

6장

<SQL 첫걸음> 6장 (25~30강):
데이터베이스 객체의 작성과 삭제

목차

1. 데이터베이스 객체
2. 테이블 작성 · 삭제 · 변경
3. 제약과 기본키
4. 인덱스 구조
5. 인덱스 작성과 삭제
6. 뷰 작성과 삭제

1. 데이터베이스 객체

- 데이터베이스 객체
 - 스키마



데이터베이스 객체

- 데이터베이스 객체 (이하 객체)
 - 테이블이나 뷰, 인덱스 등 데이터베이스 내에 정의하는 모든 것
- 객체와 명령의 차이점
 - 객체 = 데이터베이스 내에서 실체를 가진다 ex) 테이블, 뷰, 인덱스
 - 명령 = 실체가 없다 ex) SELECT, INSERT 등 SQL 명령
- 객체는 이름을 가진다
 - 데이터베이스 내에서 객체를 작성할 때는 이름이 겹치지 않도록 함
 - 이름은 객체의 종류와는 관계가 없음

ex) foo라는 이름의 테이블을 만들면, 테이블은 물론이고 뷰와 같은 다른 종류의 객체 역시 똑같은 이름으로 작성할 수 없음

데이터베이스 객체

cf. 데이터베이스 객체의 명명규칙 (제약 사항)

- 기존 이름이나 예약어와 중복하지 않는다.
- 숫자로 시작할 수 없다.
- 언더스코어(_) 이외의 기호는 사용할 수 없다.
- 한글을 사용할 때는 더블쿼트(MySQL에서는 백쿼트)로 둘러싼다.
- 시스템이 허용하는 길이를 초과하지 않는다.



스키마

- 데이터베이스 객체는 **스키마**라는 그릇 안에 만들어짐
 - 이 같은 특징 때문에 데이터베이스 객체는 스키마 객체라 불리기도 함
 - 데이터베이스에 테이블을 작성해서 구축하는 작업을 스키마 설계라고 부름
- 어떤 것이 스키마가 되는지는 데이터베이스 제품에 따라 다름
 - MySQL에서 스키마는 CREATE DATABASE 명령으로 작성한 **“데이터베이스”**
 - 한편 Oracle 등에서는 데이터베이스와 데이터베이스 사용자가 계층적 스키마

스키마

- 네임스페이스 (namespace)
 - 스키마(테이블)와 테이블(열)은 무언가를 담는 그릇 역할을 한다는 점에서 비슷함
 - 각각의 그릇 안에서는 중복하지 않도록 이름을 지정하는 것
=> 즉 객체의 이름이 같아도 스키마가 서로 다르다면 상관없음!
 - 이처럼 이름이 충돌하지 않도록 기능하는 그릇을 네임스페이스라고 부르기도 함



2. 테이블 작성 · 삭제 · 변경

- 테이블 작성
- 테이블 삭제
- 테이블 변경
- ALTER TABLE로 테이블 관리

테이블 작성 · 삭제 · 변경

- DDL과 DML

- DML (Data Manipulation Language) = 데이터를 조작하는 명령
ex) SELECT, INSERT, DELETE, UPDATE
- DDL (Data Definition Language) = 데이터를 정의하는 명령
=> 스키마 내의 객체를 관리할 때 사용

- DDL은 데이터베이스 객체 모두 같은 문법을 사용

- CREATE로 작성, DROP으로 삭제, ALTER로 변경
- 뒤이어 어떤 종류의 객체를 작성, 삭제, 변경할지를 지정
ex) 테이블 작성은 CREATE TABLE, 뷰 작성은 CREATE VIEW

테이블 작성

- `CREATE TABLE` 테이블명 열 정의
 - 테이블명 뒤에는 괄호로 묶어 열 정의
 - 열을 정의할 때는 테이블에 필요한 열을 콤마로 구분하여 연속해 지정
- 열명 자료형 [DEFAULT 기본값] [NULL | NOT NULL]
 - 열명 : 명명규칙에 맞게 열에 붙일 이름 지정
 - 자료형 : 열의 자료형으로, CHAR나 VARCHAR와 같은 문자열형으로 지정할 때는 최대길이를 괄호로 묶어줘야 함
 - [기본값]: 자료형에 맞는 기본값을 설정(생략 가능)
 - [NULL | NOT NULL]: NULL 또는 생략 시 NULL 허용

