

# 데이터의 검색

## 여러 테이블에 대한 조인 검색

### 조인 검색 규칙

테이블의 이름과 속성의 이름은 . 기호로 연결함 | 속성 이름이 유니크하면 테이블명 생략 가능

### 예제 7-37

판매 데이터베이스에서 banana 고객이 주문한 제품의 이름을 검색해보자.

```
▶▶ SELECT   제품.제품명
FROM       제품, 주문
WHERE      주문.주문고객 = 'banana' AND 제품.제품번호 = 주문.주문제품;
```

결과 테이블

	제품명
1	그냥만두
2	맛난초콜릿
3	통통우동

# 데이터의 검색

## 조인 검색 – 도메인 일치

조인 속성의 이름은 달라도 되지만 도메인은 반드시 같아야 함 (외래키를 조인 속성으로 활용)



그림 7-11 2개의 테이블을 이용한 조인 검색 예: 주문 테이블과 제품 테이블

# 데이터의 검색

## 조인 검색 (예제)

조인 속성의 이름은 달라도 되지만 도메인은 반드시 같아야 함

### 예제 7-38

판매 데이터베이스에서 나이가 30세 이상인 고객이 주문한 제품의 번호와 주문일자를 검색해보자.

```
▶▶ SELECT   주문.주문제품, 주문.주문일자
FROM       고객, 주문
WHERE      고객.나이 >= 30 AND 고객.고객아이디 = 주문.주문고객;
```

결과 테이블

	주문제품	주문일자
1	p01	22/01/10
2	p06	22/02/20
3	p02	22/04/10

# 데이터의 검색

## 조인 검색 — 테이블 Alias

### 테이블 alias를 사용하면 좋음

```
SELECT c.고객이름, o.주문제품, o.수량  
FROM 고객 c, 주문 o  
WHERE c.고객아이디 = o.주문고객;
```

→ 테이블에 짧은 별칭을 부여하여 코드를 간결하게 작성

# 데이터의 검색

## 조인 검색 — Alias 예제

테이블 alias를 사용하면 좋음

### 예제 7-39

판매 데이터베이스에서 고명석 고객이 주문한 제품의 제품명을 검색해보자.

```
▶▶ SELECT   제품.제품명
FROM       고객, 제품, 주문
WHERE      고객.고객이름 = '고명석' AND 고객.고객아이디 = 주문.주문고객
           AND 제품.제품번호 = 주문.주문제품;
```

결과 테이블

	제품명
1	매운쫄면
2	콩떡파이

# 데이터의 검색

## 조인 검색 — INNER JOIN

### 표준 SQL에서는 INNER JOIN, ON 키워드를 사용함

```
SELECT 고객.고객이름, 주문.주문제품, 주문.수량  
FROM 고객  
INNER JOIN 주문  
ON 고객.고객아이디 = 주문.주문고객;
```

- WHERE 절 조인과 동일한 결과
- 표준 SQL 방식으로 더 명확함

# 데이터의 검색

## 조인 검색 — OUTER JOIN

조인 조건을 만족하지 않을 시 OUTER JOIN, ON 키워드를 사용함

```
SELECT 고객.고객이름, 주문.주문제품, 주문.수량  
FROM 고객  
LEFT OUTER JOIN 주문  
ON 고객.고객아이디 = 주문.주문고객;
```

→ 매칭 안 되는 튜플도 결과에 포함 (NULL로 채움)

→ LEFT / RIGHT / FULL OUTER JOIN

# 데이터의 검색

## LEFT OUTER JOIN 예시

```
SELECT  고객.고객이름, 주문.주문제품, 주문.주문일자
FROM    고객 LEFT OUTER JOIN 주문 ON  고객.고객아이디 = 주문.주문고객;
```

결과 테이블

	고객이름	주문제품	주문일자
1	정소화	p03	2022-01-01
2	정소화	p03	2022-03-15
3	김선우	p06	2022-01-11
4	김선우	p01	2022-03-02
5	김선우	p04	2022-04-11
6	고명석	p02	2022-02-01
7	고명석	p03	2022-05-22
8	김용욱	(null)	(null)
9	성원용	p01	2022-01-10
10	성원용	p06	2022-02-20
11	오형준	(null)	(null)
12	채광주	p02	2022-04-10

# 데이터의 검색

## RIGHT OUTER JOIN 예시

### RIGHT OUTER JOIN

```
SELECT 고객.고객이름, 주문.주문제품, 주문.수량  
FROM 주문  
RIGHT OUTER JOIN 고객  
ON 주문.주문고객 = 고객.고객아이디;
```

앞 장의 LEFT OUTER JOIN과 결과가 같게 나올까? (O)  
→ 테이블 순서를 바꾸고 방향을 바꾸면 동일한 결과

# 데이터의 검색

## RIGHT OUTER JOIN 예시 (계속)

### RIGHT OUTER JOIN 결과

앞 장의 LEFT OUTER JOIN과 결과가 같게 나올까? (O)

→ LEFT OUTER JOIN에서 고객을 왼쪽에 놓은 것과  
RIGHT OUTER JOIN에서 고객을 오른쪽에 놓은 것은  
동일한 결과를 반환합니다.

