데이터 관리기술 Lab

Week 05

2024년 1학기
강의진행: 심철환, 이지연

목차

01

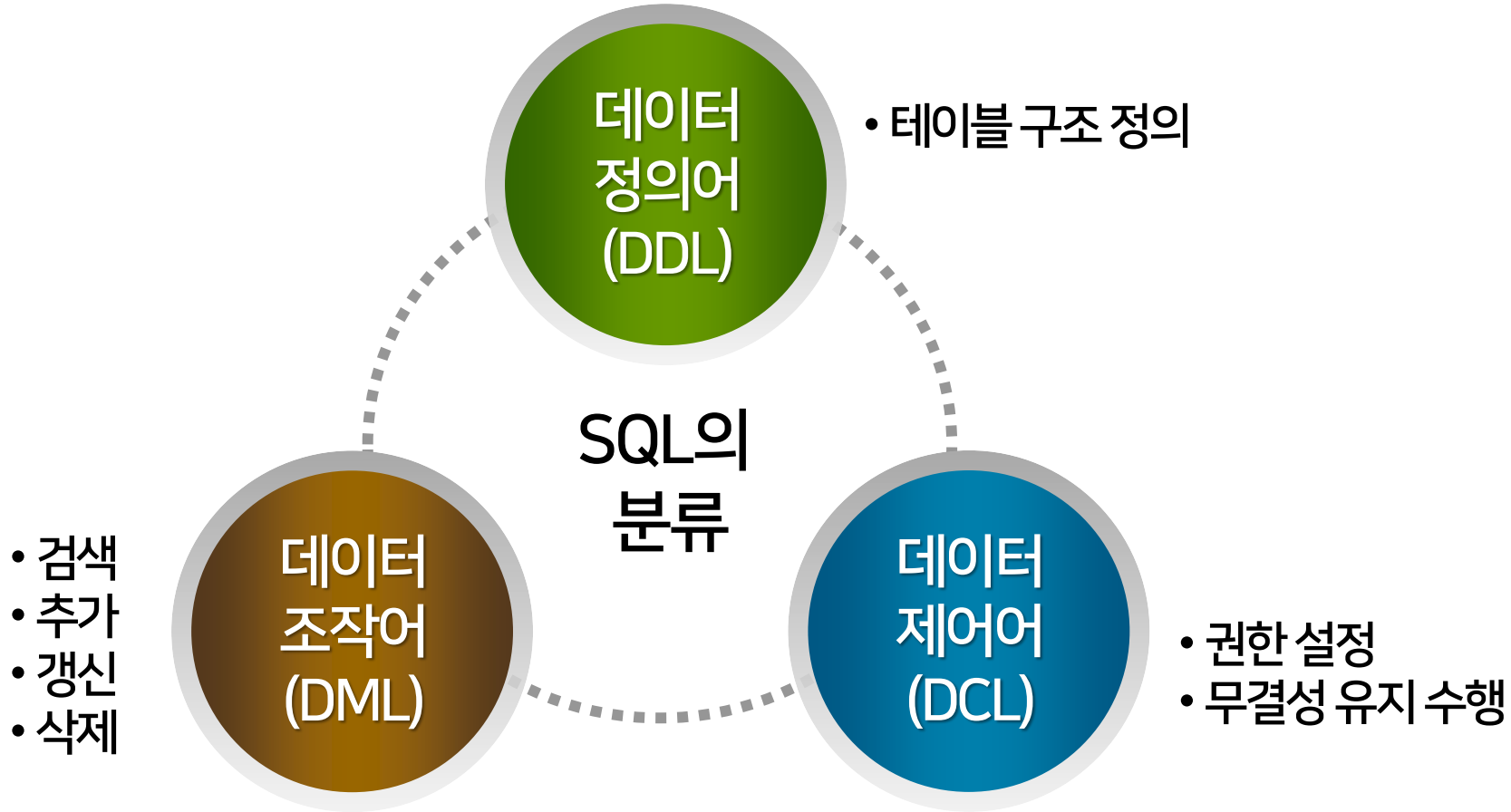
SQL 기본

1. SQL 소개
2. 데이터 정의어(DDL)
3. 데이터 조작어(DML)
4. 조인(Join)
5. 서브 쿼리(Sub query)

1. SQL 소개

SQL 소개

- SQL(Structured Query Language)
  - ✓ 관계형 데이터베이스에서 데이터 정의, 데이터 조작, 데이터 제어를 하기 위해 사용하는 언어



SQL 소개

- 데이터 정의어(DDL, Data Definition Language)
  - ✓ 데이터베이스의 구조를 정의하는 언어

SQL문	내용
CREATE	데이터베이스 개체를 생성 (schema, domain, table, view, index)
DROP	데이터베이스 개체를 삭제
ALTER	기존에 존재하는 데이터베이스 개체의 정의를 변경
RENAME	이름 변경

SQL 소개

- 데이터 조작어(DML : Data Manipulation Language)
  - ✓ 데이터의 검색, 삽입, 삭제, 변경할 때 사용하는 언어

SQL문	내용
INSERT	데이터베이스 개체에 새로운 투플을 삽입
UPDATE	데이터베이스 개체에 입력된 특정 데이터를 찾아서 수정
DELETE	데이터베이스 개체에 입력된 특정 데이터를 찾아서 삭제
SELECT	데이터베이스 개체로부터 데이터 추출 및 검색

SQL 소개

- 데이터 제어어(DCL ,Data Control Language)
  - ✓ 데이터베이스에 접근하고 객체들을 사용하도록 권한을 주고 회수하는 명령어

SQL문	내용
GRANT	데이터베이스 개체에 권한 부여
REVOKE	이미 부여된 데이터베이스 개체의 권한 취소

SQL 소개

- 트랜잭션 제어어(TCL ,Transaction Control Language)

✓ 논리적인 작업 단위를 묶어서 DML에 의해 조작된 결과를 작업단위(트랜잭션)별로 제어하는 명령어

SQL문	내용
COMMIT	한 트랜잭션에서 데이터 조작이 정상적으로 완료되었으면 그 결과를 데이터베이스에 반영하는 명령어
ROLLBACK	트랜잭션 내에서 데이터를 수정하는 도중 이상이 생겼을 때, 변경하기 이전으로 돌아가는 명령어
SAVEPOINT	트랜잭션의 특정 지점에 이름을 지정하고, 그 지점 이전에 수행한 작업에 영향을 주지 않고 그 지점 이후에 수행한 작업을 롤백(ROLLBACK)할 수 있음

Oracle 데이터 타입

데이터 타입

✓ 컬럼이 저장되는 데이터 유형

➢ Oracle에서 기본적으로 제공하는 데이터 타입

- 문자
- 숫자
- 날짜
- LOB(2진 데이터 저장)
- 사용자 정의 타입인 VARRAY
- 중첩 테이블
- OBJECT

Oracle 데이터 타입

문자형 데이터 타입

✓ 고정길이는 입력된 데이터의 길이와 관계없이 저장되는 데이터의 크기가 고정
✓ 가변길이는 실제 입력된 데이터의 길이에 맞춰 데이터의 크기가 결정

데이터 타입	설명
CHAR(n)	고정 길이 데이터(최대 2000byte) 지정된 길이보다 짧은 데이터 입력될 시 나머지 공간이 공백으로 채워짐
VARCHAR2(n)	가변 길이 데이터(최대 4000byte) 지정된 길이보다 짧은 데이터 입력될 시 나머지 공간은 채우지 않음
NCHAR(n)	데이터(최대 2000byte) 고정 길이 유니코드(다국어 지원)
NVARCHAR2(n)	데이터(최대 4000byte) 가변 길이 유니코드(다국어 지원)

Oracle 데이터 타입

- 문자형 데이터 타입

  예) 고정길이인 CHAR(10)과 가변길이인 VARCHAR2(10)에 각각 ABC라는 데이터를 입력하면 다음 그림과 같이 저장됨

데이터타입 \ 길이	1	2	3	4	5	6	7	8	9	10	사용한 길이
CHAR(10)	A	B	C								10
VARCHAR2(10)	A	B	C								3

  - 즉, 고정길이를 사용하게 되면 공간 낭비가 발생하게 됨

Oracle 데이터 타입

숫자형 데이터 타입

- ✓ 숫자타입은 다음과 같이 4가지 데이터 타입이 있지만 NUMBER형이 주로 사용됨
- ✓ NUMBER형의 P는 소수점을 포함한 전체 자릿수를 의미하고, S는 소수점 자릿수를 의미
- ✓ NUMBER는 가변 숫자이므로 P와 S를 입력하지 않으면 저장 데이터의 크기에 맞게 자동 조절

데이터 타입	설명
NUMBER(P, S)	가변길이 숫자 P는 소수점 기준으로 모든 유효숫자 자릿수를 의미, 1~38까지(기본값 38) S는 -84~127까지(기본값: 0) / 최대 22byte 양수는 소수점 이하, 음수는 소수점 이상
FLOAT(P)	NUMBER의 하위타입 P는 1~128까지(기본값: 128) / 이진수 기준 / 최대 22byte
BINARY_FLOAT	32비트 부동소수점 / 최대 4byte
BINARY_DOUBLE	64비트 부동소수점 / 최대 8byte

Oracle 데이터 타입

- 숫자형 데이터 타입

예) 1234.56을

- NUMBER(10,1)에 저장: 소수 첫째 자리가 반올림되어 1234.6이 됨
- NUMBER(10,-1)에 저장: 정수 첫째 자리가 반올림되어 1230이 됨

Oracle 데이터 타입

- 날짜형 데이터 타입
  - ✓ 일반적으로 DATE 타입이 사용

데이터 타입	설명
DATE	BC 4712년 1월 1일 ~ 9999년 12월 31일의 연, 월, 일, 시, 분, 초까지 입력가능 YYYY/MM/DD가 기본값
TIMESTAMP	연도, 월, 일, 시, 분, 초 + 밀리초까지 입력 가능

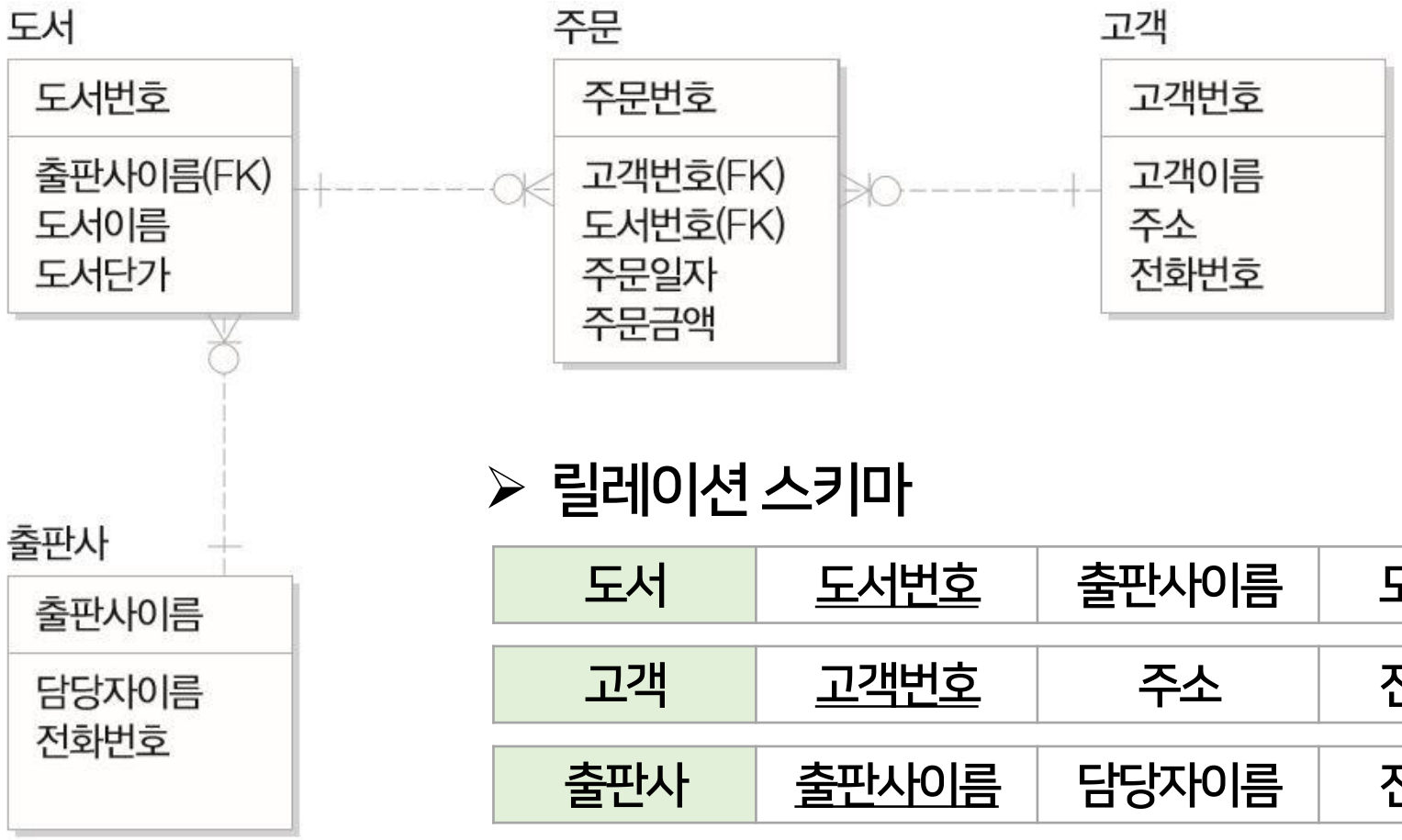
Oracle 데이터 타입

- LOB(Large Object) 데이터 타입
  - ✓ 대용량 데이터를 저장할 수 있는 데이터 타입
  - ✓ 문자형 대용량 데이터는 CLOB나 NCLOB를 사용
  - ✓ 그래픽, 이미지, 동영상 등은 BLOB를 사용

데이터 타입	설명	
CLOB (Character LOB)	문자형 대용량 객체 고정길이와 가변길이 문자 집합 지원	최대 크기는 4GB
NCLOB	유니코드(다국어 지원)를 포함한 문자형 대용량 객체	
BLOB (Binary LOB)	이진형 대용량 객체	
BFILE	대용량 바이너리 데이터를 파일형태로 저장하기 위한 데이터 타입	

실습 예제: 온라인 서점

- ER 다이어그램과 릴레이션 스키마



➢ 릴레이션 스키마

도서	도서번호	출판사이름	도서이름	도서단가
고객	고객번호	주소	전화번호	
출판사	출판사이름	담당자이름	전화번호	
주문	주문번호	고객번호	도서번호	주문일자	주문금액

실습 예제: 온라인 서점

- 테이블
  - Book 테이블

필드명	데이터 타입	속성
bookid	NUMBER(2)	PK
bookname	VARCHAR2(20)	
publisher	VARCHAR2(20)	
price	NUMBER(8)	

bookid	bookname	publisher	price
1	축구의 역사	굿스포츠	7000
2	축구 아는 여자	나무수	13000
3	축구의 이해	대한미디어	22000
4	골프 바이블	대한미디어	35000
5	피겨 교본	굿스포츠	8000
6	역도 단계별기술	굿스포츠	6000
7	야구의 추억	이상미디어	20000
8	야구를 부탁해	이상미디어	13000
9	올림픽 이야기	삼성당	7500
10	Olympic Champions	Pearson	13000

실습 예제: 온라인 서점

- 테이블
  - Customer 테이블

필드명	데이터 타입	속성
custid	NUMBER(2)	PK
name	VARCHAR2(40)	
address	VARCHAR2(50)	
phone	VARCHAR2(20)	

custid	name	address	phone
1	박지성	영국 맨체스터	000-5000-0001
2	김연아	대한민국 서울	000-6000-0001
3	장미란	대한민국 강원도	000-7000-0001
4	추신수	미국 클리블랜드	000-8000-0001
5	박세리	대한민국 대전	NULL

실습 예제: 온라인 서점

- 테이블
  - Orders 테이블

필드명	데이터 타입	속성
orderid	NUMBER(2)	PK
custid	NUMBER(2)	FK
bookid	NUMBER(2)	FK
saleprice	NUMBER(8)	
orderdate	DATE	

orderid	custid	bookid	saleprice	orderdate
1	1	1	6000	2020-07-01
2	1	3	21000	2020-07-03
3	2	5	8000	2020-07-03
4	3	6	6000	2020-07-04
5	4	7	20000	2020-07-05
6	1	2	12000	2020-07-07
7	4	8	13000	2020-07-07
8	3	10	12000	2020-07-08
9	2	10	7000	2020-07-09
10	3	8	13000	2020-07-10

2. 데이터 정의어(DDL)

데이터 정의어(DDL)

- 데이터 정의어의 기능
  - ✓ 테이블 생성, 테이블 구조의 변경, 테이블 삭제

SQL문		
테이블 생성	CREATE TABLE	
테이블 변경	ALTER TEABLE	• ADD COLUMN • DROP COLUMN • MODIFY COLUMN • ADD CONSTRAINT • DROP CONSTRAINT • RENAME COLUMN
테이블 삭제	DROP TABLE	
테이블 이름 변경	RENAME TABLE	
테이블의 행 제거	TRUNCATE TABLE	

CREATE TABLE

테이블 생성시 규칙

- ✓ 테이블명은 객체를 의미할 수 있는 적절한 이름 사용(가능한 단수형을 권고)
- ✓ 테이블 명은 다른 테이블의 이름과 중복되지 않아야 함
- ✓ 한 테이블 내에서는 칼럼명이 중복되게 지정될 수 없음
- ✓ 각 테이블 이름을 지정하고 각 컬럼들은 괄호( )로 묶어서 지정
- ✓ 각 컬럼들은 콤마(,)로 구분하고, 테이블 생성문의 끝은 항상 세미콜론(;)으로 끝남
- ✓ 컬럼은 다른 테이블까지 고려해 데이터베이스 내에서 일관성 있게 사용하는 것이 좋음
- ✓ 컬럼 뒤에 데이터 유형을 꼭 지정해야 함
- ✓ 테이블과 컬럼명은 반드시 문자로 시작해야 하고, 벤더별로 길이에 대한 한계가 있음
- ✓ 벤더에서 사전에 정의한 예약어는 쓸 수 없음
- ✓ A-Z, a-z, 0-9, _, $, # 문자만 허용

예) 테이블명이 잘못된 사례

- 10_player: 반드시 숫자가 아닌 문자로 시작해야 함
- T-player: 특수문자 '-'는 허용되지 않음

CREATE TABLE

- 테이블 생성 구문

```
CREATE TABLE 테이블명(
  ① 컬럼명 데이터타입 [NOT NULL] [기본값]
  ② [PRIMARY KEY(컬럼명)]
  ③ [UNIQUE(컬럼명)]
  ④ [FOREIGN KEY(컬럼명) REFERENCES 테이블명(컬럼명)]
     [ON DELETE 옵션] [ON UPDATE 옵션]
  ⑤ [CONSTRAINT 이름] [CHECK(조건)]);
```

① 테이블을 구성하는 각 컬럼의 이름과 데이터 타입, 기본적인 제약 사항을 정의한다.
② 기본키로 테이블에 하나만 존재한다.
③ 대체키로 테이블에 여러 개 존재할 수 있다.
④ 외래키로 테이블에 여러 개 존재할 수 있다.
⑤ 데이터 무결성을 위한 제약조건으로 테이블에 여러 개 존재할 수 있다.

CREATE TABLE

테이블 생성 구문

```
CREATE TABLE 테이블명(
    컬럼명 데이터타입 [NOT NULL] [기본값]
    [PRIMARY KEY(컬럼명)]
    [UNIQUE(컬럼명)]
    [FOREIGN KEY(컬럼명) REFERENCES 테이블명(컬럼명)]
    [ON DELETE 옵션] [ON UPDATE 옵션]
    [CONSTRAINT 이름] [CHECK(조건)]);
```

- [ ]의 내용은 생략이 가능
- SQL 문은 세미콜론(;)으로 문장의 끝을 표시
- SQL 문은 대소문자를 구분하지 않음

CREATE TABLE

- 테이블 생성
  - ✓ 테이블을 구성하는 각 컬럼의 이름, 데이터 타입, 기본 제약 사항 정의
  - ✓ 기본키 정의
  - ✓ 대체키 정의
  - ✓ 외래키 정의
  - ✓ 데이터 무결성을 위한 제약조건 정의

CREATE TABLE

- 컬럼의 정의
  - ✓ 테이블을 구성하는 각 컬럼의 데이터 타입을 선택한 후에 널 값 허용 여부와 기본 값 필요 여부를 결정
  - ✓ CREATE TABLE 문으로 생성되는 테이블의 컬럼은 기본적으로 널 값이 허용 됨

➢ NOT NULL

- 컬럼이 널 값을 허용하지 않음을 의미하는 키워드
- 특히, 기본키를 구성하는 모든 컬럼은 널 값을 가질 수 없도록 반드시 NOT NULL 키워드를 표기

예) bookname VARCHAR2(20) NOT NULL

CREATE TABLE

컬럼의 정의

➢ DEFAULT

- 컬럼의 기본 값을 지정하는 키워드
- 기본 값을 지정할 때 숫자 데이터는 그대로 표현
- 문자열이나 날짜 데이터는 작은따옴표로 묶어서 표현 (문자열의 경우:대소문자 구분)

예) 적립금 NUMBER DEFAULT 0
예) 담당자 VARCHAR2(20) DEFAULT '홍길동'

예) CHAR(2) DEFAULT 'a0'
예) CHAR(2) DEFAULT 'A0'

'a0' ≠ 'A0

CREATE TABLE

키의 정의

✓ CREATE TABLE 문으로 테이블을 정의할 때 기본키, 대체키, 외래키를 지정할 수 있음

➢ PRIMARY KEY

- 기본키를 지정하는 키워드
- 모든 테이블에서 기본키는 반드시 하나만 지정할 수 있고 여러 개의 속성으로 구성 가능

예) PRIMARY KEY(고객번호)
예) PRIMARY KEY(고객번호, 도서번호)

➢ UNIQUE

- 대체키를 지정하는 키워드
- 대체키로 지정된 컬럼의 값은 유일성을 가지며, 기본키와 달리 널 값이 허용됨
- 한 테이블에서 여러 개 지정 가능

예) UNIQUE(고객명)

CREATE TABLE

- 키의 정의
  - FOREIGN KEY
    - 외래키를 지정하는 키워드
    - 외래키가 어떤 테이블의 무슨 속성을 참조하는지 REFERENCES 키워드 다음에 제시해야 함
    - 이는 참조 무결성 제약조건을 유지하기 위함
    - 이렇게 하면 참조되는 테이블에서 투플을 함부로 삭제하거나 변경하지 못함
    - 참조되는 테이블에서 투플을 삭제하거나 변경할 때 처리하는 방법을 다양하게 선택할 수 있음

예) FOREIGN KEY(소속부서) REFERENCES 부서(부서번호)

