파이썬 실습1

- 상품 찾기 프로그램
  - A 매장에 존재하는 상품들 중에서 고객 C가 찾고자 하는 상품들이 있으면 yes, 없으면 no를 상품별로 출력
  - A 매장의 상품 수와 각 상품 번호를 입력
  - 고객 C가 요청하는 상품 수와 각 상품 번호를 입력

```
enter product num(S) >> 5
enter product index(S) >>8 3 7 9 2
enter product num(C) >> 3
enter product index(C) >>5 7 9


result: ['no', 'yes', 'yes']
```

이진탐색(binary search)

- **1**차원 데이터가 정렬되어 있을 때, 특정 데이터를 효율적으로 탐색하는 알고리즘
- 데이터를 미리 정렬하여 최악의 경우에도 **logN** 번의 항목 비교만 하는 효율적 탐색 방법 **(N**은 정렬된 데이터 수**)**
- 삽입 및 삭제가 빈번하게 발생할 시 정렬을 유지하기 위해 최악의 경우 **O(n)**이 소요되는 단점이 있음
- 이진탐색의 알고리즘

```
binary_search(left, right, k):
   if left > right: return None
   middle = (left + right)//2
   if binary[middle] == k: return middle
   if binary[middle] > k:
         binary_search(left, middle-1, k) #중간인덱스 앞부분 탐색
   else:
         binary_seasrch(middle+1, right, k) #중간인덱스 뒷부분 탐색
```

이진탐색(binary search)

- 이진탐색 예제, **binary** 리스트

7	10	20	23	34	36	40	42	48	55	65	77

  - 55을 탐색하면?

```
1단계: binary_search(0,11,55) # left← 0, right← 11, k← 55
 [1] if left > right: False
 [2] middle = (left + right)//2 #middle← 5
 [3] if binary[middle] == 55: False
 [4] if binary[middle] > 55: False
 [5] else:
          binary_search(6, 11, 55)
```

7	10	20	23	34	36	40	42	48	55	65	77

이진탐색(binary search)

2단계: binary_search(6,11,55) # left← 6, right← 11, k← 55
[1] if left > right: False
[2] middle = (left + right)//2 #middle← 8
[3] if binary[middle] == 55: False
[4] if binary[middle] > 55: False
[5] else:
        binary_search(9, 11, 55)

7	10	20	23	34	36	40	42	48	55	65	77

3단계: binary_search(9,11,55) # left← 9, right← 11, k← 55
[1] if left > right: False
[2] middle = (left + right)//2 #middle← 10
[3] if binary[middle] == 55: False
[4] if binary[middle] > 55: True
        binary_search(9,9,55)

7	10	20	23	34	36	40	42	48	55	65	77

이진탐색트리(binary search tree)

- **탐색(search):** 레코드의 집합에서 특정한 코드를 찾아 내는 작업
- 탐색에 관련된 용어
  - 레코드(record): 하나 이상의 필드로 구성된 집합체
  - 테이블(table): 레코드들의 집합
  - 필드(field): 하나의 성질을 가지는 데이터들의 집합

<학생 정보 테이블1>

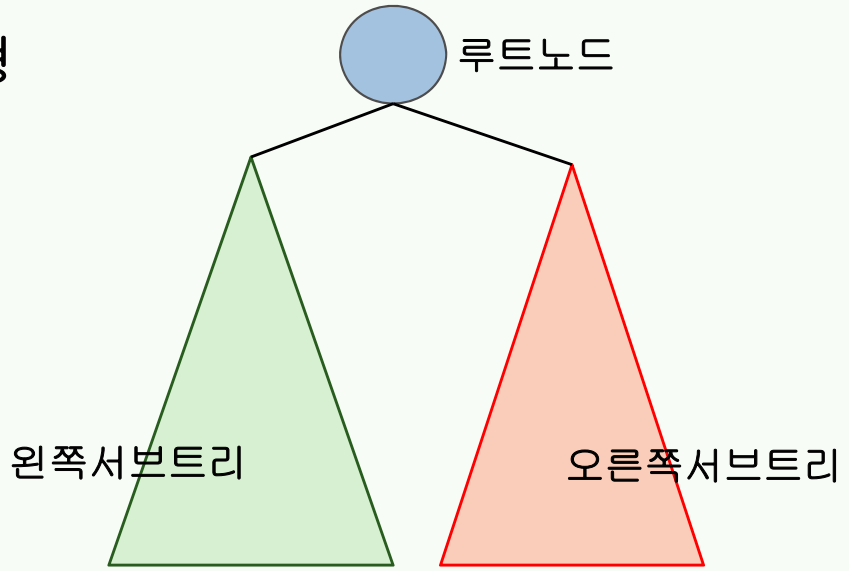
필드

학번	이름	전화번호	주소

레코드

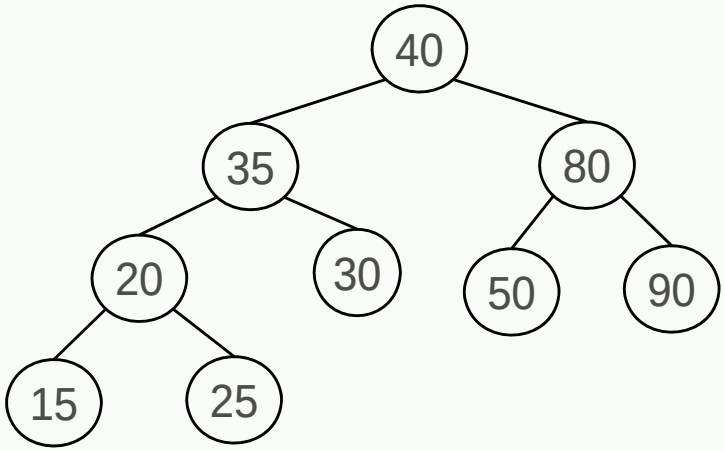
이진탐색트리(binary search tree)

- 이진탐색트리는 이진탐색의 개념을 트리 형태의 구조에 접목한 자료구조
- 이진탐색을 수행하기 위해 단순연결리스트를 변형
- 이진탐색트리의 정의
  - 모든 노드의 키는 유일
  - 왼쪽 서브 트리의 키들은 루트의 키보다 작음
  - 오른쪽 서브 트리의 키들은 루트의 키보다 큼
  - 왼쪽과 오른쪽 서브 트리도 이진 탐색 트리
- 탐색작업을 효율적으로 하기 위한 자료구조
- **Key(왼쪽 서브 트리) ≤ Key(루트 노드) ≤ Key(오른쪽 서브 트리)**
  - 이진탐색을 중위순회 → 오름차순 정렬

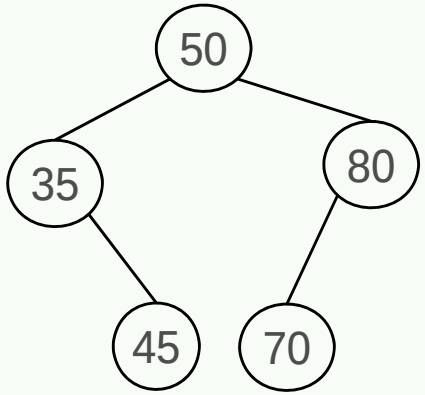


이진탐색트리

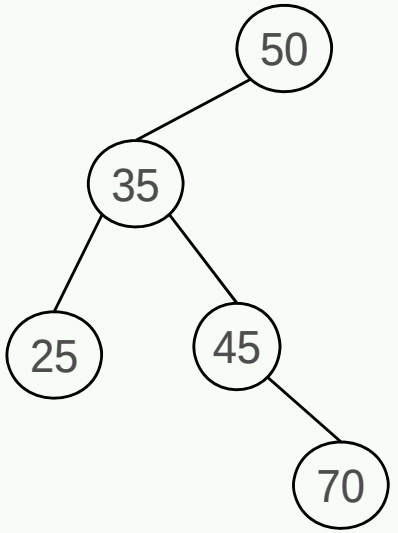
- 다음 트리 중 이진탐색트리는?



