# 6-1 PL/SQL 시작하기

1. PL/SQL 이란?
2. PL/SQL 런타임 구조
3. PL/SAL 기본 구조
4. PL/SQL BLOCK 기본 구성
5. PL/SQL 블록 작성시 기본 규칙과 권장 사항
6. PL/SQL 문 내에서의 SQL 문장사용하기
7. PL/SQLDPTJDML FPRTLZKF
8. PL/SQL에서의 블록 구문작성 지침
9. 중첩된 PL./SQL 블록 작성하기
10. PL/SQL에서 연산자 사용하기

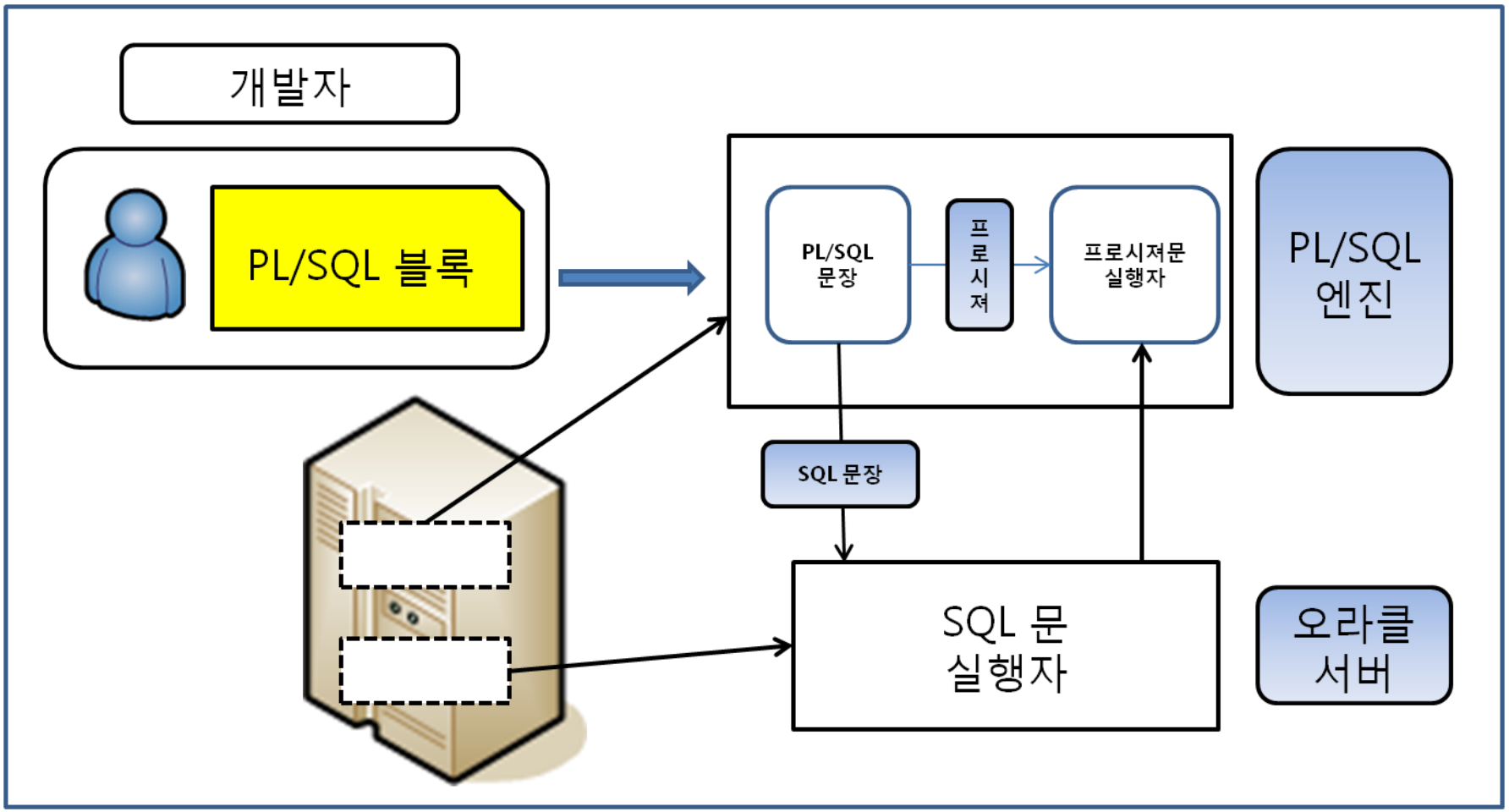
# 1. PL/SQL 이란?