CREATE TABLE

- 키의 정의
  - FOREIGN KEY

• 참조되는 테이블에서 투플 삭제 시 처리 방법을 지정하는 옵션

- ON DELETE NO ACTION: 투플을 삭제하지 못하게 함
- ON DELETE CASCADE: 관련 투플을 함께 삭제함
- ON DELETE SET NULL: 관련 투플의 외래키 값을 NULL로 변경함
- ON DELETE SET DEFAULT: 관련 투플의 외래키 값을 미리 지정한 기본값으로 변경함

• 참조되는 테이블에서 투플 변경 시 처리 방법을 지정하는 옵션

- ON UPDATE NO ACTION: 투플을 변경하지 못하게 함
- ON UPDATE CASCADE: 관련 투플에서 외래키 값을 함께 변경함
- ON UPDATE SET NULL: 관련 투플의 외래키 값을 NULL로 변경함
- ON UPDATE SET DEFAULT: 관련 투플의 외래키 값을 미리 지정한 기본값으로 변경함

예) FOREIGN KEY(소속부서) REFERENCES 부서(부서번호)
ON DELETE CASCADE ON UPDATE CASCADE

CREATE TABLE

- 키의 정의
  - ➢ 참조 무결성 제약조건 유지를 위한 투플 삭제의 예

부서번호	부서이름
1	인사부
2	연구부
3	홍보부

사원번호	사원이름	소속부서
1001	정소화	3
1002	김용욱	1
1003	고명석	2

- 부서 테이블에서 홍보부 투플을 삭제하려고 할 때
  - ON DELETE NO ACTION: 부서 테이블을 참조하는 사원테이블이 존재하므로 부서 테이블의 투플을 삭제하지 못하게 함
  - ON DELETE CASCADE: 사원테이블에서 홍보부에 근무하는 정소화 사원 투플도 함께 삭제
  - ON DELETE SET NULL: 사원 테이블에서 정소화 사원의 소속부서 속성 값을 NULL로 변경함
  - ON DELETE SET DEFAULT: 사원 테이블에서 정소화 사원의 소속부서 속성 값을 기본값으로 변경

CREATE TABLE

- 키의 정의
  - ➢ 참조 무결성 제약조건 유지를 위한 투플 변경의 예

부서번호	부서이름
1	인사부
2	연구부
3	홍보부

사원번호	사원이름	소속부서
1001	정소화	3
1002	김용욱	1
1003	고명석	2

- 부서 테이블에서 홍보부 투플을 변경하려고 할 때
  - ON UPDATE NO ACTION: 부서 테이블을 참조하는 사원테이블이 존재하므로 부서 부서 테이블의 투플을 변경하지 못하게 함
  - ON UPDATE CASCADE: 사원테이블에서 홍보부에 근무하는 정소화 사원 투플도 함께 변경함
  - ON UPDATE SET NULL: 사원 테이블에서 정소화 사원의 소속부서 속성 값을 NULL로 변경함
  - ON UPDATE SET DEFAULT: 사원 테이블에서 정소화 사원의 소속부서 속성 값을 기본값으로 변경

CREATE TABLE

- 데이터 무결성 제약조건의 정의
  - CHECK
    - 테이블에 정확하고 유효한 데이터를 유지하기 위해 특정 속성에 대한 제약조건을 지정
    - 테이블에선 CHECK 키워드로 지정한 제약조건을 만족하는 투플만 존재하게 됨
    - 테이블에 새로운 투플을 삽입하거나 기존 투플을 수정할 때도 이 제약 조건을 반드시 지켜야 함

예) CHECK(재고량>=0 AND 재고량 <=10000)

- CONSTRAINT 키워드와 함께 고유 이름을 부여할 수도 있음
- 제약조건을 여러 개 지정할 때 고유 이름을 부여하면 테이블이 생성된 이후에 제약조건을 수정하거나 제거할 때 식별하기가 쉬움

예) CONSTRAINT CHK_CPY CHECK(제조업체='롯데제과')

CREATE TABLE

예제1

다음과 같은 속성을 가진 Book 테이블을 생성하시오.

- bookid(도서번호) – NUMBER
- bookname(도서이름) – VARCHAR2(20)
- publisher(출판사) – VARCHAR2(20)
- price(가격) – NUMBER

[제약사항이 없는 경우]

```
CREATE TABLE    Book1 (
 bookid         NUMBER,
 bookname       VARCHAR2(20),
 publisher      VARCHAR2(20),
 price          NUMBER);
```

CREATE TABLE

[기본키 지정]

```
CREATE TABLE    Book2 (
 bookid         NUMBER      PRIMARY KEY,
 bookname       VARCHAR2(40),
 publisher      VARCHAR2(40),
 price          NUMBER);
```

```
CREATE TABLE    Book3 (
 bookid         NUMBER,
 bookname       VARCHAR2(40),
 publisher      VARCHAR2(40),
 price          NUMBER
 PRIMARY KEY    (bookid) );
```

[bookid] 속성이 없어서 두개의 속성 bookname, publiser가 기본키가 될 때

```
CREATE TABLE    Book4 (
 bookname       VARCHAR2(40),
 publisher      VARCHAR2(40),
 price          NUMBER
 PRIMARY KEY    (bookname, publisher) );
```

CREATE TABLE

[복잡한 제약사항을 추가할 때]

Bookname는 NULL 값을 가질 수 없고, publisher는 같은 값이 있으면 안된다.
Price에 값이 입력되지 않을 경우 기본값 10000을 저장한다. 또 가격은 최소 1,000원이상으로 한다.

```
CREATE TABLE    Book5 (
 bookname       VARCHAR2(40)    NOT NULL,
 publisher      VARCHAR2(40)    UNIQUE,
 price          NUMBER          DEFAULT 10000   CHECK(price > 1000) ,
 PRIMARY KEY    (bookname, publisher) );
```

CREATE TABLE

예제2

다음과 같은 속성을 가진 Customer 테이블을 생성하시오.

- custid(고객번호) – NUMBER(2), 기본키
- name(이름) - VARCHAR2(40)
- address(주소) - VARCHAR2(40)
- phone(전화번호) – VARCHAR2(20)

```
CREATE TABLE    Customer (
 custid         NUMBER(2)   PRIMARY KEY,
 name           VARCHAR2(40),
 address        VARCHAR2(50),
 phone          VARCHAR2(20));
```

CREATE TABLE

예제3

다음과 같은 속성을 가진 Orders 테이블을 생성하시오.

- ordered(주문번호) – NUMBER(2), 기본키
- custid(고객번호) – NUMBER(2), NOT NULL 제약조건, 외래키(Customer.custid, 연쇄삭제)
- bookid(도서번호) - NUMBER(2), NOT NULL 제약조건
- saleprice(판매가격) – NUMBER(8)
- orderdate(판매일자) - DATE

```
CREATE TABLE     Orders (
 orderid         NUMBER(2) ,
 custid          NUMBER(2)   NOT NULL,
 bookid          NUMBER(2)   NOT NULL,
 saleprice       NUMBER(8),
 orderdate       DATE,
 PRIMARY KEY(orderid),
 FOREIGN KEY(custid) REFERENCES Customer(custid) ON DELETE CASCADE);
```

ALTER TABLE

- 테이블 변경 구문

```
ALTER TABLE 테이블명
    ADD 컬럼명 데이터타입 [NOT NULL] [기본값];
    MODIFY 컬럼명;
    DROP COLUMN 컬럼명;
    ADD CONSTRAINT 제약조건명 제약조건(컬럼명);
    DROP CONSTRAINT 제약조건명;
    RENAME COLUMN 기존_컬럼명 to 새로운_컬럼명;
```

- ALTER 문은 생성된 테이블의 컬럼과 속성에 관한 제약을 변경하며, 기본키 및 외래키를 변경함
- ADD, DROP는 컬럼을 추가하거나 제거할 때 사용
- MODIFY는 컬럼의 기본값을 설정하거나 삭제할 때 사용

ALTER TABLE

예제1

Book1 테이블에 VARCHAR2(13)의 자료형을 가진 isbn 속성을 추가하시오.

```
ALTER TABLE Book1  ADD isbn VARCHAR2(13);
```

예제2

Book1 테이블의 isbn 속성의 데이터 타입을 NUMBER형으로 변경하시오.

```
ALTER TABLE Book1 MODIFY isbn NUMBER;
```

예제3

Book1 테이블의 isbn 속성을 삭제하시오.

```
ALTER TABLE Book1 DROP COLUMN isbn;
```

ALTER TABLE

예제4

Book1 테이블의 bookname 속성에 NOT NULL 제약조건을 적용하시오.

```
ALTER TABLE Book1 MODIFY bookname VARCHAR(20) NOT NULL;
```

예제5

Book1 테이블의 bookid 속성을 기본키로 변경하시오.

```
ALTER TABLE Book1 ADD PRIMARY KEY(bookid);
```

RENAME TABLE

- 테이블 이름 변경

```
RENAME 기존 테이블명 TO 새로운 테이블명;
```

예제1

Book1 테이블의 이름을 Book6테이블로 변경하시오.

```
RENAME   Book1 TO  Book6;
```

DROP TABLE

- 테이블 삭제 구문

```
DROP TABLE 테이블명 [CASCADE CONSTRAINT];
```

- DROP 명령어를 사용하면 테이블의 모든 데이터 및 구조를 삭제한다.
- CASCADE CONSTRAINT 옵션은 해당 테이블 관계가 있었던 참조되는 제약조건에 대해서도 삭제함을 의미함

예제

Book2~6 테이블을 삭제하시오.

```
DROP TABLE Book2;
DROP TABLE Book3;
DROP TABLE Book4;
DROP TABLE Book5;
DROP TABLE Book6;
```

TRUNCATE TABLE

- 테이블 데이터 삭제 구문

```
TRUNCATE TABLE 테이블명;
```

- 테이블 자체가 삭제되는 것이 아니고, 해당 테이블에 들어 있던 모든 행들이 제거되고 저장 공간을 재사용 가능하도록 해제함
- 테이블의 구조를 완전히 삭제하기 위해서는 DROP TABLE 실행해야 함
- 데이터를 삭제하는 명령어는 DELETE 명령어도 있음
- TRUNCATE TABLE 명령어로 삭제하는 경우 정상적인 복구가 불가능하므로 주의

```
예) TRUNCATE TABLE Book1;
```

3. 데이터 조작어

데이터 조작어(DML)

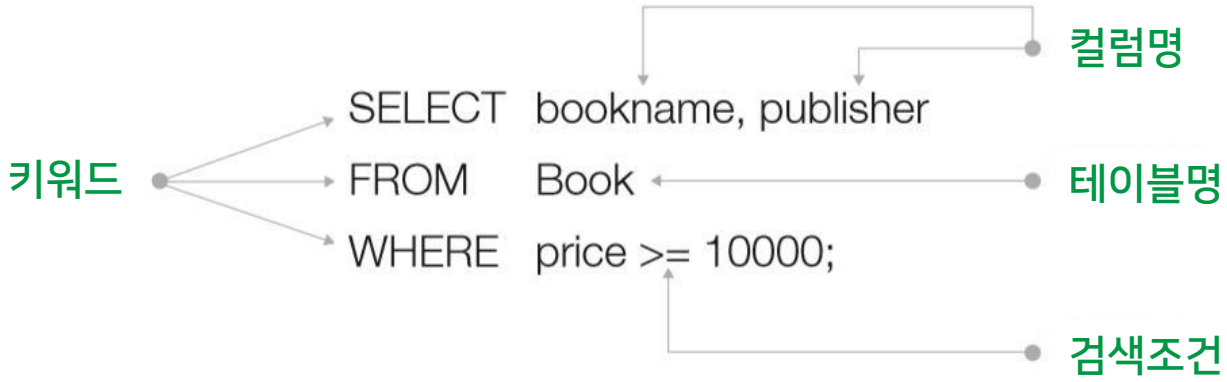
- 데이터 조작어의 기능
  - ✓ 테이블 검색, 새로운 데이터 삽입, 데이터 수정, 데이터 삭제

SQL문	
데이터 검색	SELECT
데이터 삽입	INSERT
데이터 수정	UPDATE
데이터 삭제	DELETE

SELECT

- SELECT 문의 기본 문법

SELECT	[ALL | DISTINCT] 컬럼명 [ALIAS명]
FROM	테이블명
WHERE	검색조건
GROUP BY	컬럼명
HAVING	그룹 검색조건
ORDER BY	컬럼명 [ASC | DESC]



SELECT

- SELECT
  - ✓ DISTINCT 옵션 : 중복된 데이터가 있을 경우 1건으로 처리
  - ✓ 애스터리스트(*) 사용: 해당 테이블의 모든 컬럼 정보를 보고 싶은 경우
  - ✓ ALIAS : 조회된 결과에 일종의 별명을 부여해 컬럼 레이블을 변경할 수 있음
    - 컬럼명 바로 뒤에 옴
    - 컬럼명과 ALIAS 사이에 AS, as 키워드를 사용할 수 있음(생략 가능)
    - 이중 인용부호는 ALIAS가 공백, 특수문자를 포함할 경우와 대소문자 구분이 필요할 때 사용

예) SELECT SUM(saleprice) 총매출

예) SELECT SUM(saleprice) AS “전체 매출”

SELECT

예제1

도서의 이름과 가격을 검색하시오.

```
SELECT  bookname, price
FROM    Book;
```

```
SELECT  price, bookname
FROM    Book;
```

BOOKNAME	PRICE
축구의 역사	7000
축구아는 여자	13000
축구의 이해	22000
골프 바이블	35000
피겨 교본	8000
역도 단계별기술	6000
야구의 추억	20000
야구를 부탁해	13000
올림픽 이야기	7500
Olympic Champions	13000

PRICE	BOOKNAME
7000	축구의 역사
13000	축구아는 여자
22000	축구의 이해
35000	골프 바이블
8000	피겨 교본
6000	역도 단계별기술
20000	야구의 추억
13000	야구를 부탁해
7500	올림픽 이야기
13000	Olympic Champions

예제2

도서의 도서번호, 도서이름, 출판사, 가격을 검색하시오.

```
SELECT  bookid, bookname, publisher, price
FROM    Book;
```

```
SELECT  *
FROM    Book;
```

BOOKID	BOOKNAME	PUBLISHER	PRICE
1	축구의 역사	굿스포츠	7000
2	축구아는 여자	나무수	13000
3	축구의 이해	대한미디어	22000
4	골프 바이블	대한미디어	35000
5	피겨 교본	굿스포츠	8000
6	역도 단계별기술	굿스포츠	6000
7	야구의 추억	이상미디어	20000
8	야구를 부탁해	이상미디어	13000
9	올림픽 이야기	삼성당	7500
10	Olympic Champions	Pearson	13000

SELECT

예제3

도서 테이블에 있는 출판사를 검색하시오.

```
SELECT  publisher
FROM    Book;
```

PUBLISHER
굿스포츠
나무수
대한미디어
대한미디어
굿스포츠
굿스포츠
이상미디어
이상미디어
삼성당
Pearson

예제4

도서 테이블에 있는 출판사를 중복 없이 검색하시오.

```
SELECT DISTINCT publisher
FROM   Book;
```

PUBLISHER
굿스포츠
삼성당
대한미디어
Pearson
나무수
이상미디어

SELECT

산술 연산자

산술연산자	설명
( )	연산자 우선순위를 변경하기 위한 괄호(괄호 안의 연산이 우선)
*	곱하기
/	나누기
+	더하기
-	빼기

예) SELECT name, height-weight AS "키-몸무게"
예) SELECT weight/((height/100)*(height/100)) AS BMI

SELECT

합성 연산자

✓ 문자와 문자를 연결하는 합성(CONCATENATION) 연산자를 사용하면 별도의 프로그램 도움 없이도 SQL 문장만으로 유용한 리포트를 출력할 수 있음

➢ 합성(CONCATENATION) 연산자의 특징

- 문자와 문자를 연결하는 경우 2개의 수직 바(||)를 사용(Oracle)
- 문자와 문자를 연결하는 겨우 + 표시를 사용(SQL Server)
- CONCAT(string1, string2) 함수를 사용할 수 있음 (Oracle, SQL Server)
- 컬럼과 문자 또는 다른 칼럼과 연결
- 문자 표현식의 결과에 의해 새로운 컬럼 생성

예) SELECT name || '학생,' || height || 'cm,' || weight || 'kg' AS 체격정보

WHERE

WHERE 절

```
SELECT      [ALL | DISTINCT]   컬럼명  [ALIAS명]
FROM        테이블명
WHERE       조건식;
```

➢ WHERE 절은 FROM 절 다음에 위치하며 조건식은 다음과 같이 구성

- 컬럼명(보통 조건식의 좌측에 위치)
- 비교연산자
- 문자, 숫자, 표현식(보통 조건식의 우측에 위치)
- 비교 컬럼명(Join 시)

WHERE

- 비교 연산자

연산자	설명
=	같다
>	보다 크다
>=	보다 크거나 같다
<	보다 작다
<=	보다 작거나 같다

WHERE

- 부정 비교 연산자

연산자	설명
!=	같지 않다
^=	같지 않다
<>	같지 않다(ISO 표준, 모든 운영체제에서 사용 가능
NOT 컬러명 =	~와 같지 않다
NOT 컬럼명 >	~보다 크지 않다

WHERE

- SQL 연산자

연산자	설명
BETWEEN a AND b	a와 b의 값 사이의 값을 갖는다(a와 b값이 포함)
IN (list)	리스트에 있는 값 중에서 어느 하나라도 일치
LIKE '비교문자열'	비교문자열과 형태가 일치(%, _)
IS NULL	NULL 값을 갖음

WHERE절

- SQL 연산자 - LIKE 연산자
  - ✓ 문자열의 패턴을 비교할 때 사용
  - ✓ 찾는 속성이 텍스트 혹은 날짜 데이터를 포함하면 비교 값에는 반드시 영문 작은따옴표(' ')로 둘러싸야 함

기호	설명
%	0개 이상의 문자 (문자의 내용과 개수는 상관 없음)
_	1개의 문자 (문자의 내용은 상관 없음)

사용 예	설명
LIKE '데이터%'	데이터로 시작하는 문자열 (데이터로 시작하기만 하면 길이는 상관 없음)
LIKE '%데이터'	데이터로 끝나는 문자열 (데이터로 끝나기만 하면 길이는 상관 없음)
LIKE '%데이터%'	데이터가 포함된 문자열
LIKE '데이터 _ _ _'	데이터로 시작하는 6자 길이의 문자열
LIKE '_ _ 한%'	세 번째 글자가 '한'인 문자열

WHERE절

SQL 연산자 - IS NULL, NOT NULL 연산자

- NULL 값 비교 연산은 정해진 문구 사용
- NULL 값을 찾을 때는 '='연산자가 아닌 IS NULL을 사용
- NULL이 아닌 값을 찾을 때는 '<>'가 아닌 IN NOT NULL을 사용

➢ NULL 의 특성

- 아직 지정되지 않은 값
- NULL 값은 '0', '' (빈 문자), ' ' (공백) 등과 다른 특별한 값
- NULL 값과의 수치연산은 NULL 값을 리턴
- NULL 값은 비교연산자를 통해 비교할 수 없고, 비교한다면 거짓(FALSE)을 리턴
- 어떤 값과 비교할 수도 없으며, 특정 값보다 크다/작다고 표현할 수 없음

WHERE

- 부정 SQL 연산자

연산자	설명
NOT BETWEEN a AND b	a와 b의 값 사이에 있지 않다(a, b 값을 포함하지 않음)
NOT IN (list)	List 값과 일치하지 않음
IS NOT NULL	NULL 값을 갖지 않음

WHERE

논리 연산자

연산자	설명
AND	앞의 있는 조건과 뒤에 오는 조건이 참(TRUE)이 되면 결과도 참(TRUE)이 됨. 즉, 앞의 조건과 뒤의 조건을 동시에 만족해야 함
OR	앞의 조건이 참(TRUE)이거나 두의 조건이 참(TRUE)이 돼야 결과도 참(TRUE)이 됨. 즉, 앞뒤의 조건 중 하나만 참(TRUE)이면 됨
NOT	뒤에 오는 조건에 반대되는 결과를 되돌려 줌

WHERE

연산자 우선순위

연산자 우선순위	설명
1	괄호( )
2	비교 연산자, SQL 연산자
3	NOT 연산자
4	AND
5	OR

WHERE

예제1

가격이 20,000원 미만인 도서를 검색하시오.

```
SELECT  *
FROM    Book
WHERE  price < 20000;
```

BOOKID	BOOKNAME	PUBLISHER	PRICE
1	축구의 역사	굿스포츠	7000
2	축구아는 여자	나무수	13000
5	피겨 교본	굿스포츠	8000
6	역도 단계별기술	굿스포츠	6000
8	야구를 부탁해	이상미디어	13000
9	올림픽 이야기	삼성당	7500
10	Olympic Champions	Pearson	13000

WHERE

예제2

가격이 10,000원 이상 20,000 이하인 도서를 검색하시오.

```
SELECT  *
FROM    Book
WHERE   price >= 10000 AND price <= 20000;
```

```
SELECT  *
FROM    Book
WHERE   price BETWEEN 10000 AND 20000;
```

BOOKID	BOOKNAME	PUBLISHER	PRICE
2	축구아는 여자	나무수	13000
7	야구의 추억	이상미디어	20000
8	야구를 부탁해	이상미디어	13000
10	Olympic Champions	Pearson	13000

WHERE

예제3

'축구의 역사'를 출간한 출판사를 검색하시오.

```
SELECT  bookname, publisher
FROM    Book
WHERE   bookname LIKE '축구의 역사';
```

BOOKNAME	PUBLISHER
축구의 역사	굿스포츠

예제4

도서이름에 '축구' 가 포함된 출판사를 검색하시오.

```
SELECT  bookname, publisher
FROM    Book
WHERE   bookname LIKE '%축구%';
```

BOOKNAME	PUBLISHER
축구의 역사	굿스포츠
축구아는 여자	나무수
축구의 이해	대한미디어

WHERE절

예제5

도서이름의 왼쪽 두 번째 위치에 '구'라는 문자열을 갖는 도서를 검색하시오.

```
SELECT  *
FROM    Book
WHERE  bookname LIKE '_구%';
```

BOOKID	BOOKNAME	PUBLISHER	PRICE
1	축구의 역사	굿스포츠	7000
2	축구아는 여자	나무수	13000
3	축구의 이해	대한미디어	22000
7	야구의 추억	이상미디어	20000
8	야구를 부탁해	이상미디어	13000

WHERE

예제6

출판사가 '굿스포츠' 혹은 '대한미디어' 인 도서를 검색하시오.

```
FROM    Book
WHERE   publisher IN ('굿스포츠', '대한미디어');
```

BOOKID	BOOKNAME	PUBLISHER	PRICE
1	축구의 역사	굿스포츠	7000
3	축구의 이해	대한미디어	22000
4	골프 바이블	대한미디어	35000
5	피겨 교본	굿스포츠	8000
6	역도 단계별기술	굿스포츠	6000

예제7

출판사가 '굿스포츠' 혹은 '대한미디어' 가 아닌 도서를 검색하시오.

```
SELECT  *
FROM    Book
WHERE   publisher NOT IN ('굿스포츠', '대한미디어');
```

