## 3-1 커서의 개요

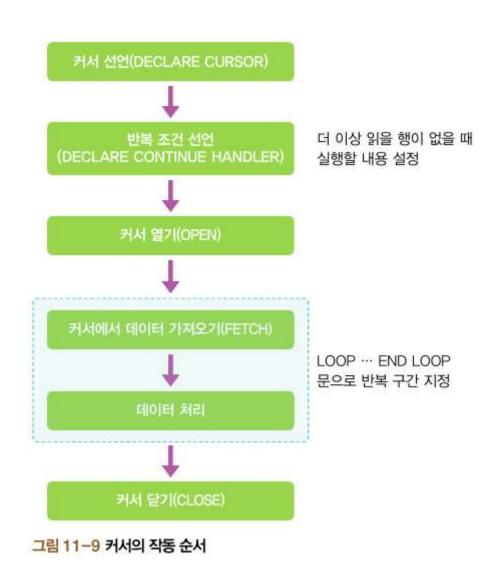
- 커서(cursor)
  - 일반 프로그래밍 언어로 파일 처리를 하는 것과 비슷한 방식으로 테이블의 행 집합을 다룰 수 있음
  - 쿼리의 결과 행 집합에서 한 행씩 옮겨가며 명령을 처리



그림 11-8 파일 처리의 동작

## 3-1 커서의 개요

■ 커서의 작동 순서



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

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
     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

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

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
    DECLARE id VARCHAR(10); -- 사용자 아이디를 저장할 변수
    DECLARE hap BIGINT; -- 총구매액을 저장할 변수
    DECLARE userGrade CHAR(5); -- 고객 등급 변수
8
    DECLARE endOfRow BOOLEAN DEFAULT FALSE;
10
    DECLARE userCuror CURSOR FOR -- 커서 선언
11
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
    OPEN userCuror; -- 커서 열기
21
22
    grade loop: LOOP
       FETCH userCuror INTO id, hap; -- 첫 행 값 대입
23
       IF endOfRow THEN
24
25
          LEAVE grade_loop;
```

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';
          WHEN (hap >= 1000) THEN SET userGrade ='우수고객';
30
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
     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 11-3] 커서 활용하기

### 2-3 고객 등급이 완성되었는지 확인

CALL gradeProc(); SELECT \* FROM userTBL;

	userID	userName	birthYear	addr	mobile 1	mobile2	height	mDate	grade
•	KHD	강호동	1970	경북	011	2222222	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	8888888	180	2006-04-04	일반고객
	LKK	이경규	1960	경남	018	99999999	170	2004-12-12	유령고객
	NHS	남희석	1971	충남	016	6666666	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

- 1 cookDB 초기화하기
  - 1-1 cookDB.sal 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;
```

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 11-5] 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
);
```

- 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 ;
```

#### 2-2 데이터를 삭제했을 때 작동하는 트리거 생성

```
1 DROP TRIGGER IF EXISTS backUserTBL_DeleteTrg;
2 DELIMITER //
3 CREATE TRIGGER backUserTBL_DeleteTrg -- 트리거 이름
4 AFTER DELETE -- 삭제 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 ' 삭제 ', CURDATE(), CURRENT_USER());
11 END //
12 DELIMITER ;
```

#### 2-3 데이터를 수정하고 삭제

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

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

### SELECT \* FROM backup\_userTBL;

	userID	userName	birthYear	addr	mobile 1	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	2222222	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;

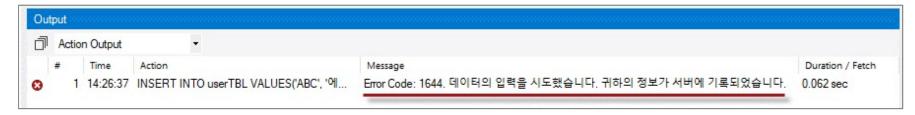
	userID	userName	birthYear	addr	mobile 1	mobile2	height	mDate	modType	modDate	modUser
•	KJD	김제동	1974	경남	NULL	NULL	173	2013-03-03	수정	2018-09-08	root@localhost
	KHD	강호동	1970	경북	011	2222222	182	2007-07-07	삭제	2018-09-08	root@localhost
	LHJ	이휘재	1972	경기	011	8888888	180	2006-04-04	삭제	2018-09-08	root@localhost
	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

