```
================================================================================
<< Less 04 : Interacting with the Oracle Server >>
================================================================================
        중요 학습 내용:

        • PL/SQL을 통해서 SELECT 문을 실행.
        • PL/SQL을 통해서 DML문을 실행.

        • PL/SQL Cursor 개념: 메모리를 가리키는 pointer
          커서의 종류
                (1) Implicit cursor(SQL cursor)
                (2) Explicit cursor


        • Cursor 속성


• 본 문서의 실습은 별도의 언급이 없는 한, SQL*Developer를 이용하여
  hr 계정으로 데이터베이스에 원격하여 수행합니다.


<< SQL Statements in PL/SQL >>


        오라클 엔진에는 SQL문장을 처리하는 SQL Engine 부분과
        PL/SQL을 처리하는 PL/SQL 엔진부분이 분리되어 있습니다.

        하나의 PL/SQL 코드에 PL/SQL 문장요소들은 PL/SQL 엔진부분이
        처리하고 SQL문장요소들은 SQL Engine 부분이 처리합니다.

        즉, PL/SQL엔진과 SQL 엔진사이에 Switch가 발생됩니다.


        • Retrieve a row from the database by using the SELECT command.

        • Make changes to rows in the database by using DML commands.

        • Control a transaction with
          the COMMIT, ROLLBACK, or SAVEPOINT command.


        ================================================================
        (참고) PL/SQL 익명블록에서 DDL을 아래처럼 실행할 수는 없습니다.
        ================================================================
                BEGIN
                 CREATE TABLE My_emp_table AS SELECT * FROM employees;
                END;
                /

                ERROR at line 2:
                ORA-06550: line 2, column 1:
                PLS-00103: Encountered the symbol "CREATE" when
                expecting one of the following:
                begin case declare exit for goto if loop mod null pragma
                raise return select update while with <an identifier>
                <a double-quoted delimited-identifier> <a bind variable> <<
                close current delete fetch lock insert open rollback
                savepoint set sql execute commit forall merge pipe


        ================================================================
        (참고) DDL 문장을 PL/SQL 익명블록에서 사용하려면
               다음처럼 Dynamic SQL (EXECUTE IMMEDIATE 'DDL문')을
               사용해야 합니다(다음코스에서 학습합니다).
        ================================================================
                SQL> BEGIN
                        EXECUTE IMMEDIATE
                                'CREATE TABLE My_emp_table '||
```

```
                    'AS SELECT * FROM employees';
     END;
     /
PL/SQL procedure successfully completed.


SQL> BEGIN
          EXECUTE IMMEDIATE 'DROP TABLE My_emp_table';
     END;
     /
PL/SQL procedure successfully completed.
```

<< SELECT Statements in PL/SQL >>

[문법] BEGIN--END; [Executive section] 내에 다음처럼 기술.

```
SELECT select_list
          INTO {variable_name[, variable_name]...| record_name}
FROM   table
WHERE condition ;
```

==========

<< 작성 시 주의사항 >>

==========

- Every value retrieved must be stored in a variable using the INTO clause
- select_list의 컬럼수 및 데이터타입 순서와 into 절에 명시되는 변수들의 갯수 및 데이터타입 순서가 일치해야 함.
- 쿼리 결과는 반드시 하나의 행에서만 값이 반환되어야 합니다.
- 쿼리가 하나 이상의 행을 반환하거나 반환하는 행이 없으면 오류가 발생됩니다.

