1. **PL/SQL**

|  |
| --- |
| - SQL : 구조화된 질의 언어  - 만약다음과 같은 질의가 필요하다면?  주민등록번호 검증 로직   * SQL 로 처리하기는 불가능하다. 이러한 문제를 해결해 주기 위한 문법이 PL/SQL 이다. |

**1.1 PL/SQL?**

“Procedural language extension to Structured Query Language(SQL)의 약자임

🡺 “SQL을 확장한 순차적 처리 언어”

PL/SQL은 데이터 베이스 질의어인 SQL과 일반 프로그래밍 언어의 특성을 결합한 언어이다.

Oracle6(1989년) 버전부터 등장했고, 현재 그 기능이 계속 추가되고 있다.

**PL/SQL의 기본 구조**

기본 단위는 블록(block), 하나의 블록은 세 부분으로 구분됨

1. 선언부(declarative part) : 실행부에서 사용할 변수나 상수를 선언하는 부분

* 변수와 상수를 선언할 수 있다.
* ‘DECLARE; 을 사용해서 나타냄

1. 실행부(executable part) : 실제 처리할 로직을 담당하는 부분

* 선언된 변수에 값을 할당
* IF 나 WHLE 같은 문장을 사용
* SQL 문장들을 사용
* PL/SQL에서 엔진 역할을 담당하는 부분임
* 여러 개의 SQL문장을 순차적으로 처리할 수 있다.(블록 단위 처리)
* ‘BEGIN’으로 시작되어 ‘END’로 끝남

1. 예외처리부(exception-building part) : 실행부에서 발생할 수 있는 각종 오류들을 처리하는 부분

* Oracle은 오류 발생시 자동으로 ORA-XXX 오류 번호와 함께 메시지가 리턴됨
* 사용자가 직접 이러한 메시지를 대체하거나 오류가 발생할 경우 처리할 로직을 기술할 부분임
* ‘EXCEPTION’ 키워드를 사용함

|  |
| --- |
| **DECLARE**  counter INTEGER;  **BEGIN**  Counter := counter + 1;  IF counter IS NULL THEN  Dbms\_output.put\_line(‘Result : COUNTER IS NULL’ );  END IF;  **END**; |

\*) 선언부에서 counter 라는 INTEGER형 변수를 사용함, ;(세미콜론) 로 종료됨

\*) 변수에 할당시 ‘=’ 가 아닌 ‘:=’ 를 사용함

\*) EXCEPTION은 생략 가능

**PL/SQL의 실행**

🡺 SQL \*Plus, SQL DEVELOPER, TOAD, 오랜지 등의 툴에서 모두 사용 가능

|  |
| --- |
| **DECLARE**  counter INTEGER;  **BEGIN**  Counter := counter + 1;  IF counter IS NULL THEN  Dbms\_output.put\_line(‘Result : COUNTER IS NULL’ );  END IF;  **END**;  *PL/SQL 처리가 정상적으로 완료되었습니다.* |

\*) 결과값인 ‘Result : COUNTER IS NULL’ 결과 출력이 나오지 않음

SQL > **SET SERVEROUTPUT ON;**

메시지 출력 요청 쿼리문

\*) 다시 위 구문을 실행하면 출력결과 확인 가능

SQL> / 🡺 이전 명령을 다시 수행하기 위한 구문

Ex) 구구단 2단 출력

|  |
| --- |
| **DECLARE**  counter INTEGER;  I INTEGER;  **BEGIN**  FOR i IN 1..10 Loop  counter := ( 2 + i );  dbms\_output.put\_line(‘ 2 \* ‘ || i || ’ = ‘ || counter );  END LOOP;  **END**;  / |

