ORACLE B-TREE

DATA STRUCTURE

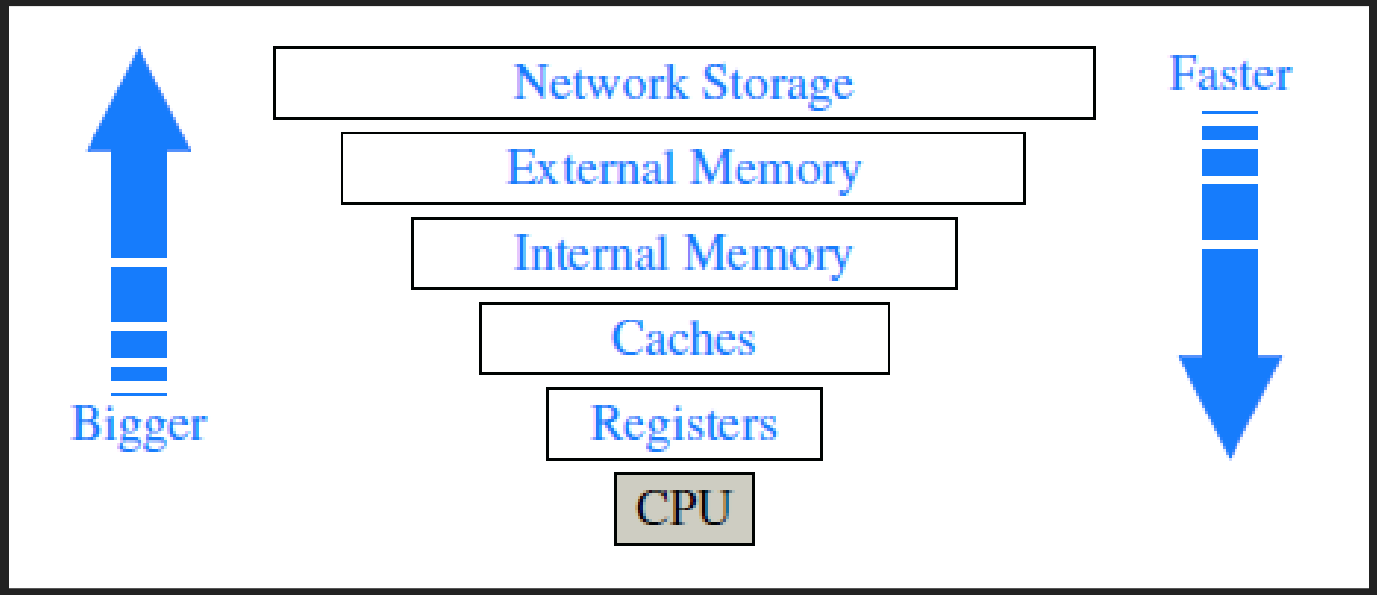
Created by Jongwon

INTRO...

- Performance
- Data Structure
- Oracle B-Tree Structure
- Oracle B-Tree Operation
- Reverse Key Index
- Index Key Compression

PERFORMANCE?

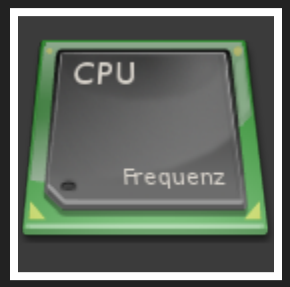
MEMORY SYSTEMS

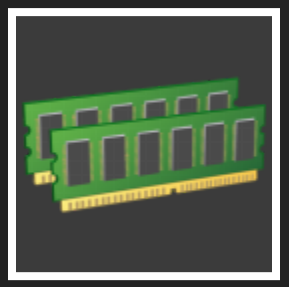


C entral P rocessing U nit

I nput / O utput

문제를 푸는 상황과 비교해 봅시다.





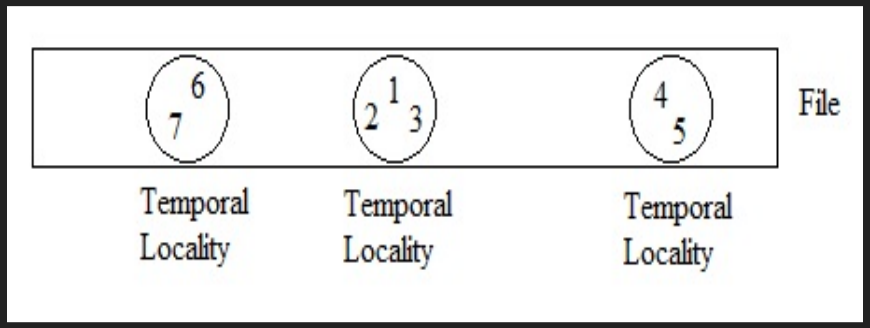
ACCESS TIME?

