

## 2. 데이터베이스 활용과 클라우드 서버 환경 준비

Section 2 . SQL 심화 / 데이터 모델링 / 데이터베이스연동

# SQL 심화



# 목차

**01**    Stored Procedure (저장 프로시저)

**02**    Stored Function (저장 함수)

**03**    Curosr (커서)

**04**    Trigger (트리거)

## 01 스토어드 프로시저

- 스토어드 프로시저(Stored Procedure, 저장 프로시저)
  - 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( ) 으로 호출

# 01 스토어드 프로시저

## 스토어드 프로시저

- 기본 형식

```
형식 :  
DELIMITER $$  
CREATE PROCEDURE 스토어드 프로시저이름( IN 또는 OUT 파라미터 )  
BEGIN  
  
    이 부분에 SQL 프로그래밍 코딩..  
  
END $$  
DELIMITER ;  
CALL 스토어드 프로시저이름();
```

## 스토어드 프로시저

- 스토어드 프로시저 생성 예

```
USE sqlDB;
DROP PROCEDURE IF EXISTS userProc;
DELIMITER $$
CREATE PROCEDURE userProc()
BEGIN
    SELECT * FROM userTbl; -- 스토어드 프로시저 내용
END $$
DELIMITER ;

CALL userProc();
```

## 스토어드 프로시저

- 스토어드 프로시저의 수정과 삭제

- 수정 : ALTER PROCEDURE
- 삭제 : DROP PROCEDURE

- 매개 변수의 사용

- 입력 매개 변수를 지정하는 형식

```
IN 입력_매개변수_이름 데이터_형식
```

- 입력 매개 변수가 있는 스토어드 프로시저 실행 방법

```
CALL 프로시저_이름(전달 값);
```

## 스토어드 프로시저

### ◦ 매개 변수의 사용

- 출력 매개 변수 지정 방법

```
OUT 출력_매개변수_이름 데이터_형식
```

- 출력 매개 변수에 값 대입하기 위해 주로 SELECT... INTO문 사용
- 출력 매개 변수가 있는 스토어드 프로시저 실행 방법

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



## 스토어드 프로시저

- 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

## 스토어드 프로시저의 개요

- 스토어드 프로시저 실습
  - 입력 매개 변수 1개의 스토어드 프로시저 생성

```
USE sqlDB;
DROP PROCEDURE IF EXISTS userProc1;
DELIMITER $$
CREATE PROCEDURE userProc1(IN userName VARCHAR(10))
BEGIN
    SELECT * FROM userTbl WHERE name = userName;
END $$
DELIMITER ;

CALL userProc1('조관우');
```

	userID	name	birthYear	addr	mobile1	mobile2	height	mDate
▶	JKW	조관우	1965	경기	018	9999999	172	2010-10-10

## 스토어드 프로시저

- 스토어드 프로시저 실습

- 입력 매개 변수 2개의 스토어드 프로시저 생성

```
DROP PROCEDURE IF EXISTS userProc2;
DELIMITER $$
CREATE PROCEDURE userProc2(
    IN userBirth INT,
    IN userHeight INT
)
BEGIN
    SELECT * FROM userTbl
        WHERE birthYear > userBirth AND height > userHeight;
END $$
DELIMITER ;

CALL userProc2(1970, 178);
```

	userID	name	birthYear	addr	mobile1	mobile2	height	mDate
▶	LSG	이승기	1987	서울	011	1111111	182	2008-08-08
	SSK	성시경	1979	서울	NULL	NULL	186	2013-12-12

## 스토어드 프로시저의 개요

- 스토어드 프로시저 실습

- 출력 매개변수 설정

- 스토어드 프로시저 생성 및 테스트로 사용할 테이블 생성

```
DROP PROCEDURE IF EXISTS userProc3;
DELIMITER $$
CREATE PROCEDURE userProc3(
    IN txtValue CHAR(10),
    OUT outValue INT
)
BEGIN
    INSERT INTO testTBL VALUES(NULL,txtValue);
    SELECT MAX(id) INTO outValue FROM testTBL;
END $$
DELIMITER ;
```

## 스토어드 프로시저의 개요

- 스토어드 프로시저 실습

- 출력 매개변수 설정

- 스토어드 프로시저 생성 및 테스트로 사용할 테이블 생성

```
CREATE TABLE IF NOT EXISTS testTBL(  
    id INT AUTO_INCREMENT PRIMARY KEY,  
    txt CHAR(10)  
);
```

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

출력 결과:

현재 입력된 ID 값 ==> 1

## 스토어드 프로시저

- 스토어드 프로시저 실습

- 스토어드 프로시저 안에 SQL 프로그래밍 활용
  - While문 활용한 2중 반복문

```
DROP TABLE IF EXISTS guguTBL;
CREATE TABLE guguTBL (txt VARCHAR(100)); -- 구구단 저장용 테이블

DROP PROCEDURE IF EXISTS whileProc;
DELIMITER $$
CREATE PROCEDURE whileProc()
BEGIN
    DECLARE str VARCHAR(100); -- 각 단을 문자열로 저장
    DECLARE i INT; -- 구구단 앞자리
    DECLARE k INT; -- 구구단 뒷자리
    SET i = 2; -- 2단부터 계산
```

