06

CHAPTER

스토어드 프로그램

1-1 스토어드 프로그램의 개요

- 스토어드 프로그램(stored program)
  - MySQL에서 제공하는 프로그래밍 언어 기능을 통틀어서 일컫는 말
  - 일반 쿼리를 묶는 역할을 할 뿐만 아니라 프로그래밍 기능도 제공함으로써 시스템의 성능이 향상됨
  - 종류에는 스토어드 프로시저, 스토어드 함수, 트리거, 커서 등이 있음

1-2 스토어드 프로시저의 개요

- 스토어드 프로시저(stored procedure, 저장 프로시저)
  - MySQL에서 제공하는 프로그래밍 기능
  - 쿼리의 집합으로서 어떠한 동작을 일괄 처리하는 데 사용
  - 자주 사용되는 일반적인 쿼리를 하나하나 실행하는 것이 아니라 모듈화하여 필요할 때마다 호출하기 때문에 MySQL을 한층 더 편리하게 운영할 수 있음

1-2 스토어드 프로시저의 개요

- 스토어드 프로시저의 정의 형식

```
CREATE
    [DEFINER = { user | CURRENT_USER }]
    PROCEDURE sp_name ([proc_parameter[, ...]])
    [characteristic ...] routine_body

proc_parameter:
    [ IN | OUT | INOUT ] param_name type

type:
    Any valid MySQL data type

characteristic:
    COMMENT 'string'
  | LANGUAGE SQL
  | [NOT] DETERMINISTIC
  | { CONTAINS SQL | NO SQL | READS SQL DATA | MODIFIES SQL DATA }
  | SQL SECURITY { DEFINER | INVOKER }

routine_body:
    Valid SQL routine statement
```

1-2 스토어드 프로시저의 개요

- 간단하게 정리한 형식

```
DELIMITER $$
CREATE PROCEDURE 스토어드프로시저이름(IN 또는 OUT 매개변수)
BEGIN

    이 부분에 SQL 프로그래밍 코딩

END $$
DELIMITER ;
CALL 스토어드프로시저이름();
```

- 스토어드 프로시저의 생성 예

```
1 USE cookDB;
2 DROP PROCEDURE IF EXISTS userProc;
3 DELIMITER $$
4 CREATE PROCEDURE userProc()
5 BEGIN
6 SELECT * FROM userTBL; -- 스토어드 프로시저 내용
7 END $$
8 DELIMITER ;
9
10 CALL userProc();
```

  - 2행 : 만들어진 프로시저가 있다면 삭제
  - 3~7행: DELIMITER $$ ... END $$ 문으로 스토어드 프로시저를 묶음
  - 8행: DELIMITER ; 문으로 종료 문자를 세미콜론으로 변경
  - 10행: 생성한 스토어드 프로시저를 호출

1-3 매개변수와 오류 처리

- 입력 매개변수를 지정하는 형식

```
IN 입력매개변수이름 데이터형식
```

- 입력 매개변수가 있는 스토어드 프로시저를 실행하는 형식

```
CALL 프로시저이름(전달값);
```

- 출력 매개변수를 지정하는 형식

```
OUT 출력매개변수이름 데이터형식
```

- 출력 매개변수가 있는 스토어드 프로시저를 실행하는 형식

```
CALL 프로시저이름(@변수명);
SELECT @변수명;
```

[실습 6-1] 스토어드 프로시저 생성하고 활용하기

1 cookDB 초기화하기

1-1 cookDB.sql 파일을 열어 실행

1-2 열린 쿼리 창을 모두 닫고 새 쿼리 창을 염

2 입력 매개변수가 있는 스토어드 프로시저 실행하디

2-1 1개의 입력 매개변수가 있는 스토어드 프로시저 생성

```
1 USE cookDB;
2 DROP PROCEDURE IF EXISTS userProc1;
3 DELIMITER $$
4 CREATE PROCEDURE userProc1(IN uName VARCHAR(10))
5 BEGIN
6     SELECT * FROM userTBL WHERE userName = uName;
7 END $$
8 DELIMITER ;
9
10 CALL userProc1('이경규');
```

	userID	userName	birthYear	addr	mobile1	mobile2	height	mDate
	LKK	이경규	1960	경남	018	99999999	170	2004-12-12

[실습 6-1] 스토어드 프로시저 생성하고 활용하기

2-2 2개의 입력 매개변수가 있는 스토어드 프로시저 생성

```
1 DROP PROCEDURE IF EXISTS userProc2;
2 DELIMITER $$
3 CREATE PROCEDURE userProc2(
4       IN userBirth INT,
5       IN userHeight INT
6 )
7 BEGIN
8    SELECT * FROM userTBL
9       WHERE birthYear > userBirth AND height > userHeight;
10 END $$
11 DELIMITER;
12
13 CALL userProc2(1970, 178);
```

	userID	userName	birthYear	addr	mobile1	mobile2	height	mDate
▶	LHJ	이휘재	1972	경기	011	88888888	180	2006-04-04
	NHS	남희석	1971	충남	016	66666666	180	2017-04-04

[실습 6-1] 스토어드 프로시저 생성하고 활용하기

3 출력 매개변수가 있는 스토어드 프로시저 실행하기

3-1 출력 매개변수가 있는 스토어드 프로시저를 생성한 후 테스트로 사용할 테이블을 만듦

```
1 DROP PROCEDURE IF EXISTS userProc3;
2 DELIMITER $$
3 CREATE PROCEDURE userProc3(
4     IN txtValue CHAR(10),
5     OUT outValue INT
6 )
7 BEGIN
8     INSERT INTO testTBL VALUES(NULL, txtValue);
9     SELECT MAX(id) INTO outValue FROM testTBL;
10 END $$
11 DELIMITER ;
12
13 CREATE TABLE IF NOT EXISTS testTBL(
14     id INT AUTO_INCREMENT PRIMARY KEY,
15     txt CHAR(10)
16 );
```

