.PUT_LINE('Customer inserted with ID: ' || :NEW.cust_id);
END;
```

4. **Statement-level trigger on employee table:**

```plsql
CREATE OR REPLACE TRIGGER statement_level_trigger
```

```
AFTER INSERT ON employee
BEGIN
  DBMS_OUTPUT.PUT_LINE('A new employee record has been inserted.');
END;
```

5. **After trigger to update rows in the book relation:**

```plsql
CREATE OR REPLACE TRIGGER update_book_relation
AFTER UPDATE ON book
FOR EACH ROW
BEGIN
  DBMS_OUTPUT.PUT_LINE('Book record updated with ID: ' || :NEW.book_id);
END;
```