SQL Pogramming

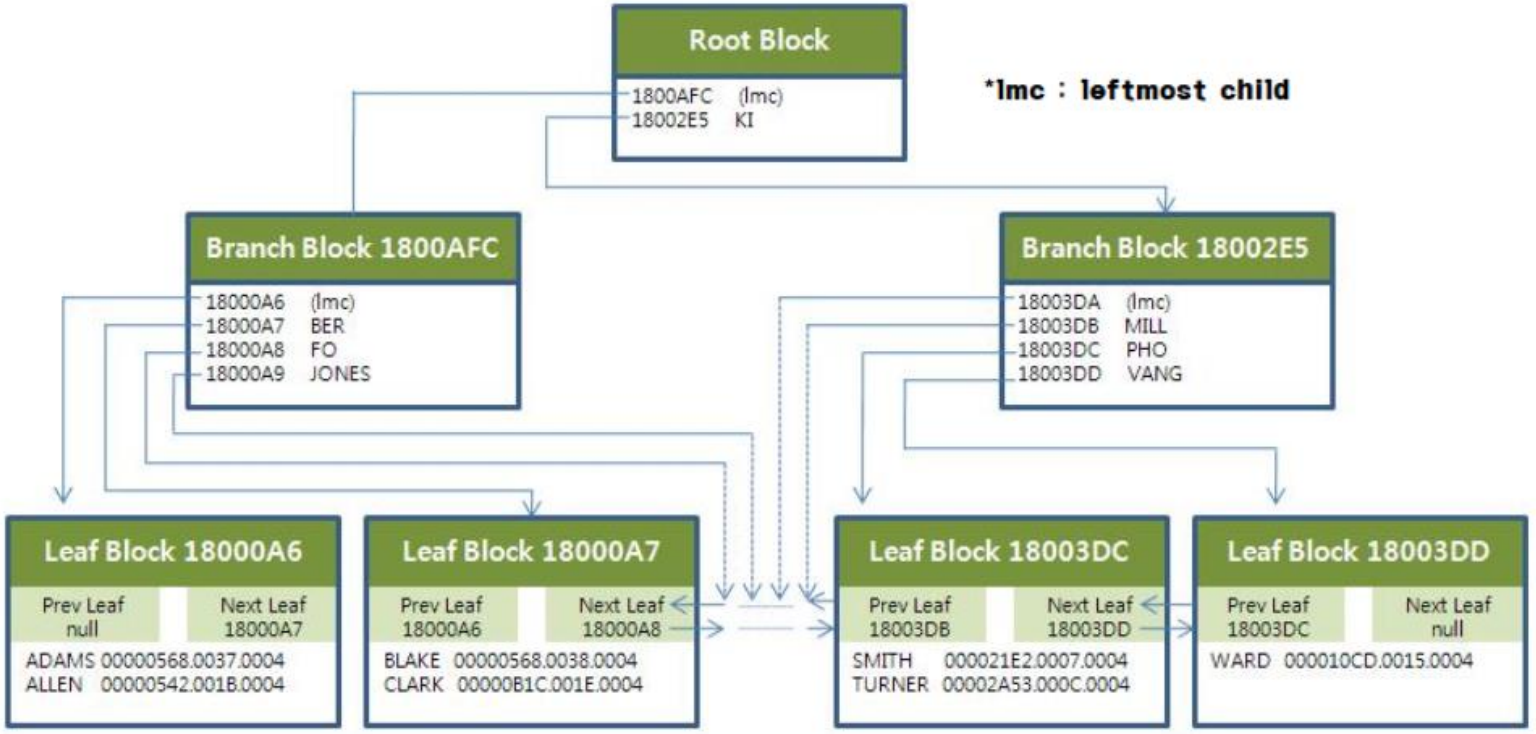
- Day 17 -

2023. 04

목차

Day 17. 인덱스 이해하기

■ B*트리 구조 (Balanced Tree 구조)



▶ 루트(브랜치) 노드는 하위 노드의 DBA(Data Block Address)를 가지며, 리프 노드는 테이블 레코드의 주소(ROWID)를 가짐

▶ 루트(브랜치) 노드 상의 레코드 개수는 브랜치(리프) 노드의 블록 개수와 일치함

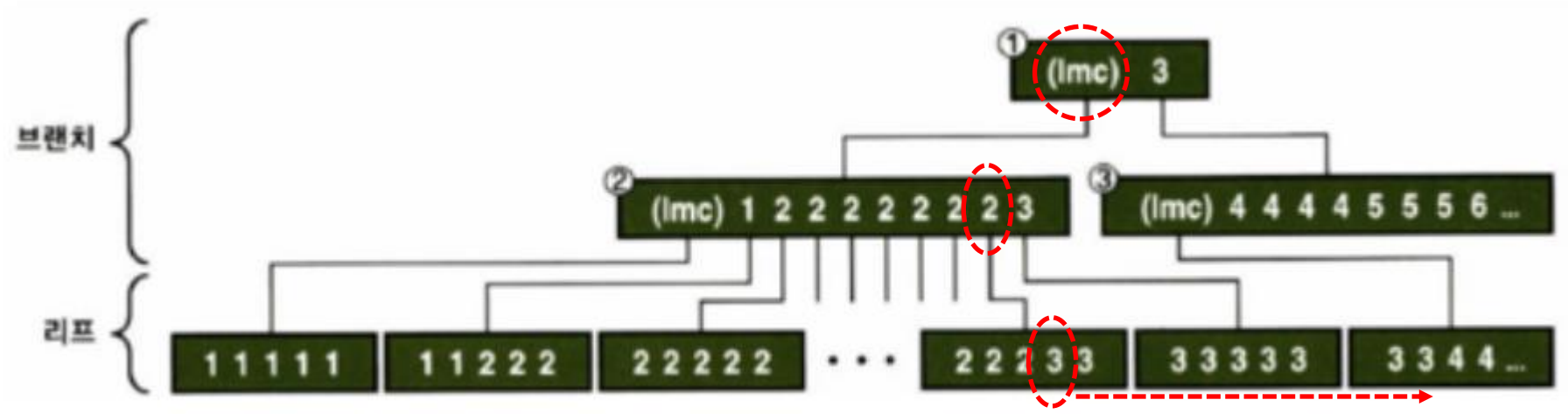
▶ 루트(브랜치) 노드 상의 키 값은 브랜치(리프) 노드가 갖는 값의 범위를 의미함 (키 값이 하위 노드의 첫번째 레코드와 정확히 일치하지 않음) → 해당 블록에 기록된 키 값보다 크거나 같고 다음 블록의 키 값보다 작거나 같은 키 값들을 가진 블록으로 정의할 수 있음