BOOKID	BOOKNAME	PUBLISHER	PRICE
2	축구아는 여자	나무수	13000
7	야구의 추억	이상미디어	20000
8	야구를 부탁해	이상미디어	13000
10	Olympic Champions	Pearson	13000

WHERE

예제8

고객 테이블에서 핸드폰이 값이 null인 것을 찾으시오.

```
SELECT  *
FROM    customer
WHERE   phone IS NULL;
```

CUSTID	NAME	ADDRESS	PHONE
5	박세리	대한민국 대전	(null)

예제9

고객 테이블에서 핸드폰이 값이 null이 아닌 것을 찾으시오.

```
SELECT  *
FROM    customer
WHERE   phone IS NOT NULL;
```

CUSTID	NAME	ADDRESS	PHONE
1	박지성	영국 맨체스타	000-5000-0001
2	김연아	대한민국 서울	000-6000-0001
3	장미란	대한민국 강원도	000-7000-0001
4	추신수	미국 클리블랜드	000-8000-0001

WHERE

예제10

축구에 관한 도서 중 가격이 20,000원 이상인 도서를 검색하시오.

```
SELECT  *
FROM    Book
WHERE   bookname LIKE '%축구%' AND price >= 20000;
```

BOOKID	BOOKNAME	PUBLISHER	PRICE
3	축구의 이해	대한미디어	22000

WHERE

예제11

출판사가 '굿스포츠' 혹은 '대한미디어' 인 도서를 검색하시오.

```
SELECT  *
FROM    Book
WHERE  publisher ='굿스포츠' OR publisher ='대한미디어';
```

BOOKID	BOOKNAME	PUBLISHER	PRICE
1	축구의 역사	굿스포츠	7000
3	축구의 이해	대한미디어	22000
4	골프 바이블	대한미디어	35000
5	피겨 교본	굿스포츠	8000
6	역도 단계별기술	굿스포츠	6000

집계함수

집계 함수(Aggregate Function)

- ✓ 특성 속성 값을 통계적으로 계산한 결과를 검색하기 위해 집계 함수를 이용
- ✓ 여러 행들의 그룹이 모여 그룹당 단 하나의 결과를 돌려주는 함수
- ✓ GROUP BY 절은 행들을 소그룹화 함
- ✓ SELECT절, HAVING절, ORDER BY 절에 사용할 수 있음
- ✓ 집계 함수는 널인 속성 값은 제외하고 계산
- ✓ 집계 함수는 그룹에 대한 정보를 제공하므로 주로 숫자 유형에 사용되지만, MAX·MIN·COUNT 함수는 문자·날짜 유형에도 적용 가능한 함수 임

집계함수

- 집계 함수의 종류

집계 함수	사용 목적
COUNT(*)	NULL 값을 포함한 행의 수를 출력
COUNT(표현식)	표현식의 값이 NULL 값인 것을 제외한 행의 수를 출력
SUM([DISTINCT | ALL] 표현식)	표현식의 NULL 값을 제외한 합계를 출력
AVG([DISTINCT | ALL] 표현식)	표현식의 NULL 값을 제외한 평균을 출력
MAX([DISTINCT | ALL] 표현식)	표현식의 최대값을 출력 (문자, 날짜 데이터 타입도 사용 가능)
MIN([DISTINCT | ALL] 표현식)	표현식의 최소값을 출력 (문자, 날짜 데이터 타입도 사용 가능)
STDEV([DISTINCT | ALL] 표현식)	표현식의 표준편차를 출력
VARIANCE/VAR([DISTINCT | ALL] 표현식)	표현식의 분산을 출력
기타 통계 함수	벤더별로 다양한 통계식을 제공

집계함수

예제1

고객이 주문한 도서의 총 판매액을 구하시오.

```
SELECT  SUM(saleprice)
FROM    Orders;
```

SUM(SALEPRICE)
118000

예제2

고객이 주문한 도서의 총 판매액을 '총매출'이란 이름으로 출력하시오.

```
SELECT  SUM(saleprice) AS 총매출
FROM    Orders;
```

총매출
118000

집계함수

예제3

2번 김연아 고객이 주문한 도서의 총 판매액을 구하시오.

```
SELECT  SUM(saleprice) AS 총매출
FROM    Orders
WHERE   custid=2;
```

총매출
15000

예제4

고객이 주문한 도서의 총 판매액, 평균값, 최저가, 최고가를 구하시오.

```
SELECT  SUM(saleprice) AS Total,
        AVG(saleprice) AS Average,
        MIN(saleprice) AS Minimum,
        MAX(saleprice) AS Maximum
FROM    Orders;
```

TOTAL	AVERAGE	MINIMUM	MAXIMUM
118000	11800	6000	21000

집계함수

예제5

온라인 서점의 도서 판매 건수를 구하시오.

```
SELECT  COUNT(*)
FROM    Orders;
```

COUNT(*)
10

- 집계함수 COUNT는 행의 개수를 셈
- COUNT(*): 전체 행의 수 출력 (null 값 포함)
- COUNT(publisher): 출판사의 행의 수 (null 값 제외)
- COUNT(DISTINCT(publisher)): 중복을 제거한 출판사의 행의 수를 셈 (null 값 제외)

집계함수

- NULL 값에 대한 연산과 집계 함수
  - ✓ 'NULL+숫자' 연산의 결과는 NULL
  - ✓ 집계 함수 계산 시 NULL이 포함된 행은 집계에서 빠짐
  - ✓ 해당되는 행이 하나도 없을 경우 SUM, AVG 함수의 결과는 NULL이 되며, COUNT 함수의 결과는 0.

Mybook

bookid	price
1	10000
2	20000
3	NULL

```
SELECT  price+100
FROM    Mybook
WHERE   bookid=3
```

PRICE+100
(null)

집계함수

- NULL 값에 대한 연산과 집계 함수

Mybook

bookid	price
1	10000
2	20000
3	NULL

```
SELECT  SUM(price), AVG(price), COUNT(*),
COUNT(price)
FROM    Mybook;
```

SUM(PRICE)	AVG(PRICE)	COUNT(*)	COUNT(PRICE)
30000	15000	3	2

```
SELECT  SUM(price), AVG(price), COUNT(*)
FROM    Mybook
WHERE   bookid>=4;
```

SUM(PRICE)	AVG(PRICE)	COUNT(*)
(null)	(null)	0

GROUP BY, HAVING

- GROUP BY절과 HAVING 절의 특징
  - ✓ GROUP BY 절을 통해 소그룹별 기준을 정한 후, SELECT 절에 집계함수를 사용한다.
  - ✓ 집계함수의 통계 정보는 NULL 값을 가진 행을 제외하고 수행한다.
  - ✓ GROUP BY 절에서는 SELECT 절과는 달리 ALIAS 명을 사용할 수 없다.
  - ✓ 집계함수는 WHERE절에는 올 수 없다.
    (집계함수를 사용할 수 있는 GROUP BY 절보다 WHERE 절이 먼저 수행된다.)
  - ✓ WHERE 절은 전체 데이터를 GROUP으로 나누기 전에 행들을 미리 제거한다.
  - ✓ HAVING 절은 GROUP BY 절의 기준 항목이나 소그룹의 집계함수 이용한 조건을 표시할 수 있다.
  - ✓ GROUP BY 절에 의한 소그룹별로 만들어진 집계 데이터 중, HAVING 절에서 제한 조건을 두어 조건을 만족하는 내용만 출력한다.
  - ✓ HAVING 절은 일반적으로 GROUP BY 절 뒤에 위치한다.

GROUP BY, HAVING

예제1

고객별로 주문한 도서의 총 수량과 총 판매액을 구하시오.

```
SELECT  custid, COUNT(*) AS 도서수량, SUM(saleprice) AS 총액
FROM    Orders
GROUP BY    custid;
```

CUSTID	도서수량	총액
1	3	39000
2	2	15000
3	3	31000
4	2	33000

ORDERID	CUSTID	BOOKID	SALEPRICE	ORDERDATE
2	1	3	21000	20/07/03
6	1	2	12000	20/07/07
1	1	1	6000	20/07/01
9	2	10	7000	20/07/09
3	2	5	8000	20/07/03
4	3	6	6000	20/07/04
10	3	8	13000	20/07/10
8	3	10	12000	20/07/08
7	4	8	13000	20/07/07
5	4	7	20000	20/07/05

GROUP BY, HAVING

예제2

가격이 8,000원 이상인 도서를 구매한 고객에 대하여 고객별 주문 도서의 총 수량을 구하시오.
단, 두 권 이상 구매한 고객만 구하시오.

```
SELECT  custid, COUNT(*) AS 도서수량
FROM    Orders
WHERE   saleprice >= 8000
GROUP BY    custid
HAVING  count(*) >= 2;
```

CUSTID	도서수량
1	2
4	2
3	2

GROUP BY, HAVING

- GROUP BY와 HAVING 절의 문법과 주의사항

GROUP BY 〈속성〉	
주의사항	GROUP BY로 투플을 그룹으로 묶은 후 SELECT 절에는 GROUP BY에서 사용한 〈속성〉과 집계 함수만 나올 수 있다.
맞는 예	SELECT custid, SUM(saleprice) FROM Orders GROUP BY custid;
틀린 예	SELECT bookid, SUM(saleprice) /* SELECT 절에 bookid 속성이 올 수 없다 */ FROM Orders GROUP BY custid;

GROUP BY, HAVING

- GROUP BY와 HAVING 절의 문법과 주의사항

HAVING 〈검색조건〉	
주의사항	WHERE 절과 HAVING 절이 같이 포함된 SQL 문은 검색조건이 모호해질 수 있다. HAVING 절은 ① 반드시 GROUP BY 절과 같이 작성해야 하고 ② WHERE 절보다 뒤에 나와야 한다. 그리고 ③ 〈검색조건〉에는 SUM, AVG, MAX, MIN, COUNT와 같은 집계 함수가 와야 한다.
맞는 예	SELECT custid, COUNT(*) AS 도서수량 FROM Orders WHERE saleprice >= 8000 GROUP BY custid HAVING count(*) >= 2;
틀린 예	SELECT custid, COUNT(*) AS 도서수량 FROM Orders HAVING count(*) >= 2 /* 순서가 틀렸다 */ WHERE saleprice >= 8000 GROUP BY custid;

ORDER BY

ORDER BY 정렬

- ✓ SQL 문장으로 조회한 데이터들을 다양한 목적에 맞게 특정 컬럼을 기준으로 정렬·출력하는데 사용
- ✓ ORDER BY 절에 컬럼명 대신에 SELECT절에서 사용한 ALIAS명이나 컬럼 순서를 나타내는 정수도 사용 가능
- ✓ 별도로 정렬 방식을 지정하지 않으면 기본적으로 오름차순이 적용되며, SQL 문장의 제일 마지막에 위치

➢ 정렬 방식

- ASC(Ascending): 오름차순 정렬(기본 값으로 생략 가능)
- DESC(Descending): 내림차순 정렬

ORDER BY

- ORDER BY 절의 특징
  - ✓ 기본적인 정렬 순서는 오름차순(ASC)이다.
  - ✓ 숫자형 데이터 타입은 오름차순으로 정렬했을 경우에 가장 작은 값부터 출력된다.
  - ✓ 날짜형 데이터 타입은 오름차순으로 정렬했을 경우 날짜 값이 가장 빠른 값이 먼저 출력된다. '01-JAN-2012'는 '01-SEP-2012'보다 먼저 출력
  - ✓ Oracle에서는 NULL 값을 가장 큰 값으로 간주해 오름차순으로 정렬했을 경우에는 가장 마지막에, 내림차순으로 정렬했을 경우에는 가장 먼저 위치한다.
  - ✓ 반면 SQL Server에서는 NULL 값을 가장 작은 값으로 간주하므로 오름차순으로 정렬했을 경우에는 가장 먼저, 내림차순으로 정렬했을 경우에는 가장 마지막에 위치한다.
  - ✓ ORDER BY 절에서 컬럼명, ALIAS명, 컬럼 순서를 같이 혼용하는 것이 가능하다.

ORDER BY

예제1

도서를 이름순으로 검색하시오.

```
SELECT  *
FROM    Book
ORDER BY    bookname;
```

BOOKID	BOOKNAME	PUBLISHER	PRICE
10	Olympic Champions	Pearson	13000
4	골프 바이블	대한미디어	35000
8	야구를 부탁해	이상미디어	13000
7	야구의 추억	이상미디어	20000
6	역도 단계별기술	굿스포츠	6000
9	올림픽 이야기	삼성당	7500
2	축구아는 여자	나무수	13000
1	축구의 역사	굿스포츠	7000
3	축구의 이해	대한미디어	22000
5	피겨 교본	굿스포츠	8000

예제2

도서를 가격순으로 검색하고, 가격이 같으면 이름순으로 검색하시오.

```
SELECT  *
FROM    Book
ORDER BY    price, bookname;
```

BOOKID	BOOKNAME	PUBLISHER	PRICE
6	역도 단계별기술	굿스포츠	6000
1	축구의 역사	굿스포츠	7000
9	올림픽 이야기	삼성당	7500
5	피겨 교본	굿스포츠	8000
10	Olympic Champions	Pearson	13000
8	야구를 부탁해	이상미디어	13000
2	축구아는 여자	나무수	13000
7	야구의 추억	이상미디어	20000
3	축구의 이해	대한미디어	22000
4	골프 바이블	대한미디어	35000

ORDER BY

예제3

도서를 출판사의 오름차순으로 검색하시오.
만약 가격이 같다면 가격의 내림차순으로 출력하시오.

```
SELECT  *
FROM    Book
ORDER BY    publisher ASC, price DESC;
```

BOOKID	BOOKNAME	PUBLISHER	PRICE
10	Olympic Champions	Pearson	13000
5	피겨 교본	굿스포츠	8000
1	축구의 역사	굿스포츠	7000
6	역도 단계별기술	굿스포츠	6000
2	축구아는 여자	나무수	13000
4	골프 바이블	대한미디어	35000
3	축구의 이해	대한미디어	22000
9	올림픽 이야기	삼성당	7500
7	야구의 추억	이상미디어	20000
8	야구를 부탁해	이상미디어	13000

ORDER BY

- SQL 문의 실행 순서

```
SELECT      custid, COUNT(*) AS 도서수량   ⑤
FROM        Orders                        ①
WHERE       saleprice >= 8000             ②
GROUP BY    custid                        ③
HAVING      count(*) >= 2                 ④
ORDER BY    custid;                       ⑥
```

① FROM Orders

ORDERID	CUSTID	BOOKID	SALEPRICE	ORDERDATE
1	1	1	6000	20/07/01
2	1	3	21000	20/07/03
3	2	5	8000	20/07/03
4	3	6	6000	20/07/04
5	4	7	20000	20/07/05
6	1	2	12000	20/07/07
7	4	8	13000	20/07/07
8	3	10	12000	20/07/08
9	2	10	7000	20/07/09
10	3	8	13000	20/07/10

② WHERE saleprice >= 8000

ORDERID	CUSTID	BOOKID	SALEPRICE	ORDERDATE
2	1	3	21000	20/07/03
3	2	5	8000	20/07/03
5	4	7	20000	20/07/05
6	1	2	12000	20/07/07
7	4	8	13000	20/07/07
8	3	10	12000	20/07/08
10	3	8	13000	20/07/10

③ GROUP BY custid

CUSTID	COUNT(*)
1	2
2	1
4	2
3	2

④ HAVING count(*) >= 2

CUSTID	COUNT(*)
1	2
4	2
3	2

⑤ SELECT custid, count(*) AS 도서수량

CUSTID	도서수량
1	2
4	2
3	2

⑥ ORDER BY custid;

CUSTID	도서수량
1	2
3	2
4	2

INSERT

- INSERT 문의 기본 문법

➢ 단일 행 INSERT 문

```
INSERT INTO 테이블명 [(컬럼 리스트])
VALUES (값 리스트);
```

➢ 서브 쿼리를 이용한 다중 행 INSERT 문

```
INSERT INTO 테이블명 [(컬럼 리스트])
서브 쿼리;
```

INSERT

예제1

Book 테이블에 새로운 도서 '스포츠 의학'을 삽입하시오.

```
CREATE TABLE    Book (
 bookid         NUMBER,
 bookname       VARCHAR2(20),
 publisher      VARCHAR2(20),
 price          NUMBER);
```

```
INSERT INTO Book(bookid, bookname, publisher, price)
VALUES (11, '스포츠 의학', '한솔의학서적', 90000);
```

```
SELECT * FROM Book;
```

BOOKID	BOOKNAME	PUBLISHER	PRICE
11	스포츠 의학	한솔의학서적	90000

INSERT

예제2

Book 테이블에 새로운 도서 '스포츠 의학'을 삽입하시오.

```
INSERT INTO Book
VALUES (12, '스포츠 의학', '한솔의학서적', 90000);
```

```
INSERT INTO Book(bookid, bookname, price ,publisher)
VALUES (13, '스포츠 의학', 90000 , '한솔의학서적');
```

```
INSERT INTO Book(bookid, bookname, publisher)
VALUES (14, '스포츠 의학', '한솔의학서적');
```

```
SELECT * FROM Book;
```

BOOKID	BOOKNAME	PUBLISHER	PRICE
11	스포츠 의학	한솔의학서적	90000
12	스포츠 의학	한솔의학서적	90000
13	스포츠 의학	한솔의학서적	90000
14	스포츠 의학	한솔의학서적	(null)

INSERT

예제3

Customer 테이블을 다음과 같은 데이터를 삽입하시오.

```
INSERT INTO Customer VALUES (1, '박지성', '영국 맨체스타', '000-5000-0001');
INSERT INTO Customer VALUES (2, '김연아', '대한민국 서울', '000-6000-0001');
INSERT INTO Customer VALUES (3, '장미란', '대한민국 강원도', '000-7000-0001');
INSERT INTO Customer VALUES (4, '추신수', '미국 클리블랜드', '000-8000-0001');
INSERT INTO Customer VALUES (5, '박세리', '대한민국 대전', NULL);
```

```
SELECT * FROM Customer;
```

CUSTID	NAME	ADDRESS	PHONE
1	박지성	영국 맨체스타	000-5000-0001
2	김연아	대한민국 서울	000-6000-0001
3	장미란	대한민국 강원도	000-7000-0001
4	추신수	미국 클리블랜드	000-8000-0001
5	박세리	대한민국 대전	(null)

UPDATE

- UPDATE 문의 기본 문법

```
UPDATE      테이블명
SET         컬럼명1=값1 [, 컬럼명2=값2, …]
[WHERE      검색조건];
```

UPDATE

예제1

Customer 테이블에서 고객번호가 5인 고객의 주소를 '대한민국 부산'으로 변경하시오.

```
UPDATE Customer
SET    address ='대한민국 부산'
WHERE  custid=5;
```

```
SELECT * FROM Customer;
```

CUSTID	NAME	ADDRESS	PHONE
1	박지성	영국 맨체스타	000-5000-0001
2	김연아	대한민국 서울	000-6000-0001
3	장미란	대한민국 강원도	000-7000-0001
4	추신수	미국 클리블랜드	000-8000-0001
5	박세리	대한민국 부산	(null)

UPDATE

예제2

Customer 테이블에서 박세리 고객의 주소를 김연아 고객의 주소로 변경하시오.

```
UPDATE Customer
SET address = (SELECT    address
               FROM      Customer
               WHERE     name='김연아')
WHERE  name ='박세리';
```

```
SELECT * FROM Customer;
```

CUSTID	NAME	ADDRESS	PHONE
1	박지성	영국 맨체스타	000-5000-0001
2	김연아	대한민국 서울	000-6000-0001
3	장미란	대한민국 강원도	000-7000-0001
4	추신수	미국 클리블랜드	000-8000-0001
5	박세리	대한민국 서울	(null)

DELETE

- DELETE 문의 기본 문법

```
DELETE    FROM  테이블명
[WHERE    검색조건];
```

예제1

Customer 테이블에서 고객번호가 5인 고객을 삭제한 후 결과를 확인하시오.

```
DELETE  FROM    Customer
WHERE  custid=5;
```

```
SELECT * FROM Customer;
```

CUSTID	NAME	ADDRESS	PHONE
1	박지성	영국 맨체스타	000-5000-0001
2	김연아	대한민국 서울	000-6000-0001
3	장미란	대한민국 강원도	000-7000-0001
4	추신수	미국 클리블랜드	000-8000-0001

예제2

Customer 테이블의 모든 고객을 삭제하시오

```
DELETE  FROM    Customer;
```

```
SELECT * FROM    Customer;
```

CUSTID	NAME	ADDRESS	PHONE

MERGE

- MERGE 문의 기본 문법

```
MERGE
  INTO 타겟 테이블명
  USING 소스 테이블명
  ON (조인 조건식)
  WHEN MATCHED THEN
    UPDATE
      SET 수정할 컬럼명1 = 수정될 새로운 값1…
  WHEN NOT MATCHED THEN
    INSERT [(컬럼1, 컬럼2, …)]
   VALUES (값1, 값2, …)
;
```

- ✓ MERGE문을 사용하면 새로운 행을 입력하거나, 기존 행을 수정하는 작업을 한번에 할 수 있음
- ✓ ON 절의 조인 조건에 따라 조인에 성공한 행들에 대해서는 WHEN MATCHED THEN 아래 UPDATE 구문을 수행하고,
- ✓ 조인에 실패한 행들에 대해서는 WHEN NOT MATCHED THEN 아래 INSERT 구문을 실행

4. 조인(Join)

조인(Join)

- Customer 테이블을 Orders 테이블과 조건 없이 연결 한 경우
  - ✓ Customer와 Orders 테이블의 합체 결과 투플의 개수는 고객이 다섯 명이고 주문이 열개이므로 5×10 해서 50이 됨

```
SELECT  *
FROM    Customer, Orders;
```

