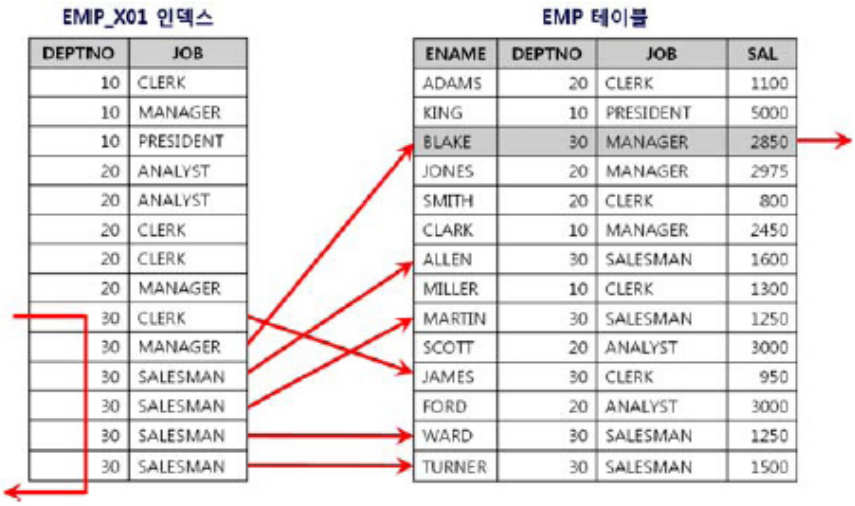


[그림 Ⅲ-4-20] 인덱스 칼럼 추가 전

출처 : http://wiki.gurubee.net

"WHERE DEPTNO = 30"
이 실행되고 인덱스를 탔을 때 조회하는 과정.

인덱스는 데이터 레코드(튜플)에 빠르게 접근하기 위해 <키, 포인터> 쌍으로 구성되는 데이터 구조입니다.

위 그림에서 인덱스의 키값은 DEPTNO, 포인터는 DEPTNO에 저장된 레코드의 물리적인 주소(JOB)입니다.
"키값인 DEPTNO가 정렬되어 있기 때문에" 인덱스를 통해 레코드를 빠르게 접근할 수 있습니다.

인덱스가 없으면 특정한 값을 찾기 위해 모든 데이터 페이지를 다 뒤져야 되는 TABLE SCAN이 발생합니다.

TABLE SCAN이란 테이블에 있는 모든 레코드를 순차적으로 읽는 것을 말합니다.
일반적으로 적용 가능한 인덱스가 없거나 분포도가 넓은 데이터를 검색할 때 TABLE SCAN을 사용합니다.

기본키를 위한 인덱스를 기본 인덱스라 하고, 기본 인덱스가 아닌 인덱스들을 보조 인덱스라고 합니다.
대부분 관계형 데이터베이스 관리 시스템에서는 모든 기본키에 대해서 자동적으로 기본 인덱스를 생성합니다.

레코드의 물리적 순서가 인덱스의 엔트리 순서와 일치하게 유지되도록 구성되는 인덱스를 클러스터 인덱스라고 합니다.

대표적인 인덱스로는 m-원 검색 트리, B-트리, B*-트리, B+-트리 등이 있습니다.

m-원 검색 트리, B-트리, B*-트리, B+-트리 대해서는 다음에 포스팅을 하도록 하겠습니다~

3. 인덱스 구조와 작동 원리(B-TREE 인덱스 기준)

- 테이블은 여러 칼럼이 있고, 데이터도 기본적인 방법으로는 들어오는 순서대로 입력됨
- 즉, 어떤 기준으로 정렬되는 것이 아니라 입력되는 순서대로 그냥 들어가서 정렬없이 저장된다(IOT 제외)
- 인덱스는 컬럼이 두개뿐!!

"인덱스"도 결국 "테이블"이다.

