

9. 인덱스 튜닝

2021.04.12

박경현

- **인덱스 기본 원리**

- 책 전체를 한 장씩 찾는 대신 인덱스를 사용하면 바로 위치를 알 수 있다.

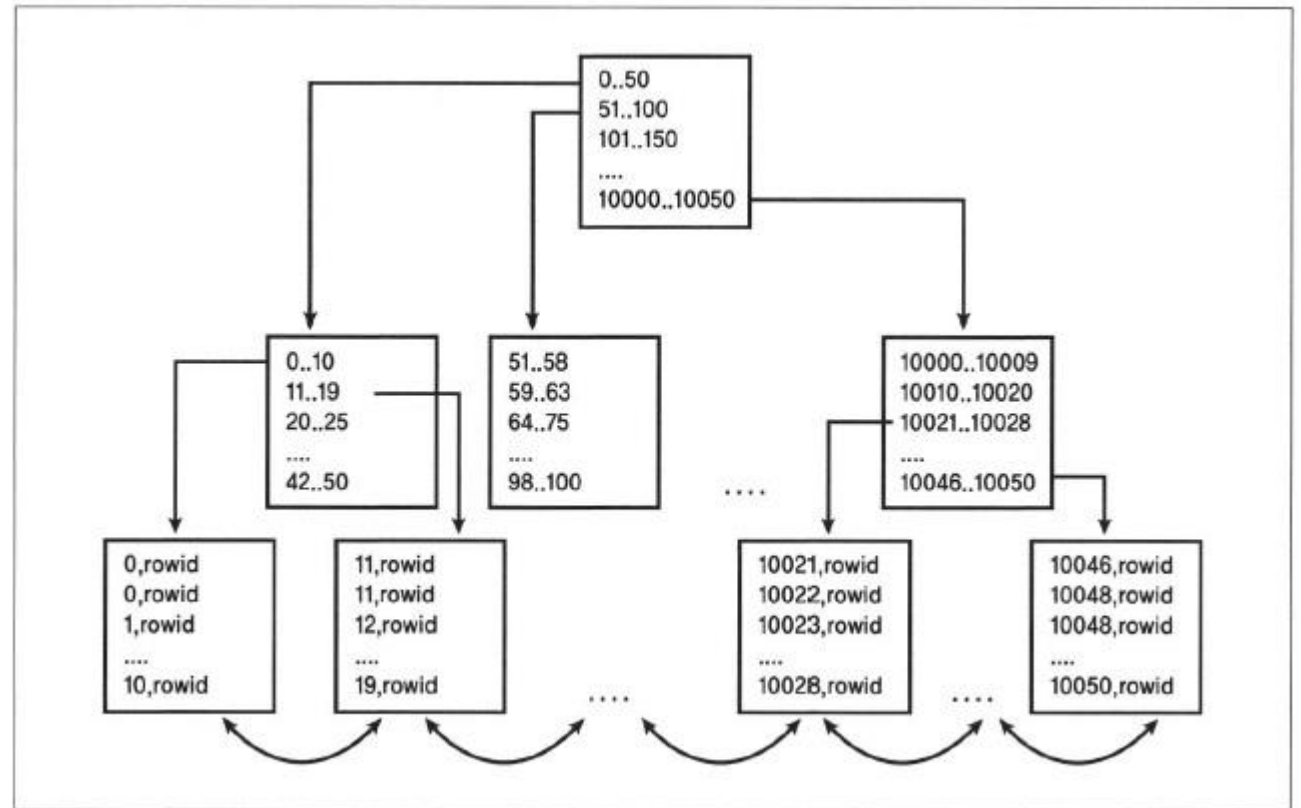
- **인덱스 구조**

- **일반적으로 B* Tree 인덱스 구조를 사용한다.**

- Root에서 시작해서 Branch를 거쳐 leaf에 연결
 - Leaf에 모든 데이터가 저장되어 있다.
 - 루트에서 리프까지 거리를 Height라 하는데 성능에 영향미침
 - 키 값이 같으면 rowid순으로 정렬된다.
 - 정방향 역방향 스캔이 가능하도록 양방향 연결 리스트구조다.