	CUSTID	NAME	ADDRESS	PHONE	ORDERID	CUSTID_1	BOOKID	SALEPRICE	ORDERDATE
1	1	박지성	영국 맨체스타	000-5000-0001	1	1	1	6000	20/07/01
2	1	박지성	영국 맨체스타	000-5000-0001	2	1	3	21000	20/07/03
3	1	박지성	영국 맨체스타	000-5000-0001	3	2	5	8000	20/07/03
4	1	박지성	영국 맨체스타	000-5000-0001	4	3	6	6000	20/07/04
5	1	박지성	영국 맨체스타	000-5000-0001	5	4	7	20000	20/07/05
6	1	박지성	영국 맨체스타	000-5000-0001	6	1	2	12000	20/07/07
7	1	박지성	영국 맨체스타	000-5000-0001	7	4	8	13000	20/07/07
8	1	박지성	영국 맨체스타	000-5000-0001	8	3	10	12000	20/07/08
9	1	박지성	영국 맨체스타	000-5000-0001	9	2	10	7000	20/07/09
10	1	박지성	영국 맨체스타	000-5000-0001	10	3	8	13000	20/07/10
11	2	김연아	대한민국 서울	000-6000-0001	1	1	1	6000	20/07/01
12	2	김연아	대한민국 서울	000-6000-0001	2	1	3	21000	20/07/03
13	2	김연아	대한민국 서울	000-6000-0001	3	2	5	8000	20/07/03
14	2	김연아	대한민국 서울	000-6000-0001	4	3	6	6000	20/07/04
15	2	김연아	대한민국 서울	000-6000-0001	5	4	7	20000	20/07/05
16	2	김연아	대한민국 서울	000-6000-0001	6	1	2	12000	20/07/07
17	2	김연아	대한민국 서울	000-6000-0001	7	4	8	13000	20/07/07
18	2	김연아	대한민국 서울	000-6000-0001	8	3	10	12000	20/07/08
19	2	김연아	대한민국 서울	000-6000-0001	9	2	10	7000	20/07/09
20	2	김연아	대한민국 서울	000-6000-0001	10	3	8	13000	20/07/10
… 연속됨 …									
45	5	박세리	대한민국 대전	(null)	5	4	7	20000	20/07/05
46	5	박세리	대한민국 대전	(null)	6	1	2	12000	20/07/07
47	5	박세리	대한민국 대전	(null)	7	4	8	13000	20/07/07
48	5	박세리	대한민국 대전	(null)	8	3	10	12000	20/07/08
49	5	박세리	대한민국 대전	(null)	9	2	10	7000	20/07/09
50	5	박세리	대한민국 대전	(null)	10	3	8	13000	20/07/10

조인(Join)

- 조인
  - ✓ 조인이란 두 개 이상의 테이블들을 연결해 데이터를 출력하는 것
  - ✓ 일반적인 경우 행들은 PK와 FK 값의 연관에 의해 조인이 성립
  - ✓ 어떤 경우에는 이러한 PK, FK의 관계가 없어도 논리적인 값들의 연관만으로 조인이 성립될 수 있음
  - ✓ FROM 절에 여러 테이블이 나열되더라도 SQL에서 데이터를 처리할 때는 단 두개의 집합 간에만 조인이 일어남. FROM 절에 A, B, C 테이블이 나열됐더라도 특정 2개의 테이블만 먼저 조인 처리되고, 조인의 결과인 중간 데이터 집합과 남은 한 개의 테이블이 다음 차례로 조인

조인(Join)

- 여러 테이블에 대한 조인 검색
  - ✓ 조인 검색: 여러 개의 테이블을 연결하여 데이터를 검색하는 것
  - ✓ 조인 컬럼: 조인 검색을 위해 테이블을 연결해 주는 컬럼
    - 연결하려는 테이블 간에 조인 컬럼의 이름은 달라도 되지만 도메인을 같아야 함
    - 일반적으로 외래키를 조인 컬럼으로 이용함
  - ✓ FROM 절에 검색에 필요한 모든 테이블을 나열
  - ✓ WHERE 절에 조인 컬럼의 값이 같아야 함을 의미하는 조인 조건을 제시
  - ✓ 컬럼 이름 앞에 해당 컬럼이 소속된 테이블의 이름을 표시
    - 같은 이름의 컬럼이 서로 다른 테이블에 존재할 수 있기 때문에

조인(Join)

조인의 종류

종류	설명
등가 조인 (EQUI Join) 내부 조인 (Inner Join)	두 개의 테이블 간에 컬럼 값들이 서로 정확하게 일치하는 경우에 사용되는 방법
비등가 조인 (Non EQUI Join)	두 개의 테이블 간에 논리적인 연관 관계를 갖고 있으나, 컬럼 값들이 서로 일치하지 않은 경우에 사용
외부 조인 (Outer Join)	포괄 조인이라고도 하며, 테이블의 값이 Null일 경우에도 데이터를 출력해야 할 때 사용. 왼쪽(Left) 외부 조인, 오른쪽(right) 외부 조인, 전체(full) 외부 조인이 있음
크로스 조인 (Cross Join)	두 테이블 조인 시 가능한 모든 경우의 레코드를 Select 하는 조인으로 별도의 조건을 기술하지 않는 조인 방식
자연 조인 (Natural Join)	양쪽 테이블에서 데이터 타입 및 컬럼명이 같은 값이 존재할 대 자동으로 등가조인을 수행
셀프 조인 (Self Join)	셀프 조인이라고도 하며, 자신의 테이블을 조인함

등가 조인

- 등가 조인(EQUI JOIN), 내부 조인(INNER JOIN)
  - ✓ 두 개의 테이블 간에 컬럼 값들이 정확하게 일치하는 경우에 사용되는 방법으로 PK↔FK의 관계를 기반으로 함
  - ✓ 일반적으로 테이블 설계 시 나타난 PK↔FK의 관계를 이용하는 것인지 반드시 PK↔FK의 관계로만 EQUI JOIN이 성립하는 것은 아님.
  - ✓ Join의 조건은 Where 절에 기술하게 되는데 "=" 연산자를 사용해서 표현

```
SELECT    테이블명1.컬럼명, 테이블2.컬럼명, …
FROM      테이블1, 테이블2
WHERE     테이블2.컬럼명=테이블1.컬럼명;
```

```
SELECT    테이블명1.컬럼명, 테이블2.컬럼명, …
FROM      테이블1 INNER JOIN 테이블2
ON        테이블2.컬럼명=테이블1.컬럼명;
```

등가 조인

예제1

고객과 고객의 주문에 관한 데이터를 모두 보이시오.

```
SELECT  *
FROM    Customer, Orders
WHERE   Customer.custid=Orders.custid;
```

```
SELECT  *
FROM    Customer INNER JOIN Orders
ON      Customer.custid=Orders.custid;
```

CUSTID	NAME	ADDRESS	PHONE	ORDERID	CUSTID_1	BOOKID	SALEPRICE	ORDERDATE
1	박지성	영국 맨체스타	000-5000-0001	1	1	1	6000	20/07/01
1	박지성	영국 맨체스타	000-5000-0001	2	1	3	21000	20/07/03
2	김연아	대한민국 서울	000-6000-0001	3	2	5	8000	20/07/03
3	장미란	대한민국 강원도	000-7000-0001	4	3	6	6000	20/07/04
4	추신수	미국 클리블랜드	000-8000-0001	5	4	7	20000	20/07/05
1	박지성	영국 맨체스타	000-5000-0001	6	1	2	12000	20/07/07
4	추신수	미국 클리블랜드	000-8000-0001	7	4	8	13000	20/07/07
3	장미란	대한민국 강원도	000-7000-0001	8	3	10	12000	20/07/08
2	김연아	대한민국 서울	000-6000-0001	9	2	10	7000	20/07/09
3	장미란	대한민국 강원도	000-7000-0001	10	3	8	13000	20/07/10

등가 조인

예제2

고객과 고객의 주문에 관한 데이터를 고객별로 정렬하여 보이시오.

```
SELECT  *
FROM    Customer, Orders
WHERE  Customer.custid=Orders.custid
ORDER BY    Customer.custid;
```

CUSTID	NAME	ADDRESS	PHONE	ORDERID	CUSTID_1	BOOKID	SALEPRICE	ORDERDATE
1	박지성	영국 맨체스타	000-5000-0001	2	1	3	21000	20/07/03
1	박지성	영국 맨체스타	000-5000-0001	6	1	2	12000	20/07/07
1	박지성	영국 맨체스타	000-5000-0001	1	1	1	6000	20/07/01
2	김연아	대한민국 서울	000-6000-0001	9	2	10	7000	20/07/09
2	김연아	대한민국 서울	000-6000-0001	3	2	5	8000	20/07/03
3	장미란	대한민국 강원도	000-7000-0001	4	3	6	6000	20/07/04
3	장미란	대한민국 강원도	000-7000-0001	10	3	8	13000	20/07/10
3	장미란	대한민국 강원도	000-7000-0001	8	3	10	12000	20/07/08
4	추신수	미국 클리블랜드	000-8000-0001	7	4	8	13000	20/07/07
4	추신수	미국 클리블랜드	000-8000-0001	5	4	7	20000	20/07/05

등가 조인

예제3

고객의 이름과 고객이 주문한 도서의 판매가격을 검색하시오.

```
SELECT  name, saleprice
FROM    Customer, Orders
WHERE   Customer.custid=Orders.custid;
```

```
SELECT  name, saleprice
FROM    Customer INNER JOIN Orders
ON      Customer.custid=Orders.custid;
```

NAME	SALEPRICE
박지성	6000
박지성	21000
김연아	8000
장미란	6000
추신수	20000
박지성	12000
추신수	13000
장미란	12000
김연아	7000
장미란	13000

등가 조인

예제4

고객별로 주문한 모든 도서의 총 판매액을 구하고, 고객별로 정렬하시오.

```
SELECT      name, SUM(saleprice)
FROM        Customer, Orders
WHERE       Customer.custid=Orders.custid
GROUP BY    Customer.name
ORDER BY    Customer.name;
```

```
SELECT      name, SUM(saleprice)
FROM        Customer INNER JOIN Orders
ON          Customer.custid=Orders.custid
GROUP BY    Customer.name
ORDER BY    Customer.name;
```

NAME	SUM(SALEPRICE)
김연아	15000
박지성	39000
장미란	31000
추신수	33000

등가 조인

예제5

고객의 이름과 고객이 주문한 도서의 이름을 구하시오.

```
SELECT  c.name, b.bookname
FROM    Customer c, Orders o, Book b
WHERE   c.custid=o.custid AND o.bookid=b.bookid;
```

```
SELECT      c.name, b.bookname
FROM        Customer c
INNER JOIN  Orders o ON c.custid=o.custid
INNER JOIN  Book b ON o.bookid=b.bookid;
```

NAME	BOOKNAME
박지성	축구의 역사
박지성	축구아는 여자
박지성	축구의 이해
김연아	피겨 교본
장미란	역도 단계별기술
추신수	야구의 추억
추신수	야구를 부탁해
장미란	야구를 부탁해
장미란	Olympic Champions
김연아	Olympic Champions

등가 조인

고객의 이름과 고객이 주문한 도서의 이름을 구하시오.

CUSTID	NAME	ADDRESS	PHONE
1	박지성	영국 맨체스타	000-5000-0001
2	김연아	대한민국 서울	000-6000-0001
3	장미란	대한민국 강원도	000-7000-0001
4	추신수	미국 클리블랜드	000-8000-0001
5	박세리	대한민국 대전	(null)

BOOKID	BOOKNAME	PUBLISHER	PRICE
1	축구의 역사	굿스포츠	7000
2	축구아는 여자	나무수	13000
3	축구의 이해	대한미디어	22000
4	골프 바이블	대한미디어	35000
5	피겨 교본	굿스포츠	8000
6	역도 단계별기술	굿스포츠	6000
7	야구의 추억	이상미디어	20000
8	야구를 부탁해	이상미디어	13000
9	올림픽 이야기	삼성당	7500
10	Olympic Champions	Pearson	13000

ORDERID	CUSTID	BOOKID	SALEPRICE	ORDERDATE
1	1	1	6000	20/07/01
2	1	3	21000	20/07/03
3	2	5	8000	20/07/03
4	3	6	6000	20/07/04
5	4	7	20000	20/07/05
6	1	2	12000	20/07/07
7	4	8	13000	20/07/07
8	3	10	12000	20/07/08
9	2	10	7000	20/07/09
10	3	8	13000	20/07/10

등가 조인

예제6

가격이 20,000원인 도서를 주문한 고객의 이름과 도서의 이름을 구하시오.

```
SELECT    c.name, b.bookname
FROM      Customer c, Orders o, Book b
WHERE     c.custid=o.custid AND o.bookid=b.bookid AND b.price=20000;
```

```
SELECT    c.name, b.bookname
FROM      Customer c
INNER JOIN  Orders o ON  c.custid=o.custid
INNER JOIN  Book b   ON  o.bookid=b.bookid
WHERE     b.price=20000;
```

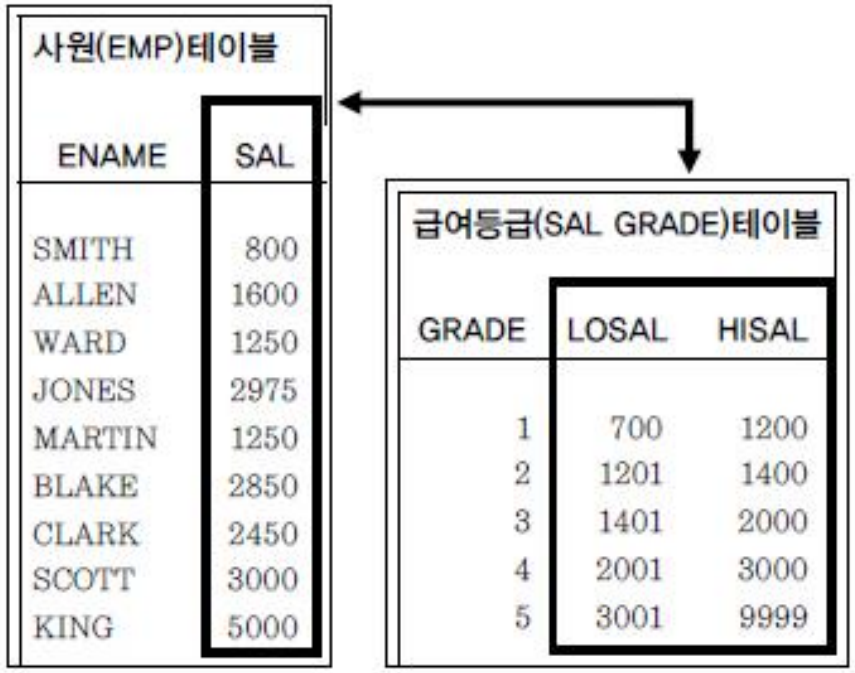
NAME	BOOKNAME
추신수	야구의 추억

비등가 조인

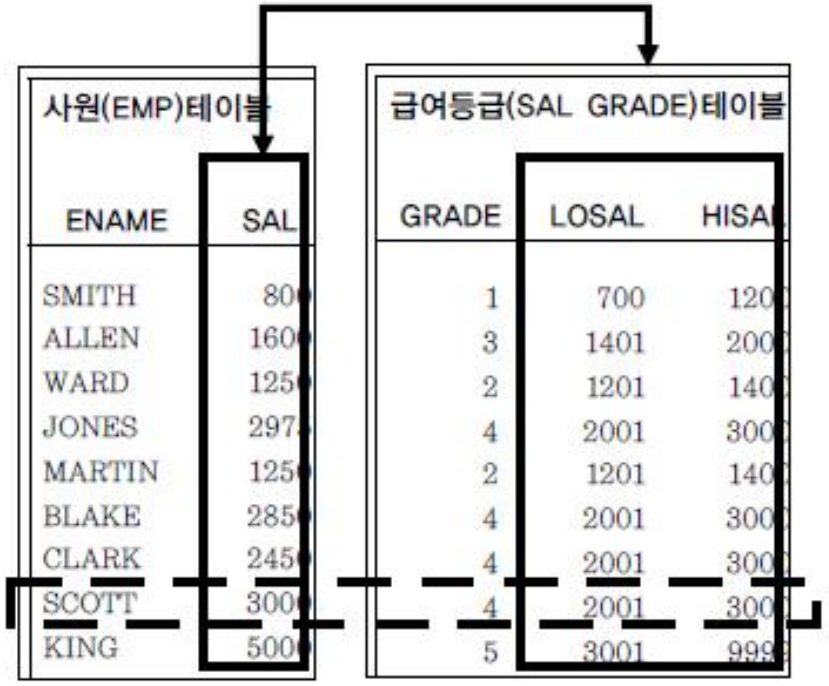
- 비등가 조인(Non EQUI JOIN)
  - ✓ 두 테이블 간에 논리적인 연관 관계를 갖고 있으나, 컬럼 값들이 서로 일치하지 않을 경우에 사용
  - ✓ Non EQUI JOIN 경우 "="연산자가 아닌 다른(Between, >, >=, <, <=) 연산자들을 사용해 조인을 수행
  - ✓ 두 테이블 간에 컬럼 값들이 서로 정확하게 일치하지 않는 경우에는 EQUI JION을 사용할 수 없으므로 이런 경우에는 Non EQUI JOIN을 시도할 수 있으나 데이터 모델에 따라 Non EQUI JOIN이 불가능할 경우도 있음

```
SELECT    테이블명1.컬럼명, 테이블2.컬럼명, …
FROM      테이블1, 테이블2
WHERE     테이블1.컬럼명 BETWEEN 테이블2.컬럼명 AND 테이블2.컬럼명2;
```

비등가 조인



<두 개의 테이블 관계도>



<데이터 재배열 후>

외부 조인

외부 조인(OUTER JOIN)

✓ 외부 조인은 조인을 만족하지 않은 행들도 함께 반환할 때 사용

✓ 조인 컬럼 뒤에 (+) 기호를 표시하여 나타냄

```
SELECT    테이블명1.컬럼명, 테이블2.컬럼명, …
FROM      테이블1, 테이블2
WHERE     테이블2.컬럼명(+)=테이블1.컬럼명;
```

- 외부조인 시 (+) 기호의 위치에 주의
- 외부조인의 기준이 되는 테이블(조인할 데이터가 없는 경우에도 모든 데이터를 표시하는 테이블)은 테이블1임
- 즉 (+) 표시의 반대편에 있는 테이블이 외부 조인의 기준 테이블이 됨

외부 조인

예제7

도서를 구매하지 않은 고객을 포함하여 고객의 이름과 고객이 주문한 도서의 판매가격을 구하시오.

```
SELECT      Customer.name, saleprice
FROM        CUSTOMER, ORDERS
WHERE       CUSTOMER.custid=ORDERS.custid(+);
```

```
SELECT      Customer.name, saleprice
FROM        Customer
LEFT OUTER JOIN  Orders
ON          Customer.custid=Orders.custid;
```

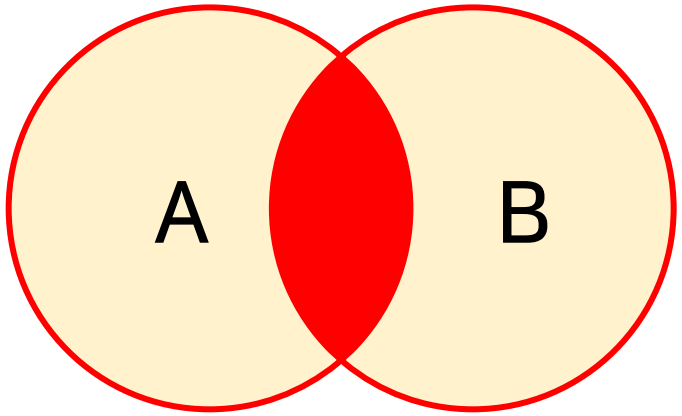
NAME	SALEPRICE
박지성	6000
박지성	21000
김연아	8000
장미란	6000
추신수	20000
박지성	12000
추신수	13000
장미란	12000
김연아	7000
장미란	13000
박세리	(null)

표준 조인

- FROM 절 조인 형태
  - ✓ ANSI/ISO SQL에서 표시하는 FROM 절의 조인 형태
    - INNER JOIN
    - OUTER JOIN
    - ON 조건절
    - CROSS JOIN
    - NATURAL JOIN
  - ✓ INNER JOIN은 조인의 DEFAULT 옵션으로 조인 조건을 만족하는 행들만 반환. DEFAULT 옵션이므로 생략 가능하지만, CROSS JOIN, OUTER JOIN과는 같이 사용할 수 없음
  - ✓ NATURAL JOIN은 inner JOIN의 하위 개념으로 볼 수 있으며, 두 테이블 간에 동일한 이름을 갖는 모든 컬럼들에 대해 EQUI JOIN을 수행
  - ✓ ANSI/ISO SQL 표준방식의 JOIN 문법에서 가장 두드러진 특징은 ON 조건절을 통해 JOIN 조건과 데이터 제한 조건을 분리해 기술하는 것임

내부 조인(Inner Join)

- 내부 조인 작성 시 주의사항
  - ✓ 조인 조건: 테이블 간의 관계를 표현
  - ✓ 조인에 사용되는 테이블 개수가 n이라면 조인 조건의 개수는 (n-1)개 이상이어야 함
  - ✓ FROM 절에서 테이블 별명을 지정할 수 있음
  - ✓ 동일한 이름의 컬럼이 여러 테이블에 있을 경우 테이블 이름(또는 별명)을 컬럼 앞에 지정해야 함
  - ✓ 조인 조건에 "="가 있는 것이 내부 조인
  - ✓ 내부(등가) 조인을 그림으로 나타내면 다음과 같이 두 테이블 A, B의 교집합에 해당하는 데이터를 추출해 낼 수 있음



내부 조인

예제

김밥천국 테이블을 생성하고 데이터를 삽입하시오.

```
CREATE TABLE 김밥천국(
        메뉴명  NVARCHAR2(10) NOT NULL,
        가격   NUMBER(4,0)
);

INSERT INTO 김밥천국  VALUES ('김밥',2500);
INSERT INTO 김밥천국  VALUES ('쫄면', 6000);
INSERT INTO 김밥천국  VALUES ('돈가스', 7000);
INSERT INTO 김밥천국  VALUES ('오징어덮밥', 7000);
INSERT INTO 김밥천국  VALUES ('돌솥비빔밥', 7000);
INSERT INTO 김밥천국  VALUES ('오므라이스', 6500);
INSERT INTO 김밥천국  VALUES ('잔치국수', 6000);
```

