SQL Cursors Overview

Certainly! Below are several more examples of using cursors in Oracle PL/SQL, covering different use cases such as implicit cursors, explicit cursors, cursors with parameters, and cursor with FOR loops.

1. Basic Implicit Cursor (SQL statement directly inside PL/SQL block)

In Oracle, every DML operation (SELECT, INSERT, UPDATE, DELETE) implicitly uses a cursor. If you're not using an explicit cursor, Oracle automatically manages a cursor for you.

```sql
DECLARE
    v_emp_name employees.first_name%TYPE;
BEGIN
    -- Implicit cursor for SELECT INTO
    SELECT first_name INTO v_emp_name
    FROM employees
    WHERE employee_id = 100;

    -- Display the result
    DBMS_OUTPUT.PUT_LINE('Employee Name: ' || v_emp_name);
END;
```

2. Explicit Cursor Example

An explicit cursor gives you more control over the execution and management of SQL queries. Below is an example where we manually declare and manage the cursor.

```sql
DECLARE
    -- Declare cursor
    CURSOR emp_cursor IS
        SELECT employee_id, first_name, salary
        FROM employees
        WHERE department_id = 10;

    -- Declare variables to store fetched data
    v_emp_id employees.employee_id%TYPE;
    v_first_name employees.first_name%TYPE;
    v_salary employees.salary%TYPE;
BEGIN
    -- Open the cursor
    OPEN emp_cursor;

    -- Loop through each record fetched by the cursor
    LOOP
        FETCH emp_cursor INTO v_emp_id, v_first_name, v_salary;

        -- Exit the loop if no more rows are available
        EXIT WHEN emp_cursor%NOTFOUND;

        -- Display employee details
        DBMS_OUTPUT.PUT_LINE('ID: ' || v_emp_id || ', Name: ' || v_first_name || ', 
Salary: ' || v_salary);
    END LOOP;

    -- Close the cursor
    CLOSE emp_cursor;
END;
```

3. Cursor with Parameters

You can define a cursor that takes parameters, allowing you to dynamically change the query execution based on input.

```sql
DECLARE
    -- Declare cursor with parameters
    CURSOR emp_cursor(p_dept_id IN NUMBER) IS
        SELECT employee_id, first_name
        FROM employees
        WHERE department_id = p_dept_id;

    -- Declare variables to store fetched data
    v_emp_id employees.employee_id%TYPE;
    v_first_name employees.first_name%TYPE;
BEGIN
    -- Open cursor with a parameter value
    OPEN emp_cursor(10);  -- Fetching employees from department 10
    -- Loop through each record fetched by the cursor
    LOOP
        FETCH emp_cursor INTO v_emp_id, v_first_name;

        -- Exit the loop if no more rows are available
        EXIT WHEN emp_cursor%NOTFOUND;

        -- Display employee details
        DBMS_OUTPUT.PUT_LINE('Employee ID: ' || v_emp_id || ', Name: ' ||
v_first_name);
    END LOOP;

    -- Close the cursor
    CLOSE emp_cursor;
END;
```

4. Cursor FOR Loop (Simplified Version)

You can use the `FOR` loop to automatically handle cursor opening, fetching, and closing.

```sql
BEGIN
    -- Cursor FOR loop automatically handles opening, fetching, and closing the
cursor
    FOR emp_record IN (SELECT employee_id, first_name, salary FROM employees WHERE
department_id = 20) LOOP
        -- Display employee details
        DBMS_OUTPUT.PUT_LINE('ID: ' || emp_record.employee_id || ', Name: ' ||
emp_record.first_name || ', Salary: ' || emp_record.salary);
    END LOOP;
END;
```

