suanlab

데이터베이스
Database

ORACLE®

DATABASE

16
프로시저와 함수

PROCEDURE
FUNCTION

프로시저(Procedure)

- 프로시저(Procedure)
  - 넓은 의미는 어떤 업무를 처리하기 위한 절차
  - 결과값 반환 없이 특정 로직을 처리
  - 질의의 집합으로 어떤 동작을 일괄 처리
  - 테이블에서 데이터 추출 및 조작, 결과를 다른 테이블에 저장하거나 갱신

- 프로시저 문법

```
CREATE OR REPLACE PROCEDURE 프로시저 이름
( 매개변수 이름 1 [ IN ¦ OUT ¦ IN OUT ]  데이터 타입,
  매개변수 이름 2 [ IN ¦ OUT ¦ IN OUT ]  데이터 타입, … )
IS ¦ AS
     변수 및 상수 선언
BEGIN
     실행 문장
     EXCEPTION 문장
END;
```

- 프로시저 실행

```
EXECUTE 프로시저 이름();
```

```
EXEC 프로시저 이름();
```

PL/SQL 변수 종류

- 일반 변수

```
count      NUMBER();
emp_name   VARCHAR2(10);
```

- 상수: CONSTANT 키워드 사용 (변경 불가)

```
count      CONSTANT   NUMBER();
emp_name   CONSTANT   VARCHAR2(10);
```

- %TYPE: 테이블 열 1개의 데이터 형식에 접근

```
emp_name   EMPLOYEES.EMPLOYEE_ID%TYPE
emp_email  EMPLOYEES.EMAIL%TYPE
```

- %ROWTYPE: 테이블 전체 열의 데이터 형식에 접근

```
emp   EMPLOYEES%ROWTYPE
dept  DEPARTMENTS%ROWTYPE
```

- 레코드(Record): 여러 개의 열의 데이터 형식을 지정

```
TYPE user_type IS RECORD (name VARCHAR2(10), email VARCHAR2(20));
user user_type;
```

- 컬렉션(Collection): 배열(Array)과 유사하며, VARRAY, 중첩 테이블(Nested Table), 연관 배열(Associative Array) 등이 있음

```
TYPE v_array_type IS VARRAY(5) OF NUMBER(10);
v_array v_array_type;
TYPE nest_tbl_type IS TABLE OF VARCHAR2(10);
nest_tbl nest_tbl_type;
TYPE a_array_type IS TABLE OF NUMBER(10) INDEX BY VARCHAR2(10);
a_array a_array_type;
```

CREATE OR REPLACE PROCEDURE

- 첫 번째 직원 출력 프로시저

```
CREATE OR REPLACE PROCEDURE first_emp AS
    emp_name VARCHAR2(20);
BEGIN
    SELECT first_name || ' ' || last_name INTO emp_name
    FROM employees WHERE employee_id = 100;
    DBMS_OUTPUT.PUT_LINE(emp_name);
END;
```

- 서버 출력 허용

```
SET SERVEROUTPUT ON;
```

- 프로시저 실행

```
EXECUTE first_emp();
```

- 직원 정보 출력 프로시저
  - 매개변수 사용(IN: 입력, OUT: 출력, IN OUT: 입출력 동시)

```
CREATE OR REPLACE PROCEDURE print_emp (
    emp_id IN EMPLOYEES.EMPLOYEE_ID%TYPE
) AS
    emp_name VARCHAR2(20);
BEGIN
    SELECT first_name || ' ' || last_name INTO emp_name
    FROM employees WHERE employee_id = emp_id;
    DBMS_OUTPUT.PUT_LINE(emp_name);
END;
```

- 프로시저 실행

```
EXECUTE print_emp(100);
```

CREATE OR REPLACE PROCEDURE

- 직원 평균 salary 출력 프로시저

```
CREATE OR REPLACE PROCEDURE emp_avg_salary (
    avg_salary OUT NUMBER
) AS
BEGIN
    SELECT AVG(salary) INTO avg_salary
    FROM employees;
END emp_avg_salary;
```

- 프로시저 실행

```
DECLARE
    avg_salary NUMBER;
BEGIN
    emp_avg_salary(avg_salary);
    DBMS_OUTPUT.PUT_LINE(avg_salary);
END;
```