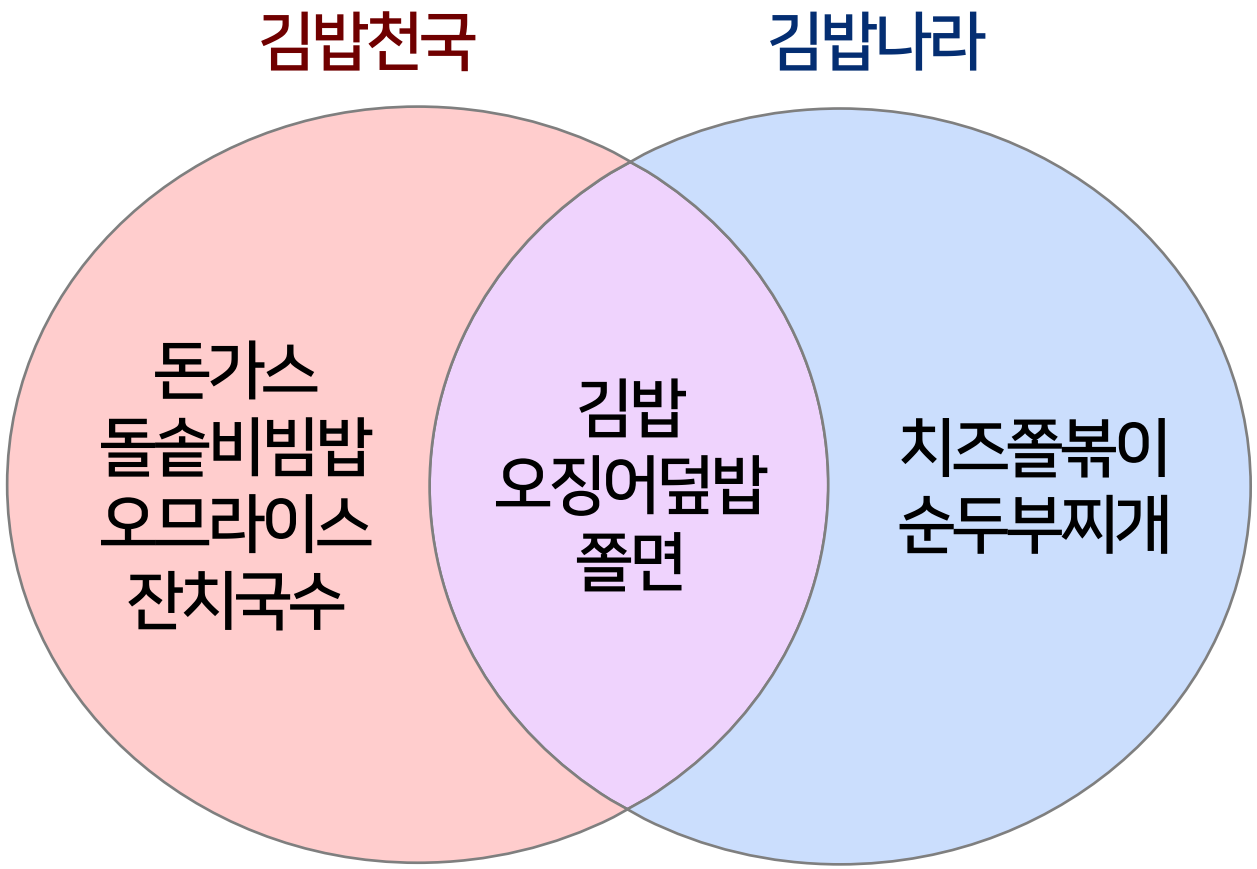
김밥나라 테이블을 생성하고 데이터를 삽입하시오.

```
CREATE TABLE 김밥나라(
        메뉴명 NVARCHAR2(10) NOT NULL,
        가격  NUMBER(4,0)
);

INSERT INTO 김밥나라  VALUES ('김밥', 2000);
INSERT INTO 김밥나라  VALUES ('쫄면', 5500);
INSERT INTO 김밥나라  VALUES ('오징어덮밥', 7000);
INSERT INTO 김밥나라  VALUES ('치즈쫄볶이', 4000);
INSERT INTO 김밥나라  VALUES ('순두부찌개', 6000);
```

내부 조인

김밥천국	김밥나라
김밥	김밥
쫄면	쫄면
돈가스	
오징어덮밥	오징어덮밥
돌솥비빔밥	
오므라이스	
잔치국수	
	치즈쫄볶이
	순두부찌개



내부 조인

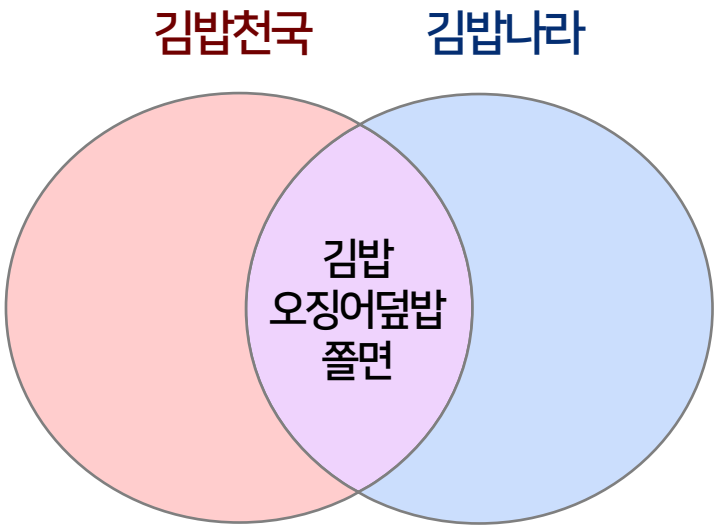
예제1

김밥천국과 김밥나라 테이블을 내부조인 하시오.

```
SELECT *
FROM 김밥천국, 김밥나라
WHERE 김밥천국.메뉴명=김밥나라.메뉴명;
```

```
SELECT 천.메뉴명, 천.가격
FROM 김밥천국 천, 김밥나라 나
WHERE 천.메뉴명=나.메뉴명;
```

```
SELECT 천.메뉴명, 천.가격
FROM 김밥천국 천
INNER JOIN 김밥나라 나 ON 천.메뉴명=나.메뉴명;
```



메뉴명	가격	메뉴명_1	가격_1
김밥	2500	김밥	2000
쫄면	6000	쫄면	5500
오징어덮밥	7000	오징어덮밥	7000

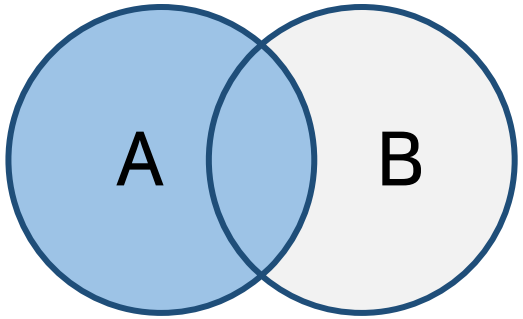
메뉴명	가격
김밥	2500
쫄면	6000
오징어덮밥	7000

외부 조인(Outer Join)

- 외부 조인이란
  - ✓ 두 테이블간 조인에서 조인 기준 열의 어느 한쪽이 NULL이어도 강제로 출력
  - ✓ 외부 조인은 좌우로 나누어 지정
    - LEFT OUTER JOIN (LEFT JOIN): 왼쪽 테이블을 기준으로 조인
    - RIGHT OUTER JOIN(RIGHT JOIN): 오른쪽 테이블을 기준으로 조인
    - FULL OUTER JOIN(FULL JOIN): 모든 NULL 값까지 출력

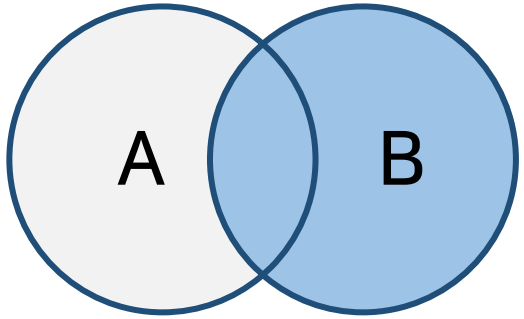
외부 조인(Outer Join)

LEFT JOIN



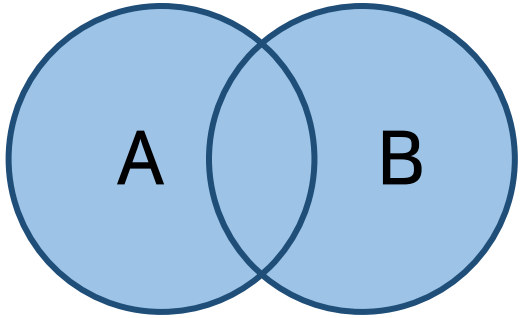

```
SELECT <select_list>
FROM  table A
LEFT JOIN Table B
ON A.key=B.key
```

RIGHT JOIN




```
SELECT <select_list>
FROM  table A
RIGHT JOIN Table B
ON A.key=B.key
```

FULL OUTER JOIN




```
SELECT <select_list>
FROM  table A
FULL OUTER JOIN Table B
ON A.key=B.key
```

외부 조인

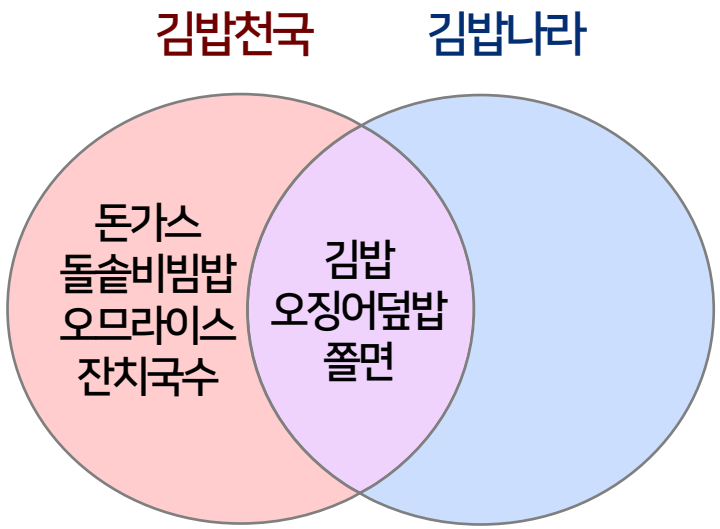
- LEFT JOIN

예제2

김밥천국과 김밥나라 테이블을 왼쪽 외부 조인하시오.

```
SELECT 천.메뉴명 AS 천메뉴, 천.가격 AS 천가격,
       나.메뉴명 AS 나메뉴, 나.가격 AS 나가격
FROM 김밥천국 천, 김밥나라 나
WHERE 천.메뉴명=나.메뉴명(+) ;
```

```
SELECT 천.메뉴명 AS 천메뉴, 천.가격 AS 천가격,
       나.메뉴명 AS 나메뉴, 나.가격 AS 나가격
FROM 김밥천국 천
LEFT JOIN 김밥나라 나 ON 천.메뉴명=나.메뉴명;
```



천메뉴	천가격	나메뉴	나가격
김밥	2500	김밥	2000
쫄면	6000	쫄면	5500
오징어덮밥	7000	오징어덮밥	7000
잔치국수	6000	(null)	(null)
돈가스	7000	(null)	(null)
오므라이스	6500	(null)	(null)
돌솥비빔밥	7000	(null)	(null)

외부 조인

- LEFT JOIN

예제3

김밥천국에만 있는 메뉴와 가격을 출력하시오.

```
SELECT 천.메뉴명 AS 천메뉴, 천.가격 AS 천가격,
       나.메뉴명 AS 나메뉴, 나.가격 AS 나가격
FROM 김밥천국 천
LEFT JOIN 김밥나라 나 ON 천.메뉴명=나.메뉴명
WHERE 나.메뉴명 IS NULL;
```

천메뉴	천가격	나메뉴	나가격
잔치국수	6000	(null)	(null)
돈가스	7000	(null)	(null)
오므라이스	6500	(null)	(null)
돌솥비빔밥	7000	(null)	(null)

외부 조인

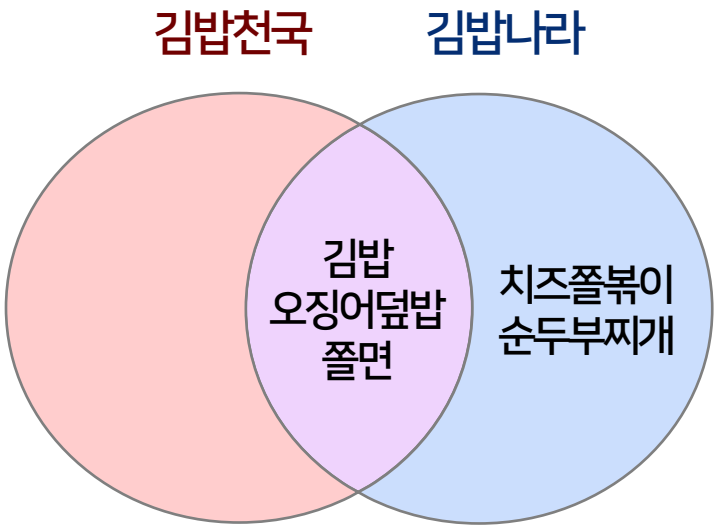
- RIGHT JOIN

예제4

김밥천국과 김밥나라 테이블을 오른쪽 외부 조인하시오.

```
SELECT 천.메뉴명 AS 천메뉴, 천.가격 AS 천가격,
       나.메뉴명 AS 나메뉴, 나.가격 AS 나가격
FROM 김밥천국 천, 김밥나라 나
WHERE 천.메뉴명(+)=나.메뉴명;
```

```
SELECT 천.메뉴명 AS 천메뉴, 천.가격 AS 천가격,
       나.메뉴명 AS 나메뉴, 나.가격 AS 나가격
FROM 김밥천국 천
RIGHT JOIN 김밥나라 나 ON 천.메뉴명=나.메뉴명;
```



천메뉴	천가격	나메뉴	나가격
김밥	2500	김밥	2000
쫄면	6000	쫄면	5500
오징어덮밥	7000	오징어덮밥	7000
(null)	(null)	순두부찌개	6000
(null)	(null)	치즈쫄볶이	4000

외부 조인

- RIGHT JOIN

예제5

김밥나라에만 있는 메뉴와 가격을 출력하시오.

```
SELECT 천.메뉴명 AS 천메뉴, 천.가격 AS 천가격,
       나.메뉴명 AS 나메뉴, 나.가격 AS 나가격
FROM 김밥천국 천
RIGHT JOIN 김밥나라 나 ON 천.메뉴명=나.메뉴명
WHERE 천.메뉴명 IS NULL;
```

천메뉴	천가격	나메뉴	나가격
(null)	(null)	순두부찌개	6000
(null)	(null)	치즈폴볶이	4000

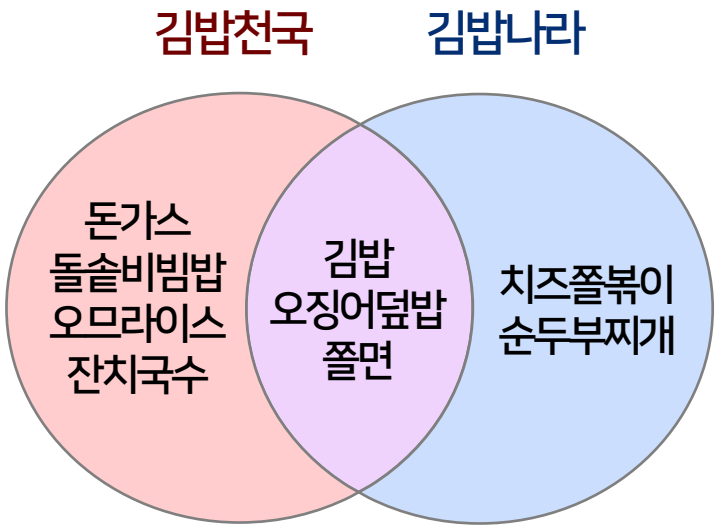
외부 조인

- FULL OUTER JOIN

예제6

김밥천국과 김밥나라 테이블을 FULL 외부 조인하시오.

```
SELECT 천.메뉴명 AS 천메뉴, 천.가격 AS 천가격,
       나.메뉴명 AS 나메뉴, 나.가격 AS 나가격
FROM 김밥천국 천, 김밥나라 나
WHERE 천.메뉴명=나.메뉴명(+)
UNION
SELECT 천.메뉴명 AS 천메뉴, 천.가격 AS 천가격,
       나.메뉴명 AS 나메뉴, 나.가격 AS 나가격
FROM 김밥천국 천, 김밥나라 나
WHERE 천.메뉴명(+)=나.메뉴명;
```



천메뉴	천가격	나메뉴	나가격
김밥	2500	김밥	2000
쫄면	6000	쫄면	5500
오징어덮밥	7000	오징어덮밥	7000
잔치국수	6000	(null)	(null)
돈가스	7000	(null)	(null)
오므라이스	6500	(null)	(null)
돌솥비빔밥	7000	(null)	(null)
(null)	(null)	순두부찌개	6000
(null)	(null)	치즈쫄볶이	4000

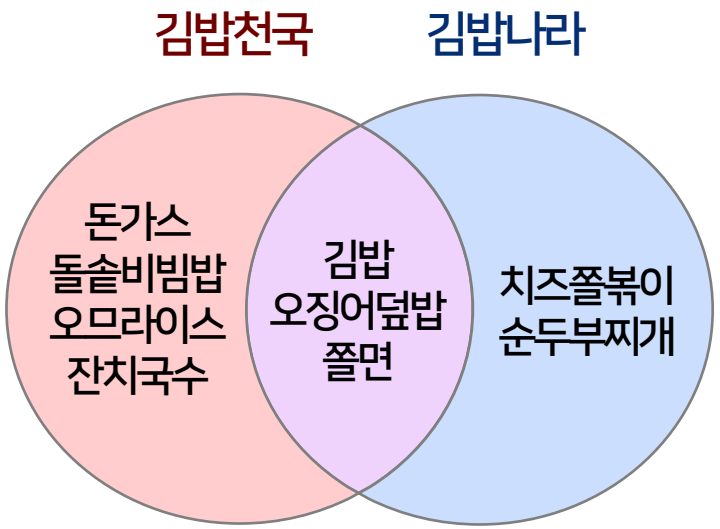
외부 조인

- FULL OUTER JOIN

예제6

김밥천국과 김밥나라 테이블을 FULL 외부 조인하시오.

```
SELECT 천.메뉴명 AS 천메뉴, 천.가격 AS 천가격,
       나.메뉴명 AS 나메뉴, 나.가격 AS 나가격
FROM 김밥천국 천
FULL  JOIN 김밥나라 나 ON 천.메뉴명=나.메뉴명;
```



천메뉴	천가격	나메뉴	나가격
김밥	2500	김밥	2000
쫄면	6000	쫄면	5500
오징어덮밥	7000	오징어덮밥	7000
잔치국수	6000	(null)	(null)
돈가스	7000	(null)	(null)
오므라이스	6500	(null)	(null)
돌솥비빔밥	7000	(null)	(null)
(null)	(null)	순두부찌개	6000
(null)	(null)	치즈쫄볶이	4000

외부 조인

FULL OUTER JOIN

예제7

김밥천국와 김밥나라 공통으로 있는 메뉴는 빼고 출력하시오.

```
SELECT 천.메뉴명 AS 천메뉴, 천.가격 AS 천가격,
       나.메뉴명 AS 나메뉴, 나.가격 AS 나가격
FROM 김밥천국 천
FULL JOIN 김밥나라 나 ON 천.메뉴명=나.메뉴명
WHERE 천.메뉴명 IS NULL OR 나.메뉴명 IS NULL;
```

천메뉴	천가격	나메뉴	나가격
돈가스	7000	(null)	(null)
돌솥비빔밥	7000	(null)	(null)
오므라이스	6500	(null)	(null)
잔치국수	6000	(null)	(null)
(null)	(null)	순두부찌개	6000
(null)	(null)	치즈폴볶이	4000

크로스 조인

크로스 조인(Cross Join)

✓ 크로스 조인은 테이블 상호간에 연결될 수 있는 모든 경우의 수를 나타내는 조인

예제8

김밥천국과 김밥나라를 크로스 조인하시오.

```
SELECT *
FROM 김밥천국, 김밥나라
```

메뉴명	가격	메뉴명_1	가격_1
김밥	2000	김밥	2500
김밥	2000	쫄면	6000
김밥	2000	돈가스	7000
김밥	2000	오징어덮밥	7000
김밥	2000	돌솥비빔밥	7000
김밥	2000	오므라이스	6500
김밥	2000	잔치국수	6000
쫄면	5500	김밥	2500
쫄면	5500	쫄면	6000
쫄면	5500	돈가스	7000
쫄면	5500	오징어덮밥	7000
쫄면	5500	돌솥비빔밥	7000
쫄면	5500	오므라이스	6500
쫄면	5500	잔치국수	6000
오징어덮밥	7000	김밥	2500
오징어덮밥	7000	쫄면	6000
오징어덮밥	7000	돈가스	7000
오징어덮밥	7000	오징어덮밥	7000
오징어덮밥	7000	돌솥비빔밥	7000
오징어덮밥	7000	오므라이스	6500
오징어덮밥	7000	잔치국수	6000
치즈쫄볶이	4000	김밥	2500
치즈쫄볶이	4000	쫄면	6000
치즈쫄볶이	4000	돈가스	7000
치즈쫄볶이	4000	오징어덮밥	7000
치즈쫄볶이	4000	돌솥비빔밥	7000
치즈쫄볶이	4000	오므라이스	6500
치즈쫄볶이	4000	잔치국수	6000
순두부찌개	6000	김밥	2500
순두부찌개	6000	쫄면	6000
순두부찌개	6000	돈가스	7000
순두부찌개	6000	오징어덮밥	7000
순두부찌개	6000	돌솥비빔밥	7000
순두부찌개	6000	오므라이스	6500
순두부찌개	6000	잔치국수	6000

다중 테이블 조인

예제

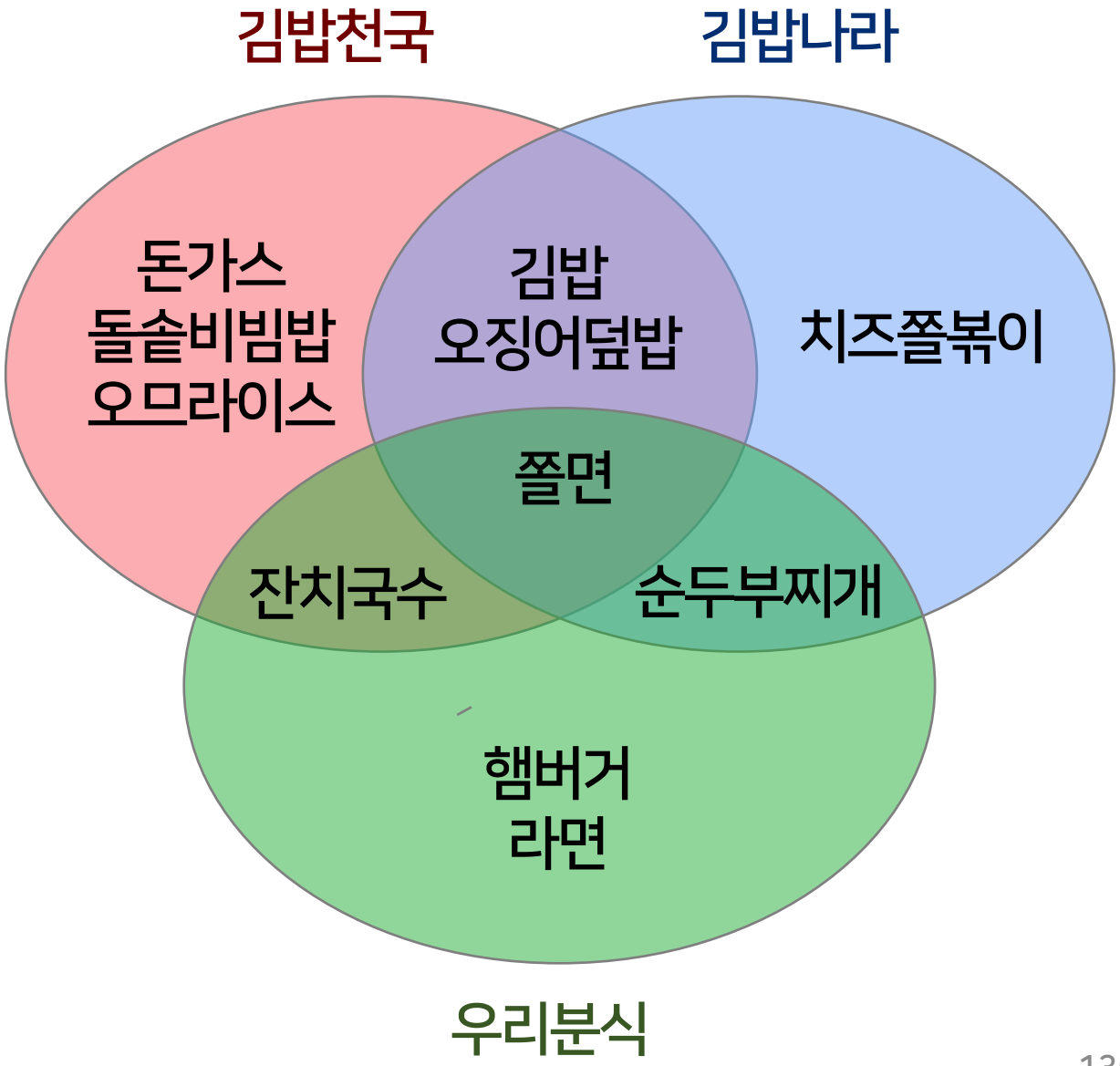
우리분식 테이블을 생성하고 데이터를 삽입하시오.

```
CREATE TABLE 우리분식(
        메뉴명 NVARCHAR2(10) NOT NULL,
        가격  NUMBER(4,0)
);

INSERT INTO 우리분식  VALUES ('라면', 3000);
INSERT INTO 우리분식  VALUES ('쫄면', 5500);
INSERT INTO 우리분식  VALUES ('잔치국수', 4500);
INSERT INTO 우리분식  VALUES ('순두부찌개', 6000);
INSERT INTO 우리분식  VALUES ('햄버거', 7000);
```