## 스토어드 프로시저

- 스토어드 프로시저 실습

- 스토어드 프로시저 안에 SQL 프로그래밍 활용

- While문 활용한 2중 반복문

```
WHILE (i < 10) DO -- 바깥 반복문. 2단~9단까지
    SET str = ''; -- 각 단의 결과를 저장할 문자열 초기화
    SET k = 1; -- 구구단 뒷자리는 항상 1부터 9까지
    WHILE (k < 10) DO
        SET str = CONCAT(str, ' ', i, 'x', k, '=', i*k); -- 문자열 만들기
        SET k = k + 1; -- 뒷자리 증가
    END WHILE;
    SET i = i + 1; -- 앞자리 증가
    INSERT INTO guguTBL VALUES(str); -- 각 단의 결과를 테이블에 입력
END WHILE;

END $$

DELIMITER ;

CALL whileProc();
SELECT * FROM guguTBL;
```

## 스토어드 프로시저

- 스토어드 프로시저 실습

- 스토어드 프로시저 안에 SQL 프로그래밍 활용
  - While문 활용한 2중 반복문

	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



## 스토어드 프로시저의 개요

### ◦ 스토어드 프로시저 실습

- 현재 저장된 프로시저의 이름 및 내용 확인
- INFORMATION\_SCHEMA 데이터베이스의 ROUTINES 테이블을 조회하면 내용 확인 가능
  - SELECT routine\_name, routine\_definition FROM INFORMATION\_SCHEMA.ROUTINES WHERE routine\_schema = 'sqlldb' AND routine\_type = 'PROCEDURE';

ROUTINE_NAME	ROUTINE_DEFINITION
▶ caseProc	BEGIN DECLARE bYear INT; DECLARE tti CHAR(3); -- 띠 SELECT birthYear into bYear FROM userTbl WHERE name = userName; CASE WHEN ...
errorProc	BEGIN DECLARE i INT; -- 1씩 증가하는 값 DECLARE hap INT UNSIGNED; -- 합계 DECLARE saveHap INT UNSIGNED; -- 합계 DECLARE EXIT ...
ifelseProc	BEGIN DECLARE bYear INT; -- 변수 선언 SELECT birthYear into bYear FROM userTbl WHERE name = userName; IF (bYear >= 1980) THEN SELECT '...
userProc	BEGIN SELECT * FROM userTbl; -- 스토어드 프로시저 내용 END
userProc1	BEGIN SELECT * FROM userTbl WHERE name = userName; END
userProc2	BEGIN SELECT * FROM userTbl WHERE birthYear > userBirth AND height > userHeight; END
userProc3	BEGIN INSERT INTO testTBL VALUES(NULL,txtValue); SELECT MAX(id) INTO outValue FROM testTBL; END
whileProc	BEGIN DECLARE str VARCHAR(100); -- 각 단을 문자열로 저장 DECLARE i INT; -- 구구단 앞자리 DECLARE k INT; -- 구구단 뒷자리 SET i = 2; -...

## 스토어드 프로시저

- 스토어드 프로시저 실습

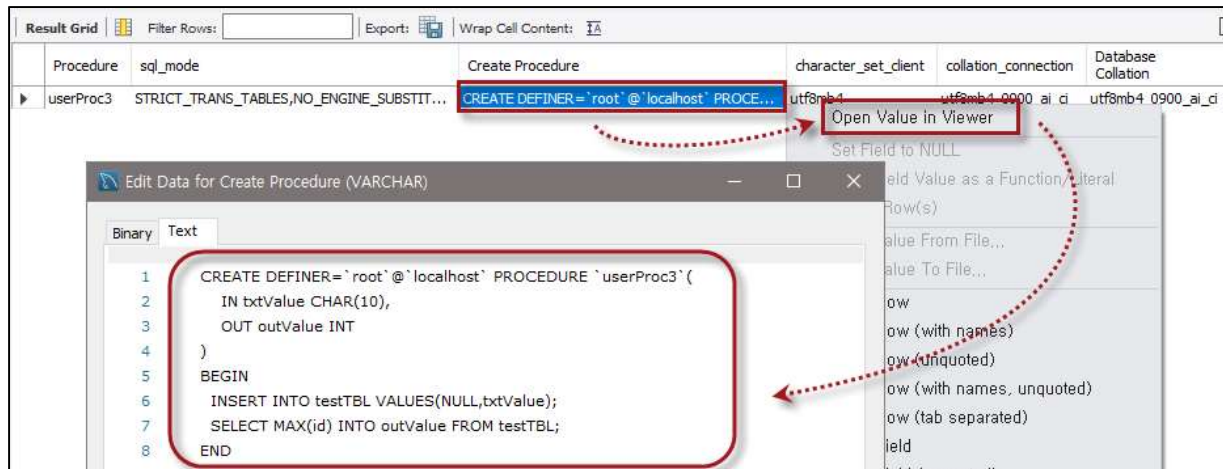
- 파라미터를 확인하려면 INFORMATION\_SCHEMA 데이터베이스의 ROUTINES 테이블 조회
  - SELECT parameter\_mode, parameter\_name, dtd\_identifier  
FROM INFORMATION\_SCHEMA.PARAMETERS  
WHERE specific\_name = 'userProc3';