(a)



(b)



(c)

이진탐색트리 생성(1)

- 생성 **ADT**

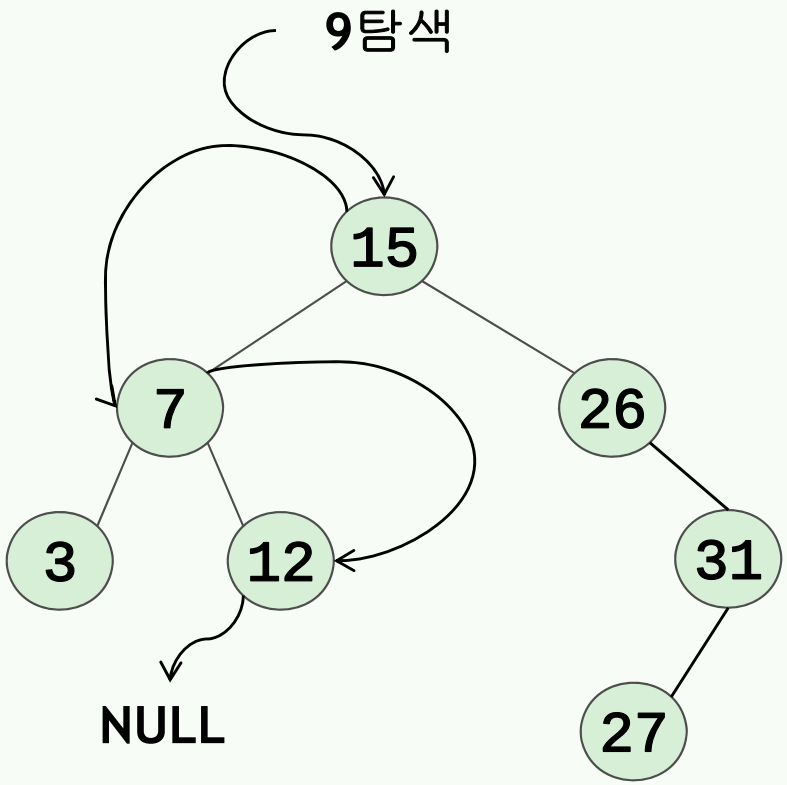
```
// C언어식 표현
struct binarySearchNode{
        int key; void value;
        BSNode * left;
        BSNode * right;
}BSNode;
struct binarySearchRootNode{
        BSNode * root;
}BSRNode;
```

```
#파이썬 표현
class Node:
    def __init__(self,key,value,left=None,right=None):
        self.key = key
        self.value = value
        self.left = left
        self.right = right

class BST:
    def __init__(self):
        self.root = None
```

이진 탐색 트리의 삽입

- 알고리즘

❖ insert_node(T, z)
  ❖ 트리 T에서 z노드 값과 같은 key값의 탐색 시작
  ❖ 탐색이 실패하면 탐색이 끝난 위치에 노드 z를 삽입




```
insert_node(T,z)

p←NULL; //부모노드  포인터
t←root;   //탐색을 위한 포인터
while t≠NULL do
  p←t;
  if z->key < p->key
      then t←p->left;
      else t←p->right;
if p=NULL
  then root←z;// 트리가 비어있음
  else if z->key < p->key
          then p->left←z
          else p->right←z
```

이진탐색트리 생성(2)

- 추가(삽입) 연산자 알고리즘

❖ 기존 노드가 없는 경우는 추가노드가 루트노드가 됨

❖ 기존 노드가 있는 경우

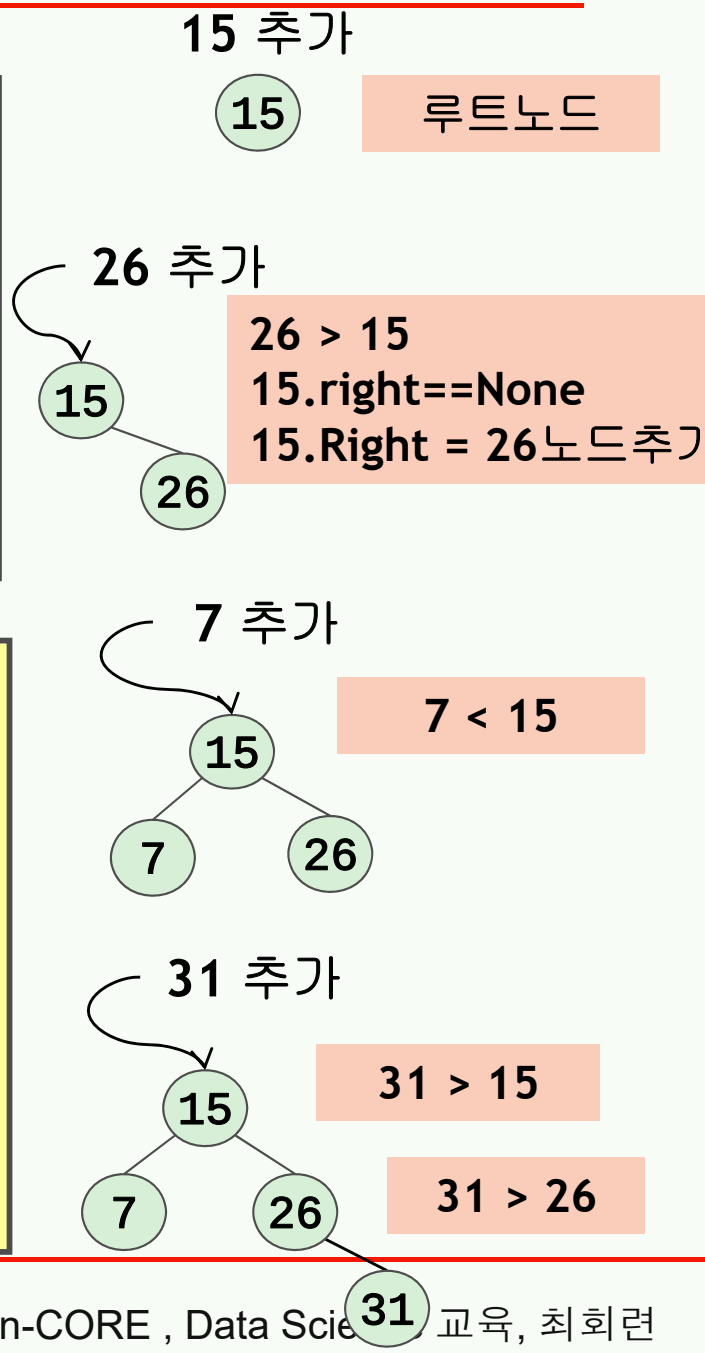
❖ 추가 노드 키 값 < 루트 노드의 키 값

❖ 추가는 루트 노드의 왼쪽 자식을 기준으로 순환

❖ 추가 노드 키 값 > 루트 노드의 키 값

❖ 추가는 루트 노드의 오른쪽 자식을 기준으로 순환

```
put(r, key, value):
   if r == None
            then return r = 추가노드;
   if key < r->key
            then r.left = put(r->left, key,value);
   else if key > r->key
            then r.right = put(r->right, key,value);
   return r
```