==========

(참고) 위의 오류들은
NO_DATA_FOUND and TOO_MANY_ROWS 예외을 이용해서
처리해주면 오류를 방지할 수 있습니다(Less 8)

여러행이 반환되는 쿼리의 경우에
explicit cursors 를 이용하여 처리할 수 있습니다
(Less 7)

==========

<< SELECT 문 사용 예제(4-8) >>

(예제 1)

```
SQL> SET SERVEROUTPUT ON

SQL> DECLARE
          fname VARCHAR2(25);
     BEGIN
          SELECT first_name INTO fname
          FROM employees
          WHERE employee_id=200;

          DBMS_OUTPUT.PUT_LINE(' First Name is : '||fname);
     END;
     /
First Name is : Jennifer
```

```
PL/SQL procedure successfully completed.

SQL>
```

```
=====================================================================
SQL> SET SERVEROUTPUT ON

SQL> DECLARE
        fname VARCHAR2(25);
     BEGIN
        SELECT first_name INTO fname
        FROM employees
        WHERE last_name='King'; -- 2개 rows 따라서 아래의 error 발생

        DBMS_OUTPUT.PUT_LINE(' First Name is : '||fname);
     END;
     /
ERROR at line 1:
ORA-01422: exact fetch returns more than requested number of rows
ORA-06512: at line 4
=====================================================================
(참고) 위의 에러와 관련된 Exception : TOO_MANY_ROWS
=====================================================================

(예제 2)

SQL> SET SERVEROUTPUT ON

SQL> DECLARE
        fname VARCHAR2(25);
        lname VARCHAR2(25) := 'Oracle';
     BEGIN
        SELECT first_name INTO fname
        FROM employees
        WHERE employee_id=400; -- rows 가 없음, 아래의 error 발생

        DBMS_OUTPUT.PUT_LINE(' First Name is : '||fname);
        DBMS_OUTPUT.PUT_LINE(' Last  Name is : '||lname);
        lname := Null ;
        DBMS_OUTPUT.PUT_LINE(' Last  Name is : '||lname);
     END;
     /
DECLARE
*
ERROR at line 1:
ORA-01403: no data found
ORA-06512: at line 5

=====================================================================
(참고) 위의 오류와 관련된 Exception : NO_DATA_FOUND
=====================================================================
<< 위의 문장이 Exception 처리가 되었을 때..>>
=====================================================================
SQL> SET SERVEROUTPUT ON

SQL> DECLARE
        fname VARCHAR2(25);
     BEGIN
        SELECT first_name INTO fname
        FROM employees
        WHERE employee_id=400; -- rows 가 없음, 아래의 error 발생

        DBMS_OUTPUT.PUT_LINE(' First Name is : '||fname);
     EXCEPTION
      WHEN NO_DATA_FOUND THEN
        DBMS_OUTPUT.PUT_LINE(
                '쿼리문이 반환하는 데이터가 없습니다');
```

```
    END;
    /
쿼리문이 반환하는 데이터가 없습니다


PL/SQL 프로시저가 성공적으로 완료되었습니다.

SQL>
```

```
================================================================================
<< Retrieving Data in PL/SQL >>
================================================================================
SQL> SET SERVEROUTPUT ON

SQL> DECLARE
       emp_hiredate hr.employees.hire_date%TYPE;
       emp_salary   hr.employees.salary%TYPE;
     BEGIN
       SELECT hire_date, salary INTO emp_hiredate, emp_salary
       FROM hr.employees
       WHERE employee_id = 100;

       /* 위의 emp_hiredate, emp_salary 변수의 내용을 */
       /* 출력시키는 코딩을 작성해서 채우세요.        */
     END;
     /


SQL> DECLARE
       sum_sal NUMBER(10,2);
       deptno  NUMBER NOT NULL := 60;
     BEGIN
       SELECT SUM(salary) -- group function
                          INTO sum_sal
       FROM employees2
       WHERE department_id = deptno;
       DBMS_OUTPUT.PUT_LINE ('The sum of salary is '|| sum_sal);
     END;
     /
The sum of salary is 28800

PL/SQL procedure successfully completed.

(주의) sum_sal := SUM(employees.salary); <-- 이렇게는 사용못합니다.



<< Naming Conventions >>


아래의 코드에서 SELECT 문장 다음에 사용된 모든 식별자는
데이터베이스 테이블의 컬럼이름과 동일합니다.
into 절의 식별자는 pl/sql 변수이므로 혼동할 가능성이 없으나
WHERE-절에서는 혼동될 수 있음.
================================================================================
• WHERE 절에 모호함을 피하기 위하여  Naming convention 을 지정하시오.
• Table의 column 이름을 Identifiers(특히, 변수의 이름)로 사용하지
  마시오.
  (이유) 테이블의 Column이름이 local variables의 이름보다
         우선 순위가 높아서 먼저 확인되기 때문입니다.
         따라서, 문장 에러가 발생될 수 있습니다.

• 테이블의 이름보다  local variables와 formal parameters의 이름이
  먼저 확인됩니다.
================================================================================
(예제 1)
```

```
SQL> DECLARE
        hire_date    hr.employees.hire_date%TYPE;
        sysdate      hire_date%TYPE;
        employee_id  hr.employees.employee_id%TYPE := 176;
     BEGIN
        SELECT hire_date, sysdate INTO hire_date, sysdate
        FROM hr.employees2
        WHERE employee_id = employee_id;
     END;
     /

DECLARE
*
ERROR at line 1:
ORA-01422: exact fetch returns more than requested number of rows
ORA-06512: at line 6

-- SELECT 문의 WHERE 절에서 employee_id가 모두 테이블의
-- 컬럼이름으로 처리되었기 때문에 오류가 발생됨.
=================================================================
SQL> DECLARE
        last_name VARCHAR2(25) := 'King';
     BEGIN
        DELETE FROM employees2
        WHERE last_name = last_name; <-- 모든행이 지워짐.
     end ;
     /

PL/SQL procedure successfully completed.


SQL> ROLLBACK ; -- 반드시 롤백해야만 합니다.
```

```
================================================================================
4-14 Inserting Data
================================================================================
SQL> BEGIN
        INSERT INTO hr.employees2 (
                employee_id, first_name, last_name, email,
                hire_date, job_id, salary)
        VALUES(employees_seq.NEXTVAL, 'Ruth', 'Cores', 'RCORES',
                sysdate, 'AD_ASST', 4000);
     END;
     /

(참고) SQL functions, USER , SYSDATE 함수 사용가능.
```

```
================================================================================
4-15 Updating Data
================================================================================
SQL> DECLARE
        sal_increase hr.employees.salary%TYPE := 800;
     BEGIN
        UPDATE hr.employees2
        SET salary = salary + sal_increase
        WHERE job_id = 'ST_CLERK1';
     END;
     /

행이 수정되지 않아도 오류가 발생되지 않음.
```

```
================================================================================
4-16 Deleting Data
```

```
SQL> DECLARE
        deptno employees.department_id%TYPE := 10;
     BEGIN
        DELETE FROM hr.employees2
        WHERE department_id = deptno;
     END;
     /
```