다중 테이블 조인

김밥천국	김밥나라	우리분식
김밥	김밥	
쫄면	쫄면	쫄면
돈가스		
오징어덮밥	오징어덮밥	
돌솥비빔밥		
오므라이스		
잔치국수		잔치국수
	치즈쫄볶이	
	순두부찌개	순두부찌개
		햄버거
		라면



다중 테이블 조인

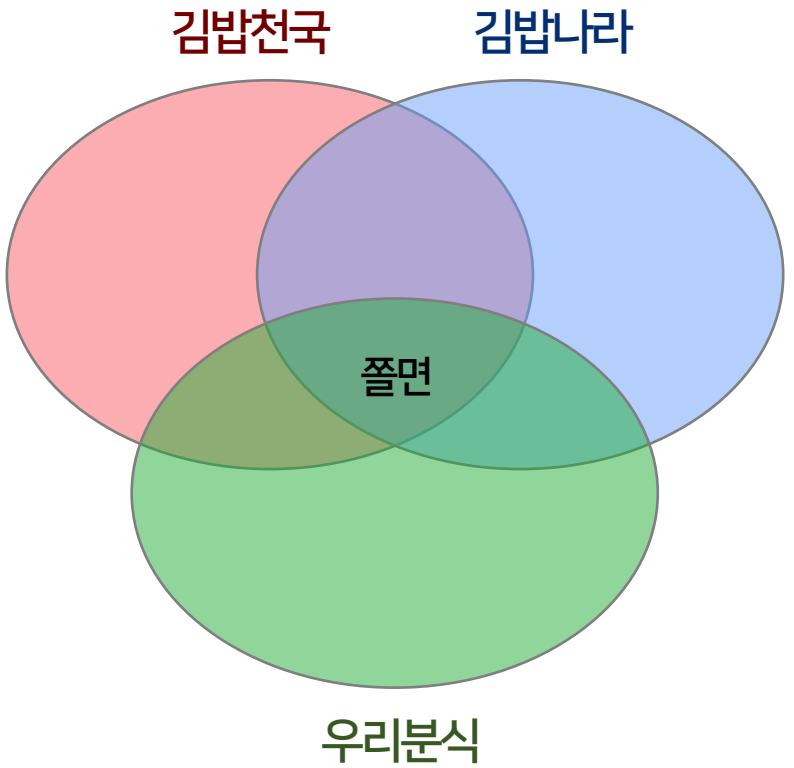
내부 조인

예제9

김밥천국, 김밥나라, 우리분식 테이블을 내부 조인하시오.

```
SELECT 천.메뉴명 AS 천메뉴, 나.메뉴명 AS 나메뉴,
       우.메뉴명 AS 우메뉴
FROM 김밥천국 천
JOIN 김밥나라 나 ON 천.메뉴명=나.메뉴명
JOIN 우리분식 우 ON 나.메뉴명=우.메뉴명;
```

천메뉴	나메뉴	우메뉴
쫄면	쫄면	쫄면



다중 테이블 조인

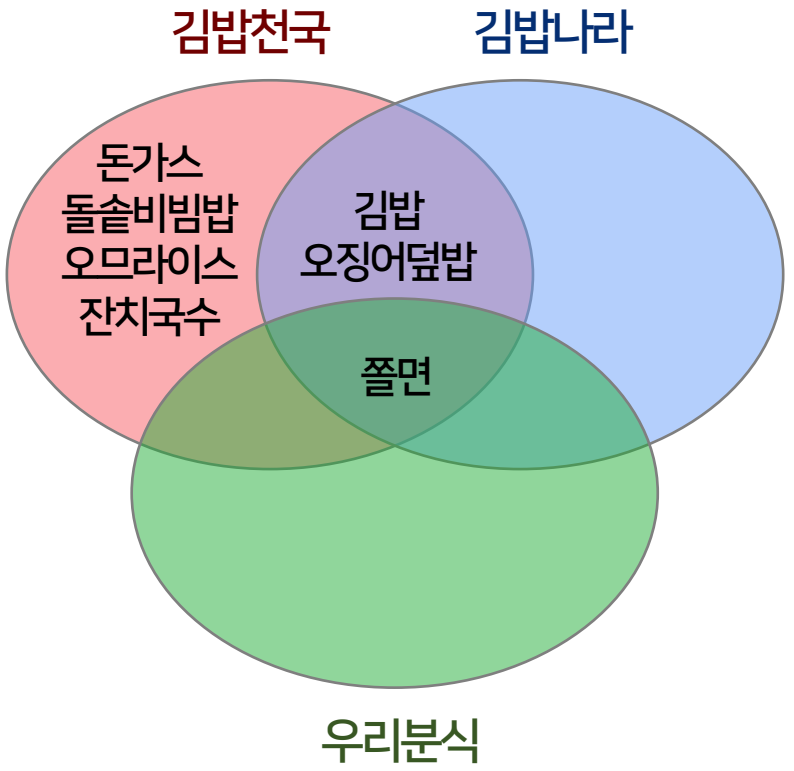
외부 조인

예제10-1

세개의 테이블에 대해서 LEFT JOIN을 실행하시오

```
SELECT 천.메뉴명 AS 천메뉴, 나.메뉴명 AS 나메뉴,
       우.메뉴명 AS 우메뉴
FROM 김밥천국 천
LEFT JOIN 김밥나라 나 ON 천.메뉴명=나.메뉴명
LEFT JOIN 우리분식 우 ON 나.메뉴명=우.메뉴명;
```

천메뉴	나메뉴	우메뉴
쫄면	쫄면	쫄면
잔치국수	(null)	(null)
돈가스	(null)	(null)
오므라이스	(null)	(null)
돌솥비빔밥	(null)	(null)
김밥	김밥	(null)
오징어덮밥	오징어덮밥	(null)



다중 테이블 조인

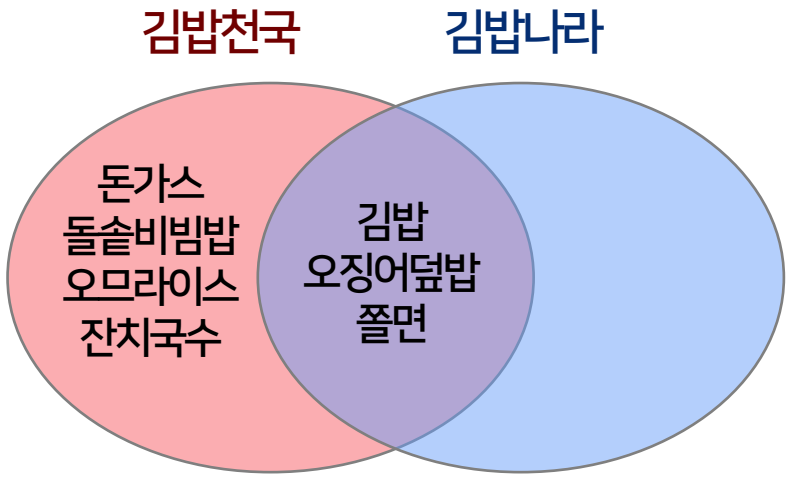
외부 조인

```
SELECT 천.메뉴명 AS 천메뉴, 나.메뉴명 AS 나메뉴,
       우.메뉴명 AS 우메뉴
FROM 김밥천국 천
LEFT JOIN 김밥나라 나 ON 천.메뉴명=나.메뉴명      ①
LEFT JOIN 우리분식 우 ON 나.메뉴명=우.메뉴명;     ②
```

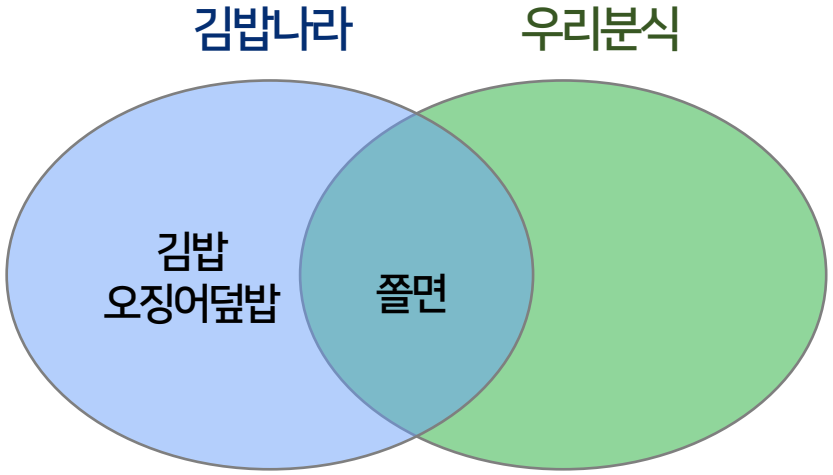
- 첫 번째와 두 번째 조인 그림을 보면 전부 LEFT JOIN
- 먼저 김밥천국과 김밥나라의 LEFT JOIN 후,
  김밥나라와 우리분식의 LEFT JOIN

천메뉴	나메뉴	우메뉴
쫄면	쫄면	쫄면
잔치국수	(null)	(null)
돈가스	(null)	(null)
오므라이스	(null)	(null)
돌솥비빔밥	(null)	(null)
김밥	김밥	(null)
오징어덮밥	오징어덮밥	(null)

1번째 조인



2번째 조인



다중 테이블 조인

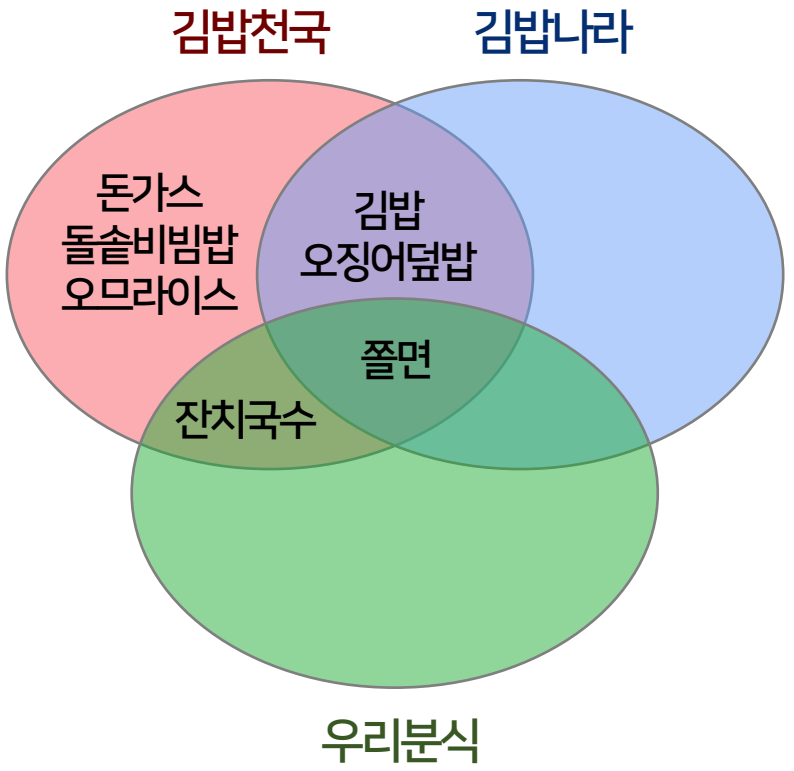
외부 조인

예제10-2

다중 외부 조인에서 두번째 JOIN의 조건을 바꾸면?

```
SELECT 천.메뉴명 AS 천메뉴, 나.메뉴명 AS 나메뉴,
       우.메뉴명 AS 우메뉴
FROM 김밥천국 천
LEFT JOIN 김밥나라 나 ON 천.메뉴명=나.메뉴명
LEFT JOIN 우리분식 우 ON 천.메뉴명=우.메뉴명;
```

메뉴명	메뉴명_1	메뉴명_2
쫄면	쫄면	쫄면
잔치국수	(null)	잔치국수
돈가스	(null)	(null)
오므라이스	(null)	(null)
김밥	김밥	(null)
오징어덮밥	오징어덮밥	(null)
돌솥비빔밥	(null)	(null)



다중 테이블 조인

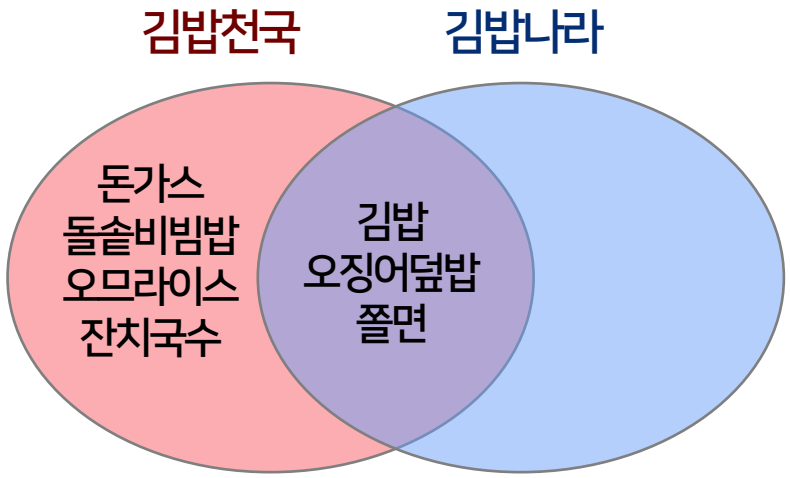
외부 조인

```
SELECT 천.메뉴명 AS 천메뉴, 나.메뉴명 AS 나메뉴,
       우.메뉴명 AS 우메뉴
FROM 김밥천국 천
LEFT JOIN 김밥나라 나 ON 천.메뉴명=나.메뉴명      ①
LEFT JOIN 우리분식 우 ON 천.메뉴명 =우.메뉴명;    ②
```

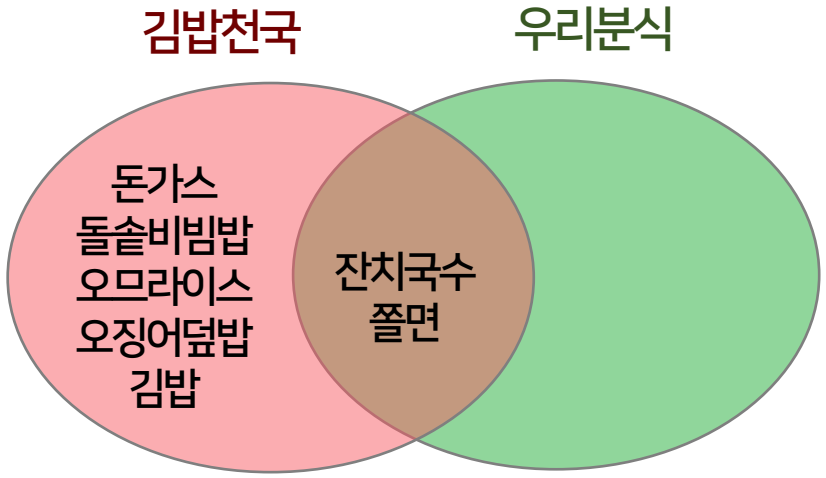
- 첫 번째 조인은 변경이 없고
- 두 번째 조인은 김밥천국과 우리분식의 LEFT JOIN

메뉴명	메뉴명_1	메뉴명_2
쫄면	쫄면	쫄면
잔치국수	(null)	잔치국수
돈가스	(null)	(null)
오므라이스	(null)	(null)
김밥	김밥	(null)
오징어덮밥	오징어덮밥	(null)
돌솥비빔밥	(null)	(null)

1번째 조인



2번째 조인



다중 테이블 조인

- LEFT JOIN을 다중 조인할 때 주의할 점
  - ✓ 조인을 2개가 아닌 3개 이상의 테이블을 조인하는 것은 어렵고 까다로운 작업
  - ✓ INNER JOIN과 달리 LEFT JOIN은 조인하는 테이블의 순서가 상당히 중요
  - ✓ 어떤 순서로 테이블을 조인하는지에 따라 결과 테이블에 조회되는 행의 개수와 구성 등이 달라짐
  - ✓ 3개 이상의 테이블을 LEFT JOIN 할 경우에는 다음 2가지를 기억할 것
    - 가장 첫 번째의 테이블을 SELECT 문에 가장 많은 열을 가져와야 할 테이블을 먼저 적을 것
    - 시작을 LEFT JOIN으로 했다면 나머지 조인도 LEFT JOIN을 할 것
      (LEFT JOIN을 쓰다가 갑자기 RIGHT JOIN이나 다른 조인을 사용하면 안됨)

다중 테이블 조인

외부 조인

예제10-3

다중 외부 조인에서 두번째 조인을 RIGHT JOIN로 바꾸면?

```
SELECT 천.메뉴명 AS 천메뉴, 나.메뉴명 AS 나메뉴,
       우.메뉴명 AS 우메뉴
FROM 김밥천국 천
LEFT JOIN 김밥나라 나 ON 천.메뉴명=나.메뉴명
RIGHT JOIN 우리분식 우 ON 천.메뉴명=우.메뉴명;
```

메뉴명	메뉴명_1	메뉴명_2
쫄면	쫄면	쫄면
잔치국수	(null)	잔치국수
(null)	(null)	순두부찌개
(null)	(null)	햄버거
(null)	(null)	라면

- 1번째 조인을 김밥천국 LEFT로 하여 김밥천국의 메뉴명이 다 나오기를 기대함
- 2번째 조인을 RIGHT 로 변경하였기 때문에 원하지 않는 결과를 얻음

다중 테이블 조인

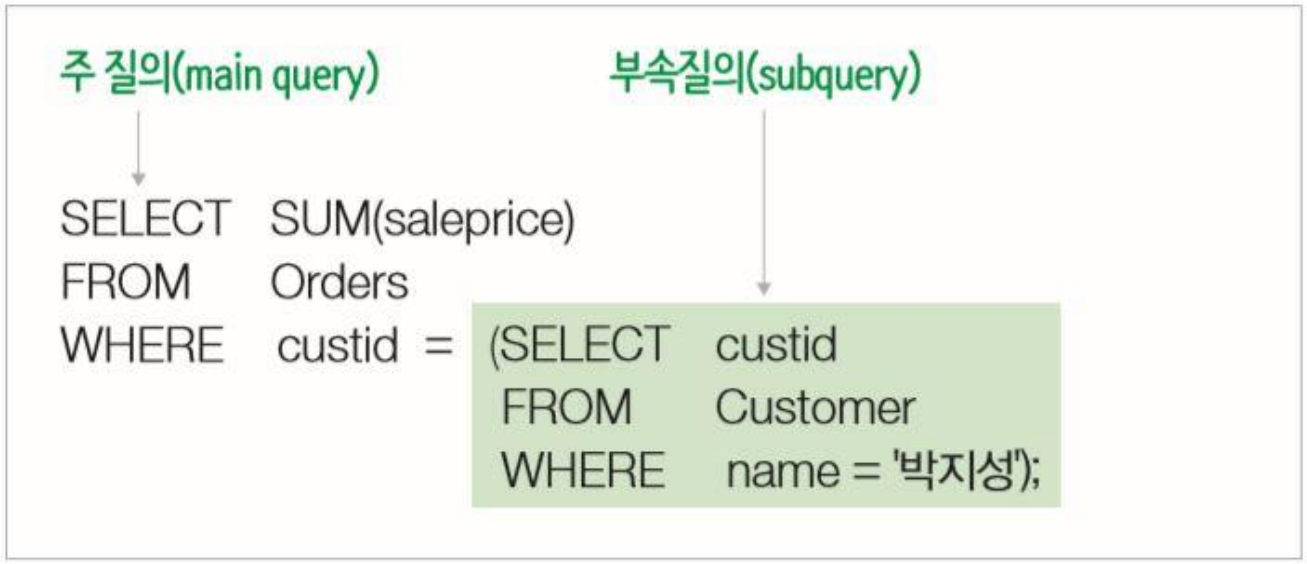
- 다중 테이블 조인을 정리하면
  - ✓ INNER JOIN은 각 테이블에서 조건에 맞는 값만 보여 줌
  - ✓ OUTER JOIN은 어떤 테이블을 주 테이블로 하느냐 즉, 모두 보여줄 테이블이 어떤 것인가에 따라 LEFT JOIN, RIGHT JOIN, FULL JOIN으로 분류
  - ✓ 주테이블을 무엇으로 지정 하는가에 따라 조인 결과가 달라지니 주의해야 함
  - ✓ 조인을 할 때 기준이 되는 주 테이블과 그 관계에 대해 잘 계산을 하면, 테이블이 3개나 4개 이상이라고 하더라도 그 관계를 해석하고 설계할 수 있음

5. 서브쿼리(Subquery)

서브 쿼리

서브 쿼리란

- ✓ 하나의 SQL 문 안에 다른 SQL 문이 중첩된nested 질의
- ✓ 다른 테이블에서 가져온 데이터로 현재 테이블에 있는 정보를 찾거나 가공할 때 사용
- ✓ 보통 데이터가 대량일 때 데이터를 모두 합쳐서 연산하는 조인보다 필요한 데이터만 찾아서 공급해주는 서브 쿼리가 성능이 더 좋음.
- ✓ 주 질의(main query, 외부질의)와 부속질의(sub query, 내부질의)로 구성



서브 쿼리

- 서브 쿼리 사용시 주의 사항
  - ✓ 서브 쿼리는 괄호로 감싸서 기술
  - ✓ 서브 쿼리는 단일 행(Single Row) 또는 복수 행(Multiple Row) 비교 연산자와 함께 사용 가능
    단일 행 비교 연산자는 서브 쿼리의 결과가 반드시 1건 이하이어야 하고, 복수 행 비교 연산자는 서브 쿼리의 결과 건수와 상관 없음
  - ✓ 중첩 서브 쿼리 및 스칼라 서브 쿼리에서는 ORDER BY를 사용하지 못함