Internal Memory의 Access Time만 해도
Cache Memory의 **10배 ~ 100배**
Disk는... Internal Memory의 **100,000배 ~ 1,000,000배**

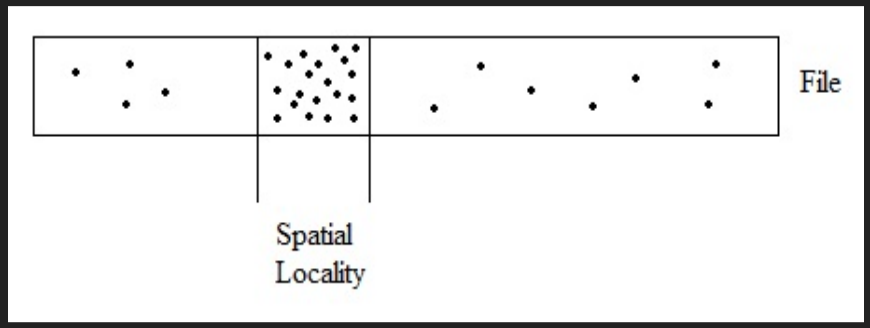
CACHING

데이터가 커서 메모리에 모두 올릴 수가 없다.
그럼 자주 쓰는 것만 올려보자.

LOCALITY



Temporal Locality



Spatial Locality

메모리는 한정된 공간이라서
안쓰던 영역이 메모리에 올라오는 경우가 있기 마련

그 영역은 자주 쓰일까?

Scheduling

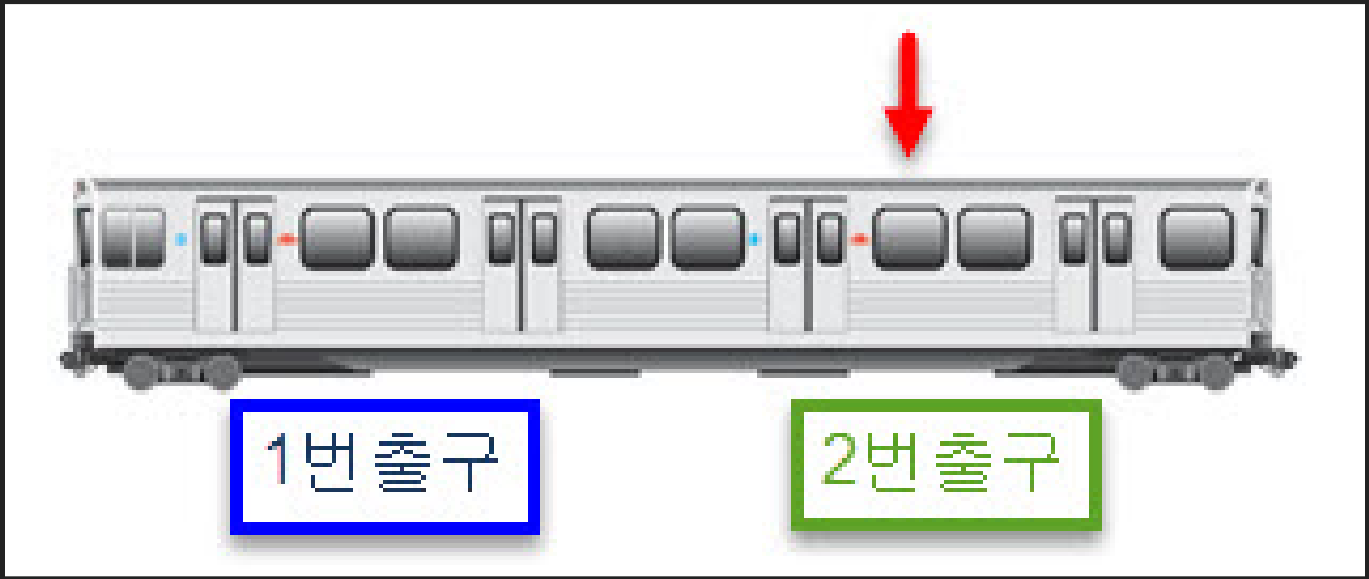
PERFORMANCE?

Access time 뿐만 아니라

ACCESS 횟수를 줄여보자.