# 실습 ① INNER JOIN vs LEFT OUTER JOIN

STEP 0 판매 DB가 세팅되어 있어야 합니다 (섹션2 실습 참고)

STEP 1 INNER JOIN – 매칭되는 것만

```
SELECT c.고객이름, o.주문제품, o.수량
FROM 고객 c
INNER JOIN 주문 o
ON c.고객아이디 = o.주문고객;
```

STEP 2 LEFT OUTER JOIN – 따로 실행! 행 수 비교!

```
SELECT c.고객이름, o.주문제품, o.수량
FROM 고객 c
LEFT OUTER JOIN 주문 o
ON c.고객아이디 = o.주문고객;
```

## ▶ 예상 결과

INNER JOIN: 주문이 있는 고객만 (10행)  
LEFT JOIN: 모든 고객 포함 – 주문 없는 고객도!  
→ 주문 없는 고객의 주문제품/수량은 NULL

STEP 3 WHERE 방식 조인 (결과는 INNER JOIN과 동일)

```
SELECT c.고객이름, o.주문제품, o.수량
FROM 고객 c, 주문 o
WHERE c.고객아이디 = o.주문고객;
```

# 데이터의 검색

## 부속 질의문을 이용한 검색

### 부속 질의문(nested query) 또는 서브 질의문(sub query)

부속 질의문을 먼저 수행하고, 주 질의문(main query)을 나중에 수행함  
단일 행 부속 질의문은 비교 연산자 사용 가능, 다중 행은 불가

#### 예제 7-40

판매 데이터베이스에서 달콤비스킷을 생산한 제조업체가 만든 제품들의 제품명과 단가를 검색해보자.

```
▶▶ { SELECT 제품명, 단가
    ② { FROM 제품
      WHERE 제조업체 = (SELECT 제조업체 ①
                        FROM 제품
                        WHERE 제품명 = '달콤비스킷');
```

결과 테이블

	제품명	단가
1	쿵떡파이	2600
2	맛난초콜릿	2500
3	달콤비스킷	1500

# 데이터의 검색

## 부속 질의문 – 집계 함수 사용

### 예제 7-41

판매 데이터베이스에서 적립금이 가장 많은 고객의 고객이름과 적립금을 검색해보자.

```
▶▶ SELECT   고객이름, 적립금
   FROM     고객
   WHERE    적립금 = (SELECT MAX(적립금) FROM 고객);
```

결과 테이블

	고객이름	적립금
1	성원용	5000

= 5000

# 데이터의 검색

## 부속 질의문을 이용 가능한 연산자

표 7-7 다중 행 부속 질의문에 사용 가능한 연산자

연산자	설명
IN	부속 질의문의 결과 값 중 일치하는 것이 있으면 검색 조건이 참
NOT IN	부속 질의문의 결과 값 중 일치하는 것이 없으면 검색 조건이 참
EXISTS	부속 질의문의 결과 값이 하나라도 존재하면 검색 조건이 참
NOT EXISTS	부속 질의문의 결과 값이 하나도 존재하지 않으면 검색 조건이 참
ALL	부속 질의문의 결과 값 모두와 비교한 결과가 참이면 검색 조건을 만족(비교 연산자와 함께 사용)
ANY 또는 SOME	부속 질의문의 결과 값 중 하나라도 비교한 결과가 참이면 검색 조건을 만족(비교 연산자와 함께 사용)

# 데이터의 검색

## IN 연산자 예제

### 예제 7-42

판매 데이터베이스에서 banana 고객이 주문한 제품의 제품명과 제조업체를 검색해보자.

```
▶▶ SELECT   제품명, 제조업체
FROM       제품
WHERE      제품번호 IN (SELECT 주문제품
                        FROM   주문
                        WHERE  주문고객 = 'banana');
```

결과 테이블

	제품명	제조업체
1	그냥만두	대한식품
2	맛난초콜릿	한빛제과
3	통통우동	민국푸드

# 데이터의 검색

## 부속 질의문 예제

### 예제 7-43

판매 데이터베이스에서 banana 고객이 주문하지 않은 제품의 제품명과 제조업체를 검색해보자.

```
▶▶ SELECT   제품명, 제조업체
   FROM     제품
   WHERE    제품번호 NOT IN (SELECT 주문제품
                               FROM   주문
                               WHERE  주문고객 = 'banana');
```

결과 테이블

	제품명	제조업체
1	콩떡파이	한빛제과
2	매운짬면	민국푸드
3	얼큰라면	대한식품
4	달콤비스킷	한빛제과

# 데이터의 검색

## 부속 질의문 예제

### 예제 7-44

판매 데이터베이스에서 대한식품이 제조한 모든 제품의 단가보다 비싼 제품의 제품명, 단가, 제조업체를 검색해보자.

```
▶▶ SELECT   제품명, 단가, 제조업체
FROM       제품
WHERE      단가 > ALL (SELECT  단가
                       FROM     제품
                       WHERE    제조업체 = '대한식품');
```

결과 테이블

	제품명	단가	제조업체
1	매운짬면	5500	민국푸드

# 데이터의 검색

## 부속 질의문 연산자 예제

### 예제 7-45

판매 데이터베이스에서 2022년 3월 15일에 제품을 주문한 고객의 고객이름을 검색해보자.

```
▶▶ SELECT   고객이름
FROM       고객
WHERE      EXISTS (SELECT *
                  FROM   주문
                  WHERE  주문일자 = '2022-03-15'
                  AND   주문.주문고객 = 고객.고객아이디);
```

결과 테이블

	고객이름
1	정소화

# 데이터의 검색

## 부속 질의문 예제

### 예제 7-46

판매 데이터베이스에서 2022년 3월 15일에 제품을 주문하지 않은 고객의 고객이름을 검색해보자.

```
▶▶ SELECT   고객이름
   FROM     고객
   WHERE    NOT EXISTS (SELECT *
                        FROM   주문
                        WHERE  주문일자 = '2022-03-15'
                        AND   주문.주문고객 = 고객.고객아이디);
```