서브 쿼리

- 동작하는 방식에 따른 서브 쿼리 분류

서브 쿼리 종류	설명
Un-Correlated(비연관) 서브 쿼리	서브 쿼리가 메인 쿼리 컬럼을 갖고 있는 않은 형태의 서브 쿼리. 메인 쿼리에 값(서브 쿼리가 실행된 결과)을 제공하기 위한 목적으로 주로 사용
Correlated(연관) 서브 쿼리	서브 쿼리가 메인 쿼리 컬럼을 갖고 있는 형태의 서브 쿼리 일반적으로 메인 쿼리가 먼저 수행돼 읽혀진 데이터를 서브 쿼리에서 조건이 맞는지 확인하고자 할 때 사용

서브 쿼리

- 반환되는 데이터의 형태에 따른 서브 쿼리 분류

서브 쿼리 종류	설명
Single Row 서브 쿼리 (단일 행 서브 쿼리)	서브 쿼리의 실행 결과가 항상 1건 이하인 서브 쿼리를 의미 단일 행 서브 쿼리는 단일 행 비교 연산자와 함께 사용 단일 행 비교 연산자에는 =, <, >, <=, >=, <> 등이 있음
Multi Row 서브 쿼리 (다중 행 부속질의)	서브 쿼리의 실행 결과가 여러 건인 서브 쿼리를 의미 다중 행 서브 쿼리는 다중 행 비교 연산자와 함께 사용 다중 행 비교 연산자에는 IN, ALL, ANY, SOME, EXISTS가 있음
Multi Column 서브 쿼리 (다중 컬럼 부속 질의)	서브 쿼리의 실행 결과로 여러 컬럼을 반환 메인 쿼리의 조건절에 여러 컬럼을 동시에 비교할 수 있음 서브 쿼리와 메인 쿼리에서 비교하고자 하는 컬럼 개수와 컬럼의 위치가 동일해야 함

서브 쿼리

- 단일 행 서브 쿼리
  - ✓ 서브 쿼리가 단일 행 비교 연산자(=, <, <=, >, >=, <> 등)와 함께 사용될 때는 서브 쿼리의 결과 건수가 반드시 1건 이하이어야 함
  - ✓ 만약 서브 쿼리의 결과 건수가 2건 이상을 반환하면 SQL문은 실행시간(Run Time) 오류가 발생

서브 쿼리

다중 행 서브 쿼리

✓ 서브 쿼리의 결과가 2건 이상 반환될 수 있다면, 반드시 다중 행 비교 연산자(IN, ALL, ANY, SOME)와 함께 사용해야 함. 그렇지 않으면 SQL 문은 오류를 반환

연산자	설명
IN	서브 쿼리의 결과 값 중 일치하는 것이 있으면 검색조건이 참
NOT IN	서브 쿼리의 결과 값 중 일치하는 것이 없으면 검색조건이 참
EXISTS	서브 쿼리의 결과 값이 하나라도 존재하면 검색조건이 참
NOT EXISTS	서브 쿼리의 결과 값이 하나라도 존재하지 않으면 검색조건이 참
ALL	서브 쿼리의 결과 값 모두와 비교한 결과가 참이면 검색 조건을 만족
ANY 또는 SOME	서브 쿼리의 결과 값 중 하나라도 비교한 결과가 참이면 검색 조건을 만족

서브 쿼리

예제1

가장 비싼 도서의 이름을 보이시오.

```
SELECT   MAX(price)
FROM     Book;
```

MAX(PRICE)
35000

```
SELECT   bookname
FROM     Book
WHERE  price=35000;
```

BOOKNAME
골프 바이블

```
SELECT  bookname
FROM     Book
WHERE  price =    (SELECT MAX(price)
                   FROM Book);
```

BOOKNAME
골프 바이블

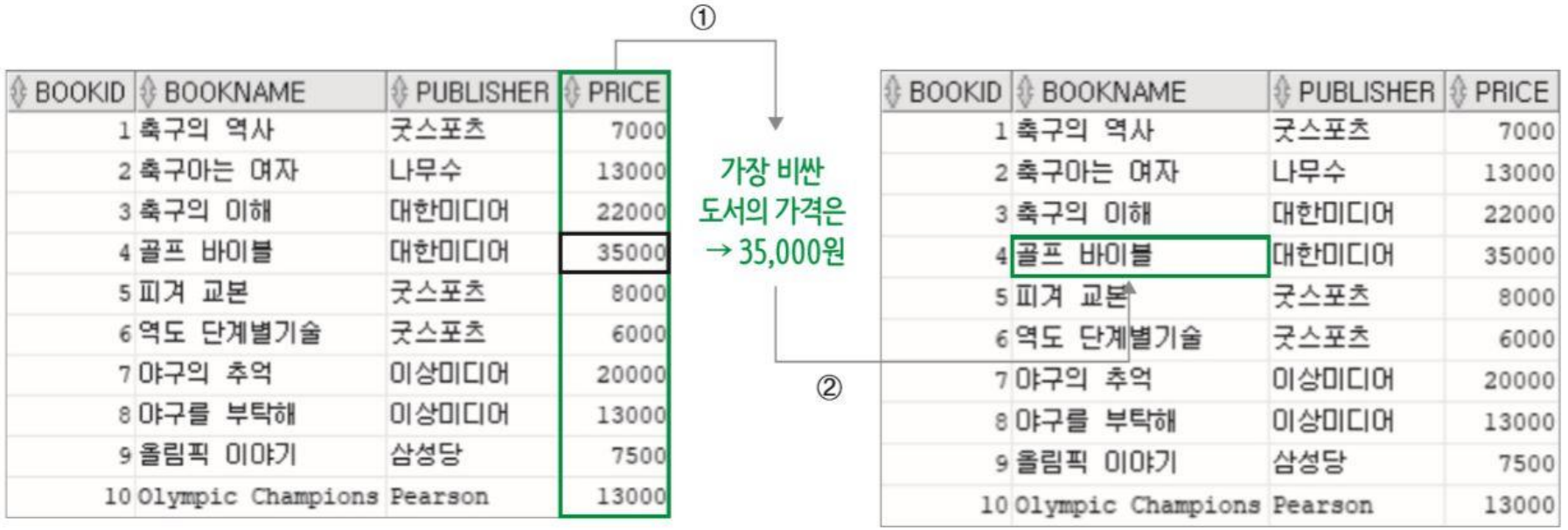
서브 쿼리

예제1

가장 비싼 도서의 이름을 보이시오.

```
SELECT  bookname
FROM    Book
WHERE   price = (SELECT MAX(price)
                 FROM Book);
```

BOOKNAME
골프 바이블



BOOKID	BOOKNAME	PUBLISHER	PRICE
1	축구의 역사	굿스포츠	7000
2	축구아는 여자	나무수	13000
3	축구의 이해	대한미디어	22000
4	골프 바이블	대한미디어	35000
5	피겨 교본	굿스포츠	8000
6	역도 단계별기술	굿스포츠	6000
7	야구의 추억	이상미디어	20000
8	야구를 부탁해	이상미디어	13000
9	올림픽 이야기	삼성당	7500
10	Olympic Champions	Pearson	13000

BOOKID	BOOKNAME	PUBLISHER	PRICE
1	축구의 역사	굿스포츠	7000
2	축구아는 여자	나무수	13000
3	축구의 이해	대한미디어	22000
4	골프 바이블	대한미디어	35000
5	피겨 교본	굿스포츠	8000
6	역도 단계별기술	굿스포츠	6000
7	야구의 추억	이상미디어	20000
8	야구를 부탁해	이상미디어	13000
9	올림픽 이야기	삼성당	7500
10	Olympic Champions	Pearson	13000

서브 쿼리

예제2

도서를 구매한 적이 있는 고객의 이름을 검색하시오.

```
SELECT   custid
FROM     Orders;
```

```
SELECT   name
FROM     Customer
WHERE  custid IN(1, 2, 3, 4);
```



```
SELECT  name
FROM     Customer
WHERE  custid IN (SELECT custid
                  FROM Orders);
```

CUSTID
1
1
2
3
4
1
4
3
2
3

NAME
박지성
김연아
장미란
추신수

서브 쿼리

예제3

'대한미디어'에서 출판한 도서를 구매한 고객의 이름을 보이시오.

```
SELECT  name
FROM    Customer
WHERE   custid IN(SELECT  custid
                  FROM    Orders
                   WHERE  bookid IN(SELECT       bookid
                                    FROM         Book
                                    WHERE        publisher='대한미디어'));
```

NAME
박지성

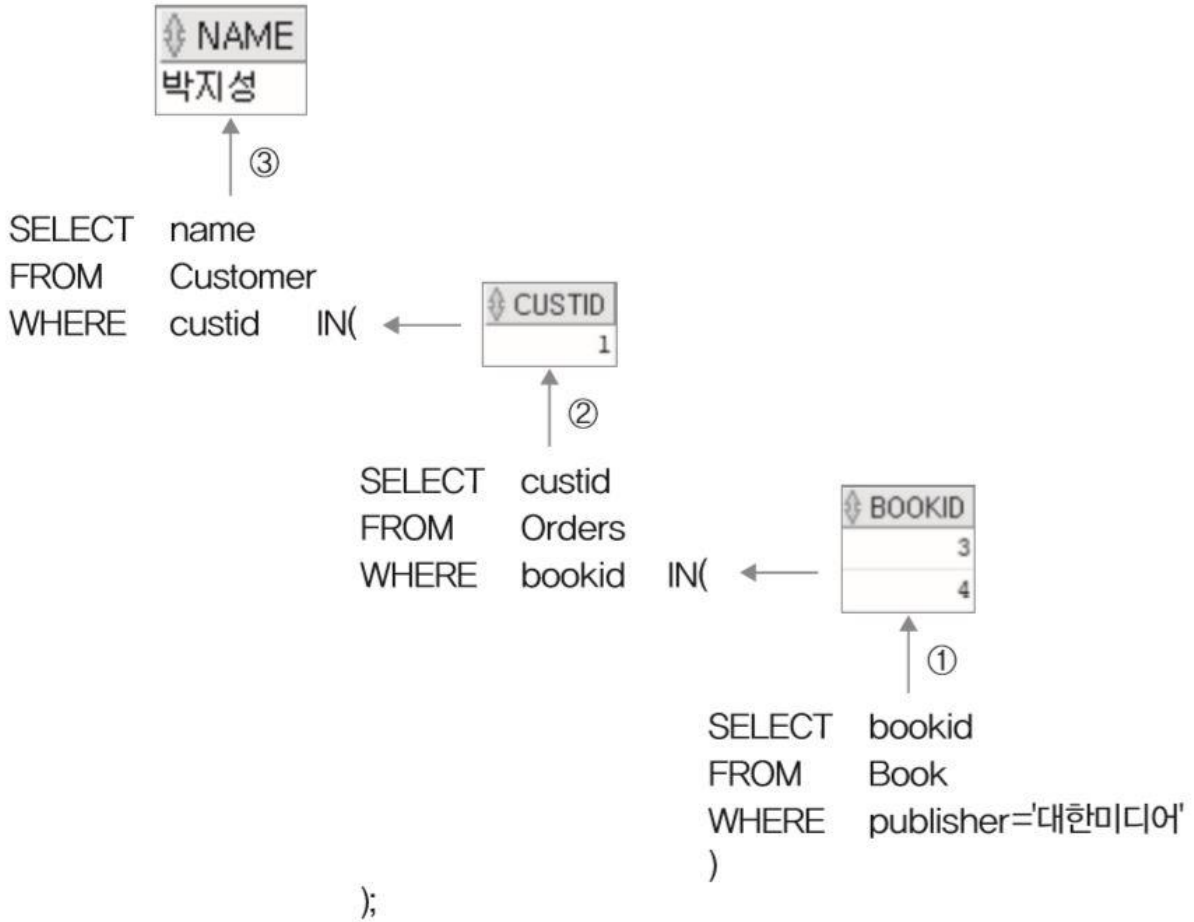
CUSTID
1

BOOKID
3
4

서브 쿼리

예제3

'대한미디어'에서 출판한 도서를 구매한 고객의 이름을 보이시오.



서브 쿼리

'대한미디어'에서 출판한 도서를 구매한 고객의 이름을 보이시오.

①

BOOKID	BOOKNAME	PUBLISHER	PRICE
1	축구의 역사	굿스포츠	7000
2	축구아는 여자	나무수	13000
3	축구의 이해	대한미디어	22000
4	골프 바이블	대한미디어	35000
5	피겨 교본	굿스포츠	8000
6	역도 단계별기술	굿스포츠	6000
7	야구의 추억	이상미디어	20000
8	야구를 부탁해	이상미디어	13000
9	올림픽 이야기	삼성당	7500
10	Olympic Champions	Pearson	13000

③

CUSTID	NAME	ADDRESS	PHONE
1	박지성	영국 맨체스타	000-5000-0001
2	김연아	대한민국 서울	000-6000-0001
3	장미란	대한민국 강원도	000-7000-0001
4	추신수	미국 클리블랜드	000-8000-0001
5	박세리	대한민국 대전	(null)

②

ORDERID	CUSTID	BOOKID	SALEPRICE	ORDERDATE
1	1	1	6000	20/07/01
2	1	3	21000	20/07/03
3	2	5	8000	20/07/03
4	3	6	6000	20/07/04
5	4	7	20000	20/07/05
6	1	2	12000	20/07/07
7	4	8	13000	20/07/07
8	3	10	12000	20/07/08
9	2	10	7000	20/07/09
10	3	8	13000	20/07/10

서브 쿼리

예제4

출판사별로 출판사의 평균 도서 가격보다 비싼 도서를 구하시오.

```
SELECT  b1.bookname
FROM    Book b1
WHERE   b1.price > (SELECT      avg(b2.price)
                    FROM        Book b2
                    WHERE       b2.publisher=b1.publisher)
```

BOOKNAME
골프 바이블
피겨 교본
야구의 추억

서브 쿼리

BOOKID	BOOKNAME	PUBLISHER	PRICE
1	축구의 역사	굿스포츠	7000
2	축구아는 여자	나무수	13000
3	축구의 이해	대한미디어	22000
4	골프 바이블	대한미디어	35000
5	피겨 교본	굿스포츠	8000
6	역도 단계별기술	굿스포츠	6000
7	야구의 추억	이상미디어	20000
8	야구를 부탁해	이상미디어	13000
9	올림픽 이야기	삼성당	7500
10	Olympic Champions	Pearson	13000

서브 쿼리

위치에 따른 서브 쿼리

명칭	위치	영문 및 동의어	설명
중첩 서브쿼리	WHERE 절	Nested subquery Predicate subquery	WHERE 절에 술어가 같이 사용되며 결과를 한정시키기 위해서 사용됨. 상관 혹은 비상관 형태임
스칼라 서브쿼리	SELECT 절	Scalar subquery	SELECT 절에 사용되며 단일 값을 반환하기 때문에 스칼라 서브 쿼리 라고도 함
인라인 뷰	FROM 절	Inline view, Table subquery	FROM 절에서 결과를 뷰(view) 형태로 반환하기 때문에 인라인 뷰라고 함

중첩 서브쿼리

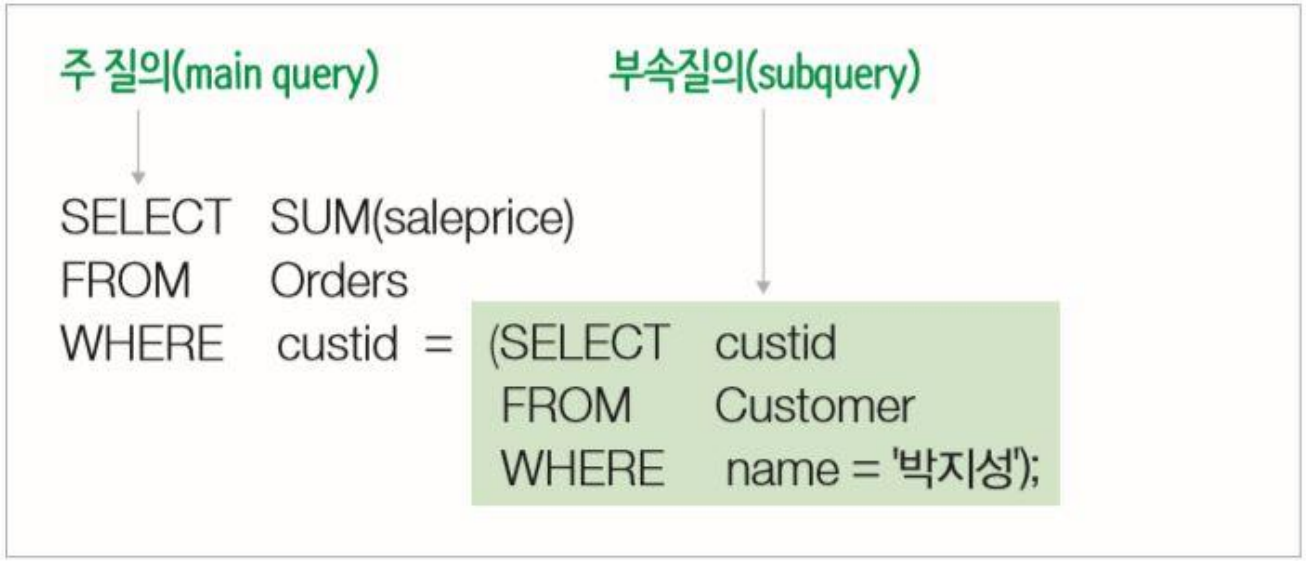
중첩 서브 쿼리 -WHERE 부속질의

- ✓ 중첩 서브 쿼리(nested subquery)는 WHERE 절에서 사용되는 부속질의.
- ✓ WHERE 절은 보통 데이터를 선택하는 조건 혹은 술어(predicate)와 같이 사용됨. 그래서 중첩 서브 쿼리를 술어 서브 쿼리(predicate subquery)라고도 함.

술어	연산자	반환 행	반환 열	상관
비교	=, <, <=, >, >=, <>	단일	단일	가능
집합	IN, NOT IN	다중	다중	가능
한정	ALL, SOME(ANY)	다중	단일	가능
존재	EXISTS, NOT EXISTS	다중	다중	필수

중첩 서브쿼리

- 중첩 서브 쿼리 -WHERE 부속질의



중첩 서브쿼리

- 비교연산자
  - ✓ 서브 쿼리가 반드시 단일 행, 단일 열을 반환해야 하며, 아닐 경우 쿼리를 처리할 수 없음.
  - ✓ 메인 쿼리의 대상 열 속성의 값과 서브 쿼리의 결과값을 비교연산자에 적용하여 참이면 메인 쿼리의 해당 행을 출력함

예제1

평균 주문금액 이하의 주문에 대해서 주문번호와 금액을 보이시오.

```
SELECT  orderid, saleprice
FROM    Orders
WHERE   saleprice <= (SELECT AVG(saleprice) FROM   Orders);
```

ORDERID	SALEPRICE
1	6000
3	8000
4	6000
9	7000

중첩 서브쿼리

예제2

각 고객의 평균 주문금액보다 큰 금액의 주문 내역에 대해서 주문번호, 고객번호, 금액을 보이시오.

```
SELECT  orderid, custid, saleprice
FROM    Orders md
WHERE   saleprice > (SELECT    AVG(saleprice)
                     FROM      Orders so
                     WHERE     md.custid=so.custid);
```

ORDERID	CUSTID	SALEPRICE
2	1	21000
3	2	8000
5	4	20000
8	3	12000
10	3	13000

- AVG() 함수는 집계함수로 단일 행을 반환. 단일 행, 단일 열을 반환하는 비교연산자의 조건을 만족함
- 위 예제는 상관 서브 쿼리 형태로, 메인 쿼리의 각 행 별로 고객번호를 서브 쿼리에 공급하고, 서브 쿼리는 그 값을 가지고 고객별 평균을 구하여 메인 쿼리의 해당 행과 비교연산을 수행한다.

중첩 서브쿼리

- IN, NOT IN
  - ✓ IN 연산자는 메인 쿼리 컬럼이 서브 쿼리에서 제공한 결과 집합에 있는지 확인하는 역할을 함.
  - ✓ IN 연산자는 서브 쿼리의 결과 다중 행, 다중 열을 반환할 수 있음
  - ✓ 메인 쿼리는 WHERE 절에 사용되는 컬럼 값을 서브 쿼리의 결과 집합과 비교해 하나라도 있으면 참이 됨
  - ✓ NOT IN은 이와 반대로 값이 존재하지 않으면 참이 됨.