이진탐색트리 생성(3), 파이썬 코드

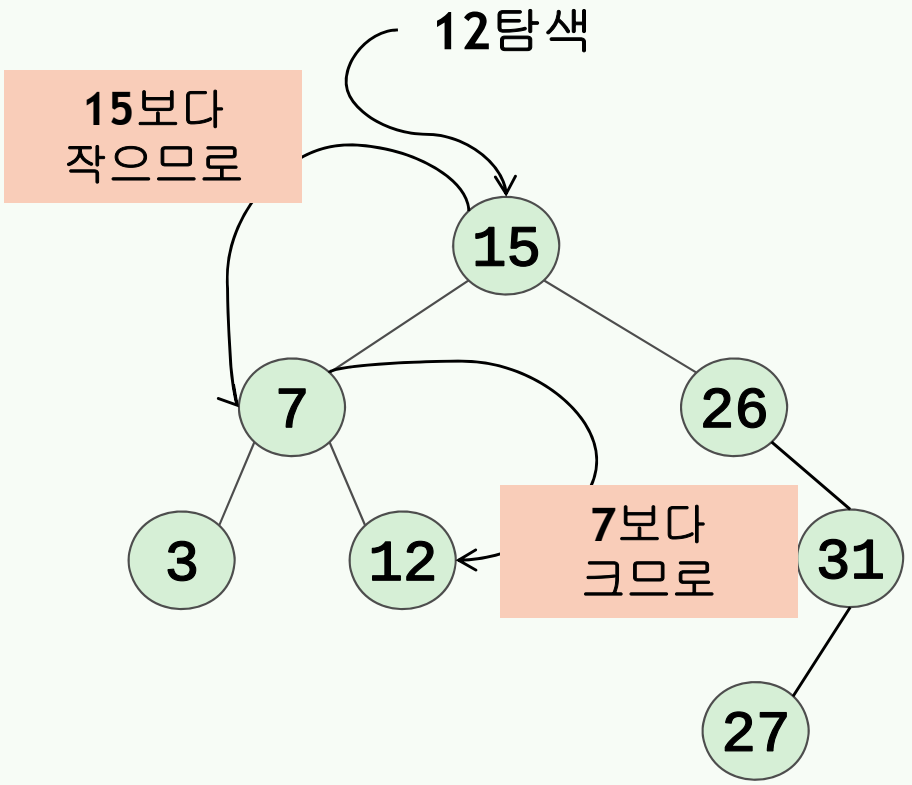
```
#class BST의 메소드
   def put(self, key,value):
       self.root = self.put_value(self.root, key, value)

    def put_value(self,r,key,value):
       if r == None:
           return Node(key,value)
       if r.key > key:
           r.left = self.put_value(r.left,key,value)
       elif r.key < key:
           r.right = self.put_value(r.right,key,value)
       else:
           r.value = value
       return r
```

```
if __name__ == '__main__':
    bs = BST()
    data = [[50,'apple'],[60,'melon'],[20,'lime'],[10,'kiwi'],\
            [40,'peach'],[25,'orange'],[15,'grape'],[80,'lemon'],\
            [70,'cherry'],[5,'pear'],[35,'mango'],[45,'plum']]
    for i in range(len(data)):
        bs.put(data[i][0],data[i][1])
```

이진 탐색 트리의 탐색 알고리즘(1)

- 비교한 결과가 같으면 탐색 성공
- 주어진 키 값 < 루트 노드의 키 값
  - 탐색은 이 루트 노드의 왼쪽 자식을 기준으로 다시 시작
- 주어진 키 값 > 루트 노드의 키 값
  - 탐색은 이 루트 노드의 오른쪽 자식을 기준으로 다시 시작

```
search(x, k):
   if x == None
            then return None;
   if k == x->key
            then return x;
   else if k < x->key
            then return search(x->left, k);
   else return search(x->right, k);
```



이진 탐색 트리의 탐색 알고리즘(2), 파이썬코드

- 순환적 코드(재귀함수 사용)

```
def get(self, key):
        return self.get_value(self.root, k)

def get_value(self, r,k):
        if r == None:
            return None
        if r.key > k:
            return self.get_value(r.left,k)
        elif r.key < k:
            return self.get_value(r.right,k)
        else:
            return r.value
```

이진 탐색 트리의 최솟값 탐색

- 루트부터 왼쪽 자식을 탐색
- **None**을 만나면(노드.**left**의 값이 **None**), **None**의 부모노드가 가진 **key**가 최솟값임

```
#class BTS의 메서드
 def min(self):
     if self.root == None:
         return None
     else:
         return self.minimum(self.root)
 def minimum(self,r):
     if r.left == None:
         return r
     return self.minimum(r.left)
```

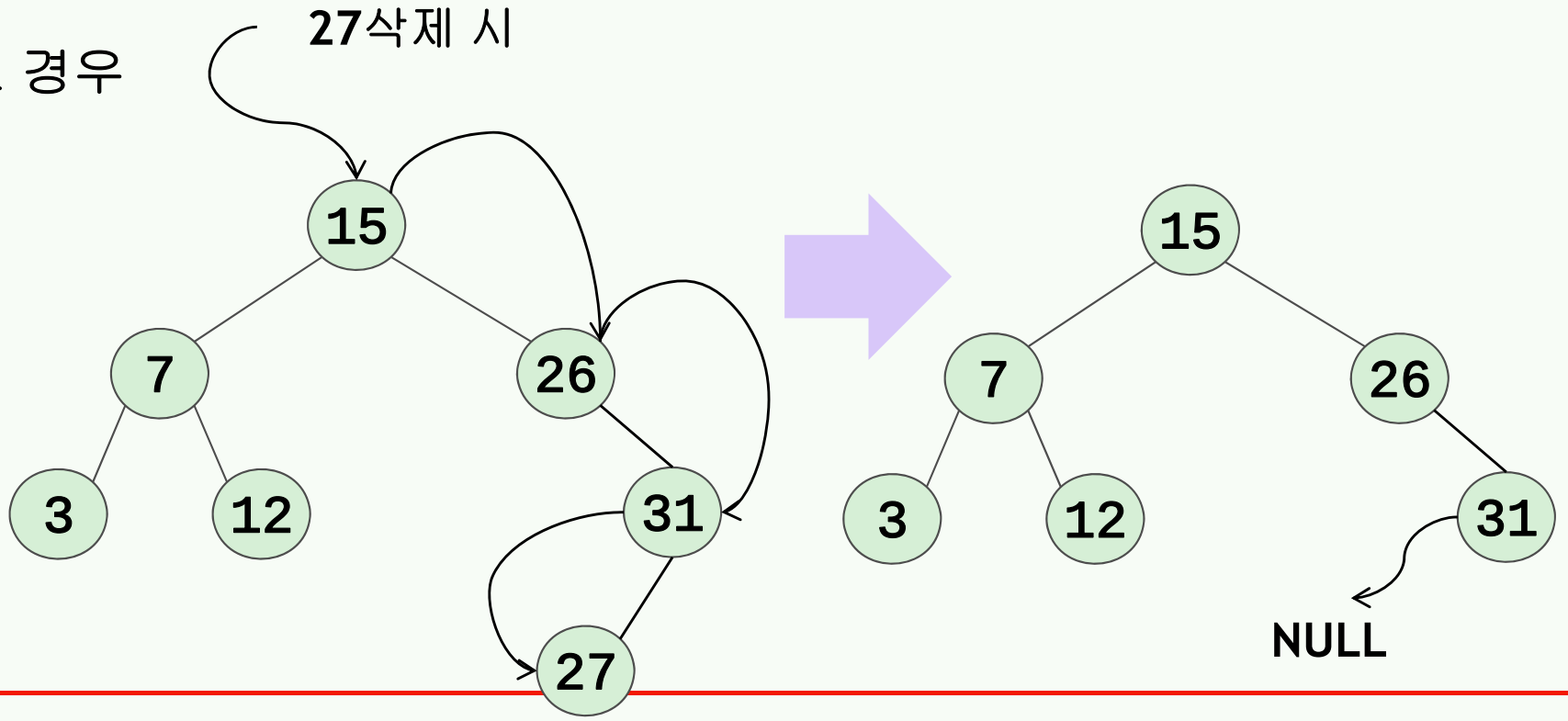
```
#main code
print('\nMinimum of bs')
 min_obj = bs.min()
 print(f'min_key: {min_obj.key}, min_value: {min_obj.value}')
```

이진 탐색 트리의 삭제(1)

