==========

<< Less 08 예외(EXCEPTION) 처리 >>

==========

중요 학습 내용:

- PL/SQL 프로그램에 예외처리 방법
- 예외 개요
- 미리 정의되지 않은 예외를 처리하는 방법
- 미리 정의된 예외를 처리하는 방법

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

<< 예외(EXCEPTION) 이란 ? >>

- 예외는 실행 중에 발생되는 오류(에러)를 처리하도록 정의된
[오류(에러)와 관련 PL/SQL 코드] 입니다.

<< 예외(EXCEPTION) 처리가 되지 않았을 경우의 코드 예제 >>

[오류의 예-1]

- 실행 시에 SELECT 문의 조건에 맞는 행이 없어서
lname 변수에 할당할 값이 없기 때문에 실행 중에 에러가 발생됨.

```
SQL> SET SERVEROUTPUT ON
SQL>

SQL> DECLARE
        lname VARCHAR2(15);
     BEGIN
        SELECT last_name INTO lname
        FROM hr.employees
        WHERE first_name='Johns';

        DBMS_OUTPUT.PUT_LINE ('Johns''s last name is : '||lname);
     END;
     /
DECLARE
*
ERROR at line 1:
ORA-01403: no data found
ORA-06512: at line 4


SQL>
```

[오류의 예-2]

- 실행 시에 SELECT 문의 조건에 맞는 행이 둘 이상 존재하여
스칼라 타입 변수인 lname 변수로 이를 처리할 수 없기 때문에
에러가 발생됨.

```
SQL> SET SERVEROUTPUT ON
```

```
SQL>

SQL> DECLARE
        lname VARCHAR2(15);
     BEGIN
        SELECT last_name INTO lname
        FROM hr.employees
        WHERE  first_name='John';

        DBMS_OUTPUT.PUT_LINE ('John''s last name is : '||lname);
     END;
     /
DECLARE
*
ERROR at line 1:
ORA-01422: exact fetch returns more than requested number of rows
ORA-06512: at line 4


SQL>
```

- 위의 2가지 예제에서 작성된 익명블록들은
  코드 구문상에서는 잘못된 점은 없습니다.

- 코드가 실행될 때,
  오류가 발생하여 프로그램이 비정상적으로 종료되었습니다.

<< 예외(EXCEPTION) 처리를 코드에 명시했을 때 예제 >>

```
SQL> SET SERVEROUTPUT ON
SQL>

SQL> DECLARE
        lname VARCHAR2(15);
     BEGIN
        SELECT last_name INTO lname
        FROM hr.employees
        WHERE first_name='John';

        DBMS_OUTPUT.PUT_LINE ('John''s last name is : '||lname);
     EXCEPTION
        WHEN TOO_MANY_ROWS THEN
                DBMS_OUTPUT.PUT_LINE (
                 'Your SELECT statement '||
                  'retrieved multiple rows.'||CHR(10) ||
                  'Consider using a cursor.');
     END;
     /
Your SELECT statement retrieved multiple rows.
Consider using a cursor.

PL/SQL procedure successfully completed.  <-- 정상종료 메세지

SQL>
```

==> 예외에 대한 처리 로직을 구현한 경우에
PL/SQL 프로그램이 정상적으로 종료됩니다.

(참고) 키보드 문자코드 값

- Chr(13) : 캐리지 리턴(carriage return)
- Chr(10) : 라인피드(new line)

<< 예외의 3가지 유형 >>

(1) 에러(오류)에 대하여 미리 정의되지 않은 예외(사용자 정의 예외)

(2) 에러(오류)에 대하여 데이터베이스 서버에 의해 미리 정의된 예외

(3) 에러(오류)없이 사용하는 예외

• (1)과 (2)의 경우에는 발생된 오류에 대하여
구성된 예외가 암시적으로 발생(호출)됩니다.

<< 미리 정의되지 않은 예외를 처리하는 절차 >>

1. PL/SQL 선언(DECLARE)부분에 다음과 같이 선언합니다.

- 예외(EXCEPTION) 이름 선언.