테이블 작성

```
1      # CREATE TABLE 테이블 작성하기
2
3      CREATE TABLE sample62 (
4          no INTEGER NOT NULL,
5          a VARCHAR(30),
6          b DATE
7      );
8
9      DESCRIBE sample62;
10
```

<						
Result Grid						
Filter Rows: <input type="text"/>						
Export: <input type="text"/>						
	Field	Type	Null	Key	Default	Extra
▶	no	int(11)	NO		NULL	
	a	varchar(30)	YES		NULL	
	b	date	YES		NULL	

테이블 삭제

- **DROP TABLE** 테이블명
 - 테이블 정의와 함께 테이블에 저장된 데이터도 삭제
 - 데이터만 삭제할 때는 DELETE 명령을 WHERE 조건 없이 사용
 - 하지만 DELETE는 행 단위로 처리하므로 삭제 행이 많으면 속도가 느림
- **TRUNCATE TABLE** 테이블명
 - DDL로 분류되는 명령으로 빠르게 모든 행을 삭제 가능
 - 삭제할 행이나 WHERE 조건을 지정할 수는 없음

테이블 변경

- ALTER TABLE 테이블명 명령
 - 테이블에 저장된 데이터는 그대로 남긴 채 구성만 변경 가능
 - 비교적 새로운 명령에 속하므로 데이터베이스에 따라 다른 방언이 존재함
- ALTER TABLE로 할 수 있는 일
 - 열 추가, 삭제, 변경
 - (1) 열 추가 (2) 열 속성 변경 (3) 열 이름 변경 (4) 열 삭제
 - 제약 추가, 삭제
 - (제약 부분에서 설명)

테이블 변경

(1) 열 추가




- ALTER TABLE 테이블명 **ADD** 열 정의
 - 기존 데이터행이 존재하면 추가한 열의 값이 모두 NULL이 됨
 - 물론 기본값이 지정되어 있으면 기본값으로 데이터가 저장됨
 - NOT NULL제약을 붙인 열을 추가하고 싶다면 먼저 NOT NULL로 제약을 건 뒤, NULL 이외의 값으로 기본값을 지정해야 함

(2) 열 속성 변경

- ALTER TABLE 테이블명 **MODIFY** 열 정의
 - 자료형이나 기본값, NOT NULL 제약 등의 속성 변경 가능 (이름은 불가)
 - 기존의 데이터 행이 존재하는 경우 변경된 속성으로 변환됨
 - 다만 에러가 발생하면 ALTER TABLE 명령은 실행되지 않음

```
11      # ALTER TABLE로 테이블에 열 추가하기
12  ●    ALTER TABLE sample62 ADD newcol INTEGER DEFAULT 0 NOT NULL;
13  ●    DESCRIBE sample62;
```

<

Result Grid |  Filter Rows: | Export:  | Wrap Cell Content: 



	Field	Type	Null	Key	Default	Extra
▶	no	int(11)	NO		NULL	
	a	varchar(30)	YES		NULL	
	b	date	YES		NULL	
	newcol	int(11)	NO		0	

테이블 변경

(1) 열 추가

```
15      # ALTER TABLE로 열 속성 변경하기
16 •    ALTER TABLE sample62 MODIFY newcol VARCHAR(20);
17 •    DESCRIBE sample62;
18
```

<

Result Grid | Filter Rows: | Export:  | Wrap Cell Content: 

	Field	Type	Null	Key	Default	Extra
▶	no	int(11)	NO		NULL	
	a	varchar(30)	YES		NULL	
	b	date	YES		NULL	
	newcol	varchar(20)	YES		NULL	