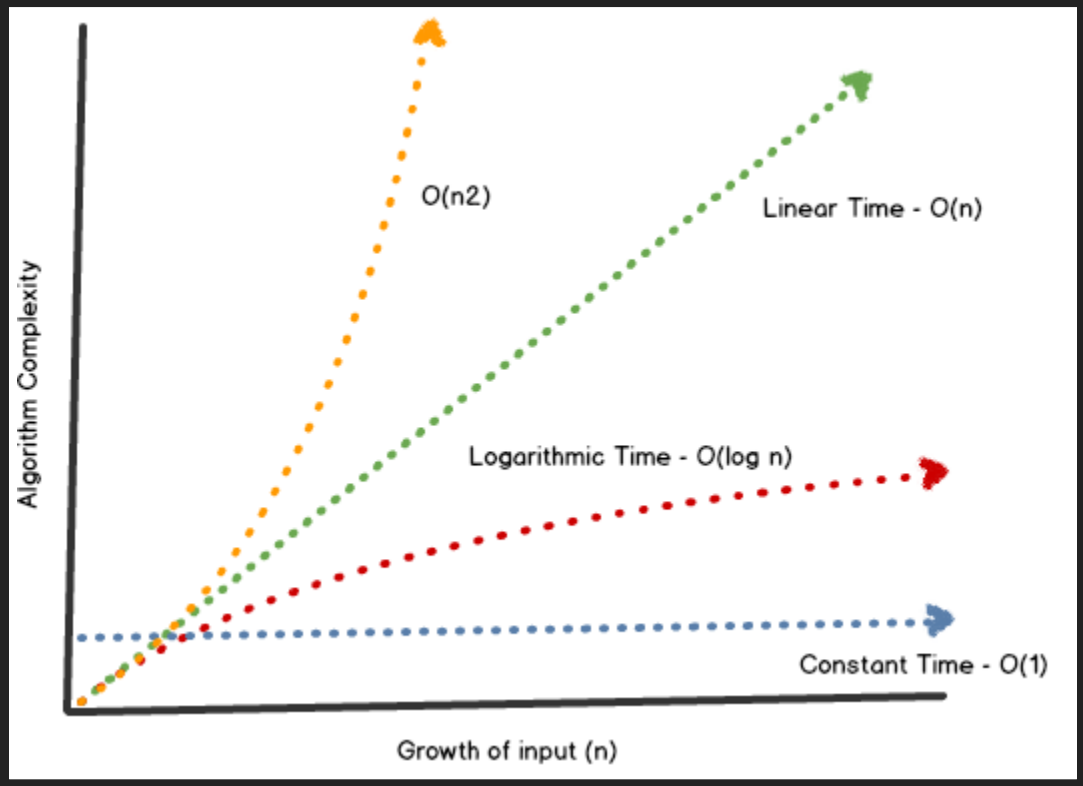
DATA STRUCTURE

ACCESS를 어떻게 하는가



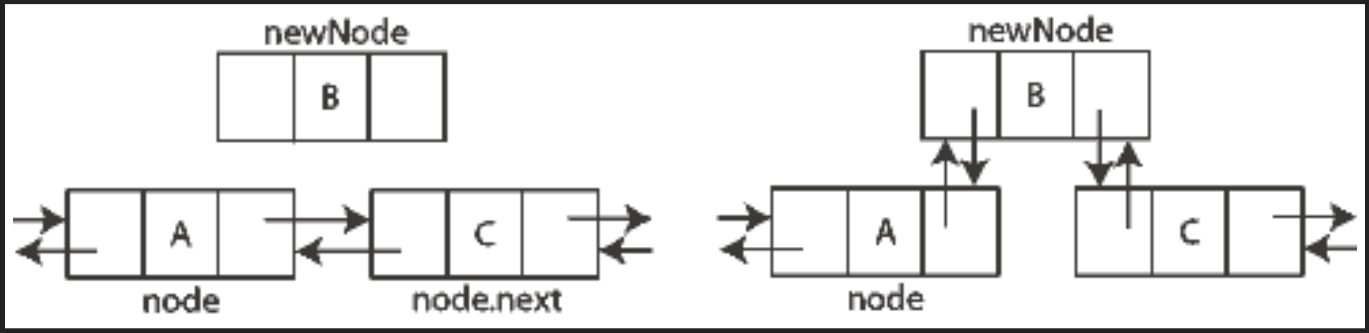
ADDRESS

BIG-O-NOTATION



특정 시점 이후에는 이 **선**을 넘을 수 없다

