

06. 함수와 저장프로시저

- ✓ 사용자 정의 함수
- ✓ 저장 프로시저
- ✓ 패키지
- ✓ 커서



Oracle내의 프로그래밍 기능

□ 함수 :

- ▶ 주로 Select문을 사용한 프로그래밍
- ▶ 반환값은 필수이며, 릴레이션을 반환할 수도 있음.

□ 저장 프로시저 :

- ▶ 매개변수 사용
- ▶ 프로시저 Call은 EXECUTE문을 이용함

□ 패키지 : 변수, 함수, 프로시저의 묶음

□ 커서 : 다중 튜플 처리

함수와 저장프로시저의 차이점

□ 함수 vs. 저장 프로시저

	함수	저장 프로시저
호출	- Prompt :EXECUTE 함수이름 - SQL내 : 함수이름 - 프로시저 내 : 함수이름	- Prompt :EXECUTE Proc. 이름 - 프로시저 내 : Proc. 이름
파라미터	입력 파라미터(묵시적) 값 변경 불가	IN, OUT, IN OUT
반환값	RETURN문으로 반환 하나의 값을 반환	변환 구문 없음 여러개 OUT 파라미터사용가능
SQL사용	함수내 SQL 사용 불가	프로시저내 SQL 사용 가능

(사용자 정의) 함수

ORA.6 #1 (1).sql

□ 함수 형식

[함수의 형식]

```
CREATE OR REPLACE FUNCTION 이름 ( 파라미터 ) RETURN 반환값유형
-- 파라미터가 없으면 ( )도 생략
AS
    변수선언부
BEGIN
    PL/SQL Statements. ...
    RETURN 반환값
END;
```

□ 데이터베이스에 상관없이 단순 연산하는 예제

[함수의 정의 예제]

```
CREATE OR REPLACE FUNCTION F_Add(a number, b number)
    RETURN NUMBER
AS
    Total NUMBER;
BEGIN
    Total := a + b;
    RETURN Total;
END; -- sum, add등은 예약어로 함수, 변수 이름으로 사용하면 안 됨
```

Simple Function : 호출

□ 호출 1 : Prompt에서 호출

[함수 호출1]

```
VAR result NUMBER;  
EXECUTE :result := F_Add(2020,35);  
  -- EXECUTE 뒤의 문장은 무명프로시저로 봐야 함.  
  -- 외부 변수를 사용하려면 ':변수이름'형식을 사용해야 함  
PRINT result;
```

□ 호출 2 : SQL 내에서 호출

[함수 호출2]

```
SELECT F_Add(2020, 35) FROM DUAL;
```

□ 호출 3 : PL/SQL 프로시저 내에서 호출

[함수 호출3]

```
DECLARE  
  result1 NUMBER;  
BEGIN  
  result1 := F_Add(2020,50);  
  DBMS_OUTPUT.PUT_LINE(result1);  
END;  
PRINT result;  
PRINT result1; -- 오류. result1은 PL/SQL 프로시저 내에서만 사용가능
```

DB와 연계한 함수 사용

- 함수 이름 : 근무년수
- SELECT 내의 이름, 임용년도는 교수 테이블의 속성

[함수 정의]

```
CREATE OR REPLACE FUNCTION 근무년수(임용년도 NUMBER)
RETURN NUMBER
AS
    근무기간 NUMBER;
BEGIN
    근무기간 := EXTRACT(YEAR FROM SYSDATE) - 임용년도;
    RETURN 근무기간;
END 근무년수;
```

[함수 호출]

```
SELECT 이름, 근무년수(임용년도) || '년' as 근무 FROM UNIV.교수;
-- 이름, 임용년도, 교수는 DB요소임
```

IF, CASE문을 이용한 다용도 함수 선언

```
CREATE OR REPLACE FUNCTION F_Multi(inValue NCHAR)    RETURN NCHAR
AS
    V_output NCHAR(20);
BEGIN
    CASE
        WHEN inValue IS NULL THEN
            V_output := ' ';
        WHEN SUBSTR(inValue, 1, 1) BETWEEN '0' and '9' THEN
            BEGIN
                IF SUBSTR(inValue, 6, 1) = '-' THEN
                    V_output := CONCAT(SUBSTR(inValue, 1, 7), '??');
                ELSIF SUBSTR(inValue, 7, 1) = '-' THEN
                    V_output := CONCAT(SUBSTR(inValue, 1, 8), '*****');
                ELSE
                    V_output := CONCAT(SUBSTR(inValue, 1, 4), '####');
                END IF;
            END;
        ELSE V_output := CONCAT( SUBSTR(inValue, 1, 1), 'OO');
    END CASE;
    RETURN  V_output;
END F_Multi;
```