3-2 출력 매개변수가 있는 스토어드 프로시저 호출

```
1 CALL userProc3 ('테스트값', @myValue);
2 SELECT CONCAT('현재 입력된 ID 값 ==>', @myValue);
```

실행 결과

```
현재 입력된 ID 값 ==> 1
```

[실습 6-1] 스토어드 프로시저 생성하고 활용하기

4 스토어드 프로시저 안에 SQL 프로그래밍하기

4-1 스토어드 프로시저 안에 IF ... ELSE 문 작성

```
1 DROP PROCEDURE IF EXISTS ifelseProc;
2 DELIMITER $$
3 CREATE PROCEDURE ifelseProc(
4       IN uName VARCHAR(10)
5 )
6 BEGIN
7       DECLARE bYear INT; -- 변수 선언
8       SELECT birthYear into bYear FROM userTBL
9           WHERE userName = uName;
10      IF (bYear >= 1970) THEN
11              SELECT '아직 젊군요..';
12      ELSE
13              SELECT '나이가 지긋하네요..';
14      END IF;
15 END $$
16 DELIMITER ;
17
18 CALL ifelseProc ('김국진');
```

[실습 6-1] 스토어드 프로시저 생성하고 활용하기

4-3 CASE 문을 활용하여 호출한 사람의 띠를 알려주는 스토어드 프로시저 만들기

```
DROP PROCEDURE IF EXISTS caseProc;
DELIMITER $$
CREATE PROCEDURE caseProc(
    IN uName VARCHAR(10)
)
BEGIN
    DECLARE bYear INT;
    DECLARE tti CHAR(3); -- 띠를 저장할 변수
    SELECT birthYear INTO bYear FROM userTBL
        WHERE userName = uName;
    CASE
        WHEN (bYear%12 = 0) THEN SET tti = '원숭이';
        WHEN (bYear%12 = 1) THEN SET tti = '닭';
        WHEN (bYear%12 = 2) THEN SET tti = '개';
        WHEN (bYear%12 = 3) THEN SET tti = '돼지';
        WHEN (bYear%12 = 4) THEN SET tti = '쥐';
        WHEN (bYear%12 = 5) THEN SET tti = '소';
        WHEN (bYear%12 = 6) THEN SET tti = '호랑이';
        WHEN (bYear%12 = 7) THEN SET tti = '토끼';
        WHEN (bYear%12 = 8) THEN SET tti = '용';
        WHEN (bYear%12 = 9) THEN SET tti = '뱀';
        WHEN (bYear%12 = 10) THEN SET tti = '말';
        ELSE SET tti = '양';
    END CASE;
    SELECT CONCAT(uName, '의 띠 ==>', tti);
END $$
DELIMITER ;

CALL caseProc ('박수홍');
```

[실습 6-1] 스토어드 프로시저 생성하고 활용하기

4-4 WHILE 문을 활용하여 구구단을 문자열로 생성하고 테이블에 입력하는 스토어드 프로시저를 만들기

```
1 DROP TABLE IF EXISTS guguTBL;
2 CREATE TABLE guguTBL (txt VARCHAR(100)); -- 구구단 저장용 테이블
3
4 DROP PROCEDURE IF EXISTS whileProc;
5 DELIMITER $$
6 CREATE PROCEDURE whileProc()
7 BEGIN
8 DECLARE str VARCHAR(100); -- 각 단을 문자열로 저장
9 DECLARE i INT; -- 구구단 앞자리
10 DECLARE k INT; -- 구구단 뒷자리
11 SET i = 2; -- 2단부터 계산
12
13 WHILE (i < 10) DO -- 바깥 반복문(2~9단 반복)
14 SET str = ''; -- 각 단의 결과를 저장할 문자열 초기화
15 SET k = 1; -- 구구단 뒷자리는 항상 1부터 9까지
16 WHILE (k < 10) DO
17 SET str = CONCAT(str, ' ', i, 'x', k, '=', i * k); -- 문자열 만들기
18 SET k = k + 1; -- 뒷자리 증가
19 END WHILE;
20 SET i = i + 1; -- 앞자리 증가
21 INSERT INTO guguTBL VALUES(str); -- 각 단의 결과를 테이블에 입력
22 END WHILE;
23 END $$
24 DELIMITER ;
25
26 CALL whileProc();
27 SELECT * FROM guguTBL;
```

[실습 6-1] 스토어드 프로시저 생성하고 활용하기

4-4 WHILE 문을 활용하여 구구단을 문자열로 생성하고 테이블에 입력하는 스토어드 프로시저를 만들기

txt
2x1=2 2x2=4 2x3=6 2x4=8 2x5=10 2x6=12 2x7=14 2x8=16 2x9=18
3x1=3 3x2=6 3x3=9 3x4=12 3x5=15 3x6=18 3x7=21 3x8=24 3x9=27
4x1=4 4x2=8 4x3=12 4x4=16 4x5=20 4x6=24 4x7=28 4x8=32 4x9=36
5x1=5 5x2=10 5x3=15 5x4=20 5x5=25 5x6=30 5x7=35 5x8=40 5x9=45
6x1=6 6x2=12 6x3=18 6x4=24 6x5=30 6x6=36 6x7=42 6x8=48 6x9=54
7x1=7 7x2=14 7x3=21 7x4=28 7x5=35 7x6=42 7x7=49 7x8=56 7x9=63
8x1=8 8x2=16 8x3=24 8x4=32 8x5=40 8x6=48 8x7=56 8x8=64 8x9=72
9x1=9 9x2=18 9x3=27 9x4=36 9x5=45 9x6=54 9x7=63 9x8=72 9x9=81

