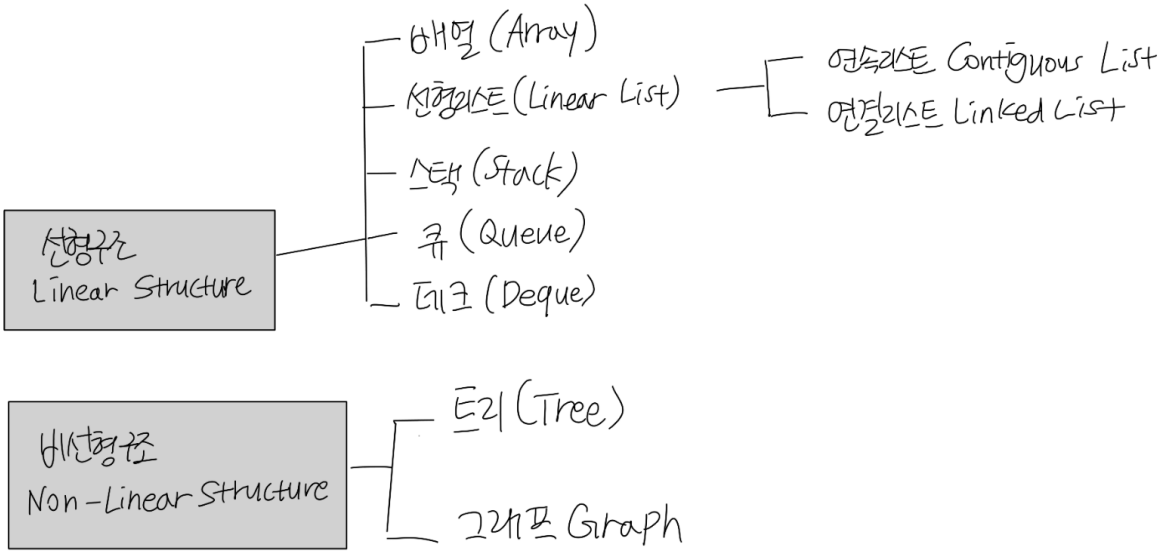
036 자료구조

1. 자료구조의 정의

- 효율적인 프로그램 작성 시 가장 우선적 고려사항은 저장공간의 효율성, 실행시간의 신속성.
- 자료구조는 프로그램에서 사용하기 위한 자료를 기억장치의 공간 내에 저장하는 방법과, 저장된 그룹 내에 존재하는 자료간의 관계, 처리방법 등을 연구분석하는 것.
    - 자료의 표현과 그것과 관련된 연산
    - 일련의 자료들을 조직화하고 구조화
    - 어떠한 자료 구조에서도 필요한 모든 연산들 처리 가능
    - 자료 구조에 따라 프로그램 실행시간 달라짐

2. 자료구조의 분류

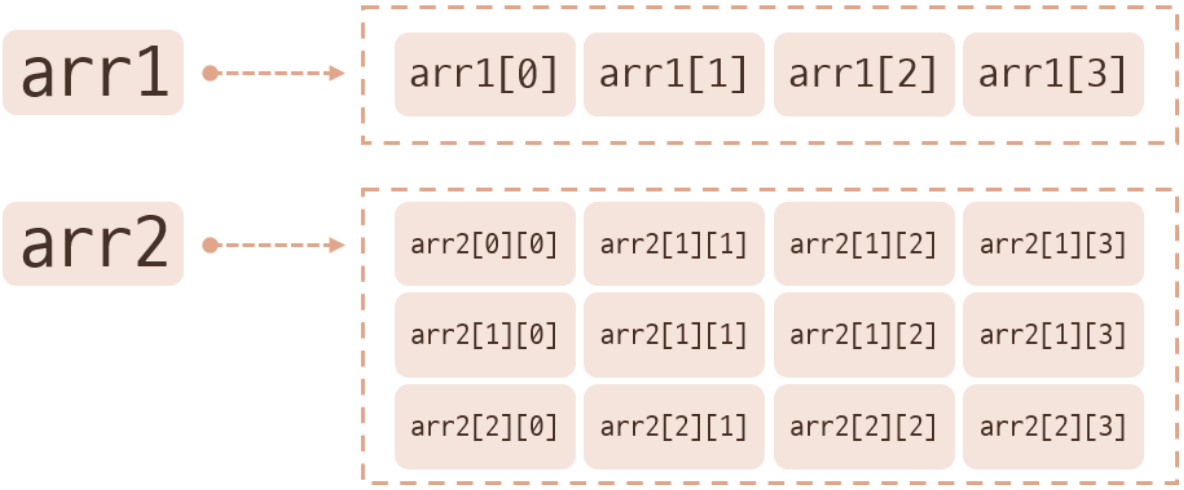
[사진1. 자료구조 분류 사진]



3. 배열

- 배열은 동일한 자료형의 데이터들이 같은 크기로 나열되어 순서를 갖고 있는 집합
    - 배열은 정적인 자료구조로 기억장소 추가 어렵고, 데이터 삭제 시 데이터가 저장되어 있던 기억장소는 빈공간으로 남아있어 메모리 낭비 발생.
    - “ 첨자 **[ ]** ”를 이용하여 데이터에 접근
    - 반복적인 데이터 처리 작업에 적합한 구조
    - 데이터마다 동일한 이름의 변수를 사용해 처리가 간편
    - 첨자 개수에 따라 **n**차원 배열이라고 부름(num[ ][ ] : 2차원)

- 1차원 배열과 2차원 배열



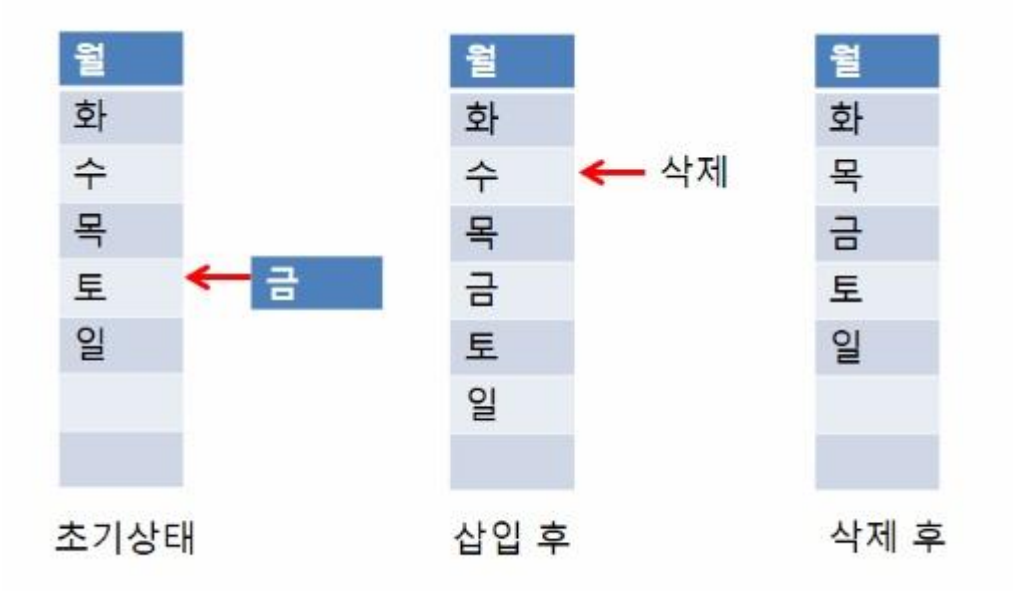
4. 선형리스트(Linear List)

