SQL Programming

PL/SQL

PL/SQL

- **Pros and cons of SQL**
- Very high-level, possible to optimize
- Not tuned to support general-purpose computation
- Strictly less expressive than general-purpose languages

- **Solutions**
- Augment SQL: Oracle's PL/SQL
- Use SQL together with a general-purpose programming language

Oracle PL/SQL

Basics

Rough form of a PL/SQL program:

```
DECLARE
BEGIN
END;
.
RUN;
```

DECLARE section is optional

. and RUN end the program and execute it

Basic Features

Local variable:

Use %TYPE to match its type to a column in the schema

Use := for assignment; = for comparison

Branch: IF (...) THEN ... ELSE ... END IF;

Loop: LOOP ... EXIT WHEN (...); ... END LOOP;

The usual data modification statements: INSERT, DELETE, UPDATE

Single-row SELECT: SELECT ... INTO ... FROM ...;

Oracle raises an exception if SELECT returns no rows or more than one row

```
CREATE TABLE Student (SID INTEGER PRIMARY
KEY,
name CHAR(30),
age INTEGER,
GPA FLOAT);

CREATE TABLE Course (
CID CHAR(10) PRIMARY KEY,
title VARCHAR(100) UNIQUE);

CREATE TABLE Take (SID INTEGER,
CID CHAR(10),
PRIMARY KEY(SID, CID));
```

```
DECLARE
thisSID Student.SID%TYPE;
thisGPA Student.GPA%TYPE;
BEGIN
thisSID := 100;
LOOP
 EXIT WHEN (thisSID > 800);
 SELECT GPA INTO thisGPA
 FROM Student
 WHERE SID = thisSID;
 IF (thisGPA < 4.0) THEN
  UPDATE Student SET GPA = 4.0
  WHERE SID = thisSID;
 END IF;
 thisSID := thisSID + 1;
END LOOP;
END;
```

SELECT … INTO

Get the last name for a specific employee ID (the primary key in the employees table)

```
DECLARE
  l_last_name  employees.last_name%TYPE;
BEGIN
  SELECT last_name
    INTO l_last_name
    FROM employees
   WHERE employee_id = 138;
  DBMS_OUTPUT.put_line (
     l_last_name);
END;
```

SELECT … INTO

Fetch an entire row from the employees table for a specific employee ID

```
DECLARE
   l_employee   employees%ROWTYPE;
BEGIN
   SELECT *
     INTO l_employee
     FROM employees
    WHERE employee_id = 138;
   DBMS_OUTPUT.put_line (
      l_employee.last_name);
END;
```

SELECT … INTO

Fetch columns from different tables

```
DECLARE
  l_last_name
     employees.last_name%TYPE;
  l_department_name
     departments.department_name%TYPE;
BEGIN
  SELECT last_name, department_name
    INTO l_last_name, l_department_name
    FROM employees e, departments d
   WHERE e.department_id=d.department_id
         AND e.employee_id=138;
  DBMS_OUTPUT.put_line (
     l_last_name ||
     ' in ' ||
     l_department_name);
END;
```

Cursors

Inside a PL/SQL program, the result of a SELECT must go somewhere:

If SELECT returns one row, it can go INTO a variable

What if SELECT returns multiple rows?

Cursor: a variable that runs through the result of a SELECT, row by row

Declare by: CURSOR *cursorName* IS;

Use inside a cursor loop:

Fetch one result row at a time: FETCH INTO;

Break the loop when there are no more rows to return:

EXIT WHEN %NOTFOUND;

OPEN/CLOSE before/after use

Cursors

If cursor is over a single table and has no aggregates or DISTINCT, we can also modify data through the cursor.

Follow the declaration by FOR UPDATE

Use WHERE CURRENT OF *cursorName* in DELETE or UPDATE

Note it is possible to declare a “row” type in Oracle: `%ROWTYPE`

```
DECLARE
thisStudent Student%ROWTYPE;
CURSOR CS145Student IS
SELECT * FROM Student WHERE SID IN
(SELECT SID FROM Take WHERE CID = 'CS145')
FOR UPDATE;
BEGIN
OPEN CS145Student;
LOOP
 FETCH CS145Student INTO thisStudent;
 EXIT WHEN (CS145Student%NOTFOUND);
 IF (thisStudent.GPA < 4.0) THEN
  UPDATE Student SET GPA = 4.0
  WHERE CURRENT OF CS145Student;
 END IF;
END LOOP;
CLOSE CS145Student;
END;
```

Stored Procedures

Creating a PL/SQL stored procedure:

```
CREATE PROCEDURE procedureName(argDeclarations) AS
BEGIN
END;
.
RUN;
```

The RUN above creates the procedure, but does not execute it
Running the procedure inside a PL/SQL program:

```
BEGIN
...
procedureName(args);
...
END;
.
RUN;
```

Stored Procedures

Example: a procedure to enroll students in CS145

```
CREATE PROCEDURE CS145Enroll (thisSID IN Take.SID%TYPE) AS
BEGIN
    INSERT INTO Take VALUES(thisSID, 'CS145');
END;
.
RUN;
```

Example: students 142 and 857 enroll in CS145

```
BEGIN
CS145Enroll(142);
CS145Enroll(857);
END;
.
RUN;
```

Subprogram Parameter Modes

- Parameter modes define the action of formal parameters. The three parameter modes are IN (the default), OUT, and IN OUT.

- Any parameter mode can be used with any subprogram. Avoid using the OUT and IN OUT modes with functions.

Subprogram Parameter Modes

- An IN parameter lets you pass a value to the subprogram being invoked. Inside the subprogram, an IN parameter acts like a constant. It cannot be assigned a value.

- An OUT parameter returns a value to the caller of a subprogram. Inside the subprogram, an OUT parameter acts like a variable. You can change its value and reference the value after assigning it.

- An IN OUT parameter passes an initial value to a subprogram and returns an updated value to the caller. It can be assigned a value and its value can be read.

Example – Using Out Mode

```
DECLARE
  emp_num       NUMBER(6) := 120;
  bonus         NUMBER(6) := 50;
  emp_last_name VARCHAR2(25);

  PROCEDURE raise_salary (emp_id IN NUMBER, amount IN NUMBER,
                          emp_name OUT VARCHAR2) IS
    BEGIN
      UPDATE employees SET salary =
        salary + amount WHERE employee_id = emp_id;
      SELECT last_name INTO emp_name
      FROM employees
      WHERE employee_id = emp_id;
  END raise_salary;

BEGIN
  raise_salary(emp_num, bonus, emp_last_name);
  DBMS_OUTPUT.PUT_LINE
    ('Salary was updated for: ' || emp_last_name);
END;
```

```
DECLARE
l_total        INTEGER := 10000;

CURSOR employee_id_cur IS
   SELECT employee_id FROM employees
   ORDER BY salary ASC;

l_employee_id   employee_id_cur%ROWTYPE;
BEGIN
OPEN employee_id_cur;
LOOP
    FETCH employee_id_cur INTO l_employee_id;
    EXIT WHEN employee_id_cur%NOTFOUND;

    assign_bonus (l_employee_id, l_total);
    EXIT WHEN l_total <= 0;
END LOOP;

CLOSE employees_cur;
END;
```

Class Exercises

- For employees working in ‘Research’ department, assign each employee to Project W (10 hours per week)

- Write a stored procedure that gives 10% increase to all employees in ‘Research’ department.

- Write a stored procedure that gives 10% increase to all employees who earn minimum in their respective departments.

References

- https://docs.oracle.com/cd/B28359_01/appdev.111/b28370/overview.htm#CJAHAGEF