테이블 변경

(2) 열 속성 변경

테이블 변경

(3) 열 이름 변경




- ALTER TABLE 테이블명 **CHANGE** “기존 열명” “신규 열 정의”
 - CHANGE 명령으로 열 이름과 더불어 열 속성도 변경할 수 있음

(4) 열 삭제

- ALTER TABLE 테이블명 **DROP** 열명
 - 테이블에 존재하지 않는 열이 지정되면 에러 발생

```
19      # ALTER TABLE 열 속성 변경하기
20  ●    ALTER TABLE sample62 CHANGE newcol c VARCHAR(20);
21  ●    DESCRIBE sample62;
```

<

Result Grid |  Filter Rows: | Export:  | Wrap Cell Content: 

	Field	Type	Null	Key	Default	Extra
▶	no	int(11)	NO		NULL	
	a	varchar(30)	YES		NULL	
	b	date	YES		NULL	
	c	varchar(20)	YES		NULL	

테이블 변경

(3) 열 이름 변경



ALTER TABLE로 테이블 관리

- 실무에서 자주 사용하는 ALTER TABLE 사용 예시
- 최대길이 연장
 - 문자열형의 경우 테이블 생성 시(CREATE TABLE) 최대길이를 지정하는데, 이 최대길이를 ALTER TABLE로 늘릴 수 있음
 - 일반적으로 최대길이를 늘리는 경우는 많지만 줄이는 경우는 적음
- 열 추가
 - 테이블 정의를 바꾸는 일인 만큼 시스템에 꽤 영향을 준다
 - 추가한 열에 대해서는 일단 확인하는 편이 낫다

3. 제약과 기본키

- 테이블 작성시 제약 정의
 - 제약 추가
 - 제약 삭제
 - 기본키



테이블 작성시 제약 정의

- CREATE TABLE로 테이블을 작성할 때 제약을 같이 정의
 - 물론 ALTER TABLE로 제약을 지정하거나 변경할 수 있음
 - NOT NULL 등 하나의 열에 대해 설정하는 제약은 열 정의에서 지정
- (a) 열 제약과 (b) 테이블 제약
 - 열 제약 = 열에 대해 정의하는 제약
 - 테이블 제약 = 복수의 열에 대해 적용하는 한 개의 제약
- 제약에는 이름을 붙일 수 있다
 - 나중에 관리하기 쉬워지므로 가능한 한 이름을 붙임
 - 제약 이름은 CONSTRAINT 키워드를 사용해서 지정할 것

제약 추가

(a) 열 제약 추가

- ALTER TABLE의 **MODIFY** 하부명령으로 열 정의를 변경할 수 있음
- 이때 제약을 위반하는 데이터가 있는지 먼저 검사함

(b) 테이블 제약 추가

- ALTER TABLE의 **ADD** 하부명령으로 추가할 수 있음
- 같은 방법으로 기본키 제약도 가능, 단 기본키는 테이블에 하나만 있어야 함

제약 삭제

(a) 열 제약 삭제

- 제약을 추가할 때와 동일하게 열 정의를 변경

(b) 테이블 제약 삭제

- 테이블 제약은 ALTER TABLE의 DROP 하부명령으로 삭제 가능
- 제약명을 지정해야 하나, 기본키 삭제시에는 생략 가능

제약 정의 · 추가 · 삭제

```
# 테이블 열에 제약 정의하기
• CREATE TABLE sample631 (
    a INTEGER NOT NULL,
    b INTEGER NOT NULL UNIQUE,
    c VARCHAR(30)
);

# 열 제약 추가/삭제
• ALTER TABLE sample631 MODIFY c VARCHAR(30) NOT NULL;
• ALTER TABLE sample631 MODIFY c VARCHAR(30);

# 테이블 제약 추가/삭제
• ALTER TABLE sample631 ADD CONSTRAINT pkey PRIMARY KEY(a);
• ALTER TABLE sample631 DROP pkey;
• ALTER TABLE sample631 DROP PRIMARY KEY;
```