- **Null값 저장 차이**

- Oracle : 모두 null값이 인덱스는 저장X
null값은 맨 뒤에 저장
 - SQL Server : 모두 null값도 인덱스에 저장O
null값은 맨 앞에 저장



• 인덱스 기본 원리

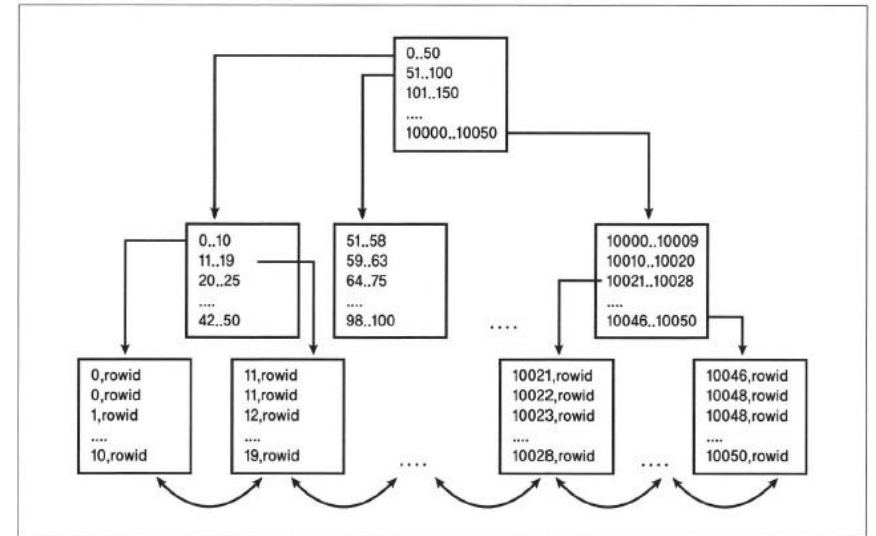
- 책 전체를 한 장씩 찾는 대신 인덱스를 사용하면 바로 위치를 알 수 있다.

• 인덱스 탐색

- 수직적 탐색
 - 루프노드에서 리프노드를 찾는 과정이 아래쪽으로 진행하기 때문에 “수직적”이라 표현
- 수평적 탐색
 - 인덱스 리프 블록에 저장된 레코드끼리 연결된 순서에 따라 좌->우, 우->좌로 스캔하기 때문에 “수평적”이라 표현

• EX) 키 값이 0번인 데이터 탐색 방법

1. 루트 블록에서 0이 들어있는 키 값 탐색 -> 첫번째 레코드 선택
2. 브랜치 블록(2)에서 0이 들어있는 키 값 탐색 -> 첫번째 레코드 선택
3. 찾아간 리프블록에서 해당하는 값을 찾으면 저장된 rowid를 이용해 테이블접근
4. 테이블에서 해당 row접근
5. 만약 원하는 정보가 아닐 경우 밑에 rowid탐색



- 인덱스 기본 원리

- 책 전체를 한 장씩 찾는 대신 인덱스를 사용하면 바로 위치를 알 수 있다.

- 인덱스 스캔 방식

- Index Range Scan

- 인덱스 루트 블록에서 수직적으로 탐색한 후에 리프블록을 필요한 범위(Range)만 스캔하는 방식

- Index Full Scan

- 수직적 탐색 없이 인덱스 리프블록을 처음부터 끝까지 수평적으로 탐색하는 방법
 - 데이터 검색을 위한 최적의 인덱스가 없을 때 차선으로 선택된다.(index는 설정 되었지만 where절에 없는 경우)

- Index Unique Scan

- 수직적 탐색만으로 데이터를 스캔하는 방식으로 Unique 인덱스를 '='조건으로 탐색하는 경우

- Index Skip Scan

- 결합 인덱스 중 하나가 빠지면 Index Full Scan을 선택, 여기서 부하를 줄이기위해 9i버전 이후 '가능성이 있는' 리프 블록을 골라서 액세스 하는 방식
 - 조건절에서 빠진 인덱스의 Distinct Value가 적고, 조건절에 포함된 Value의 개수가 많을 때 유용하다.