- IF ELSE문 사용 프로시저

```
CREATE OR REPLACE PROCEDURE if_salary (
    salary IN NUMBER
) AS
    avg_salary  NUMBER;
BEGIN
    SELECT AVG(salary) INTO avg_salary FROM employees;

    IF salary >= avg_salary THEN
        DBMS_OUTPUT.PUT_LINE('평균 이상');
    ELSE
        DBMS_OUTPUT.PUT_LINE('평균 미만');
    END IF;
END;
```

- 프로시저 실행

```
EXECUTE if_salary(7000);
```

CREATE OR REPLACE PROCEDURE

- CASE문 사용 프로시저

```
CREATE OR REPLACE PROCEDURE case_hire_date (
    emp_email IN EMPLOYEES.EMAIL%TYPE
) AS
    hire_year   NCHAR(2);
    text_msg    VARCHAR2(20);
BEGIN
    SELECT TO_CHAR(hire_date, 'YY') INTO hire_year
    FROM employees
    WHERE email = emp_email;

    CASE
        WHEN (hire_year = '01') THEN text_msg := '01년도에 입사';
        WHEN (hire_year = '02') THEN text_msg := '02년도에 입사';
        WHEN (hire_year = '03') THEN text_msg := '03년도에 입사';
        WHEN (hire_year = '04') THEN text_msg := '04년도에 입사';
        WHEN (hire_year = '05') THEN text_msg := '05년도에 입사';
        WHEN (hire_year = '06') THEN text_msg := '06년도에 입사';
        WHEN (hire_year = '07') THEN text_msg := '07년도에 입사';
        WHEN (hire_year = '08') THEN text_msg := '08년도에 입사';
        WHEN (hire_year = '09') THEN text_msg := '09년도에 입사';
        ELSE text_msg := '01~09년도 이외에 입사';
    END CASE;
    DBMS_OUTPUT.PUT_LINE(text_msg);
END;
```

- 프로시저 실행

```
EXECUTE case_hire_date('SKING');
```

CREATE OR REPLACE PROCEDURE

- WHILE문 사용 프로시저

```
CREATE OR REPLACE PROCEDURE while_print AS
    str VARCHAR(100);
    i NUMBER;
BEGIN
    i := 1;
    WHILE (i <= 10) LOOP
        str := '반복 횟수:' || '(' || i || ')';
        DBMS_OUTPUT.PUT_LINE(str);
        i := i + 1;
    END LOOP;
END;
```

- 프로시저 실행

```
EXECUTE while_print();
```

CREATE OR REPLACE PROCEDURE

- OUT 파라미터 사용 프로시저

```
CREATE OR REPLACE PROCEDURE out_emp (
    emp_id IN EMPLOYEES.EMPLOYEE_ID%TYPE,
    out_str OUT VARCHAR2
) AS
    emp_name VARCHAR2(20);
BEGIN
    SELECT first_name || ' ' || last_name INTO emp_name
    FROM employees WHERE employee_id = emp_id;
    IF emp_id = NULL THEN
        out_str := '직원: 없음';
    ELSE
        out_str := '직원: ' || emp_name;
    END IF;
END;
```

- 프로시저 실행

```
DECLARE
    out_str VARCHAR2(30);
BEGIN
    out_emp(100, out_str);
    DBMS_OUTPUT.PUT_LINE(out_str);
END;
```

- 프로시저 실행 (예외 발생)

```
DECLARE
    out_str VARCHAR2(30);
BEGIN
    out_emp(300, out_str);
    DBMS_OUTPUT.PUT_LINE(out_str);
END;
```

CREATE OR REPLACE PROCEDURE

- 예외처리 사용 프로시저

```
CREATE OR REPLACE PROCEDURE out_emp (
    emp_id IN EMPLOYEES.EMPLOYEE_ID%TYPE,
    out_str OUT VARCHAR2
) AS
    emp_name VARCHAR2(20);
BEGIN
    SELECT first_name || ' ' || last_name INTO emp_name
    FROM employees WHERE employee_id = emp_id;

    out_str := '직원: ' || emp_name;
    EXCEPTION
        WHEN NO_DATA_FOUND THEN
            out_str := '직원: 없음';
END;
```