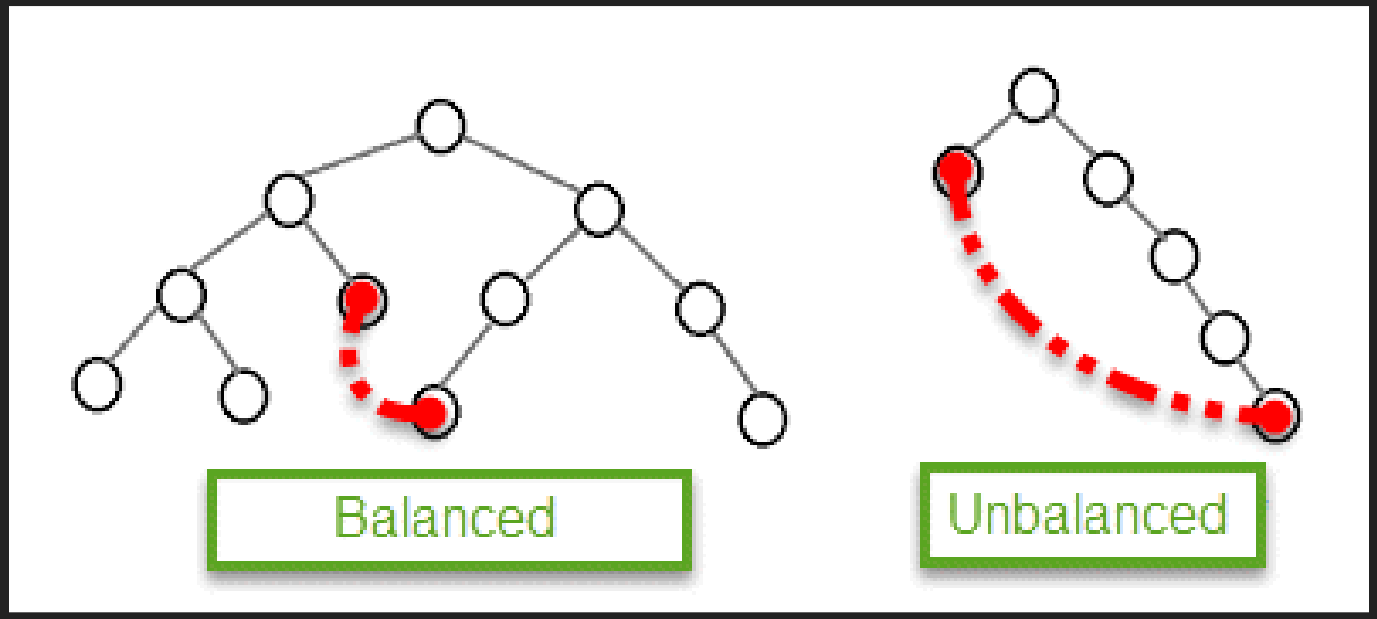
DOUBLY LINKED LIST



이전 또는 다음 블록의 주소를 한 **노드**에 함께 기록

$O(n)$

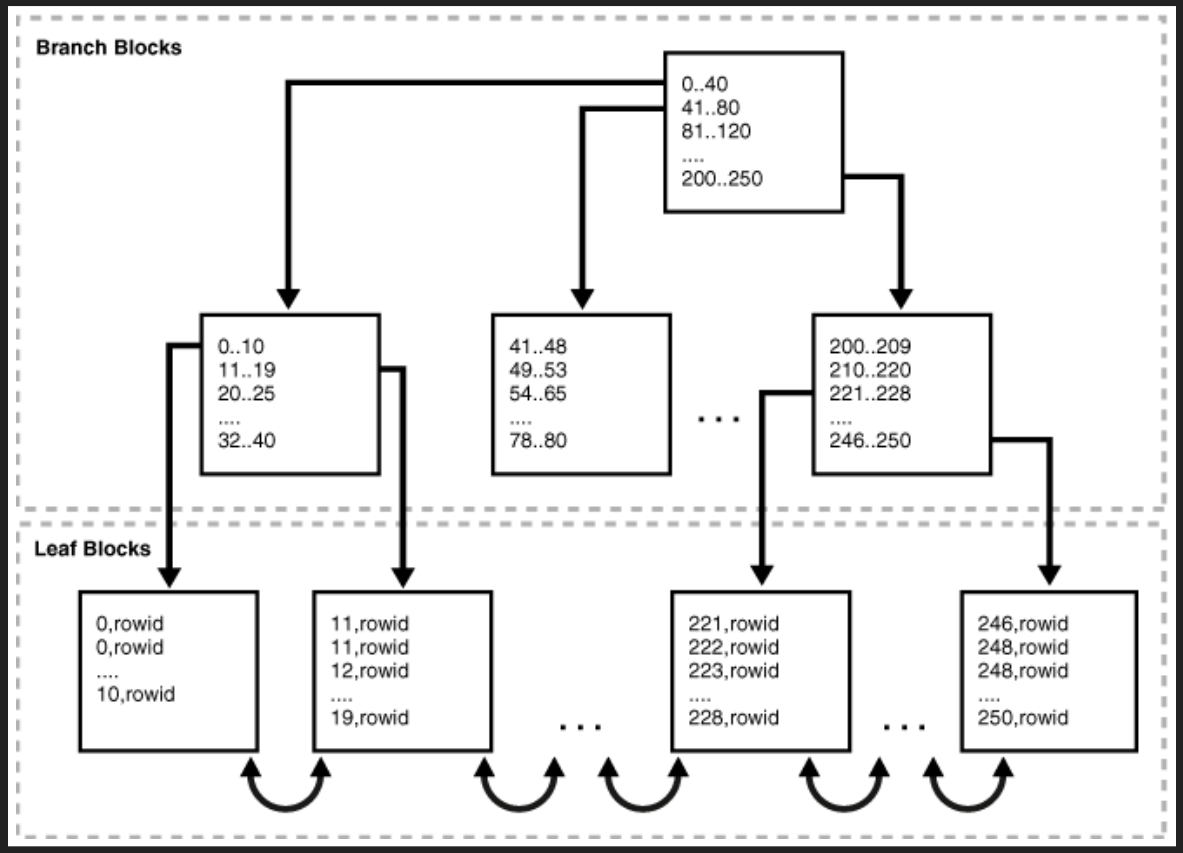
BALANCED TREE



leaf로 가는 **DEPTH**가 거의 동일하다.