기본키

- 기본키

- 테이블의 행 한 개를 특정할 수 있는 검색키
- 기본키로 지정할 열은 NOT NULL 제약이 설정되어 있어야 함

- 기본키 제약

- 기본키로 검색했을 때 복수의 행이 일치하는 데이터를 작성할 수 없음
- 즉 기본키로 설정된 열이 중복하는 데이터를 값을 가지면 제약에 위반됨
- 것처럼 열을 기본키로 지정해 유일한 값을 가지도록 하는 구조를 말함
- 행이 유일성을 필요로 한다는 다른 의미에서 유일성 제약이라 부르기도 함

- 복수의 열로 기본키를 구성해도 됨

- 이 경우 키를 구성하는 모든 열로 중복되는 값이 있는지 없는지를 검사함

```

39      # 기본키 제약에 이름 붙이기
40  ● ○ CREATE TABLE sample634 (
41          p INTEGER NOT NULL,
42          a VARCHAR(30),
43          CONSTRAINT pkey PRIMARY KEY (p)
44  );
45  ● DESCRIBE sample634;

```




<

Result Grid | Filter Rows: | Export: | Wrap Cell Content: [A](#)

	Field	Type	Null	Key	Default	Extra
▶	p	int(11)	NO	PRI	NULL	
	a	varchar(30)	YES		NULL	

기본키 제약

```
47      # 행 추가하기
48  ●   INSERT INTO sample634 VALUES (1, '첫째줄');
49  ●   INSERT INTO sample634 VALUES (2, '첫째줄');
50  ●   SELECT * FROM sample634;
51
52      # 기본키 중복으로 인한 오류 발생
53  ●   INSERT INTO sample634 VALUES (1, 'first_row');
54  ●   UPDATE sample634 SET p=1 WHERE p=2;
```

<		
Result Grid		
Filter Rows: <input type="text"/>		
Edit:   		
	p	a
▶	1	첫째줄
	2	첫째줄
★	NULL	NULL

기본키 제약

4. 인덱스 구조

- 인덱스
 - 검색에 사용하는 알고리즘
- : 이진 트리 (풀 데이터 스캔 vs. 이진 탐색)
- 유일성



인덱스 구조

- **인덱스**란 테이블에 붙여진 색인
- 인덱스의 역할은 검색속도의 향상
 - 테이블에 인덱스가 지정되어 있으면 효율적으로 검색할 수 있으므로 WHERE로 조건이 지정된 SELECT 명령의 처리 속도가 향상됨.
 - 책의 특정 부분을 찾을 때 목차나 색인을 이용하는 것과 마찬가지로
 - 검색 시에 쓰이는 키워드와 대응하는 데이터 행의 장소가 저장되어 있음
- 인덱스는 테이블에 의존하는 객체
 - 테이블과 별개의 데이터베이스 객체로 작성되나, 인덱스만으로는 의미가 없음
 - 대부분의 데이터베이스에서는 테이블을 삭제하면 인덱스도 같이 삭제됨

검색에 사용하는 알고리즘: 이진 트리

- 이진 트리 (binary tree)
 - 이진탐색 “방법”에서 검색하기 쉽게 만든 데이터 “구조”
- 1) 풀 테이블 스캔 (full table scan)
 - 인덱스가 지정되지 않은 테이블의 경우로, 모든 값을 처음부터 조사
 - 즉 행이 1000건 있다면 최대 1000번 값을 비교
- 2) 이진 탐색 (binary search)
 - 집합을 반으로 나누어 조사하는 검색방법으로, 차례로 나열된 경우 유효
 - 데이터 수가 배가 되어도 비교 횟수는 1회만 늘어남
 - => 대량의 데이터를 검색할 때는 이진 탐색이 빠르다