	PARAMETER_MODE	PARAMETER_NAME	DTD_IDENTIFIER
▶	IN	txtValue	char(10)
	OUT	outValue	int(11)

## 스토어드 프로시저

### ◦ 스토어드 프로시저 실습

- SHOW CREATE PROCEDURE문으로도 스토어드 프로시저의 내용 확인 가능
  - SHOW CREATE PROCEDURE sqladb.userProc3;



## 스토어드 프로시저

- MySQL의 성능 향상
  - 긴 쿼리가 아니라 짧은 프로시저 내용만 클라이언트에서 서버로 전송
    - 네트워크 부하 줄임으로 MySQL의 성능 향상
- 유지관리가 간편
  - 응용 프로그램에서는 프로시저만 호출
    - 데이터베이스에서 관련된 스토어드 프로시저의 내용 수정/유지보수

## 특징

- 모듈식 프로그래밍 가능

- 언제든지 실행 가능
- 스토어드 프로시저로 저장해 놓은 쿼리의 수정, 삭제 등의 관리 수월
- 모듈식 프로그래밍 언어와 동일한 장점 있음

- 보안 강화

- 사용자 별로 테이블 접근 권한 주지 않고 스토어드 프로시저에만 접근 권한을 주어 보안 강화
  - 뷰 또한 스토어드 프로시저와 같이 보안 강화 가능

## 02 스토어드 함수

### 스토어드 함수 (Stored Function)

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

```
DELIMITER $$  
CREATE FUNCTION 스토어드 함수이름( 파라미터 )  
    RETURNS 반환형식  
BEGIN  
  
    이 부분에 프로그래밍 코딩..  
    RETURN 반환값;  
  
END $$  
DELIMITER ;  
SELECT 스토어드_함수이름();
```

## 스토어드 함수와 스토어드 프로시저의 차이점

### ◦ 스토어드 함수

- 파라미터에 IN, OUT 등을 사용할 수 없음
  - 모두 입력 파라미터로 사용
- **RETURNS**문으로 반환할 값의 데이터 형식 지정
  - **본문 안에서는 RETURN**문으로 하나의 값 반환
- SELECT 문장 안에서 호출
- 함수 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

## 스토어드 함수와 스토어드 프로시저의 차이점

### ◦ 스토어드 프로시저

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



## 스토어드 함수와 스토어드 프로시저의 차이점

### ◦ 스토어드 함수

- 스토어드 함수를 사용하기 위해서는 스토어드 함수 생성 권한을 허용 해야함

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

- ex) 2개의 숫자의 합계를 계산하는 스토어드 함수

```
USE sqlDB;
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);
```

## 02스토어드 함수

### 스토어드 함수 실습

- sqlDB 데이터베이스 초기화
- 출생년도 입력하면 나이 출력되는 함수 작성
  - 테이블을 조회할 때 주로 사용되는 함수

```
USE sqlDB;
DROP FUNCTION IF EXISTS getAgeFunc;
DELIMITER $$
CREATE FUNCTION getAgeFunc(bYear INT)
    RETURNS INT
BEGIN
    DECLARE age INT;
    SET age = YEAR(CURDATE()) - bYear;
    RETURN age;
END $$
DELIMITER ;
```

## 스토어드 함수 실습

### 함수 호출

- SELECT getAgeFunc(1979);
  - 1979년생의 현재 나이가 출력됨
- 함수의 반환값을 SELECT ... INTO ... 로 저장했다가 사용 가능

```
SELECT getAgeFunc(1979) INTO @age1979;  
SELECT getAgeFunc(1997) INTO @age1989;  
SELECT CONCAT('1997년과 1979년의 나이차 ==> ', (@age1979-@age1989));
```

- 두 출생년도의 나이차가 출력됨

## 스토어드 함수 실습

함수는 주로 테이블을 조회할 때 활용됨

- `SELECT userID, name, getAgeFunc(birthYear) AS '만 나이' FROM userTbl;`

	userID	name	만 나이
▶	BBK	박비킴	46
	EJW	은지원	47
	JKW	조관우	54
	JYP	조용필	69
	KBS	김범수	40
	KKH	김경호	48
	LJB	임재범	56
	LSG	이승기	32
	SSK	성시경	40
	YJS	윤종신	50