결과 테이블

	고객이름
1	채광주
2	성원용
3	고명석
4	오형준
5	김선우
6	김용욱

# 데이터의 검색

## 부속 질의문 — 조인 질의 / EXISTS

### 조인 질의 / EXISTS 연산자

#### ① 조인 질의를 이용한 SELECT 문

```
SELECT 제품.제품명, 제품.제조업체
FROM   제품, 주문
WHERE  제품.제품번호 = 주문.주문제품 AND  주문.주문고객 = 'banana';
```

#### ② EXISTS 연산자를 이용한 SELECT 문

```
SELECT 제품명, 제조업체
FROM   제품
WHERE  EXISTS (SELECT *
                FROM   주문
                WHERE  제품.제품번호 = 주문.주문제품
                AND   주문.주문고객 = 'banana');
```

[예제 7-42]

```
SELECT 제품명, 제조업체
FROM   제품
WHERE  제품번호 IN (SELECT 주문제품
                    FROM   주문
                    WHERE  주문고객 = 'banana');
```

## 실습 ② 부속 질의문 (서브쿼리)

### STEP 1 단일 행 서브쿼리 -- 사용 가능

```
SELECT 제품명, 단가 FROM 제품
WHERE 단가 = (SELECT MAX(단가) FROM 제품);
-- 가장 비싼 제품 검색
```

### STEP 2 다중 행 서브쿼리 -- IN 사용

```
SELECT 제품명, 단가 FROM 제품
WHERE 제품번호 IN
  (SELECT 주문제품 FROM 주문
   WHERE 주문고객 = 'banana');
-- banana가 주문한 제품의 이름과 단가
```

### STEP 3 NOT IN -- 주문 안 한 고객 찾기

```
SELECT 고객이름 FROM 고객
WHERE 고객아이디 NOT IN
  (SELECT 주문고객 FROM 주문);
-- 주문을 한 번도 안 한 고객
```

### STEP 4 다중 행에 = 쓰면? -- 에러!

```
SELECT * FROM 제품 WHERE 제품번호 =
  (SELECT 주문제품 FROM 주문); -- 에러!
```

### X 예상 결과: 에러!

다중 행 반환에 = 사용 불가! → IN을 써야!

# 데이터의 삽입

## 테이블에 튜플을 직접 삽입하는 방법

### INSERT 규칙

- 속성 리스트와 속성값 리스트의 개수는 같아야 함
- 속성 리스트 순서는 테이블의 속성 순서와 반드시 일치하지 않아도 됨
- 문자나 날짜는 작은따옴표로 묶어야 함

INSERT

INTO 테이블\_이름[(속성\_리스트)]

VALUES (속성값\_리스트);

예제 7-47

판매 데이터베이스의 고객 테이블에 고객아이디가 strawberry, 고객이름이 최유경, 나이가 30세, 등급이 vip, 직업이 공무원, 적립금이 100원인 새로운 고객의 정보를 삽입해보자. 그런 다음 고객 테이블에 있는 모든 내용을 검색하여 삽입된 새로운 튜플을 확인해보자.

```

▶▶ INSERT
INTO   고객(고객아이디, 고객이름, 나이, 등급, 직업, 적립금)
VALUES ('strawberry', '최유경', 30, 'vip', '공무원', 100);

```

SELECT \* FROM 고객;

결과 테이블

	고객아이디	고객이름	나이	등급	직업	적립금
1	apple	정소화	20	gold	학생	1000
2	banana	김신우	25	vip	간호사	2500
3	carrot	고명석	28	gold	교사	4500
4	orange	김용욱	22	silver	학생	0
5	melon	성원윤	35	gold	회사원	5000
6	peach	오형준	(null)	silver	의사	300
7	pear	채광주	31	silver	회사원	500
8	strawberry	최유경	30	vip	공무원	100

# 데이터의 삽입

## INSERT 예제

### 예제 7-47

판매 데이터베이스의 고객 테이블에 고객아이디가 strawberry, 고객이름이 최유경, 나이가 30세, 등급이 vip, 직업이 공무원, 적립금이 100원인 새로운 고객의 정보를 삽입해보자. 그런 다음 고객 테이블에 있는 모든 내용을 검색하여 삽입된 새로운 튜플을 확인해보자.

#### ▶▶ INSERT

```
INTO      고객(고객아이디, 고객이름, 나이, 등급, 직업, 적립금)
VALUES    ('strawberry', '최유경', 30, 'vip', '공무원', 100);
```

```
SELECT * FROM 고객;
```

결과 테이블

	고객아이디	고객이름	나이	등급	직업	적립금
1	apple	정소화	20	gold	학생	1000
2	banana	김선우	25	vip	간호사	2500
3	carrot	고명석	28	gold	교사	4500
4	orange	김용욱	22	silver	학생	0
5	melon	성원용	35	gold	회사원	5000
6	peach	오형준	(null)	silver	의사	300
7	pear	채광주	31	silver	회사원	500
8	strawberry	최유경	30	vip	공무원	100

# 데이터의 삽입

## INSERT — 속성 리스트 생략

고객 테이블의 속성 순서와 동일하면 생략해도 됨

예제 7-47

판매 데이터베이스의 고객 테이블에 고객아이디가 strawberry, 고객이름이 최유경, 나이가 30세, 등급이 vip, 직업이 공무원, 적립금이 100원인 새로운 고객의 정보를 삽입해보자. 그런 다음 고객 테이블에 있는 모든 내용을 검색하여 삽입된 새로운 부품을 확인해보자.

```
▶ INSERT
INTO   고객(고객아이디, 고객이름, 나이, 등급, 직업, 적립금)
VALUES ('strawberry', '최유경', 30, 'vip', '공무원', 100);

SELECT * FROM 고객;
```