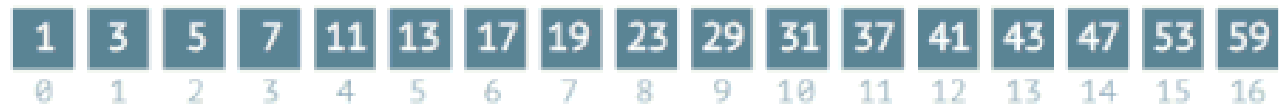
Binary search

steps: 0



Sequential search

steps: 0



www.penjee.com

이진 탐색

검색에 사용하는 알고리즘: 이진 트리

- 이진 트리가 필요한 이유

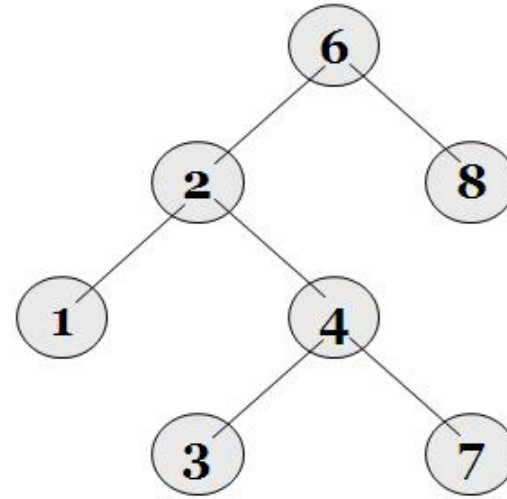
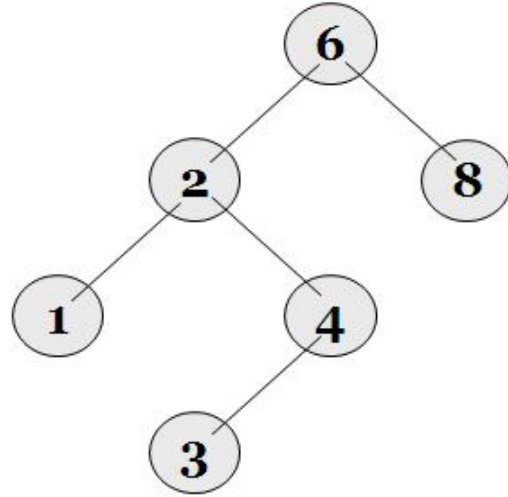
- 이진 탐색은 고속으로 검색할 수 있는 탐색 방법이지만 데이터가 미리 정렬되어 있어야 함
- 하지만 테이블 내의 행을 언제나 정렬된 상태로 두는 것은 힘든 작업
- 일반적으로 테이블에 인덱스를 작성하면 테이블 데이터와 별개인 인덱스용 데이터가 저장
=> 이때 이진 트리라는 데이터 구조로 작성됨

- 이진 트리의 구성

- 한 트리는 노드(node)라는 요소로, 각 노드는 두 개의 가지로 나뉨
- 노드의 왼쪽 가지는 작은 값으로, 오른쪽 가지는 큰 값으로 나뉘어짐
- 이진 탐색의 경우는 오른쪽의 가운데, 왼쪽의 가운데 값을 계산해야 하지만, 이진 트리에서는 구조 자체가 검색하기 쉬우므로 가지를 따라 이동하기만 하면 됨.



A binary search tree



*Not a binary search tree,
but a binary tree*

이진 탐색 트리

유일성

- 이진 트리의 유일성

- 이진 트리에서는 집합 내에 중복하는 값을 가질 수 없음
- 같은 값을 허용하기 위해서는 '같은'이라는 제 3의 가지가 필요
- 즉 이진 트리에서 같은 값을 가지는 노드를 여러 개 만들 수 없다는 특성은 키에 대하여 유일성을 가지게 할 경우에만 유용

=> 기본키 제약은 이진 트리로 인덱스를 작성하는 데이터베이스가 많음

5. 인덱스 작성과 삭제

- 인덱스 작성
- 인덱스 삭제
- EXPLAIN
 - 최적화

인덱스 작성

- 인덱스는 데이터베이스 객체이므로 DDL로 작성하거나 삭제
 - 데이터베이스 제품에 따라 데이터베이스 객체 혹은 테이블의 열처럼 취급
 - SQL Server, MySQL = 테이블 내의 객체
 - Oracle, DB2 = 스키마 객체

=> 인덱스의 네임스페이스가 데이터베이스 제품마다 다름
- **CREATE INDEX** 인덱스명 ON 테이블명 (열명1, 열명2, ...)
 - 해당 인덱스가 어느 테이블의 어느 열(복수 가능)에 관한 것인지 지정해야 함
 - 인덱스 작성 시 저장장치에 색인용 데이터가 만들어지는데, 행이 많을 수록 시간도 많이 걸리고 저장공간도 많이 소비



인덱스 삭제

(1) 테이블 내 객체 => MySQL!