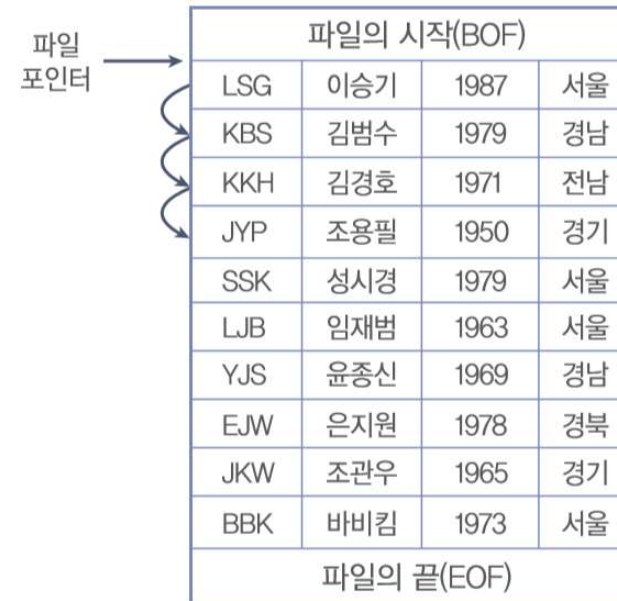
## 스토어드 함수 실습

- 현재 저장된 스토어드 함수의 이름 및 내용 확인
  - `SHOW CREATE FUNCTION getAgeFunc;`
- 스토어드 함수 삭제
  - 다른 데이터베이스 rocph와 마찬가지로 DROP문 사용
  - `DROP FUNCTION getAgeFunc;`

## 03 커서

### 커서의 개요

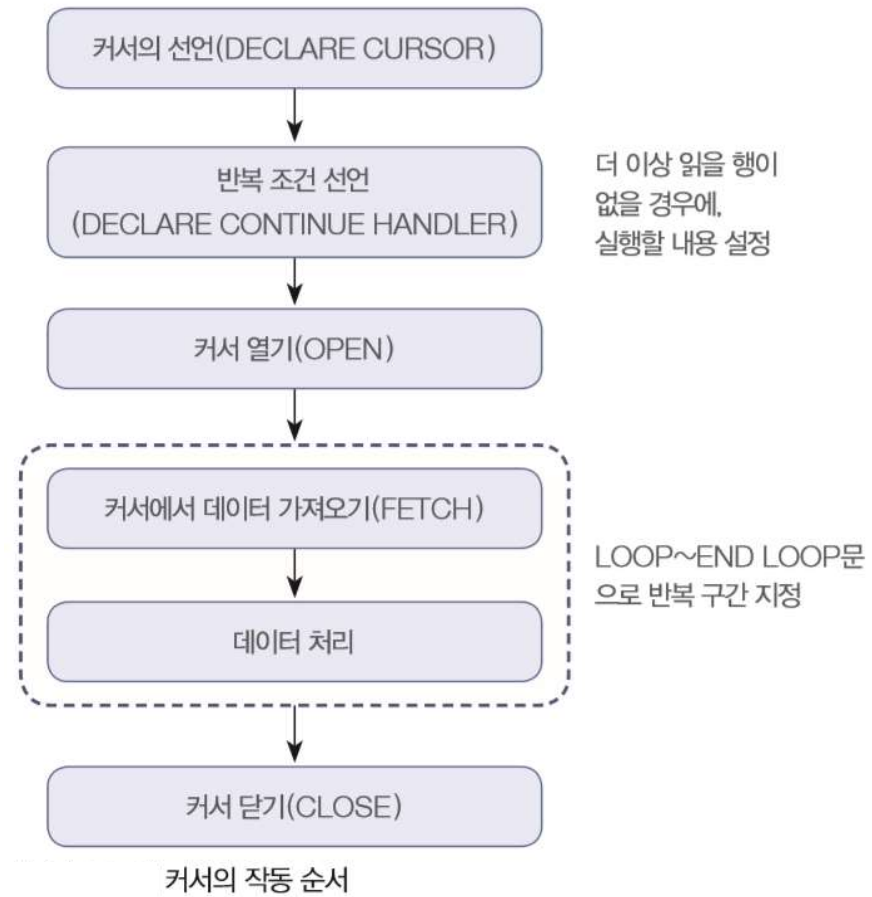
- 커서(Cursor)
  - 스토어드 프로시저 내부에 사용
  - 일반 프로그래밍 언어의 파일 처리와 방법이 비슷함
    - 행의 집합을 다루기 편리한 기능 제공
  - 테이블에서 여러 개의 행을 쿼리한 후,  
쿼리의 결과인 행 집합을 한 행씩 처리하기 위한 방식



파일의 시작(BOF)			
LSG	이승기	1987	서울
KBS	김범수	1979	경남
KKH	김경호	1971	전남
JYP	조용필	1950	경기
SSK	성시경	1979	서울
LJB	임재범	1963	서울
YJS	윤종신	1969	경남
EJW	은지원	1978	경북
JKW	조관우	1965	경기
BBK	바비킴	1973	서울
파일의 끝(EOF)			

파일 처리의 작동 개념

## 커서의 처리 순서



## 커서의 처리 순서

- 커서 이용해 고객의 평균 키 구하는 스토어드 프로시저(P. 459)
  - AVG() 내장 함수와 동일한 기능 구현(AVG() 내장 함수가 빠르고 편리)
    - 커서 이해를 위한 실습
    - 조건이 붙은 평균은 AVG() 함수대신 커서 활용

```
USE sqlDB;
DROP PROCEDURE IF EXISTS cursorProc;
DELIMITER $$
CREATE PROCEDURE cursorProc()
BEGIN
    DECLARE userHeight INT; -- 고객의 키
    DECLARE cnt INT DEFAULT 0; -- 고객의 인원 수(=읽은 행의 수)
    DECLARE totalHeight INT DEFAULT 0; -- 키의 합계

    DECLARE endOfRow BOOLEAN DEFAULT FALSE; -- 행의 끝 여부(기본을 FALSE)
```