IF, CASE문을 이용한 다용도 함수 이용

□ 개인 정보 보호를 위해 암호문자 처리

- ▶ 학번, 주민등록번호, 이름, 연락처
- ▶ 정보보호 처리를 위해 모두 하나의 함수 'F_Multi'를 이용

```
SELECT * from UNIV.학생;  
  
SELECT F_Multi(학번) AS "학번",  
       F_Multi(주민등록번호) AS "주민번호",  
       F_Multi(이름) AS "성명",  
       F_Multi(전화번호) AS "연락처"  
FROM UNIV.학생;
```


PIPELINED TABLE FUNCTION

□ 테이블을 반환하는 함수

□ 사용방법

[타입정의] - 튜플+테이블순

```
CREATE OR REPLACE TYPE 튜플타입이름 AS OBJECT
```

```
(
```

```
    열이름 데이터타입,
```

```
    ... ..
```

```
);
```

```
CREATE OR REPLACE TYPE 테이블타입이름 AS TABLE OF 튜플타입이름;
```

[테이블함수 정의]

```
CREATE OR REPLACE FUNCTION 함수이름( )
```

```
    RETURN 테이블타입이름 PIPELINED
```

```
AS
```

```
BEGIN
```

```
    PL/SQL문장들...(반복문과 PIPE_ROW( )사용)
```

```
    RETURN;
```

```
END;
```

[테이블함수 사용]

```
SELECT * FROM TABLE(함수이름( ) );
```

Pipelined Table Function 예제

□ 년도시리즈와 사건시리즈를 String으로 입력하면 테이블 반환

[타입정의] - 튜플+테이블순

-- 튜플타입정의

```
CREATE OR REPLACE TYPE EventType AS OBJECT  
( V_Year NUMBER(10), V_Event NCHAR(10) );
```

-- 테이블타입 정의

```
CREATE OR REPLACE TYPE TableType AS TABLE OF EventType;
```

[테이블함수 정의]

```
CREATE OR REPLACE FUNCTION 사건테이블생성(발생년도들 NCHAR, 사건들 NCHAR)  
    RETURN TableType PIPELINED  
AS  
BEGIN  
    ..... 다음 슬라이드 참조  
END;
```

[테이블함수 사용]

```
SELECT *  
FROM TABLE(사건테이블생성('1392,1492,1592,1692,1792',  
    '조선건국,콜롬부스,임진왜란,호접지몽,기요틴'));
```

[테이블함수 정의]

```
CREATE OR REPLACE FUNCTION 사건테이블생성(발생년도들 NCHAR, 사건들 NCHAR)  
RETURN TableType PIPELINED
```

```
AS
```

```
V_Years NVARCHAR2(100) := 발생년도들; -- 함수의 파라미터는 In용. 값 변경 불가  
V_Events NVARCHAR2(100) := 사건들; -- 함수의 파라미터는 In용. 값 변경 불가  
V_EventTuple EventType; -- 튜플한개  
V_YearsPos NUMBER; -- 년도 문자열에서 추출할 위치  
V_EventsPos NUMBER; -- 사건 문자열에서 추출할 위치  
V_Year NUMBER(10); -- 추출한 1개의 년도  
V_Event NCHAR(10); -- 추출한 1개의 사건
```

```
BEGIN
```

```
LOOP
```

```
V_YearsPos := INSTR(V_Years, ',');
```

```
V_EventsPos := INSTR(V_Events, ',');
```

```
IF V_YearsPos > 0 AND V_EventsPos > 0 THEN
```

```
V_Year := TO_NUMBER( SUBSTR(V_Years, 1, V_YearsPos-1) );
```

```
V_Event := SUBSTR(V_Events, 1, V_EventsPos-1);
```

```
V_EventTuple := EventType(V_Year, V_Event);
```

```
PIPE ROW(V_EventTuple);
```

```
V_Years := SUBSTR(V_Years, V_YearsPos+1);
```

```
V_Events := SUBSTR(V_Events, V_EventsPos+1);
```

```
ELSE
```

```
V_EventTuple := EventType(V_Years, V_Events); -- Last Data Insertion
```

```
PIPE ROW(V_EventTuple);
```

```
EXIT;
```

```
END IF;
```

```
END LOOP;
```

```
RETURN;
```

```
END 사건테이블생성;
```

□ DB 서버에 저장해 놓은 프로시저.