(주의) 예외(EXCEPTION)의 이름은 30-Bytes 까지만 가능

- [PRAGMA EXCEPTION_INIT] 문으로 오라클 서버 오류 번호와
선언된 예외(EXCEPTION)의 이름을 결합.

[PRAGMA EXCEPTION_INIT] 문은 PL/SQL 블록을 실행할 때
PL/SQL 컴파일러가 블록 안의 모든 예외 이름을
연관된 오라클-서버의 오류번호로 해석하도록 지시합니다.

(참고) "PRAGMA 키워드"의 역할

PRAGMA 키워드 다음에 명시한 명령문(예, EXCEPTION_INIT)이
컴파일러 지시어임을 나타냅니다.

2. 실행(BEGIN-END;)부분에서 EXCEPTION 블록을 사용하여
발생된 예외에 해당하는 예외-처리 로직(EXCEPTION-HANDLER)을
구현합니다.

<< 미리 정의되지 않은 예외 구현 예제 >>

- HR.DEPARTMENTS 테이블에 한 행을 입력 시에
NOT NULL 제약조건이 명시된 department_name 컬럼이
NULL인 상태인 경우에 아래의 오류가 발생됩니다.

```
SQL> INSERT INTO hr.departments (department_id, department_name)
     VALUES (400, null) ;
VALUES (400, null)
            *
ERROR at line 2:
ORA-01400: cannot insert NULL
           into ("HR"."DEPARTMENTS"."DEPARTMENT_NAME")

SQL>
```

[예외 구현 예제-1]

- 위의 INSERT 구문을 PL/SQL 프로그램을 통해서 실행시킬 때
  발생되는 ORA-01400 에러에 대한 예외처리를 구현해 봅니다.

```
SQL> SET SERVEROUTPUT ON
SQL>

SQL> DECLARE

        -- 1. 예외(EXCEPTION) 이름 선언.

        insert_excep EXCEPTION ;

        -- 2. 예외(EXCEPTION)와 오류번호를 결합

        PRAGMA EXCEPTION_INIT (insert_excep, -01400);

     BEGIN
        INSERT INTO hr.departments(department_id, department_name)
        VALUES (280, NULL);


     -- 3. 예외-처리 로직( EXCEPTION-HANDLER)

     EXCEPTION
        WHEN insert_excep
             THEN
                  DBMS_OUTPUT.PUT_LINE
                    ('['||SQLCODE||'] :'||' INSERT OPERATION FAILED');
                  DBMS_OUTPUT.PUT_LINE(SQLERRM);
     END;
     /
[-1400] : INSERT OPERATION FAILED
ORA-01400: cannot insert NULL into ("HR"."DEPARTMENTS"."DEPARTMENT_NAME")

PL/SQL procedure successfully completed.

SQL>
```

[예외 구현 예제-2]

- FOREIGN KEY 제약조건으로 참조되는 데이터를 삭제하려고 할 때
  ORA-02292 오류가 발생됩니다.

```
  ERROR at line 1:
  ORA-02292: integrity constraint (HR.EMP_DEPT_FK) violated
             - child record found
```