중첩 서브쿼리

예제3

'대한민국'에 거주하는 고객에게 판매한 도서의 총 판매액을 구하시오.

```
SELECT  SUM(saleprice) "total"
FROM    Orders
WHERE   custid IN (SELECT       custid
                   FROM         Customer
                   WHERE        address LIKE '%대한민국%');
```

	total
1	46000

- Customer 테이블에 '대한민국'이라는 주소를 가진 고객의 수는 3명이다.
- 따라서 단일 행을 반환하는 비교 연산자를 사용할 수 없다.
- IN 연산자는 다중 행을 반환할 수 있으므로 메인 쿼리의 custid를 서브 쿼리의 결과 집합과 비교할 수 있다.
- 구하려는 데이터가 '대한민국에 거주하지 않는 고객 ' 이었을 경우 NOT IN연산자를 사용

중첩 서브쿼리

```
SELECT  SUM(saleprice) "total"
FROM    Orders
WHERE   custid IN (SELECT       custid
                   FROM         Customer
                   WHERE        address LIKE '%대한민국%');
```

CUSTID	NAME	ADDRESS
2	김연아	대한민국 서울
3	장미란	대한민국 강원도
5	박세리	대한민국 대전

ORDERID	CUSTID	BOOKID	SALEPRICE	ORDERDATE
1	1	1	6000	20/07/01
2	1	3	21000	20/07/03
3	2	5	8000	20/07/03
4	3	6	6000	20/07/04
5	4	7	20000	20/07/05
6	1	2	12000	20/07/07
7	4	8	13000	20/07/07
8	3	10	12000	20/07/08
9	2	10	7000	20/07/09
10	3	8	13000	20/07/10

중첩 서브쿼리

ALL, SOME(ANY)

- ✓ ALL, SOME(ANY) 연산자는 비교 연산자와 함께 사용
- ✓ ALL은 모두, SOME(ANY)은 어떠한(최소한 하나라도)이라는 의미를 가짐
- ✓ 비교 연산자를 중심으로 왼쪽에는 스칼라 값이나 열이름이 위치하고 오른쪽에는 서브 쿼리가 위치
- ✓ ALL이나 SOME는 부속질의의 대상 범위를 지정하는 역할을 함

```
scalar_expression { 비교 연산자 ( =, <>, !=, >, >=, !>, <, <=, !< ) }
{ ALL | SOME | ANY } (부속질의)
```

금액 > SOME(SELECT 단가 FROM 상품) → 금액이 상품 테이블에 있는 어떠한(SOME) 단가보다 큰(>) 경우 참이 되어 해당 행의 데이터를 출력

- ALL의 경우 부속질의의 결과 집합 전체를 대상으로 하므로 결과 집합의 최대값(MAX)과 비교하는 것과 같음
- SOME의 경우 부속질의 결과 집합 중 어떠한 값을 의미하므로 최소값과 비교하는 것과 같음

중첩 서브쿼리

예제4

3번 고객이 주문한 도서의 최고 금액보다 더 비싼 도서를 구입한 주문의 주문번호와 금액을 보이시오.

```
SELECT  orderid, saleprice
FROM    Orders
WHERE   saleprice > ALL (SELECT saleprice
                         FROM   Orders
                         WHERE custid='3');
```

	SALEPRICE
1	6000
2	12000
3	13000

	ORDERID	SALEPRICE
1	5	20000
2	2	21000

- 서브 쿼리의 결과로 세 개의 행을 반환하며, saleprice 값은 6000, 12000, 13000이다.
- 메인 쿼리의 '> ALL' 연산자는 결과 집합의 모든 값보다 큰 값을 뜻하므로 가장 큰 13000원보다 큰 금액이 결과로 출력

중첩 서브쿼리

- EXISTS, NOT EXISTS
  - ✓ 데이터의 존재 유무를 확인하는 연산자
  - ✓ 메인 쿼리에서 서브 쿼리로 제공된 속성의 값을 가지고 서브 쿼리의에 조건을 만족하여 값이 존재하면 참이 되고, 메인 쿼리는 해당 행의 데이터를 출력함.
  - ✓ NOT EXISTS의 경우 이와 반대로 동작함.

```
WHERE [NOT] EXISTS (부속질의)
```

  - ✓ EXISTS 연산자는 다른 연산자와 달리 왼쪽에 스칼라 값이나 열을 명시하지 않음
  - ✓ 때문에 반드시 서브 쿼리에 주 질의의 열이름이 제공되어야 함
  - ✓ 서브 쿼리는 필요한 값이(조건을 만족하는 행) 발견되면 참 값을 반환

중첩 서브쿼리

예제5

주문이 있는 고객의 이름과 주소를 보이시오.

```
SELECT  name, address
FROM    Customer cs
WHERE  EXISTS (SELECT    *
                FROM      Orders od
                WHERE    cs.custid=od.custid);
```

NAME	ADDRESS
박지성	영국 맨체스타
김연아	대한민국 서울
장미란	대한민국 강원도
추신수	미국 클리블랜드

CUSTID	NAME	ADDRESS	PHONE
1	박지성	영국 맨체스타	000-5000-0001
2	김연아	대한민국 서울	000-6000-0001
3	장미란	대한민국 강원도	000-7000-0001
4	추신수	미국 클리블랜드	000-8000-0001
5	박세리	대한민국 대전	(null)

① ② ③ ④ ⑤ ×

없음
false

ORDERID	CUSTID	BOOKID	SALEPRICE	ORDERDATE	
1	1	1	6000	20/07/01	true
2	1	3	21000	20/07/03	
3	2	5	8000	20/07/03	true
4	3	6	6000	20/07/04	true
5	4	7	20000	20/07/05	true
6	1	2	12000	20/07/07	
7	4	8	13000	20/07/07	
8	3	10	12000	20/07/08	
9	2	10	7000	20/07/09	
10	3	8	13000	20/07/10	

중첩 서브쿼리

예제6

EXISTS 연산자를 사용하여 '대한민국'에 거주하는 고객에게 판매한 도서의 총 판매액을 구하시오.

```
SELECT  SUM(saleprice) "total"
FROM    Orders od
WHERE   EXISTS (SELECT *
                FROM    Customer cs
                WHERE   address LIKE '%대한민국%' AND cs.custid=od.custid);
```

total
46000

CUSTID	NAME	ADDRESS
2	김연아	대한민국 서울
3	장미란	대한민국 강원도
5	박세리	대한민국 대전

ORDERID	CUSTID	BOOKID	SALEPRICE	ORDERDATE
1	1	1	6000	20/07/01
2	1	3	21000	20/07/03
3	2	5	8000	20/07/03
4	3	6	6000	20/07/04
5	4	7	20000	20/07/05
6	1	2	12000	20/07/07
7	4	8	13000	20/07/07
8	3	10	12000	20/07/08
9	2	10	7000	20/07/09
10	3	8	13000	20/07/10

스칼라 서브쿼리

- 스칼라 서브쿼리(scalar subquery)
  - ✓ SELECT 절에서 사용되는 서브 쿼리로, 서브 쿼리의 결과 값을 단일 행, 단일 열의 스칼라 값으로 반환
  - ✓ 만약 결과값이 다중 행이거나 다중 열이라면 DBMS는 그중 어떠한 행, 어떠한 열을 출력해야 하는지 알 수 없어 에러를 출력, 결과가 없을 경우에는 NULL을 출력
  - ✓ 스칼라 서브 쿼리는 원칙적으로 스칼라 값이 들어갈 수 있는 모든 곳에 사용 가능하며, 일반적으로 SELECT 문과 UPDATE SET 절에 사용됨.
  - ✓ 메인 쿼리와 서브 쿼리와의 관계는 상관/비상관 모두 가능함.

스칼라 서브쿼리

- 스칼라 서브쿼리(scalar subquery)

주 질의 → 스칼라 부속질의 →

```
SELECT custid, (SELECT name
                FROM Customer cs
                WHERE cs.custid=od.custid), SUM(saleprice)
FROM            Orders od
GROUP BY        custid;
```

스칼라 서브쿼리

예제1

서점의 고객별 판매액을 보이시오(고객이름과 고객별 판매액 출력)

```
SELECT      (SELECT  name
             FROM    Customer cs
             WHERE   cs.custid=od.custid) AS name, SUM(saleprice) AS total
FROM        Orders od
GROUP BY    od.custid;
```

NAME	TOTAL
박지성	39000
김연아	15000
장미란	31000
추신수	33000

스칼라 서브쿼리

```
1 SELECT    custid,
            SUM(saleprice)  'total'
  FROM      Orders od
  GROUP  BY custid ;
```

CUSTID	TOTAL
1	39000
2	15000
4	33000
3	31000

```
SELECT    name
FROM      Customer cs
WHERE     cs.custid = od.custid
```

Custid 1의 이름은?

```
2 SELECT    SELECT    name
            FROM      Customer cs
            WHERE     cs.custid = od.custid   name,
            SUM(saleprice) 'total'
  FROM      Orders od
  GROUP BY od.custid ;
```

NAME	TOTAL
박지성	39000
김연아	15000
추신수	33000
장미란	31000

스칼라 서브쿼리

예제2

Orders 테이블에 각 주문에 맞는 도서이름을 입력하시오.

```
ALTER TABLE Orders ADD bookname VARCHAR2(40);

UPDATE Orders
SET    bookname=(SELECT   bookname
                 FROM     Book
                 WHERE    Book.bookid=Orders.bookid);
```

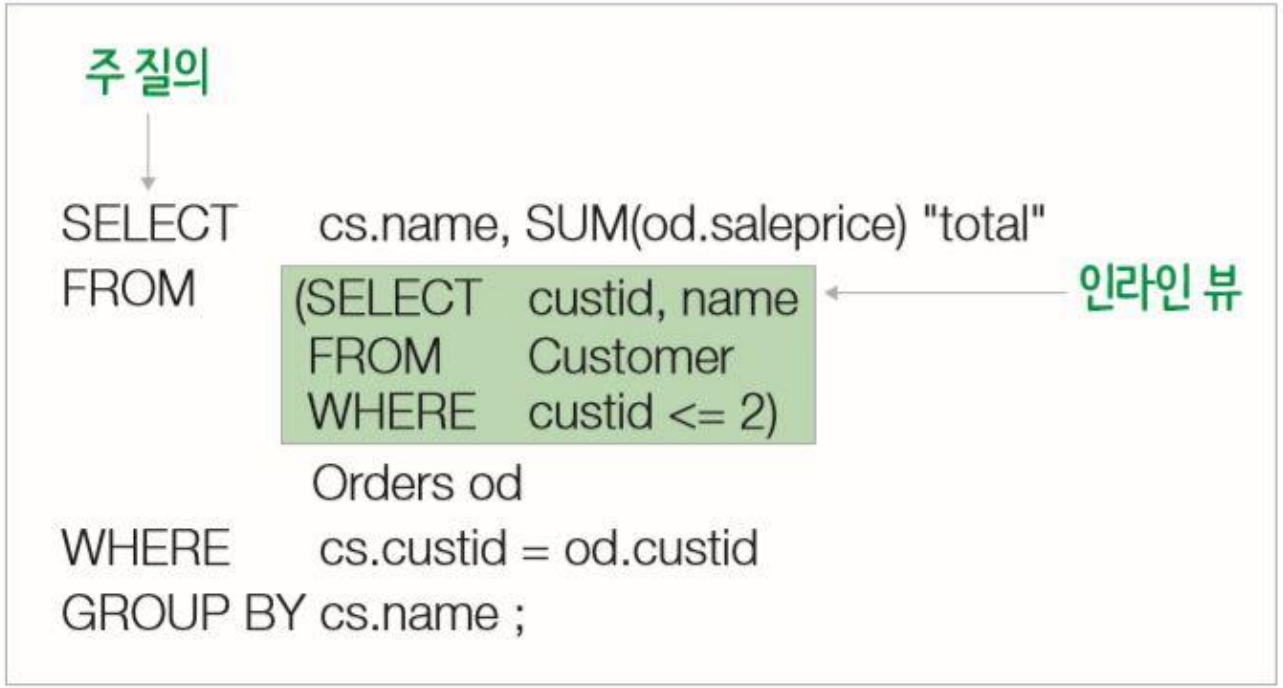
ORDERID	CUSTID	BOOKID	SALEPRICE	ORDERDATE	BOOKNAME
1	1	1	6000	20/07/01	축구의 역사
2	1	3	21000	20/07/03	축구의 이해
3	2	5	8000	20/07/03	피겨 교본
4	3	6	6000	20/07/04	역도 단계별기술
5	4	7	20000	20/07/05	야구의 추억
6	1	2	12000	20/07/07	축구아는 여자
7	4	8	13000	20/07/07	야구를 부탁해
8	3	10	12000	20/07/08	Olympic Champions
9	2	10	7000	20/07/09	Olympic Champions
10	3	8	13000	20/07/10	야구를 부탁해

인라인 뷰

- 인라인 뷰(inline view)란?
  - ✓ FROM 절에서 사용되는 서브 쿼리
  - ✓ 뷰(view)란 기존 테이블로부터 일시적으로 만들어진 가상의 테이블
  - ✓ 테이블 이름 대신 인라인 뷰 서브 쿼리를 사용하면 보통의 테이블과 같은 형태로 사용할 수 있음
  - ✓ 서브 쿼리 결과 반환되는 데이터는 다중 행, 다중 열 이어도 상관 없음
  - ✓ 가상의 테이블인 뷰 형태로 제공되기 때문에 상관 서브 쿼리로 사용될 수는 없음

인라인 뷰

- 인라인 뷰(inline view)란?



인라인 뷰

예제1

고객번호가 2 이하인 고객의 판매액을 보이시오(고객이름과 고객별 판매액 출력).

```
SELECT      cs.name, SUM(od.saleprice) "total"
FROM        (SELECT   custid, name
             FROM    Customer
             WHERE   custid <= 2) cs,  Orders od
WHERE       cs.custid=od.custid
GROUP BY    cs.name;
```

NAME	total
박지성	39000
김연아	15000

- FROM 절에는 두 개의 테이블이 있다
- Customer 테이블에서 고객번호가 2이하인 행만 선택한 cs테이블, Orders 테이블 od이다.
- 먼저 cs테이블을 계산해서 가상의 테이블(뷰)을 만들고 od 테이블과 조인한다.
- 나머지는 일반적인 SQL문의 처리순서와 같다.

연습문제

연습문제

[문제1-1] 온라인 서점 데이터베이스를 구축하려고 한다. Book, Orders, Customer 테이블의 구조를 만들고, bookstore.sql 파일을 실행시켜 데이터를 삽입하시오.

<Book> 테이블

필드명	데이터 타입	속성
bookid	NUMBER(2)	PK
bookname	VARCHAR2(20)	
publisher	VARCHAR2(20)	
price	NUMBER(8)	

<Customer> 테이블

필드명	데이터 타입	속성
custid	NUMBER(2)	PK
name	VARCHAR2(40)	
address	VARCHAR2(50)	
phone	VARCHAR2(20)	

<Orders> 테이블

필드명	데이터 타입	속성
orderid	NUMBER(2)	PK
custid	NUMBER(2)	FK
bookid	NUMBER(2)	FK
saleprice	NUMBER(8)	
orderdate	DATE	

연습문제

[문제1-2] <온라인 서점 데이터베이스> 를이용하여 다음 질문에 대해 SQL 문을 작성하시오.

(1) 모든 도서의 이름과 가격을 검색하시오.
(2) 도서 테이블의 도서번호, 도서이름, 출판사, 가격을 검색하시오.
(3) 도서 테이블에 있는 모든 출판사를 검색하시오.
(4) 가격이 20,000원 미만인 도서를 검색하시오.
(5) 가격이 10,000원 이상 20,000 이하인 도서를 검색하시오.
(6) 출판사가 '굿스포츠' 혹은 '대한미디어' 인 도서를 검색하시오.
(7) '축구의 역사'를 출간한 출판사를 검색하시오.
(8) 도서이름에 '축구' 가 포함된 출판사를 검색하시오.
(9) 도서이름의 왼쪽 두 번째 위치에 '구'라는 문자열을 갖는 도서를 검색하시오.
(10) 고객 테이블에서 핸드폰이 값이 NULL인 고객 검색하시오.
(11) 축구에 관한 도서 중 가격이 20,000원 이상인 도서를 검색하시오.
(12) 출판사가 '굿스포츠' 혹은 '대한미디어' 인 도서를 검색하시오.

연습문제

[문제1-2] <온라인 서점 데이터베이스> 를 이용하여 다음 질문에 대해 SQL 문을 작성하시오.

<정렬, 그룹>

(13) 도서를 이름순으로 검색하시오.

(14) 도서를 가격순으로 검핵하고, 가격이 같으면 이름순으로 검색하시오

(15) 도서를 출판사의 오름차순으로 정렬하시오. 만약 가격이 같다면 가격의 내림차순으로 정렬하시오.

(16) 고객이 주문한 도서의 총 판매액을 구하시오.

(17) 2번 김연아 고객이 주문한 도서의 총 판매액을 구하시오.

(18) 고객이 주문한 도서의 총 판매액, 평균값, 최저가, 최고가를 구하시오.

(19) 도서 판매 건수를 구하시오.

(20) 고객별로 주문한 도서의 총 수량과 총 판매액을 구하시오.

(21) 가격이 8,000원 이상인 도서를 구매한 고객에 대하여 고객별 주문 도서의 총 수량을 구하시오.
단, 두 권 이상 구매한 고객만 구하시오.

연습문제

[문제1-2] <온라인 서점 데이터베이스> 를 이용하여 다음 질문에 대해 SQL 문을 작성하시오.

<조인>

(22) 고객과 고객의 주문에 관한 데이터를 모두 보이시오.

(23) 고객과 고객의 주문에 관한 데이터를 고객별로 정렬하여 보이시오.

(24) 고객의 이름과 고객이 주문한 도서의 판매가격을 검색하시오.

(25) 고객별로 주문한 모든 도서의 총 판매액을 구하고, 고객별로 정렬하시오.

(26) 고객의 이름과 고객이 주문한 도서의 이름을 구하시오.

(27) 가격이 20,000원인 도서를 주문한 고객의 이름과 도서의 이름을 구하시오.

(28) 도서를 구매하지 않은 고객을 포함하여 고객의 이름과 고객이 주문한 도서의 판매가격을 구하시오.

연습문제

[문제1-2] <온라인 서점 데이터베이스> 를 이용하여 다음 질문에 대해 SQL 문을 작성하시오.

<서브 쿼리>

(29) 가장 비싼 도서의 이름을 보이시오.

(30) 도서를 구매한 적이 있는 고객의 이름을 검색하시오.

(31) '대한미디어'에서 출판한 도서를 구매한 고객의 이름을 보이시오

(32) 출판사별로 출판사의 평균 도서 가격보다 비싼 도서를 구하시오.

(33) 평균 주문금액 이하의 주문에 대해서 주문번호와 금액을 보이시오.

(34) 각 고객의 평균 주문금액보다 큰 금액의 주문 내역에 대해서 주문번호, 고객번호, 금액을 보이시오.

(35) '대한민국'에 거주하는 고객에게 판매한 도서의 총 판매액을 구하시오.

(36) 3번 고객이 주문한 도서의 최고 금액보다 더 비싼 도서를 구입한 주문의 주문번호와 금액을 보이시오.

(37) EXISTS 연산자를 사용하여 주문이 있는 고객의 이름과 주소를 보이시오.

(38) EXISTS 연산자를 사용하여 '대한민국'에 거주하는 고객에게 판매한 도서의 총 판매액을 구하시오.

(39) 서점의 고객별 판매액을 보이시오(고객이름과 고객별 판매액 출력)

(40) 고객번호가 2 이하인 고객의 판매액을 보이시오(고객이름과 고객별 판매액 출력).

연습문제

[문제1-3] <온라인 서점 데이터베이스> 서점의 고객이 요구하는 다음 질문에 대해 SQL 문을 작성하시오.

(1) 도서번호가 1인 도서의 이름
(2) 가격이 20,000원 이상인 도서의 이름
(3) 박지성의 총구매액
(4) 박지성이 구매한 도서의 수
(5) 박지성이 구매한 도서의 출판사 수
(6) 박지성이 구매한 도서의 이름, 가격, 정가와 판매가격의 차이
(7) 박지성이 구매하지 않은 도서의 이름

연습문제

[문제1-4] <온라인 서점 데이터베이스> 서점 운영자와 경영자가 요구하는 다음 질문에 대해 SQL 문을 작성하시오.

(1) 서점의 도서의 총수
(2) 서점에 도서를 출고하는 출판사의 총수
(3) 모든 고객의 이름, 주소
(4) 2020년 7월4 ~ 7월7일 사이에 주문받은 도서의 주문번호
(5) 2020년 7월4일 ~ 7월7일 사이에 주문받은 도서를 제외한 도서의 주문번호
(6) 성이 '김 ' 씨인 고객의 이름과 주소
(7) 성이 '김'씨이고 이름이 '아 ' 로 끝나는 고객의 이름과 주소
(8) 주문하지 않은 고객의 이름(서브쿼리)
(9) 주문 금액의 총액과 주문의 평균 금액
(10) 고객의 이름과 고객별 구매액
(11) 고객의 이름과 고객이 구매한 도서목록
(12) 도서의 가격(Book테이블)과 판매가격(Orders 테이블)의 차이가 가장 많은 주문
(13) 도서의 판매액 평균보다 자신의 구매액 평균이 더 높은 고객의 이름

연습문제

[문제1-5] <온라인 서점 데이터베이스> 다음의 심화된 질문에 대한 SQL 문을 작성하시오.

(1) 박지성이 구매한 도서의 출판사와 같은 출판사에서 도서를 구매한 고객의 이름
(2) 두 개 이상의 서로 다른 출판사에서 도서를 구매한 고객의 이름
(3) 전체 고객의 30% 이상이 구매한 도서

연습문제

[문제1-6] <온라인 서점 데이터베이스> 다음 질의에 대해 DDL문과 DML문을 작성하시오.

(1) 새로운 도서('스포츠 세계', '대한미디어', 10000원)이 서점에 입고되었다. 삽입이 안 될 경우 필요한 데이터가 더 있는지 찾아보시오.
(2) '삼성당 ' 에서 출판한 도서를 삭제하시오.
(3) '이상미디어 ' 에서 출판한 도서를 삭제하시오. 삭제가 안되면 원인을 생각해 보시오
(4) 출판사 '대한미디어' 를 ' 대한출판사' 로 이름을 바꾸시오.
(5) (테이블 생성) 출판사에 대한 정보를 저장하는 테이블 Bookcompany(name, address, begin)을 생성하고자 한다. Name은 기본키이며 varchar(20), address는 varchar(20), begin은 date 타입으로 선언하여 생성하시오.
(6) (테이블 수정) Bookcompay 테이블에 인터넷 주소를 저장하는 webaddress 속성을 varchar(30)으로 추가하시오.
(7) Bookcompany 테이블에 임의의 투플 name=한빛아카데미, address=서울시 마포구, begin=1993-01-01, webaddress=http://hanbit.co.kr을 삽입하시오.

연습문제

[문제2] <극장 데이터베이스> 다음은 네 개의 지점을 둔 극장 데이터베이스이다. 밑줄 친 속성은 기본키이다.
테이블의 구조를 만들고 데이터를 입력한 후 다음 질의에 대한 SQL문을 작성하시오.

<극장>

극장번호	극장이름	위치
1	롯데	잠실
2	메가	강남
3	대한	잠실

<상영관>

극장번호	상영관번호	영화제목	가격	좌석수
1	1	어려운 영화	15000	48
2	1	멋진 영화	7500	120
3	2	재밌는 영화	8000	110

<고객>

고객번호	이름	주소
3	홍길동	강남
4	김철수	잠실
9	박영희	강남

<예약>

극장번호	상영관번호	고객번호	좌석번호	날짜
3	2	3	15	2020-09-01
3	1	4	16	2020-09-01
1	1	9	48	2020-09-01

연습문제

[문제3] <극장 데이터베이스>

<1. 단순질의>

(1) 모든 극장의 이름과 위치를 보이시오.

(2) '잠실'에 있는 극장을 보이시오.

(3) '잠실'에 사는 고객의 이름을 오름차순으로 보이시오.

(4) 가격이 8,000원 이하인 영화의 극장번호, 상여관번호, 영화제목을 보이시오.

(5) 극장 위치와 고객의 주소가 같은 고객을 보이시오.

<2. 집계질의>

(1) 극장의 수는 몇 개인가?

(2) 상영되는 영화의 평균 가격은 얼마인가?

(3) 2020년 9월 1일에 영화를 관람한 고객의 수는 얼마인가?

연습문제

[문제3] <극장 데이터베이스>

<3. 부속질의 조인>

(1) '대한' 극장에서 상영된 영화제목을 보이시오.

(2) '대한' 극장에서 영화를 본 고객의 이름을 보이시오.

(3) '대한' 극장의 전체 수입을 보이시오.

<4. 그룹질의>

(1) 극장별 상영관 수를 보이시오.

(2) '잠실'에 있는 극장의 상영관을 보이시오.

(3) 2020년 9월 1일의 극장별 평균 관람 고객 수를 보이시오.

(4) 2020년 9월 1일에 가장 많은 고객이 관람한 영화를 보이시오.

<5. DML>

(1) 각 테이블에 데이터를 삽입하는 INSERT 문을 하나씩 실행시켜 보시오.

(2) 영화의 가격을 10%씩 인상하시오.