▶ 브랜치 노드 상의 레코드 개수는 리프 노드의 블록 개수와 일치함

▶ 리프 노드 상의 인덱스 레코드와 테이블 레코드 간에는 1:1 관계임 (키 값도 서로 일치함)

Day 17. 인덱스 이해하기

■ 브랜치 블록 스캔 (값이 3인 레코드 찾기)

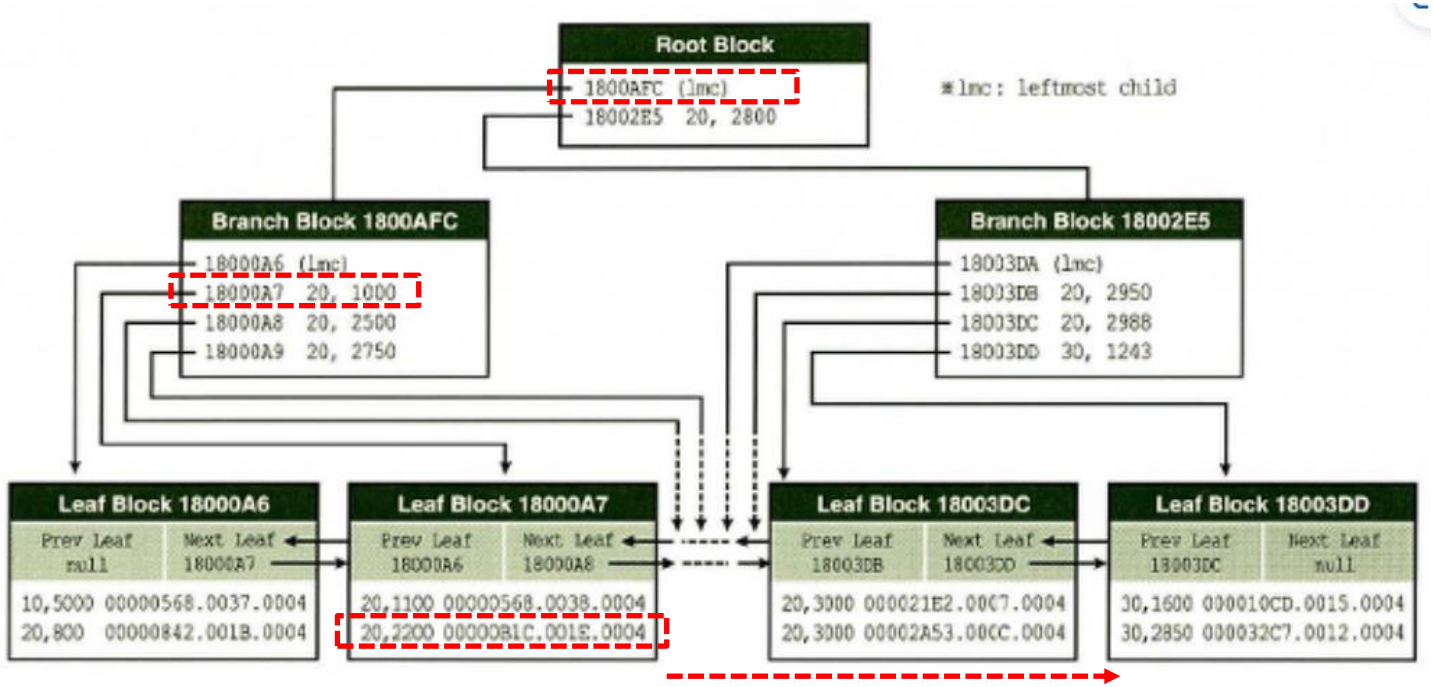


▶ ①번 루트 블록에서 (lmc)가 가리키는 ②번 브랜치 노드로 따라 내려가야 함. 만약 같은 값인 3을 따라 ③번 브랜치로 내려가면 ②번 브랜치 끝에 놓인 3을 놓치기 때문

▶ ②번 브랜치에서 맨 뒤쪽 3부터 거꾸로 스캔하다가 2를 만나는 순간 리프 블록으로 내려감

▶ 거기서 키 값이 3인 첫번째 레코드를 찾아 오른쪽으로 리프 블록을 스캔해 나가면 조회 조건을 만족하는 값을 모두 찾을 수 있음

■ 결합 인덱스 구조와 탐색



(검색 조건 : DEPT_NO = 20 & SAL >= 2,000)

▶ DEPT_NO가 20인 첫번째 레코드가 스캔 시작점이라고 생각하기 쉬운데, 두번째 리프 블록 그 중에서도 두번째 레코드부터 스캔이 시작된다는 것을 반드시 기억해야 함

▶ 수직적 탐색 과정에서 DEPT_NO뿐만 아니라 SAL 값에 대한 필터링도 같이 이루어지기 때문임

▶ 마지막 블록 첫번째 레코드에서 스캔을 멈추게 됨

Day 17. 인덱스 이해하기

■ 인덱스 설계와 SQL 튜닝의 핵심 원리

▶ 인덱스를 스캔하는 범위(Range)를 얼만큼 줄일 수 있느냐!

▶ 테이블로 액세스하는 횟수를 얼만큼 줄일 수 있느냐!