- 인덱스 기본 원리

- 책 전체를 한 장씩 찾는 대신 인덱스를 사용하면 바로 위치를 알 수 있다.

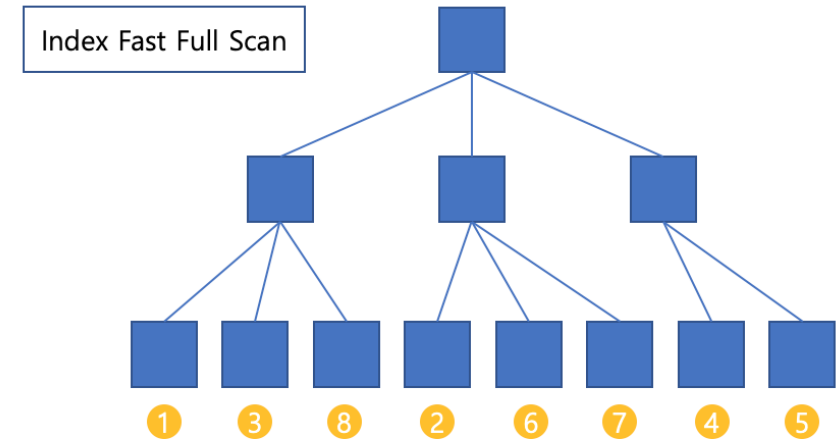
- 인덱스 스캔 방식

- Index Fast Full Scan

- 장점 : 물리적으로 저장된 위치를 읽기 때문에 Multiblock 탐색이 가능
 - 단점 : 인덱스에 포함된 칼럼으로만 조회할 때 사용가능, 결과 집합 순서 보장이 안됨.

- Index Range Scan Descending

- Index Range Scan과 동일한 스캔 방식이다
 - 뒤에서부터 앞으로 스캔하기 때문에 내림차순으로 정렬된 결과 집합을 얻는다.



- 인덱스 기본 원리

- 책 전체를 한 장씩 찾는 대신 인덱스를 사용하면 바로 위치를 알 수 있다.

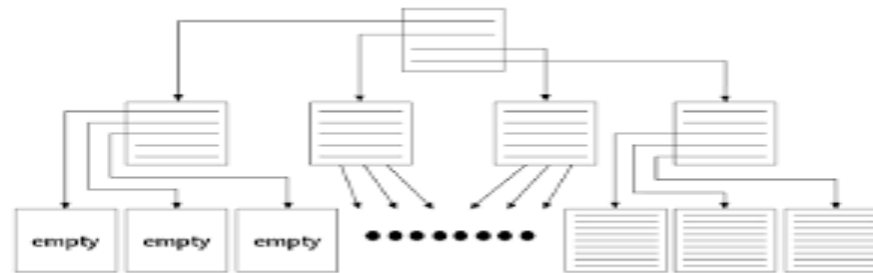
- Index의 문제점

- Unbalanced Index

- 다른 리프 노드에 비해 루트 블록과 거리가 더 멀거나 가까운 리프노드가 생길 수 있음. -> 그러나 B-tree에서는 불가능

- Index Skew

- 왼쪽 또는 오른쪽에 빈 공간이 생기는 원인
 - 무분별한 삭제로 인해 생긴다. (EX delete from t where no <= 50000)
 - SQL Server에서는 주기적으로 확인해서 정리함



[그림 Ⅲ-4-10] Index Skew

• 인덱스 기본 원리

- 책 전체를 한 장씩 찾는 대신 인덱스를 사용하면 바로 위치를 알 수 있다.