- 위의 오라클-서버 오류번호 -2292(무결성 제약조건 위반)을 트랩

```
SQL> SET SERVEROUT ON
SQL>
SQL> SET VERIFY OFF
SQL>

SQL> DECLARE
        v_p_deptno      number(4) :=&p_deptno;

        e_emps_remaining EXCEPTION ;

        PRAGMA EXCEPTION_INIT (e_emps_remaining, -2292) ;
```

```
     BEGIN
        DELETE FROM departments
        WHERE  department_id = v_p_deptno ;
     -- COMMIT ;
     EXCEPTION
        WHEN e_emps_remaining
           THEN
                DBMS_OUTPUT.PUT_LINE
                (' '||CHR(10) ||'Cannot remove dept '||
                 TO_CHAR(v_p_deptno) ||
                 '. Employees exist. '||CHR(10)||CHR(13)||' ');

                DBMS_OUTPUT.PUT_LINE(SQLERRM);
     END ;
     /
Enter value for p_deptno: 10

Cannot remove dept 10. Employees exist.

ORA-02292: integrity constraint (HR.EMP_DEPT_FK) violated - child record found

PL/SQL procedure successfully completed.

SQL>

--SQL*Developer에서 위의 코드를 다시 실행하여, 치환변수에 120 값을
--지정합니다.

--SQL*Plus에서는 / , 엔터키를 쳐서 문장을 실행한 후, 값입력을
--요구하는 메시지가 표시될 때, 120 입력합니다.


SQL> /
Enter value for p_deptno: 120  <-- 120을 입력합니다.

PL/SQL procedure successfully completed.

SQL>

        ==> HR.EMPLOYEES 테이블에 HR.DEPARTMENTS.DEPARTEMNT_ID를
            참조하는 행이 입력된 것이 없기 때문에
            PL/SQL 익명블록의 내용이 실행되어
            HR.DEPARTMENTS.DEPARTEMNT_ID가 120인 행이 삭제됩니다.


SQL> SELECT *
     FROM hr.departments
     WHERE department_id = 120 ;

no rows selected

SQL>

        ==> 행이 삭제되었으므로 표시되는 내용이 없습니다.



SQL> ROLLBACK ;

Rollback complete.

SQL>

        ==> 행을 삭제 전의 상태로 되돌리기 위하여
            트랜잭션을 ROLLBACK 시킵니다.
```

<< 오류가 발생되었을 때, 해당 오류에 대하여 예외(EXCEPTION)가 처리되는 과정 >>

- 발생된 오류에 대한 예외(EXCEPTION) 발생 시,
  예외-처리 로직(EXCEPTION-HANDLER)으로 트랩(Trapping)됩니다.

  - 블록의 실행부분에서 발생된 오류 때문에 예외가 발생하면
    블록의 EXCEPTION 섹션에 있는
    해당 예외-처리 로직(EXCEPTION-HANDLER)으로 전달됩니다.

    PL/SQL의 해당 예외-처리 로직(EXCEPTION-HANDLER)이
    예외를 성공적으로 처리한 경우에는,
    예외가 상위 블록 또는 환경으로 전달되지 않고
    PL/SQL 블록이 성공적으로 종료됩니다.

- 발생된 오류에 대한 예외(EXCEPTION) 발생 시,
  호출 환경으로 전달(Propagation).

  - 블록의 실행부분에서 예외가 발생했는데
    해당하는 예외-처리 로직(EXCEPTION-HANDLER)이
    블록의 EXCEPTION 섹션에 없으면,
    PL/SQL 블록은 실패하고
    예외는 호출환경(calling environment)으로 전달됩니다.

    호출환경은 PL/SQL 프로그램을 호출한 PL/SQL 블록 또는
    SQL*Plus 같은 프로그램 일 수 있습니다.

<< 예외의 3가지 유형 >>

(1) 에러(오류)에 대하여 미리 정의되지 않은 예외(사용자 정의 예외)

(2) 에러(오류)에 대하여 데이터베이스 서버에 의해 미리 정의된 예외

(3) 에러(오류)없이 사용하는 예외

- (1)과 (2)의 경우에는 발생된 오류에 대하여
  구성된 예외가 암시적으로 발생(호출)됩니다.

  - 오라클 서버가 예외를 암시적으로 발생시켜며,
    실행부분의 EXCEPTION-섹션에 예외 처리 로직을 구현해 놓았으면,
    해당 예외가 예외 처리 로직에 자동으로 전달되어 처리됩니다.

    - 오라클 서버에 미리 정의되지 않은 예외

      - 선언부에서 예외의 이름을 선언해야함.

      - 선언된 예외와 오류-번호를 연결.

    - 오라클 서버에 미리 정의된 예외

      - 선언부에서 예외 관련 내용을 선언할 필요가 없음.

- (3)의 경우에는, 예외가 개발자에 의해 명시적으로 발생됩니다.
  이를 "사용자 정의 예외(User-Defined Exception)"라고 합니다.

  개발자가 정한 조건에 만족하지 않을 경우 발생시키는 예외입니다.

  - 선언부에서 예외의 이름만 선언합니다.

- 실행부에서 [RAISE 예외이름]문을 사용하여
  예외를 발생시킵니다.