- 프로시저 실행 (예외 발생)

```
DECLARE
    out_str VARCHAR2(30);
BEGIN
    out_emp(300, out_str);
    DBMS_OUTPUT.PUT_LINE(out_str);
END;
```

- 프로시저 실행

```
DECLARE
    out_str VARCHAR2(30);
BEGIN
    out_emp(100, out_str);
    DBMS_OUTPUT.PUT_LINE(out_str);
END;
```

CREATE OR REPLACE PROCEDURE

- IN OUT 파라미터 사용 프로시저

```
CREATE OR REPLACE PROCEDURE in_out_emp (
    emp_name IN OUT VARCHAR2
) AS
BEGIN
    SELECT first_name || ' ' || last_name INTO emp_name
    FROM employees
    WHERE first_name = emp_name OR last_name = emp_name;

    emp_name := '직원: ' || emp_name;
    EXCEPTION
        WHEN NO_DATA_FOUND THEN
            emp_name := '직원: 없음';
END;
```

- 프로시저 실행

```
DECLARE
    emp_name VARCHAR2(30) := 'Lisa';
BEGIN
    in_out_emp(emp_name);
    DBMS_OUTPUT.PUT_LINE(emp_name);
END;
```

- 프로시저 실행 (예외 발생)

```
DECLARE
    emp_name VARCHAR2(30) := 'King';
BEGIN
    in_out_emp(emp_name);
    DBMS_OUTPUT.PUT_LINE(emp_name);
END;
```

CREATE OR REPLACE PROCEDURE

- rowtype 사용 프로시저

```
CREATE OR REPLACE PROCEDURE rowtype_emp (
    emp_id IN EMPLOYEES.EMPLOYEE_ID%TYPE
) AS
    emp_row EMPLOYEES%ROWTYPE;
BEGIN
    SELECT first_name, last_name, job_id
        INTO emp_row.first_name, emp_row.last_name, emp_row.job_id
    FROM employees WHERE employee_id = emp_id;
    DBMS_OUTPUT.PUT_LINE(emp_row.first_name ¦¦ ' ¦ ' ¦¦
                         emp_row.last_name ¦¦ ' ¦ ' ¦¦
                         emp_row.job_id);
END;
```

- 프로시저 실행

```
EXECUTE rowtype_emp(100);
```

```
EXECUTE rowtype_emp(200);
```

CREATE OR REPLACE PROCEDURE

- record 사용 프로시저

```
CREATE OR REPLACE PROCEDURE record_emp (
    emp_id IN EMPLOYEES.EMPLOYEE_ID%TYPE
) AS
    TYPE emp_type IS RECORD (first_name VARCHAR2(10),
                             last_name VARCHAR2(10),
                             job_id VARCHAR2(10));
    emp_record emp_type;
BEGIN
    SELECT first_name, last_name, job_id
        INTO emp_record.first_name, emp_record.last_name,
emp_record.job_id
    FROM employees WHERE employee_id = emp_id;
    DBMS_OUTPUT.PUT_LINE(emp_record.first_name ¦¦ ' ¦ ' ¦¦
                         emp_record.last_name ¦¦ ' ¦ ' ¦¦
                         emp_record.job_id);
END;
```

- 프로시저 실행

```
EXECUTE record_emp(100);
```

```
EXECUTE record_emp(200);
```

CREATE OR REPLACE PROCEDURE

- collection 사용 프로시저

```
CREATE OR REPLACE PROCEDURE collection_ex AS
    TYPE v_array_type IS VARRAY(5) OF NUMBER(10);
    TYPE nest_tbl_type IS TABLE OF VARCHAR2(10);
    TYPE a_array_type IS TABLE OF NUMBER(10) INDEX BY VARCHAR2(10);
    v_array v_array_type;
    nest_tbl nest_tbl_type;
    a_array    a_array_type;
    idx VARCHAR2(10);
BEGIN
    v_array := v_array_type(1, 2, 3, 4, 5);
    nest_tbl := nest_tbl_type('A', 'B', 'C', 'D', 'E');
    a_array('A') := 1;
    a_array('B') := 2;
    a_array('C') := 3;
    a_array('D') := 4;
    a_array('E') := 5;
    FOR i IN 1 .. 5 LOOP
        DBMS_OUTPUT.PUT_LINE(v_array(i) || ' | ' || nest_tbl(i));
    END LOOP;
    idx := a_array.FIRST;
    WHILE idx IS NOT NULL LOOP
        DBMS_OUTPUT.PUT_LINE(idx || ' : ' || a_array(idx));
        idx := a_array.NEXT(idx);
    END LOOP;
END;
```

