



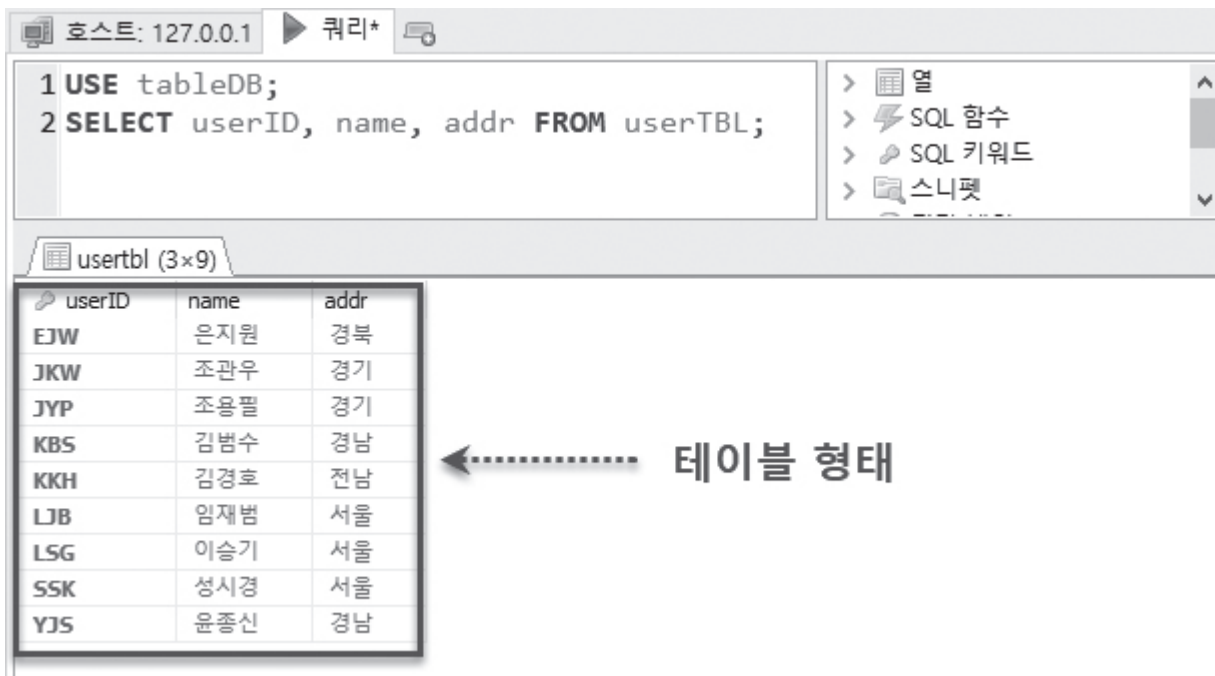
Ch 08. 테이블과 뷰

이것이 MariaDB다

8.2 뷰

8.2.1 뷰의 개념

- 일반 사용자 입장에서는 테이블과 동일하게 사용하는 개체
- 한 번 생성해 놓으면 테이블이라 생각하고 사용 가능
- 예시



The screenshot shows a SQL query editor window. The top bar indicates the host is 127.0.0.1 and the query is named '쿼리*'. The query text is:

```
1 USE tableDB;  
2 SELECT userID, name, addr FROM userTBL;
```

Below the query, the result set is displayed for the table 'usertbl' (3x9). The result set is a table with three columns: userID, name, and addr. The data is as follows:

userID	name	addr
EJW	은지원	경북
JKW	조관우	경기
JYP	조용필	경기
KBS	김범수	경남
KKH	김경호	전남
LJB	임재범	서울
LSG	이승기	서울
SSK	성시경	서울
YJS	윤종신	경남

An arrow points from the text '테이블 형태' (Table form) to the result set table.



8.2 뷰

8.2.1 뷰의 개념

■ 뷰 생성 구문과 쿼리 결과

```
USE tableDB;
```

```
CREATE VIEW v_userTBL
```

```
AS
```

```
    SELECT userid, name, addr FROM userTBL;
```

```
SELECT * FROM v_userTBL;
```

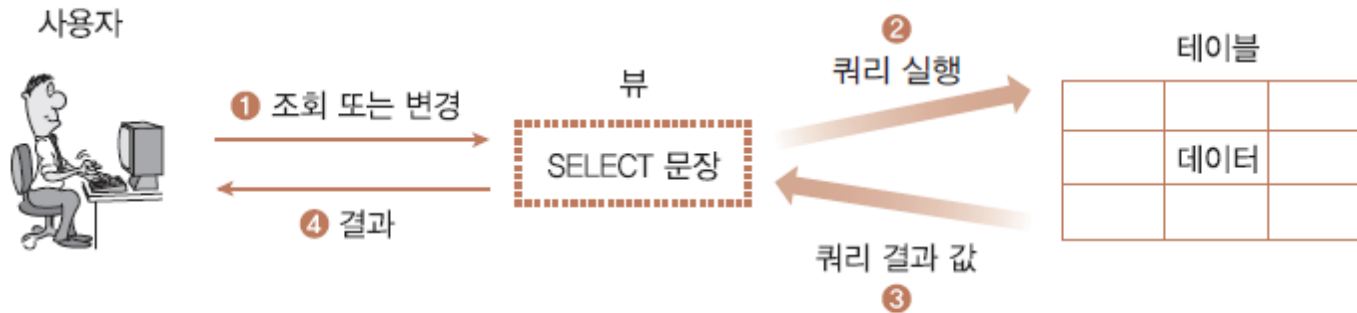
v_userTBL (3×9)		
userid	name	addr
EJW	은지원	경북
JKW	조관우	경기
JYP	조용필	경기
KBS	김범수	경남
KKH	김경호	전남
LJB	임재범	서울
LSG	이승기	서울
SSK	성시경	서울
YJS	윤종신	경남