- 일정한 순서에 의해 나열된 자료구조
    - 배열을 이용하는 연속리스트**(Contiguous List)**, 포인터를 이용하는 연결리스트**(Linked List)**로 구분
        - 포인터 **Pointer** : 현재 위치에서 다음 노드의 위치를 알려주는 요소

- 프런트 포인터**(F, Front Pointer)** : 리스트를 구성하는 최초의 노드 위치를 가리키는 요소
- 널 포인터**(Null Pointer, Nil Pointer)** : 다음 노드가 없음을 나타내는 포인터(아무것도 가리키지 않는 상태), 일반적으로 마지막 노드의 링크 부분에 0, ^, \0 등의 기호 표시

○ 연속리스트(Contiguous List)

- 배열과 같이 연속되는 기억장소에 저장되는 자료구조
- 기억장소를 연속적으로 배정받기 때문에 기억장소 이용효율은 밀도가 **1** 로서 가장 좋다.
  - 밀도가 **1** : 일정한 면적에 무엇이 빽빽히 들어있는 정도. 배정된 기억장소를 빈 공간 없이 꽉 차게 사용
- 연속리스트는 중간에 데이터를 삽입하기 위해서 연속된 빈 공간이 있어야 하며, 삽입/삭제 시 자료 이동 필요

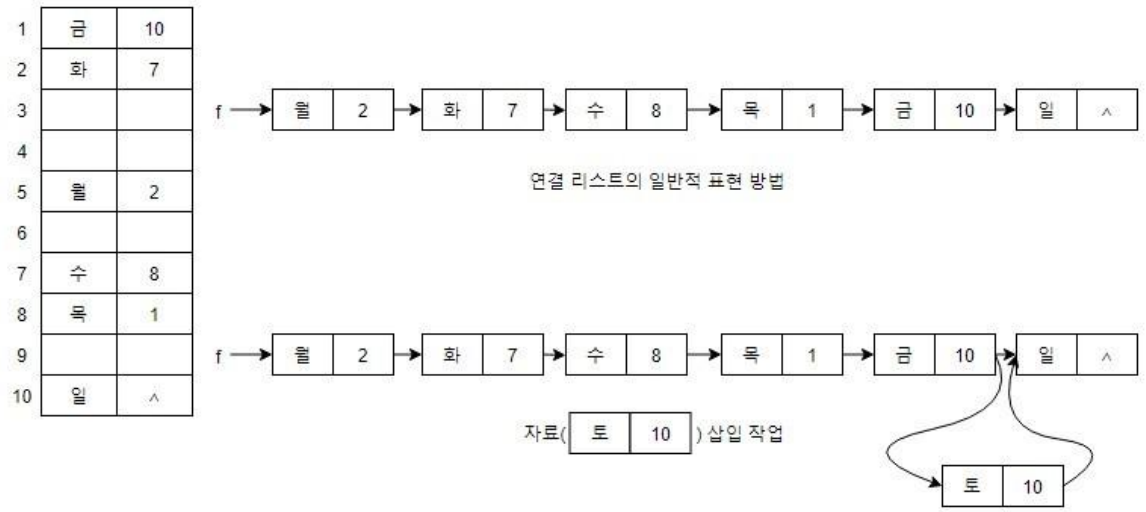


○ 연결리스트(Linked List)

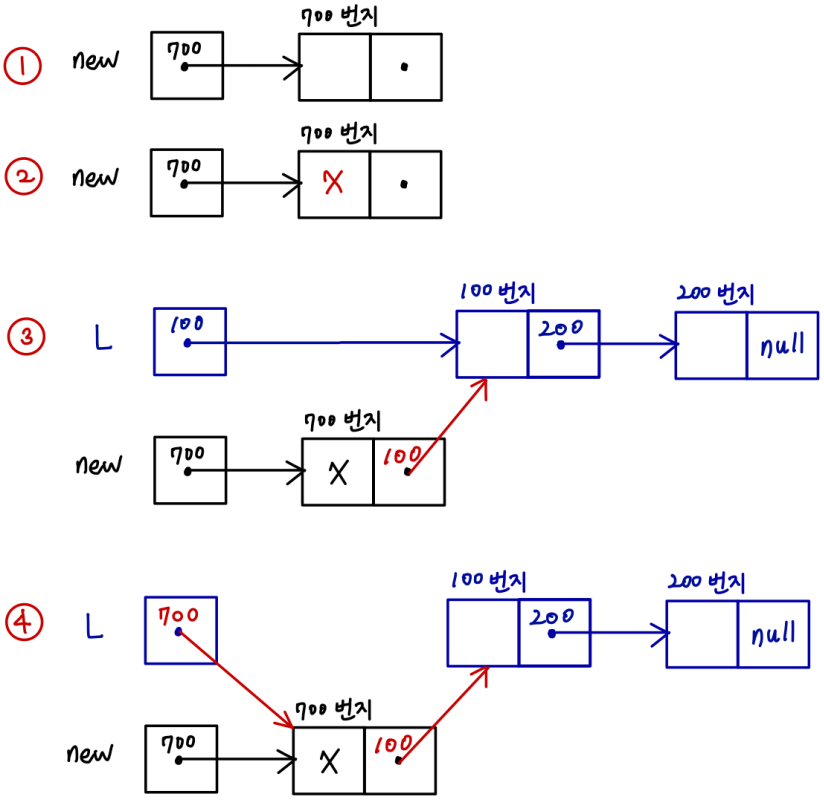
- 자료들을 반드시 연속적으로 배열시키지 않고 임의의 기억공간에 기억시키되, 자료 항목의 순서에 따라 노드의 포인터 부분을 이용하여 서로 연결시킨 자료구조
  - 노드**(Node)** : 자료를 저장하는 데이터부분과 다음 노드를 가리키는 포인터인 링크 부분으로 구성된 기억 공간

- 연결리스트는 노드의 삽입 / 삭제 용이
- 기억공간이 연속적으로 놓여있지 않아도 저장할 수 있음
- 연결을 위한 링크(포인터) 부분이 필요하기 때문에
  순차리스트에 비해 공간의 이용효율이 좋지 않다.
- 연결을 위한 포인터를 찾는 시간 필요하기 때문에 접근 속도가 느림
- 중간 노드 연결이 끊어지면 그 다음 노드를 찾기 힘듦

[사진3. 연결리스트(예시1), 연결리스트 기억장치 내에서의 표현 방법]



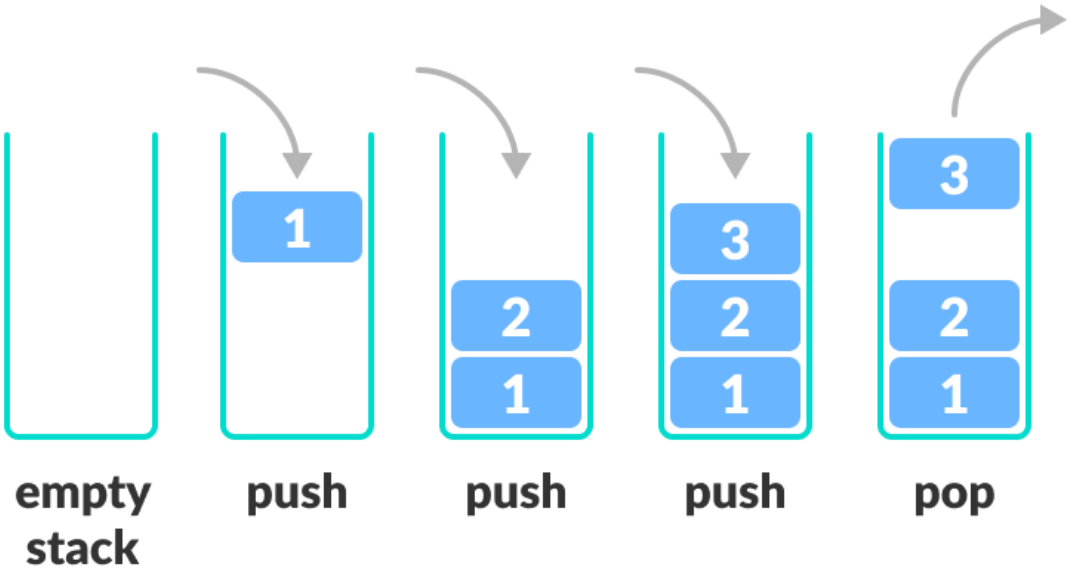
[사진4. 연결리스트(예시2)]



5. 스택(Stack)

리스트의 한쪽 끝으로만 자료의 삽입, 삭제 작업이 이루어지는 자료구조.