[실습 6-1] 스토어드 프로시저 생성하고 활용하기

5 스토어드 프로시저 오류 처리하기

5-1 DECLARE ... HANDLER 문을 이용하여 스토어드 프로시저에 발생한 오류 처리

```
DROP PROCEDURE IF EXISTS errorProc;
DELIMITER $$
CREATE PROCEDURE errorProc()
BEGIN
	DECLARE i INT; -- 1씩 증가하는 값
	DECLARE hap INT; -- 합계(정수형), 오버플로를 발생시킬 예정
	DECLARE saveHap INT; -- 합계(정수형), 오버플로가 발생하기 직전의 값 저장

	DECLARE EXIT HANDLER FOR 1264 -- 정수형 오버플로가 발생하면 이 부분 수행
		BEGIN
		SELECT CONCAT('INT 오버플로 직전의 합계 --> ', saveHap);
		SELECT CONCAT('1+2+3+4+...+', i, '=오버플로');
		END;
	SET i = 1; -- 1부터 증가
	SET hap = 0; -- 합계 누적

	WHILE (TRUE) DO -- 무한 루프
		SET saveHap = hap; -- 오버플로 직전의 합계 저장
		SET hap = hap + i; -- 오버플로가 발생하면 11행과 12행 수행
		SET i = i + 1;
	END WHILE;
END $$
DELIMITER ;

CALL errorProc();
```

[실습 6-1] 스토어드 프로시저 생성하고 활용하기

6-1 동적 SQL을 활용하여 테이블 이름을 입력 매개변수로 전달

```
DROP PROCEDURE IF EXISTS nameProc;
DELIMITER $$
CREATE PROCEDURE nameProc(
IN tableName VARCHAR(20)
)
BEGIN
        SET @sqlQuery = CONCAT('SELECT * FROM ', tableName);
        PREPARE myQuery FROM @sqlQuery;
        EXECUTE myQuery;
        DEALLOCATE PREPARE myQuery;
END $$
DELIMITER ;

CALL nameProc('userTBL');
select @sqlquery;
```

1-4 스토어드 프로시저의 장점

- MySQL의 성능을 향상할 수 있다
  - 스토어드 프로시저를 사용하면 네트워크의 부하를 크게 줄여 결과적으로 MySQL의 성능을 향상할 수 있음

- 유지 관리가 간편하다
  - 스토어드 프로시저 이름만 호출하도록 설정하면 데이터베이스에서 관련된 스토어드 프로시저의 내용을 일관되게 수정하거나 유지·보수할 수 있음

- 모듈식 프로그래밍이 가능하다
  - 한 번 스토어드 프로시저를 생성해놓으면 언제든지 실행할 수 있음
  - 스토어드 프로시저로 저장한 쿼리의 수정, 삭제와 같은 관리도 수월

1-4 스토어드 프로시저의 장점

- 보안을 강화할 수 있다
  - 사용자별로 테이블 접근 권한을 주지 않고 스토어드 프로시저에만 접근 권한을 주면 보안을 강화할수 있음
  - 예를 들어 cookDB의 회원 테이블(userTBL)에는 회원의 이름, 전화번호, 주소, 출생 연도, 키 등의 개인 정보가 들어 있음
  - 이럴 때 다음과 같은 프로시저를 생성한 후 배송 담당자에게 회원 테이블 대신 스토어드 프로시저의 접근 권한을 주면 보안 문제가 해결

```
DELIMITER $$
CREATE PROCEDURE delivProc(
    IN id VARCHAR(10)
)
BEGIN
    SELECT userID, userName, addr, mobile1, mobile2
       FROM userTBL
        WHERE userID = id;
END $$
DELIMITER ;
```

```
CALL delivProc ('LHJ');
```

2-1 스토어드 함수의 개요

- 스토어드 함수를 생성하려면
  - 시스템 변수인 log_bin_trust_function_creators를 ON으로 변경

- 두 수의 합계를 계산하는 스토어드 함수

```
SET GLOBAL log_bin_trust_function_creators = 1;
```

```
USE cookDB;
DROP FUNCTION IF EXISTS userFunc;
DELIMITER $$
CREATE FUNCTION userFunc(value1 INT, value2 INT)
RETURNS INT
BEGIN
        RETURN value1 + value2;
END $$
DELIMITER ;

SELECT userFunc(100, 200);
```

  - 4행: 정수형 매개변수 2개를 전달받음
  - 5행: 이 함수가 반환하는 데이터 형식을 지정
  - 7행: RETURN 문으로 정수형을 반환
  - 11행: SELECT 문에서 함수를 호출하면서 2개의 매개변수를 전달

[실습 6-2] 스토어드 함수 사용하기

1 cookDB 초기화하기

1-1 cookDB.sql 파일을 열어 실행

1-2 열린 쿼리 창을 모두 닫고 새 쿼리 창을 염

2 스토어드 함수 만들고 활용하기

2-1 출생 연도를 입력하면 나이가 출력되는 함수 생성

```
1 USE cookDB;
2 DROP FUNCTION IF EXISTS getAgeFunc;
3 DELIMITER $$
4 CREATE FUNCTION getAgeFunc(bYear INT)
5       RETURNS INT
6 BEGIN
7       DECLARE age INT;
8       SET age = YEAR(CURDATE()) - bYear;
9       RETURN age;
10 END $$
11 DELIMITER ;
```

2-2 앞에서 작성한 함수를 SELETE 문에서 호출

```
SELECT getAgeFunc(1979);
```

[실습 6-2] 스토어드 함수 사용하기

2-3 두 출생 연도의 나이 차 출력

```
SELECT getAgeFunc(1979) INTO @age1979;
SELECT getAgeFunc(1997) INTO @age1997;
SELECT CONCAT('1997년과 1979년의 나이차 ==> ', (@age1979-@age1997));
```

2-4 작성한 함수를 테이블을 검색하는 쿼리에 활용

```
SELECT userID, userName, getAgeFunc(birthYear) AS '만 나이' FROM userTBL;
```

	userID	userName	만 나이
▶	KHD	강호동	48
	KJD	김제동	44
	KKJ	김국진	53
	KYM	김용만	51
	LHJ	이휘재	46
	LKK	이경규	58
	NHS	남희석	47
	PSH	박수홍	48
	SDY	신동엽	47
	YJS	유재석	46