- ▶ 어떤 연산(6장의 PL/SQL 프로그래밍 문장)들을 모아 놓은 것
 - 필요할 때마다 Call하여 실행.
- ▶ 파라미터 선언부는 생략 가능

[저장 프로시저 정의 형식]

```
CREATE [OR REPLACE] PROCEDURE 프로시저이름(  
    파라미터선언부  
)  
AS  
    변수선언부  
  
BEGIN  
  
    PL/SQL 문장들  
    ...  
END
```

[저장 프로시저 실행 형식]

```
EXECUTE 프로시저이름( Parameter );
```

저장프로시저 Example

□ 정의

- ▶ 파라미터 없고, 커서 불필요(SQL문의 결과가 단일값)

[정의 : Simple 저장 프로시저]

```
CREATE OR REPLACE PROCEDURE SP_Simple  -- 파라미터 없음
AS
    Age NUMBER; -- 변수 선언
BEGIN
    -- 정소화의 나이를 프로시저 변수 'Age'에 할당
    SELECT 나이 INTO Age FROM 고객
        WHERE 고객이름 = '정소화';
    DBMS_OUTPUT.PUT_LINE ('정소화의 나이는 ' || Age); -- 변수 값 출력
END SP_Simple ;
```

□ 실행

[실행 : Simple 저장 프로시저]

```
SET SERVEROUTPUT ON;

EXECUTE SP_Simple( );
```

저장 프로시저 오류 예제(1)

- 컴파일 오류 : SP_Simple 이미 존재하는 경우.
 - ▶ 대체하고자 하면, 또는 여러번 수정 후 컴파일하고자 하면
 - CREATE PROCEDURE가 아닌 CREATE or REPLACE PROCEDURE 사용

```
CREATE PROCEDURE SP_Simple
AS
    Age NUMBER; -- 변수 선언
BEGIN
    -- 정소화의 나이를 프로시저 변수 'Age'에 할당
    SELECT 나이 INTO Age FROM 고객
        WHERE 고객이름 = '정소화';
    DBMS_OUTPUT.PUT_LINE ('정소화의 나이는 ' || Age); -- 변수 값 출력
END SP_Simple ;
```

저장 프로시저 오류 예제(2)

□ 컴파일 오류는 없지만 ...

```
CREATE OR REPLACE PROCEDURE SP_Simple
AS
    Age NUMBER; -- 변수 선언
BEGIN
    -- 정소화의 나이를 프로시저 변수 'Age'에 할당
    SELECT 나이 INTO Age FROM 고객;
    DBMS_OUTPUT.PUT_LINE ('정소화의 나이는 ' || Age); -- 변수 값 출력
END SP_Simple ;
```

□ 실행시간 오류

```
EXECUTE SP_Simple( );
```

- ▶ SQL결과값이 여러개. 커서가 필요한 경우

파라미터가 있는 저장 프로시저 형식

[저장 프로시저 정의 형식]

```
CREATE [OR REPLACE] PROCEDURE 프로시저이름(  
파라미터선언부  
)  
AS  
.....
```

□ 파라미터 선언부의 형식

▶ 입력파라미터

[입력 파라미터 선언부 형식]

파라미터변수 IN 데이터타입 [:= 디폴트값]

▶ 출력파라미터

[출력 파라미터 선언부 형식]

파라미터변수 OUT 데이터타입

□ 실행시 파라미터 전달 방식

[파라미터 전달 방식]

EXECUTE 저장프로시저이름(파라미터값);

저장프로시저 with Input Parameter

□ 파라미터 처리방법은 일반적인 프로그래밍과 거의 유사

▶ 호출하는 문장

- In Parameter : R-Value(값을 가진 변수, 상수, 연산식)
- Out Parameter : L-Value(값을 받을 변수)

```
CREATE OR REPLACE PROCEDURE SP_In (  
    U_ID IN 고객.고객아이디%TYPE  
    -- 아래 문장도 유효  
    -- U_ID IN CHAR -- ';' & 크기 입력 불가  
)  
AS  
    U_Name VARCHAR(18); -- ';' & 크기 입력해야 함  
BEGIN  
    SELECT 고객이름 INTO U_Name FROM 고객  
        WHERE 고객아이디 = U_ID;  
    DBMS_OUTPUT.PUT_LINE (U_Name);  
END ;  
  
SET SERVEROUTPUT ON;  
EXECUTE SP_In('apple');
```

