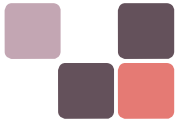


7장. 데이터베이스 언어 SQL

- SQL의 소개
- SQL을 이용한 데이터 정의
- SQL을 이용한 데이터 조작
- 뷰
- 삽입 SQL

04 뷰

- 뷰는 기본 테이블을 들여다 볼 수 있는 창의 역할을 담당

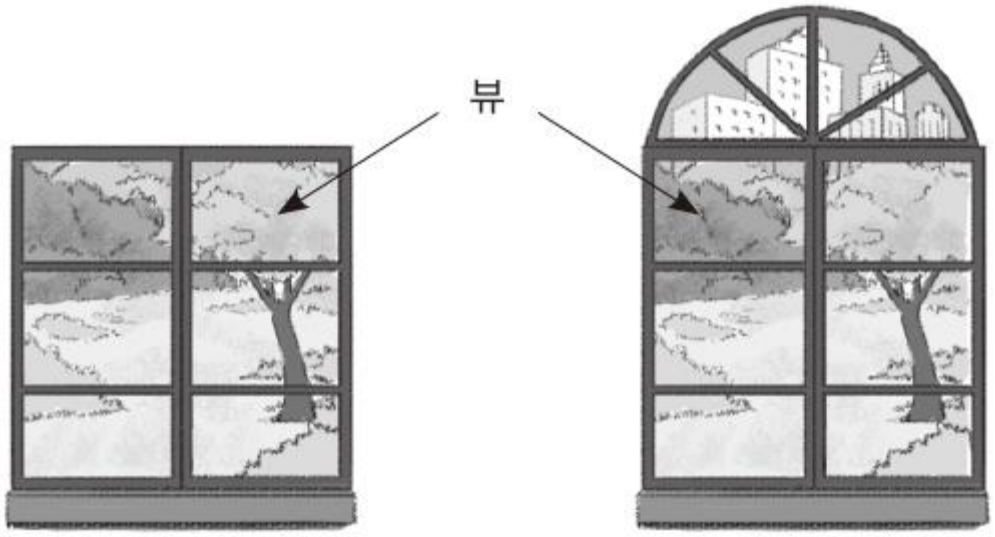
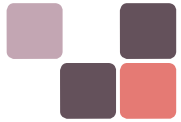


그림 7-12 뷰의 창 역할

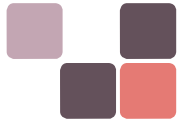
04 뷰



❖ 뷰(View)

- 다른 테이블을 기반으로 만들어진 가상 테이블
- 데이터를 실제로 저장하지 않고 논리적으로만 존재하는 테이블이지만, 일반 테이블과 동일한 방법으로 사용함
- 다른 뷰를 기반으로 새로운 뷰를 만드는 것도 가능함
- 뷰를 통해 기본 테이블의 내용을 쉽게 검색할 수는 있지만, 기본 테이블의 내용을 변화시키는 작업은 제한적으로 이루어짐
  - 기본 테이블: 뷰를 만드는데 기반이 되는 물리적인 테이블

04 뷰



뷰 생성 : CREATE VIEW 문

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

- CREATE VIEW 키워드와 함께 생성할 뷰의 이름과 뷰를 구성하는 속성의 이름을 나열
  - 속성 리스트를 생략하면 SELECT 절에 나열된 속성의 이름을 그대로 사용
- AS 키워드와 함께 기본 테이블에 대한 SELECT 문 작성
  - SELECT 문은 생성하려는 뷰의 정의를 표현하며 ORDER BY는 사용 불가
- WITH CHECK OPTION
  - 뷰에 삽입이나 수정 연산을 할 때 SELECT 문에서 제시한 뷰의 정의 조건을 위반하면 수행되지 않도록 하는 제약조건을 지정

04 뷰

❖ 뷰 생성 : CREATE VIEW 문

예제 7-55

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

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

SELECT  *  FROM  우수고객;
```

결과 테이블

	고객아이디	고객이름	나이
1	banana	김선우	25

뷰가 생성된 후에 우수고객 뷰에 vip 등급이 아닌 고객 데이터를 삽입하거나 뷰의 정의 조건을 위반하는 수정 및 삭제 연산을 시도하면 실행을 거부함 (WITH CHECK OPTION 때문)

04 뷰

❖ 뷰 생성 : CREATE VIEW 문

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

=

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

04 뷰

❖ 뷰 생성 : CREATE VIEW 문

예제 7-56

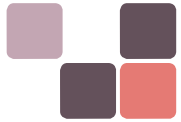
제품 테이블에서 제조업체별 제품수로 구성된 뷰를 업체별제품수라는 이름으로 생성해보자.
그런 다음 업체별제품수 뷰의 모든 내용을 검색해보자.

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

   SELECT  *  FROM  업체별제품수;
```