[실습 6-2] 스토어드 함수 사용하기

2-5 현재 저장된 스토어드 함수의 이름과 내용 확인

```
SHOW CREATE FUNCTION getAgeFunc;
```

2-6 스토어드 함수 삭제

```
DROP FUNCTION getAgeFunc;
```

3-1 커서의 개요

- 커서(cursor)
  - 일반 프로그래밍 언어로 파일 처리를 하는 것과 비슷한 방식으로 테이블의 행 집합을 다룰 수 있음
  - 쿼리의 결과 행 집합에서 한 행씩 옮겨가며 명령을 처리

그림 11-8 파일 처리의 동작

3-1 커서의 개요

- 커서의 작동 순서

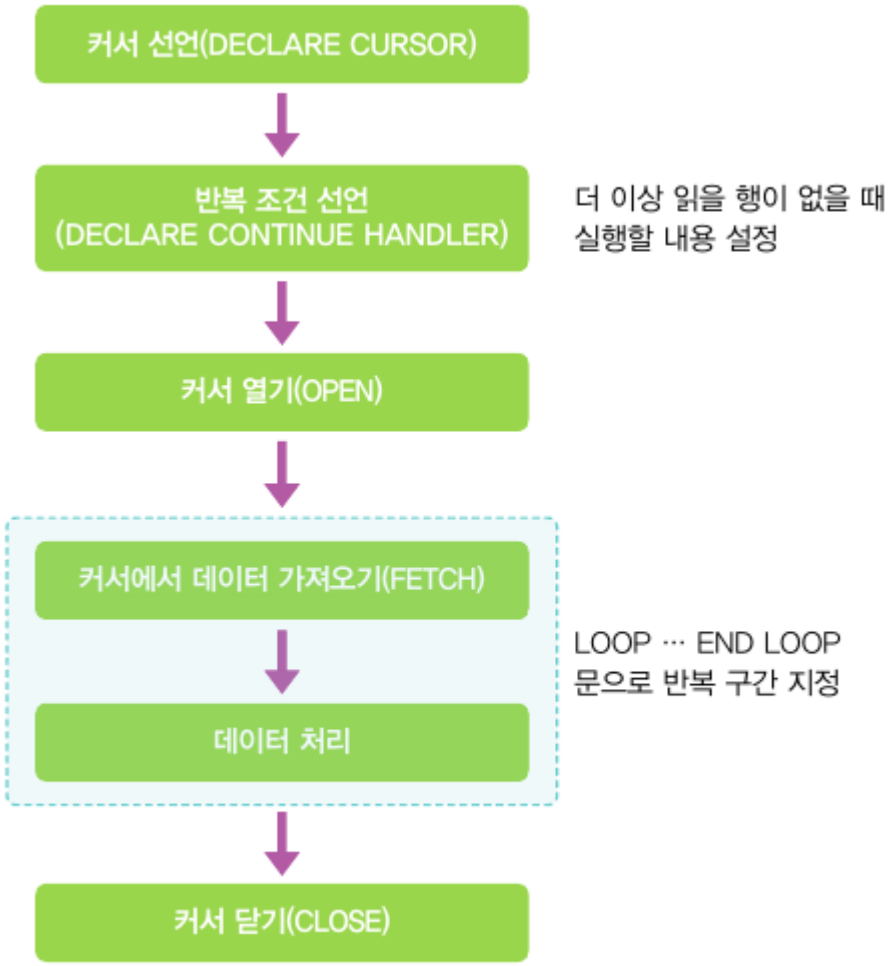


그림 11-9 커서의 작동 순서

[실습 6-3] 커서 활용하기

1 고객의 평균 키를 구하는 스토어드 프로시저 작성하기

1-1 고객의 평균 키를 구하는 스토어드 프로시저 작성

```
1 DROP PROCEDURE IF EXISTS cursorProc;
2 DELIMITER $$
3 CREATE PROCEDURE cursorProc()
4 BEGIN
5       DECLARE userHeight INT; -- 고객의 키
6       DECLARE cnt INT DEFAULT 0; -- 고객의 인원수(읽은 행의 수)
7       DECLARE totalHeight INT DEFAULT 0; -- 키의 합계
8
9       DECLARE endOfRow BOOLEAN DEFAULT FALSE; -- 행의 끝 여부(기본은 FALSE)
10
11     DECLARE userCuror CURSOR FOR -- 커서 선언
12          SELECT height FROM userTBL;
13
14     DECLARE CONTINUE HANDLER -- 행의 끝이면 endOfRow 변수에 TRUE 대입
15          FOR NOT FOUND SET endOfRow = TRUE;
16
17      OPEN userCuror; -- 커서 열기
18
19      cursor_loop: LOOP
20          FETCH userCuror INTO userHeight; -- 고객의 키 1개 대입
21
22          IF endOfRow THEN -- 더 이상 읽을 행이 없으면 LOOP 종료
23              LEAVE cursor_loop;
```

[실습 6-3] 커서 활용하기

1-1 고객의 평균 키를 구하는 스토어드 프로시저 작성

```
24          END IF;
25
26          SET cnt = cnt + 1;
27          SET totalHeight = totalHeight + userHeight;
28      END LOOP cursor_loop;
29
30      -- 고객의 평균 키 출력
31      SELECT CONCAT('고객 키의 평균 ==> ', (totalHeight/cnt));
32
33      CLOSE userCuror; -- 커서 닫기
34  END $$
35  DELIMITER ;
```

- 5~7행: 고객의 평균 키를 계산하기 위해 변수를 선언
- 9행, 14~15행: LOOP 부분을 종료하기 위한 조건을 지정, 만약 행의 끝이라면 endOfRow 변수에 TRUE가 대입되어 22~24행이 수행되고 LOOP가 종료됨
- 17행: 준비한 커서를 염
- 19~28행: 행의 끝까지 반복하면서 고객의 키를 하나씩 totalHeight 변수에 누적함. 또한 26행에서 고객의 수를 셈
- 31행: LOOP를 빠져나와 평균 키(키의 합계/고객의 수)를 출력
- 33행: 커서를 닫음

