15장 프로시저 , 함수와 트리거

이번 장에서는 자주 사용되는 쿼리문을 모듈화하기 위한 저장 프로시저와 함수를 생성하고 고치고 지우는 작업들을 학습하고 커서에 대한 개념과 커서를 사용하기 위한 방법을 살펴보도록 하겠습니다.

학습 내용

- 프로시저
- 함수
- 트리거

학습목표

- 프로시저의 개념과 장점을 이해할 수 있습니다.
- 프로시저를 생성, 수정, 제거할 수 있습니다.
- 프로시저의 INPUT 매개변수, OUTPUT 매개변수를 사용할 수 있습니다.
- 함수의 개념을 설명할 수 있고 함수를 정의하고 사용할 수 있습니다.
- 특정 테이블의 데이터에 변경이 가해졌을 때 자동으로 수행되는 트리거를 작성할 수 있습니다.
- 트리거 및 new와 old 테이블의 개념을 말할 수 있습니다.

01. 프로시저

- 프로시저는 일련의 작업들을 하나로 묶어서 저장해 두었다가 호출하여 이런 작업들이 실행할 수 있게 해줍니다.

- 저장 프로시저를 생성하기 위한 CREATE PROCEDURE의 형식은 다음과 같습니다.

```
CREATE [OR REPLACE ] PROCEDURE prcedure_name
( argument1 [mode] data_taye,
  argument2 [mode] data_taye . . .
)
IS
local_variable declaration
BEGIN
statement1;
statement2;
. . .
END;
/
```

01. 프로시저

- 저장 프로시저를 생성하려면 CREATE PROCEDURE 다음에 새롭게 생성하고자하는 프로시저 이름을 기술합니다.

- 이렇게 해서 생성한 저장 프로시저는 여러 번 반복해서 호출해서 사용할 수 있다는 장점이 있습니다.

- OR REPLACE 옵션은 이미 학습한 대로 이미 같은 이름으로 저장 프로시저를 생성할 경우 기존 프로시저는 삭제하고 지금 새롭게 기술한 내용으로 재 생성하도록 하는 옵션입니다.

- 프로시저를 실행시키기 위해서는 EXECUTE 명령어를 사용합니다. EXECUTE 뒤에 호출하고자 하는 프로시저 이름을 명시해야 합니다.

```
EXECUTE procedure_name
```

01. 프로시저

- 저장 프로시저를 작성한 후 사용자가 저장 프로시저가 생성되었는지 확인하려면 USER_SOURCE 살펴보면 됩니다.

- USER_SOURCE의 내용을 조회하면 어떤 저장 프로시저가 생성되어 있는지와 해당 프로시저의 내용이 무엇인지 확인할 수 있습니다.

```
select name, text from user_source
where name like('%SP_SALARY%');
```

01. 프로시저

❖ DROP PROCEDURE 문으로 프로시저를 제거합니다.

```
DROP PROCEDURE procedure_name
```

```
drop procedure sp_salary;
```

01. 프로시저

- 프로시저는 어떤 값을 전달받아서 그 값에 의해서 서로 다른 결과물을 구하게 됩니다.
- 값을 프로시저에 전달하기 위해서 프로시저 이름 다음에 괄호로 둘러싼 부분에 전달 받을 값을 저장할 변수를 기술합니다.
- 이를 ARGUMENT 우리나라 말로 매개 변수라 합니다.
- 프로시저는 매개 변수의 값에 따라 서로 다른 동작을 수행하게 됩니다.
- [MODE] 는 IN과 OUT, INOUT 세 가지를 기술할 수 있는데 IN 데이터를 전달 받을 때 쓰고 OUT은 수행된 결과를 받아갈 때 사용합니다. INOUT은 두 가지 목적에 모두 사용됩니다.

01. 프로시저

- CREATE PROCEDURE로 프로시저를 생성할 때 MODE를 지정하여 매개변수를 선언할 수 있는데 MODE 에 IN, OUT, INOUT 세 가지를 기술할 수 있습니다.
- IN 데이터를 전달 받을 때 쓰고 OUT은 수행된 결과를 받아갈 때 사용합니다.
- INOUT은 두 가지 목적에 모두 사용됩니다.
- 이번에는 MODE를 지정하면 어떠한 기능이 부여되는지 자세히 살펴보기로 합시다.