$O(log\ n)$

ORACLE B-TREE STRUCTURE

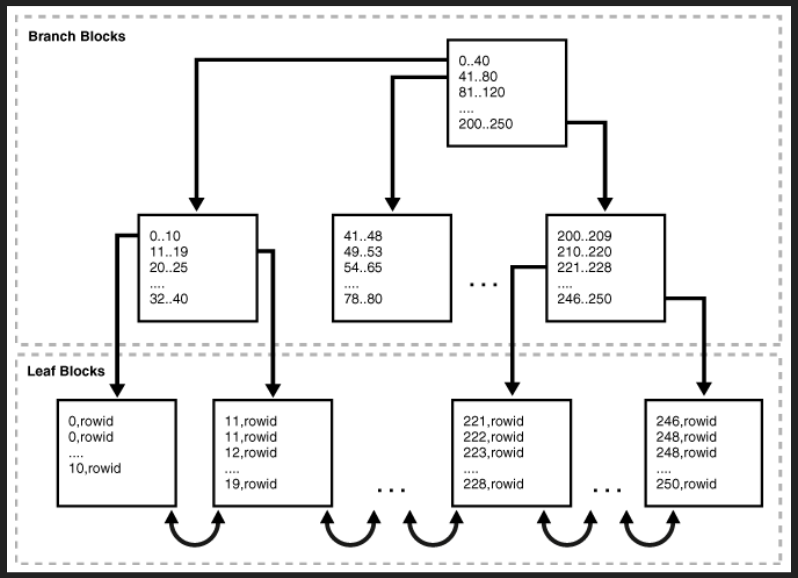


Scan을 위한 양방향 Linked list 구조의 leaf 노드

B-TREE OPERATION

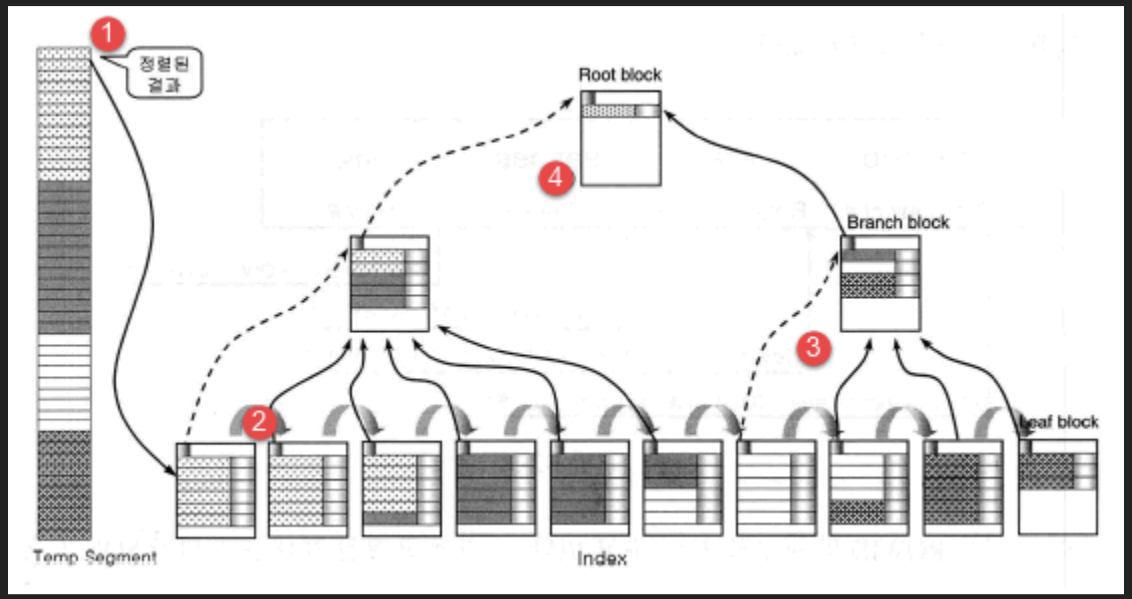
- Selection
- Index Block Creation
- Index Block Split(Data Insert)
- Data Delete
- Data Update