- 실행부분의 EXCEPTION-섹션에 예외 처리 로직을 구현하여
  해당 예외가 예외 처리 로직에 전달되어 처리됩니다.

<< 실행부분의 EXCEPTION-섹션에 예외 처리 로직(EXCEPTION-HANDLER) 구현문법 >>

[기본 문법]

```
EXCEPTION
        WHEN exception1 [OR exception2 ...] THEN
                statement1;
                statement2;
                ...
      [ WHEN exception3 [OR exception4 ...] THEN
                statement1;
                statement2;
                ... ]
      [ WHEN OTHERS THEN
                statement1;
                statement2;
                ... ]
```

- 실행부(BEGIN-END;)내의 마지막에 EXCEPTION 키워드로
  예외처리부를 시작합니다.
- 여러 개의 처리기를 명시할 수 있지만,
  예외는 그 중에 하나에서만 트랩됩니다.
- WHEN OTHERS 처리기는 맨 마지막에만 기술합니다.

<< 예제-1 >>

```
SQL> SET SERVEROUTPUT ON
SQL>

SQL> DECLARE
        lname VARCHAR2(15);
     BEGIN
        SELECT last_name INTO lname
        FROM hr.employees
        WHERE first_name='John';
        DBMS_OUTPUT.PUT_LINE ('John''s last name is : '||lname);
     EXCEPTION
        WHEN OTHERS THEN
             null;
             DBMS_OUTPUT.PUT_LINE(SQLERRM);
     END;
     /
ORA-01422: exact fetch returns more than requested number of rows

PL/SQL procedure successfully completed.

SQL>
```

<< 오라클-서버에 미리 정의된 예외(Predefined Exceptions) >>

(참고) 오라클-서버에 미리 정의된 예외는

STANDARD 패키지에 의하여 정의됩니다.

예외 이름 [예러코드]	관련 에러내용
ACCESS_INTO_NULL [ORA-06530]	: 초기화 되지 않은 객체의 속성에 값을 할당하려고 할 때.
CASE_NOT_FOUND [ORA-06592]	: CASE문의 WHEN 절에서 아무것도 선택되지 않고 ELSE 절이 없음.
COLLECTION_IS_NULL [ORA-06531]	: 초기화되지 않은 중첩테이블 또는 가별배열에 EXISTS 이외의 Collection methods를 적용하려고 할때
CURSOR_ALREADY_OPEN [ORA-06511]	: 이미 열려 있는 커서를 열려고 할때
DUP_VAL_ON_INDEX [ORA-00001]	: UNIQUE 제약을 갖는 컬럼에 중복되는 데이터가 INSERT될때
INVALID_CURSOR [ORA-01001]	: 잘못된 커서 연산
INVALID_NUMBER [ORA-01722]	: 문자열을 숫자로 변환못함
LOGIN_DENIED [ORA-01017]	: 오라클에 유효하지 않은 사용자 이름 또는 암호로 로그인했을 때
NO_DATA_FOUND [ORA-01403]	: 단일 행 SELECT문이 아무런 데이터 행을 반환하지 못할때
NOT_LOGGED_ON [ORA-01012]	: PL/SQL 프로그램이 오라클에 연결하지 않은 상태에서 db 호출을 실행할 때
PROGRAM_ERROR [ORA-06501]	: PL/SQL에 내부 문제가 있음
ROWTYPE_MISMATCH [ORA-06504]	: 할당에 사용된 PL/SQL 커서변수 및 Host커서 변수에 호환되지 않는 반환 유형이 있을 때
STORAGE_ERROR [ORA-06500]	: PL/SQL에 메모리가 부족 하거나 메모리가 손상되었을 때
SUBSCRIPT_BEYOND_COUNT [ORA-06533]	: Collection에 있는 요소의 수보다 큰 인덱스 번호를 사용하여 중첩테이블 또는 가변배열 요소를 참조할 때
SUBSCRIPT_OUTSIDE_LIMIT [ORA-06532]	: 범위를 벗어난 인덱스 번호 (예: -1)를 사용하여 중첩테이블 또는 가변배열 요소를 참조할때
SYS_INVALID_ROWID [ORA-01410]	: 문자열이 유효한 ROWID를 나타내지 않아서 문자열을 범용 ROWID로 변환하지 못했을 때
TIMEOUT_ON_RESOURCE [ORA-00051]	: 오라클이 자원을 기다리는 동안 시간 초과가 발생함
TOO_MANY_ROWS [ORA-01422]	: 단일 행 select에서 하나 이상의 행을 반환할 때
VALUE_ERROR [ORA-06502]	: 산술, 변환, 절단 또는 크기 제약조건 오류가 발생했을 때
ZERO_DIVIDE [ORA-01476]	: 0으로 나눌때