## 커서의 처리 순서

- 커서 이용해 고객의 평균 키 구하는 스토어드 프로시저

```
DECLARE userCuror CURSOR FOR-- 커서 선언
    SELECT height FROM userTbl;

DECLARE CONTINUE HANDLER -- 행의 끝이면 endOfRow 변수에 TRUE를 대입
    FOR NOT FOUND SET endOfRow = TRUE;

OPEN userCuror; -- 커서 열기

cursor_loop: LOOP
    FETCH userCuror INTO userHeight; -- 고객 키 1개를 대입

    IF endOfRow THEN -- 더이상 읽을 행이 없으면 Loop를 종료
        LEAVE cursor_loop;
    END IF;
```

## 커서의 처리 순서

- 커서 이용해 고객의 평균 키 구하는 스토어드 프로시저

```
        SET cnt = cnt + 1;
        SET totalHeight = totalHeight + userHeight;
    END LOOP cursor_loop;

    -- 고객 키의 평균을 출력한다.
    SELECT CONCAT('고객 키의 평균 ==> ', (totalHeight/cnt));

    CLOSE userCuror; -- 커서 닫기
END $$
DELIMITER ;
```

- 스토어드 프로시저 호출

```
CALL cursorProc();
```

결과 값:

고객 키의 평균==> 175.8000

## 커서의 처리 순서

- 테이블에 열 하나 추가 후 구매총액 따라 회원등급 설정
- userTBL에 고객 등급을 입력할 열을 추가
  - ALTER TABLE userTbl ADD grade VARCHAR(5); -- 고객 등급 열 추가
- 스토어드 프로시저 작성

```
DROP PROCEDURE IF EXISTS gradeProc;
DELIMITER $$
CREATE PROCEDURE gradeProc()
BEGIN
    DECLARE id VARCHAR(10); -- 사용자 아이디를 저장할 변수
    DECLARE hap BIGINT; -- 총 구매액을 저장할 변수
    DECLARE userGrade CHAR(5); -- 고객 등급 변수

    DECLARE endOfRow BOOLEAN DEFAULT FALSE;

    DECLARE userCuror CURSOR FOR-- 커서 선언
        SELECT U.userid, sum(price*amount)
```

## 커서의 처리 순서

- 테이블에 열 하나 추가 후 구매총액 따라 회원등급 설정
- 스토어드 프로시저 작성

```
FROM buyTbl B
      RIGHT OUTER JOIN userTbl U
      ON B.userid = U.userid
GROUP BY U.userid, U.name ;
```

```
DECLARE CONTINUE HANDLER
FOR NOT FOUND SET endOfRow = TRUE;
```

```
OPEN userCuror; -- 커서 열기
grade_loop: LOOP
    FETCH userCuror INTO id, hap; -- 첫 행 값을 대입
    IF endOfRow THEN
        LEAVE grade_loop;
    END IF;
```

## 커서의 처리 순서

- 테이블에 열 하나 추가 후 구매총액 따라 회원등급 설정
- 스토어드 프로시저 작성

```
CASE
    WHEN (hap >= 1500) THEN SET userGrade = '최우수고객';
    WHEN (hap >= 1000) THEN SET userGrade = '우수고객';
    WHEN (hap >= 1) THEN SET userGrade = '일반고객';
    ELSE SET userGrade = '유령고객';
END CASE;

UPDATE userTbl SET grade = userGrade WHERE userID = id;
END LOOP grade_loop;

CLOSE userCuror; -- 커서 닫기
END $$
DELIMITER ;
```

## 커서의 처리 순서

- 테이블에 열 하나 추가 후 구매총액 따라 회원등급 설정
- 스토어드 프로시저 호출 및 고객 등급 확인

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

	userID	name	birthYear	addr	mobile1	mobile2	height	mDate	grade
▶	BBK	바비킴	1973	서울	010	0000000	176	2013-05-05	최우수고객
	EJW	은지원	1972	경북	011	8888888	174	2014-03-03	일반고객
	JKW	조관우	1965	경기	018	9999999	172	2010-10-10	유령고객
	JYP	조용필	1950	경기	011	4444444	166	2009-04-04	일반고객
	KBS	김범수	1979	경남	011	2222222	173	2012-04-04	우수고객
	KKH	김경호	1971	전남	019	3333333	177	2007-07-07	유령고객
	LJB	임재범	1963	서울	016	6666666	182	2009-09-09	유령고객
	LSG	이슬기	1987	서울	011	1111111	182	2008-08-08	유령고객
	SSK	성시경	1979	서울	NULL	NULL	186	2013-12-12	일반고객
	YJS	윤종신	1969	경남	NULL	NULL	170	2005-05-05	유령고객

## 04 트리거

### 트리거(Trigger)의 개요

- 트리거란?