SELECTION



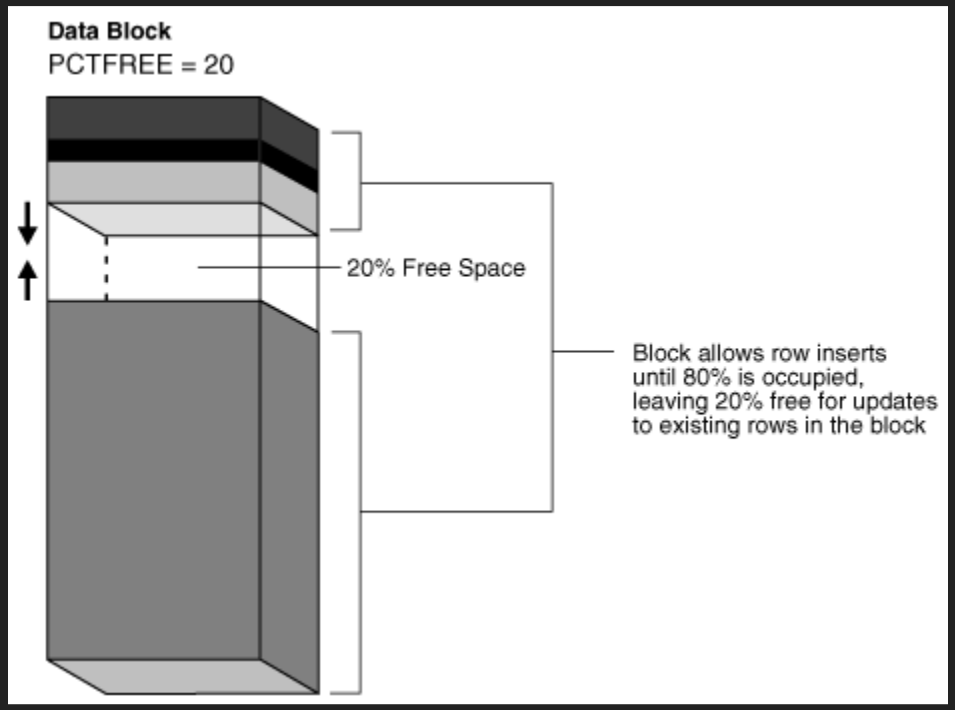
1. Root 블록을 찾는다.
2. 주어진 값 이상의 최소값을 찾아 해당 블록을 찾는다.
3. Leaf 블록을 찾을 때 까지 반복한다.
4. KEY가 존재하면 ROWID를 이용해 테이블을 Access한다.

INDEX BLOCK CREATION



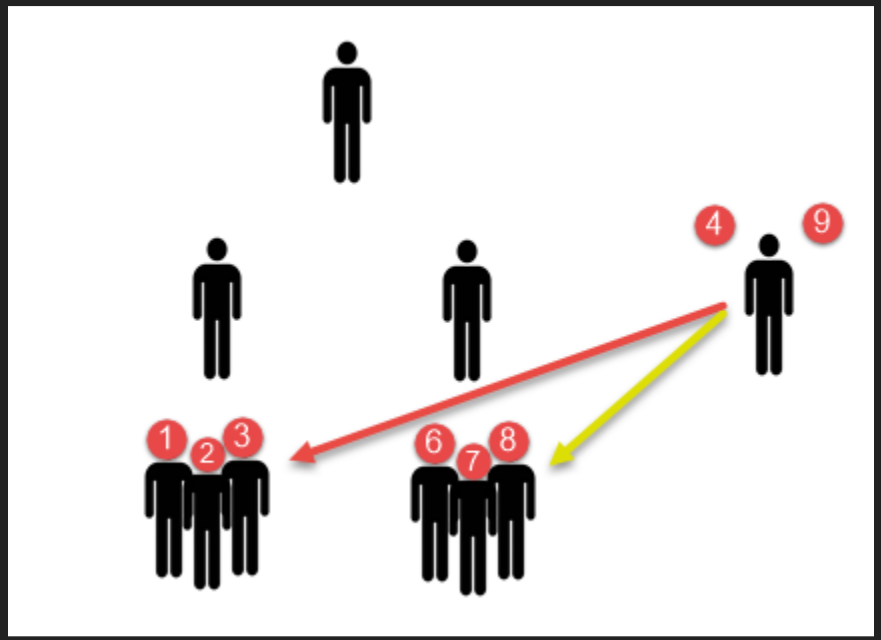
1. 테이블을 Access 하여 정렬을 수행하여 Leaf 블록에 기록한다.
2. Leaf블록이 차면 Branch블록을 만들어 블록 헤더에 주소를 기록하고, 새로 Leaf블록을 할당한다.
3. 위 작업을 반복하여 Branch블록도 차면 새로운 Branch블록을 할당한다.
4. 이 때 새로운 Root블록도 만든다. 이 작업을 반복한다.