02. 함수

- 함수는 실행환경에 매개변수를 사용하지 않고 결과 값을 되돌려 주기 위한 용도로 사용됩니다.

```
CREATE [OR REPLACE ] FUNCTION function_name
( argument1 [mode] data_taye,
  argument2 [mode] data_taye . . .
)
IS
RETURN data_type;
BEGIN
statement1;
statement2;
RETURN variable_name;
END;
```

02. 함수

- 프로시저를 만들 때에는 PROCEDURE라고 기술하지만, 함수를 만들 때에는 FUNCTION이라고 기술합니다.
- 함수는 결과를 되돌려 받기 위해서 함수가 되돌려 받게 되는 자료형과 되돌려 받을 값을 기술해야 합니다.
- 저장 함수는 호결과를 얻어오기 위해서 호출 방식에 있어서도 저장 프로서저와 차이점이 있습니다.

```
EXECUTE :varable_name := function_name(argument_list);
```

03. 트리거

❖ 다음은 트리거(trigger)의 사전적인 의미입니다.

① (총의) 방아쇠; =HAIR TRIGGER.
② 제동기, 제륜(制輪) 장치.
③ (연쇄 반응. 생리 현상. 일련의 사건 등을 유발하는) 계기,유인,자극.

❖ 오라클에서의 트리거는 어떤 이벤트가 발생했을 때 내부적으로 실행되도록 데이터베이스에 저장된 프로시저를 말합니다.

03. 트리거

- 트리거를 만들기 위한 CREATE TRIGGER 문의 형식은 다음과 같습니다.

```
CREATE TRIGGER trigger_name
timing[BEFORE|AFTER] event[INSERT|UPDATE|DELETE]
ON table_name
[FOR EACH ROW]
[WHEN conditions]
BEGIN
statement
END
```

03 트리거

❖ 트리거의 타이밍

- **[BEFORE] 타이밍은 어떤 테이블에 INSERT, UPDATE, DELETE 문이 실행될 때 해당 문장이 실행되기 전에 트리거가 가지고 있는 BEGIN ~ END 사이의 문장을 실행합니다.**

- **[AFTER] 타이밍은 INSERT, UPDATE, DELETE 문이 실행되고 난 후에 트리거가 가지고 있는 BEGIN ~ END 사이의 문장을 실행합니다.**

❖ 트리거의 이벤트

- **사용자가 어떤 DML(INSERT, UPDATE, DELETE)문을 실행했을 때 트리거를 발생시킬 것인지를 결정합니다.**

❖ 트리거의 몸체

- **해당 타이밍에 해당 이벤트가 발생하게 되면 실행될 기본 로직이 포함되는 부분으로 BEGIN ~ END에 기술합니다.**

03 트리거

❖ 트리거의 유형

- 트리거의 유형은 FOR EACH ROW에 의해 문장 레벨 트리거와 행 레벨 트리거로 나눈다.
- FOR EACH ROW가 생략되면 문장 레벨 트리거이고 행 레벨 트리거를 정의하고자 할 때에는 반드시 FOR EACH ROW를 기술해야만 합니다.
- 문장 레벨 트리거는 어떤 사용자가 트리거가 설정되어 있는 테이블에 대해 DML(INSERT, UPDATE, DELETE)문을 실행할 때 단 한번만 트리거를 발생시킬 때 사용합니다.
- 행 레벨 트리거는 DML(INSERT, UPDATE, DELETE)문에 의해서 여러 개의 행이 변경된다면 각 행이 변경될 때마다 트리거를 발생시키는 방법입니다. 만약 5개의 행이 변경되면 5번 트리거가 발생됩니다.

03 트리거

❖ 트리거 조건

- 트리거 조건은 행 레벨 트리거에서만 설정할 수 있으며 트리거 이벤트에 정의된 테이블에 이벤트가 발생할 때 보다 구체적인 데이터 검색 조건을 부여할 때 사용됩니다.