4-17 Merging Rows

(주의) merge는 결정문입니다.
즉, 하나의 merge문에서 target 테이블의 동일한 행을
여러번 갱신할 수 없읍니다.
또한 target 테이블에 대해 insert 및 update 객체 권한이 있어야 하며
source 테이블에 대해서는 select 권한이 있어야 합니다.

```
SQL> create table copy_emp2
     as select * from hr.employees
        where department_id = 10 ;


SQL> BEGIN
        MERGE INTO copy_emp2 c
                USING employees2 e
                ON (e.employee_id = c.employee_id)
        WHEN MATCHED THEN
        UPDATE SET
                c.first_name = e.first_name,
                c.last_name = e.last_name,
                c.email = e.email,
                c.phone_number = e.phone_number,
                c.hire_date = e.hire_date,
                c.job_id = e.job_id,
                c.salary = e.salary,
                c.commission_pct = e.commission_pct,
                c.manager_id = e.manager_id,
                c.department_id = e.department_id
        WHEN NOT MATCHED THEN
        INSERT VALUES(  e.employee_id, e.first_name,   e.last_name,
                        e.email,       e.phone_number, e.hire_date,
                        e.job_id,      e.salary,
                        e.commission_pct, e.manager_id,
                        e.department_id);
     END;
     /
```

<< Cursor >>

A cursor is a pointer to the private memory area
allocated by the Oracle server.