- `DROP INDEX` 인덱스명 `ON` 테이블명

(2) 스키마 객체

- `DROP INDEX` 인덱스명

- 인덱스를 작성하면 검색이 빨라짐

- 인덱스 열을 WHERE 조건식에 지정하여 SELECT 명령으로 검색하면 속도 향상
- 그러나 모든 SELECT 명령에 적용되는 만능 인덱스는 없음
- 한편, INSERT 명령의 경우에는 인덱스를 최신 상태로 갱신하는 처리가 늘어나므로 (SELECT 명령과 달리) 처리속도가 조금 떨어짐

EXPLAIN

- EXPLAIN 명령
 - 실제로 인덱스를 사용해 검색하는지를 확인
 - EXPLAIN 뒤에 확인하고 싶은 SELECT 명령 등의 SQL 명령을 지정
 - 다만 이 SQL 명령은 실제로는 실행되지 않으며, 어떤 상태로 실행되는지를 데이터베이스가 설명해줄 뿐임
 - 표준 SQL에는 존재하지 않는 데이터베이스 제품 의존형 명령이지만, 어떤 제품이 라도 이와 비슷한 명령을 지원함
- EXPLAIN SQL 명령

테이블 작성

(2. 테이블 작성 · 삭제 · 변경)

```
1      # CREATE TABLE로 테이블 작성하기
2
3  ● ○ CREATE TABLE sample62 (
4          no INTEGER NOT NULL,
5          a VARCHAR(30),
6          b DATE
7      );
8
9  ● DESCRIBE sample62;
10
```

<						
Result Grid						
Filter Rows: <input type="text"/>						
Export: <input type="text"/>						
	Field	Type	Null	Key	Default	Extra
▶	no	int(11)	NO		NULL	
	a	varchar(30)	YES		NULL	
	b	date	YES		NULL	

인덱스 사용 (1) a열

```
56      # EXPLAIN으로 인덱스 사용 확인하기
57 •    CREATE INDEX ix ON sample62(a);
58      # DROP INDEX ix ON sample62;
59 •    EXPLAIN SELECT * FROM sample62 WHERE a='a';
```

<div><div><</div><div>Result Grid</div><div><div></div><div>Filter Rows:</div><div></div></div><div>Export: <div></div></div><div>Wrap Cell Content: <div></div></div></div>												
	id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
▶	1	SIMPLE	sample62	NULL	ref	ix	ix	93	const	1	100.00	NULL

(2) no열

```
60 •    EXPLAIN SELECT * FROM sample62 WHERE no>10;
```

<div><div><</div><div>Result Grid</div><div><div></div><div>Filter Rows:</div><div></div></div><div>Export: <div></div></div><div>Wrap Cell Content: <div></div></div></div>												
	id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
▶	1	SIMPLE	sample62	NULL	ALL	NULL	NULL	NULL	NULL	1	100.00	Using where

최적화

- 실행 계획
 - 내부 처리에서는 SELECT 명령을 실행하기 앞서 실행 계획을 세움
 - 인덱스가 지정된 열이 WHERE 조건으로 지정되어 있으니 인덱스를 사용하자
<= EXPLAIN 명령은 이 실행계획을 확인하는 명령
 - 인덱스의 유무뿐만 아니라 인덱스를 사용할 것인지 여부에 대해서도 데이터베이스 내부의 최적화 처리를 통해 판단
- 판단 기준으로 인덱스의 품질도 고려
 - 예를 들어 예 / 아니오 2가지 값만 가지는 열이 있다면, 해당 열에 인덱스를 지정해도 이진 탐색에 의한 효율화를 기대할 수 없음
=> 데이터의 종류가 적으면 적을 수록 인덱스의 효율도 떨어짐.
 - 반대로 여러 값 또는 종류의 데이터가 존재하면 그만큼 효율은 좋아짐