(참고) 가장 일반적인 NO_DATA_FOUND 및 TOO_MANY_ROWS 예외는
예외 처리 로직을 구현하는 것이 좋습니다.

<< 미리 정의된 예외 사용 예제 >>

```
SQL> SET SERVEROUTPUT ON
SQL>

SQL> DECLARE
        lname VARCHAR2(15);
     BEGIN
        SELECT last_name INTO lname
        FROM employees
        WHERE first_name='John';
        DBMS_OUTPUT.PUT_LINE ('John''s last name is : '||lname);
     EXCEPTION
        WHEN TOO_MANY_ROWS THEN
                DBMS_OUTPUT.PUT_LINE (
                 ' Your select statement retrieved multiple rows.'
                ||CHR(10)||'Consider using a cursor.');
     END;
     /
Your select statement retrieved multiple rows.
Consider using a cursor.

PL/SQL procedure successfully completed.

SQL>
```

<< SQLCODE, SQLERRM 함수 >>

- WHEN OTHERS문으로 트랩(Trap)되는 오류들의
  실제 오류 코드와 메시지를 볼 때 사용한다.

- SQLCODE : 오류코드의 숫자 값을 반환.

함수반환값	의 미
0	오류 없이 성공적으로 종료
1	사용자가 정의한 오류 번호
+100	NO_DATA_FOUND 예외의 반환값
음수	위에 것을 제외한 오라클 서버 오류 번호

- SQLERRM : 오류번호와 연관된 메시지를 반환.

<< SQLCODE, SQLERRM 함수를 활용하는 방법 >>

1> SQLCODE, SQLERRM 함수가 반환하는
   프로그램 에러정보를 저장할 테이블 생성

```
SQL> CREATE TABLE hr.errors_log_tbs
        (e_user         VARCHAR2(30)
        ,e_date         DATE DEFAULT SYSDATE
```

```
        ,error_code    NUMBER(7)
        ,error_message VARCHAR2(255)
        ) ;

Table created.

SQL>
```

2> SQLCODE, SQLERRM 함수가 포함된 Exception 처리기를 포함하여
   PL/SQL 코드 작성 및 실행.

```
        -- HR.DEPARTMENTS 테이블의 DEPARTMENT_ID 컬럼은
           HR.EMPLOYEES 테이블의 DEPARTMENT_ID 컬럼에 정의된
           FOREIGN KEY 제약조건으로 DEPARTMENT_ID가 (10~ 110) 까지는
           참조되고 있습니다.



SQL> DECLARE
        v_error_code    NUMBER;
        v_error_message VARCHAR2(255);

     BEGIN
        DELETE FROM hr.departments
        WHERE  department_id = &p_dept ;
     EXCEPTION
            WHEN OTHERS THEN
               ROLLBACK;
               v_error_code := SQLCODE ;
               v_error_message := SQLERRM ;
               INSERT INTO hr.errors_log_tbs
                  (e_user, e_date, error_code,error_message)
               VALUES(USER,SYSDATE,v_error_code,v_error_message);
               commit;
     END;
     /
Enter value for p_dept: 10  <-- 10 입력 후 [Enter]

PL/SQL procedure successfully completed.

SQL>
```

3> PL/SQL 프로그램 에러정보를 저장한 테이블(예, errors_log_tbs)에
   조회하여 발생된 에러를 확인.

```
SQL> col E_USER format a5
     col E_DATE format a11
     col ERROR_CODE format 999999
     col ERROR_MESSAGE format a40
     set linesize 80

     SELECT *
     FROM hr.errors_log_tbs ;

E_USE E_DATE      ERROR_CODE ERROR_MESSAGE
----- ----------- ---------- ----------------------------------------
HR    27-JAN-14        -2292 ORA-02292: integrity constraint (HR.EMP_
                             DEPT_FK) violated - child record found


SQL>
```

<< 예외가 개발자에 의해 명시적으로 발생되는 경우. >>

==> 사용자 정의 예외(User-Defined Exceptions)라고 합니다.

