6 . 1 스토어드 프로시저

❖ 스토어드 프로시저의 개요

▪ 스토어드 프로시저(Stored Procedure)

- 저장 프로시저라고도 불림
- MySQL에서 제공되는 프로그래밍 기능
- 쿼리문의 집합으로 어떠한 동작을 일괄 처리하기 위한 용도로 사용
- 쿼리 모듈화
  - **필요할 때마다 호출만 하면 훨씬 편리하게 MySQL 운영**
  - **CALL 프로시저_이름( ) 한 줄로 해결되는 편리함**

6.1 스토어드 프로시저

❖ 스토어드 프로시저의 개요

▪ 기본 형식

```
형식 :
DELIMITER $$
CREATE PROCEDURE 스토어드 프로시저이름( IN 또는 OUT 파라미터 )
BEGIN

    이 부분에 SQL 프로그래밍 코딩..

END $$
DELIMITER ;
CALL 스토어드 프로시저이름();
```

6 . 1 스토어드 프로시저

❖ 스토어드 프로시저의 개요

▪ 스토어드 프로시저의 수정과 삭제

- 수정 : ALTER PROCEDURE
- 삭제 : DROP PROCEDURE

▪ 매개 변수의 사용 (입력 매개 변수를 지정하는 형식)

- **IN 입력_매개 변수_이름 데이터_형식**

- 입력 매개 변수가 있는 스토어드 프로시저 실행 방법
  - **CALL 프로시저_이름(전달 값);**
- 출력 매개 변수 지정 방법
  - **OUT 출력_매개 변수_이름 데이터_형식**
  - **출력 매개 변수에 값 대입하기 위해 주로 SELECT… INTO문 사용**

6.1 스토어드 프로시저

❖ 스토어드 프로시저의 개요

- **프로그래밍 기능**
- **더 강력하고 유연한 기능 포함하는 스토어드 프로시저 생성**
- **스토어드 프로시저 내의 오류 처리**
  - 스토어드 프로시저 내부에서 오류가 발생했을 경우
    - **DECLARE 액션 HANDLER FOR 오류조건 처리할_문장 구문**
    - **7장의 후반부에서 공부했던 내용**

6.1 스토어드 프로시저

❖ 스토어드 프로시저의 특징

▪ MySQL의 성능 향상

- 긴 쿼리가 아니라 짧은 프로시저 내용만 클라이언트에서 서버로 전송
  - **네트워크 부하 줄임 → MySQL의 성능 향상**

▪ 유지관리가 간편

- 응용 프로그램에서는 프로시저만 호출
  - **데이터베이스 단에서 관련된 스토어드 프로시저의 내용 일관되게 수정/유지보수**

6 . 1 스토어드 프로시저

❖ 스토어드 프로시저의 특징

▪ 모듈식 프로그래밍 가능

- 언제든지 실행 가능 + 편리한 관리
- 모듈식 프로그래밍 언어와 동일한 장점

▪ 보안 강화에 편리

- 스토어드 프로시저에만 접근 권한을 주어 DB가 안전해짐

6. 2 스토어드 함수

❖ 스토어드 함수 (Stored Function)

- 사용자가 직접 만들어서 사용하는 함수
- 스토어드 프로시저와 상당히 유사
  - 형태와 사용 용도에 있어 차이 있음
- 스토어드 함수의 개요

```
DELIMITER $$
CREATE FUNCITON 스토어드 함수이름( 파라미터 )
    RETURNS  반환형식
BEGIN

    이 부분에 프로그래밍 코딩..
    RETURN 반환값;

END $$
DELIMITER ;
SELECT 스토어드_함수이름();
```

6. 2 스토어드 함수

❖ 스토어드 함수 와 스토어드 프로시저의 차이점

▪ 스토어드 함수

- 파라미터에 IN, OUT 등을 사용할 수 없음
  - **모두 입력 파라미터로 사용**
- RETURNS문으로 반환할 값의 데이터 형식 지정
  - **본문 안에서는 RETURN문으로 하나의 값 반환**
- SELECT 문장 안에서 호출
- 안에서 집합 결과 반환하는 SELECT 사용 불가
  - **SELECT… INTO… 는 집합 결과 반환하는 것이 아니므로 예외적으로 스토어드 함수에서 사용 가능**
- 어떤 계산 통해서 하나의 값 반환하는데 주로 사용