■ 인덱스 특징

▶ 테이블을 기반으로 선택적으로 생성할 수 있는 구조

▶ 테이블에 대한 검색 속도를 높이기 위해 사용되는 오브젝트

▶ 인덱스의 기본적인 목적은 검색 성능의 최적화임.
즉 검색 조건을 만족하는 데이터를 인덱스를 통하여 효과적으로 찾을 수 있도록 도움

▶ 원하는 데이터를 쉽게 찾을 수 있도록 돕는 책의 찾아보기(목차)와 유사한 개념

▶ 검색 조건에 의해 해당 테이블을 검색해야 하는 데이터는 많은데 반환되는 데이터가 적은 경우에 효과가 큼

▶ 정렬된 상태로 데이터가 저장되어 빠른 속도로 원하는 데이터에 접근할 수 있음. But, 항상 빠른 것은 아님

▶ 반대로, INSERT / UPDATE / DELETE 등과 같은 DML 작업은 테이블 뿐만 아니라 인덱스도 함께 변경해야 하기 때문에 오히려 느려질 수 있다는 단점이 존재함

■ 인덱스 스캔 설명

▶ 인덱스를 구성하는 컬럼의 값을 기반으로 데이터를 추출하는 액세스 기법

▶ 인덱스에 존재하지 않는 컬럼의 값이 필요한 경우에는 현재 읽은 레코드 식별자(ROWID)를 이용하여 테이블을 액세스 해야 함. BUT, SQL문에서 필요한 모든 컬럼이 인덱스 구성 컬럼에 포함된 경우 테이블에 대한 액세스는 발생하지 않음

▶ 인덱스는 인덱스 구성 컬럼의 순서대로 정렬됨.
인덱스 구성 컬럼이 A+B라면 먼저 컬럼A로 정렬되고 컬럼A의 값이 동일한 경우에는 컬럼B로 정렬됨.
컬럼B까지 모두 동일하면 레코드 식별자로 정렬됨.

▶ 인덱스가 구성 컬럼으로 정렬되어 있기 때문에 인덱스를 경유하여 데이터를 읽으면 그 결과 또한 정렬되어 반환됨

Day 17. 인덱스 이해하기

■ 인덱스 스캔 효율

▶ 비교 연산자 종류와 컬럼 순서에 따른 인덱스 레코드의 군집성

○ 테이블과 달리 인덱스 레코드는 '같은 값을 갖는 레코드들이 항상 서로 군집해 있음 (모여 있음) → '=' 비교가 전제

○ 인덱스 구성 컬럼들이 모두 '=' 조건으로 비교될 때는 조건을 만족하는 레코드들이 연속되게 모여 있음

○ 선행 컬럼은 모두 '=' 비교이고 맨 마지막 컬럼만 범위 검색 조건일 때도 조건을 만족하는 레코드들이 모여 있음

○ 맨 마지막 컬럼이 아닌 중간 컬럼이 범위 검색 조건일 때는 해당 컬럼까지의 인덱스 레코드는 서로 모여 있지만 그 이후의 컬럼 조건까지 만족하는 레코드는 흩어져 있음

● (**규칙**) → 선행 컬럼이 모두 '='조건인 상태에서 첫번째 나타나는 범위 검색 조건까지만 만족하는 인덱스 레코드는 모두 연속되게 모여 있지만, 그 이하 조건까지 만족하는 레코드는 비교 연산자 종류에 상관없이 흩어짐

▶ BETWEEN 조건 → IN-LIST 변경 시 인덱스 스캔 효율

○ INLIST ITERATOR 실행계획

○ CONCATENATION 실행계획 (UNION ALL 방식 변환)

○ IN-LIST 개수가 많지 않아야 유리 (인덱스 수직 탐색이 여러 번 발생함 - 브랜치 블록 반복 탐색)

▶ INDEX SKIP SCAN을 통한 비효율 해소

○ 인덱스 선두 컬럼이 누락됐을 때 뿐만 아니라 부등호, BETWEEN, LIKE 같은 범위검색 조건일 때도 유용

○ 데이터 상황에 따라서는 '='조건 컬럼들을 인덱스 선두에 위치시킨 것에 버금가는 효과를 얻을 수 있음

▶ 범위 검색 조건을 남용할 때 발생하는 비효율

○ 화면 구성 및 사용자 선택에 따라 조건절이 다양하게 바뀔 때 SQL을 간편하게 작성하려고 조건절을 모두 LIKE로 구사하는 개발자가 다수 있음

○ 물론 운영하는 데이터가 적을 경우에는 큰 차이가 없으므로 크게 상관없지만 대용량 데이터를 다루는 곳에서는 엄청난 차이를 초래하므로, LIKE를 사용했을 때 데이터베이스 내부적으로 어떤 Searching이 발생하는지 이해하고 있는 개발자와 그렇지 못한 개발자의 질적 차이는 무척 큼

▶ LIKE 검색 → BETWEEN 검색 스캔 효율

○ BETWEEN을 사용한다면 적어도 손해 볼 일은 없음 (범위 검색 조건의 스캔 시작점이 다를 수 있음)

Day 17. 인덱스 이해하기

■ 전체 테이블 스캔

- ▶ 전체 테이블 스캔(Full Table Scan) 방식으로 데이터를 검색한다는 것은 테이블에 존재하는 모든 데이터를 읽어 가면서 조건에 맞으면 결과로서 추출하고 조건에 맞지 않으면 버리는 방식으로 검색함
- ▶ 1억 건의 데이터를 보유한 테이블에 대해 전체 테이블 스캔을 해야 한다면 어마 어마한 작업량이 발생하고 결과를 얻기까지 상당한 시간이 소요될 것
- ▶ 그러나 상황에 따라 몇 천만 건의 데이터를 반드시 읽어서 레포트를 생성시켜야 한다면 인덱스를 이용하는 것 보다 훨씬 생산적인 일이 될 수도 있음 (일반적인 응용 프로그램 보다는 데이터웨어하우징 등의 업무에 해당)
- ▶ 전체 테이블 스캔은 데이터를 몇 개의 블록 단위로 한꺼번에 메모리로 퍼 올리기 때문에 유리한 상황도 다수 존재함
- ▶ SQL문에 조건이 존재하지 않는 경우
- ▶ SQL문의 조건에 사용 가능한 인덱스가 존재하지 않는 경우
- ▶ 옵티마이저의 취사 선택