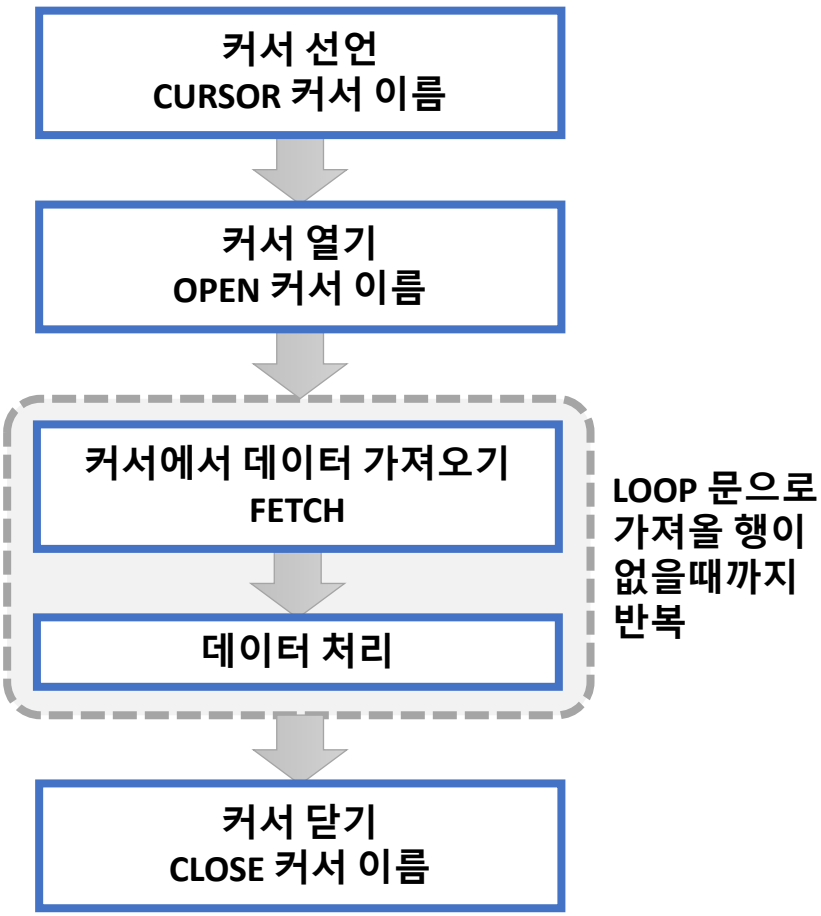
- 프로시저 실행

```
EXECUTE collection_ex();
```

커서(Cursor)

- 커서(Cursor)
  - 일반 프로그래밍 언어의 파일 처리 방법과 유사
  - 행의 집합을 다룰 수 있는 편리한 기능 제공
  - 테이블에서 여러 개의 행을 질의 후, 질의 결과인 행 집합을 한 행씩 처리
  - 프로시저 내부에서 커서 사용

- 커서 처리 순서



CREATE OR REPLACE PROCEDURE

- cursor 사용 프로시저

```
CREATE OR REPLACE PROCEDURE cursor_salary AS
    sal NUMBER := 0;
    cnt NUMBER := 0;
    total NUMBER := 0;
    CURSOR emp_cursor IS SELECT salary FROM employees;
BEGIN
    OPEN emp_cursor;
    LOOP
        FETCH emp_cursor INTO sal;
        EXIT WHEN emp_cursor%NOTFOUND;
        total := total + sal;
        cnt := cnt + 1;
    END LOOP;
    CLOSE emp_cursor;
    DBMS_OUTPUT.PUT_LINE('평균 SALARY: ' || (total / cnt));
END;
```

