```
/* 스토어드 프로시저
어떠한 동작을 일괄 처리하기 위한 쿼리문의 집합
모듈화하여 호출하여 사용
*/


USE sqlDB;
DROP PROCEDURE IF EXISTS userProc;


DELIMITER $$   -- delimiter 수정
CREATE PROCEDURE userProc()
BEGIN
    SELECT * FROM userTbl; -- 스토어드 프로시저 내용
END $$
DELIMITER ;

CALL userProc();

-- <실습 1> --
USE sqlDB;
DROP PROCEDURE IF EXISTS userProc1;
```

```
-- 매개변수
-- in 입력매개변수이름 데이터타입
-- call 프로시저이름(전달값)
-- out 출력매개변수이름 데이터타입

DELIMITER $$
CREATE PROCEDURE userProc1(IN userName VARCHAR(10))
BEGIN
   SELECT * FROM userTbl WHERE name = userName;
END $$
DELIMITER ;

CALL userProc1('조관우');



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);



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 ;



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);
```

```
DROP PROCEDURE IF EXISTS ifelseProc;
DELIMITER $$
CREATE PROCEDURE ifelseProc(
    IN userName VARCHAR(10)
)
BEGIN
    DECLARE bYear INT; -- 변수 선언
    SELECT birthYear into bYear FROM userTbl
        WHERE name = userName;
    IF (bYear >= 1980) THEN
            SELECT '아직 젊군요..';
    ELSE
            SELECT '나이가 지긋하네요..';
    END IF;
END $$
DELIMITER ;

CALL ifelseProc ('조용필');


DROP PROCEDURE IF EXISTS caseProc;
DELIMITER $$
CREATE PROCEDURE caseProc(
    IN userName VARCHAR(10)
```

```
)
BEGIN
    DECLARE bYear INT;
    DECLARE tti  CHAR(3);-- 띠
    SELECT birthYear INTO bYear FROM userTbl
        WHERE name = userName;
    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(userName, '의 띠 ==>', tti);
END $$
DELIMITER ;

CALL caseProc ('김범수');
```

```
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

    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;



DROP PROCEDURE IF EXISTS errorProc;
DELIMITER $$
CREATE PROCEDURE errorProc()
BEGIN
    DECLARE i INT; -- 1씩 증가하는 값
    DECLARE hap INT; -- 합계 (정수형). 오버플로 발생시킬 예정.
    DECLARE saveHap INT; -- 합계 (정수형). 오버플로 직전의 값을 저장.

    DECLARE EXIT HANDLER FOR 1264 -- INT형 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; -- 합계를 누적

    WHILE (TRUE) DO  -- 무한 루프.
        SET saveHap = hap; -- 오버플로 직전의 합계를 저장
        SET hap = hap + i;  -- 오버플로가 나면 11, 12행을 수행함
        SET i = i + 1;
    END WHILE;
```

```
END $$
DELIMITER ;

CALL errorProc();

SELECT * FROM INFORMATION_SCHEMA.ROUTINES;

SELECT routine_name, routine_definition FROM INFORMATION_SCHEMA.ROUTINES
    WHERE routine_schema = 'sqldb' AND routine_type = 'PROCEDURE';

SELECT param_list, body FROM MYSQL.PROC
    WHERE db='sqldb' AND type='PROCEDURE' AND name='userProc3';

-- SHOW CREATE PROCEDURE sqldb.delivProc;
SHOW CREATE PROCEDURE sqldb.userProc1;

DROP PROCEDURE IF EXISTS nameProc;
DELIMITER $$
CREATE PROCEDURE nameProc(
    IN tblName VARCHAR(20)
)
BEGIN
  SELECT * FROM tblName;
END $$
DELIMITER ;
```

CALL nameProc ('userTBL'); -- error, 테이블이름을 파라미터로 사용할 수 없다.

/* procedure 특징

1. mysql성능 향상

2. 유지관리 편리

3. 모듈식 프로그래밍

4. 보안을 강화 */

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

CALL delivProc ('LJB');
```

```
-- stored function
-- 사용자가 만든 함수

USE sqlDB;
DROP FUNCTION IF EXISTS userFunc;

DELIMITER $$

CREATE FUNCTION userFunc(value1 INT, value2 INT) -- in out 등 없이 모두 입력 파라미터
    RETURNS INT  -- 리턴의 데이터 형식
BEGIN
    RETURN value1 + value2; -- 하나의 리턴문, 프로시저는 out파라미터 사용
END $$
DELIMITER ;

SELECT userFunc(100, 200); -- call이 아니라 select로 실행



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 ;

SELECT getAgeFunc(1979);

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

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

SHOW CREATE FUNCTION getAgeFunc;  -- 현재 저장된 스토어드 함수

DROP FUNCTION getAgeFunc;