사원 테이블

사번	이름	주소	급여
1000	홍길동	서울	400
1001	강감찬	대전	250
1002	일지매	경기	520
1003	나한지	제주	200

IDX_사원_이름 인덱스

데이터에 대한 물리적 주소.

Key	ROWID
강감찬	AAASHOAAEAAAACXAAM
나한지	AAASHOAAEAAAACXAAN
일지매	AAASHOAAEAAAACXAAO
홍길동	AAASHOAAEAAAACXAAP

↑ "key"에 따라 정렬되어 있음.

- 테이블은 컬럼이 여러개이고 데이터도 정렬없이 입력된 순서대로 들어가 있지만, 인덱스는 컬럼이 Key 컬럼과 ROWID 컬럼 두개로 이루어져 있음

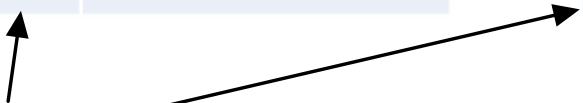
- Key 컬럼이란 인덱스를 생성하라고 지정한 컬럼 값이 됨

IDX_사원_이름 인덱스

Key	ROWID
강감찬	AAASHOAAEAAAACXAAM
나한지	AAASHOAAEAAAACXAAN
일지매	AAASHOAAEAAAACXAAO
홍길동	AAASHOAAEAAAACXAAP

IDX_사원_주소 인덱스

Key	ROWID
경기	AAASHOAAEAAAACXAAO
대전	AAASHOAAEAAAACXAAM
서울	AAASHOAAEAAAACXAAP
제주	AAASHOAAEAAAACXAAN



- 이름 컬럼과 주소 컬럼에 인덱스를 생성한 것

B-Tree 인덱스 (Balanced Tree)

B-Tree는 위에서 표현된 binary search tree의 한계를 극복하고자 나온 자료구조입니다.

DBMS에서 가장 범용적으로 사용되고 있는 자료구조이며, 파생으로 나온 B+Tree Index, B*-Tree Index 등이 있습니다.

Balanced Tree는 구조적으로는 Binary Search Tree와 비슷하지만, 데이터 높이(층)를 자동으로 바로잡아주는 기능이 있습니다.

이는 데이터의 Insert, delete등의 시간을 희생시키면서 Search 시간을 줄일 수 있습니다.

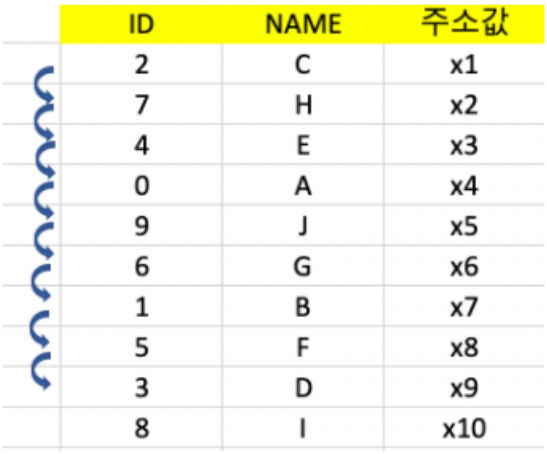
이해를 돕기 위하여, 아래 숫자와 이름이 들어가있는 테이블에 대한 B-Tree 인덱스를 만들어 보도록 하겠습니다.

아래 데이터는 데이터 파일로 저장되어있습니다.

ID	NAME	주소값
2	C	x1
7	H	x2
4	E	x3
0	A	x4
9	J	x5
6	G	x6
1	B	x7
5	F	x8
3	D	x9
8	I	x10

이 테이블에서 인덱스없이 "ID=3"를 search 한다면,

가장 윗줄 2번부터 순차적으로 총 8번의 동작을 가진 후 값을 찾을 수 있을것입니다.