■ 인덱스 스캔 유형

- ▶ INDEX UNIQUE SCAN
- ▶ INDEX RANGE SCAN
- ▶ INDEX SKIP SCAN
- ▶ INDEX FULL SCAN
- ▶ INDEX FAST FULL SCAN

Day 17. 인덱스 이해하기

■ TABLE FULL SCAN 이해하기

```
SELECT *
  FROM CM_ITEM
 WHERE ITEM_CD IN ('000570756', '000570764')
```

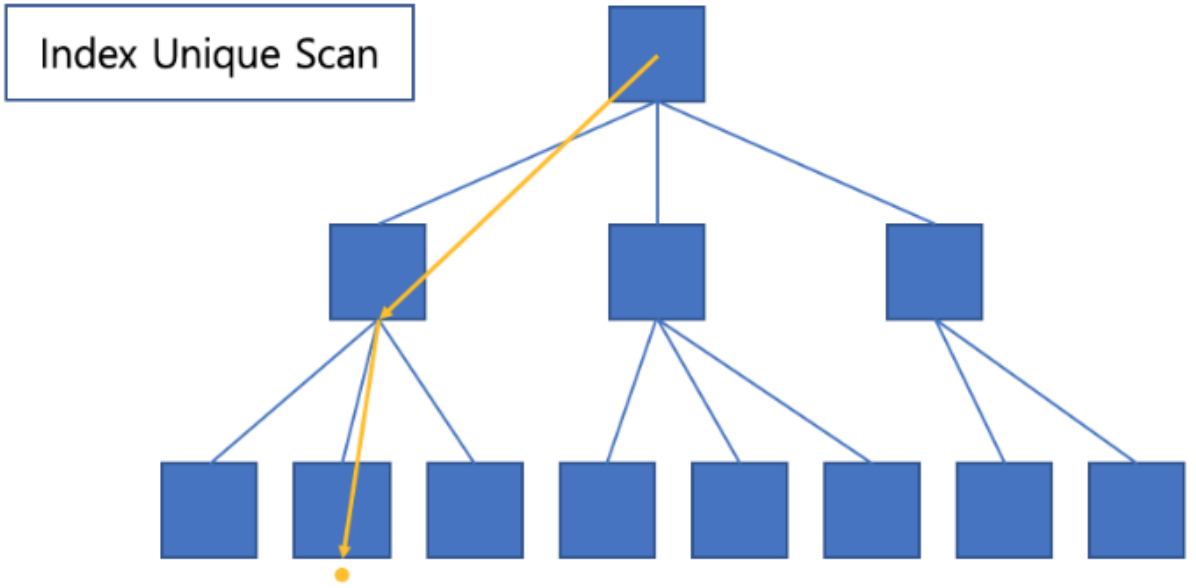
테이블 전체 데이터를 모두 읽어서 모두 버리고 2건의 데이터만 결과로 취함

TABLE			
P.Addr	상품코드	상품명	
…	…	…	X
0x0118	000570756	상품명-00138	O
0x0133	000570790	상품명-00112	X
0x0115	000570916	상품명-00199	X
0x0163	000570903	상품명-00101	X
0x0128	000570995	상품명-00122	X
0x0167	000571609	상품명-00131	X
0x0128	000572702	상품명-00105	X
0x0104	000572166	상품명-00111	X
0x0130	000570806	상품명-00195	X
0x0133	000570764	상품명-00178	O
…	…	…	X

```
SELECT *
  FROM LO_OUT_M M1
       JOIN LO_OUT_D M2 ON M2.INVOICE_NO = M1.INVOICE_NO
```

■ INDEX UNIQUE SCAN 이해하기

- ▶ INDEX UNQUE SCAN은 유일 인덱스(UNIQUE INDEX)를 사용하여 단 하나의 데이터를 추출하는 방식
- ▶ 유일 인덱스 구성 컬럼에 모두 '='로 값이 주어지는 결과는 최대 1건이 됨
- ▶ INDEX UNQUE SCAN 은 유일 인덱스 구성(복합) 컬럼에 대해 모두 '='로 값이 주어진 경우에만 가능한 인덱스 스캔 방식

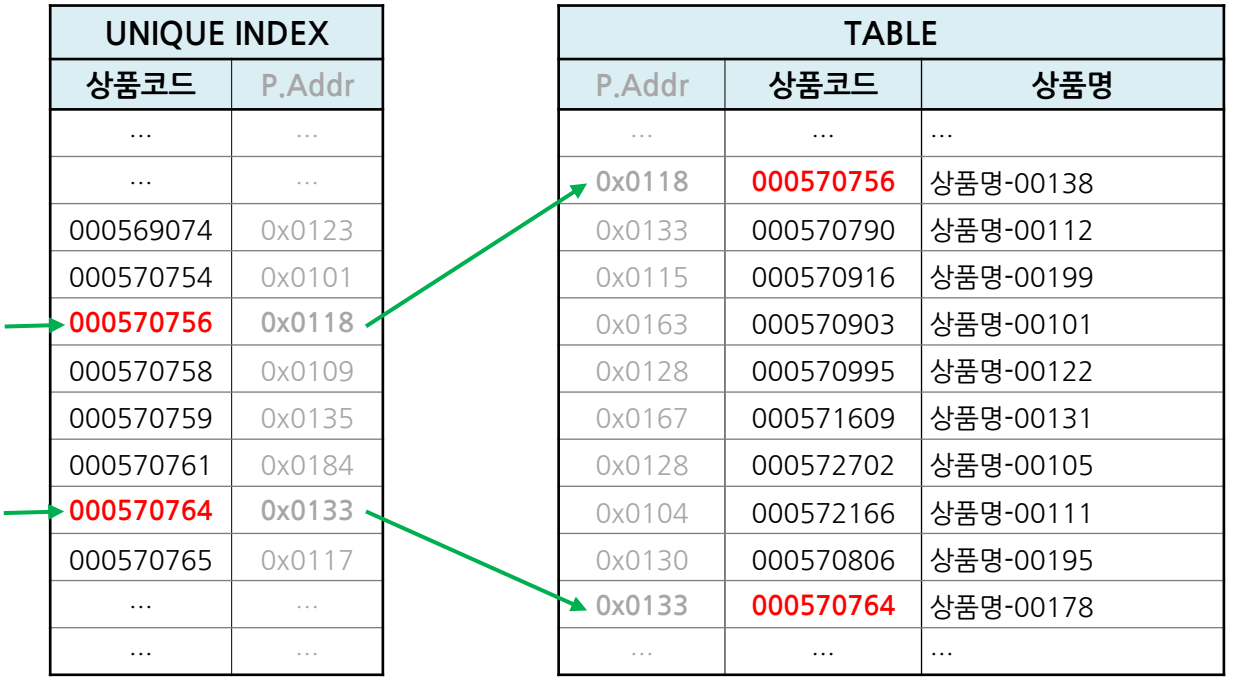


Day 17. 인덱스 이해하기

■ INDEX UNIQUE SCAN 이해하기

```
SELECT *
  FROM CM_ITEM
 WHERE ITEM_CD IN ('000570756', '000570764')
```

인덱스 2건 + 테이블 2건만 읽음
(읽어 들이는 것은 블록 단위)