[실습 6-3] 커서 활용하기

1-2 스토어드 프로시저 호출

```
CALL cursorProc();
```

실행 결과

```
고객 키의 평균==> 177.0000
```

2 고객 등급을 분류하는 스토어드 프로시저 작성하기

2-1 회원 테이블(userTBL)에 고객 등급(grade) 열 추가

```
USE cookDB;
ALTER TABLE userTBL ADD grade VARCHAR(5); -- 고객 등급 열 추가
```

[실습 6-3] 커서 활용하기

2-2 회원 테이블의 고객 등급 열에 최우수 고객, 우수고객, 일반고객, 유령고객 등의 값을 입력하는 스토어드 프로시저를 작성

```
1 DROP PROCEDURE IF EXISTS gradeProc;
2 DELIMITER $$
3 CREATE PROCEDURE gradeProc()
4 BEGIN
5     DECLARE id VARCHAR(10); -- 사용자 아이디를 저장할 변수
6     DECLARE hap BIGINT; -- 총구매액을 저장할 변수
7     DECLARE userGrade CHAR(5); -- 고객 등급 변수
8
9     DECLARE endOfRow BOOLEAN DEFAULT FALSE;
10
11   DECLARE userCuror CURSOR FOR -- 커서 선언
12         SELECT U.userid, sum(price * amount)
13             FROM buyTBL B
14                RIGHT OUTER JOIN userTBL U
15                ON B.userid = U.userid
16             GROUP BY U.userid, U.userName;
17
18   DECLARE CONTINUE HANDLER
19         FOR NOT FOUND SET endOfRow = TRUE;
20
21    OPEN userCuror; -- 커서 열기
22    grade_loop: LOOP
23         FETCH userCuror INTO id, hap; -- 첫 행 값 대입
24         IF endOfRow THEN
25             LEAVE grade_loop;
```

[실습 6-3] 커서 활용하기

2-2 회원 테이블의 고객 등급 열에 최우수 고객, 우수고객, 일반고객, 유령고객 등의 값을 입력하는 스토어드 프로시저를 작성

```
26        END IF;
27
28        CASE
29            WHEN (hap >= 1500) THEN SET userGrade = '최우수고객';
30            WHEN (hap >= 1000) THEN SET userGrade ='우수고객';
31            WHEN (hap >= 1) THEN SET userGrade ='일반고객';
32            ELSE SET userGrade ='유령고객';
33        END CASE;
34
35        UPDATE userTBL SET grade = userGrade WHERE userID = id;
36    END LOOP grade_loop;
37
38    CLOSE userCuror; -- 커서 닫기
39 END $$
40 DELIMITER ;
```

- 5~7행: 사용할 변수를 정의
- 11~16행: 커서를 정의하는데, 결과로 사용자 아이디와 사용자별 총구매액이 나옴
- 22~36행: LOOP를 반복하면서 한 행씩 처리
- 28~33행: 총구매액(hap)에 따라 고객의 등급을 분류
- 35행: 분류한 고객의 등급(grade)을 업데이트

[실습 6-3] 커서 활용하기

2-3 고객 등급이 완성되었는지 확인

```
CALL gradeProc();
SELECT * FROM userTBL;
```

userID	userName	birthYear	addr	mobile1	mobile2	height	mDate	grade
KHD	강호동	1970	경북	011	22222222	182	2007-07-07	우수고객
KJD	김제동	1974	경남	NULL	NULL	173	2013-03-03	일반고객
KKJ	김국진	1965	서울	019	33333333	171	2009-09-09	유령고객
KYM	김용만	1967	서울	010	44444444	177	2015-05-05	일반고객
LHJ	이휘재	1972	경기	011	88888888	180	2006-04-04	일반고객
LKK	이경규	1960	경남	018	99999999	170	2004-12-12	유령고객
NHS	남희석	1971	충남	016	66666666	180	2017-04-04	유령고객
PSH	박수홍	1970	서울	010	00000000	183	2012-05-05	최우수고객
SDY	신동엽	1971	경기	NULL	NULL	176	2008-10-10	유령고객
YJS	유재석	1972	서울	010	11111111	178	2008-08-08	유령고객

4-1 트리거

- 트리거(trigger)
  - 테이블에 부착되는(attach) 프로그램 코드
  - 해당 테이블에 데이터 삽입, 수정, 삭제 작업이 발생하면 자동으로 실행
  - 트리거는 제약 조건과 더불어 데이터의 무결성을 보장하는 장치의 역할을 함

[실습 6-4] 트리거 생성하고 작동 확인하기

1 cookDB 초기화하기

1-1 cookDB.sql 파일을 열어 실행

1-2 열린 쿼리 창을 모두 닫고 새 쿼리 창을 염

2 트리거 생성하고 작동 확인하기

2-1 cookDB에 간단한 테이블 생성

```
USE cookDB;
CREATE TABLE IF NOT EXISTS testTBL (id INT, txt VARCHAR(10));
INSERT INTO testTBL VALUES (1, '이엑스아이디');
INSERT INTO testTBL VALUES (2, '블랙핑크');
INSERT INTO testTBL VALUES (3, '에이핑크');
```

2-2 로 만든 테이블(testTBL)에 트리거 부착

```
DROP TRIGGER IF EXISTS testTrg;
DELIMITER //
CREATE TRIGGER testTrg -- 트리거 이름
    AFTER DELETE -- 삭제 후에 작동하도록 지정
    ON testTBL -- 트리거를 부착할 테이블
    FOR EACH ROW -- 각 행마다 적용
BEGIN
    SET @msg = '가수 그룹이 삭제됨'; -- 트리거 실행 시 작동하는 코드
END //
DELIMITER ;
```

[실습 6-4] 트리거 생성하고 작동 확인하기

2-3 데이터 변경(삽입, 수정, 삭제) 작업 수행

```
SET @msg = '';
INSERT INTO testTBL VALUES (4, '여자친구');
SELECT @msg;
UPDATE testTBL SET txt = '레드벨벳' WHERE id = 3;
SELECT @msg;
DELETE FROM testTBL WHERE id = 4;
SELECT @msg;
```