Ex) 예외 처리부

|  |
| --- |
| **DECLARE**  counter INTEGER;  **BEGIN**  counter := 10;  counter := counter / 0 ;  dbms\_output.put\_line(counter );  **END**;  *Error report:*  *ORA-01476 : 제수가 0 입니다.* |
| **DECLARE**  counter INTEGER;  **BEGIN**  counter := 10;  counter := counter / 0 ;  dbms\_output.put\_line(counter );  EXCEPTION WHEN OTHERS THEN  dbms\_output.put\_line(‘ERROR’);  **END**;  *anonymous block completed*  *ERRORS* |

\*) PL/SQL 의 구성요소

변수와 상수 / %TYPE / %ROWTYPE

콜렉션

레코드

IF문 / CASE문 / LOOP문 / WHILE-LOOP문 / FOR…LOOP문 / GOTO 문 / NULL문

커서

🡺 추후 개별 확인

1. **함수**

|  |
| --- |
| - 함수 : 보통 오라클에서 제공하는 SQL 함수를 의미하고  - PL/SQL에서 함수 : 사용자 정의 함수(user defined function)를 보통 의미한다. |

**2.1 함수의 구문 형식 이해**

|  |
| --- |
| **CREATE OR REPLACE FUNCTION** 함수명 ( 파라미터1 데이터타입,  파라미터2 데이터타입,  …)  **RETURN** 데이터 타입 **IS [AS]**  변수 선언… ;  **BEGIN**  처리내용…. ;  **RETURN** 리턴값;  **END**; |

\*) **CREATE OR REPLACE FUNCTION :** 뷰를 생성할 때와 같이 CREATE OR REPLACE를 사용하여서 기존에 작성된 함수를 수정할 경우, 별도로 DROP 시키지 않아도 된다.

\*) 함수명 : 생성할 함수명을 명시한다.

\*) 파라미터1, … : 함수의 파라미터로 오는 파라미터 이름과 데이터 타입을 명시한다.

\*) RETURN : 반환할 데이터 타입을 명시한다.

\*) BEGIN…END : 함수가 처리할 내용을 명시하며, 맨 마지막에 함수가 반환할 값을 RETURN 다음에 명시한다.

Ex) 함수 사용예

|  |
| --- |
| **CREATE OR REPLACE FUNCTION** emp\_salaris ( emp\_id NUMBER )  **RETURN** NUMBER **IS**  nSalaries NUMBER(9);  **BEGIN**  nSalaries := 0;  SELECT salary  INTO nSalaries  FROM employees  WHERE employee\_id = emp\_id;  **RETURN** nSalaries;  **END**;  *FUNCTION emp\_salaries Compiled* |
| [함수 사용]  **SELECT** emp\_salaris(100)  **FROM DUAL**; |
| [출력 결과]  EMP\_SALARIES(100)  --------------------------  24000  *1rows selected* |
| [함수 사용]  **SELECT** employee\_id, first\_name, emp\_salaries(employee\_id)  **FROM** employees  **WHERE** department\_id = 100; |

Ex) EMPLOYEES 테이블에는 사원이 소속된 부서의 부서번호만 가지고 있다. 따라서 부서명을 알

고 싶을 경우 사용할 수 있는 방법은 두 가지 이다.

- DEPARTMENTS 테이블과의 조인

- SELECT 리스트에서 서브쿼리를 사용

서브쿼리 자체도 DEPARTMENTS 테이블과 내부적으로 조인을 맺게 된다.

🡺 만약, 단순히 부서이름만 가져온다고 한다면 서브쿼리 사용이 좋다.