결과 테이블

	고객아이디	고객이름	나이	등급	직업	적립금
1	apple	정소화	20	gold	학생	1000
2	banana	김선우	25	vip	간호사	2500
3	carrot	고명석	28	gold	교사	4500
4	orange	김용욱	22	silver	학생	0
5	melon	성원윤	35	gold	회사원	5000
6	peach	오형준	(null)	silver	의사	300
7	pear	채광주	31	silver	회사원	500
8	strawberry	최유경	30	vip	공무원	100

```
INSERT
INTO   고객
VALUES ('strawberry', '최유경', 30, 'vip', '공무원', 100);
```

# 데이터의 삽입

## INSERT — 일부 속성만 삽입

직업 속성을 빼고 넣음 / NULL을 직접 넣어도 됨

예제 7-48

판매 데이터베이스의 고객 테이블에 고객아이디가 tomato, 고객이름이 정은심, 나이가 36세, 등급이 gold, 적립금은 4,000원, 직업은 아직 모르는 새로운 고객의 정보를 삽입해보자. 그런 다음 고객 테이블에 있는 모든 내용을 검색하여, 삽입된 정은심 고객의 직업 속성이 널 값인지 확인해보자.

```
▶▶ INSERT
INTO   고객(고객아이디, 고객이름, 나이, 등급, 적립금)
VALUES ('tomato', '정은심', 36, 'gold', 4000);
```

```
SELECT * FROM 고객;
```

결과 테이블	고객아이디	고객이름	나이	등급	직업	적립금
1	apple	정소화	20	gold	학생	1000
2	banana	김선우	25	vip	간호사	2500
3	carrot	고명석	28	gold	교사	4500
4	orange	김용욱	22	silver	학생	0
5	melon	성원용	35	gold	회사원	5000
6	peach	오형준	(null)	silver	의사	300
7	pear	채광수	31	silver	회사원	500
8	strawberry	최유경	30	vip	공무원	100
9	tomato	정은심	36	gold	(null)	4000

```
INSERT
INTO   고객
VALUES ('tomato', '정은심', 36, 'gold', NULL, 4000);
```

# 데이터의 삽입

## 부속 질의문을 이용한 데이터 삽입

```
INSERT  
INTO 테이블_이름[(속성_리스트)]  
SELECT 문;
```

### 예) INSERT

```
INTO 한빛제품(제품명, 재고량, 단가)  
SELECT 제품명, 재고량, 단가  
FROM 제품  
WHERE 제조업체 = '한빛제과';
```

한빛제과에서 제조한 제품의 제품명, 재고량, 단가를  
제품 테이블에서 검색하여 한빛제품 테이블에 삽입함

## 실습 ③ INSERT 삽입

### STEP 1 기본 삽입 – 속성 리스트 명시

```
INSERT INTO 고객(고객아이디, 고객이름, 나이, 등급, 직업, 적립금)
VALUES ('tomato', '정현수', 36, 'gold', '교수', 0);
SELECT * FROM 고객 WHERE 고객아이디 = 'tomato';
```

### STEP 2 속성 리스트 생략 – 테이블 순서대로

```
INSERT INTO 고객
VALUES ('grape', '안창호', 29, 'silver', '개발자', 100);
SELECT * FROM 고객 WHERE 고객아이디 = 'grape';
```

### STEP 3 일부 속성만 – 나머지는 NULL/DEFAULT

```
INSERT INTO 고객(고객아이디, 고객이름, 등급)
VALUES ('kiwi', '이순신', 'vip');
SELECT * FROM 고객 WHERE 고객아이디 = 'kiwi';
-- 나이=NULL, 직업=NULL, 적립금=0(DEFAULT)
```

### STEP 4 정리 – 추가한 데이터 삭제

```
DELETE FROM 고객 WHERE 고객아이디 IN ('tomato', 'grape', 'kiwi');
```

# 데이터의 수정

## WHERE 조건문을 이용한 데이터 수정

주의: 바로 업데이트하기 전에 반드시 SELECT로 확인해야 함

```
UPDATE 테이블_이름  
SET   속성_이름1 = 값1, 속성_이름2 = 값2, ...  
[WHERE 조건];
```

### 예제 7-49

제품 테이블에서 제품번호가 p03인 제품의 제품명을 통큰파이로 수정해보자.

# 데이터의 수정

## UPDATE 예제

바로 업데이트하기 전에 반드시 확인해야 함

### 예제 7-49

제품 테이블에서 제품번호가 p03인 제품의 제품명을 통큰파이로 수정해보자.

```
▶▶ UPDATE   제품
SET         제품명 = '통큰파이'   결과 테이블
WHERE      제품번호 = 'p03';

SELECT * FROM 제품;
```

	제품번호	제품명	재고량	단가	제조업체
1	p01	그냥만두	5000	4500	대한식품
2	p02	매운짬면	2500	5500	민국푸드
3	p03	통큰파이	3600	2600	한빛제과
4	p04	맛난초콜릿	1250	2500	한빛제과
5	p05	얼큰라면	2200	1200	대한식품
6	p06	통통우동	1000	1550	민국푸드
7	p07	달콤비스킷	1650	1500	한빛제과

# 데이터의 수정

## UPDATE 예제 (계속)

### 예제 7-50

제품 테이블에 있는 모든 제품의 단가를 10% 인상해보자. 그런 다음 제품 테이블의 모든 내용을 검색하여 인상 내용을 확인해보자.

```
▶▶ UPDATE   제품
   SET       단가 = 단가 * 1.1;   결과 테이블

SELECT * FROM 제품;
```

	제품번호	제품명	재고량	단가	제조업체
1	p01	그냥만두	5000	4950	대한식품
2	p02	매운짬면	2500	6050	민국푸드
3	p03	통큰파이	3600	2860	한빛제과
4	p04	맛난초콜릿	1250	2750	한빛제과
5	p05	얼큰라면	2200	1320	대한식품
6	p06	통통우동	1000	1705	민국푸드
7	p07	달콤비스킷	1650	1650	한빛제과