결과 테이블

	제조업체	제품수
1	대한식품	2
2	민국푸드	2
3	한빛제과	3

제품수 속성은 기본 테이블인
제품 테이블에 원래 있던 속성이 아니라
집계 함수를 통해 새로 계산된 것이므로
속성의 이름을 명확히 제시해야 함



❖ 뷰 활용 : SELECT 문

- 뷰는 일반 테이블과 같은 방법으로 원하는 데이터를 검색할 수 있음
  - 뷰에 대한 SELECT 문이 내부적으로는 기본 테이블에 대한 SELECT 문으로 변환되어 수행
- 검색 연산은 모든 뷰에 수행 가능

예제 7-57

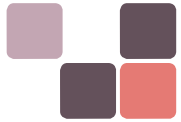
우수고객 뷰에서 나이가 25세 이상인 고객에 대한 모든 내용을 검색해보자.

```
▶▶ SELECT * FROM 우수고객 WHERE 나이 >= 25;
```

결과 테이블

	고객아이디	고객이름	나이
1	banana	김선우	25

04 뷰



❖ 뷰 활용 : INSERT, UPDATE, DELETE 문

- 뷰에 대한 삽입·수정·삭제 연산은 실제로 기본 테이블에 수행되므로 결과적으로는 기본 테이블이 변경됨
- 뷰에 대한 삽입·수정·삭제 연산은 제한적으로 수행됨
  - 변경 가능한 뷰 vs. 변경 불가능한 뷰
- 변경 불가능한 뷰의 특징
  - 기본 테이블의 기본키를 구성하는 속성이 포함되어 있지 않은 뷰
  - 기본 테이블에 있던 내용이 아닌 집계 함수로 새로 계산된 내용을 포함하는 뷰
  - DISTICNT 키워드를 포함하여 정의한 뷰
  - GROUP BY 절을 포함하여 정의한 뷰
  - 여러 개의 테이블을 조인하여 정의한 뷰는 변경이 불가능한 경우가 많음

04 뷰

❖ 뷰 활용 : INSERT, UPDATE, DELETE 문

```
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

제품1 뷰는 변경 가능한 뷰인가?

04 뷰

❖ 뷰 활용 : INSERT, UPDATE, DELETE 문

```
CREATE  VIEW   제품2
AS  SELECT     제품명, 재고량, 제조업체
    FROM       제품
WITH CHECK OPTION;

SELECT  *  FROM  제품2;
```

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

제품2 뷰는 변경 가능한 뷰인가?

❖ 뷰 활용 : INSERT, UPDATE, DELETE 문

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

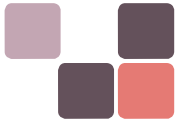
04 뷰

❖ 뷰 활용 : INSERT, UPDATE, DELETE 문

```
SELECT * FROM 제품;
```

	제품번호	제품명	재고량	단가	제조업체
1	p01	그냥만두	5000	4500	대한식품
2	p02	매운쫄면	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

제품1 뷰에 대한 삽입 연산은 실제로 기본 테이블인 제품 테이블에 수행된다.
즉, 새로운 제품의 데이터는 제품 테이블에 삽입된다.



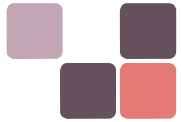
❖ 뷰 활용 : INSERT, UPDATE, DELETE 문

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

제품2 뷰에 대한 삽입 연산은 실패함(오류 발생)

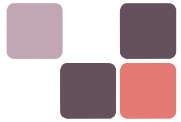
➔ 제품2 뷰는 제품 테이블의 기본키인 제품번호 속성을 포함하고 있지 않기 때문에
제품2 뷰를 통해 새로운 투플을 삽입하려고 하면
제품번호 속성이 널 값이 되어 삽입 연산에 실패하게 됨

04 뷰



❖ 뷰의 장점

- 질의문을 좀 더 쉽게 작성할 수 있다.
  - GROUP BY, 집계 함수, 조인 등을 이용해 뷰를 미리 만들어 놓으면, 복잡한 SQL 문을 작성하지 않아도 SELECT 절과 FROM 절만으로도 원하는 데이터의 검색이 가능
- 데이터의 보안 유지에 도움이 된다.
  - 자신에게 제공된 뷰를 통해서만 데이터에 접근하도록 권한 설정이 가능
- 데이터를 좀 더 편리하게 관리할 수 있다.
  - 제공된 뷰와 관련이 없는 다른 내용에 대해 사용자가 신경 쓸 필요가 없음

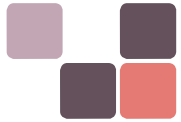


❖ 뷰 삭제 : DROP VIEW 문

- 뷰를 삭제해도 기본 테이블은 영향을 받지 않음

```
DROP  VIEW  뷰_이름;
```

- 만약, 삭제할 뷰를 참조하는 제약조건이 존재한다면?
  - 예) 삭제할 뷰를 이용해 만들어진 다른 뷰가 존재하는 경우
  - 뷰 삭제가 수행되지 않음
  - 관련된 제약조건을 먼저 삭제해야 함



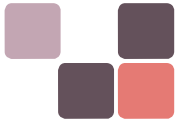
❖ 뷰 삭제 : DROP VIEW 문

예제 7-59

우수고객 뷰를 삭제해보자.