## 1. PL/SQL 이란?

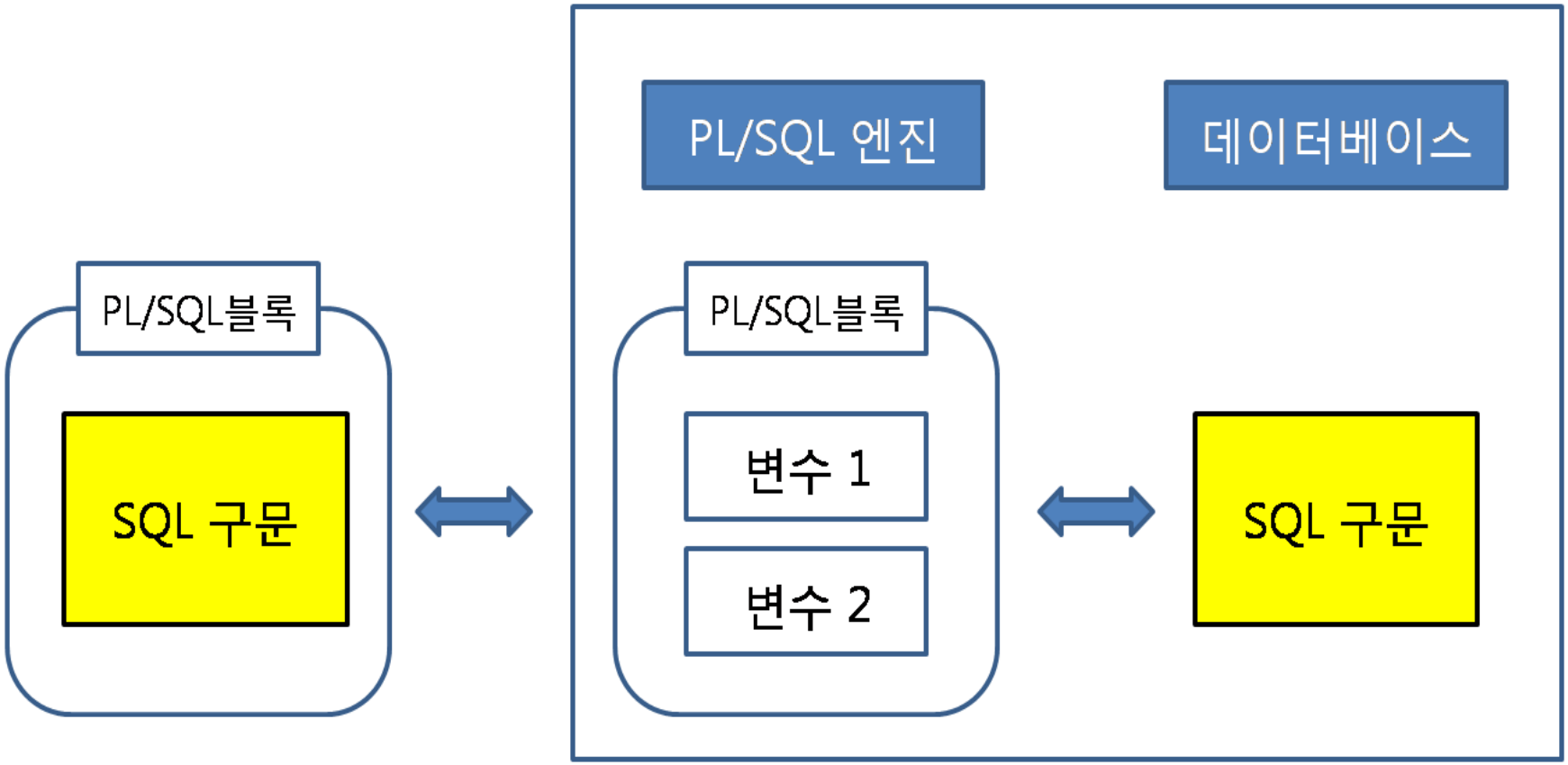
-오라클에서 제공하는 프로그래밍 언어

- Procedural Language/SQL 의 약자
- 일반 프로그래밍 언어적인 요소를 거의 다 가지고 있어서 실무에서 요구되는 절차적인 데이터 처리를 다 할 수 있음. 특히 SQL 과 연동되어서 막강한 기능을 구현할 수 있음.
- 데이터 트랜잭션 처리능력이나 정보보호, 데이터에 대한 보안, 예외처리 기능, 객체지향 등 데이터베이스와 관련된 중요한 모든 기능을 지원하기 때문에 데이터베이스 업무를 처리하기에 최적화된 언어임.

# 2. PL/SQL 의 런타임 구조 - 1



# 2. PL/SQL 의 런타임 구조 - 2



# 3. PL/SQL 기본 구조

- 선언부(DECLARE), 실행부(BEGIN), 예외처리부 (EXCEPTION)로 구성됨.

- Anonymous PL/SQL Block(익명블록) 과 Stored PL/SQL Block(저장된 블록)이 있음.

- 익명 블록은 주로 일회성으로 사용할 경우 많이 사용이 되고, 저장된 블록은 서버에 파싱해서 저장해 놓고 주기적으로 반복해서 사용할 경우 사용됨.

- Stored PL/SQL Block 은 다른 용어로 서브프로그램 또는 프로그램 단위라고도 하며, 스키마를 구성하는 오브젝트로서 파싱 된 후 오라클 서버 내부에 저장되거나 오라클 툴 안에 라이브러리 형태로 저장되어 있음.

# 4. PL/SQL BLOCK 기본구성

| | |
|---|---|
| Declare (선 언 부) | 모든 변수나 상수를 선언하는 부분 |
| Executable (실 행 부) | 제어문, 반복문, 함수정의 등의 로직을 기술함 |
| Exception (예외 처리부) | 실행 도중 에러 발생시 해결하는 문장들을 기술함 |