# 데이터의 수정

## UPDATE — 서브쿼리 활용

WHERE 조건문의 값에 대해 SELECT문을 이용한 데이터 수정

### 예제 7-51

판매 데이터베이스에서 정소화 고객이 주문한 제품의 주문수량을 5개로 수정해보자. 그런 다음 주문 테이블의 모든 내용을 검색하여 수정 내용을 확인해보자.

```
▶ UPDATE 주문
SET 수량 = 5
WHERE 주문고객 IN (SELECT 고객아이디
                   FROM 고객
                   WHERE 고객이름 = '정소화');
```

SELECT \* FROM 주문;

결과 테이블

	주문번호	주문고객	주문제품	수량	배송지	주문일자
1	o01	apple	p03	5	서울시 마포구	22/01/01
2	o02	melon	p01	5	인천시 계양구	22/01/10
3	o03	banana	p06	45	경기도 부천시	22/01/11
4	o04	carrot	p02	8	부산시 금정구	22/02/01
5	o05	melon	p06	36	경기도 용인시	22/02/20
6	o06	banana	p01	19	충청북도 보은군	22/03/02
7	o07	apple	p03	5	서울시 영등포구	22/03/15
8	o08	pear	p02	50	강원도 춘천시	22/04/10
9	o09	banana	p04	15	전라남도 목포시	22/04/11
10	o10	carrot	p03	20	경기도 안양시	22/05/22

## 실습 ④ UPDATE 수정

### STEP 1 특정 튜플 수정 – WHERE 필수!

```
SELECT * FROM 제품 WHERE 제품번호 = 'p04'; -- 변경 전 확인
UPDATE 제품 SET 단가 = 600 WHERE 제품번호 = 'p04';
SELECT * FROM 제품 WHERE 제품번호 = 'p04'; -- 변경 후 확인
```

### STEP 2 여러 속성 동시 수정

```
UPDATE 제품 SET 재고량 = 2000, 단가 = 550
  WHERE 제품번호 = 'p04';
SELECT * FROM 제품 WHERE 제품번호 = 'p04';
```

### STEP 3 WHERE 없으면? – 전체 수정! (위험!)

```
-- 주의: 아래는 모든 제품의 단가가 바뀜!
-- UPDATE 제품 SET 단가 = 1000;
-- ★ WHERE 빠뜨리면 전체 튜플이 수정됨!
```

### X 예상 결과: 에러!

WHERE 없는 UPDATE = 전체 행 수정!  
실무에서 가장 위험한 실수 중 하나!

### STEP 4 원래 값으로 복원

```
UPDATE 제품 SET 재고량=1250, 단가=500 WHERE 제품번호='p04';
```

# 데이터의 삭제

## WHERE 조건문을 이용한 데이터 삭제

### 삭제는 충분히 검증하고 실행해야 함

조인에 의해 특정 테이블 삭제는 충분히 쿼리문을 돌려서 삭제 후 데이터 상황을 검증해야 함

```
DELETE  
FROM 테이블_이름  
[WHERE 조건];
```

# 데이터의 삭제

## DELETE 예제

### 예제 7-52

주문 테이블에서 주문일자가 2022년 5월 22일인 주문 내역을 삭제해보자. 그런 다음 주문 테이블의 모든 내용을 검색하여 삭제 여부를 확인해보자.

▶▶ DELETE

```
FROM 주문
WHERE 주문일자 = '2022-05-22';
```

```
SELECT * FROM 주문;
```

결과 테이블

	주문번호	주문고객	주문제품	수량	배송지	주문일자
1	o01	apple	p03	5	서울시 마포구	22/01/01
2	o02	melon	p01	5	인천시 계양구	22/01/10
3	o03	banana	p06	45	경기도 부천시	22/01/11
4	o04	carrot	p02	8	부산시 금정구	22/02/01
5	o05	melon	p06	36	경기도 용인시	22/02/20
6	o06	banana	p01	19	충청북도 보은군	22/03/02
7	o07	apple	p03	5	서울시 영등포구	22/03/15
8	o08	pear	p02	50	강원도 춘천시	22/04/10
9	o09	banana	p04	15	전라남도 목포시	22/04/11

# 데이터의 삭제

## DELETE — 서브쿼리 활용

WHERE 조건문의 값에 대해 SELECT문을 이용한 데이터 삭제

### 예제 7-53

판매 데이터베이스에서 정소화 고객이 주문한 내역을 주문 테이블에서 삭제해보자. 그런 다음 주문 테이블의 모든 내용을 검색하여 삭제 여부를 확인해보자.

```
▶▶ DELETE
FROM      주문
WHERE     주문고객 IN (SELECT  고객아이디
                        FROM      고객
                        WHERE     고객이름 = '정소화');

SELECT * FROM 주문;
```

# 데이터의 삭제

## DELETE — 전체 삭제

### WHERE 조건문 없이 데이터 삭제하는 방법

테이블 구조는 남아 있음 (데이터는 모두 삭제됨)

#### 예제 7-54

판매 데이터베이스의 주문 테이블에 존재하는 모든 튜플을 삭제해보자. 그런 다음 주문 테이블의 모든 내용을 검색하여 삭제 여부를 확인해보자.