	ID	NAME	주소값
	2	C	x1
	7	H	x2
	4	E	x3
	0	A	x4
	9	J	x5
	6	G	x6
	1	B	x7
	5	F	x8
	3	D	x9
	8	I	x10

이제 ID 칼럼을 기준으로하여 B-Tree 인덱스를 만들어 보도록 하겠습니다.

위에서 보이는것 처럼, "ID=3"에 대한 정보를 찾는다면, 총 3번의 동작으로 데이터를 찾을 수 있습니다.

데이터베이스에는 인덱스와 실제데이터가 따로 분류되어 저장됩니다.

- 3을 찾는다고 가정할때, Root Node로 접근하여 인덱스 레코드를 확인합니다.
- 인덱스 레코드를 확인하였을 때, 0부터 5까지는 페이지2에 저장되어 있는것을 확인할 수 있습니다.
- 이제 Branch Node로 내려가서, 페이지 2로 접근합니다.
- 페이지 2에서, 3부터 5까지는 페이지 5에 저장되어 있는것을 확인합니다.
- 이제 Leaf Node로 내려가서 페이지 5로 접근합니다.
- Leaf Node에는 다음 페이지로 향하는 길이 없으며, 디스크에 저장되어 있는 주소를 찾아갈 수 있는 주소가 있습니다.
- Leaf Node에서 "3"을 찾고 레코드 주소 x9을 활용하여
- 디스크에 있는 데이터 (ID 3, Name D)를 찾아냅니다

<인덱스 보충 설명>

- 인덱스는 테이블의 일종이다.
- 인덱스는 인덱스 키로 정렬되어 있기 때문에, 원하는 데이터를 빠르게 조회한다.
- 인덱스는 인덱스 키를 기준으로 오름차순 및 내림차순 탐색이 가능하다.
- 하나의 테이블에 여러 개의 인덱스를 생성할 수 있고, 하나의 인덱스는 여러 개의 칼럼으로 구성될 수 있다.
- 'CREATE INDEX 인덱스명 ON 테이블명(컬럼1, 컬럼2, 컬럼3, ...)' 문을 사용해서 인덱스를 생성할 수 있다.
- 인덱스 스캔 종류 : Index Unique Scan, Index Range Scan, Index Full Scan

- **Index Unique Scan** : 인덱스가 'Unique 인덱스'여야 하고, 인덱스를 구성하는 선두 컬럼이 조건절(Where 절)에서 '='로 비교되어야 한다. 인덱스에서 원하는 데이터 한 건을 찾는 순간 더 이상의 Scan을 하지 않는다.

- **Index Range Scan** : 인덱스를 구성하는 선두 컬럼에 대해 범위 검색을 하는 방법이다. 'Unique 인덱스'가 아니거나 인덱스를 구성하는 선두 컬럼이 조건절에서 비교연산(<, >, between, like)되어질 때 이 방식으로 처리된다. Index Scan에서 가장 많이 쓰이는 방식이다.

- **Index Full Scan** : 최적의 인덱스가 없을 때, 차선으로 선택되는 방식이다. 인덱스 선두 컬럼이 조건절에 없으면, Table Full Scan을 고려하나, Table Full Scan보다 I/O를 줄일 수 있거나 정렬된 결과를 쉽게 얻을 수 있을 경우 선택한다.

- Unique index

1. 중복된 값이 입력되지 않는다.
2. 테이블에도 중복된 값을 허용하지 않는다.
3. PK나 UK가 정의되면 해당 컬럼에 자동으로 만들어지는 인덱스는 Unique index이다.

- Non unique index

1. 중복된 값을 허용한다.
2. 수동으로 생성하는 인덱스의 기본이다.