- 가장 나중에 삽입된 자료가 가장 먼저 삭제되는 후입선출**(LIFO; Last In First Out)** 방식으로 자료 처리
- 스택의 모든 기억공간이 꽉 채워져 있는 상태에서 데이터가 삽입되면 **"오버플로우(Overflow)"** 발생, 더 이상 삭제할 데이터가 없는 상태에서 데이터 삭제 시 **"언더플로우(Underflow)"** 발생



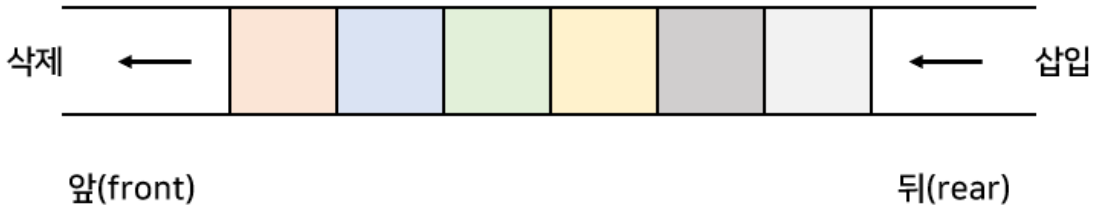
-> **Top** : 스택으로 할당된 기억공간 가장 마지막으로 삽입된 자료가 기억된 위치

-> **Bottom** : 스택의 가장 밑바닥

6. 큐(Queue)

리스트 한 쪽에서는 삽입, 다른 한 쪽에서는 삭제

- 가장 먼저 삽입된 자료가 가장 먼저 삭제되는 선입선출**(FIFO; First In First Out)** 방식
- 시작과 끝을 표시하는 두 개의 포인터가 있다



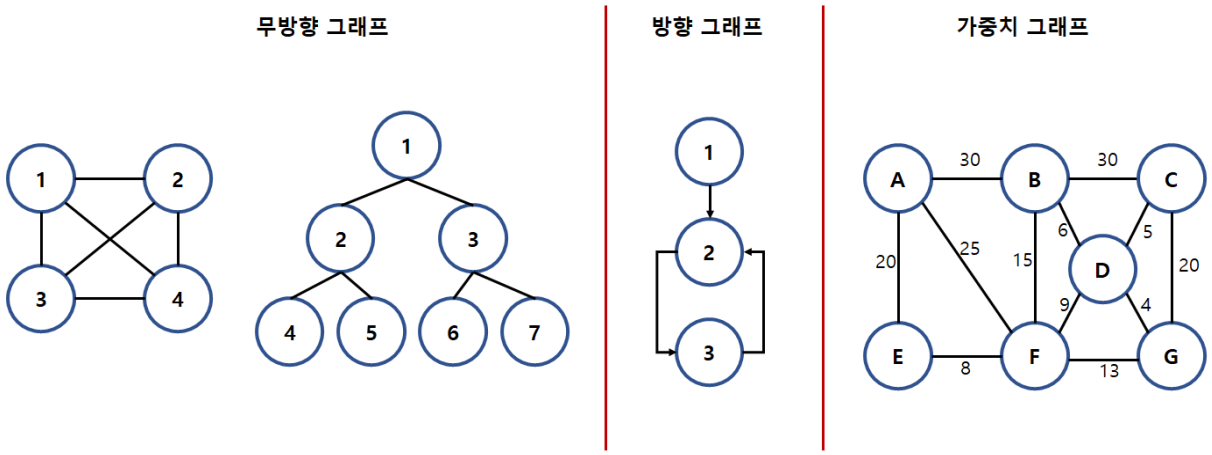
- 프런트**(F, Front)** 포인터 : 가장 먼저 삽입된 자료의 기억 공간을 가리키는 포인터, 삭제작업 할 때 사용
- 리어**(R, Rear)** 포인터 : 가장 마지막에 삽입된 자료가 위치한 기억 공간을 가리키는 포인터, 삽입 작업할 때 사용
- 큐는 운영체제의 작업 스케줄링에 사용

7. 그래프(Graph)

G는 정점 **V(Vertex)**와 간선 **E(Edge)**의 두 집합

- 간선의 방향성 유무에 따라 방향 그래프 / 무방향 그래프로 구분
- 통신망(Network), 교통망, 이항관계, 연립방정식, 유기화학 구조식, 무향선분 해법 등에 응용됨
- 트리**(Tree)**는 사이클이 없는 그래프**(Graph)**

(참고용) 그래프 종류

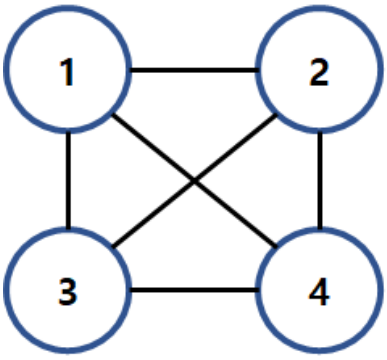


★ 방향 / 무방향 그래프의 최대 간선 수

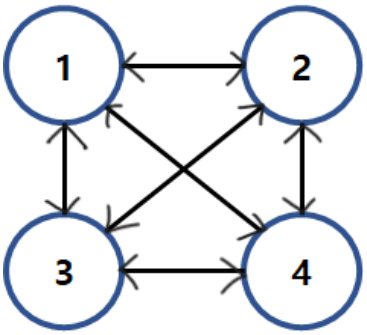
n개의 정점으로 구성된 무방향 그래프에서 최대 간선 수는 n(n-1)/2,

방향 그래프에서 최대 간선 수는 n(n-1)