▶▶ DELETE

FROM 주문;

SELECT \* FROM 주문;

결과 테이블

주문번호	주문고객	주문제품	수량	배송지	주문일자
------	------	------	----	-----	------

## 실습 ⑤ DELETE 삭제

### STEP 1 삽입 후 삭제 테스트

```
INSERT INTO 고객 VALUES ('test', '테스트', 99, 'gold', '기타', 0);  
SELECT * FROM 고객 WHERE 고객아이디 = 'test'; -- 있음
```

### STEP 2 특정 튜플 삭제 – WHERE 필수!

```
DELETE FROM 고객 WHERE 고객아이디 = 'test';  
SELECT * FROM 고객 WHERE 고객아이디 = 'test'; -- 없음!
```

### STEP 3 WHERE 없으면? – 전체 삭제! (위험!)

```
-- 주의: 아래는 모든 고객이 삭제됨!  
-- DELETE FROM 고객;  
-- ★ WHERE 빠뜨리면 전체 튜플 삭제!
```

### X 예상 결과: 에러!

```
WHERE 없는 DELETE = 전체 행 삭제!  
DROP TABLE은 테이블 자체 삭제  
DELETE FROM은 튜플만 삭제 (구조 유지)
```

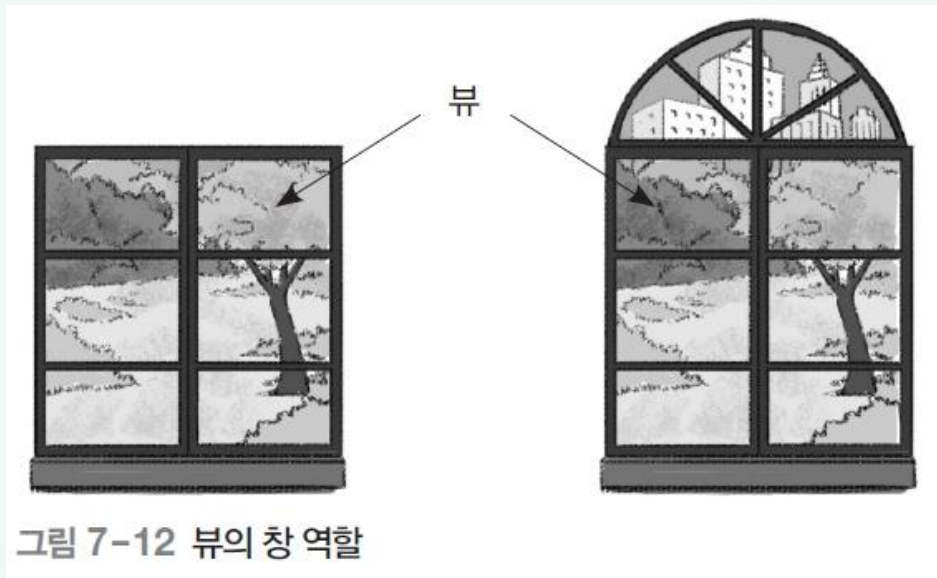
### STEP 4 외래키 참조 중인 튜플 삭제 시도

```
DELETE FROM 고객 WHERE 고객아이디 = 'apple';  
-- apple은 주문 테이블에서 참조 중 → ON DELETE 옵션에 따라!
```

# 뷰(View)의 개념

원본 테이블을 기반으로 만들어진 가상 테이블(virtual table)

뷰의 생성과 삭제도 DDL임



# 뷰의 개념 (계속)

고객 테이블

고객아이디	고객이름	나이	등급	직업	적립금
apple	정소화	20	gold	학생	1000
banana	김선우	25	vip	간호사	2500
carrot	고명석	28	gold	교사	4500
orange	김용욱	22	silver	학생	0
melon	성원웅	35	gold	회사원	5000
peach	오형준	NULL	silver	의사	300
pear	채광주	31	silver	회사원	500

제품 테이블

제품번호	제품명	재고량	단가	제조업체
p01	그냥만두	5000	4500	대한식품
p02	매운졸면	2500	5500	민국푸드
p03	콩떡파이	3600	2600	한빛제과
p04	맛난초콜릿	1250	2500	한빛제과
p05	얼큰라면	2200	1200	대한식품
p06	통통우동	1000	1550	민국푸드
p07	달콤비스킷	1650	1500	한빛제과

주문 테이블

주문번호	주문고객	주문제품	수량	배송지	주문일자
o01	apple	p03	10	서울시 마포구	2022-01-01
o02	melon	p01	5	인천시 계양구	2022-01-10
o03	banana	p06	45	경기도 부천시	2022-01-11
o04	carrot	p02	8	부산시 금정구	2022-02-01
o05	melon	p06	36	경기도 용인시	2022-02-20
o06	banana	p01	19	충청북도 보은군	2022-03-02
o07	apple	p03	22	서울시 영등포구	2022-03-15
o08	pear	p02	50	강원도 춘천시	2022-04-10
o09	banana	p04	15	전라남도 목포시	2022-04-11
o10	carrot	p03	20	경기도 안양시	2022-05-22

그림 7-13 뷰 예제에서 사용하는 판매 데이터베이스 : 고객, 제품, 주문 테이블

# 뷰의 생성 — CREATE VIEW

## SELECT 문과 함께 CREATE VIEW로 생성

WITH CHECK OPTION: 제약조건 | 일반적으로 ORDER BY 사용 불가

```
CREATE VIEW 뷰_이름[(속성_리스트)]  
AS SELECT 문  
[WITH CHECK OPTION];
```

# 뷰의 생성 — 예제

## 예제 7-55

고객 테이블에서 등급이 vip인 고객의 고객아이디, 고객이름, 나이, 등급으로 구성된 뷰를 우수고객이라는 이름으로 생성해보자. 그런 다음 우수고객 뷰의 모든 내용을 검색해보자.

```
▶▶ CREATE VIEW 우수고객(고객아이디, 고객이름, 나이, 등급)
AS SELECT 고객이름, 나이, 등급
FROM 고객
WHERE 등급 = 'vip'
WITH CHECK OPTION;

SELECT * FROM 우수고객;
```