블록내의 각 부분에 포함되는 명령들 중 DECLARE, BEGIN , EXCEPTION 과 같은 예약어들은 ; (세미콜론) 으로 끝나지 않지만 나머지 명령어들은 SQL 문장처럼 세미콜론(;)으로 끝이 납니다

# 4. PL/SQL BLOCK 기본구성

**** 여기서 잠깐 ****

PL/SQL 은 기본적으로 처리된 PL/SQL 문장의 결과를 화면에 출력하지 않습니다.
그래서 결과를 화면에 출력하고 싶으면 아래와 같이 미리 사전작업이 필요합니다.

SQL> SET SERVEROUTPUT ON ; <- 화면 출력기능을 활성화 시킵니다.

# 4. PL/SQL BLOCK 기본구성

```
SCOTT>DECLARE  -- 선언부 시작됨
  2    vno NUMBER(4) ;
  3    vname  VARCHAR2(10) ;
  4
  5  BEGIN -- 실행부 시작됨
  6    SELECT empno , ename  INTO  vno , vname
  7    FROM  emp
  8    WHERE  empno=7900 ;
  9
 10  DBMS_OUTPUT.PUT_LINE(vno||'    '||vname) ;
 11
 12  END ; -- 실행부 종료됨
 13  /
7900   JAMES  -- 출력 결과입니다

PL/SQL procedure successfully completed.
```

데이터베이스에서 수행된 결과 값을 저장할 변수 두 개를 선언했습니다.

데이터베이스에서 수행된 결과 값을 위에서 선언한 두 변수에 저장했습니다.

두 변수에 저장된 값을 화면에 출력합니다.

작성된 PL/SQL 블록을 실행시킵니다.

# 5. PL/SQL 블록 작성시 기본 규칙과 권장 사항

-문장은 여러 줄에 걸쳐질 수 있으나, 키워드는 분리될 수 없음(SQL 문법과 동일)

- 블록의 내용을 읽기 쉽도록 공백문자를 사용하여 키워드 내지는 문장을 적절하게 분리함으로써 의미분석이 되도록 하며, 들여쓰기도 권장함.