저장프로시저 with In, Out Parameter

```
CREATE OR REPLACE PROCEDURE SP_InOut1 (  
    Pi_ID IN CHAR,  
    Pi_Name IN CHAR,  
    Po_적립금 OUT NUMBER  
) AS  
    --v_count VARCHAR(10);  
BEGIN  
    INSERT INTO 고객(고객아이디, 고객이름, 등급) VALUES(Pi_ID, Pi_Name, 'Silver');  
    SELECT MAX(적립금) INTO Po_적립금 FROM 고객;  
END ;  
SET SERVEROUTPUT ON;  
DECLARE  
    Out적립 NUMBER;  
BEGIN  
    SP_InOut1('fruits', '홍길동', Out적립);  
    DBMS_OUTPUT.PUT_LINE (Out적립);  
END;
```

저장프로시저 : PL/SQL 문장 포함

□ IF, CASE, 반복문, DDL, DML등 PL/SQL문장이 모두 가능

```
CREATE OR REPLACE PROCEDURE SP_IF (  
    Pi_고객이름 고객.고객이름%type  
) AS  
    V_적립금 NUMBER; -- 출생년도를 저장할 변수  
BEGIN  
    SELECT 적립금 INTO V_적립금 FROM 고객  
        WHERE 고객이름 = Pi_고객이름;  
    IF V_적립금 >= 4000 THEN  
        DBMS_OUTPUT.PUT_LINE ('단골 고객이네요');  
    ELSE  
        DBMS_OUTPUT.PUT_LINE ('많이 이용하면 더 많은 혜택이 있어요');  
    END IF;  
END ;  
  
SET SERVEROUTPUT ON;  
EXECUTE SP_IF ('정소화');
```

저장 프로시저의 예외 처리(1)

□ 오류 구문의 예 :

- ▶ No Data Found라는 원치않는 오류 발생
- ▶ IF문을 수행하기 전에 Select문에서 오류발생.
 - 이런 일은 흔함. 깔끔한 예외처리가 필요함

```
CREATE OR REPLACE PROCEDURE SP_Error (  
    Pi_고객이름 IN 고객.고객이름%TYPE,  
    Po_고객아이디 OUT 고객.고객아이디%TYPE  
) AS  
BEGIN  
    SELECT 고객아이디 INTO Po_고객아이디 FROM 고객  
        WHERE 고객이름 = Pi_고객이름;  
    IF Po_고객아이디 = NULL THEN  
        Po_고객아이디 := 'Anonymous';  
    END IF;  
END ;  
DECLARE  
    비회원여부 NCHAR(10);  
BEGIN  
    SP_Error('조나단', 비회원여부);  
    -- SP_Error('정소화', 비회원여부); -- 오류없음  
    DBMS_OUTPUT.PUT_LINE (비회원여부);  
END;
```

저장 프로시저의 예외 처리(2)

□ 예외 처리 구문을 포함 : EXCEPTION 부분

```
CREATE OR REPLACE PROCEDURE SP_Error (  
    Pi_고객이름 IN 고객.고객이름%TYPE,  
    Po_고객아이디 OUT 고객.고객아이디%TYPE  
) AS  
BEGIN  
    SELECT 고객아이디 INTO Po_고객아이디 FROM 고객  
        WHERE 고객이름 = Pi_고객이름;  
    EXCEPTION WHEN NO_DATA_FOUND THEN  
        Po_고객아이디 := 'Anonymous';  
END ;  
  
DECLARE  
    비회원여부 NCHAR(10);  
BEGIN  
    SP_Error('조나단', 비회원여부);  
    -- SP_Error('정소화', 비회원여부); -- 오류없음  
    DBMS_OUTPUT.PUT_LINE (비회원여부);  
END;
```