UNIQUE INDEX	
상품코드	P.Addr
…	…
…	…
000569074	0x0123
000570754	0x0101
000570756	0x0118
000570758	0x0109
000570759	0x0135
000570761	0x0184
000570764	0x0133
000570765	0x0117
…	…
…	…

TABLE		
P.Addr	상품코드	상품명
…	…	…
0x0118	000570756	상품명-00138
0x0133	000570790	상품명-00112
0x0115	000570916	상품명-00199
0x0163	000570903	상품명-00101
0x0128	000570995	상품명-00122
0x0167	000571609	상품명-00131
0x0128	000572702	상품명-00105
0x0104	000572166	상품명-00111
0x0130	000570806	상품명-00195
0x0133	000570764	상품명-00178
…	…	…

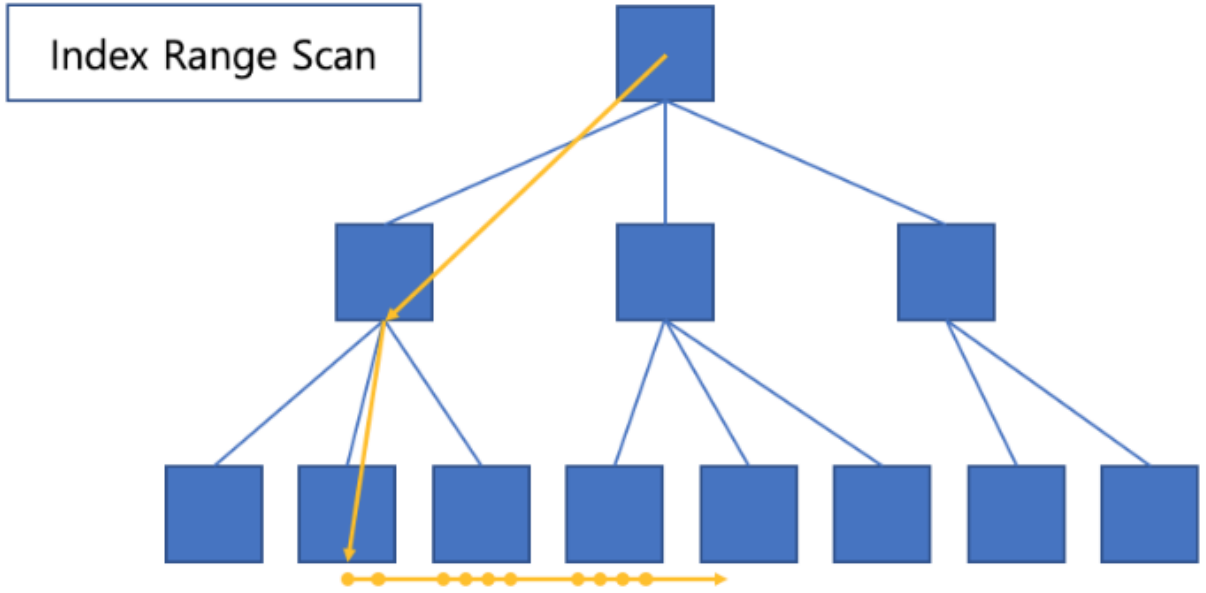
Day 17. 인덱스 이해하기

■ INDEX RANGE SCAN 이해하기

▶ 가장 일반적인 인덱스의 탐색 방식이며, 인덱스를 수직 탐색 후 필요한 범위까지만 탐색하는 방식.

▶ 인덱스의 데이터는 정렬된 상태로 저장이 되므로 범위에 대한 탐색이 필요할 때 효율적으로 데이터를 탐색할 수 있으며, 리프 블록에서 다음 리프 블록의 정보를 갖고 있어 다시 브랜치 블록을 읽을 필요없이 다음 데이터를 바로 읽을 수 있음

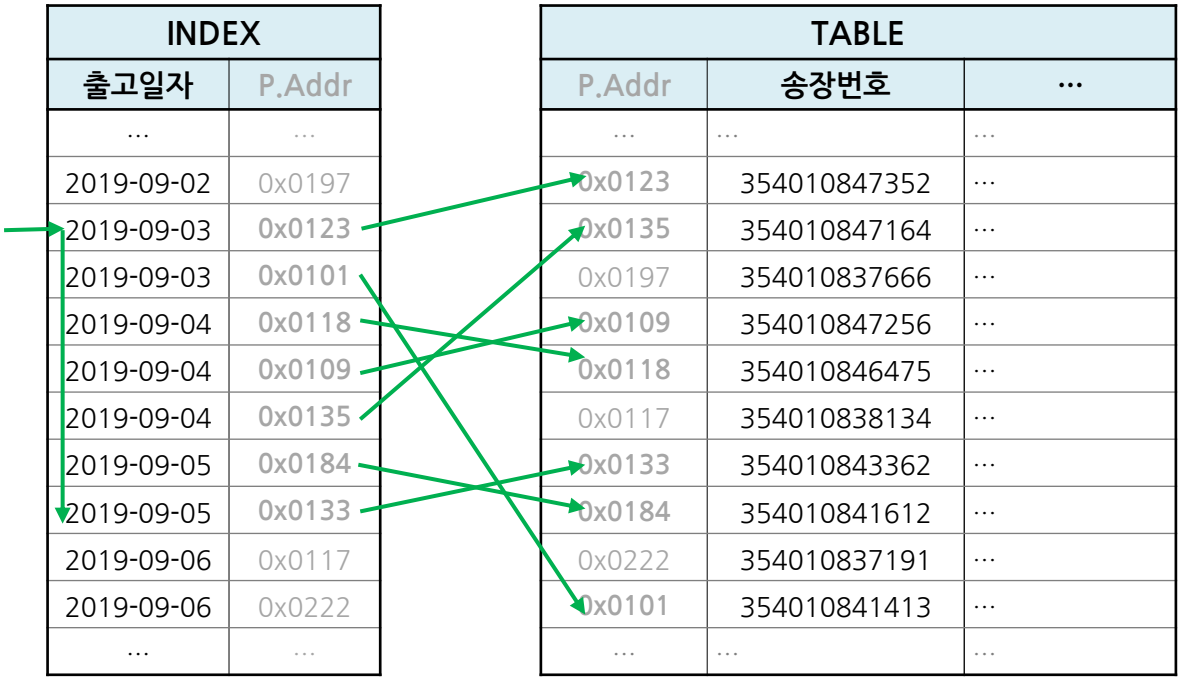
▶ 이런 특징을 잘 활용하면, 정렬 작업을 생략하거나 최대값, 최소값을 찾는데 유용하게 활용됨