- 더 낳은 방법은 함수를 사용하는 것이다.

|  |
| --- |
| -- 파라미터로 부서번호가 넘어오면 해당 부서번호에 맞는 부서명을 찾아 반환하는 함수임  **CREATE OR REPLACE FUNCTION** get\_dep\_name ( dep\_id NUMBER )  **RETURN** VARCHAR2 **IS**  sDepName VARCHAR2(30);  **BEGIN**  SELECT department\_name  INTO sDepName  FROM departments  WHERE department\_id = dep\_id;  **RETURN** sDepName;  **END**;  *FUNCTION get\_dep\_name Compiled* |
| [서브 쿼리를 사용하는 방법]  **SELECT** a.employee\_id, a.first\_name || ‘ ‘ || a.last\_name names,  **(SELECT** b.department\_name  **FROM** departments b  **WHERE** a.department\_id = b.department\_id ) dep\_names  **FROM** employees a  **WHERE** a.department\_id = 100 |
| [함수를 사용하는 방법]  **SELECT** a.employee\_id, a.first\_name || ‘ ‘ || a.last\_name names,  get\_dep\_name( a.department\_id ) dep\_names  **FROM** employees a  **WHERE** a.department\_id = 100;  🡺 조인이 수행되지 않는다. 가상의 테이블이 만들어지지 않으므로(부서명만 리턴함) 데이터 양이 많을수록 성능상의 차이가 발생하게 된다. |

1. **프로시저**

|  |
| --- |
| - 특정한 처리를 수행하는 PL/SQL 서브프로그램이다.  - 함수와 마찬가지로 데이터베이스에 저장된 객체이다.  따라서 내장프로시져라고도 한다.  - 파라미터들을 받아 특정 처리를 수행하기는 하지만, 함수와는 달리 값을 반환하지 않는다. |

**1.1 구문형식의 이해**

|  |
| --- |
| **CREATE OR REPLACE PROCEDURE** 프로시저명  (파리미터1 데이터타입 [IN | OUT | INOUT ],  …  )  **IS [AS]**  변수 선언부… ;  **BEGIN**  프로시저 본문처리… ;  **EXCEPTION**  예외처리… ;  **END;** |

**1.2 사용예**

신규 사원이 입사할 경우 담당자는 매번 새로운 사번으로 신입사원을 등록하고 이메일 계정, 출입카드 등을 발급하는 일을 한다.

이러한 작업을 하려면 보통 여러 개의 테이블에 데이터를 INSERT 해야 한다.(번거로움)

신입사원 등록 프로시저를 만들어 놓고, 신입사원이 들어올 때마다 이 프로시저를 호출한다면 매우 편리할 것이다.

1. 프로시저 생성

|  |
| --- |
| **CREATE OR REPLACE PROCEDURE** register\_emp  (f\_name VARCHAR2,  f\_name VARCHAR2,  l\_name CARCHAR2,  e\_acct VARCHAR2  j\_id VARCHAR2) **IS**  **BEGIN**  INSERT INTO employees (employee\_id, first\_name, last\_name, email, hire\_date, job\_id)  VALUES (EMPLOYEES\_SEQ.NEXTVAL, f\_name, l\_name, e\_acct, sysdate, j\_id);  **COMMIT**;  **EXCEPTION WHEN OTHERS THEN**  DBMS\_OUTPUT.PUT\_LINE(f\_name || ‘ ‘ || l\_name || ‘ register is failed!’ );  **ROLLBACK**;  **END;**  *PROCEDURE register\_emp compiled* |

\*) 파라미터로 신입사원의 성과 이름, 이메일 계정을 입력받아 테이블에 데이터틀 INSERT한다.

성곻하면 COMMIT을 예외가 발생하면 오류 메시지를 보여주고 ROLLBACK한다.

\*) 특이점은 파라미터 명시시 구체적인 자리수를 지정하지 않아도 된다.

만약, 구체적인 자리수를 명시하면 오류가 발생한다.

1. 프로시저 사용

|  |
| --- |
| **[구문 형식]**  **EXEC** 혹은 **EXECUTE** 프로시저명(파라미터…) |
| **EXCUTE** register\_emp (‘조영’, ‘대’, ‘djoyoung’, ‘IT\_PROG’);  *Anonymous block completed* |

1. 등록 확인

|  |
| --- |
| **SELECT** first\_name, last\_name, email, job\_id, hire\_date  **FROM** employees  **WHERE** last\_name = ‘대’; |