### 결과 테이블

	고객아이디	고객이름	나이	등급
1	banana	김선우	25	vip

# 뷰의 생성 — 속성 리스트 생략

## 우수고객 뷰

속성이 고객 테이블과 같으므로 속성 리스트 생략 가능 | vip가 아닌 등급 삽입/수정 시 거부됨

```
CREATE VIEW    우수고객
AS SELECT     고객아이디, 고객이름, 나이, 등급
FROM         고객
WHERE        등급 = 'vip'
WITH CHECK OPTION;
```

# 뷰의 생성 — 집계 함수

집계함수 사용 시 반드시 속성의 이름(제품수)을 제시해야 함

## 예제 7-56

제품 테이블에서 제조업체별 제품수로 구성된 뷰를 업체별제품수라는 이름으로 생성해보자.  
그런 다음 업체별제품수 뷰의 모든 내용을 검색해보자.

```
▶▶ CREATE VIEW   업체별제품수(제조업체, 제품수)
AS SELECT       제조업체, COUNT(*)
FROM           제품
GROUP BY       제조업체
WITH CHECK OPTION;
```

결과 테이블

	제조업체	제품수
1	대한식품	2
2	민국푸드	2
3	한빛제과	3

```
SELECT * FROM   업체별제품수;
```

# 뷰의 활용

**뷰에서 일반 테이블처럼 쿼리 가능**  
 뷰를 통해 삽입·수정·삭제가 허용될 수 있음

## 예제 7-57

우수고객 뷰에서 나이가 20세 이상인 고객에 대한 모든 내용을 검색해보자.

▶▶ `SELECT * FROM 우수고객 WHERE 나이 >= 20;`

결과 테이블

	고객아이디	고객이름	나이	등급
1	banana	김선우	25	vip

# 뷰의 활용

## 변경이 가능한 뷰와 불가능한 뷰

CREATE VIEW 제품1

AS SELECT 제품번호, 재고량, 제조업체

FROM 제품

WITH CHECK OPTION;

SELECT \* FROM 제품1;

	제품번호	재고량	제조업체
1	p01	5000	대한식품
2	p02	2500	민국푸드
3	p03	3600	한빛제과
4	p04	1250	한빛제과
5	p05	2200	대한식품
6	p06	1000	민국푸드
7	p07	1650	한빛제과

CREATE VIEW 제품2

AS SELECT 제품명, 재고량, 제조업체

FROM 제품

WITH CHECK OPTION;

SELECT \* FROM 제품2;

	제품명	재고량	제조업체
1	그냥만두	5000	대한식품
2	매운짬면	2500	민국푸드
3	콩떡파이	3600	한빛제과
4	맛난초콜릿	1250	한빛제과
5	얼큰라면	2200	대한식품
6	통통우동	1000	민국푸드
7	달콤비스킷	1650	한빛제과

# 뷰 제품1에 신규 데이터 삽입

## 예제 7-58

제품번호가 p08, 재고량이 1,000, 제조업체가 신선식품인 새로운 제품의 정보를 제품1 뷰에 삽입해보자. 그런 다음 제품1 뷰에 있는 모든 내용을 검색해보자.

```
▶▶ INSERT INTO 제품1 VALUES ('p08', 1000, '신선식품');
```

```
SELECT * FROM 제품1;
```

결과 테이블

	제품번호	재고량	제조업체
1	p01	5000	대한식품
2	p02	2500	민국푸드
3	p03	3600	한빛제과
4	p04	1250	한빛제과
5	p05	2200	대한식품
6	p06	1000	민국푸드
7	p07	1650	한빛제과
8	p08	1000	신선식품

# 원본 테이블에도 동일하게 삽입됨

원본 테이블 제품에도 동일하게 신규 데이터 삽입 (제품명은 Null)

SELECT \* FROM 제품;

	제품번호	제품명	재고량	단가	제조업체
1	p01	그냥만두	5000	4500	대한식품
2	p02	매운짬면	2500	5500	민국푸드
3	p03	쿵떡파이	3600	2600	한빛제과
4	p04	맛난초콜릿	1250	2500	한빛제과
5	p05	얼큰라면	2200	1200	대한식품
6	p06	통통우동	1000	1550	민국푸드
7	p07	달콤비스킷	1650	1500	한빛제과
8	p08	(null)	1000	(null)	신선식품

## 뷰 제품2에 삽입 시 에러?

뷰 제품2에 삽입 시 에러가 날까? (O)

```
INSERT INTO 제품2 VALUES ('시원냉면', 1000, '신선식품');
```

## 뷰 제품2 삽입 에러 — 이유

### 왜 에러가 나는가?

뷰 제품2는 기본키(제품번호)가 없기 때문에 기본키가 Null로서 제품 데이터 삽입이 안 되기 때문임

```
INSERT INTO 제품2 VALUES ('시원냉면', 1000, '신선식품');
```

## 변경 불가능한 뷰 — 추가 설명

### 뷰 업체별제품수에도 삽입 시 동일한 에러 발생

기본키(제품번호)가 없기 때문에 삽입 불가

뷰 테이블로의 데이터 조작은 기본 테이블의 내용을 자동으로 바꾸기 때문에 주의해야 함

뷰 테이블이 기본키를 품고 있으면 변경 가능

## 변경이 불가능한 뷰의 특징

### 1. 기본키를 구성하지 않는 뷰

원본 테이블을 변경할 수 없다

### 2. 집계함수로 계산된 내용을 포함하는 뷰

원본 테이블을 변경할 수 없다

### 3. DISTINCT/GROUP BY 키워드를 포함하여 정의한 뷰

변경할 수 없다

### 4. 여러 개의 테이블을 조인하여 정의한 뷰

대부분 변경이 불가능하다

## 뷰 테이블의 대표적인 장점

### 1. 질의문을 좀 더 쉽게 작성할 수 있다

예: GROUP BY 등 복잡한 쿼리를 뷰로 미리 정의

### 2. 데이터 보안 유지에 도움이 된다

뷰를 통해 데이터 접근이 가능하도록 권한 설정

### 3. 데이터를 좀 더 편리하게 관리할 수 있다

기본키를 갖고 있지 않는 뷰는 원본 테이블에 영향이 없음

# 뷰의 삭제 — DROP VIEW

## DROP VIEW 뷰\_테이블

CASCADE CONSTRAINTS 옵션을 쓰면 제약조건을 함께 삭제 가능

DROP VIEW 뷰\_이름;

### 예제 7-59

우수고객 뷰를 삭제해보자.