ex) 정점이 4개인 경우 무방향/방향 그래프 최대 간선 수



무방향 : 4(4-1)/2 = 6



방향 : 4(4-1) = 12

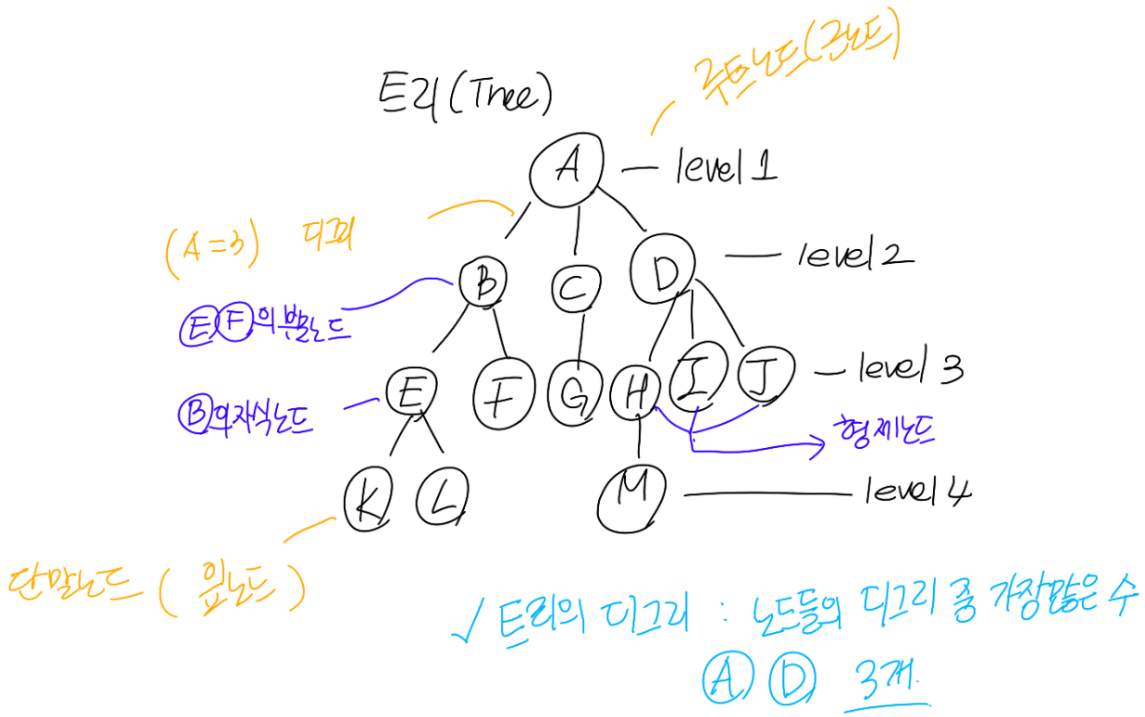
037 트리(Tree)

1. 트리의 개요

정점**(Node,** 노드**),** 선분**(Branch,** 가지**)**을 이용해 사이클을 이루지 않도록 구성한 그래프**(Graph)**의 특수한 형태

- 하나의 기억 공간을 노드**(Node)**, 노드와 노드를 연결하는 선을 링크**(Link)**.
- 가족의 계보**(**족보**),** 조직도 등을 표현하기에 적합

[사진. 트리 관련 용어]



- 노드**(Node) :** 트리의 기본 요소로서 자료 항목과 다른 항목에 대한 가지**(Branch)**를 합친 것

  (ex) A,B,C,D,E,F,G,H,I,J,K,L,M
- 근 노드**(Root Node) :** 트리의 맨 위에 있는 노드

  (ex) A

- 디그리**(Degree,** 차수**) :** 각 노드에서 뻗어 나온 가지의 수

  (ex) A = 3, B = 2, C = 1, D = 3
- 단말 노드**(Terminal Node) =** 잎 노드**(Leaf Node) :** 자식이 하나도 없는 노드**,** 즉 **Degree**가 **0**인 노드

  (ex) K,L,F,G,M,I,J
- 자식노드**(Son Node) :** 어떤 노드에 연결된 다음 레벨의 노드들

  (ex) D의 자식 노드 : H,I,J
- 부모 노드**(Parent Node) :** 어떤 노드에 연결된 이전 레벨의 노드들

  (ex) E,F의 부모 노드 : B
- 형제 노드**(Brother Node, Sibling) :** 동일한 부모를 갖는 노드들

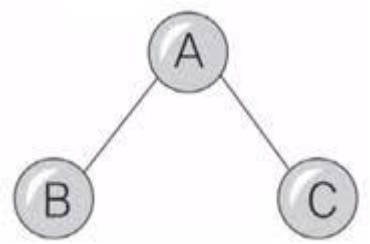
  (ex) H의 형제 노드 : I, J
- 트리의 디그리 **:** 노드들의 디그리 중에서 가장 많은 수

  (ex) 노드 A나 D가 3개의 디그리를 가지므로 트리의 디그리는 3

2. 트리의 운행법

트리를 구성하는 각 노드들을 찾아가는 방법을 운행법(Traversal)

- 이진 트리를 운행하는 방법은 산술식의 표기법과 연관성을 가짐
- 이진트리의 운행법



Root가 앞**(Pre)**에 있으면 **Preorder**

안**(In)**에 있으면 **Inorder**

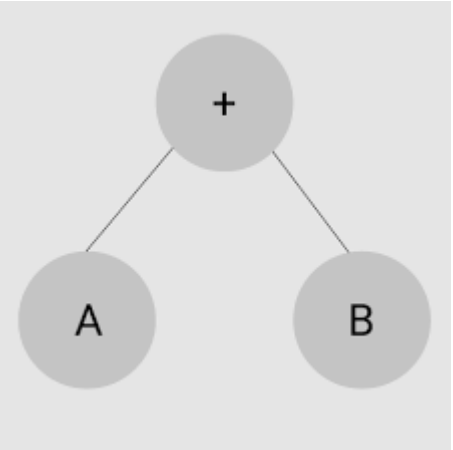
뒤**(Post)**에 있으면 **Postorder**

- **Preorder** 운행 : Root -> Left -> Right (A, B, C)
- **Inorder** 운행 : Left -> Root -> Right (B, A, C)
- **Postorder** 운행 : Left -> Right -> Root (B, C, A)

3. 수식의 표기법

산술식을 계산하기 위해 기억공간에 기억시키는 방법, 이진 트리를 많이 사용

이진트리로 만들어진 수식을 인오더, 프리오더, 포스트오더로 운행하면서 각각 중위**(Infix),** 전위**(Prefix),** 후위**(Postfix)** 표기법이 됨



- 전위 표기법**(Prefix)** : 연산자 -> Left -> Right (+AB)
- 중위 표기법**(Infix)** : Left -> 연산자 -> Right (A+B)
- 후위 표기법**(Postfix)** : Left -> Right -> 연산자 (AB+)

038 정렬(Sort)

1. 삽입정렬(Insertion Sort)

삽입정렬은 가장 간단한 정렬 방식, 이미 순서화된 파일에 새로운 하나의 레코드를 순서에 맞게 삽입시켜 정렬

- 두 번째 키와, 첫 번째 키를 비교해 순서대로 나열(1회전) -> 세 번째 키를 첫 번째, 두 번째 키와 비교해 순서대로 나열(2회전) -> n번째 키를 앞의 n-1개의 키와 비교해 알맞은 순서에 삽입해 정렬
- 평균 / 최악 모두 수행시간 복잡도 : O(n^2)