저장프로시저 with InOut Parameter

□ 입력과 출력을 동시에 할 수 있는 파라미터

```
CREATE OR REPLACE PROCEDURE SP_InOut (  
    Pio_고객이름 IN OUT 고객.고객이름%TYPE  
) AS  
    V_등급 고객.등급%TYPE;  
BEGIN  
    SELECT 등급 into V_등급 FROM 고객 WHERE 고객이름 = Pio_고객이름;  
    EXCEPTION WHEN NO_DATA_FOUND THEN  
        Pio_고객이름 := (Pio_고객이름||'-');  
END ;  
  
DECLARE  
    세부고객이름 고객.고객이름%TYPE := '정소화';  
BEGIN  
    SP_INOUT(세부고객이름);  
    DBMS_OUTPUT.PUT_LINE (세부고객이름);  
END;  
  
DECLARE  
    세부고객이름 고객.고객이름%TYPE := '조나단';  
BEGIN  
    SP_INOUT(세부고객이름);  
    DBMS_OUTPUT.PUT_LINE (세부고객이름);  
END;
```

저장 프로시저 : 일부 검색

□ 고객이름으로 고객 아이디 일부분만 검색

```
CREATE OR REPLACE PROCEDURE SP_PartialID (  
    Pi_고객이름 IN 고객.고객이름%TYPE,  
    Po_부분아이디 OUT CHAR  
)  
AS  
BEGIN  
    SELECT RPAD( SUBSTR(고객아이디,1,3), LENGTH(고객아이디), '*') INTO Po_부분아이디  
    FROM 고객 WHERE 고객이름 = Pi_고객이름;  
    EXCEPTION WHEN NO_DATA_FOUND THEN  
        Po_부분아이디 := '비회원';  
END;  
  
DECLARE  
    V_고객아이디 CHAR(15);  
BEGIN  
    SP_PartialID('정소화', V_고객아이디);  
    DBMS_OUTPUT.PUT_LINE (V_고객아이디);  
END;
```

GUI로 저장 프로시저 내용 조회 및 삭제

□ 선언된 프로시저 및 그 내용 조회와 삭제 가능

The screenshot displays the Oracle SQL Developer interface. On the left, the 'Object Explorer' shows a tree structure for a schema named '로컬-HMart'. Under the '프로시저' (Procedures) folder, the procedure 'SP_IN' is selected. The main editor window shows the SQL code for 'create or replace PROCEDURE SP_In'. The code defines a procedure with an input parameter 'U_ID' and a variable 'U_Name', which is used in a SELECT statement to retrieve the customer name from the '고객' table.

On the right, a context menu is open, showing options for the selected procedure. The '삭제(H)...' (Delete) option is highlighted, indicating the process of deleting the procedure. A speech bubble with the Korean word '삭제' (Delete) points to this option.

```
create or replace PROCEDURE SP_In (  
    U_ID IN 고객.고객아이디%TYPE  
    -- 아래 문장도 유효  
    -- U_ID IN CHAR -- 크기 입력 불가  
)  
AS  
    U_Name VARCHAR(18); -- 크기 입력해야  
BEGIN  
    SELECT 고객이름 INTO U_Name FROM 고객  
        WHERE 고객아이디 = U_ID;  
    DBMS_OUTPUT.PUT_LINE (U_Name);  
END ;
```


□ 서버에 암호화되어 보관됨. 코드 내용을 볼 수 없음(보안 효과)



저장 프로시저 암호화 예제

□ VARCHAR2에 저장 프로시저를 생성하는 DDL구문 기록

- ▶ 문자열을 표현할 때는 작은 따옴표 두개(' ')로 감싸야 함
- ▶ 결합연산(II) 이므로 한 줄을 마칠 때는 공백을 한칸씩 둘 것

```
DECLARE
  Hidden_Source VARCHAR2(32767);
BEGIN
  Hidden_Source :=
    'CREATE OR REPLACE PROCEDURE SP_ENCR ( ' ||
    'Pi_고객이름 IN 고객.고객이름%TYPE, Po_부분아이디 OUT CHAR ) AS ' ||
    'BEGIN ' ||
    'SELECT RPAD( SUBSTR(고객아이디,1,3), LENGTH(고객아이디), "**") INTO Po_부분아이디 ' ||
    'FROM 고객 WHERE 고객이름 = Pi_고객이름; ' ||
    'EXCEPTION WHEN NO_DATA_FOUND THEN Po_부분아이디 := "비회원"; ' ||
    'END;' ;
EXECUTE IMMEDIATE DBMS_DDL.WRAP(DDL => Hidden_Source);
END;
```

```
DECLARE
  V_고객아이디 CHAR(15);
BEGIN
  SP_ENCR('정소화', V_고객아이디);
  DBMS_OUTPUT.PUT_LINE (V_고객아이디);
END;
```

```
DECLARE
  V_고객아이디 CHAR(15);
BEGIN
  SP_ENCR('트럼프', V_고객아이디);
  DBMS_OUTPUT.PUT_LINE (V_고객아이디);
END;
```