6. 2 스토어드 함수

❖ 스토어드 함수 와 스토어드 프로시저의 차이점

▪ 스토어드 프로시저

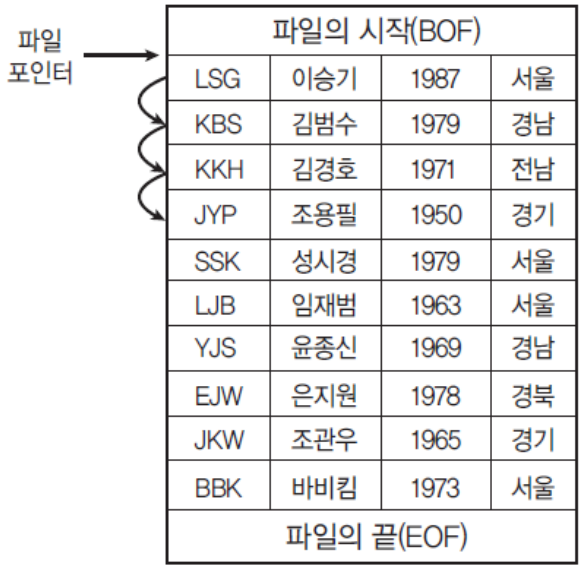
- 파라미터에 IN, OUT 등을 사용 가능
- 별도의 반환하는 구문이 없음
  - **꼭 필요하다면 여러 개의 OUT파라미터 사용해서 값 반환 가능**
- CALL로 호출
- 안에 SELECT문 사용 가능
- 여러 SQL문이나 숫자 계산 등의 다양한 용도로 사용

6. 3 커서

❖ 커서의 개요

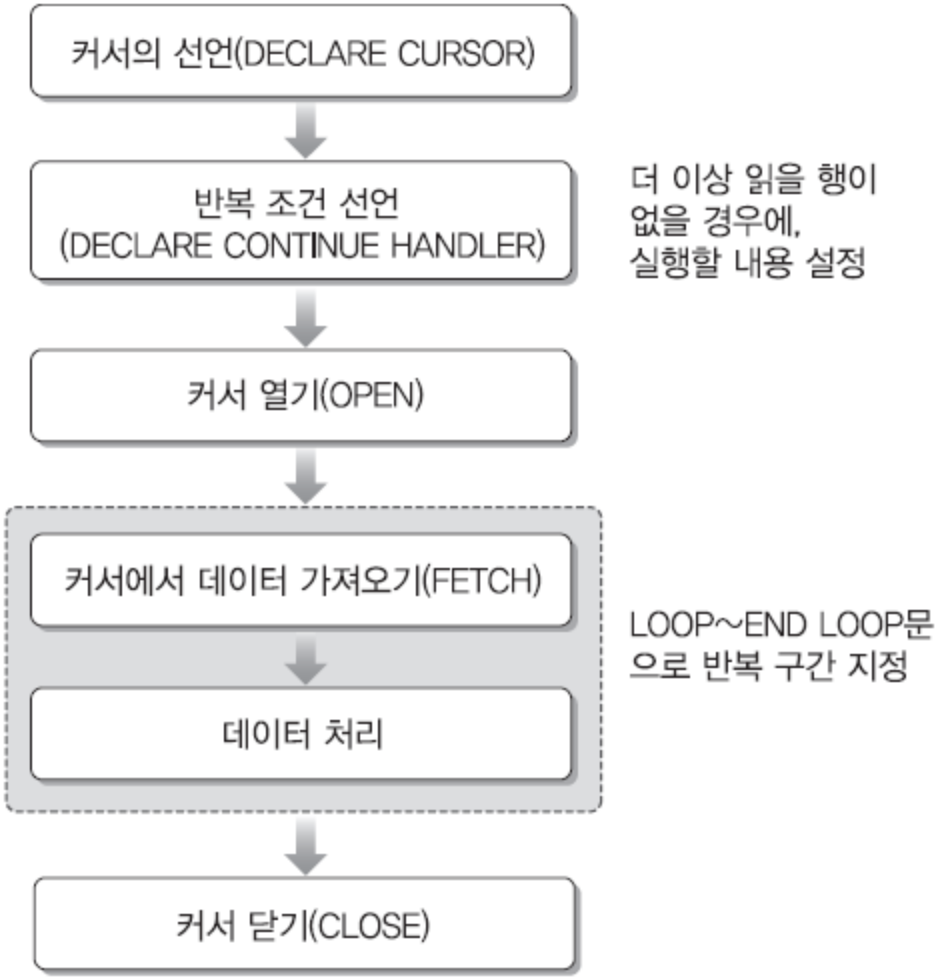
▪ 커서(Cursor)

- 스토어드 프로시저 내부에 사용
- 일반 프로그래밍 언어의 파일 처리와 방법이 비슷함
  - **행의 집합을 다루기에 편리한 많은 기능을 제공**
- 테이블에서 여러 개의 행을 쿼리한 후, 쿼리의 결과인 행 집합을 한 행씩 처리하기 위한 방식



6. 3 커서

❖ 커서의 처리 순서



6. 4 트리거

❖ 트리거(Trigger)

▪ 트리거의 개요

- 사전적 의미로 '방아쇠' - 방아쇠 당기면 '자동'으로 총알이 나가듯 실행
- 테이블에 DML문(Insert, Update, Delete 등) 이벤트가 발생될 때 작동
- 테이블에 부착되는 프로그램 코드
- 직접 실행 불가 - 테이블에 이벤트 일어나야 자동 실행
- IN, OUT 매개 변수를 사용할 수 없음
- MySQL은 View에 트리거 부착 불가!!!