• Index의 문제점

• Index Sparse

- 인덱스 블록 전반에 걸쳐 밀도가 떨어지는 현상
- 지워진 자리에 새로운 값이 들어오면 재사용 되지만
인덱스 효율이 낮은 문제
- 지워진 자리에 새로운 값이 입력되지 않으면 영영 재사용 불가
- 인덱스 공간 사용량이 계속 커지는 현상

```
SQL> create table t as select rownum no from big_table where rownum <= 1000000 ;
SQL> create index t_idx on t(no) ;
SQL> select /*+ index(t) */ count(*) from t where no > 0;
```

```
COUNT(*)
-----
1000000
```

```
Statistics
```

```
-----
0 recursive calls
0 db block gets
2001 consistent gets ... ..
```

```
SQL> delete from t where mod(no, 10) < 5 ;
```

500000 행이 삭제되었습니다.

```
SQL> commit;
```

```
SQL> select /*+ index(t) */ count(*) from t where no > 0;
```

```
COUNT(*)
-----
500000
```

```
Statistics
```

```
-----
0 recursive calls
0 db block gets
2001 consistent gets ... ..
```

- 인덱스 기본 원리

- 책 전체를 한 장씩 찾는 대신 인덱스를 사용하면 바로 위치를 알 수 있다.

- 인덱스 튜닝 기초

- 인덱스 튜닝이 불가능한 경우

- 범위 스캔이 불가능하거나 인덱스 사용이 불가능한 경우

- 1. 인덱스를 가공한 경우(함수 사용)

```
select *  
from emp  
where substr(fname, 1, 2) = 'df';
```

- 2. 부정형 비교를 사용한 경우

```
select *  
from emp  
where fname <> 'df';
```

- 3. Is not null 조건을 사용한 경우

```
select *  
from emp  
where fname is not null
```

- Oracle에서만 인덱스 사용이 불가능 한 경우(SQL Server는 상관X)

- 1. Null값을 가진 단일행인 경우

- Oracle은 Null을 저장하지 않기 때문에 다 비교해야 한다.

```
select *  
from emp  
where fname is null
```


- 인덱스 기본 원리

- 책 전체를 한 장씩 찾는 대신 인덱스를 사용하면 바로 위치를 알 수 있다.

- 인덱스 튜닝 방법

인덱스 컬럼 가공 사례	튜닝 방안
<pre>select * from 업체 where substr(업체명,1,2) = '대한'</pre>	<pre>select * from 업체 where 업체명 like '대한%'</pre>
<pre>select * from 사원 where 월급여 * 12 = 36000000</pre>	<pre>select * from 사원 where 월급여 = 36000000/12</pre>
<pre>select * from 주문 where to_char(일시,'yyyymmdd') = :dt</pre>	<pre>select * from 주문 where 일시 >= to_date(:dt,'yyyymmdd') and 일시 < to_date(:dt,'yyyymmdd')+1</pre>
<pre>select * from 고객 where 연령 직업 = '30공무원'</pre>	<pre>select * from 고객 where 연령 = 30 and 직업 = '공무원'</pre>
<pre>select * from 회원사지점 where 회원번호 지점번호 = :str</pre>	<pre>select * from 회원사지점 where 회원번호 = substr(:str, 1, 2) and 지점번호 = substr(:str, 3, 4)</pre>

- 인덱스 기본 원리

- 책 전체를 한 장씩 찾는 대신 인덱스를 사용하면 바로 위치를 알 수 있다.

- 묵시적 형변환 피하기

- DBMS에서는 사용자의 편리를 위해 숫자형과 문자형이 만나도 똑같이 처리해준다.

```
select *  
from emp  
where empid = '101';
```

```
select *  
from emp  
where empid = 101;
```

- 여기서 empid가 숫자형일 경우, 문자형을 숫자형으로 형 변환 시켜서 Index 사용가능.
 - 여기서 empid가 문자형일 경우, 뒤에 숫자형이 들어오면 empid가 묵시적으로 숫자형으로 바뀌기 때문에 Index full Scan

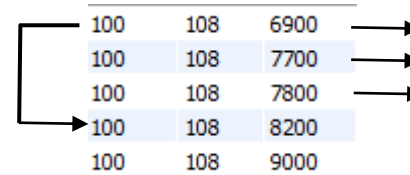
- 인덱스 스캔 효율화

- 레코드간 논리적, 물리적 순서에 따르지 않고 한 건을 읽는 방식인 랜덤 액세스를 효율적으로 사용하는 방법.