- 노드 삭제를 위한 탐색의 3가지 경우 고려: 삭제 노드의 상태 고려

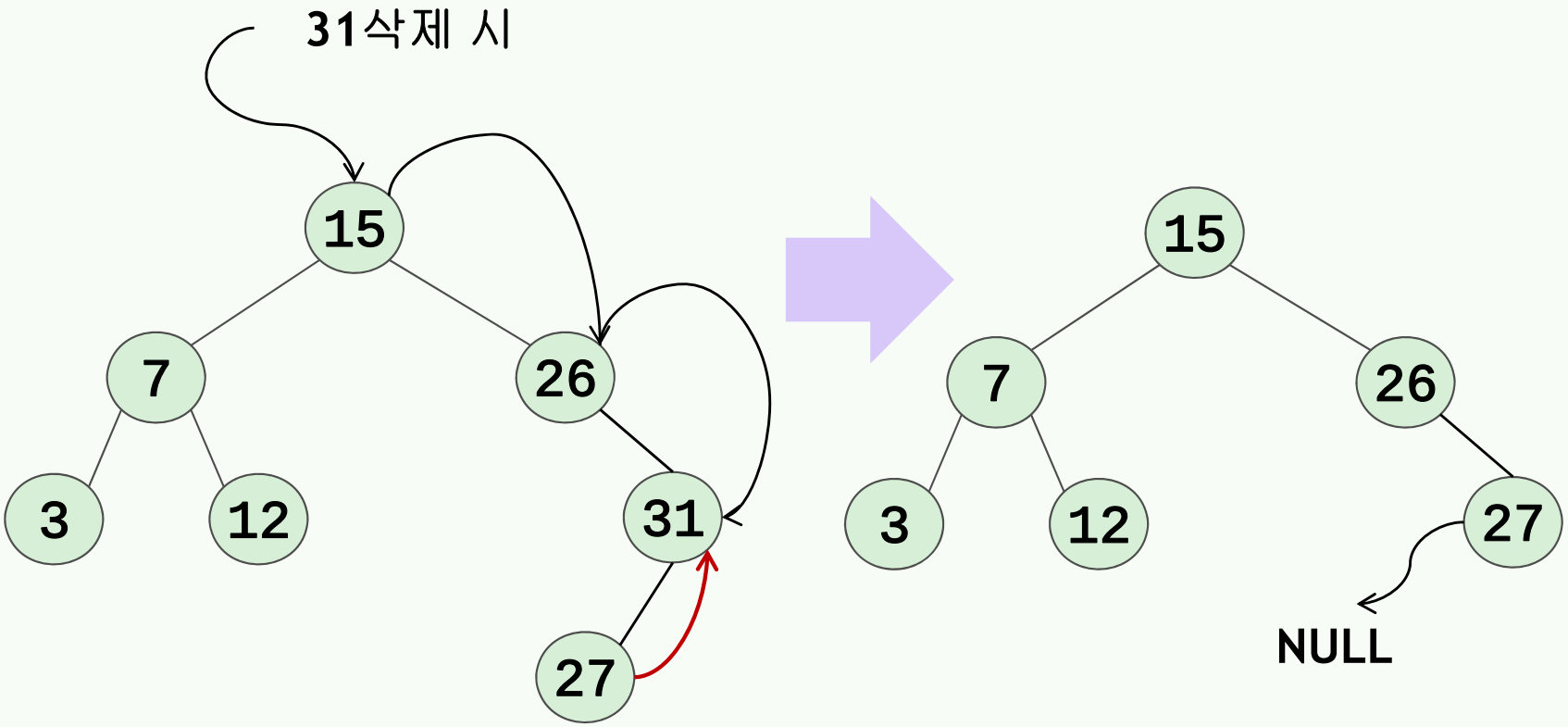
❖ 삭제하려는 노드가 단말 노드인 경우
❖ 삭제하려는 노드가 하나의 왼쪽, 또는 오른쪽 서브 트리 중 하나만 가지고 있는 경우
❖ 삭제하려는 노드가 두 개의 서브 트리를 모두 가지고 있는 경우

- 단말 노드 경우



이진 탐색 트리의 삭제(2)

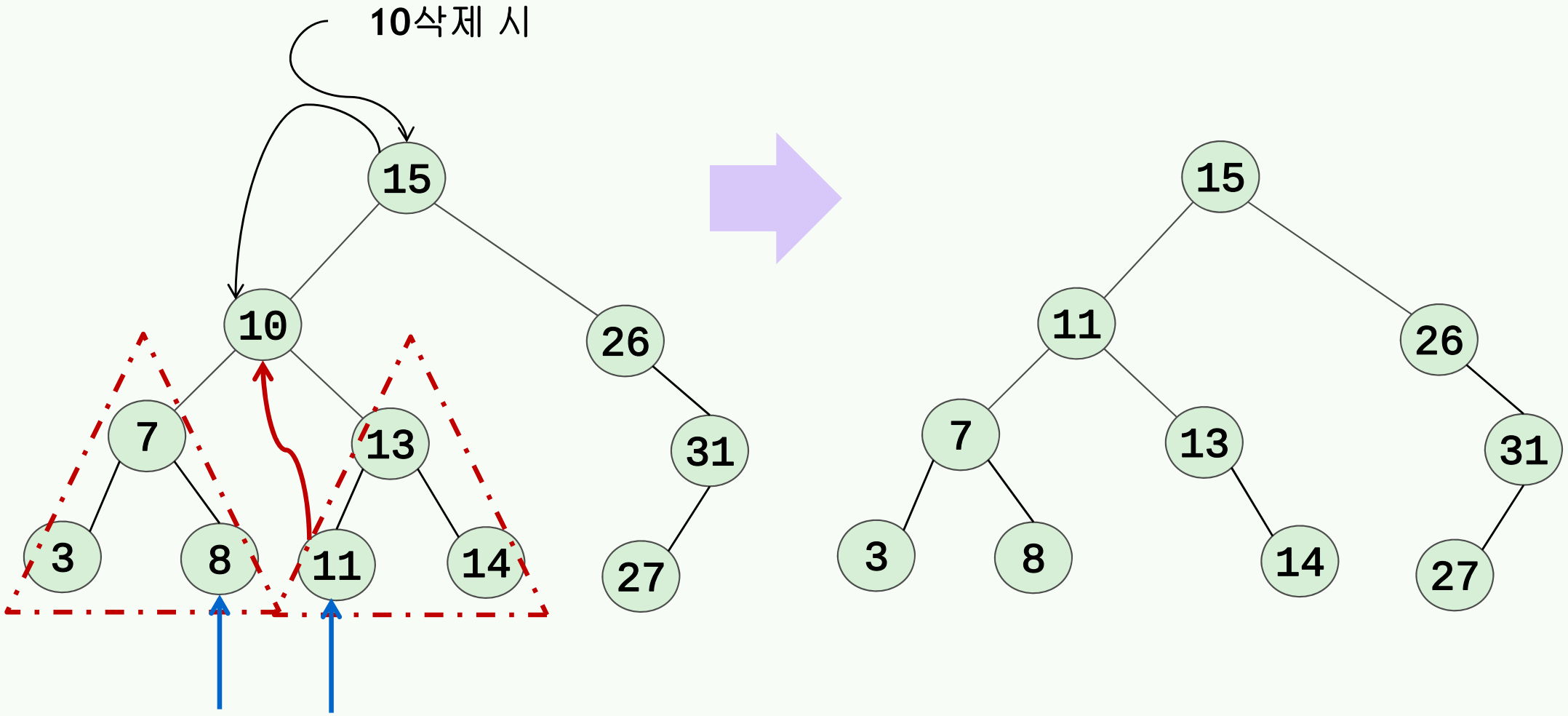
- 서브트리의 노드가 하나만 가지고 있는 경우
  - 삭제하고 자식 노드는 부모 노드에 연결



이진 탐색 트리의 삭제(3)