6. 4 트리거

❖ 트리거의 종류

▪ AFTER 트리거

- 테이블에 INSERT, UPDATE, DELETE 등의 작업이 일어났을 때 작동
- 이름이 뜻하는 것처럼 해당 작업 후에 (After) 작동

▪ BEFORE 트리거

- BEFORE 트리거는 이벤트가 발생하기 전에 작동
- INSERT, UPDATE, DELETE 세 가지 이벤트로 작동

6. 4 트리거

❖ 트리거의 사용

▪ 트리거 문법

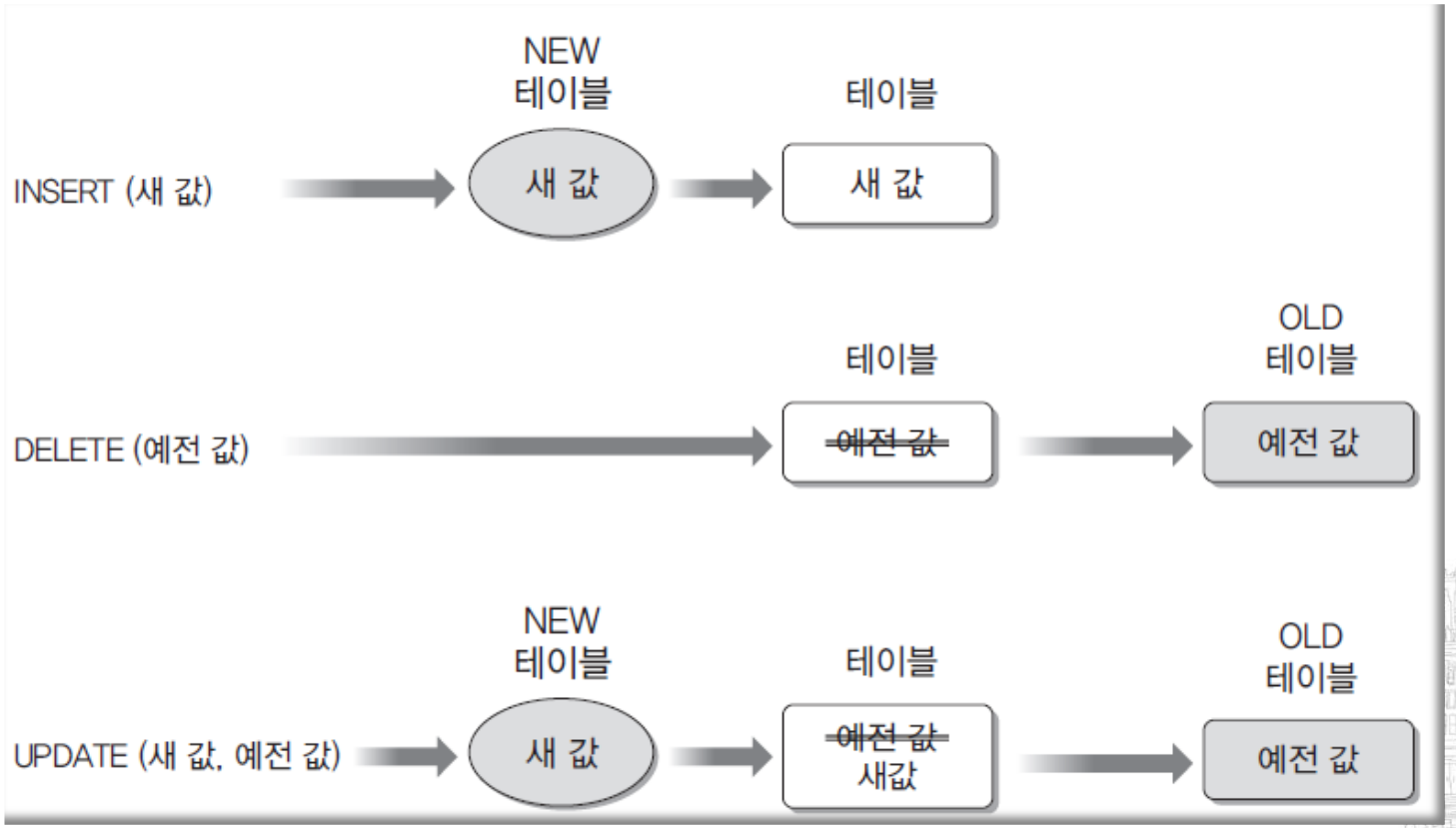
```
CREATE
    [DEFINER = { user | CURRENT_USER }]
    TRIGGER trigger_name
    trigger_time trigger_event
    ON tbl_name FOR EACH ROW
    [trigger_order]
    trigger_body

trigger_time: { BEFORE | AFTER }

trigger_event: { INSERT | UPDATE | DELETE }

trigger_order: { FOLLOWS | PRECEDES } other_trigger_name
```