	@msg

	@msg

	@msg
	가수 그룹이 삭제됨

4-2 트리거의 종류

- AFTER 트리거
  - 테이블에 변경(삽입, 수정, 삭제) 작업이 일어났을 때 작동하는 트리거
  - 변경 작업이 일어난 후(after) 실행

- BEFORE 트리거
  - 테이블에 변경(삽입, 수정, 삭제) 작업이 일어나기 전(before)에 실행

- 트리거 생성 형식

```
CREATE
    [DEFINER = { user | CURRENT_USER }]
    TRIGGER trigger_name
    trigger_time trigger_event
    ON TBL_name FOR EACH ROW
    [trigger_order]
    trigger_body

trigger_time: { BEFORE | AFTER }

trigger_event: { INSERT | UPDATE | DELETE }

trigger_order: { FOLLOWS | PRECEDES } other_trigger_name
```

[실습 6-5] AFTER 트리거 생성하고 작동 확인하기

1 백업 테이블 생성하기

1-1 데이터를 저장할 백업 테이블 생성

```
USE cookDB;
DROP TABLE buyTBL; -- 구매 테이블은 실습에 필요 없으므로 삭제
CREATE TABLE backup_userTBL
( userID char(8) NOT NULL,
  userName varchar(10) NOT NULL,
  birthYear int NOT NULL,
  addr char(2) NOT NULL,
  mobile1 char(3),
  mobile2 char(8),
  height smallint,
  mDate date,
  modType char(2), -- 변경된 유형('수정' 또는 '삭제')
  modDate date, -- 변경된 날짜
  modUser varchar(256) -- 변경한 사용자
);
```

[실습 6-5] AFTER 트리거 생성하고 작동 확인하기

2 트리거 생성하고 작동 확인하기

2-1 데이터를 수정했을 때 작동하는 트리거 생성

```
1 DROP TRIGGER IF EXISTS backUserTBL_UpdateTrg;
2 DELIMITER //
3 CREATE TRIGGER backUserTBL_UpdateTrg -- 트리거 이름
4       AFTER UPDATE -- 변경 후 작동하도록 지정
5       ON userTBL -- 트리거를 부착할 테이블
6       FOR EACH ROW
7 BEGIN
8     INSERT INTO backup_userTBL VALUES (OLD.userID, OLD.userName, OLD.birthYear,
9         OLD.addr, OLD.mobile1, OLD.mobile2, OLD.height, OLD.mDate,
10        '수정', CURDATE(), CURRENT_USER());
11 END //
12 DELIMITER ;
```

[실습 6-5] AFTER 트리거 생성하고 작동 확인하기

2-2 데이터를 삭제했을 때 작동하는 트리거 생성

```
DROP TRIGGER IF EXISTS backUserTBL_DeleteTrg;
DELIMITER //
CREATE TRIGGER backUserTBL_DeleteTrg -- 트리거 이름
AFTER DELETE -- 삭제 후 작동하도록 지정
ON userTBL -- 트리거를 부착할 테이블
FOR EACH ROW
BEGIN
INSERT INTO backup_userTBL VALUES(OLD.userID, OLD.userName, OLD.birthYear,
OLD.addr, OLD.mobile1, OLD.mobile2, OLD.height, OLD.mDate,
' 삭제 ', CURDATE(), CURRENT_USER());
END //
DELIMITER ;
```

2-3 데이터를 수정하고 삭제

```
UPDATE userTBL SET addr = '제주' WHERE userID = 'KJD';
DELETE FROM userTBL WHERE height >= 180;
```

[실습 6-5] AFTER 트리거 생성하고 작동 확인하기

2-4 방금 수정하고 삭제한 내용이 백업 테이블에 잘 보관되었는지 확인

```
SELECT * FROM backup_userTBL;
```

userID	userName	birthYear	addr	mobile1	mobile2	height	mDate	modType	modDate	modUser
KJD	김제동	1974	경남	NULL	NULL	173	2013-03-03	수정	2018-09-08	root@localhost
KHD	강호동	1970	경북	011	22222222	182	2007-07-07	삭제	2018-09-08	root@localhost
LHJ	이휘재	1972	경기	011	88888888	180	2006-04-04	삭제	2018-09-08	root@localhost
NHS	남희석	1971	충남	016	66666666	180	2017-04-04	삭제	2018-09-08	root@localhost
PSH	박수홍	1970	서울	010	00000000	183	2012-05-05	삭제	2018-09-08	root@localhost

3 테이블의 모든 행 삭제하기

3-1 DELETE 문 대신 TRUNCATE TABLE 문 사용

```
TRUNCATE TABLE userTBL;
```

3-2 백업 테이블 확인

```
SELECT * FROM backup_userTBL;
```

userID	userName	birthYear	addr	mobile1	mobile2	height	mDate	modType	modDate	modUser
KJD	김제동	1974	경남	NULL	NULL	173	2013-03-03	수정	2018-09-08	root@localhost
KHD	강호동	1970	경북	011	22222222	182	2007-07-07	삭제	2018-09-08	root@localhost
LHJ	이휘재	1972	경기	011	88888888	180	2006-04-04	삭제	2018-09-08	root@localhost
NHS	남희석	1971	충남	016	66666666	180	2017-04-04	삭제	2018-09-08	root@localhost
PSH	박수홍	1970	서울	010	00000000	183	2012-05-05	삭제	2018-09-08	root@localhost