- 예약어는 식별자명으로 사용될 수 없으나, Alias 로는 사용될 수 있음. 즉 이중부호(" ")를 함께 사용할 수는 있다는 뜻입니다. (예: " TABLE " )

- 식별자명은 기본 오라클 Naming Rule 을 준수함

- 리터럴(문자,날짜)은 단일인용부호(')로 표시해야 하며 널 값은 NULL 상수로 기술함

- 주석처리를 하고자 할 경우에 단일 행은 - - (하이픈 두 개), 복수 행은 /* 로 시작하고 */ 로 종료함

- PL/SQL 블록내의 명령(수식)에서는 오라클 함수를 사용할 수 있으나 그룹함수와 DECODE 함수는 SQL 문장에 포함되어야만 사용될 수 있습니다. 만약 다른 경우에 그룹함수와 DECODE 함수를 사용할 경우 에러가 발생하며 PL/SQL 에서는 에러(EXCEPTION)처리와 관련된 함수에는 SQLCODE 함수와 SQLERRM 함수가 별도로 존재합니다.

# 6. PL/SQL 문 내에서의 SQL 문장 사용하기

- END 키워드는 트랜잭션의 끝이 아니라 PL/SQL 블록의 끝을 나타냄
- PL/SQL은 DDL(데이터 정의어) 문을 직접 지원하지 않음.
  DDL 문은 동적 SQL 문임. 동적 SQL문은 런타임에 문자열로 작성되며
  파라미터의 위치 표시자를 포함할 수 있음.
  따라서 동적 SQL을 사용하면 PL/SQL에서 DDL 문을 실행할 수 있음.
- PL/SQL은 GRANT 또는 REVOKE와 같은 DCL(데이터 제어어) 문을 직접
  지원하지 않음, 그러나 앞의 DDL 문과 마찬가지로 동적 SQL을  사용하여
  DCL 문을 실행할 수 있음.

# 6. PL/SQL 문 내에서의 SQL 문장 사용하기

## 1) PL/SQL 내에서의 SELECT 문장 사용하기

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

# 6. PL/SQL 문 내에서의 SQL 문장 사용하기

**- 사용 예 1 :**
**professor 테이블에서 교수번호가 1001 번인 교수의 교수번호와 급여를 조회 한 후 변수에 저장해서 화면에 출력하세요**

```
SCOTT>DECLARE
  2    v_profno   professor.profno%TYPE ;
  3    v_pay      professor.pay%TYPE ;
  4  BEGIN
  5    SELECT profno , pay INTO v_profno ,v_pay
  6    FROM professor
  7    WHERE profno=1001 ;
  8
  9  DBMS_OUTPUT.PUT_LINE(v_profno||' 번 교수의 급여는 '||v_pay||' 입니다') ;
 10
 11  END ;
 12  /
1001 번 교수의 급여는 550 입니다

PL/SQL procedure successfully completed.
```

# 6. PL/SQL 문 내에서의 SQL 문장 사용하기

**사용 예 2:**
**emp2 테이블을 사용하여 사원번호를 입력 받아서 사원의 사 번과 이름, 생일을 출력하세요.**

```
SCOTT>DECLARE
  2    v_empno   emp2.empno%TYPE ;
  3    v_name    emp2.name%TYPE ;
  4    v_birth   emp2.birthday%TYPE;
  5  BEGIN
  6    SELECT empno, name, birthday
  7    INTO v_empno ,v_name, v_birth
  8    FROM emp2
  9    WHERE empno = '&empno' ;
 10    DBMS_OUTPUT.PUT_LINE(v_empno||'  '||v_name||'  '||v_birth);
 11  END ;
 12  /

Enter value for empno: 20000102
20000102  김설악  22-MAR-83

PL/SQL procedure successfully completed.
```

# 6. PL/SQL 문 내에서의 SQL 문장 사용하기

**사용 예 3:**
**교수번호를 입력 받은 후 professor 테이블을 조회하여 해당 교수의 교수번호와 교수이름, 부서번호, 입사일을 출력하세요.**

```
SCOTT>DECLARE
  2    v_profno    professor.profno%TYPE ;
  3    v_name      professor.name%TYPE ;
  4    v_deptno    professor.deptno%TYPE ;
  5    v_hdate     professor.hiredate%TYPE ;
  6
  7  BEGIN
  8    SELECT profno, name, deptno , hiredate
  9    INTO v_profno, v_name, v_deptno, v_hdate
 10    FROM professor
 11    WHERE profno = '&교수번호' ;
 12
 13    DBMS_OUTPUT.PUT_LINE(v_profno||'  '||v_name||'  '||v_deptno||'  '||v_hdate);
 14
 15  END;
 16  /
```