[코드 절차]

(1) 선언부분에 예외의 이름을 선언합니다.

(2) 실행 부분의 적절한 위치에 RAISE문을 사용하여
명시적으로 예외를 발생시킵니다.

(3) 실행 부분의 EXCEPTION-섹션에
발생시킨 예외를 처리하는 로직을 구현합니다.

<< 사용자 정의 예외 사용 실습 >>

- 아래의 UPDATE 문장은 WHERE 절을 만족하는 행이
HR.DEPARTMENTS 테이블에 없기 때문에 UPDATE한 행이 없습니다.
원래 오라클-서버에서는 이와 같은 상황이 에러는 아닙니다.

```
SQL> UPDATE hr.departments
     SET department_name='Oracle'
     WHERE department_id = 300 ;

0 rows updated.

SQL>
```

- 위의 경우처럼 실제로 오류는 아니지만,
프로그램 로직 상에서는 오류로 개발자가 간주하여 처리하고 할 때
개발자는 User-Defined Exceptions을 정의하여 처리할 수 있습니다.

[User-Defined Exceptions 예제-1]

- 300 번 부서가 없습니다.

```
SQL> SET VERIFY OFF

--Developer에서 아래코드를 실행하고,
--치환변수 값 입력을 요구할 때,
--name 치환변수에는 Oracle을,
--deptno 치환변수에는 300 입력하세요.

SQL> DECLARE
        invalid_department EXCEPTION;
        v_name             VARCHAR2(20) :='&name';
        v_deptno           NUMBER       := &deptno;
     BEGIN
        UPDATE hr.departments
        SET department_name = v_name
        WHERE department_id = v_deptno;   -- 300 번 부서가 없음

        IF SQL%NOTFOUND
           THEN RAISE invalid_department; -- 사용자 정의 예외를 직접 호출.
        END IF;

        --COMMIT;

     EXCEPTION
        WHEN invalid_department THEN
```

```
                DBMS_OUTPUT.PUT_LINE('No such department id.');
     END;
     /
No such department id.

PL/SQL procedure successfully completed.

SQL>


--Developer에서 위의코드를 다시 실행하고,
--치환변수 값 입력을 요구할 때,
--name 치환변수에는 DB-2을,
--deptno 치환변수에는 120 입력하세요.

SQL> DECLARE
        invalid_department EXCEPTION;
        v_name             VARCHAR2(20) :='&name';
        v_deptno           NUMBER       := &deptno;
     BEGIN
        UPDATE hr.departments
        SET department_name = v_name
        WHERE department_id = v_deptno;

        IF SQL%NOTFOUND
           THEN RAISE invalid_department;
        END IF;

        --COMMIT;

     EXCEPTION
        WHEN invalid_department THEN
                DBMS_OUTPUT.PUT_LINE('No such department id.');
     END;
     /

PL/SQL procedure successfully completed.

SQL>


SQL> SELECT *
     FROM hr.departments
     WHERE department_id=120 ;

DEPARTMENT_ID DEPARTMENT_NAME                MANAGER_ID LOCATION_ID
------------- ------------------------------ ---------- -----------
          120 DB-2                                             1700

SQL>


SQL> ROLLBACK ;

Rollback complete.

SQL>



[ User-Defined Exceptions 예제-2 ]


SQL> DEFINE p_department_desc = 'Information Technology'
SQL>

SQL> DEFINE p_department_number = 300
```

```
SQL>


SQL> DECLARE
        e_invalid_department EXCEPTION;
     BEGIN
        UPDATE hr.departments
        SET department_name = '&p_department_desc'
        WHERE   department_id = &p_department_number ;

        IF SQL%NOTFOUND
           THEN RAISE e_invalid_department ;
        END IF ;

--      COMMIT ;
     EXCEPTION
                WHEN e_invalid_department
                THEN
                  DBMS_OUTPUT.PUT_LINE('No such department id. ');
     END;
     /
No such department id.

PL/SQL procedure successfully completed.

SQL>
```