Day 17. 인덱스 이해하기

■ INDEX RANGE SCAN 이해하기 → 컬럼1

```
SELECT *
  FROM LO_OUT_M
 WHERE OUTBOUND_DATE BETWEEN TO_DATE('20190903', 'YYYY-MM-DD') AND TO_DATE('20190905', 'YYYY-MM-DD ' )
```



INDEX	
출고일자	P.Addr
…	…
2019-09-02	0x0197
2019-09-03	0x0123
2019-09-03	0x0101
2019-09-04	0x0118
2019-09-04	0x0109
2019-09-04	0x0135
2019-09-05	0x0184
2019-09-05	0x0133
2019-09-06	0x0117
2019-09-06	0x0222
…	…

TABLE		
P.Addr	송장번호	…
…	…	…
0x0123	354010847352	…
0x0135	354010847164	…
0x0197	354010837666	…
0x0109	354010847256	…
0x0118	354010846475	…
0x0117	354010838134	…
0x0133	354010843362	…
0x0184	354010841612	…
0x0222	354010837191	…
0x0101	354010841413	…
…	…	…

Day 17. 인덱스 이해하기

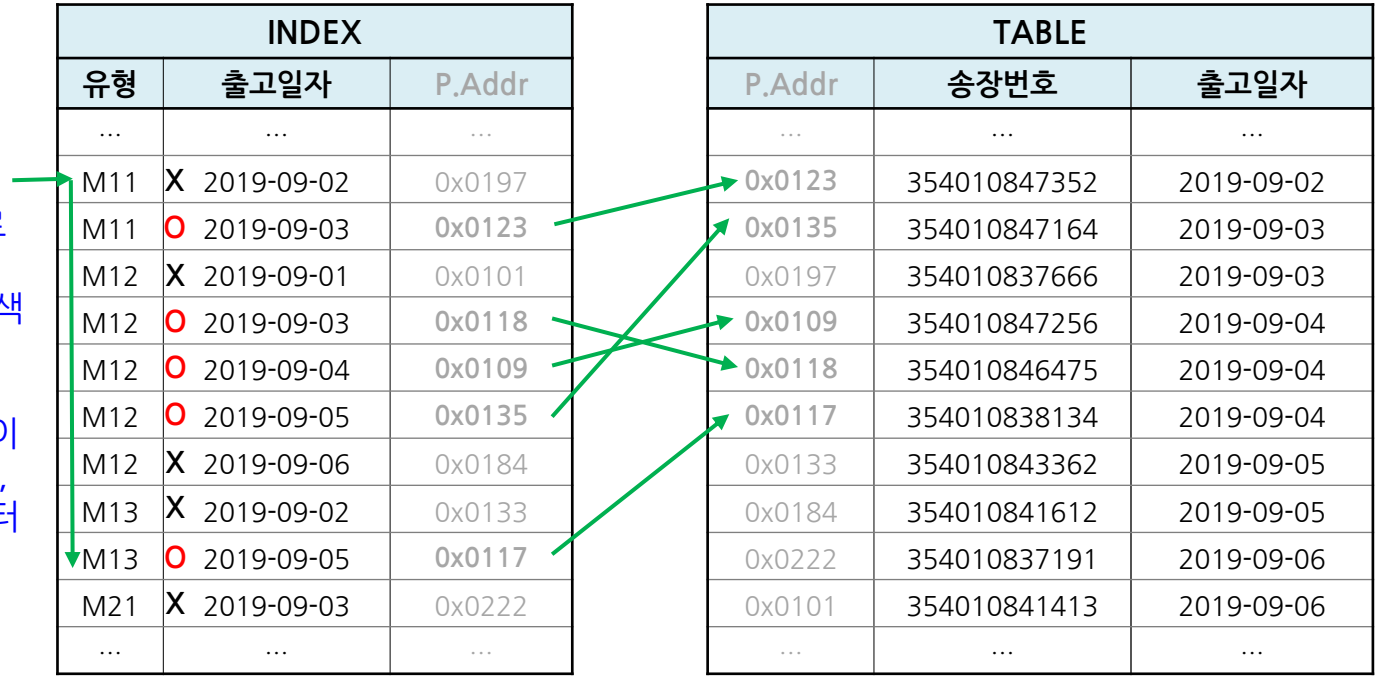
■ INDEX RANGE SCAN 이해하기 → 컬럼1 + 컬럼2

```
SELECT *
  FROM LO_OUT_M
 WHERE OUTBOUND_DATE BETWEEN TO_DATE('20190903', 'YYYY-MM-DD') AND TO_DATE('20190905', 'YYYY-MM-DD ' )
   AND OUT_TYPE_DIV  LIKE 'M1%'
```

위 SQL은 INDEX RANGE SCAN 방식으로 탐색할 것이라고 예측했지만,
실제로는 INDEX SKIP SCAN 방식으로 탐색하는 실행계획이 생성되었음.

INDEX RANGE SCAN 시, 출고일자 조건이 9/3 ~ 9/5로 한정되어 있지만 M11, M12, M13으로 시작하는 9/6 이후의 모든 데이터도 확인해야 하고 버려야 하는 데이터 량이 너무 많다고 판단한 듯!

→ 뒤에서 INDEX SKIP SCAN 방식 설명



INDEX

유형		출고일자	P.Addr
…		…	…
M11	X	2019-09-02	0x0197
M11	O	2019-09-03	0x0123
M12	X	2019-09-01	0x0101
M12	O	2019-09-03	0x0118
M12	O	2019-09-04	0x0109
M12	O	2019-09-05	0x0135
M12	X	2019-09-06	0x0184
M13	X	2019-09-02	0x0133
M13	O	2019-09-05	0x0117
M21	X	2019-09-03	0x0222
…		…	…

TABLE

P.Addr	송장번호	출고일자
…	…	…
0x0123	354010847352	2019-09-02
0x0135	354010847164	2019-09-03
0x0197	354010837666	2019-09-03
0x0109	354010847256	2019-09-04
0x0118	354010846475	2019-09-04
0x0117	354010838134	2019-09-04
0x0133	354010843362	2019-09-05
0x0184	354010841612	2019-09-05
0x0222	354010837191	2019-09-06
0x0101	354010841413	2019-09-06
…	…	…