[실습 6-5] AFTER 트리거 생성하고 작동 확인하기

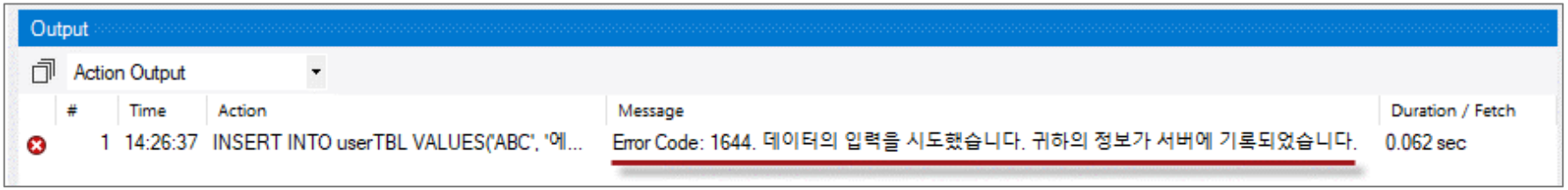
4 경고 메시지 보내기

4-1 INSERT 트리거 생성

```
1 DROP TRIGGER IF EXISTS userTBL_InsertTrg;
2 DELIMITER //
3 CREATE TRIGGER userTBL_InsertTrg -- 트리거 이름
4       AFTER INSERT -- 데이터 삽입 후 작동하도록 설정
5       ON userTBL -- 트리거를 부착할 테이블
6       FOR EACH ROW
7 BEGIN
8       SIGNAL SQLSTATE '45000'
9          SET MESSAGE_TEXT = '데이터의 입력을 시도했습니다. 귀하의 정보가 서버에 기록되었습니다.';
10 END //
11 DELIMITER ;
```

4-2 새로운 데이터 삽입

```
INSERT INTO userTBL VALUES ('ABC', '에비씨', 1977, '서울', '011', '1111111', 181, '2019-12-25');
```



4-2 트리거의 종류

- NEW 테이블과 OLD 테이블
  - NEW 테이블의 경우 대상 테이블에 삽입 또는 수정 작업이 발생했을 때 변경될 새 데이터가 잠깐 저장
  - OLD 테이블의 경우 삭제 또는 수정 명령이 수행될 때 삭제 또는 수정되기 전의 예전 값이 저장
  - 트리거가 작동할 때 삽입되거나 수정되는 새 데이터를 참조하려면 NEW 테이블을 확인하고, 변경되기 전의 예전 데이터를 참조하려면 OLD 테이블을 확인

그림 11-15 NEW와 OLD 테이블의 작동

- BEFORE 트리거
  - 테이블에 변경이 가해지기 전에 작동하는 트리거

[실습 6-6] BEFORE INSERT 트리거 생성하기

1 BEFORE INSERT 트리거 생성하기

1-1 BEFORE INSERT 트리거 작성

```
1 USE cookDB;
2 DROP TRIGGER IF EXISTS userTBL_InsertTrg; -- 앞에서 실습한 트리거 제거
3 DROP TRIGGER IF EXISTS userTBL_BeforeInsertTrg;
4 DELIMITER //
5 CREATE TRIGGER userTBL_BeforeInsertTrg -- 트리거 이름
6       BEFORE INSERT -- 데이터를 삽입하기 전 작동하도록 지정
7       ON userTBL -- 트리거를 부착할 테이블
8       FOR EACH ROW
9 BEGIN
10      IF NEW.birthYear < 1900 THEN
11          SET NEW.birthYear = 0;
12      ELSEIF NEW.birthYear > YEAR(CURDATE()) THEN
13          SET NEW.birthYear = YEAR(CURDATE());
14      END IF;
15 END //
16 DELIMITER ;
```

- 6행: 데이터를 삽입하기 전에 처리되는 트리거를 생성
- 10~14행: 새로 삽입되는 값이 들어 있는 NEW 테이블을 검사하여 1900 미만이면 0으로, 올해 연도를 초과하면 올해 연도로 변경

[실습 6-6] BEFORE INSERT 트리거 생성하기

1-2 출생 연도에 문제가 있는 데이터 2개를 회원 테이블에 삽입

```
INSERT INTO userTBL VALUES
   ('AAA', '에이', 1877, '서울', '011', '11112222', 181, '2019-12-25');
INSERT INTO userTBL VALUES
   ('BBB', '비이', 2977, '경기', '011', '11113333', 171, '2011-3-25');
```

1-3 SELECT * FROM userTBL; 문으로 확인

	userID	userName	birthYear	addr	mobile1	mobile2	height	mDate
▶	AAA	에이	0	서울	011	11112222	181	2019-12-25
	BBB	비이	2019	경기	011	11113333	171	2011-03-25
*	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

2 생성된 트리거 확인하기

2-1 cookDB에 생성된 트리거 확인

```
SHOW TRIGGERS FROM cookDB;
```

	Trigger	Event	Table	Statement	Timing	Created	sql_mode
	testTrg	DELETE	testtbl	-- 각 행마다 적용시킴 BEGIN SET @msg = '가수...	AFTER	2018-09-08 14:37:12.93	STRICT_TRANS_TABLES,NO_EN
	userTBL_BeforeInsertTrg	INSERT	usertbl	BEGIN IF NEW.birthYear < 1900 THEN S...	BEFORE	2018-09-08 14:41:19.74	STRICT_TRANS_TABLES,NO_EN
	backUserTBL_UpdateTrg	UPDATE	usertbl	BEGIN INSERT INTO backup_userTBL VALUES...	AFTER	2018-09-08 14:37:58.18	STRICT_TRANS_TABLES,NO_EN
	backUserTBL_DeleteTrg	DELETE	usertbl	BEGIN INSERT INTO backup_userTBL VALUES...	AFTER	2018-09-08 14:38:05.84	STRICT_TRANS_TABLES,NO_EN