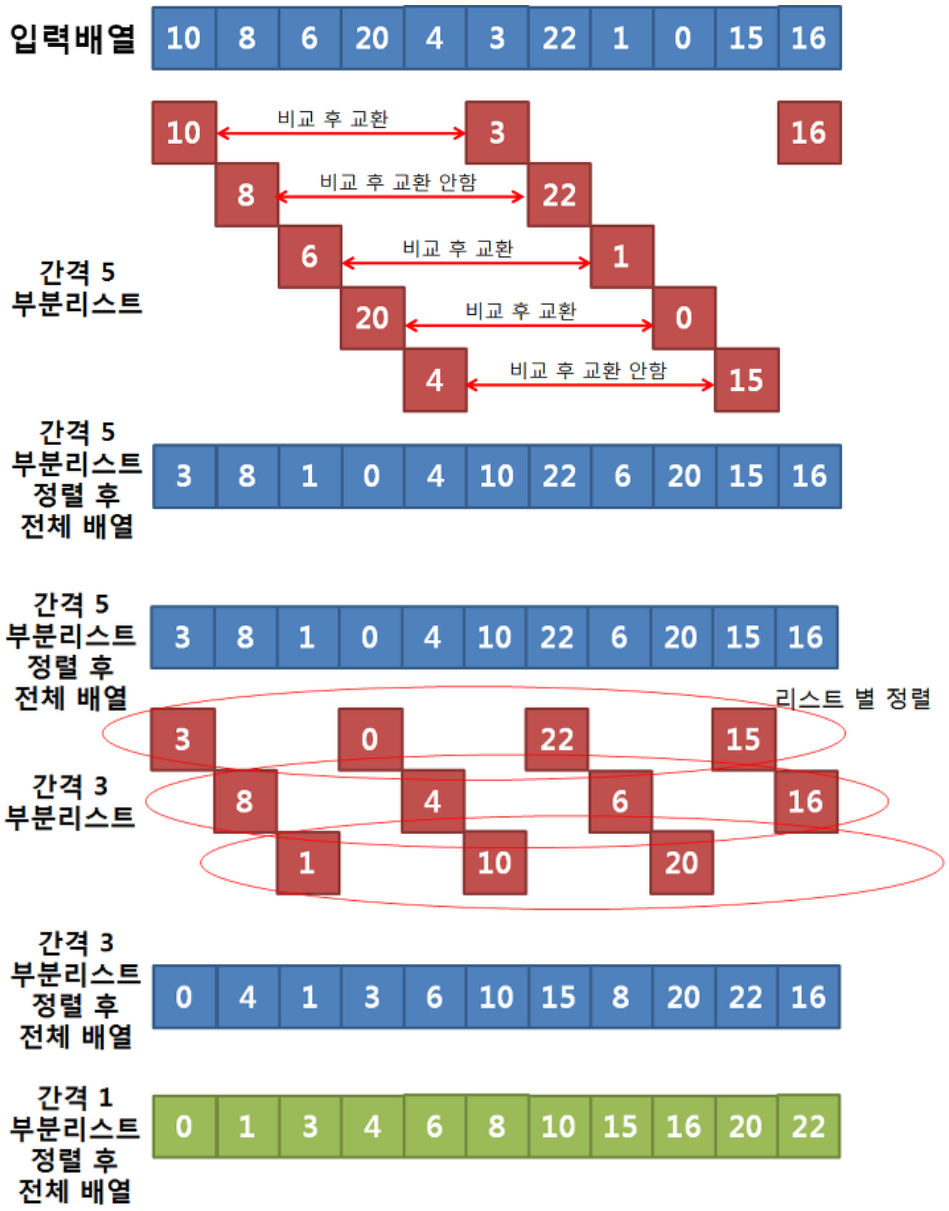
(ex) 8,5,6,2,4를 삽입정렬로 정렬

2. 쉘 정렬(Shell Sort)

삽입 정렬(Insertion Sort)을 확장한 개념

- 입력 파일을 어떤 매개변수**(h)**의 값으로 서브파일 구성, 각 서브파일을 Insertion 정렬 방식으로 순서 배열하는 과정 반복하는 정렬방식

  즉, 임의의 레코드 키와 h만큼 떨어진 곳의 레코드 키를 비교해 순서화 되어 있지 않으면 서로 교환하는 것을 반복하는 정렬
- 입력파일이 부분적으로 정렬되어 있는 경우에 유리한 방식
- 평균 수행시간 복잡도 : O(n^1.5) / 최악 수행시간 복잡 : O(n^2)



3. 선택 정렬(Selection Sort)

선택정렬은 **n**개의 레코드 중 최소값을 찾아 첫 번째 레코드 위치에 놓고**,** 나머지 **(n-1)**개 중 다시 최소값을 찾아 두 번째 레코드 위치에 놓는 방식

- 평균 **/** 최악 모두 수행시간 복잡도 **: O(n^2)**

(ex) 8,5,6,2,4를 선택정렬로 정렬

4. 버블 정렬(Bubble Sort)

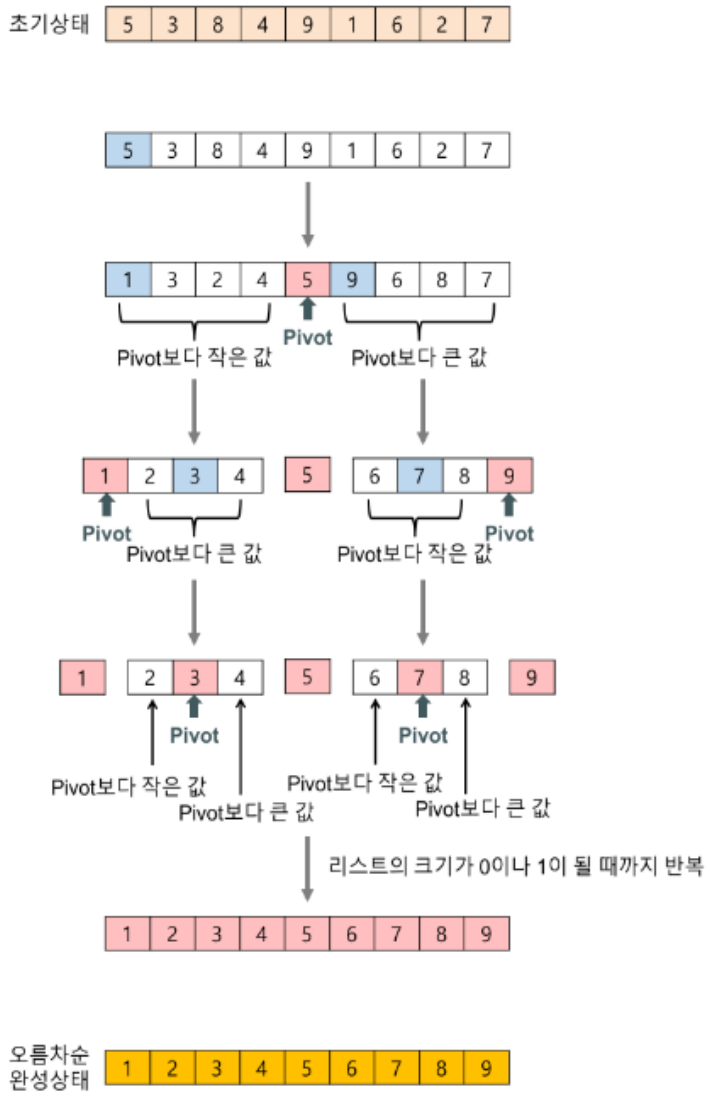
- 버블 정렬은 주어진 파일에서 인접한 두 개의 레코드 키 값을 비교해 그 크기에 따라 레코드 위치 서로 교환하는 정렬 방식
- 계속 정렬 여부를 플래그 비트(f)로 결정
- 평균 / 최악 모두 수행시간 복잡도 : O(n^2)

(ex) 8,5,6,2,4를 버블정렬로 정렬

5. 퀵 정렬(Quick Sort)

레코드의 많은 자료 이동을 없애고 하나의 파일을 부분적으로 나누어 가면서 정렬하는 방법
키를 기준으로 작은 값은 왼쪽, 큰 값은 오른쪽 서브파일로 분해시키는 방식으로 정렬

- 위치에 관계없이 임의의 키를 분할원소로 사용
- 정렬방식 중 가장 빠른 방식
- 프로그램에서 되부름을 이용하기 때문에 스택(Stack) 필요
- 분할(Divid), 정복(Conquer)을 통해 자료 정렬
    - 분할(Divid) : 기준값인 피봇(Pivot)을 통해 정렬할 자료들을 2개의 부분집합으로 나눔
    - 정복(Conquer) : 부분집합의 원소들 중 피봇(Pivot)보다 작은 원소들은 왼쪽, 피봇보다 큰 원소들은 오른쪽 부분집합으로 정렬
- 평균 수행시간 복잡도 : O(nlog₂n) / 최악 수행시간 복잡도 : O(n^2)