▶▶ DROP VIEW 우수고객;

# 삽입 SQL의 개념과 특징

Embedded SQL (ESQL) — 응용 프로그래밍 안에 삽입하여 사용하는 SQL 문

일반 명령문이 위치할 수 있는 곳이면 어디든지 삽입 가능

삽입 SQL문 앞에 EXEC SQL을 붙임

프로그램에 선언된 일반 변수 (앞에 :을 붙임)를 삽입 SQL문에서 사용 가능

여러 개의 행을 반환하려면 커서(cursor)가 필요함

# 커서가 필요 없는 삽입 SQL

## CREATE TABLE, INSERT/DELETE 및 UPDATE문

결과로 하나의 행을 반환하기 때문에 커서가 필요 없음

```
int main() {  
  
    EXEC SQL BEGIN DECLARE SECTION;  
        char p_no[4], p_name[21];  
    ❶      int price;  
    EXEC SQL END DECLARE SECTION;  
  
    printf("제품번호를 입력하세요 : ");  
    ❷      scanf("%s", p_no);  
  
    EXEC SQL SELECT 제품명, 단가 INTO :p_name, :price  
    ❸      FROM   제품  
            WHERE 제품번호 = :p_no;  
  
    ❹      printf("\n 제품명 = %s", p_name);  
            printf("\n 단가 = %d", price);  
  
    return 0;  
}
```

그림 7-14 입력된 제품번호에 해당되는 제품명과 단가를 검색하는 프로그램

# 커서가 필요 없는 삽입 SQL (왜 길이가 4?)

## CREATE TABLE, INSERT/DELETE 및 UPDATE문

결과로 하나의 행을 반환하기 때문에 커서가 필요 없음

```
int main() {  
  
    EXEC SQL BEGIN DECLARE SECTION;  
        char p_no[4], p_name[21];  
    ❶ int price;  
    EXEC SQL END DECLARE SECTION;  
  
    printf("제품번호를 입력하세요 : ");  
    ❷ scanf("%s", p_no);  
  
    EXEC SQL SELECT 제품명, 단가 INTO :p_name, :price  
    ❸ FROM 제품  
        WHERE 제품번호 = :p_no;  
  
    ❹ printf("\n 제품명 = %s", p_name);  
    printf("\n 단가 = %d", price);  
  
    return 0;  
}
```

그림 7-14 입력된 제품번호에 해당되는 제품명과 단가를 검색하는 프로그램

# 커서가 필요 없는 삽입 SQL (설명)

## CREATE TABLE, INSERT/DELETE 및 UPDATE문

결과로 하나의 행을 반환하기 때문에 커서가 필요 없음

```
int main() {  
  
    EXEC SQL BEGIN DECLARE SECTION;  
        char p_no[4], p_name[21];  
    ❶ int price;  
    EXEC SQL END DECLARE SECTION;  
  
    printf("제품번호를 입력하세요 : ");  
    ❷ scanf("%s", p_no);  
  
    EXEC SQL SELECT 제품명, 단가 INTO :p_name, :price  
    ❸ FROM 제품  
        WHERE 제품번호 = :p_no;  
  
    ❹ printf("\n 제품명 = %s", p_name);  
    printf("\n 단가 = %d", price);  
  
    return 0;  
}
```

그림 7-14 입력된 제품번호에 해당되는 제품명과 단가를 검색하는 프로그램

# 커서가 필요한 삽입 SQL

## 커서 이름의 선언 및 오픈

결과로 여러 행을 반환하기 때문에 커서가 필요함 | SELECT 문을 실행하기 위한 별도 명령어

```
EXEC SQL DECLARE 커서_이름 CURSOR FOR SELECT 문;
```

예 EXEC SQL OPEN product\_cursor; product\_cursor라는 이름의 커서에 연결된 SELECT 문을 실행함

```
EXEC SQL OPEN 커서_이름;
```

예 EXEC SQL DECLARE product\_cursor CURSOR FOR  
SELECT 제품명, 단가 FROM 제품;      제품 테이블에서 제품명과 단가를 모두 검색하는 SELECT 문을  
위한 커서를 product\_cursor라는 이름으로 선언함

# 커서가 필요한 삽입 SQL

## 커서 이동 및 종료

커서가 가리키는 행으로부터 속성값을 변수에 저장 | 반복문과 함께 사용 | 커서 미사용 시 종료

```
EXEC SQL FETCH 커서_이름 INTO 변수_리스트;
```

예 EXEC SQL CLOSE product\_cursor; *product\_cursor* 커서를 더는 사용하지 않음

```
EXEC SQL CLOSE 커서_이름;
```

예 EXEC SQL FETCH product\_cursor INTO :p\_name, :price; *product\_cursor* 커서를 이동해 결과 테이블의 다음 행에 접근하여 제품명 속성의 값을 *p\_name* 변수에 저장하고 단가 속성의 값을 *price* 변수에 저장함



</SQL>

감사합니다.

대전대학교 컴퓨터공학과