6. 뷰 작성과 삭제

- 뷰
- 뷰 작성과 삭제
- 뷰의 약점

: Materialized View, 함수 테이블



뷰

- 뷰 (View)
 - SELECT 명령을 기록하는 데이터 베이스 객체
 - 본래 데이터베이스 객체로 등록할 수 없는 SELECT 명령을, 객체로서 이름을 붙여 관리할 수 있도록 한 것
 - 뷰를 참조하면 그에 정의된 SELECT 명령의 실행결과를 테이블처럼 사용 가능
- 뷰의 역할
 - 뷰를 정의할 때는 이름과 SELECT 명령을 지정
 - 뷰를 만든 후에는 SELECT 명령에서 뷰의 이름을 지정해서 참조할 수 있음
 - => 복잡한 SELECT 명령을 간략하게 표현 가능
 - 실체가 존재하지 않는다는 의미로 가상 테이블이라 불리기도 함

뷰 작성과 삭제

- 뷰는 데이터베이스 객체이므로 DDL로 작성하거나 삭제
 - 작성할 때는 CREATE VIEW, 삭제할 때는 DROP VIEW를 사용

(1) 뷰의 작성

- **CREATE VIEW** 뷰명 AS SELECT 명령
 - AS로 SELECT 명령을 지정하는데, 별명을 붙일 때와는 달리 생략 불가
 - 뷰명 뒤에 (열명1, 열명2,...)와 같이 지정할 수 있음
 - 열 이외 자료형이나 제약 등은 정의할 수 없음

(2) 뷰 삭제

- **DROP VIEW** 뷰명
 - 일단 뷰를 삭제하면 더 이상 참조 불가

뷰의 약점

1) CPU사용으로 인한 처리속도 저하

- 뷰는 데이터베이스 객체로서 저장장치에 저장되나, 데이터베이스에 저장되는 것은 SELECT명령뿐이기 때문에 테이블과 달리 대량의 저장공간을 필요로 하지는 않음
- 저장공간을 소비하지 않는 대신, 계산능력을 필요로 하기 때문에 CPU 자원을 사용
- 뷰를 참조하면 뷰에 등록된 SELECT 명령이 실행되는데 결과가 일시적으로 보존됨
- 즉 뷰를 참조할 때마다 매번 SELECT 명령이 실행되어야 함
- 데이터 양이 많은 경우, 또는 뷰를 중첩해서 사용하는 경우에도 처리 속도가 느림

뷰의 약점

c.f. Materialized View

- 앞의 상황을 회피하기 위해 사용하는 방법이 Materialized View
- 처음 참조되었을 때 데이터를 저장한 후, 재참조 시 저장해둔 데이터를 불러옴
- 뷰에 지정된 테이블의 데이터가 자주 변경되지않는다면 뷰의 약점을 보완 가능
- 그러나 MySQL에서는 사용 불가 (현재는 Oracle과 DB2에서만 사용 가능)

뷰의 약점

2) 상관 서브쿼리의 경우 사용 불가

- 부모 쿼리와 어떤 식으로든 연관된 (상관) 서브쿼리의 경우 사용 불가
- SELECT 명령은 단독 실행 가능해야 하는데, 위 경우에는 불가하기 때문
- 이 같은 뷰의 약점은 함수 테이블을 사용하여 회피할 수 있음

c.f. 함수 테이블

- 함수 테이블은 테이블을 결과 값으로 반환해주는 사용자 정의 함수
- 함수에는 인수를 지정할 수 있기 때문에 인수의 값에 따라 WHERE 조건을 붙여 결과값을 바꿀 수 있음
- 즉 테이블을 가지고 상관 서브쿼리처럼 조작 가능