5. Cursor with Bulk Collect (Fetching Multiple Rows)

In some cases, you may want to fetch multiple rows into a collection (e.g., a `PL/SQL` table or an array) for better performance.

```sql
DECLARE
    -- Declare a collection to hold multiple rows
    TYPE emp_table IS TABLE OF employees%ROWTYPE;
    emp_data emp_table;

    -- Declare cursor to select employees from department 30
    CURSOR emp_cursor IS
        SELECT * FROM employees WHERE department_id = 30;
BEGIN
    -- Fetch all rows into the collection using BULK COLLECT
    OPEN emp_cursor;
    FETCH emp_cursor BULK COLLECT INTO emp_data;
    CLOSE emp_cursor;

    -- Loop through the collection and display the results
    FOR i IN 1 .. emp_data.COUNT LOOP
        DBMS_OUTPUT.PUT_LINE('ID: ' || emp_data(i).employee_id || ', Name: ' ||
emp_data(i).first_name);
    END LOOP;
END;
```

6. Cursor with UPDATE (Using Cursor for DML Operations)

You can use cursors to fetch data and then perform updates based on that data.

```sql
DECLARE
    -- Declare cursor to select employees from department 50
    CURSOR emp_cursor IS
        SELECT employee_id, salary FROM employees WHERE department_id = 50 FOR
UPDATE;

    v_emp_id employees.employee_id%TYPE;
    v_salary employees.salary%TYPE;
BEGIN
    -- Open cursor
    OPEN emp_cursor;

    -- Loop through each record and update the salary
    LOOP
        FETCH emp_cursor INTO v_emp_id, v_salary;
        EXIT WHEN emp_cursor%NOTFOUND;

        -- Update the salary (e.g., increase by 10%)
        UPDATE employees
```

```
            SET salary = v_salary * 1.10
            WHERE CURRENT OF emp_cursor;  -- Using CURRENT OF to update the current row

    END LOOP;

    -- Commit the changes
    COMMIT;

    -- Close the cursor
    CLOSE emp_cursor;
END;
```

7. Cursor with Multiple FETCH Statements (Handling Multiple Cursors)

Sometimes, you may want to handle multiple cursors in the same block.

```sql
DECLARE
    -- Declare first cursor to fetch department info
    CURSOR dept_cursor IS
        SELECT department_id, department_name FROM departments;

    -- Declare second cursor to fetch employees
    CURSOR emp_cursor IS
        SELECT employee_id, first_name FROM employees WHERE department_id = 10;

    v_dept_id departments.department_id%TYPE;
    v_dept_name departments.department_name%TYPE;
    v_emp_id employees.employee_id%TYPE;
    v_emp_name employees.first_name%TYPE;
BEGIN
    -- Process department cursor
    OPEN dept_cursor;
    FETCH dept_cursor INTO v_dept_id, v_dept_name;
    IF dept_cursor%FOUND THEN
        DBMS_OUTPUT.PUT_LINE('Department: ' || v_dept_name);
    END IF;

    -- Process employee cursor based on department 10
    OPEN emp_cursor;
    LOOP
        FETCH emp_cursor INTO v_emp_id, v_emp_name;
        EXIT WHEN emp_cursor%NOTFOUND;
        DBMS_OUTPUT.PUT_LINE('Employee: ' || v_emp_name || ', ID: ' || v_emp_id);
    END LOOP;

    -- Close both cursors
    CLOSE dept_cursor;
    CLOSE emp_cursor;
END;
```

Summary:

- **Implicit Cursors**: Automatically handled by Oracle for simple SQL operations (e.g., `SELECT INTO`).
- **Explicit Cursors**: Give you more control (e.g., `OPEN`, `FETCH`, `CLOSE`).
- **Cursor FOR Loop**: A more concise way to handle cursors without manually opening and closing them.
- **Cursor with Parameters**: Allows you to dynamically pass values to a cursor's query.
- **Bulk Collect**: Fetch multiple rows into a collection for improved performance.
- **Cursor with DML**: Use a cursor to fetch rows and perform DML operations on those rows (e.g., `UPDATE`, `DELETE`).

Let me know if you need more detailed explanations or further examples!