- 사전적 의미로 '방아쇠'
- 방아쇠 당기면 '자동'으로 총알이 나가듯이 테이블에 무슨 일이 일어나면 '자동'으로 실행
- 제약 조건과 더불어 데이터 무결성을 위해 MySQL에서 사용할 수 있는 기능
- 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

## 04 트리거

### 트리거(Trigger) 의 개요

- 트리거 실습
  - testDB에 테이블 생성

```
CREATE DATABASE IF NOT EXISTS testDB;
USE testDB;
CREATE TABLE IF NOT EXISTS testTbl (id INT, txt VARCHAR(10));
INSERT INTO testTbl VALUES(1, '레드벨벳');
INSERT INTO testTbl VALUES(2, '잇지');
INSERT INTO testTbl VALUES(3, '블랙핑크');
```



## 04 트리거

### 트리거(Trigger) 의 개요

- 트리거 실습
  - testTbl에 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 적용시킴
BEGIN
    SET @msg = '가수 그룹이 삭제됨' ; -- 트리거 실행 시 작동되는 코드들
END //
DELIMITER ;
```

## 트리거(Trigger) 의 개요

- 트리거 실습

- 데이터 삽입, 수정, 삭제

```
SET @msg = '';
INSERT INTO testTbl VALUES(4, '마마무');
SELECT @msg;
UPDATE testTbl SET txt = '블핑' WHERE id = 3;
SELECT @msg;
DELETE FROM testTbl WHERE id = 4;
SELECT @msg;
```

	@msg
▶	

	@msg
▶	

	@msg
▶	가수 그룹이 삭제됨

## 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

## 트리거의 사용

- 트리거 문법

```
CREATE
  [DEFINER = 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
```

## 트리거의 사용

- AFTER 트리거의 사용

- 예제 요구 사항

- 회원 테이블에 update나 delete를 시도하면 수정 또는 삭제된 데이터를 별도의 테이블에 보관하고 변경된 일자와 변경한 사람을 기록

## 트리거의 사용

### ◦ AFTER 트리거의 사용

- insert나 update 작업이 일어나는 경우, 변경되기 전의 데이터를 저장할 테이블 생성

```
USE sqlDB;
DROP TABLE buyTbl; -- 구매 테이블은 실습에 필요 없으므로 삭제
CREATE TABLE backup_userTbl
( userID CHAR(8) NOT NULL PRIMARY KEY,
  name   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 타입. '수정' 또는 '삭제'
  modDate DATE, -- 변경된 날짜
  modUser VARCHAR(256) -- 변경한 사용자
);
```

## 트리거의 사용

- AFTER 트리거의 사용

- 변경(Update) 발생시 작동하는 backUserTbl\_UpdateTrg 트리거 생성

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

## 트리거의 사용

### ◦ AFTER 트리거의 사용

- 삭제(Delete) 발생시 작동하는 backUserTbl\_DeleteTrg 트리거 생성

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



## 트리거의 사용

### ◦ AFTER 트리거의 사용

- 데이터 업데이트 및 삭제

```
UPDATE userTbl SET addr = '몽고' WHERE userID = 'JKW';  
DELETE FROM userTbl WHERE height >= 177;
```

- 수정 또는 삭제된 내용이 잘 보관되어 있는지 결과 확인

```
SELECT * FROM backup_userTbl;
```

	userID	name	birthYear	addr	mobile1	mobile2	height	mDate	modType	modDate	modUser
▶	JKW	조관우	1965	경기	018	9999999	172	2010-10-10	수정	2020-03-17	root@localhost
	KKH	김경호	1971	전남	019	3333333	177	2007-07-07	삭제	2020-03-17	root@localhost
	LJB	임재범	1963	서울	016	6666666	182	2009-09-09	삭제	2020-03-17	root@localhost
	LSG	이슬기	1987	서울	011	1111111	182	2008-08-08	삭제	2020-03-17	root@localhost
	SSK	성시경	1979	서울	NULL	NULL	186	2013-12-12	삭제	2020-03-17	root@localhost

## 트리거의 사용

### ◦ AFTER 트리거의 사용

- 테이블의 모든 행 데이터 삭제
- DELETE 대신 TRUNCATE TABLE문 사용

```
TRUNCATE TABLE userTbl;
```

- 백업 테이블 확인

```
SELECT * FROM backup_userTbl;
```

	userID	name	birthYear	addr	mobile1	mobile2	height	mDate	modType	modDate	modUser
▶	JKW	조관우	1965	경기	018	9999999	172	2010-10-10	수정	2020-03-17	root@localhost
	KKH	김경호	1971	전남	019	3333333	177	2007-07-07	삭제	2020-03-17	root@localhost
	LJB	임재범	1963	서울	016	6666666	182	2009-09-09	삭제	2020-03-17	root@localhost
	LSG	이승기	1987	서울	011	1111111	182	2008-08-08	삭제	2020-03-17	root@localhost
	SSK	성시경	1979	서울	NULL	NULL	186	2013-12-12	삭제	2020-03-17	root@localhost

- TRUNCATE TABLE로 삭제 시에는 트리거가 작동하지 않음, DELETE 트리거는 DELETE문에만 작동

## 트리거의 사용

- AFTER 트리거의 사용
  - INSERT 트리거를 생성