PCTFREE



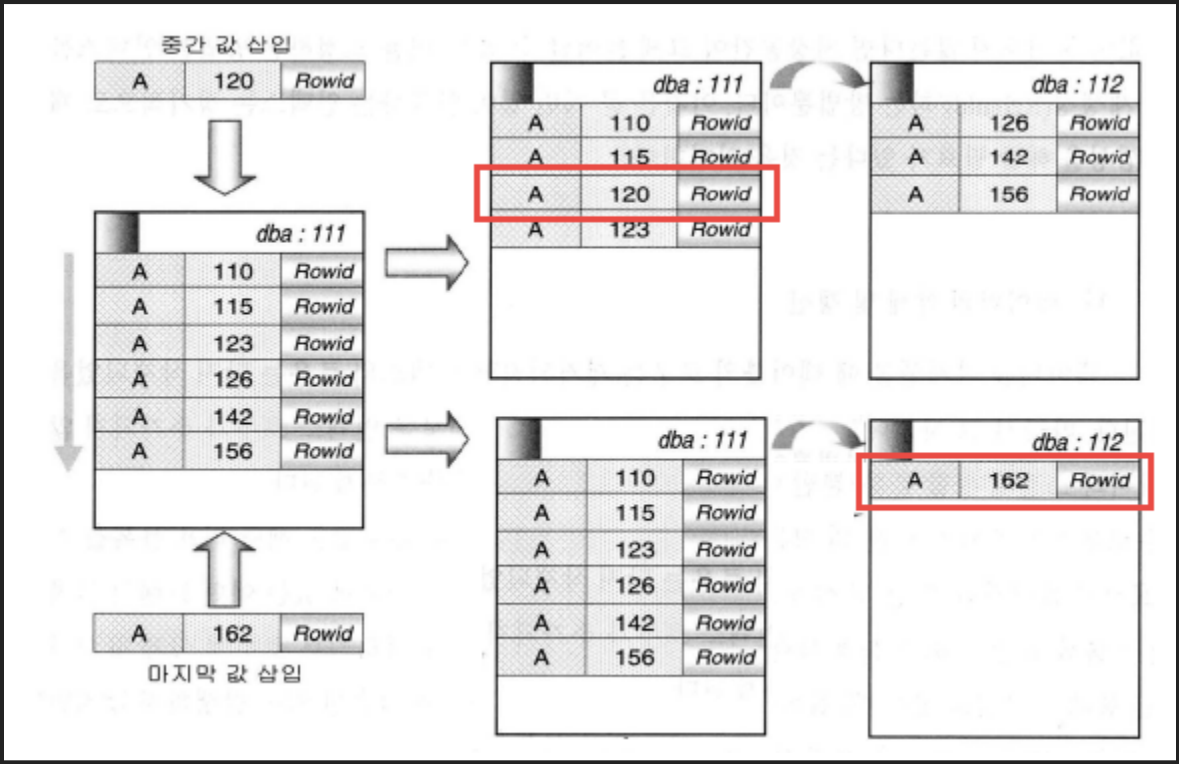
미래의 UPDATE를 위한 가용공간

INDEX BLOCK SPLIT



어떻게 분할할 것인가

INDEX BLOCK SPLIT



분할한 양쪽을 $\frac{2}{3}$ 만큼씩 채우도록 하면서
양쪽을 모두 새로 편성한다.

DATA DELETE

테이블의 row는 제거되나
index의 row는 삭제된 **표시**만 추가된다.
그 자리에 새로운 index row가 추가되지 않는다면

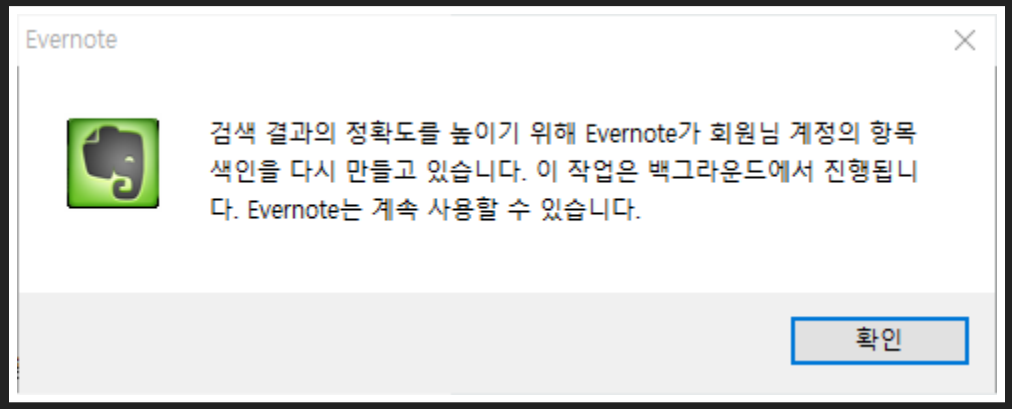
저장공간의 낭비, 스캔할 블록 증가.