저장 프로시저의 암호화 결과

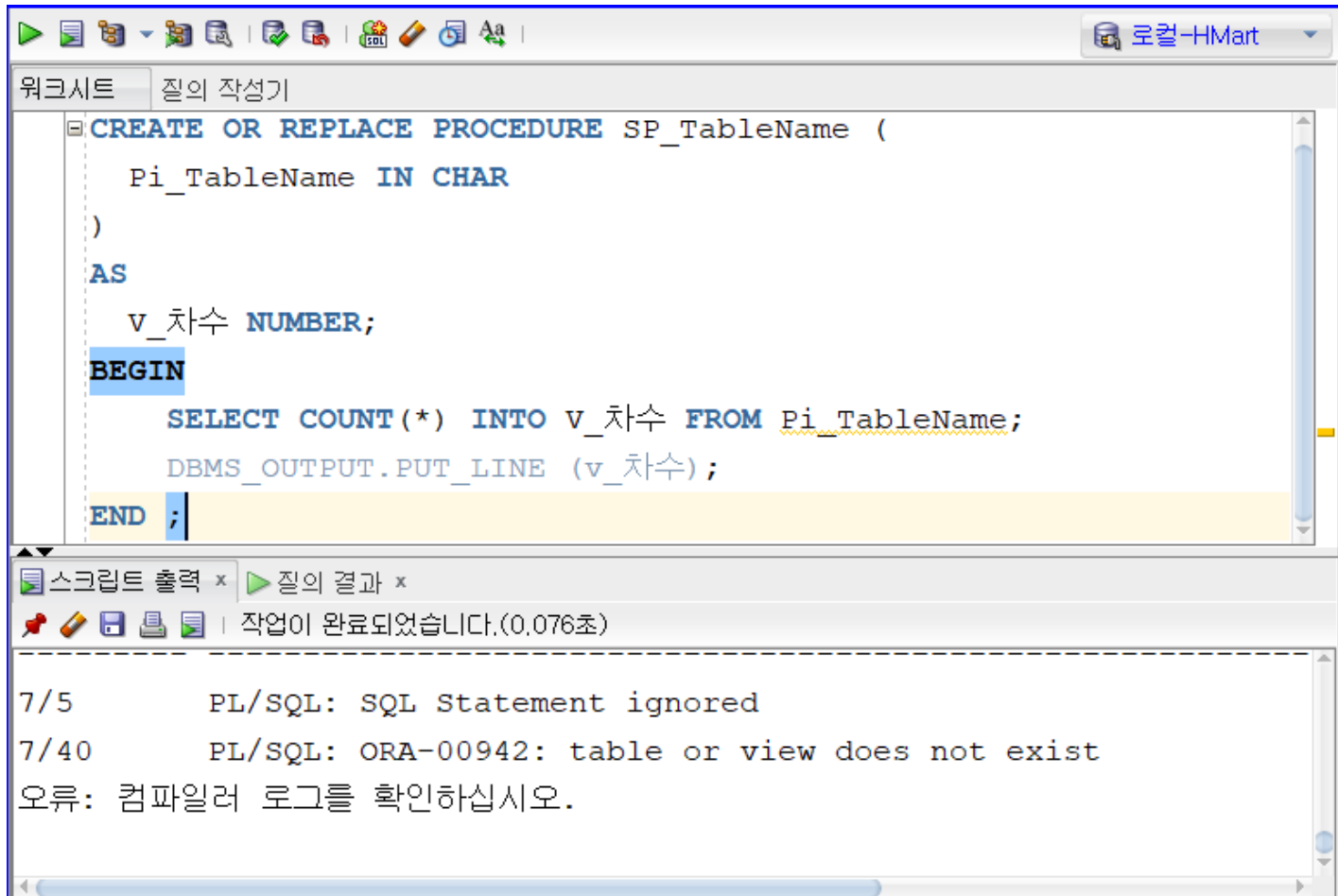
The screenshot shows a database IDE with a left-hand tree view and a main code editor. In the tree view, the '프로시저' (Procedures) folder is expanded, and 'SP_ENCR' is highlighted with a red dashed circle. The main editor displays the SQL code for the 'SP_ENCR' procedure, which is wrapped in a 'create or replace PROCEDURE' statement. The code includes a series of 'abcd' strings and a long alphanumeric string, all of which are encrypted. The status bar at the bottom indicates the file is '로컬-HMart' and the current line is 12:1.

```
create or replace PROCEDURE SP_ENCR wrapped
a000000
369
abcd
abcd
abcd
abcd
abcd
abcd
abcd
abcd
abcd
abcd
abcd
abcd
abcd
7
14a 140
vY2D2zSzw3uQvQ69BK/HqoxVKi0owg3mu7SfhNQTSzOj3mLeEyPUY1UX4KaR1VQH+2tMKEJrt
1WIJddTKGvJiBDYP+pg0hCvgZ5XAYXUkh18UXK2sxDeg9WeeukDr+U6gBWEFu4sKYCYIB/ix
9Zwfp8BUtuxqwq13zRIQbzz2dRiome7INTBZEW8dpq7H8XQXAvLuCvQXnIeb+ZA6yqqXwEeh
clNTH4zI0vFW4jhOTT18R4LRhAgJaeT1TuDptfy2I42RviFShyEREbaJcPPo6CWez+gcKrLG
WuwLcwOTDSwgoV+zJ3c6ROwajSc=
```