- 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

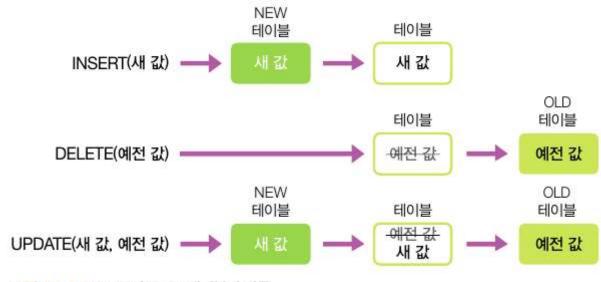


그림 11-15 NEW와 OLD 테이블의 작동

- BEFORE 트리거
  - 테이블에 변경이 가해지기 전에 작동하는 트리거

- 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
    BEFORE INSERT -- 데이터를 삽입하기 전 작동하도록 지정
7 ON userTBL -- 트리거를 부착할 테이블
  FOR EACH ROW
9 BEGIN
   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

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	mobile 1	mobile2	height	mDate
١	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	Statem	nent	Timing	Created	sql_mode
testTra	DELETE	testtbl	각 행	마다 적용시킴 BEGIN SET @msa = '가수	AFTER	2018-09-08 14:37:12.93	STRICT TRANS TABLES,NO E
userTBL BeforeInsertTra	INSERT	usertbl	BEGIN	IF NEW.birthYear < 1900 THEN S	BEFORE	2018-09-08 14:41:19.74	STRICT TRANS TABLES, NO E
backUserTBL UpdateTro	UPDATE	usertbl	BEGIN	INSERT INTO backup userTBL VALUES	AFTER	2018-09-08 14:37:58.18	STRICT TRANS TABLES, NO E
backUserTBL DeleteTro	DELETE	usertbl	BEGIN	INSERT INTO backup userTBL VALUES	AFTER	2018-09-08 14:38:05.84	STRICT TRANS TABLES, NO E

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

## 4-3 다중 트리거와 중첩 트리거

- 다중 트리거(multiple trigger)
  - 하나의 테이블에 동일한 트리거가 여러 개 부착되어 있는 것
- 중첩 트리거(nested trigger)
  - 트리거가 또 다른 트리거를 작동시키는 것

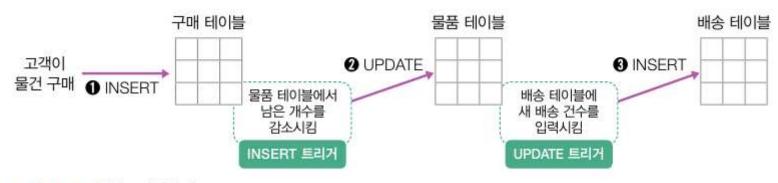


그림 11-18 중첩 트리거의 예

- ① 고객이 물건을 구매하면 그 구매 기록이 구매 테이블에 ① 삽입(INSERT)된다.
- ② 구매 테이블에 부착된 INSERT 트리거가 작동하면서 물품 테이블의 남은 개수에서 구매한 개수를 빼도록 ② 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 3 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

- 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
);
```

### 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);
```

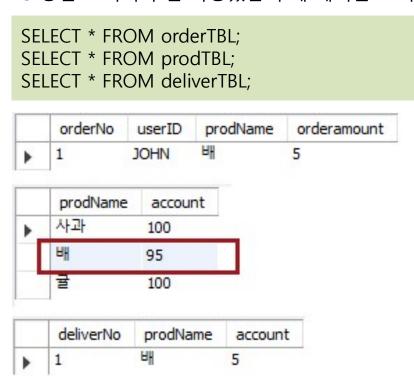
- 2 중첩 트리거의 작동 확인하기
  - 2-1 트리거를 구매 테이블(orderTBL)과 물품 테이블(prodTBL)에 부착

```
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;
```

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



2-4 배송 테이블의 열 이름을 변경하여 삽입이 실패하도록 조작

ALTER TABLE deliverTBL CHANGE prodName productName VARCHAR(5);

### 2-5 다시 삽입 작업 수행

INSERT INTO orderTBL VALUES (NULL, 'DANG', '사과', 9);

실행 결과

Error Code: 1054. Unknown column 'prodName' in 'field list'

### 2-6 세 테이블 확인

SELECT \* FROM orderTBL; SELECT \* FROM prodTBL; SELECT \* FROM deliverTBL;

orderNo	userID	prodName	orderamount
1	JOHN	FIH	5

	prodName	account
•	사과	100
	배	95
	귤	100

	deliverNo	prodName	account	
•	1	배	5	