- 인덱스 생성 조건

1. 테이블의 크기가 최소 수십만에서 수 천만 건 이상으로 커야한다.(일반적으로 수천 건 이내의 매우 작은 테이블에서 인덱스는 별 도움이 되지 않는다.)
2. 조건과 조인에 자주 사용되는 컬럼에 생성한다.
3. 검색 량이 테이블의 5% 미만 정도의 행을 검색하는 경우 생성한다.
4. FK컬럼에 반드시 인덱스를 생성해준다.(만약, FK이면서 동시에 PK인 컬럼은 굳이 인덱스를 생성해주지 않아도 된다.)

- 복합 인덱스

1. 두 개 이상의 컬럼으로 구성된 인덱스
2. 두 개 이상의 컬럼으로 인덱스가 만들어지면, 컬럼을 지정한 순서에 따라 정렬이 된다.
3. **인덱스로 지정한 컬럼들을 조건절에서 동시에 사용하거나 선두 컬럼만 단독적으로 조건절에 사용했을 때, 인덱스를 타게 된다.**
4. **인덱스 내 선두 컬럼이 아닌 다른 컬럼을 조건절에 사용하면, 인덱스를 타지 않는다.**
5. **그러므로 조건과 조인에 자주 사용되는 컬럼을 인덱스의 선두 컬럼으로 지정해야 한다.**

- JOIN이 수행될 때 가장 먼저 고려되는 것은 JOIN에 이용한 컬럼의 인덱스 존재 여부이다.
- JOIN은 여러 테이블을 읽는 작업이므로, 보통 이중에 한 테이블을 먼저 스캔한 후 다음 테이블을 JOIN 조건에 따라 스캔한다.
- 만약 JOIN되는 컬럼 중에 인덱스가 없는 컬럼이 있다면, 항상 인덱스가 없는 컬럼이 속한 테이블을 먼저 스캔한다.

1. Index Range Scan

- 가장 일반적인 인덱스 스캔
- root block -> leaf block을 수직적으로 탐색한 후에 필요한 leaf 블럭만 스캔
- 스캔 범위, 즉 Range를 얼마나 줄일 수 있느냐에 따라 성능이 좌우됨
- 인덱스 선두 컬럼이 조건절에 사용되어야 함
- 결과는 인덱스 컬럼 순으로 정렬된 상태이다. 따라서 order by를 생략할 수 있으며 min/max 추출에 유리하다.

2. Index Range Scan Descending

- Index Range Scan과 동일

3. Index Full Scan

- 일단 수직적 탐색을 한다. 이는 단지 가장 좌측의 leaf block을 찾기 위해서다.
- 일단 가장 좌측의 leaf block을 찾고 나면, 인덱스 leaf block을 처음부터 끝까지 수평적으로 스캔한다.
- 인덱스 선두 컬럼이 조건절에 없다면 Full Scan이 고려되는데, 만약 일부에 대해서 필터하고 나머지에 대해서만 Full Scan을 할 수 있다면 이 Index Full Scan이 사용된다.

4. Index Fast Full Scan

- Index Full Scan보다 빠른 이유는 인덱스 구조를 무시하고 인덱스 segment 전체를 multiblock read하기 때문

5. Index Unique Scan

- = 조건을 사용하는 경우
- 단, = 가 아닌 BETWEEN 조건이 들어오면 Index Range Scan 동작 (Unique 인덱스를 사용하더라도 Index Range Scan)
- Unique 결합 인덱스를 사용하는 경우, 만약 조건절에 모든 인덱스 컬럼을 다 사용하지 않으면 Index Range Scan 동작, 왜냐하면 추출 데이터가 단 1건이라는 보장이 없기 때문이다.
- Unique 인덱스는 말 그대로 중복 값이 없다. 따라서 수직적 탐색만을 수행한다.

6. Index Skip Scan

- 9i부터 가능
- 인덱스 선두 컬럼이 조건절에 빠진 경우, 그리고 선두 컬럼의 distinct 수가 적고, 후행 컬럼의 distinct 수가 많을 때 유용한다.
- 어쨌든 Full Scan 보다 I/O가 적다고 판단되어야 동작한다.