저장프로시저 : 테이블 이름 전달

□ In 파라미터 변수를 From절에 사용

- ▶ 오류 발생. 변수이름을 테이블 이름으로 인식



The screenshot shows a SQL IDE window with a toolbar at the top. The main editor displays a PL/SQL procedure named `SP_TableName` with a parameter `Pi_TableName` of type `CHAR`. The procedure body includes a `BEGIN` block with a `SELECT COUNT(*)` statement using `Pi_TableName` in the `FROM` clause, followed by `DBMS_OUTPUT.PUT_LINE` and an `END` statement. Below the editor, there are tabs for '스크립트 출력' (Script Output) and '질의 결과' (Query Results). The '스크립트 출력' tab is active, showing the following output:

```
7/5      PL/SQL: SQL Statement ignored
7/40     PL/SQL: ORA-00942: table or view does not exist
오류: 컴파일러 로그를 확인하십시오.
```

저장 프로시저 : 테이블 이름 전달. 동적질의

- VARCHAR2 변수에 질의문을 구성
- 동적 질의 수행

```
CREATE OR REPLACE PROCEDURE SP_TableName_Dynamic (  
    Pi_표이름 IN CHAR )  
AS  
    V_수 NUMBER;  
    동적질의 VARCHAR2(300);  
BEGIN  
    동적질의 := 'SELECT COUNT(*) FROM ' || Pi_표이름;  
    EXECUTE IMMEDIATE 동적질의 INTO V_수;  
    DBMS_OUTPUT.PUT_LINE (Pi_표이름 || '수는 ' || V_수 || '입니다');  
END ;  
  
EXEC SP_TableName_Dynamic('고객');  
EXEC SP_TableName_Dynamic('제품');  
EXEC SP_TableName_Dynamic('주문');
```



저장 프로시저의 장점

- 서버 중심의 보안 관리에 편리함
- 네트워크 전송량의 감소. 단, 서버의 부하 증가
- 모듈식 루틴 관리 가능
 - ▶ 유지 관리의 용이성
- 예외 처리 가능
- 트리거와 연계

□ 함수와 저장 프로시저의 묶음

- ▶ System 패키지와 사용자 정의 패키지로 구분

□ System 패키지

- ▶ 오라클에서 제공하는 모든 패키지를 확인

```
SELECT * FROM ALL_OBJECTS WHERE OBJECT_TYPE = 'PACKAGE';
```

- ▶ DBMS_OUTPUT 패키지의 모든 프로시저 확인

- ▣ 많이 사용하는 PUT_LINE 프로시저를 확인할 수 있음

```
SELECT * FROM ALL PROCEDURES WHERE OBJECT_NAME = 'DBMS_OUTPUT';
```

- ▶ 패키지 프로시저의 소스 코드 확인

```
SELECT TEXT FROM ALL_SOURCE WHERE NAME = 'DBMS_OUTPUT';
```