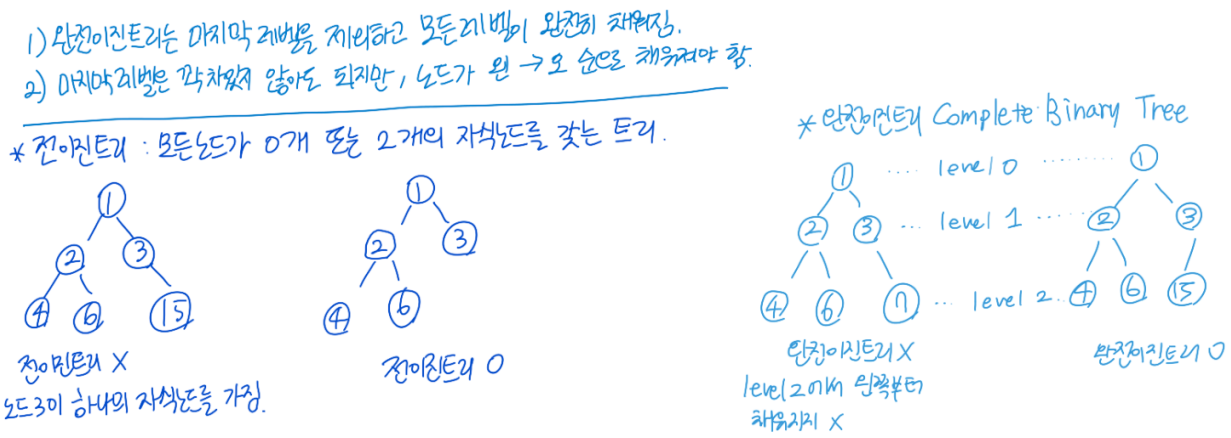


6. 힙 정렬(Heap Sort)

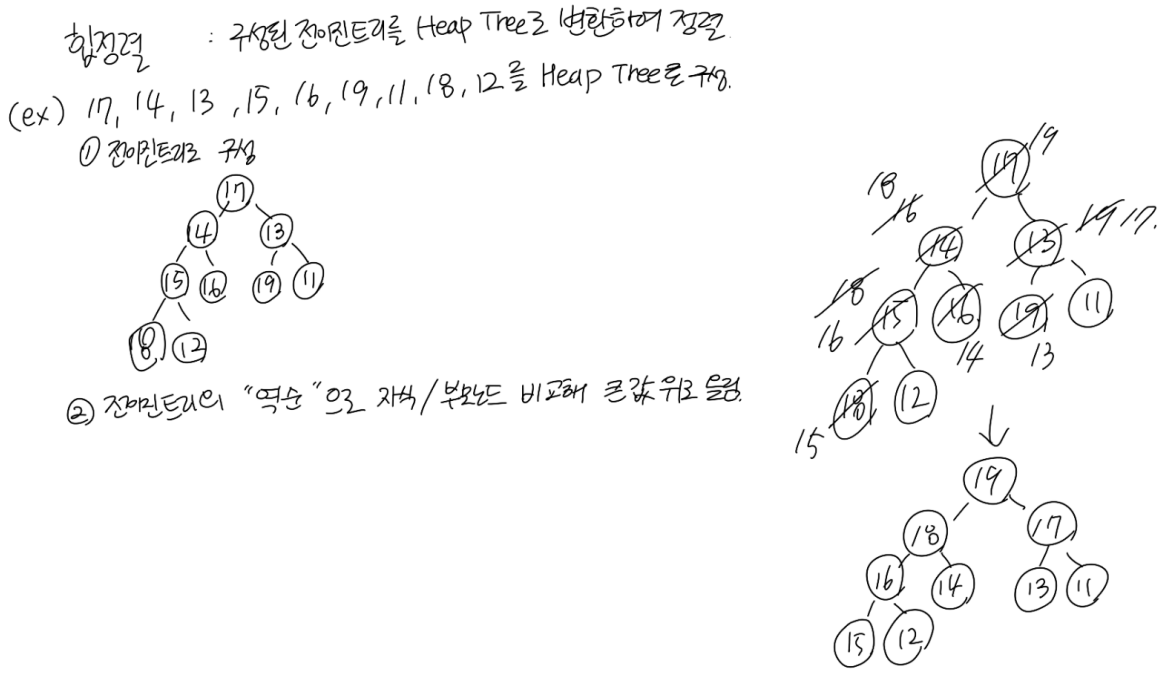
힙 정렬은 전이진 트리(완전이진트리, **Complete Binary Tree**)를 이용한 정렬 방식

- 완전이진트리는 마지막 레벨을 제외하고 모든 레벨이 완전히 채워짐
- 마지막 레벨은 꽉 차있지 않아도 되지만, 노드가 왼 -> 오 순으로 채워져야 함
- 전이진트리는 모든 노드가 0개 또는 2개의 자식노드를 갖는 트리
- 평균 / 최악 모두 시간복잡도는 $O(nlog_2n)$

[완전이진트리 / 전 이진트리 예시 사진]



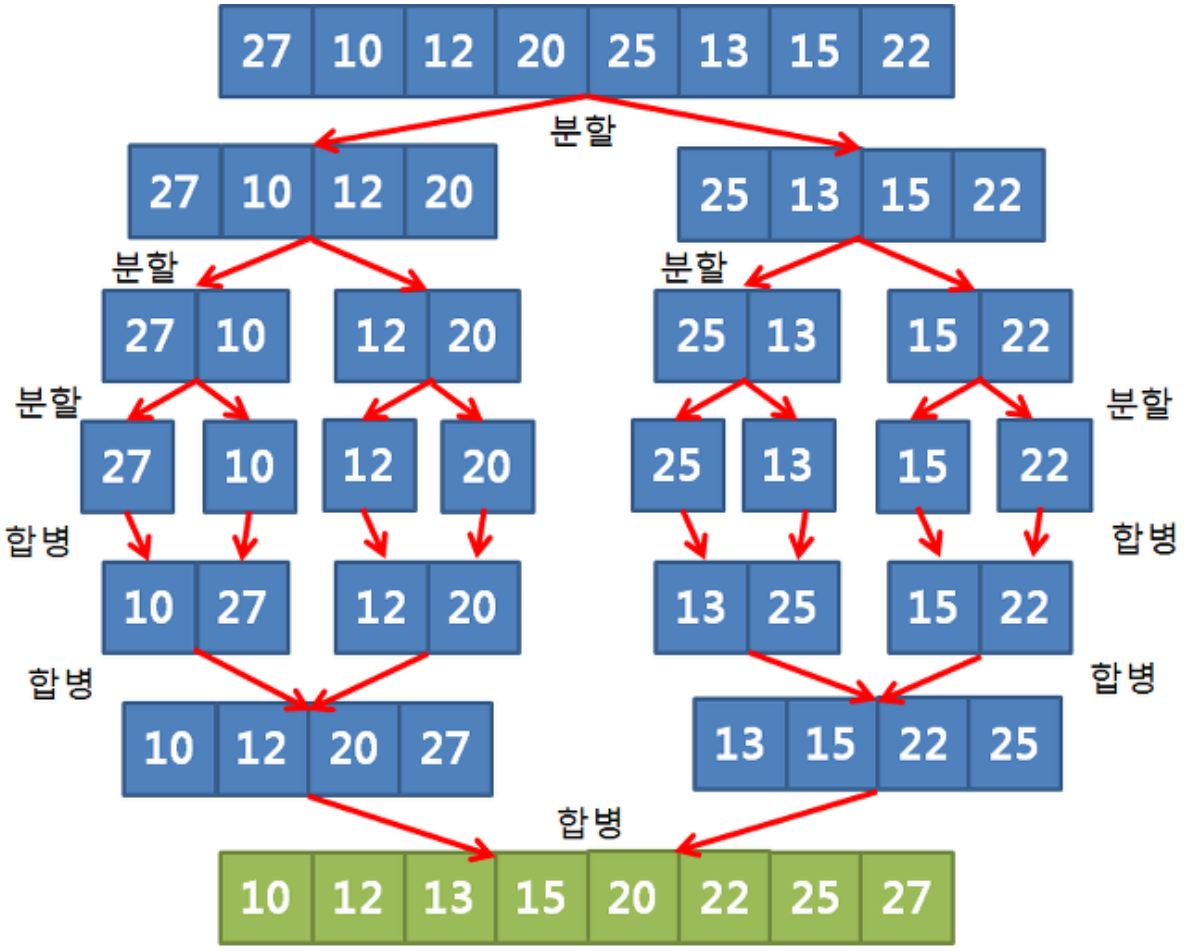
[힙 정렬 구현]