# 6. PL/SQL 문 내에서의 SQL 문장 사용하기

## 2) PL/SQL 내에서의 DML 문장 사용하기

### (1) INSERT 문장 수행하기 예 1:

```
SCOTT>CREATE TABLE pl_test
  2  (no   number ,
  3   name   varchar2(10)) ;

Table created.

SCOTT>CREATE SEQUENCE pl_seq ;
Sequence created.
```

실습을 위해 연습용 테이블과 시퀀스를 생성합니다.

# 6. PL/SQL 문 내에서의 SQL 문장 사용하기

- **PL/SQL 에서 INSERT 를 수행합니다.**

```
SCOTT> BEGIN
  2    INSERT INTO pl_test
  3    VALUES(pl_seq.NEXTVAL,'AAA');
  4  END ;
  5   /

PL/SQL procedure successfully completed.

SCOTT>/
PL/SQL procedure successfully completed.

SCOTT>SELECT * FROM pl_test ;

    NO      NAME
---------- ----------
         1   AAA
         2   AAA

SCOTT>commit ;
 Commit complete.
```

# 6. PL/SQL 문 내에서의 SQL 문장 사용하기

**(2) INSERT 문장 수행하기 예 2 :**

```
SCOTT>CREATE TABLE pl_test2
  2  (no   number ,
  3   name   varchar2(10),
  4   addr   varchar2(10) );

Table created.
```

실습용 테이블 생성하기

**-사용자로부터 번호(no) , 이름(name) , 주소(addr) 값을 입력 받은 후 pl_test2 테이블에 입력하는 PL/SQL 문장을 작성하세요.**

```
SCOTT> SET VERIFY OFF
SCOTT>DECLARE
  2  v_no number := '&no';
  3  v_name varchar2(10) := '&name' ;
  4  v_addr varchar2(10) := '&addr' ;
  5
```

뒷장에 계속.....

# 6. PL/SQL 문 내에서의 SQL 문장 사용하기

```
  6  BEGIN
  7   INSERT INTO pl_test2
  8   VALUES(v_no, v_name, v_addr) ;
  9  END ;
 10  /
Enter value for no: 10
Enter value for name: AAA
Enter value for addr: 서울

PL/SQL procedure successfully completed.

SCOTT>SELECT * FROM pl_test2;

    NO    NAME     ADDR
---------- ---------- ----------
    10     AAA        서울
```

# 6. PL/SQL 문 내에서의 SQL 문장 사용하기

- **PL/SQL 에서 UPDATE 를 수행합니다.**

```
SCOTT>BEGIN
  2    UPDATE pl_test
  3    SET name='BBB'
  4    WHERE no = 2 ;
  5  END ;
  6  /

PL/SQL procedure successfully completed.

SCOTT>SELECT * FROM pl_test ;

    NO    NAME
---------- ----------
      1    AAA
      2    BBB

SCOTT>commit;
Commit complete.
```

# 6. PL/SQL 문 내에서의 SQL 문장 사용하기

**- PL/SQL 에서 MERGE 작업을 수행합니다.**

```
SCOTT>CREATE TABLE pl_merge1
  2  ( no   number ,
  3    name varchar2(10));

Table created.

SCOTT>CREATE TABLE pl_merge2
  2  AS SELECT * FROM pl_merge1 ;

Table created.

SCOTT>INSERT INTO pl_merge1
VALUES(1,'AAA');
1 row created.

SCOTT>INSERT INTO pl_merge1
VALUES(2,'BBB');
1 row created.
```

```
SCOTT>INSERT INTO pl_merge2
VALUES(1,'CCC');
1 row created.

SCOTT>INSERT INTO pl_merge2
VALUES(3,'DDD');
1 row created.

SCOTT>commit;
Commit complete.

SCOTT>SELECT * FROM pl_merge1 ;

     NO   NAME
---------- ----------
      1   AAA
      2   BBB
```