- 서브 트리가 왼쪽, 오른쪽 두 개 가지고 있는 경우
- 삭제 노드와 가장 비슷한 값을 가진 노드를 서브 트리에서 찾아 삭제 노드의 위치에 연결, 다음의 노드 중에 임의로 하나 선택
  - 삭제 노드의 왼쪽 서브트리의 가장 큰 값인 맨 오른쪽 노드의 값
  - 삭제 노드의 오른쪽 서브트리의 가장 작은 값인 맨 왼쪽 노드의 값

이진 탐색 트리의 삭제(4)



이진 탐색 트리 분석

- 이진 탐색 트리의 탐색, 삽입, 삭제 연산의 시간 복잡도

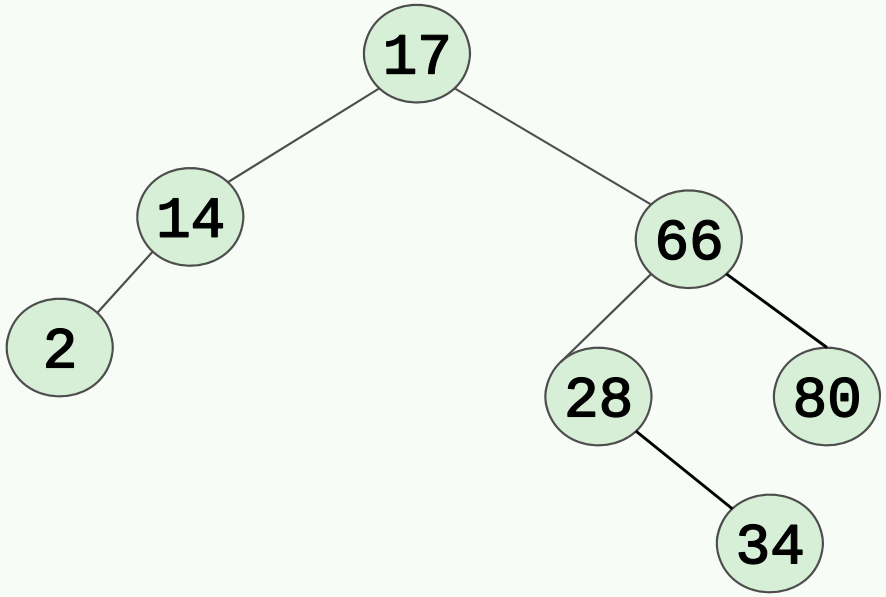
❖ H : 트리의 높이 → O(H)
❖ 따라서 n개의 노드를 가진 이진 탐색 트리의 경우 최소 이진 트리의 높이 → $log_2(n+1)$
❖ 최소 이진 트리의 높이를 가지는 이진 트리 : 균형 잡힌 이진 트리
❖ 이진 탐색 트리 연산의 평균적인 시간 복잡도: O($log_2 n$)

- 시간의 복잡도를 낮추기 위해서는 균형 잡힌 이진 트리의 구성이 중요

https://visualgo.net/ko/bst

퀴즈

- 다음 이진 탐색 트리에서 평균 검색 회수?



- **17** 삭제 시 이진 탐색 트리의 모양은?

파이썬 실습2

- 가래떡을 원하는 길이의 합이 되도록 자르는 프로그램
  - 가래떡을 판매할 때, 주어진 가래떡들을 정해진 높이의 절단기를 통해 절단
  - 절단된 가래떡의 합이 손님이 원하는 가래떡의 길이가 되도록 절단기의 최적 높이를 설정하는 문제임
  - 예를 들어
    - 가래떡의 길이가 19, 14, 10, 17 로 주어진 경우
    - 손님이 4 개의 가래떡을 자른 합의 결과인 6을 원할 때
    - 절단기를 15로 맞춰 자르면 4, 0, 0, 2 가 되어 손님이 원하는 6을 맞출 수 있음
    - 이 경우의 결과값은 15가 됨,
    - 즉 손님이 요청한 총 길이가 M일 때 적어도 M만큼의 가래떡을 얻기 위해서 절단기에 설정할 수 있는 높이의 최댓값을 구하는 문제임
    - 입력값 범위는 떡의 개수 N과 손님이 요청한 떡의 길이M (1<=N<=1000000m 1<=M<=2000000000), 각 떡의 길이 L, 0 <= L <= 1000000000

RDB-mysql

데이터베이스 구축: SQL

2020.09. 04. 금요일

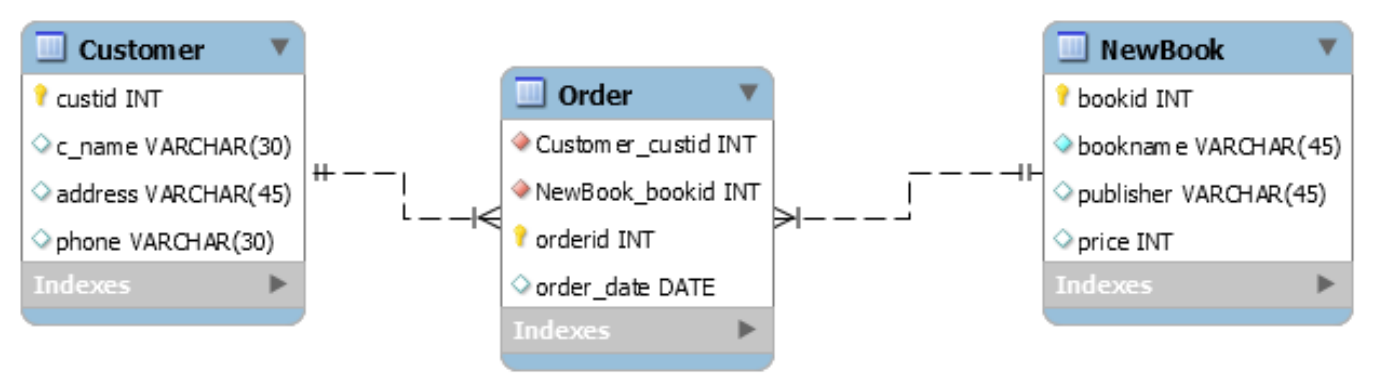
최회련

En-CORE

Data Science Edu.

SQL - 데이터 정의어(DDL)

- 서점정보시스템 구축(도서, 고객, 주문 등)
  - 도서개체유형, 고객개체유형은 다대다의 주문관계로 표현 함
  - 논리 스키마로 변형 시 다대다의 관계는 관계성릴레이션을 생성함



  - 논리 스키마를 데이터베이스로 구축 - 릴레이션을 테이블로 작성

SQL - 데이터 정의어(DDL)

- 서점정보시스템 구축(도서, 고객, 주문 등) 중에서 도서 테이블 생성 예제
  - 테이블 이름: NewBook
  - 테이블 속성 및 데이터 타입: 도서번호(INTEGER), 도서이름(VARCHAR(20)), 출판사(VARCHAR(20)), 가격(INTEGER)

```
CREAT TABLE NewBook
(
        bookid          INTEGER,
        bookname        VARCHAR(20),
        publisher       VARCHAR(20),
        price           INTEGER,
        PRIMARY KEY     (bookid)
);
```

```
CREAT TABLE NewBook
(
    bookid          INTEGER     PRIMARY KEY,
    bookname        VARCHAR(20),
    publisher       VARCHAR(20),
    price           INTEGER,
);
```

  - 만약 primary key 가 단일이 아닌 두 개 속성의 조합이라면?
    - bookid 가 없고, bookname과 publisher가 primary key가 되는 경우?