```
DROP TRIGGER IF EXISTS userTbl_InsertTrg;
DELIMITER //
CREATE TRIGGER userTbl_InsertTrg -- 트리거 이름
    AFTER INSERT -- 입력 후에 작동하도록 지정
    ON userTBL -- 트리거를 부착할 테이블
    FOR EACH ROW
BEGIN
    SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = '데이터의 입력을 시도했습니다. 귀하의 정보가 서버에 기록되었습니다.';
END //
DELIMITER ;
```

## 트리거의 사용

- AFTER 트리거의 사용

- 데이터 입력

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

Output				
Action Output				
	Time	Action	Message	Duration / Fetch
✖	1 20:11:44	INSERT INTO userT...	Error Code: 1644. 데이터의 입력을 시도했습니다. 귀하의 정보가 서버에 기록되었습니다.	0.000 sec

- 경고 메시지가 출력된 후에, INSERT 작업은 롤백이 되고 userTbl에는 데이터가 삽입되지 않음

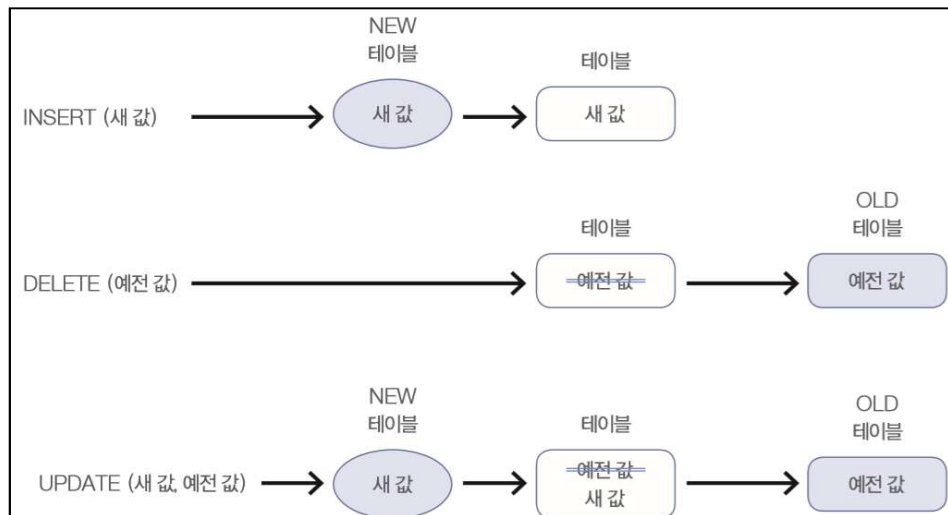
## 트리거의 사용

- AFTER 트리거의 사용
  - TRUNCATE TABLE 테이블이름
    - DELETE FROM 테이블이름문과 동일한 효과
    - DML문이 아니라 트리거를 작동시키지 않음
  - SIGNAL SQLSTATE '45000' 문
    - 사용자가 오류를 강제로 발생시키는 함수
    - 사용자가 정의한 오류 메시지 출력
    - 사용자가 시도한 INSERT는 롤백

## 트리거의 사용

### ◦ 트리거가 생성하는 임시 테이블

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



## 트리거의 사용

- BEFORE 트리거의 사용

- 테이블에 변경이 가해지기 전 작동
- BEFORE 트리거 활용 예
  - BEFORE INSERT 트리거를 부착해 놓으면 입력될 데이터 값을 미리 확인해서 문제가 있을 경우에 다른 값으로 변경
- BEFORE 트리거 실습
  - 값이 입력될 때, 출생년도의 데이터를 검사해서 데이터에 문제가 있으면 값을 변경시켜서 입력시키는 BEFORE INSERT 트리거 작성

## 트리거의 사용

- BEFORE 트리거의 사용
  - 트리거 생성

```
USE sqlDB;
DROP TRIGGER IF EXISTS userTbl_BeforeInsertTrg;
DELIMITER //
CREATE TRIGGER userTbl_BeforeInsertTrg  -- 트리거 이름
    BEFORE INSERT -- 입력 전에 작동하도록 지정
    ON userTBL -- 트리거를 부착할 테이블
    FOR EACH ROW
BEGIN
    IF NEW.birthYear < 1900 THEN
        SET NEW.birthYear = 0;
    ELSEIF NEW.birthYear > YEAR(CURDATE()) THEN
        SET NEW.birthYear = YEAR(CURDATE());
    END IF;
END //
DELIMITER ;
```



## 트리거의 사용

### ◦ BEFORE 트리거의 사용

- 값 입력 (두 값 모두 출생년도에 문제 있음)

```
INSERT INTO userTbl VALUES  
('AAA', '에이', 1877, '서울', '011', '1112222', 181, '2022-12-25');  
INSERT INTO userTbl VALUES  
('BBB', '비이', 2977, '경기', '011', '1113333', 171, '2019-3-25');
```

- SELECT \* FROM userTbl문으로 확인

	userID	name	birthYear	addr	mobile1	mobile2	height	mDate
▶	AAA	에이	0	서울	011	1112222	181	2022-12-25
	BBB	비이	2020	경기	011	1113333	171	2019-03-25
	BBK	바비킴	1973	서울	010	0000000	176	2013-05-05
	EJW	은지원	1972	경북	011	8888888	174	2014-03-03

## 트리거의 사용

- BEFORE 트리거의 사용

- SHOW TRIGGERS문으로 데이터베이스에 생성된 트리거 확인