# 6. PL/SQL 문 내에서의 SQL 문장 사용하기

```
SCOTT>SELECT * FROM pl_merge2;

    NO    NAME
---------- ----------
      1    CCC
      3    DDD

SCOTT>  BEGIN
  2     MERGE INTO pl_merge2 m2
  3     USING pl_merge1 m1
  4     ON(m1.no = m2.no)
  5     WHEN MATCHED THEN
  6       UPDATE SET
  7        m2.name = m1.name
  8     WHEN NOT MATCHED THEN
  9       INSERT VALUES(m1.no , m1.name);
 10   END ;
 11   /

PL/SQL procedure successfully completed.
```

```
SCOTT>SELECT * FROM pl_merge1 ;
    NO    NAME
---------- ----------
      1    AAA
      2    BBB

SCOTT>SELECT * FROM pl_merge2 ;
    NO    NAME
---------- ----------
      1    AAA
      3    DDD
      2    BBB  ← 이 줄이 추가되었음을
                  알 수 있습니다.
```

# 7. PL/SQL 에서의 렉시칼 - 문자집합

- **식별자:**
식별자는 PL/SQL 객체에게 부여되는 이름임.
즉 테이블 이름이나 변수명 등은 모두 식별자임.
오라클 키워드는 식별자로  사용할 수 없음.
식별자 중에서 특별히 아래와 같은 경우 식별자를 따옴표로 묶어서 사용 가능함.

  - **식별자의 대소문자 구분이 필요한 경우**
  - **공백과 같은 문자 포함할 경우**
  - **예약어를 사용해야 할 경우**

이러한 변수를 연이어 사용할 때는 항상 큰 따옴표(쌍 따옴표)로 묶어야 하지만 따옴표로 묶인 식별자를 사용하는 것은 권장하지 않음.

# 7. PL/SQL 에서의 렉시칼 - 문자집합

**-구분자:**

구분자는 특별한 의미를 지닌 기호입니다. 예를 들어 SQL 문장을 끝낼 때는 끝내는 의미를 가진 ; (세미콜론)을 사용합니다.

| 기 호 | 의 미 | | 기 호 | 의 미 |
|---|---|---|---|---|
| + | 더하기 연산자 | | <> | 부등호 연산자 |
| - | 빼기 / 부정 연산자 | | != | 부등호 연산자 |
| * | 곱하기 연산자 | | \|\| | 연결 연산자 |
| / | 나누기 연산자 | | -- | 단일행 주석 표시자 |
| = | 등호 연산자 | | /* | 주석 시작 구분자 |
| @ | 원격 엑세스 표시자 | | */ | 주석 종료 구분자 |
| ; | 명령문 종료자 | | := | 할당 연산자 |

# 7. PL/SQL 에서의 렉시칼 - 문자집합

- **리터럴:**

엄밀히 말하면 변수에 할당되는 모든 값은 리터럴임.
이 말은 식별자가 아닌 모든 문자, 숫자, 부울 또는 날짜 값은 리터럴 이라는 의미.
리터럴은 종류가 여러 가지가 있는데 주요 리터럴은 다음과 같이 분류.

- **문자 리터럴:** 모든 문자열 리터럴은 데이터 유형이 CHAR 또는 VARCHAR2이므로 문자 리터럴이라고 함(예: abcd 및 12f 등)

- **숫자 리터럴:** 숫자 리터럴은 정수 또는 실수 값을 나타냅니다(예: 123 및 1.234 등)

- **부울 리터럴:** 부울 변수에 할당된 값은 부울 리터럴. TRUE, FALSE 및 NULL 은 부울 리터럴이거나 키워드.

# 7. PL/SQL 에서의 렉시칼 - 문자집합

**-주석**

프로그래밍을 하면서 해당 프로그램의 의도나 순서등의 설명을 기록하는 것은 아주 좋은 습관입니다. 이렇게 직접 프로그래밍 코드의 성능에는 영향을 주지 않지만 나중을 위해서 설명이나 해설 등을 기록해 두는 것을 주석이라고 합니다.

이 주석은 한 줄 주석기호인 –(하이픈 2개) 를 사용할 수도 있고 /* 주석 */ 의 방식으로 여러 줄에 걸쳐서 기록할 수도 있습니다.