- 인덱스 선행 칼럼이 범위 조건일 때의 비효율

- 조건절에서 모두 등치 조건으로 비교되면 리프 블록을 스캔하면 읽는 레코드는 모두 테이블 액세스로 이어짐
 - Ex) Index가 deptid, mgrid, salary일 경우
 - between 조건을 맨 마지막에 접근하면 총 4차례의 접근이 발생

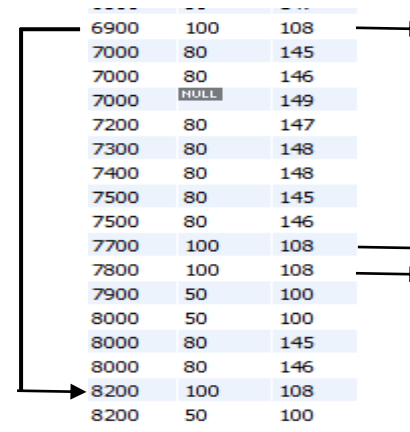
```
select *
from emp
where deptid = 100
and mgrid = 108
and salary between 5000 and 8000;
```



100	108	6900
100	108	7700
100	108	7800
100	108	8200
100	108	9000

- Between 조건을 맨 앞으로 접근하면 16차례의 접근이 발생

```
select salary, deptid, mgrid
from emp
where salary between 5000 and 8200
and deptid = 100
and mgrid = 108
```



6900	100	108
7000	80	145
7000	80	146
7000	NULL	149
7200	80	147
7300	80	148
7400	80	148
7500	80	145
7500	80	146
7700	100	108
7800	100	108
7900	50	100
8000	50	100
8000	80	145
8000	80	146
8200	100	108
8200	50	100

- 인덱스 스캔 효율화

- 레코드간 논리적, 물리적 순서에 따르지 않고 한 건을 읽는 방식인 랜덤 액세스를 효율적으로 사용하는 방법.

- 인덱스 선행 칼럼이 범위 조건일 때의 비효율 처리방법

- 범위 조건을 In-List방식으로 바꿈
 - 총 5번의 접근이 발생
 - 수직적 탐색이 많이 발생하기 때문에 Height가 높으면 비효율적

```
select salary, deptid, mgrid
from emp
where salary in (6000,6100,6200,6300, ..., 8000)
and deptid = 100
and mgrid = 108
```

6900	100	108
7000	80	145
7000	80	146
7000	NULL	149
7200	80	147
7300	80	148
7400	80	148
7500	80	145
7500	80	146
7700	100	108
7800	100	108
7900	50	100
8000	50	100
8000	80	145
8000	80	146
8200	100	108
8200	50	100

- 인덱스 스캔 효율화

- 레코드간 논리적, 물리적 순서에 따르지 않고 한 건을 읽는 방식인 랜덤 액세스를 효율적으로 사용하는 방법.

- Like와 Between 성능 비교

< 쿼리 1 >

```
SELECT COUNT(*)  
  
FROM 월별고객판매집계 T  
  
WHERE 판매월 BETWEEN '200901' AND '200902'  
  
AND 판매구분 = 'A';
```

< 쿼리 2 >

```
SELECT COUNT(*)  
  
FROM 월별고객판매집계 T  
  
WHERE 판매월 LIKE '2009%'  
  
AND 판매구분 = 'A';
```

< 쿼리 3 >

```
SELECT COUNT(*)  
  
FROM 월별고객판매집계 T  
  
WHERE 판매월 >= '200901'  
  
AND 판매월 < '200903'  
  
AND 판매구분 = 'A';
```