6. 4 트리거

❖ 트리거의 사용

▪ 트리거가 생성하는 임시 테이블

- INSERT, UPDATE, DELETE 작업이 수행되면 임시 사용 시스템 테이블
- 이름은 'NEW'와 'OLD'



6. 4 트리거

❖ 트리거의 사용

▪ BEFORE 트리거의 사용

- 테이블에 변경이 가해지기 전 작동
- BEFORE 트리거의 좋은 활용 예
  - **BEFORE INSERT 트리거를 부착해 놓으면 입력될 데이터 값을 미리 확인해서 문제가 있을 경우에 다른 값으로 변경**
- 생성된 트리거 확인
  - **SHOW TRIGGERS FROM sqlDB;**
- 트리거를 삭제하자.
  - **DROP TRIGGER userTbl_BeforeInsertTrg;**

6. 4 트리거

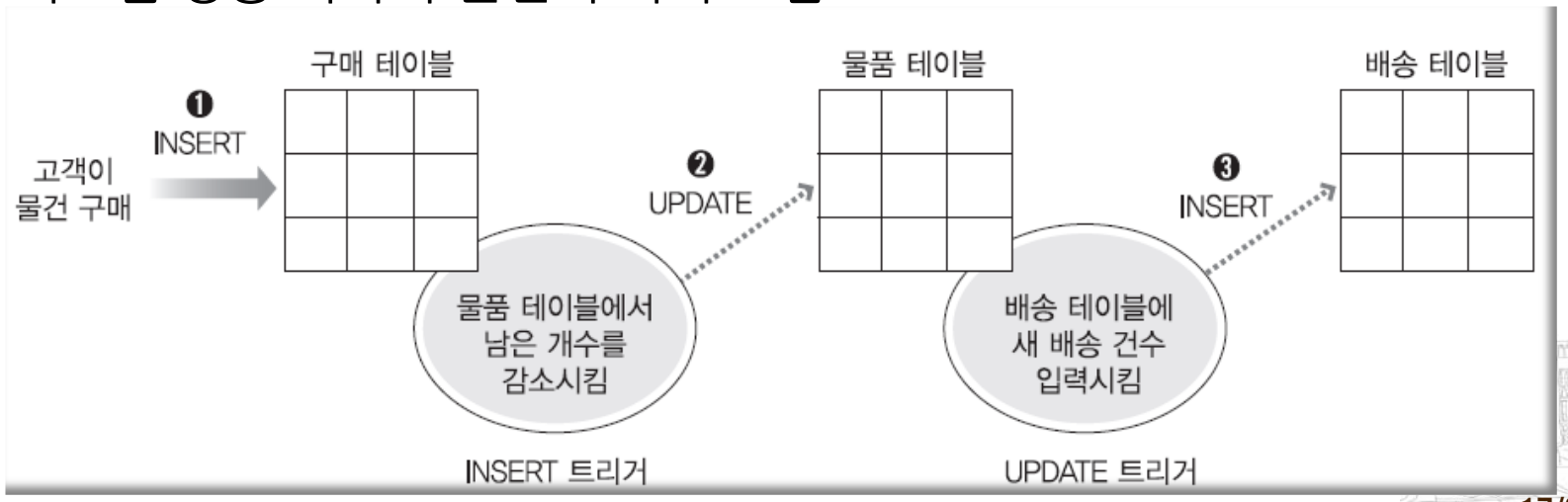
❖ 기타 트리거에 관한 내용

▪ 다중 트리거 (Multiple Triggers)

- 하나의 테이블에 동일한 트리거가 여러 개 부착되어 있는 것
- Ex) AFTER INSERT 트리거가 한 개 테이블에 2개 이상 부착

▪ 중첩 트리거 (Nested Triggers)

- 트리거가 또 다른 트리거를 작동시키는 것
- 시스템 성능 저하의 원인이 되기도 함



6. 4 트리거

❖ 기타 트리거에 관한 내용

▪ 트리거의 작동 순서

- 하나의 테이블에 여러 개의 트리거가 부착된 경우
- 트리거의 작동 순서 지정 가능
  - **{ FOLLOWS | PRECEDES } other_trigger_name 에서 설정**
- 중첩 트리거 작동 학습
  - **테이블 내용을 전체 살펴보는 것으로 결과 확인**