- implicit cursor (sql cursor)
  : Oracle server는 암시적 커서를 사용하여
    SQL문의 구문을 분석하고 실행합니다.

- explicit cursor (user-defined cursor)
  : 프로그래머가 명시적으로 선언합니다.

4-21 SQL Cursor Attributes for Implicit Cursors

```
<< SQL Cursor Attributes >>
=============================================================
SQL%FOUND        : 가장 최근의 SQL문이
                   하나 이상의 row를 returned 했을 때
                   TRUE로 평가되는 Boolean 속성
=============================================================
SQL%NOTFOUND     : 가장 최근의 SQL문이
                   하나의 row도 returned 하지 않았을 때
                   TRUE로 평가되는 Boolean 속성
=============================================================
SQL%ROWCOUNT     : 가장 최근의 SQL문에 의해 영향을 받은
                   row의 개수(정수값).
=============================================================
이것들을 사용하는 이유 중 하나:

  SQL 커서 속성을 사용하면 마지막으로 암시적 커서를
  사용했을 때 발생한 작업을 평가할 수 있으며,
  이 속성은 SQL문이 아니라 PL/SQL문에 사용됩니다.

=============================================================
PL/SQL 블록의 Executive 섹션에서
SQL%ROWCOUNT, SQL%FOUND, and SQL%NOTFOUND 속성을
사용하여 DML 문의 실행에 대한 정보를 모을 수 있습니다.
=============================================================
(주의)

  - DML 문이 기본 테이블의 행에 영향을 주지 않았을 때
    오류를 반환하지 않음.
-------------------------------------------------------------
  - SELECT 문에서 검색된 row가 없을 때에는
    exception을 반환.
=============================================================

SQL> set serverout on
     VARIABLE rows_deleted1 VARCHAR2(30)
     VARIABLE rows_deleted2 VARCHAR2(30)

SQL> DECLARE
        empno  hr.employees2.employee_id%TYPE   := 101;
        deptno hr.employees2.department_id%TYPE := 20 ;

     BEGIN
        DELETE FROM hr.employees2
        WHERE employee_id = empno;

        :rows_deleted1 := (SQL%ROWCOUNT || ' row deleted.');

        DELETE FROM hr.employees2
        where department_id = deptno;

        :rows_deleted2 := (SQL%ROWCOUNT || ' row deleted.');
    END;
    /

PL/SQL procedure successfully completed.

SQL> PRINT rows_deleted1 rows_deleted2

ROWS_DELETED1
-------------
1 row deleted.


ROWS_DELETED2
-------------
2 row deleted.
```

```
SQL> VARIABLE rows_deleted3 VARCHAR2(30)
     VARIABLE rows_deleted4 VARCHAR2(30)

SQL> DECLARE
      empno  hr.employees2.employee_id%TYPE   := 102;
      deptno hr.employees2.department_id%TYPE :=30 ;
     BEGIN
      DELETE FROM hr.employees2
      WHERE employee_id = empno;

      DELETE FROM hr.employees2
      where department_id = deptno;

     :rows_deleted3 := (SQL%ROWCOUNT || ' row deleted.');

     :rows_deleted4 := (SQL%ROWCOUNT || ' row deleted.');
    END;
    /

SQL> print rows_deleted1 rows_deleted2 rows_deleted3 rows_deleted4

ROWS_DELETED1
--------------------------------
1 row deleted.


ROWS_DELETED2
--------------------------------
2 row deleted.


ROWS_DELETED3
--------------------------------
6 row deleted.


ROWS_DELETED4
--------------------------------
6 row deleted.


SQL> rollback ;

Rollback complete.

SQL>
```