7. 2-Way 합병 정렬(Merge Sort)

2-Way Merge Sort는 이미 정렬되어 있는 두 개의 파일을 한 개의 파일로 합병하는 정렬 방식

- 두 개의 키들을 한 쌍으로 하여 각 쌍에 대한 순서 정함
- 순서대로 정렬된 각 쌍의 키들을 합병해 하나의 정렬된 서브리스트로 만듦
- 위 과정에서 정렬된 서브리스트들을 하나의 정렬된 파일이 될 때까지 반복
- 평균 / 최악 모두 시간복잡도는 $O(n\log_2 n)$



(ex) 71,2,38,5,7,61,11,26,53,42를 2-Way 합병 정렬로 정렬

8. 기수 정렬(Radix Sort) = Bucket Sort

기수 정렬은 Queue를 이용해 자릿수(Digit) 별로 정렬

- 레코드의 키 값을 분석해 같은 수 또는 같은 문자끼리 그 순서에 맞는 버킷에 분배했다가 버킷의 순서대로 레코드를 꺼내 정렬
- 평균 / 최악 모두 시간 복잡도는 O(dn)

039 데이터베이스 개요

1. 데이터 저장소

소프트웨어 개발 과정에서 다루어야 할 데이터들을 논리적인 구조로 조직화 or 물리적인 공간에 구축

- 논리 데이터저장소, 물리 데이터저장소로 구분
- 논리 데이터저장소 : 데이터 및 데이터 간의 연관성, 제약조건을 식별해 논리적인 구조로 조직화
- 물리 데이터 저장소 : 논리데이터 저장소에 저장된 데이터, 구조들을 소프트웨어가 운영될 환경의 물리적 특성을 고려해 하드웨어적인 저장장치에 저장
- 논리 데이터저장소를 거쳐 물리 데이터저장소를 구축하는 과정은 데이터베이스를 구축하는 과정과 동일

2. 데이터베이스

특정 조직의 업무를 수행하는데 필요한 상호 관련된 데이터들의 모임

- 통합된 데이터**(Integrated Data)** : 자료의 중복을 배제한 데이터의 모임
- 저장된 데이터**(Stored Data)** : 컴퓨터가 접근할 수 있는 저장매체에 저장된 자료

- 운영 데이터**(Operational Data)** : 조직의 고유한 업무를 수행하는 데 존재가치가 확실하고, 없어서는 안될 필요한 자료
    - 데이터베이스에 데이터들을 저장하여 유지, 관리하는 데 사용되는 데이터
    - 데이터베이스 작업의 예시 : 급여기록, 통화기록, 고객정보, 직원데이터 및 판매데이터와 같은 대량의 특정정보를 저장, 수정, 관리 및 검색할 수 있는 능력
    - 하지만 단순한 입,출력 자료 or 작업처리상 일시적으로 필요한 임시자료는 운영자료로 취급 x
- 공용데이터**(Shared Data)** : 여러 응용시스템들이 공동으로 소유하고 유지하는 자료

3. DBMS(DataBase Management System; 데이터베이스 관리 시스템)

DBMS란 사용자와 데이터베이스 사이에서 사용자의 요구에 따라 정보를 생성, 데이터베이스를 관리해 주는 소프트웨어

- DBMS는 기존의 파일 시스템이 갖는 데이터의 종속성/중복성 문제를 해결하기 위해 제안된 시스템.
- 모든 응용 프로그램들이 데이터베이스를 공용할 수 있도록 관리.
- DBMS는 데이터베이스의 구성, 접근방법, 유지관리에 대한 모든 책임을 짐.
- DBMS 필수 기능에는 정의**(Definition),** 조작**(Manipulation),** 제어**(Control)** 기능이 있음
    - 정의기능 : 모든 응용 프로그램들이 요구하는 데이터 구조를 지원하기 위해 데이터베이스에 저장될 데이터의 형**(Type)**과 구조에 대한 정의, 이용방식, 제약조건 등을 명시하는 기능
    - 조작기능 : 데이터 검색, 갱신, 삽입, 삭제 등을 체계적으로 처리하기 위해 사용자, 데이터베이스 사이의 인터페이스 수단을 제공하는 기능
    - 제어기능

- 데이터베이스를 접근하는 갱신, 삽입, 삭제 작업이 정확하게 수행되어 데이터의 **"무결성"**이 유지되도록 제어해야 함. (무결성 : **DB**에 저장된 데이터 값과 실제 값이 일치하는 정확성)
- 정당한 사용자가 허가된 데이터만 접근할 수 있도록 보안**(Security)**을 유지, 권한**(Authority)**을 검사할 수 있어야 함.
- 여러 사용자가 데이터베이스를 동시에 접근하여 데이터를 처리할 때 처리결과가 항상 **"정확성"**을 유지하도록 병행제어**(Concurrency Control)**를 할 수 있어야 함.

4. DBMS 장단점

장점

➔ 데이터의 논리적 **/** 물리적 독립성 보장

➔ 데이터의 중복을 피할 수 있어 기억공간 절약

➔ 저장된 자료를 공동으로 이용

➔ 데이터의 일관성 유지 (원인과 결과의 의미가 연속적으로 보장되어 변하지 않는 상태)

➔ 보안 유지

➔ 데이터를 표준화

➔ 데이터를 통합하여 관리

➔ 항상 최신의 데이터 유지

➔ 데이터의 실시간 처리 가능

단점

➔ 데이터베이스 전문가 부족

➔ 전산화 비용 증가

➔ 대용량 디스크로의 집중적인 **Access**로 과부화**(Overhead)** 발생

◆ DBMS는 고가의 제품, 컴퓨터 시스템의 지원을 많이 사용. 특히, 주기억장치를 많이 차지하기 때문에 DBMS를 운영하기 위해서는 메모리

용량이 더 필요하고

더 빠른 CPU 요구. 결과적으로 시스템 운영비의 오버헤드를 가중시키게 됨.