```
SHOW TRIGGERS FROM sqlDB;
```

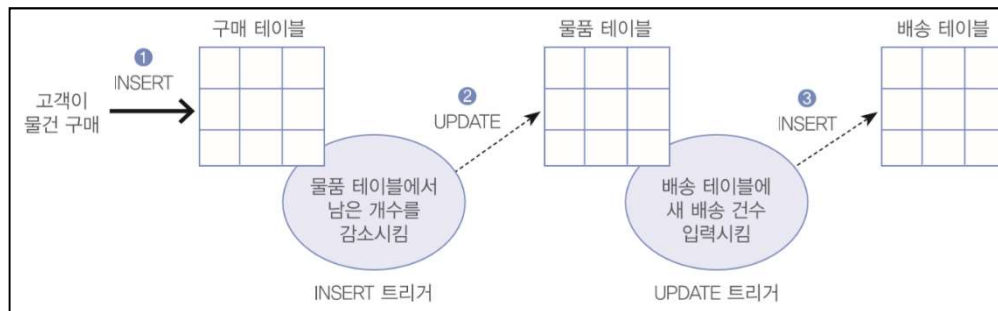
Trigger	Event	Table	Statement	Timing	Created	sql_mode
userTbl_BeforeInsertTrg	INSERT	usertbl	BEGIN IF NEW.birthYear < 1900 THEN SET NEW.birthY...	BEFORE	2020-03-17 21:48:45.30	STRICT_TRANS_TABLES,NO_AUTO_...

- 트리거 삭제

```
DROP TRIGGER userTbl_BeforeInsertTrg;
```

## 기타 트리거에 관한 내용

- 다중 트리거 (Multiple Triggers)
  - 하나의 테이블에 동일한 트리거가 여러 개 부착되어 있는 것
    - ex) AFTER INSERT 트리거 한 개 테이블에 2개 이상 부착
- 중첩 트리거 (Nested Triggers)
  - 트리거가 또 다른 트리거를 작동시키는 것



## 기타 트리거에 관한 내용

- 트리거의 작동 순서
  - 하나의 테이블에 여러 개의 트리거가 부착된 경우 트리거의 작동 순서 지정 가능

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

## 기타 트리거에 관한 내용

- 중첩 트리거 작동 실습
  - 연습용 DB 생성

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

## 기타 트리거에 관한 내용

- 중첩 트리거 작동 실습

- 테이블 생성

```
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 물건수량
CREATE TABLE deliverTbl -- 배송 테이블
    (deliverNo INT AUTO_INCREMENT PRIMARY KEY, -- 배송 일련번호
     prodName VARCHAR(5), -- 배송할 물건
     account INT UNIQUE); -- 배송할 물건개수
```

## 기타 트리거에 관한 내용

- 중첩 트리거 작동 실습
  - 데이터 입력

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

## 기타 트리거에 관한 내용

- 중첩 트리거 작동 실습

- 구매 테이블(orderTbl)과 물품 테이블(prodTbl)에 트리거 부착

```
-- 물품 테이블에서 개수를 감소시키는 트리거
DROP TRIGGER IF EXISTS orderTrg;
DELIMITER //
CREATE TRIGGER orderTrg -- 트리거 이름
    AFTER INSERT
    ON orderTBL -- 트리거를 부착할 테이블
    FOR EACH ROW
BEGIN
    UPDATE prodTbl SET account = account - NEW.orderamount
        WHERE prodName = NEW.prodName ;
END //
DELIMITER ;
```



## 트리거에 관한 내용

### ◦ 중첩 트리거 작동 실습

- 구매 테이블(orderTbl)과 물품 테이블(prodTbl)에 트리거 부착

```
-- 배송 테이블에 새 배송 건을 입력하는 트리거
DROP TRIGGER IF EXISTS prodTrg;
DELIMITER //
CREATE TRIGGER prodTrg -- 트리거 이름
    AFTER UPDATE
    ON prodTBL -- 트리거를 부착할 테이블
    FOR EACH ROW
BEGIN
    DECLARE orderAmount INT;
    -- 주문 개수 = (변경 전의 개수 - 변경 후의 개수)
    SET orderAmount = OLD.account - NEW.account;
    INSERT INTO deliverTbl(prodName, account)
        VALUES(NEW.prodName, orderAmount);
END //
DELIMITER ;
```

## 트리거에 관한 내용

### 중첩 트리거 작동 실습

- 고객이 구매한 데이터 입력

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

- 중첩 트리거가 잘 작동했는지 세 테이블 확인

```
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

## 트리거에 관한 내용

- 중첩 트리거 작동 실습

- 배송 테이블(deliverTbl)의 열 이름을 변경해서 (3)번의 INSERT가 실패하도록 실습

```
ALTER TABLE deliverTBL CHANGE prodName productName VARCHAR(5);
```

- 데이터 입력

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

오류 메시지:

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

## 트리거에 관한 내용

- 중첩 트리거 작동 실습
  - 테이블 확인

```
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

- 데이터가 변경되지 않았음.
- (3)번의 INSERT가 실패하면 (1)번 INSERT, (2)번 UPDATE 모두 롤백됨