사용자 생성 패키지

□ 관리가 편하고 효율적

- ▶ 자주 함께 사용하는 변수, 함수, 저장 프로시저 등을 묶음

[사용자 생성 패키지 HEADER 형식]

CREATE OR REPLACE PACKAGE 패키지이름
AS

선언부 - 변수, 커서, 예외, 함수, 저장 프로시저
END {패키지 이름};

[사용자 생성 패키지 BODY 형식]

CREATE OR REPLACE PACKAGE BODY 패키지이름
AS

구현부 - 변수, 커서, 예외, 함수, 저장 프로시저
END {패키지 이름};

[패키지 내의 프로시저 실행]

EXEC 패키지이름.저장프로시저이름

패키지 예제

□ 하나의 전역변수, 하나의 Function, 하나의 저장프로시저

```
CREATE OR REPLACE PACKAGE Pack1 AS
    Total NUMBER; -- global variable
    FUNCTION F_Add(a number, b number) RETURN NUMBER;
    PROCEDURE SP_Simple;
END Pack1;

CREATE OR REPLACE PACKAGE BODY Pack1 AS
    FUNCTION F_Add(a number, b number) RETURN NUMBER    AS
    BEGIN
        Total := a + b;
        RETURN Total;
    END F_Add;
    PROCEDURE SP_Simple    AS
        Age NUMBER; -- local variable
    BEGIN
        SELECT 나이 INTO Age FROM 고객 WHERE 고객이름 = '정소화';
        DBMS_OUTPUT.PUT_LINE ('정소화의 나이는 ' || Age); -- 변수 값 출력
    END SP_Simple;
END Pack1;
```

[패키지 내의 함수, 프로시저 실행]

```
SELECT Pack1.F_Add(10,20) from dual;
EXECUTE Pack1.SP_Simple;
```

□ 부하 불일치 해결

- ▶ 쿼리 결과 - 테이블(여러개의 튜플) ::: 프로시저의 처리 : 한 Record씩
- ▶ 프로시저가 커서를 사용하여 반복문으로 처리
 - 이 때 커서는 한 튜플을 처리할 때마다 테이블의 그 다음 튜플을 가리킴

□ 커서를 이용한 처리 순서



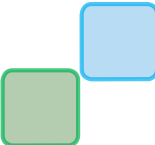
커서 사용 예제

□ 질의

- ▶ '고객의 적립금 중 2000이상 적립금 평균값과 2000미만 적립금 평균값의 차이를 구하라.
- ▶ 그리고 2000미만 고객의 등급을 BASIC으로 바꿔라'

```
CREATE OR REPLACE PROCEDURE SP_Cursor AS
  V_High NUMBER := 0; -- 2000이상 적립금 합계
  V_Low NUMBER := 0; -- 2000미만 적립금 합계
  V_H_Num NUMBER := 0 ; -- 2000이상 고객의 수
  V_L_Num NUMBER := 0 ; -- 2000미만 고객의 수
  V_ID VARCHAR2(20); -- 고객 고객아이디
  V_적립금 NUMBER; -- 고객 적립금
  CURSOR C IS
    SELECT 고객아이디, 적립금 FROM 고객; -- 커서 정의
```

---- 다음 슬라이드 계속



```
BEGIN
OPEN C; -- 커서 열기
-- 데이터 인출 및 처리
LOOP
    FETCH C INTO V_ID, V_적립금;
    EXIT WHEN C%NOTFOUND; -- 데이터가 없으면 LOOP 종료
    IF V_적립금 >= 2000 THEN
        BEGIN V_High := V_High + V_적립금; V_H_Num := V_H_Num + 1; END;
    ELSE
        BEGIN
            V_Low := V_Low + V_적립금;
            V_L_Num := V_L_Num + 1;
            UPDATE 고객 SET 등급 = 'BASIC' WHERE 고객아이디 = V_ID;
            -- 고객아이디와 VID가 데이터타입이 완전 일치해야 함
            DBMS_OUTPUT.PUT_LINE(V_ID);
        END;
    END IF;
END LOOP;
CLOSE C; -- 커서 닫기
DBMS_OUTPUT.PUT_LINE('결과 : ' || TO_CHAR((V_High/V_H_Num)-(V_Low/V_L_Num)) );
END ;

SET SERVEROUTPUT ON;
EXECUTE SP_Cursor();
```