<< PL/SQL 프로그램이 호출되는 환경(Calling Environments) >>

• SQL*Plus 또는 SQL*Developer	: 오류 번호와 메시지를 화면에 표시합니다.
• Procedure Builder	: 오류 번호와 메시지를 화면에 표시합니다.
• Oracle Developer Forms	: ERROR_CODE 및 ERROR_TEXT 패키지 함수를 통해 트리거에서 오류번호와 메시지를 액세스 합니다.
• 선행 컴파일러 응용 프로그램	: SQLCA 데이터 구조를 통해 예외 번호를 액세스 합니다.
• 상위 PL/SQL 블록	: 상위 블록의 예외처리 루틴에서 예외를 트랩합니다.

<< Subblock에서 Outerblock으로 예외전달(Exception-Propagating) >>

- 서브블록에서 예외를 처리하거나 상위 블록에 전달할 수 있음.
- Subblock에 나머지 실행부분은 처리되지 않음.
- 호출된 블록내에 Exception handler를 구현할 수도 있고 또는 외부 block으로 전달할수도 있습니다.
- 서브 블록에서 예외를 처리하여 정상적으로 종료하면 서브블록 END 문 바로 다음의 상위 블록으로 제어가 넘어갑니다.
- 서브 블록에서 예외가 처리될 수 없으면, 예외는 처리기를 찾을 때까지 다음 상위블록으로 전달되고, 여기서도 처리되지 못하면 호스트 환경으로 넘어갑니다.
- 예외를 상위 블록에 전달하면 해당 서브 블록에 남아있던 실행 가능한 작업은 무시 됩니다.

• 이런 방식으로 예외가 처리되므로
  상위 블록에는 보다 범용적인 예외처리를 지정할 수 있고
  자체 블록에는 고유한 오류 처리를 필요로 하는 명령문을
  포함시킬 수 있습니다.

```
==========================================================
DECLARE
        . . .
        no_rows exception;
        integrity exception;
        PRAGMA EXCEPTION_INIT (integrity, -2292);
BEGIN
        FOR c_record IN emp_cursor
        LOOP
                BEGIN
                        SELECT ...;
                        UPDATE ...;

                        IF SQL%NOTFOUND THEN RAISE no_rows;
                        END IF;
                EXCEPTION
                WHEN no_rows
                        THEN ...
                END;
        END LOOP;
        ......;
        ......;

EXCEPTION
        WHEN integrity THEN ...
        WHEN no_rows THEN ...
END;
/
```

<< RAISE_APPLICATION_ERROR 프로시저 >>

[기본 문법]

```
raise_application_error (error_number,message[, {TRUE | FALSE}]);
```

• TRUE|FALSE : TRUE이면 이전 오류스택에 해당 오류가 추가되고
               FALSE(default)면 이전의 모든 오류를 해당오류로 대체.

• 실행 부분과 EXCEPTION-섹션에서 사용 가능하며
  다른 오라클-서버 오류와 동일한 방식으로
  사용자에게 오류조건(오류번호와 오류메세지)을 반환합니다.

• 이 프로시저를 사용하여 내장 서브 프로그램에서
  오류코드 -20000부터 -20999의 범위 내에서
  사용자가 정의한 오류 메세지를 발생시킬 수 있음.

• 응용 프로그램 오류를 보고할 수 있고 처리되지 않은
  예외가 반환되는 것을 방지할 수 있음.

