## SQL 문의 개요

- **SQL(Structured Query Language, 구조화된 질의 언어)**
  - 데이터베이스에서 사용되는 일종의 공통 언어
  - NCITS(국제표준화위원회)에서 ANSI/ISO SQL이라는 명칭의 SQL 표준을 관리하고 있음
  - 1992년에 제정된 ANSI-92 SQL과 1999년에 제정된 ANSI-99 SQL을 대부분의 DBMS 회사에서 SQL 표준으로 사용하고 있음
  - 각 회사는 ANSI-92/99 SQL의 표준을 준수하면서도 자신의 제품 특성을 반영한 SQL에 별도의 이름을 붙임
  - MySQL에서는 그냥 SQL, 오라클에서는 PL/SQL, SQL Server에서는 Transact SQL(T-SQL) 사용



## 고급 SELECT 문

### – 부속 질의문을 이용한 검색

- SELECT 문 안에 또 다른 SELECT 문을 포함하는 질의
  - 상위 질의문(주 질의문): 다른 SELECT 문을 포함하는 SELECT 문
  - 부속 질의문(서브 질의문): 다른 SELECT 문 안에 들어 있는 SELECT 문
    - » 괄호로 묶어서 작성, ORDER BY 절을 사용할 수 없음
    - » 단일 행 부속 질의문: 하나의 행을 결과로 반환
    - » 다중 행 부속 질의문: 하나 이상의 행을 결과로 반환



## 고급 SELECT 문

### – 부속 질의문을 이용한 검색

- 부속 질의문을 먼저 수행하고, 그 결과를 이용해 상위 질의문을 수행
- 부속 질의문과 상위 질의문을 연결하는 연산자가 필요
  - 단일 행 부속 질의문은 비교 연산자(=, <>, >, >=, <, <=) 사용 가능
  - 다중 행 부속 질의문은 비교 연산자 사용 불가



## 고급 SELECT 문

판매 데이터베이스에서 달콤비스킷을 생산한 제조업체가 만든 제품들의 제품명과 단가를 검색해보자.

결과 테이블

| | 제품명 | 단가 |
|---|---|---|
| 1 | 쿵떡파이 | 2600 |
| 2 | 맛난초콜릿 | 2500 |
| 3 | 달콤비스킷 | 1500 |

'달콤비스켓'의 제조업체는 '한빛제과'밖에 없기 때문에 단일 행 부속 질의문
(비교 연산자 =를 이용)

## 고급 SELECT 문

판매 데이터베이스에서 적립금이 가장 많은 고객의 고객이름과 적립금을 검색해보자.

최대 적립금은 단일 값이므로 단일 행 부속 질의문
(비교 연산자 =를 이용)

5/29

## 고급 SELECT 문

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) |

6/29

## 고급 SELECT 문

판매 데이터베이스에서 banana 고객이 주문한 제품의 제품명과 제조업체를 검색해보자.

결과 테이블

| | 제품명 | 제조업체 |
|---|---|---|
| 1 | 그냥만두 | 대한식품 |
| 2 | 맛난초콜릿 | 한빛제과 |
| 3 | 통통우동 | 민국푸드 |

'banana' 고객이 주문한 제품은 여러 개이므로
다중 행 부속 질의문
(IN 연산자를 이용)

7/29

## 고급 SELECT 문

판매 데이터베이스에서 banana 고객이 주문하지 않은 제품의 제품명과 제조업체를 검색해보자.

결과 테이블

| | 제품명 | 제조업체 |
|---|---|---|
| 1 | 쿵떡파이 | 한빛제과 |
| 2 | 매운쫄면 | 민국푸드 |
| 3 | 얼큰라면 | 대한식품 |
| 4 | 달콤비스킷 | 한빛제과 |

8/29

## 고급 SELECT 문

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

결과 테이블

| | 제품명 | 단가 | 제조업체 |
|---|---|---|---|
| 1 | 매운쫄면 | 5500 | 민국푸드 |

9/29

## 고급 SELECT 문

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

결과 테이블

| | 제품명 | 단가 | 제조업체 |
|---|---|---|---|
| 1 | 매운쫄면 | 5500 | 민국푸드 |

10/29

## 고급 SELECT 문

판매 데이터베이스에서 2019년 3월 15일에 제품을 주문한 고객의 고객이름을 검색해보자.

결과 테이블

| | 고객이름 |
|---|---|
| 1 | 정소화 |

11/29

## 고급 SELECT 문

판매 데이터베이스에서 2019년 3월 15일에 제품을 주문하지 않은 고객의 고객이름을 검색해보자.

결과 테이블

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

12/29

## 고급 SELECT 문

❶ 조인 질의를 이용한 SELECT 문

❷ EXISTS 연산자를 이용한 SELECT 문

13/29

## 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

14/29

## 뷰 view

❖ **뷰는 기본 테이블을 들여다 볼 수 있는 창의 역할을 담당**

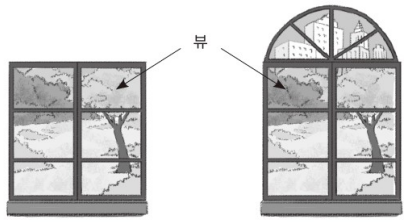


그림 7-12 뷰의 창 역할

15/29

## 뷰 view

❖ **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

16/29

## 뷰 view

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

```
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)



## 뷰 view

```
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;
```



## 뷰 view

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

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 view

- 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

## 뷰 view

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

결과 테이블

| | 고객아이디 | 고객이름 | 나이 |
|---|---|---|---|
| 1 | banana | 김선우 | 25 |



## 뷰 view

- **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



## 뷰 view

```
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 view

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

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 |

## 뷰 view

```
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.

25/29

## 뷰 view

```
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

26/29

## 뷰 view

- 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 필요 X

27/29

## 뷰 view

- **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

28/29

## 뷰 view

우수고객 뷰를 삭제해보자.

29/29