Day 17. 인덱스 이해하기

■ INDEX SKIP SCAN 이해하기

▶ 루트 또는 브랜치 블록에서 읽은 컬럼 값 정보를 이용해 조건에 부합하는 레코드를 포함할 가능성이 있는 리프 블록만 골라서 액세스하는 방식

▶ Multi Column Index에서 후행 컬럼을 조건절에 사용할 때 활용할 수 있음

▶ 성별처럼 선두 컬럼의 고유 값의 개수가 적고 후행 컬럼의 고유 값이 많을 때 효과적임

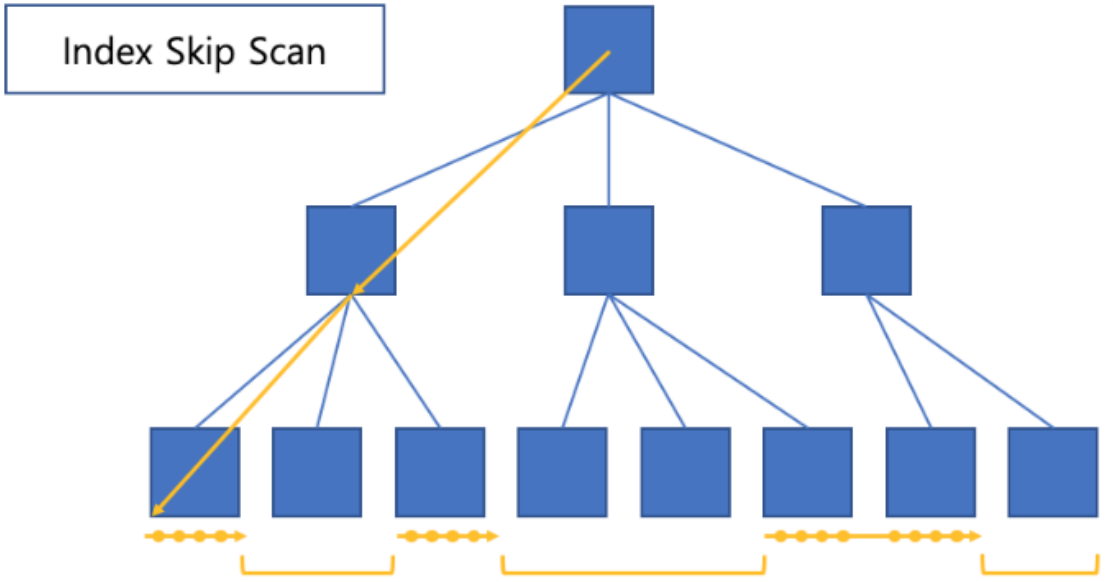
ex) 부서 → 10 / 20 / 30 / 40, 직원수 → 500 / 100 / 200 / 200 (총 1,000), 인덱스 → 부서번호 + 급여

▶ 검색하고 싶은 조건이 급여가 5,000~7,000인 직원이라고 하면, INDEX SKIP SCAN을 사용한다고 했을 때 먼저 10번 부서에서 5,000~7,000인 직원을 찾고 이후의 10번 부서의 데이터는 확인할 필요가 없기 때문에 SKIP하여 20번 부서로 넘어가서 똑같이 반복함. 이런 식으로 필요 없는 부분을 SKIP하게 되면 FULL TABLE SCAN보다 성능이 좋을 수 있음

▶ 가장 좌측에 있는 리프 블록은 항상 방문해야 함

▶ 가장 우측에 있는 리프 블록도 항상 방문해야 함

▶ 첫번째 리프 블록을 방문한 후에 다른 리프 블록으로 곧바로 점프하는 것처럼 묘사했지만, 리프 블록에 있는 정보만으로 다음에 방문해야 할 블록을 찾을 방법은 없음 → 항상 그 위쪽에 있는 브랜치 블록을 재방문해서 다음 방문할 리프 블록에 대한 주소 정보를 얻어야 함 → 버퍼 Pinning 기법 활용 (NL조인 연상)
→ 추가적인 블록I/O는 발생하지 않음



Day 17. 인덱스 이해하기

■ INDEX SKIP SCAN 이해하기

```
SELECT *
  FROM LO_OUT_M
 WHERE OUTBOUND_DATE BETWEEN TO_DATE('20190903', 'YYYY-MM-DD') AND TO_DATE('20190905', 'YYYY-MM-DD')
   AND OUT_TYPE_DIV  LIKE 'M1%'
```

■ INDEX_SS이 작동하기 위한 조건

- ▶ Distinct Value 개수가 적은 선두 컬럼이 조건절에서 누락됐고 후행 컬럼의 Distinct Value 개수가 많을 때 효과적
- ▶ 3개 컬럼으로 구성된 인덱스에서 중간 컬럼에 대한 조건절이 누락된 경우
- ▶ 3개 컬럼으로 구성된 인덱스에서 두 개의 선두 컬럼이 누락된 경우
- ▶ 선두 컬럼이 부등호, BETWEEN, LIKE 같은 범위 검색 조건일 때도 사용