<< RAISE_APPLICATION_ERROR 사용 예 >>

--Developer에서 아래의 코드를 실행하고,

```
--치환변수 값 입력을 요구할 때,
--p_department_name 치환변수에는 ORACLE을,
--p_department_id 치환변수에는 100 입력하세요.


SQL> DECLARE
        v_department_name  VARCHAR2(30) :='&p_department_name' ;
        v_department_id    NUMBER :=&p_department_id ;
        v_d_name  VARCHAR2(30) ;
        lname    VARCHAR2(30) ;

     BEGIN
        UPDATE hr.departments
        SET department_name = v_department_name
        WHERE   department_id = v_department_id ;

        SELECT department_name INTO v_d_name
        FROM hr.departments
        WHERE department_id=v_department_id ;

        DBMS_OUTPUT.PUT_LINE('AFTER UPDATE, DEPARTMENT_NAME: '||v_d_name);

        -- UPDATE는 정상 수행되어 1행이 변경됨.

        IF SQL%NOTFOUND THEN
           RAISE_APPLICATION_ERROR(-20202, 'No such department id.');
        END IF ;

        -- SQL%NOTFOUND는 FALSE를 반환하므로 IF 문의 로직이 실행되지 않음

     -- COMMIT ;

        SELECT last_name INTO lname
        FROM hr.employees
        WHERE  first_name='Johns';

        /* SELECT 문 실행 시에 조건을 만족하는 행이 없어서
           lname 변수에 할당할 값이 없습니다.

           오류가 발생됩니다.
           이 오류는 사전 정의된 NO_DATA_FOUND 예외로 처리되어
           아래의  EXCEPTION 섹션에 명시된 NO_DATA_FOUND 예외에 대한
           예외 처리 로직으로 전달됩니다.
        */

     EXCEPTION
        WHEN NO_DATA_FOUND
        THEN
                RAISE_APPLICATION_ERROR
                (-20201,'There is not Such Employee name .');

                -- 강제로 오류를 발생시켰습니다.
                -- 실행한 PL/SQL 익명블록 프로그램이
                -- 강제로 비정상 종료가 되었습니다.
                -- 이 때문에 앞에서 실행한 UPDATE 문장은
                -- ROLLBACK되고 트랜잭션이 종료됩니다.
     END;
     /
Enter value for p_department_name: ORACLE <-- ORACLE 입력 후, [Enter]
Enter value for p_department_id: 100      <-- 100 입력 후, [Enter]
AFTER UPDATE, DEPARTMENT_NAME: ORACLE
DECLARE
*
ERROR at line 1:
ORA-20201: There is not Such Employee name .
ORA-06512: at line 44
```

```
SQL>
```

(참고)

```
SQL> SELECT department_name
     FROM hr.departments
     WHERE department_id = 100 ;

DEPARTMENT_NAME
------------------------------
Finance

SQL>
```

==> 사용자가 직접 발생시킨 오류 때문에
UPDATE문장이 롤백되어 변경전 값이 표시됩니다.

위의 예제에서 UPDATE 문장을 정상적으로 처리되었지만
그 이후의 SELECT 문장처리 시에 발생된
EXCEPTION을 EXCEPTION-HANDLER에서
강제로 에러를 발생시켜 끝났기 때문에
전체 실행 단위에서는 오류로 완료되었으므로
프로그램 내의 트랜잭션은 ROLLBACK됩니다.

만약 아래처럼 SELECT 문에 COMMIT의 주석을 제거하면
UPDATE는 정상적으로 처리되었을 것입니다.

```
SQL> DECLARE
        lname   varchar2(30) ;
     BEGIN
        UPDATE  departments
        SET     department_name = '&p_department_name'
        WHERE   department_id = &p_department_no ;

        IF SQL%NOTFOUND THEN
           RAISE_APPLICATION_ERROR(-20202,
                'No such department id.');
        END IF ;

        COMMIT ;

        SELECT last_name INTO lname FROM employees
        WHERE  first_name='Johns';

     EXCEPTION
        WHEN NO_DATA_FOUND
        THEN
                RAISE_APPLICATION_ERROR
                (-20201,'There is not Such Employee name .');
     END;
     /
Enter value for p_department_name: Oracle
Enter value for p_department_no: 100
DECLARE
*
ERROR at line 1:
ORA-20201: There is not Such Employee name .
ORA-06512: at line 21

SQL>
```

```
SQL> SELECT department_name
     FROM hr.departments
     WHERE department_id = 100 ;

DEPARTMENT_NAME
------------------------------
Oracle

SQL>
```