-- trigger
-- 테이블에서 삽입, 수정, 삭제 등의 작업이 발행할 경우 자동으로 작동되는 개체
-- 데이터의 무결성 확보
```

```
-- 트리거가 부착된 테이블에 이벤트가 발생하면 자동으로 부착된 트리거가 발동


drop database testdb;
CREATE DATABASE IF NOT EXISTS testDB;
USE testDB;
CREATE TABLE IF NOT EXISTS testTbl (id INT, txt VARCHAR(10));
INSERT INTO testTbl VALUES(1, '이엑스아이디');
INSERT INTO testTbl VALUES(2, '애프터스쿨');
INSERT INTO testTbl VALUES(3, '에이오에이');


DROP TRIGGER IF EXISTS testTrg;
DELIMITER //
CREATE TRIGGER testTrg   -- 트리거 이름
    AFTER   DELETE -- 삭제후에 작동하도록 지정
    ON testTbl -- 트리거를 부착할 테이블
    FOR EACH ROW -- 각 행마다 적용시킴
BEGIN
        SET @msg = '가수 그룹이 삭제됨' ; -- 트리거 실행시 작동되는 코드들
END //
DELIMITER ;


SET @msg = '';
INSERT INTO testTbl VALUES(4, '나인뮤지스');
SELECT @msg;
UPDATE testTbl SET txt = '에이핑크' WHERE id = 3;
SELECT @msg;
```

```
DELETE FROM testTbl WHERE id = 4;
SELECT @msg;
```

-- after trigger - insert, update, delete 등의 작업이 일어났을 때 작동하는 트리거

-- before trigger - 작업이 일어나기 전에 발생하는 트리거

-- 회원테이블에 update나 insert를 시도하면, 수정 또는 삭제된 데이터를 별도의 테이블에 보관하고 변경된

-- 일자와 변경한 사람을 기록한다

```
USE sqlDB;
DROP TABLE buyTbl; -- 구매테이블은 실습에 필요없으므로 삭제.
drop table backup_usertbl;
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
);

DROP TRIGGER IF EXISTS backUserTbl_UpdateTrg;
DELIMITER //
CREATE TRIGGER backUserTbl_UpdateTrg   -- 트리거 이름
    AFTER UPDATE -- 변경 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,
        '수정', CURDATE(), CURRENT_USER() );
END //
DELIMITER ;

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 ;

UPDATE userTbl SET addr = '몽고' WHERE userID = 'JKW';
DELETE FROM userTbl WHERE height >= 180;

SELECT * FROM backup_userTbl;

TRUNCATE TABLE userTbl;   -- delete trigger는 truncate에는 적용되지 않음

SELECT * FROM backup_userTbl;

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'   -- 사용자 오류를 강제로 발생시키는 함수, insert는 적용되지 않는다
        SET MESSAGE_TEXT = '데이터의 입력을 시도했습니다. 귀하의 정보가 서버에 기록되었습니다.';
END //
DELIMITER ;
```

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



-- new table - insert, update 시 새로운 데이터를 잠시 저장

-- olde table - 삭제 변경 전의 데이터를 저장


-- before trigger



USE sqlDB;

DROP TRIGGER IF EXISTS userTbl_BeforeInsertTrg;

DELIMITER //

CREATE TRIGGER userTbl_BeforeInsertTrg   -- 트리거 이름

    BEFORE INSERT -- 입력 전에 작동하도록 지정 - 값을 미리 확인

    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 ;


INSERT INTO userTbl VALUES
```

```
   ('AAA', '에이', 1877, '서울', '011', '1112222', 181, '2019-12-25');
INSERT INTO userTbl VALUES
   ('BBB', '비이', 2977, '경기', '011', '1113333', 171, '2011-3-25');

SHOW TRIGGERS FROM sqlDB;

DROP TRIGGER userTbl_BeforeInsertTrg;



-- 다중 trigger - 하나의 테이블에 여러 개의 trigger 부착
-- 중첩 trigger - 하나의 트리거가 다른 트리거를 작동


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

USE triggerDB;
CREATE TABLE orderTbl -- 구매 테이블
          (orderNo INT AUTO_INCREMENT PRIMARY KEY, -- 구매 일련번호
            userID VARCHAR(5), -- 구매한 회원아이디
           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

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

-- 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 ;

-- 배송테이블에 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 ;

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

SELECT * FROM orderTbl;
SELECT * FROM prodTbl;
SELECT * FROM deliverTbl;

SELECT * FROM orderTbl;
SELECT * FROM prodTbl;
SELECT * FROM deliverTbl;

ALTER TABLE deliverTBL CHANGE prodName productName VARCHAR(5);

ALTER TABLE deliverTBL CHANGE prodName productName VARCHAR(5);

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

```
SELECT * FROM orderTbl;

SELECT * FROM prodTbl;

SELECT * FROM deliverTbl;
```