8.2 뷰

8.2.1 뷰의 개념

■ 뷰의 작동 방식



- 뷰는 보통 읽기 전용으로 사용
- 뷰를 통해 원래 테이블의 데이터를 수정하는 것도 가능



8.2 뷰

8.2.1 뷰의 장점

- 보안에 도움
 - 간단한 확인 작업을 위한 뷰
 - 중요 개인 정보는 열람 금지 되어 데이터 안전
- 복잡한 쿼리 단순화
 - 긴 쿼리를 자주 사용할 경우 뷰 생성
 - 입력할 필요가 없어 시간 절약되며 오류도 줄임



8.2 뷰

8.2.1 뷰

```
CREATE VIEW v_userbuyTBL
AS
SELECT U.userid, U.name, B.prodName, U.addr,
CONCAT(U.mobile1, U.mobile2) AS '연락처'
FROM userTBL U
    INNER JOIN buyTBL B
    ON U.userid = B.userid ;

SELECT * FROM v_userbuyTBL WHERE name = '김범수';
```



8.2 뷰

8.2.1 뷰

```
USE sqlDB;
CREATE VIEW v_userbuyTBL
AS
    SELECT U.userid AS 'USER ID', U.name AS 'USER NAME',
    B.prodName AS 'PRODUCT NAME',
        U.addr, CONCAT(U.mobile1, U.mobile2) AS 'MOBILE PHONE'
    FROM userTBL U
    INNER JOIN buyTBL B
    ON U.userid = B.userid;

SELECT `USER ID`, `USER NAME` FROM v_userbuyTBL;
```



8.2 뷰

8.2.1 뷰

```
CREATE VIEW v_userbuyTBL
AS
    SELECT U.userid, U.name, B.prodName, U.addr, CONCAT(U.mobile1, U.mobile2) AS
mobile
    FROM userTBL U
        INNER JOIN buyTBL B
            ON U.userid = B.userid ;

INSERT INTO v_userbuyTBL
VALUES('PKL', '박경리', '운동화', '경기', '00000000000', '2023-2-2');

DROP TABLE IF EXISTS buyTBL, userTBL;

SELECT * FROM v_userbuyTBL;
```



10.4 트리거

❖ 트리거(Trigger)란?

- 트리거는 사전적 의미로 ‘방아쇠’
 - 방아쇠를 당기면 ‘자동’ 으로 총알이 나가듯이, 트리거는 테이블에 무슨 일이 일어나면 ‘자동’ 으로 실행
- 테이블에 삽입, 수정, 삭제 등의 작업(이벤트)이 발생할 때 자동으로 작동되는 개체
- Ex) 누군가 A라는 테이블에 행을 고의 또는 실수로 삭제
 - A테이블에서 행이 삭제되는 순간에 삭제된 행의 내용, 시간, 삭제한 사용자 등을 B테이블에 기록
 - 추후 문제 발생 시에는 B테이블의 내용 확인



10.4 트리거

10.4.1 트리거의 개요

- 트리거는 제약 조건과 더불어 데이터 무결성을 위해 MariaDB에서 사용할 수 있는 또 다른 기능
- 트리거는 테이블에 관련되어 DML문(Insert, Update, Delete 등)의 이벤트가 발생할 때 작동하는 데이터베이스 개체 중 하나
- 트리거는 테이블에 부착(Attach)되는 프로그램 코드
 - 스토어드 프로시저와 비슷한 문법으로 내용 작성
- 트리거가 부착된 테이블에 이벤트(입력, 수정, 삭제) 발생 하면 자동으로 부착된 트리거 실행



10.4 트리거

10.4.2 트리거의 종류

■ AFTER 트리거

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

■ BEFORE 트리거

- 이벤트가 발생하기 전에 작동하는 트리거
- AFTER 트리거와 마찬가지로 INSERT, UPDATE, DELETE 세 가지 이벤트로 작동



10.4 트리거

10.4.3 트리거의 사용

■ 트리거의 형식

형식 :

```
CREATE [OR REPLACE]
  [DEFINER = { user | CURRENT_USER | role | CURRENT_ROLE }]
  TRIGGER [IF NOT EXISTS] trigger_name trigger_time trigger_event
  ON tbl_name FOR EACH ROW
  [{ FOLLOWS | PRECEDES } other_trigger_name ]
  trigger_stmt
```

trigger_time: { BEFORE | AFTER }

trigger_event: { INSERT | UPDATE | DELETE }

trigger_order: { FOLLOWS | PRECEDES } other_trigger_name



10.4 트리거

10.4.3 트리거의 사용

- 트리거가 생성하는 임시 테이블

- INSERT, UPDATE, DELETE 작업이 수행되면 임시로 사용되는 시스템 테이블인 OLD, NEW



10.4 트리거

10.4.1 트리거 예제

```
USE sqlDB;
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;
```



10.4 트리거

10.4.1 트리거 예제

```
USE sqlDB;
```

```
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
);
```



10.4 트리거

10.4.1 트리거 예제

```
USE sqlDB;
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 ;
```



10.4 트리거

10.4.1 트리거 예제

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

```
SELECT * FROM backup_userTBL;
```

```
TRUNCATE TABLE userTBL;
```

```
SELECT * FROM backup_userTBL;
```



10.4 트리거

10.4.1 트리거 예제

```
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 ;
```



10.4 트리거

10.4.1 트리거 예제

```
-- 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;

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;
```