- 프로시저 실행

```
EXECUTE cursor_salary();
```

함수(Function)

- 함수(Function)
  - 프로시저의 각 프로세스를 수행하기 위해 필요한 기능
  - 일반적인 프로그래밍 언어에서 사용되는 함수와 같이 복잡한 프로그래밍 지원

- 함수 문법

```
CREATE OR REPLACE FUNCTION 함수 이름
( 매개변수 이름 1 데이터 타입,
  매개변수 이름 2 데이터 타입, … )
RETURN 데이터 타입
IS ¦ AS
     변수 및 상수 선언
BEGIN
     실행 문장
     RETURN 반환값
     EXCEPTION 문장
END;
```

- 프로시저와 함수 차이

프로시저	함수
• 특정 작업 수행	• 특정 계산 수행
• 리턴값이 없을수도 있음	• 리턴값이 반드시 존재해야 함
• 리턴값을 여러개 가질 수 있음	• 리턴값을 하나만 가질 수 있음
• 서버(DB)에서 기술	• 클라이언트에서 기술
• 수식내에서 사용 불가	• 수식내에서만 사용 가능
• 단독으로 문장 구성 가능	• 단독으로 문장 구성 불가

CREATE OR REPLACE FUNCTION

- to_yyyymmdd 함수

```
CREATE OR REPLACE FUNCTION to_yyyymmdd(date Date)
    RETURN VARCHAR2
IS
    char_date VARCHAR2(20);
BEGIN
    char_date := TO_CHAR(date, 'YYYYMMDD');
    RETURN char_date;
END;
```

- 함수 실행

```
SELECT to_yyyymmdd(SYSDATE) FROM dual;
```

- get_age 함수

```
CREATE OR REPLACE FUNCTION get_age(date Date)
    RETURN NUMBER
IS
    age NUMBER;
BEGIN
    age := TRUNC(MONTHS_BETWEEN(TRUNC(SYSDATE), to_yyyymmdd(date))
/ 12);
    RETURN age;
END;
```

- 함수 실행

```
SELECT get_age('20010101') FROM dual;
```

CREATE OR REPLACE FUNCTION

- 타입 정의

```
CREATE OR REPLACE TYPE ename_type AS OBJECT
(
    first_name VARCHAR2(20),
    last_name VARCHAR2(20)
);
```

- 테이블 타입 정의

```
CREATE OR REPLACE TYPE ename_table AS TABLE OF ename_type;
```

- 테이블 반환 함수 (Pipelined Table Function)

```
CREATE OR REPLACE FUNCTION emp_table(emp_id NUMBER)
    RETURN ename_table
    PIPELINED
IS
    ename ename_type;
BEGIN
    FOR emp IN(SELECT first_name, last_name FROM employees
               WHERE employee_id = emp_id)
    LOOP
        ename := ename_type(emp.first_name, emp.last_name);
        PIPE ROW(ename);
    END LOOP;

    RETURN;
END;
```

- 함수 실행

```
SELECT * FROM TABLE(emp_table(100));
```

[실습] 프로시저

- IF ELSIF ELSE 문을 사용하여 jobs 테이블의 최저, 최대 salary 평균값을 이용하여 입력으로 받은 salary가 최저 평균 이하인지 최대 평균 이상인지, 평균 구간인지 출력하는 프로시저 정의

- 프로시저 실행

[실습] 프로시저

- WHILE문을 사용하여 구구단 프로시저 정의

- 프로시저 실행

[실습] 프로시저

- Cursor를 이용해 employees 테이블에서 job_id가 'IT_PROG ' 인 직원의 first_name과 last_name을 가운데 공백을 두고 결합하여 출력하는 프로시저 정의

```

```

- 프로시저 실행

```

```

[실습] 함수

- job_title과 avg_salary를 가지는 job_salary_type 타입 정의

- job_salary_table_type 테이블 타입 정의

- job_title마다 평균 salary를 내림차순으로 출력하는 테이블 반환 함수 정의

-


suanlab
이수안 컴퓨터 연구소
suan computer laboratory