2-2 트리거를 삭제할 때는 DROP TRIGGER 트리거이름; 문 사용

4-3 다중 트리거와 중첩 트리거

- 다중 트리거(multiple trigger)
  - 하나의 테이블에 트리거가 여러 개 부착되어 있는 것

- 중첩 트리거(nested trigger)
  - 트리거가 또 다른 트리거를 작동시키는 것

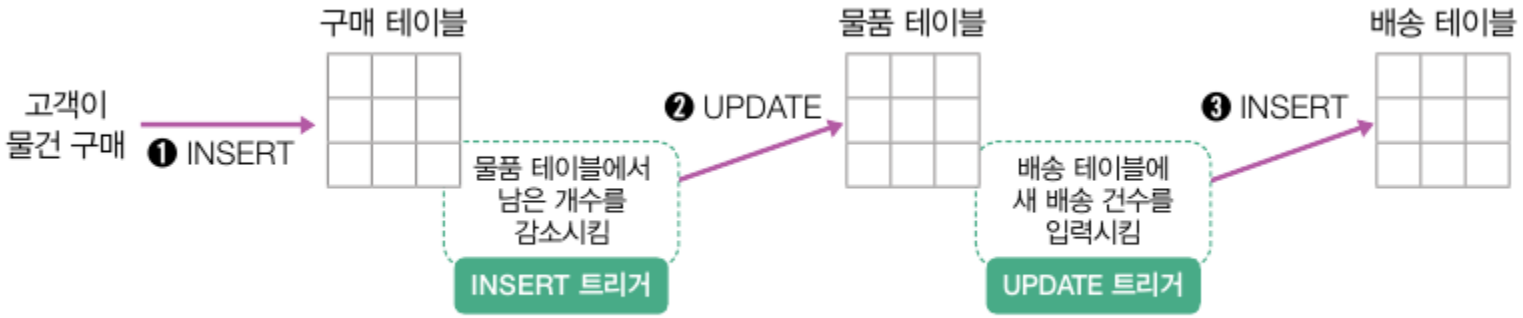


그림 11-18 중첩 트리거의 예

① 고객이 물건을 구매하면 그 구매 기록이 구매 테이블에 ❶ 삽입(INSERT)된다.

② 구매 테이블에 부착된 INSERT 트리거가 작동하면서 물품 테이블의 남은 개수에서 구매한 개수를 빼도록 ❷ 수정(UPDATE)한다(인터넷 쇼핑몰에서 물건을 구매하면 그 즉시 남은 수량이 줄어드는 것을 보았을 것이다).

③ 물품 테이블에 장착된 UPDATE 트리거가 작동하면서 배송 테이블에 배송할 내용을 ❸ 삽입 (INSERT)한다.

4-3 다중 트리거와 중첩 트리거

- 트리거의 작동 순서 지정

```
{ FOLLOWS | PRECEDES } other_trigger_name
```

  - 'FOLLOWS 트리거이름'으로 지정하면 지정한 트리거 다음에 현재 트리거가 작동
  - ' PRECEDES 트리거이름'으로 지정하면 지정한 트리거가 작동하기 이전에 현재 트리거가 작동

[실습 6-7] 중첩 트리거 생성하고 작동 확인하기

1 새 데이터베이스 만들기

1-1 새 쿼리 창을 염

1-2 연습용 데이터베이스 생성

```
USE mysql;
DROP DATABASE IF EXISTS triggerDB;
CREATE DATABASE IF NOT EXISTS triggerDB;
```

1-3 [그림 1—18]의 중첩 트리거를 실습할 테이블 만들기

```
USE triggerDB;
CREATE TABLE orderTBL -- 구매 테이블
( orderNo INT AUTO_INCREMENT PRIMARY KEY, -- 구매 일련번호
  userID VARCHAR(5), -- 구매한 회원 아이디
  prodName VARCHAR(5), -- 구매한 물건
  orderamount INT -- 구매한 개수
);
CREATE TABLE prodTBL -- 물품 테이블
( prodName VARCHAR(5), -- 물건 이름
  account INT -- 남은 물건 수량
);
CREATE TABLE deliverTBL -- 배송 테이블
( deliverNo INT AUTO_INCREMENT PRIMARY KEY, -- 배송 일련번호
  prodName VARCHAR(5), -- 배송할 물건
  account INT UNIQUE -- 배송할 물건 개수
);
```

[실습 6-7] 중첩 트리거 생성하고 작동 확인하기

1-4 물품 테이블에 데이터 3건 삽입

```
INSERT INTO prodTBL VALUES ('사과', 100);
INSERT INTO prodTBL VALUES ('배', 100);
INSERT INTO prodTBL VALUES ('귤', 100);
```

[실습 6-7] 중첩 트리거 생성하고 작동 확인하기

2 중첩 트리거의 작동 확인하기

2-1 트리거를 구매 테이블(orderTBL)과 물품 테이블(prodTBL)에 부착

```
1 -- 물품 테이블에서 개수를 감소시키는 트리거
2 DROP TRIGGER IF EXISTS orderTrg;
3 DELIMITER //
4 CREATE TRIGGER orderTrg -- 트리거 이름
5 AFTER INSERT
6 ON orderTBL -- 트리거를 부착할 테이블
7 FOR EACH ROW
8 BEGIN
9 UPDATE prodTBL SET account = account - NEW.orderamount
10 WHERE prodName = NEW.prodName;
11 END //
12 DELIMITER ;
13
14 -- 배송 테이블에 새 배송 건을 삽입하는 트리거
15 DROP TRIGGER IF EXISTS prodTrg;
16 DELIMITER //
17 CREATE TRIGGER prodTrg -- 트리거 이름
18 AFTER UPDATE
19 ON prodTBL -- 트리거를 부착할 테이블
20 FOR EACH ROW
21 BEGIN
22 DECLARE orderAmount INT;
23 -- 주문 개수 = (변경 전 개수 - 변경 후 개수)
24 SET orderAmount = OLD.account - NEW.account;
25 INSERT INTO deliverTBL(prodName, account)
26 VALUES(NEW.prodName, orderAmount);
27 END //
28 DELIMITER ;
```

[실습 6-7] 중첩 트리거 생성하고 작동 확인하기

2-2 고객이 물건을 구매했다고 가정하고 삽입 작업 수행

```
INSERT INTO orderTBL VALUES (NULL, 'JOHN', '배', 5);
```

2-3 중첩 트리거가 잘 작동했는지 세 테이블 조회

```
SELECT * FROM orderTBL;
SELECT * FROM prodTBL;
SELECT * FROM deliverTBL;
```

orderNo	userID	prodName	orderamount
1	JOHN	배	5

prodName	account
사과	100
배	95
귤	100

deliverNo	prodName	account
1	배	5