SQL - 데이터 정의어(DDL)

```
CREAT TABLE NewBook
(
        bookname        VARCHAR(20),
        publisher       VARCHAR(20),
        price           INTEGER,
        PRIMARY KEY     (bookname,publisher)
);
```

- 제약 사항을 추가하는 경우?
  - bookname은 NULL을 가질 수 없고, publisher는 동일한 데이터가 있으면 안됨,
  - price에 값이 입력되지 않는 경우는 기본값으로 10,000을 지정, 또한 최소값은 3,000으로 지정함

```
CREAT TABLE NewBook
(
        bookname        VARCHAR(20)     NOT NULL,
        publisher       VARCHAR(20)     UNIQUE,
        price           INTEGER  DEFAULT 10000 CHECK(price>=3000),
        PRIMARY KEY     (bookname,publisher)
);
```

SQL - 데이터 정의어(DDL)

- 서점정보시스템 구축(도서, 고객, 주문 등) 중에서 고객 테이블 생성 예제

```
CREAT TABLE Customer
(
    custid          INTEGER       PRIMARY KEY,
    c_name          VARCHAR(30),
    address         VARCHAR(45),
    phone           VARCHAR(30),
);
```

- 주문 테이블 생성

```
CREAT TABLE Order
(
    orderid         INTEGER       PRIMARY KEY,
    order_date      DATE,
    custid          INTEGER,
    bookid          INTEGER,
    FOREIGN KEY (custid) REFERENCES Customer(custid) ON DELETE CASCADE
    FOREIGN KEY (bookid) REFERENCES NewBook(bookid) ON DELETE CASCADE
);
```

SQL - 데이터 정의어(DDL)

ALTER 문

- 생성된 테이블의 속성과 속성에 관한 제약 변경
- 기본키 및 외래키 변경
- 문법 형식

```
ALTER TABLE  테이블 이름
            [ADD    속성이름   데이터 타입];
            [DROP  COLUMN   속성이름 ] ;
            [MODIFY COLUMN   속성이름   데이터 타입];
            [MODIFY COLUMN 속성이름  [NULL | NOT NULL]];
            [ADD PRIMARY KEY(속성이름)] ;
            [[ADD | DROP] 제약이름] ;
```

- DROP 은 테이블 삭제에도 사용
  - DROP TABLE NewBook;

SQL - 데이터 정의어(DDL)

```
CREAT TABLE NewBook
(
    bookid  INTEGER      PRIMARY KEY,
    bookname             VARCHAR(20),
    publisher            VARCHAR(20),
    price                INTEGER,
);
```

- NewBook 테이블에 writer, VARCHAR(30)을 추가

```
ALTER TABLE NewBook ADD writer VARCHAR(30);
```

- NewBook 테이블에 price, SMALLINT로 변경

```
ALTER TABLE NewBook MODIFY price SMALLINT;
```

- NewBook 테이블에 writer 삭제

```
ALTER TABLE NewBook DROP COLUMN writer;
```

- NewBook 테이블의 bookid 속성에 NOT NULL 제약조건 추가

```
ALTER TABLE NewBook MLDIFY bookid INTEGER NOT NULL;
```

SQL - 데이터 조작어(DML)

- **SELECT문, 질의어(query)라고도 함**
  - 기본 형식

```
SELECT  [ALL | DISTINCT] 속성 리스트
FROM    테이블 리스트;
```

  - SELECT: 질의 결과 추출되는 속성 리스트를 열거
  - FROM : 질의에 이용되는 테이블 리스트를 열거
  - 예시

```
SELECT  bookname, price
FROM    NewBook;
```

  - bookname을 중복에 상관없이 결과를 모두 표시

```
SELECT  ALL bookname
FROM    NewBook;
```

SQL - 데이터 조작어(DML)

- bookname을 중복없이 표시

```
SELECT  DISTINCT bookname
FROM    NewBook;
```

- price에 대한 부가세를 계산하고, 이 결과를 tax로 출력
  (tax라는 속성이 실제 저장되는 것은 아님)

```
SELECT  bookname, price*0.1 AS “tax”
FROM    NewBook;
```

- 조건 추가 시

```
SELECT  [ALL | DISTINCT] 속성 리스트
FROM    테이블 리스트
[WHERE   조건];
```

SQL - 데이터 조작어(DML)

- 예시, 가격이 25000원 이상의 책 이름에 대한 결과

```
SELECT  bookname, price
FROM    NewBook
WHERE price >= 25000;
```

- 조건에 사용되는 연산자

연산자	설명	연산자	설명
=	같다	AND	모든 조건을 만족하는 경우 검색
<>	다르다	OR	여러 조건 중 한 가지만 만족해도 검색
<	작다	NOT	조건을 만족하지 않는 것만 검색
>	크다	BETWEEN	범위의 검색, price BETWEEN 1000 AND 3000
<=	작거나 같다	LIKE	패턴 연산으로 검색조건이 부분적이고, 문자열을 이용하는 조건에 사용
>=	크거나 같다	IN, NOT IN	집합연산, IN은 집합의 원소인지 판단, 하나라도 포함되면 검색
IS NULL	속성값의 NULL여부	IS NOT NULL	속성값이 NULL아 아닌 여부

SQL - 데이터 조작어(DML)

- 예시, 출판사이름이 “고려“ 이며 가격이 3만원 이상인 책이름

```
SELECT  bookname
FROM    NewBook
WHERE publisher=“고려” AND  price >= 30000;
```

- 가격이 아직 입력되지 않은 책이름과 출판사 검색

```
SELECT  bookname, publisher
FROM    NewBook
WHERE  price IS NULL;
```

- LIKE 연산 검색에 사용되는 기호

기호	설명
%	0개 이상의 문자(문자 내용과 개수는 상관없음)
_	1개의 문자(문자 내용은 상관없음)

SQL - 데이터 조작어(DML)

- LIKE 기호 사용 예

기호 사용 예	설명
LIKE '관계%'	관계로 시작하는 문자열(길이는 상관 없음)
LIKE '%관계'	관계로 끝나는 문자열
LIKE '%관계%'	관계가 포함된 문자열
LIKE '관계_ _ _'	관계로 시작하는 5자 길이의 문자열
LIKE '_ _관%'	세번째 글자가 관인 문자열

- 책이름이 시작이 파로 시작하는 책이름과 출판사 및 가격 출력

```
SELECT  bookname, publisher, price
FROM    NewBook
WHERE  bookname LIKE   ' 파%';
```

- 출판사이름이 7글자인 출파사이름 출력

```
SELECT   publisher
FROM    NewBook
WHERE  publisher LIKE  '_ _ _ _ _ _ _';
```

SQL - 데이터 조작어(DML)

정렬검색, ORDER BY

- 형식

```
SELECT  [ALL | DISTINCT] 속성 리스트
FROM    테이블 리스트
[WHERE   조건]
[ORDER BY 속성 리스트 [ASC | DESC] ];
```

- 예시, 가격의 오름정렬 순으로 출력

```
SELECT  *
FROM    NewBook
ORDER BY price ;
```

- 출판사이름으로 오름정렬하고, 동일한 출판사는 책이름으로 내림정렬한 순으로 출력

```
SELECT  *
FROM    NewBook
ORDER BY publisher ASC,  bookname DESC;
```

SQL - 데이터 조작어(DML)

집계함수(aggregate function) 이용 검색

- 특정 속성값을 통계적으로 계산한 결과를 검색하기 위한 함수
- 열함수라고도 하며, 개수, 합계, 평균, 최댓값, 최솟값의 계산 기능을 제공함
- 집계함수는 NULL 인 속성값은 제외하고 계산
- 집계함수는 WHERE 절에서는 사용할 수 없음, SELECT 또는 HAVING 절에서만 사용
- 자주사용되는 집계함수

집계함수	설명	사용 가능한 속성 타입
COUNT	속성 값의 개수	모든 데이터
MAX	속성 값의 최댓값	
MIN	속성 값의 최솟값	
SUM	속성 값의 합계	숫자 데이터
AVG	속성 값의 평균	

SQL - 데이터 조작어(DML)

- 예제, 책 가격의 평균을 검색

```
SELECT  AVG(price)
FROM    NewBook;
```

- 출력 시 컬럼이름을 average of books 로 하고 싶다면?

```
SELECT  AVG(price) AS “average of books”
FROM    NewBook;
```

- 책이름 컬럼 수를 출력

```
SELECT  COUNT(bookname) AS “num of book”
FROM    NewBook;
```

- 다음 구문의 결과는?

