By Maha Rana

CS405 Midterm Subjective Solved

Short Questions:

1. Syntax of IF-THEN-ELSE Statement

The **IF-THEN-ELSE** statement in PL/SQL is used for conditional execution.

✅**Syntax:**

```
IF condition THEN
   -- Execute when condition is TRUE
ELSE
   -- Execute when condition is FALSE
END IF;
```

✅**Example:**

```
DECLARE
   salary NUMBER := 5000;
BEGIN
   IF salary > 4000 THEN
      DBMS_OUTPUT.PUT_LINE('High Salary');
   ELSE
      DBMS_OUTPUT.PUT_LINE('Low Salary');
   END IF;
END;
```

 Output (if salary = 5000):

```
High Salary
```

By Maha Rana

2. Syntax of HAVING Clause

The **HAVING** clause is used in SQL to filter **groups of records** created by the `GROUP BY` clause.

✅**Syntax:**

SELECT column_name, COUNT(*)

FROM table_name

GROUP BY column_name

HAVING condition;

✅**Example:**

```
SELECT deptno, AVG(salary)
FROM employees
GROUP BY deptno
HAVING AVG(salary) > 50000;
```

Explanation:

- `GROUP BY deptno` groups records based on department number.
- `HAVING AVG(salary) > 50000` filters only those departments where **average salary** is greater than `50,000`.

✅**Key Difference:**

- `WHERE` is used to filter **individual rows**.
- `HAVING` is used to filter **groups after aggregation**.

3. Difference between Primary Key and Unique Key

By Maha Rana

Feature	Primary Key	Unique Key
Uniqueness	Ensures each row is **unique**	Ensures each row is **unique**
NULL Allowed?	❌ No (NOT NULL constraint)	✅ Yes (can contain NULL values)
Number of Keys per Table	**Only One** Primary Key	**Multiple** Unique Keys allowed
Index Type	**Clustered Index**	**Non-clustered Index**
Example	`PRIMARY KEY (emp_id)`	`UNIQUE (email)`

✅Example Table:

```
CREATE TABLE employees (
   emp_id NUMBER PRIMARY KEY,  -- Primary Key (No NULL, Unique)
   email VARCHAR2(100) UNIQUE   -- Unique Key (NULL Allowed, Unique)
);
```

Long Questions:

1. Difference Between GROUP FUNCTION and GROUP BY Clause

Feature	GROUP FUNCTION	GROUP BY Clause
Definition	Performs calculations on multiple rows	Groups rows based on a column
Used With	Works with `SUM()`, `AVG()`, `COUNT()`, etc.	Used in conjunction with **GROUP FUNCTIONS**
Example	`SELECT AVG(salary) FROM employees;`	`SELECT deptno, AVG(salary) FROM employees GROUP BY deptno;`

✅Example of GROUP FUNCTION (No GROUP BY)

```
SELECT COUNT(*) FROM employees;
```

 Output: Returns the **total number of employees**.

✅Example of GROUP BY Clause

By Maha Rana

```
SELECT deptno, COUNT(*) FROM employees GROUP BY deptno;
```

 Output: Groups employees by `deptno` and counts the number of employees in each department.

✅**Key Difference:**

- **GROUP FUNCTIONS** perform operations **on multiple rows**.
- **GROUP BY** groups **rows based on a column** and then applies group functions.

---

2. Query-Based Code (Assumed Example)

✅**Retrieve Employee Names with Salary Greater Than Average Salary**

```
DECLARE
   avg_sal NUMBER;
BEGIN
   -- Get the average salary
   SELECT AVG(salary) INTO avg_sal FROM employees;

   -- Display employees earning more than the average
salary
   FOR emp IN (SELECT ename, salary FROM employees
WHERE salary > avg_sal) LOOP
      DBMS_OUTPUT.PUT_LINE(emp.ename || ' earns ' ||
emp.salary);
   END LOOP;
END;
```

 Purpose: Finds and prints all employees earning **above the average salary**.