DATA UPDATE

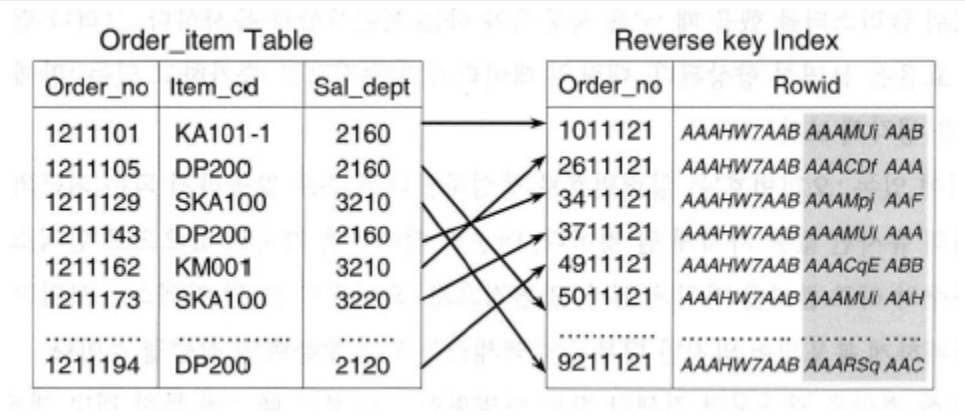
Index row는 정렬되어 저장된다.

Update 작업은 Delete 후 Insert 작업이 발생한다.

DML문이 많이 수행되는 테이블은
INDEX를 **정기적으로 재생성**할 필요가있다.



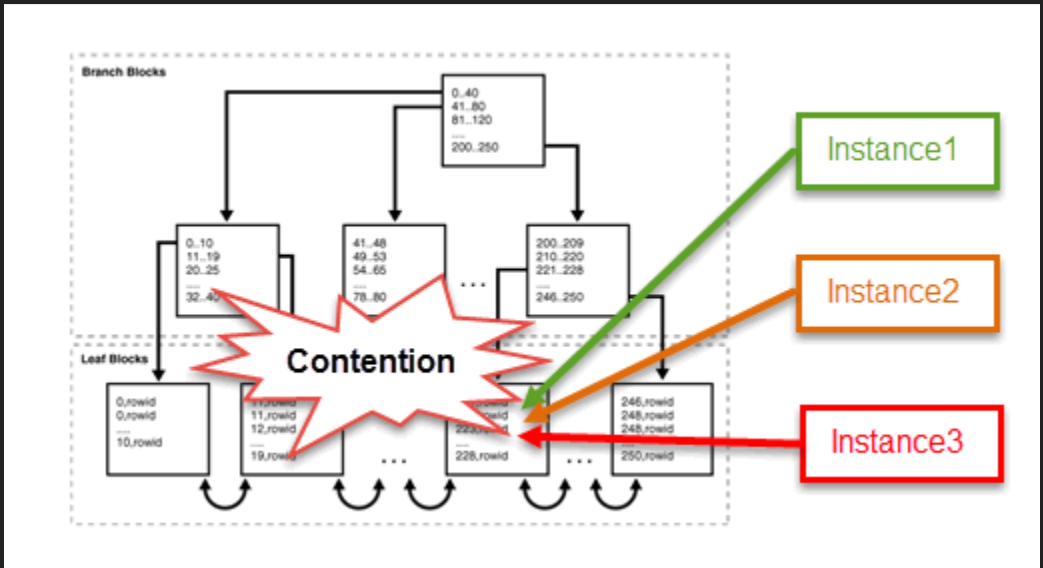
REVERSE KEY INDEX



실제로 byte를 역순으로 처리한다.

```
SELECT dump(1211101,16) FROM dual;
         -- Typ=2 Len=5: c4,2,16,c,2
         -- Reversed : 2,c,16,2,c4
```

왜 쓰는 것일까?



RAC환경에서의 Contention을 줄이기 위해 사용

INDEX KEY COMPRESSION

여러개의 컬럼을 KEY값으로 하는 Index의
중복을 제거해서 압축한다.

KEY를 prefix(중복부분)와 suffix(고유부분)로 분해

INDEX KEY COMPRESSION

- Index의 공간이 줄어든다.
- I/O 시간이 줄어든다.
- 버퍼캐시의 효율이 향상된다.

- Contention이 발생할 수 있다.
- CPU 사용량이 증가한다

현재 상황을 고려하여 INDEX를 압축한다

HIGHLIGHT

- Performance
- Data Structure
- Oracle B-Tree Structure
- Oracle B-Tree Operation
- Reverse Key Index
- Index Key Compression

REFERENCE

- Thomas Kyte.『 Expert Oracle Database 』Jpub
- 이화식.『 새로 쓴, 대용량 데이터 베이스 솔루션 』엔코아컨설팅