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

SQL - 데이터 조작어(DML)

- 출판사 이름의 중복을 없애고 전체 출판사 개수를 출력

```
SELECT  COUNT(DISTINCT publisher) AS “num of publisher”
FROM    NewBook;
```

그룹별 검색, GROUP BY

- 한 테이블에서 특정 속성의 값이 같은 투플을 모아 그룹을 생성 후
- 그룹별로 검색하기 위해 사용
- 그룹에 대한 조건을 추가하려면 GROUP BY 키워드를 HAVING 키워드와 함께 사용하면 됨
- GROUP BY 키워드가 없는 SELECT 문은 테이블 전체를 하나의 그룹으로 보고 검색하는 것임

SQL - 데이터 조작어(DML)

- 형식

```
SELECT  [ALL | DISTINCT] 속성 리스트
FROM    테이블 리스트
[WHERE   조건]
[GROUP BY 속성 리스트 [HAVING 조건] ]
[ORDER BY 속성 리스트 [ASC | DESC] ];
```

- 예시, 출판사별 책 개수를 출력

```
SELECT  publisher, COUNT(*) AS “num of book”
FROM    NewBook
GROUP BY publisher ;
```

- 출판사별 책 개수와 그 책 중에서 제일 비싼 가격을 검색, 단 책 개수는 num of books 로 가격은 max price로 출력

```
SELECT  publisher, COUNT(*) AS “num of book”, MAX(price) AS “max price”
FROM    NewBook
GROUP BY publisher ;
```

SQL - 데이터 조작어(DML)

- 책을 2권이상 출판한 출판사별로 책의 개수와 가장 비싼 단가를 검색

SQL - 데이터 조작어(DML)

- **GROUP BY와 HAVING 절의 사용 시 주의 사항**
- **GROUP BY**
  - SELECT 절에는 GROUP BY에서 사용한 속성과 집계함수만 사용될 수 있음

```
SELECT  publisher SUM(price)
FROM    NewBook;
GROUP BY publisher;
```

```
SELECT  bookid SUM(price)
FROM    NewBook;
GROUP BY publisher;
```

- **HAVING**
  - WHERE 절과 HAVING 절이 함께 사용 될 시에는 다음과 같은 조건에 맞춰 작성해야 검색조건의 모호성 방지할 수 있음
    - HAVING 절은 반드시 GROUP BY 절과 함께 작성하고 WHERE절보다 뒤에 위치해야 함
    - 또한 <검색조건>에는 집계함수(SUM,AVG,MAX,MIN,COUNT)가 와야 함

```
SELECT  publisher COUNT(*) AS 도서수량
FROM    NewBook;
WHERE price >= 10000
GROUP BY publisher
HAVING COUNT(*) >=2;
```

SQL - 데이터 조작어(DML)

- **GROUP BY 절이 포함된 SQL 문의 실행 순서**
  - SQL문은 실행순서가 없는 비절차적인 언어이지만 내부적 실행순서는 존재함
  - GROUP BY, HAVING, ORDER BY절이 포함된 SQL문의 실행 순서

```
SELECT  publisher COUNT(*) AS 도서수량        (5)
FROM    NewBook;                              (1)
WHERE price >= 10000                          (2)
GROUP BY publisher                            (3)
HAVING COUNT(*) >=2                           (4)
ORDER BY publisher;                           (6)
```

SQL - 데이터 조작어(DML)

- 여러 테이블에 대한 조인(JOIN) 검색
  - 어떤 고객이 주문한 책의 총가격이 알고 싶은 경우에는 Customer, Order 및 NewBook의 정보를 이용해야 함 – 테이블 간의 조인검색이 필요
  - 조인 검색에 필요한 속성을 조인속성이라하고
  - 조인속성의 이름은 달라도 되지만, 도메인은 반드시 동일해야 함
  - 일반적으로 테이블의 관계를 나타내는 외래키를 조인 속성으로 이용 함
  - 조인검색은 조인(join)과 부속질의(subquery)의 두 가지 방법 사용
  - 부속질의
    - 질의가 중첩되어 있다는 의미
    - 중첩질의(nested query)라고도 함
    - 부속질의를 먼저 처리하고 전체질의를 처리하는 순서를 가짐

```
SELECT  bookname
FROM    NewBook;
WHERE price = (select MAX(price) FROM NewBook);
```

SQL - 데이터 조작어(DML)

조인

- 두 테이블에 어떤 조건 부여없이 SELECT 시키면 관계대수의 카티션 프로덕트 연산이 됨

```
SELECT  *
FROM    Order, NewBook;
```

ORDER

orderid	Cusomer_custid	NewBook_bookid	order_date
k001	c01	b001	200610
K002	c03	b004	200615
K003	c06	b005	200617

New Book

bookid	bookname	publisher	price
b001	정보시스템	kk	40000
b002	산공개론	bb	35000
b003	파이썬	gg	20000
b004	데이터분석	kk	25000
b005	딥러닝	gg	35000

- 위의 예의 연산 결과는 3 * 5의 15개의 투플로 구성된 테이블 생성
- 빈 공간은 NULL로 채워짐

SQL - 데이터 조작어(DML)

조건절을 이용한 조인

- 형식

```
SELECT  [ALL | DISTINCT] 속성 리스트
FROM    테이블1,테이블2,....
WHERE <조인조건> AND <검색조건>;
```

```
SELECT  [ALL | DISTINCT] 속성 리스트
FROM    테이블1 INNER JOIN 테이블2 ON <조인 조건>
WHERE  <검색조건>;
```

- 예: 책 id와 책 주문에 관한 데이터를 모두 출력

```
SELECT  *
FROM    Order, NewBook
WHERE  Order.NewBook_bookid = NewBook.bookid;
```

- 책이름별로 정렬하여 출력

```
SELECT  *
FROM    Order, NewBook
WHERE  Order.NewBook_bookid = NewBook.bookid
ORDER BY  NewBook.bookname ;
```

SQL - 데이터 조작어(DML)

- 예: 책 id와 책 주문에 관한 데이터를 모두 출력

```
SELECT  *
FROM    Order O inner join NewBook ON
WHERE   O.NewBook_bookid = ON.bookid;
```

- 책이름별로 정렬하여 출력

SQL - 데이터 조작어(DML)

- 세 개의 테이블을 조인한 결과
  - 예: 고객이름과 고객이 주문한 도서의 이름을 출력

```
SELECT  Customer.c_name, NewBook.bookname
FROM    Customer, Order, NewBook
WHERE  Customer.custid = Order.Customer_custid AND Order.NewBook_bookid = NewBook.bookid
ORDER BY  NewBook.bookname ;
```

- 조인 연산의 특별한 경우 - 외부조인(outer join)
  - 고객 중에서 도서를 주문하지 않은 고객이 존재 하는 경우, 이 고객도 포함하여 고객이름과 도서id를 출력한다면?

```
SELECT  Customer.c_name, NewBook.bookname
FROM    Customer LEFT OUTER JOIN Order ON Customer.custid = Order.Customer_custid
```

SQL - 데이터 조작어(DML)

부속 질의문

- SELECT 내에 또 다른 SELECT문이 포함될 수 있음
- 포함된 또 다른 SELECT문을 부속 질의문 또는 서브 질의문이라 함
- 부족 질의문은 ( )로 묶어 작성하고 ORDER BY 절을 사용할 수 없으며, 상위 질의문보다 먼저 실행 됨
- 상위 질의문과 부속 질의문을 연결하는 연산자가 필요
  - 부속 질의문 분류에 따라 연산자가 다름
- 부속 질의문 분류
  - 하나의 행을 결과로 반환하는 단일 행 부속 질의문 - 일반 비교 연산자를 사용 가능
  - 하나 이상의 행을 결과로 반환하는 다중 행 부족 질의문 - 일반 비교 연산자 사용 불가능

SQL - 데이터 조작어(DML)

- 다중 행 부속 질의문에 사용가능한 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
EXITS	부속 질의문의 결과 값이 하나라도 존재하면 검색 조건이 참
NOT EXITS	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)