# 8. PL/SQL 에서의 블록 구문 작성 지침

1) 문자 리터럴이나 날짜 리터럴 사용시에는 반드시 홑 따옴표로 묶어서 표시해야 함.

2) 문장에서의 주석은 한 줄일 경우 -- (하이픈 두 개)를 써서 표시하고 여러 줄 일 경우 /* ~ */ 기호를 사용해서 표시해야 함.

3) 프로시져 내에서는 단일행 함수만을 사용해야 하며 DECODE 함수나 그룹 함수는 사용할 수 없음. 이 말의 의미는 PL/SQL 내부에 포함되어 있는 SQL 문장에서는 위 함수들을 쓸 수 있지만 그 외의 PL/SQL 문장에서는 사용 할 수 없다는 의미.

4) 시퀀스를 사용할 때 - 11g 이전 버전에서는 시퀀스를 사용하기 위해서는 SQL 문장을 이용하여 시퀀스를 변수에 할당 한 후 해당 변수 값을 사용했으나 11g 버전부터는 PL/SQL 문장에서 바로 시퀀스를 사용 할 수 있음.

5) 데이터의 형 변환에 주의해야 함. 데이터의 형 변환은 묵시적 형 변환(자동 형 변환) 과 명시적 형 변환 (수동 형 변환) 으로 나눌 수 있음. 이 부분은 SQL 에서의 규칙과 동일. 묵시적 형 변환은 문자와 숫자 , 문자와 날짜를 연산 할 때 발생하며 이 부분 때문에 성능에 의도하지 않게 나쁜 영향을 줄 수 있으므로 데이터 형의 일치에 항상 주의.

# 9. 중첩된 PL/SQL 블록 작성하기

11번 줄의 v_second 변수 때문에 에러가 발생함.
변수의 범위가 중요함

```
SCOTT>DECLARE
  2    v_first VARCHAR2(5) := 'Outer';
  3  BEGIN
  4    DECLARE
  5      v_second  VARCHAR2(5) := 'Inner';
  6    BEGIN
  7      DBMS_OUTPUT.PUT_LINE(v_first);
  8      DBMS_OUTPUT.PUT_LINE(v_second);
  9    END ;
 10   DBMS_OUTPUT.PUT_LINE(v_first);
 11   DBMS_OUTPUT.PUT_LINE(v_second);
 12  END ;
 13  /
DBMS_OUTPUT.PUT_LINE(v_second);
                     *
ERROR at line 11:
ORA-06550: line 11, column 23:
PLS-00201: identifier 'V_SECOND' must be
declared
ORA-06550: line 11, column 2:
PL/SQL: Statement ignored
```

# 9. 중첩된 PL/SQL 블록 작성하기

문제의 라인을 삭제 한 후 실행하니 정상적으로 수행 완료됩니다.

```
SCOTT> DECLARE
  2    v_first VARCHAR2(5) := 'Outer';
  3  BEGIN
  4    DECLARE
  5      v_second  VARCHAR2(5) := 'Inner';
  6    BEGIN
  7      DBMS_OUTPUT.PUT_LINE(v_first);
  8      DBMS_OUTPUT.PUT_LINE(v_second);
  9    END ;
 10   DBMS_OUTPUT.PUT_LINE(v_first);
 11  END ;
 12  /
Outer   ← 7번 줄의 결과값
Inner   ← 8번 줄의 결과값
Outer   ← 10번 줄의 결과값

PL/SQL procedure successfully completed.
```

# 10. PL/SQL 에서의 연산자 사용하기

| 연 산 자 | 의 미 |
|---|---|
| ** | 제곱 연산자 |
| + , - | 일치 , 부정 |
| * , / | 곱하기 , 나누기 |
| + , - , \|\| | 더하기 , 빼기 , 연결하기 |
| =, <, >, <=, >=, <>, !=, ~=, ^=,<br>IS NULL, LIKE, BETWEEN, IN | 비교 연산자 |
| NOT | 논리 부정 연산자 |
| AND | 두 조건 모두 참일 경우 참 을 반환 |
| OR | 두 조건 중 한가지만 참 일 경우 참 을 반환 |