3. Multiplication of First 10 Natural Numbers with 10

✅**Loop-Based PL/SQL Code**

By Maha Rana

```
DECLARE
   i NUMBER;
BEGIN
   FOR i IN 1..10 LOOP
      DBMS_OUTPUT.PUT_LINE(i || ' x 10 = ' || (i *
10));
   END LOOP;
END;
```

 Output:

```
1 x 10 = 10
2 x 10 = 20
3 x 10 = 30
4 x 10 = 40
5 x 10 = 50
6 x 10 = 60
7 x 10 = 70
8 x 10 = 80
9 x 10 = 90
10 x 10 = 100
```

Short Questions:

1. PL/SQL Command to Print a Message

In PL/SQL, we use **`DBMS_OUTPUT.PUT_LINE`** to print messages.

✅**PL/SQL Print Command:**

By Maha Rana

```plsql
BEGIN
   DBMS_OUTPUT.PUT_LINE('Hello, PL/SQL!');
END;
```

👉 **Output:**

```sql
Hello, PL/SQL!
```

- `DBMS_OUTPUT.PUT_LINE` is used to print **output in PL/SQL programs**.
- Enable **Server Output** using:

```
SET SERVEROUTPUT ON;
```

2. Finding Error in Given SQL Command

❌**Example of Incorrect SQL Query:**

```
SELECT * FROM employee WHERE name = Ali;
```

✅**Error:** Missing single quotes (`'Ali'` should be in quotes).

✅**Correct Query:**

```
SELECT * FROM employee WHERE name = 'Ali';
```

✅**Other Common Errors:**

❌**Using `SELECT` without `FROM` clause**
❌**Incorrect column names that don't exist in the table**
❌**Forgetting `WHERE` condition for `UPDATE` or `DELETE` queries.**

3. ROLLBACK Command and Its Syntax

By Maha Rana

`ROLLBACK` is a SQL command used to **undo uncommitted transactions**.

✅Syntax of ROLLBACK:

```
ROLLBACK;
```

✅Example Usage:

```
BEGIN
   INSERT INTO employees (emp_id, name, salary) VALUES
(101, 'Ali', 50000);
   ROLLBACK;  -- Undo the INSERT operation
END;
```

 Tip: If `COMMIT` is **not** executed, `ROLLBACK` **undoes** the changes.

Long Questions:

1. **Basic Syntax of a PL/SQL Block**

PL/SQL programs follow a **block structure**.

✅Basic PL/SQL Block Syntax:

```
DECLARE
   -- Variable declarations
BEGIN
   -- Executable statements
EXCEPTION
   -- Exception handling (optional)
END;
```

✅Example:

```
DECLARE
   message VARCHAR2(50) := 'Hello, PL/SQL!';
BEGIN
   DBMS_OUTPUT.PUT_LINE(message);
END;
```

Tip: The `DECLARE` and `EXCEPTION` sections are **optional**.

By Maha Rana

2. PL/SQL Code for ROLLBACK (Insert, Update & Display Row Count)

This PL/SQL block will:

1. **Insert a new row** into a table.
2. **Update an existing row**.
3. **Display the number of rows before and after ROLLBACK**.

✅PL/SQL Code:

```
DECLARE
   row_count NUMBER;
BEGIN
   -- Insert a new employee
   INSERT INTO employees (emp_id, name, salary) VALUES (102, 'Sara', 60000);

   -- Update salary of an employee
   UPDATE employees SET salary = 70000 WHERE emp_id = 101;

   -- Display number of rows affected
   SELECT COUNT(*) INTO row_count FROM employees;
   DBMS_OUTPUT.PUT_LINE('Total Rows Before ROLLBACK: ' || row_count);

   -- Undo changes
   ROLLBACK;

   -- Check row count after rollback
   SELECT COUNT(*) INTO row_count FROM employees;
   DBMS_OUTPUT.PUT_LINE('Total Rows After ROLLBACK: ' || row_count);
END;
```

Expected Output:

```
Total Rows Before ROLLBACK: X  -- (X includes new row)
Total Rows After ROLLBACK: Y   -- (Y is original row count)
```

✅Tip:

- `ROLLBACK` **undoes** all changes made **after the last COMMIT**.

- `SELECT COUNT(*)` is used to count **total rows before and after rollback**

3. Scenario-Based Entity Identification

In database design, **Entities** represent **real-world objects**. For example, if the scenario was about a **Library System**, the entities might be:

1. **Book**
2. **Student**
3. **Librarian**
4. **Loan Transaction**

✅Example ERD (Entity-Relationship Diagram) Entities:

Entity	Attributes
`Student`	Student_ID, Name, Age, Address
`Book`	Book_ID, Title, Author, ISBN
`Loan`	Loan_ID, Student_ID, Book_ID, Issue_Date, Return_Date

Short Questions:

1. What is a Foreign Key?

A **foreign key** is a column (or a set of columns) that **references the primary key** of another table. It ensures **referential integrity** between tables.