- 예: gg 출판사에서 출판한 도서를 구매한 고객의 이름을 검색
  - NewBook에서 g출판사의 id를 구함
  - 이 번호를 이용하여 Order 테이블에서 고객번호를 구함
  - Customer에서 고객번호를 이용하여 고객 이름을 구함

SQL - 데이터 조작어(DML)

```
SELECT  c_name
FROM    Customer
WHERE  custid IN ( SELECT custid
                   FROM Order
                   WHERE NewBook_bookid IN ( SELECT bookid
                                             FROM NewBook
                                             WHERE publisher = 'gg'));
```

```
SELECT  c_name
FROM    Customer, Order, NewBook ON Customer.custid = Order.Customer_custid AND
                                    Order.NewBook_bookid = NewBook.bookid
WHERE  NewBook.publisher = 'gg'));
```

SQL - 데이터 조작어(DML)

- 기타 부속 질의어에 사용되는 구문
- **EXISTS**
  - 상관 부속질의문 형식
  - 원래 단어에서 의미하는 것과 같이 조건에 맞는 투플이 존재하면 결과에 포함
  - 즉, 부속질의문의 어떤 행이 조건에 만족하면 참 임
  - NOT EXISTS는 부속질의문의 모든 행이 조건에 만족하지 않을 때만 참 임
  - 예: 주문이 있는 고객의 이름과 주소를 검색
    - 이 경우는 EXISTS 를 사용하면 Customer 와 Order의 모든 내용을 다 검색할 필요없이 Customer의 첫 행의 정보와 일치하는 Order의 행이 발견 시 True 가 되고 Customer의 두 번째 행의 정보 검색으로 진행됨

```
SELECT  c_name, address
FROM    Customer
WHERE  EXISTS( SELECT *
                        FROM Order
                        WHERE Customer.custid = Order.Customer_custid);
```

SQL - 데이터 조작어(DML)

- 집합 연산
  - MySQL에서는 합집합 명령어인 UNION 만 있음
  - 차집합과 교집합은 NOT IN 과 IN 연산자를 이용하여 구현 함

```
SELECT  [ALL | DISTINCT] 속성 리스트
FROM    테이블
<WHERE  검색조건>
UNION
SELECT  [ALL | DISTINCT] 속성 리스트
FROM    테이블
<WHERE  검색조건>
```

SQL - MySQL

- **MySQL에서 사용하는 내장 함수(built-in function)**
  - SQL 내장함수는 상수나 속성 이름을 입력 값으로 받아 단일 값을 결과로 반환
  - 모든 내장함수는 최초에 선언될 때 유효한 입력 값을 받아야 함
  - SQL 내장함수는 SELECT, WHERE, UPDATE 절 등에서 모두 사용 가능

```
SELECT  ....  함수명(인자1,인자2,...)
FROM    테이블
WHERE   ....  열이름=함수명(인자1,인자2,...);

UPDATE 테이블
SET       .... 여이름 = 함수명(인자1,인자2,...);
```

SQL - MySQL

- **MySQL**에서 제공하는 주요 내장함수

dev.mysql.com/doc/refman/8.0/en/functions.html

구분		함수 이름
단일행 함수	숫자함수	ABS,CEIL,COS,EXP,FLOOR,LN,LOG,MOD,POWER,RAND,ROUND,SIGN,TRUNCATE
	문자함수 (문자 반환)	CONCAT,LEFT,RIGHT,LOWER,UPPER,LPAD,RPAD,LTRIM,RTRIM, REPLACE,REVERSE,RIGHT,SUBSTR,TRIM
	문자함수 (숫자 반환)	ASCII,INSTR,LENGTH
	날짜/시간함수	ADDDATE,CURRENT_DATE,DATE,DATEDIFF,DAYNAME,LAST_DAY, SYSDATE,TIME
	변환함수	CAST,CONVERT,DATE_FORMAT,STR_TO_DATE
	정보함수	DATABASE,SCHEMA,ROW_COUNR,USER,VERSION
	NULL 관련 함수	COALESCE,ISNULL,IFNULL,NULLIF
집계함수		AVG,COUNT,MAX,MIN,STD,STDDEV,SUM
윈도우함수(또는 분석함수)		CUME_DIST,DENSE_RANK,FIRST_VALUE,LAST_VALUE,LEAD,NTILE,RANK, ROW_NUMBER

SQL - MySQL

- 시간/날짜 함수
  - MySQL은 WHERE 절 없이 SELECT 만 사용 가능

SQL 문	실행 결과
SELECT now();	2020-06-17 10:16:20
SELECT current_date();	2020-06-17
SELECT now()+0;	20200617101620
SELECT current_date() + 0;	20200617
SELECT DATE_FORMAT(sysdate(),'%y%m%d:%H/%i/%s')	200617:10/16/20
SELECT DATEDIFF(date1,date2) ex)SELECT DATEDIFF('2020-06-19','2020-06-16')	2 (날짜의 차이 반환)
SELECT ADDDATE(sysdate(), INTERVAL 3 day)	2020-06-20 10:16:20
SELECT SEC_TO_TIME(3000)	00:50:00
SELECT DAYOFNAME(now())	Wednesday
SELECT DAYOFWEEK(now())	수요일:3

시간/날짜함수

Format의 주요 지정자

인자	설명
%w / %W	요일 순서(0~6,Sunday=0) / 요일(Sunday ~ Saturday)
%a	요일의 약자(Sun ~ Sat)
%d	1달 중 날짜(00 ~ 31)
%j	1년 중 날짜(001 ~ 366)
%h / %H	12시간(01 ~ 12) / 24시간(00 ~ 23)
%i	분(0 ~ 59)
%m / %M	월 순서(01~12, January=0) / 월 이름(January~ December)
%b	월 이름 약어(Jan ~ Dec)
%s	초(0 ~ 59)
%y / %Y	4자리 연도의 마지막 2자리 / 4자리 연도

뷰(View)

- 하나 이상의 테이블을 합하여 만든 가상의 테이블
- 합한다는 의미는 **SELECT** 문을 통한 최종 결과를 의미
- 물리적 존재인 기본 테이블과는 달리 논리적으로만 존재함
- 질의문을 통한 결과를 하나의 가상 테이블로 정의하여 실제 테이블처럼 사용할 수 있도록 만든 데이터베이스 개체
  - 뷰 생성 시 기반이 되는 물리적인 테이블을 기본 테이블이라 함
  - 일반적으로 기본 테이블을 기반으로 만들어지지만, 다른 뷰를 기반으로도 생성 가능 함
  - 뷰를 통해 기본 테이블의 내용을 쉽게 검색할 수 있지만, 기본 테이블의 내용 변경 작업은 제한적으로 이루어짐

뷰(View)

- 뷰 생성

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

  - 등급이 vip인 우수고객의 정보를 출력하는 VIEW 생성

```
CREATE VIEW VIP_Customer(custid,name,age,address)
AS SELECT custid, name, age, address
    FROM   Customer
    Where  grade = 'vip'

WITH CHECK OPTION;
```

- 뷰 삭제
  - DROP VIEW 뷰_이름

뷰(View)

뷰 활용의 장점

- 질의문을 좀 더 쉽게 작성할 수 있음
  - 특정 조건을 만족하는 투플들로 뷰를 미리 만들어 놓으면 사용자가 WHERE 절 없이 뷰를 검색해도 특정 조건을 만족하는 데이터 검색이 가능
  - GROUP BY, 집계함수, 조인 등을 이용해 뷰를 생성하면 SELECT와 FROM 절만으로 원하는 결과를 쉽게 얻을 수 있음
- 데이터의 보안 유지에 도움
  - 여러 사용자의 요구에 맞는 다양한 뷰를 미리 정의해두고 사용자가 자신에게 제공된 뷰를 통해서만 데이터에 접근하도록 권한 설정을 하면 보안 유지에 도움이 됨
- 데이터를 좀 더 편리하게 관리할 수 있음