```
▶▶ DROP  VIEW  우수고객;
```

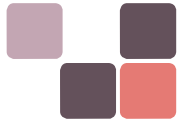
05 삽입 SQL



❖ 삽입 SQL의 개념과 특징

- 삽입 SQL(ESQL; Embedded SQL)
  - 프로그래밍 언어로 작성된 응용 프로그램 안에 삽입하여 사용하는 SQL 문
- 주요 특징
  - 프로그램 안에서 일반 명령문이 위치할 수 있는 곳이면 어디든 삽입 가능
  - 일반 명령문과 구별하기 위해 삽입 SQL 문 앞에 EXEC SQL을 붙임
  - 프로그램에 선언된 일반 변수를 삽입 SQL 문에서 사용할 때는 이름 앞에 콜론(:)을 붙여서 구분함
- 커서(cursor)
  - 수행 결과로 반환된 여러 행을 한 번에 하나씩 가리키는 포인터
  - 여러 개의 행을 결과로 반환하는 SELECT 문을 프로그램에서 사용할 때 필요

05 삽입 SQL



❖ 삽입 SQL 문에서 사용할 변수 선언 방법

- BEGIN DECLARE SECTION과 END DECLARE SECTION 사이에 선언

❖ 커서가 필요 없는 삽입 SQL

- CREATE TABLE 문, INSERT 문, DELETE 문, UPDATE 문
- 결과로 행 하나만 반환하는 SELECT 문

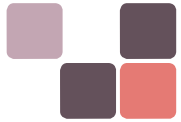
05 삽입 SQL

삽입 SQL 문은 "EXEC SQL"을 붙여서 일반 명령문과 구분함

```
int main() {

    EXEC  SQL  BEGIN  DECLARE  SECTION;
❶         char   p_no[4],  p_name[21];
          int    price;
    EXEC  SQL  END  DECLARE  SECTION;

❷   printf("제품번호를 입력하세요 : ");
    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

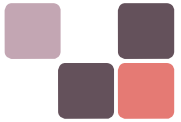
05 삽입 SQL



❖ [그림 7-14]의 프로그램

- 입력된 제품번호에 해당하는 제품명과 단가를 검색하는 프로그램
- ① : 삽입 SQL 문에서 사용할 변수 선언
  - 테이블 내에 대응되는 속성과 같은 타입으로 변수의 데이터 타입을 선언
  - C 프로그램에서는 문자열의 끝을 표시하는 널 문자('\0')을 포함할 수 있도록 변수 선언 시 대응되는 속성의 문자열 길이 보다 한 개 더 길게 선언
- ② : 검색하고자 하는 제품의 제품번호를 사용자로부터 입력받는 부분
- ③ : 제품 테이블에서 사용자가 입력한 제품번호에 해당하는 제품명과 단가를 검색하여 대응되는 각각의 변수에 저장하는 삽입 SQL 문
  - 변수는 INTO 키워드 다음에 차례대로 나열
- ④ : 검색된 제품명과 단가를 화면에 출력

05 삽입 SQL



❖ 커서가 필요한 삽입 SQL

- 커서를 선언하는 삽입 SQL 문
  - 커서를 사용하기 전에 커서의 이름과 커서가 필요한 SELECT 문을 선언

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

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 선언

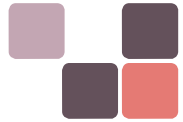
- 커서에 연결된 SELECT 문을 실행하는 삽입 SQL 문

```
EXEC SQL OPEN 커서_이름;
```

예 EXEC SQL OPEN product_cursor;

product_cursor라는 이름의 커서에 연결된 SELECT 문 실행

05 삽입 SQL



❖ 커서가 필요한 삽입 SQL

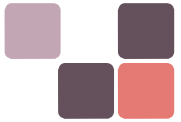
- 커서를 이동시키는 삽입 SQL 문
  - 커서를 이동하여 처리할 다음 행을 가리키도록 하고, 커서가 가리키는 행으로부터 속성 값을 가져와 변수에 저장시킴
  - 결과 테이블에는 여러 행이 존재하므로 FETCH 문은 여러 번 수행해야 함
    - for, while 문과 같은 반복문과 함께 사용

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

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 저장

05 삽입 SQL

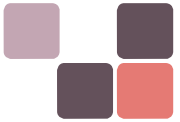


❖ 커서가 필요한 삽입 SQL

- 커서의 사용을 종료하는 삽입 SQL 문

```
EXEC  SQL  CLOSE  커서_이름;
```

예 EXEC SQL CLOSE procuct_cursor; product_cursor 커서를 더는 사용하지 않음



Thank You