✅Key Points:

- Used to **create relationships** between two tables.
- Helps **prevent orphan records** (records in the child table without a corresponding parent).
- **Referenced column must be a primary key** in the parent table.

✅Example of Foreign Key:

By Maha Rana

```
CREATE TABLE Students (
   student_id NUMBER PRIMARY KEY,
   name VARCHAR2(50),
   class_id NUMBER
);

CREATE TABLE Classes (
   class_id NUMBER PRIMARY KEY,
   class_name VARCHAR2(50)
);

ALTER TABLE Students
ADD CONSTRAINT fk_class FOREIGN KEY (class_id) REFERENCES Classes(class_id);
```

 Here, class_id in Students references class_id in Classes.
 If a class is **deleted**, it prevents **students from referencing a non-existing class**.

2. PL/SQL Code to Display Student Name and Address using `SELECT INTO`

✅**Scenario:** A **student table** is given, and we need to retrieve **Student Name and Address** using `SELECT INTO`.

✅**PL/SQL Code:**

By Maha Rana

```
DECLARE
   student_name VARCHAR2(50);
   student_address VARCHAR2(100);
   student_id NUMBER := 101; -- Assuming we fetch details for Student ID 101
BEGIN
   SELECT name, address
   INTO student_name, student_address
   FROM Students
   WHERE student_id = 101;

   DBMS_OUTPUT.PUT_LINE('Student Name: ' || student_name);
   DBMS_OUTPUT.PUT_LINE('Student Address: ' || student_address);
END;
```

Explanation:

- `SELECT INTO` is used to **fetch data into variables**.
- It **must return exactly one row**, otherwise, an error occurs (`NO_DATA_FOUND` or `TOO_MANY_ROWS`).
- **3. Sorting Faculty Names in Ascending Order**
- To sort `FacultyName` in **ascending order**, we use the **ORDER BY** clause.
- ✅**SQL Query to Sort Faculty Names:**

```
SELECT FacultyName
FROM Faculty
ORDER BY FacultyName ASC;
```

Explanation:

- `ORDER BY FacultyName ASC` **sorts names in A-Z order**.
- **`ASC` is default** (so `ORDER BY FacultyName` works the same).

By Maha Rana

Long Questions:

1. PL/SQL Code to Check Odd or Even Number Using IF-ELSE

✅**Scenario:** A table is given, and we check **if a total number is ODD or EVEN** using `IF-ELSE`.

✅PL/SQL Code:

```
DECLARE
   total_number NUMBER := 10; -- Change as needed
BEGIN
   IF MOD(total_number, 2) = 0 THEN
      DBMS_OUTPUT.PUT_LINE(total_number || ' is Even');
   ELSE
      DBMS_OUTPUT.PUT_LINE(total_number || ' is Odd');
   END IF;
END;
```

Explanation:

- `MOD(total_number, 2)` returns **0** for **even numbers** and **1** for **odd numbers**.
- Uses `IF-ELSE` to **check condition and print the result**.

✅Example Output for `total_number = 10`:

```
10 is Even
```

✅Example Output for `total_number = 15`:

```
15 is Odd
```

2. PL/SQL Basic Structure

PL/SQL programs follow a **block structure**.

✅Basic PL/SQL Block Syntax:

```
DECLARE
   -- Variable declarations
BEGIN
   -- Executable statements
EXCEPTION
   -- Exception handling (optional)
END;
```

✅Example:

```
DECLARE
   message VARCHAR2(50) := 'Hello, PL/SQL!';
BEGIN
   DBMS_OUTPUT.PUT_LINE(message);
END;
```

Explanation:

- **DECLARE** (Optional) → Declare variables.
- **BEGIN** → Contains the main executable code.
- **EXCEPTION** (Optional) → Handles errors.
- **END** → Ends the block.

By Maha Rana

----------THE END----------

May Allah grant you success, ease your efforts, and bless you with wisdom and perseverance. Keep your faith strong, for with prayer, every challenge becomes an opportunity. I hope this file helps you a lot, and please remember me in your prayers.

Best wishes for Mids!
Maha ☺