➔ 파일의 예비**(Backup)**와 회복**(Recovery)**이 어려움

➔ 시스템 복잡

➢ 데이터의 독립성

- 종속성에 대비되는 말로, **DBMS**의 궁극적 목표이기도 함.
    - 논리적 독립성 : 응용프로그램과 데이터베이스를 독립시킴으로써, 데이터의 논리적 구조를 변경시키더라도 응용프로그램은 변경**X**
    - 물리적 독립성 : 응용프로그램과 보조기억장치 같은 물리적장치를 독립시킴으로써, 데이터베이스 시스템의 성능 향상을 위해 새로운 디스크를 도입하더라도 응용프로그램에는 영향**X**, 데이터의 물리적 구조만을 변경

5. 스키마**(Schema)**

스키마(Schema)는 데이터베이스의 구조와 제약조건에 관한 전반적인 명세**(Specification)**를 기술**(Description)**한 메타데이터**(Meta-Data)**의 집합

- 스키마는 데이터베이스를 구성하는 데이터 개체**(Entity),** 속성**(Attribute),** 관계**(Relationship)** 및 데이터조작 시 데이터 값들이 갖는 제약조건 등에 관해 전반적으로 정의
- 스키마는 사용자의 관점에 따라 나뉨
    - 외부스키마 : 사용자 / 응용프로그래머가 각 개인의 입장에서 필요로 하는 데이터베이스의 논리적 구조를 정의한 것
    - 개념스키마 : 데이터베이스의 전체적인 논리적 구조, 모든 응용프로그램이나 사용자들이 필요로 하는 데이터를 종합한 조직 전체의 데이터베이스로 하나만 존재
    - 내부스키마 : 물리적 저장장치의 입장에서 본 데이터베이스의 구조, 실제로 데이터베이스에 저장될 레코드의 형식을 정의하고 저장데이터 항목의 표현방법, 내부레코드의 물리적순서 등을 나타냄.

040 데이터 입,출력

1. 데이터 입,출력의 개요

소프트웨어의 기능 구현을 위해 데이터베이스에 데이터를 입력 **or** 출력하는 작업을 의미

- 단순 입/출력 뿐 아니라 데이터를 조작하는 모든 행위를 의미, 이와 같은 작업을 위해 **SQL(Structured Query Language)**을 사용
- 데이터 입,출력을 소프트웨어에 구현하기 위해 개발 코드 내에 **SQL** 코드를 삽입하거나, 객체와 데이터를 연결하는 것을 데이터접속**(Data Mapping).**
- 트랜잭션**(Transaction)** : SQL을 통한 데이터베이스의 조작을 수행할 때 하나의 논리적 기능을 수행하기 위한 작업의 단위 or 한꺼번에 모두 수행되어야 할 일련의 연산들

2. SQL(Structured Query Language)

질의어**(Query Language)** : 데이터베이스 파일과 범용 프로그래밍 언어를 정확히 알지 못하는 단말 사용자들이 단말기를 통해서 대화식으로 쉽게 **DB**를 이용할 수 있도록 되어있는 비절차어의 일종)
국제표준 데이터베이스 언어로, 많은 회사에서 관계형 데이터베이스(RDB)를 지원하는 언어로 채택.
관계형 데이터베이스**(Relational DataBase)** : **2**차원적인 표**(Table)**를 이용해서 데이터 상호관계를 정의하는 데이터베이스.

- 관계대수, 관계해석을 기초로 한 혼합 데이터언어
    - 관계대수 : 관계형 데이터베이스에서 원하는 정보와 그 정보를 검색하기 위해서 어떻게 유도하는가를 기술하는 절차적언어.
      해를 구하기 위해 수행해야 할 연산의 순서 명시

    - 관계해석 : 관계 데이터의 연산을 표현하는 방법, 원하는 정보를 정의할 때는 계산 수식 사용
      비 절차적 특성(원하는 정보가 무엇이라는 것만 정의)
- 질의어지만 질의 기능만 있는 것이 아니라 데이터구조의 정의, 데이터조작, 데이터제어 기능을 모두 갖춤

- SQL은 데이터 정의어**(DDL),** 데이터 조작어**(DML),** 데이터 제어어**(DCL)**로 구분됨
    - 데이터 정의어**(DDL; Data Define Language)** : **SCHEMA, DOMAIN, TABLE, VIEW, INDEX**를 정의 / 변경 / 삭제할 때 사용하는 언어
    - 데이터 조작어**(DML; Data Manipulation Language)** : 데이터베이스 사용자가 응용프로그램이나 질의어를 통하여 저장된 데이터를 실질적으로 처리하는 데 사용되는 언어
    - 데이터 제어어**(DCL; Data Control Language)** : 데이터의 보안, 무결성, 회복, 병행수행 제어 등을 정의

3. 데이터 접속**(Data Mapping)**

데이터접속은 소프트웨어의 기능 구현을 위해 프로그래밍 코드와 데이터베이스의 데이터를 연결**(Mapping)**하는 것

- **SQL Mapping** : 프로그래밍 코드 내에 **SQL**을 직접 입력하여 **DBMS**의 데이터에 접속하는 기술, 관련 프레임워크에는 **JDBC, ODBC, MyBatis** 등
- **ORM(Object-Relational Mapping)** : 객체지향 프로그래밍의 객체**(Object)**와 관계형**(Relational)** 데이터베이스의 데이터를 연결**(Mapping)**하는 기술, 관련 프레임워크에는 **JPA, Hibernate, Django** 등

4. 트랜잭션**(Transaction)**

트랜잭션은 데이터베이스의 상태를 변환시키는 하나의 논리적 기능을 수행하기 위한 작업의 단위 또는 한꺼번에 모두 수행되어야 할 일련의 연산들.

- 트랜잭션을 제어하기 위해서 사용하는 명령어를 **TCL(Transaction Control Language)**라고 하며, TCL의 종류에는 **COMMIT, ROLLBACK, SAVEPOINT.**
    - **COMMIT** : 트랜잭션 처리가 정상적으로 종료되어 트랜잭션이 수행한 변경 내용을 데이터베이스에 반영하는 명령어
    - **ROLLBACK** : 하나의 트랜잭션 처리가 비정상으로 종료되어 데이터베이스의 일관성이 깨졌을 때 트랜잭션이 행한 모든 변경작업을 취소하고 이전의 상태로 되돌리는 연산
    - **SAVEPOINT(=CHECKPOINT)** : 트랜잭션 내에 **ROLLBACK**할 위치인 저장점을 지정하는 명령어

041 절차형 SQL

1. 절차형 SQL의 개요

C, JAVA등의 프로그래밍 언어와 같이 연속적인 실행 / 분기 / 반복 등의 제어가 가능한 **SQL** 의미, 데이터베이스 전용의 간단한 프로그래밍.

- 일반적인 프로그래밍 언어에 비해 효율은 떨어지지만, 단일 **SQL** 문장으로 처리하기 어려운 연속적인 작업들을 처리하는 데 적합.
- 절차형 **SQL**을 활용하여 다양한 기능을 수행하는 저장 모듈 생성 가능
- DBMS 엔진에서 직접 실행되기 때문에 입,출력 패킷이 적은 편
    - 패킷 : (pack + bucket), 우체국에서 화물을 적당한 덩어리로 나눠 행선지를 표시하는 꼬리표를 붙이는데, 이러한 방식을 데이터 통신에 접목한 것 컴퓨터 간에 데이터를 주고받을 때 네트워크를 통해서 전송되는 데이터 조각.
- BEGIN ~ END 형식으로 작성되는 블록 구조로 되어있기 때문에 기능별 모듈화 가능
- 절차형 SQL 종류에는 프로시저, 트리거, 사용자 정의 함수가 있다.
    - 프로시저**(Procedure)** : 특정 기능을 수행하는 일종의 트랜잭션 언어, 호출을 통해 실행되어 미리 저장해 놓은 **SQL** 작업 수행
    - 트리거**(Trigger)** : 데이터베이스 시스템에서 데이터의 입력, 갱신, 삭제 등의 이벤트가 발생할 때마다 관련 작업이 자동으로 수행(이벤트 : 시스템에 어떤

일이 발생, 트리거에서 이벤트는 데이터의 입력, 갱신, 삭제와 같이 데이터 조작 작업이 발생했음을 의미)

- 사용자 정의 함수 : 프로시저와 유사하게 SQL을 사용하여 일련의 작업을 연속적으로 처리, 종료 시 예약어 **Return**을 사용해 처리 결과를 단일값으로 반환.