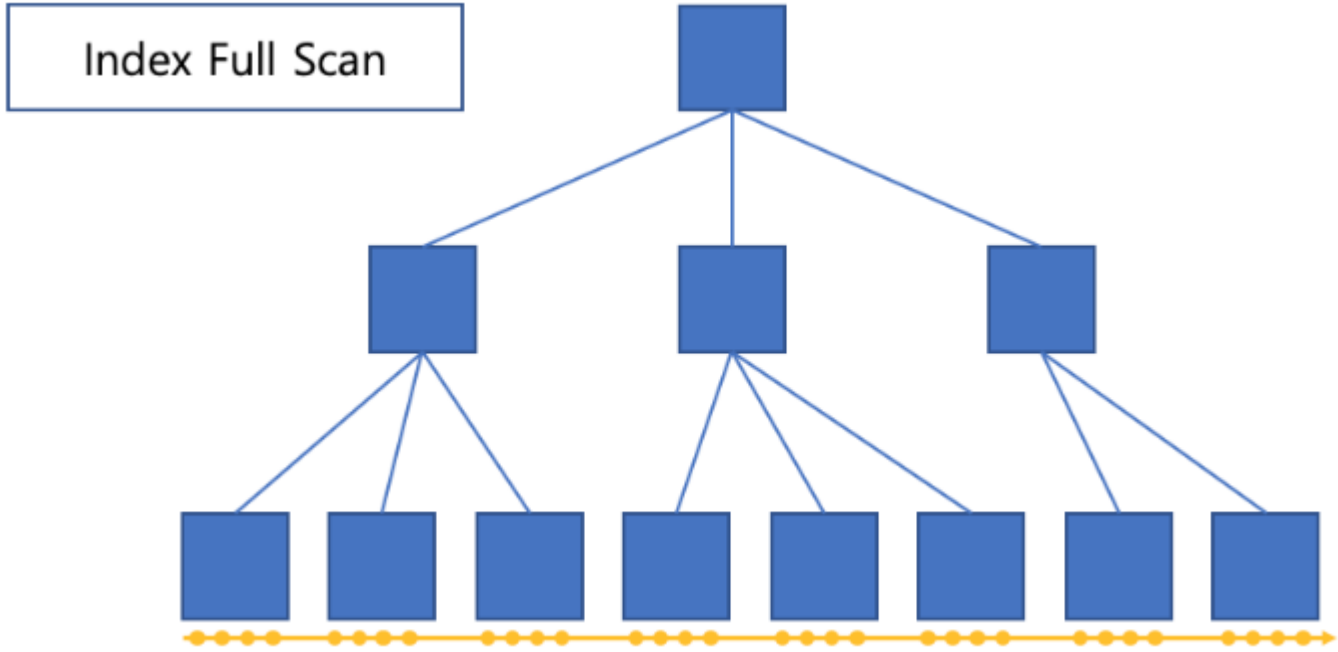
INDEX		
유형	출고일자	P.Addr
…	…	…
M11	2019-09-02	0x0197
M11	O 2019-09-03	0x0123
M12	2019-09-01	0x0101
M12	O 2019-09-03	0x0118
M12	O 2019-09-04	0x0109
M12	O 2019-09-05	0x0135
M12	2019-09-06	0x0184
M13	2019-09-02	0x0133
M13	O 2019-09-05	0x0117
M21	2019-09-03	0x0222
…	…	…

TABLE		
P.Addr	송장번호	출고일자
…	…	…
0x0123	354010847352	2019-09-02
0x0135	354010847164	2019-09-03
0x0197	354010837666	2019-09-03
0x0109	354010847256	2019-09-04
0x0118	354010846475	2019-09-04
0x0117	354010838134	2019-09-04
0x0133	354010843362	2019-09-05
0x0184	354010841612	2019-09-05
0x0222	354010837191	2019-09-06
0x0101	354010841413	2019-09-06
…	…	…

Day 17. 인덱스 이해하기

■ INDEX FULL SCAN 이해하기

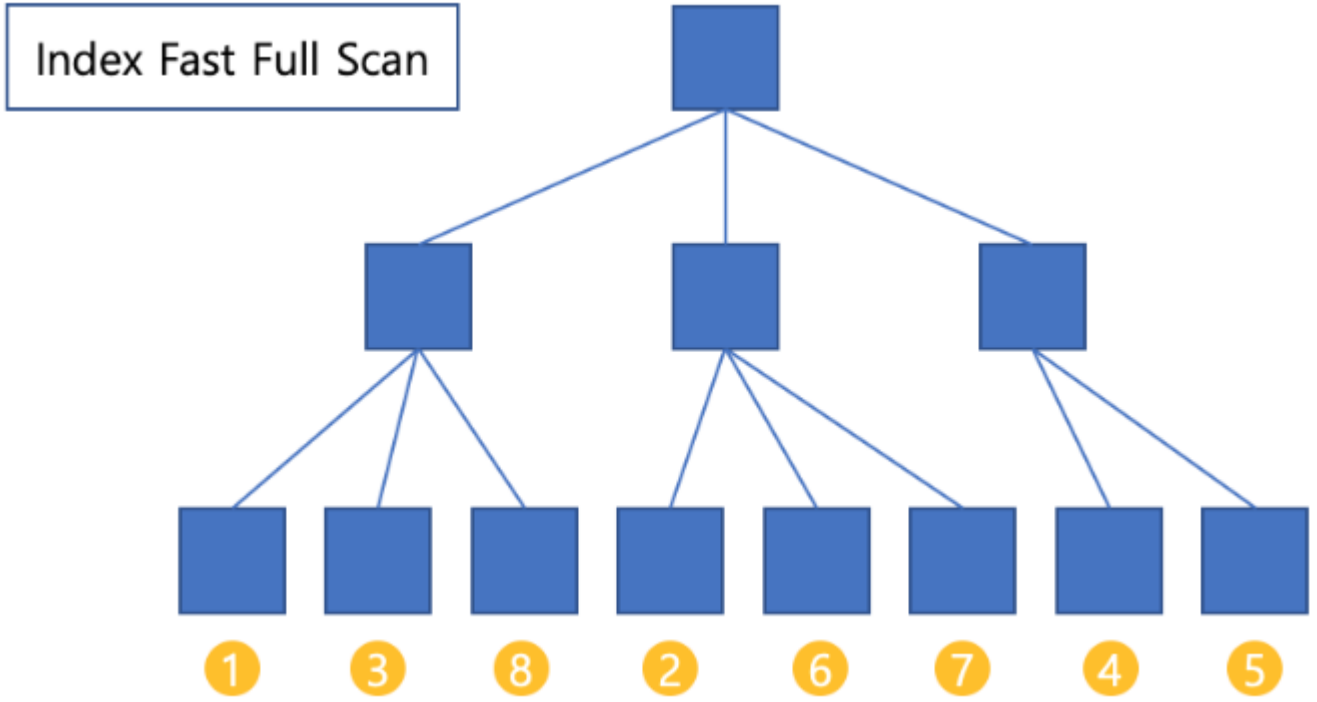
- ▶ 첫 번째 리프 블록까지 수직적 탐색 후 인덱스 전체를 탐색하는 방법
- ▶ FULL TABLE SCAN의 부담이 크거나 정렬 작업을 생략하기 위해 테이블 전체를 탐색하는 것 보다 인덱스를 사용하는 것이 유리함
- ▶ 이 방법은 WHERE 조건에 부합하는 인덱스가 생성되지 않아서 사용된 방법으로, INDEX RANGE SCAN으로 유도하는 것이 좋음
- ▶ FULL TABLE SCAN과의 가장 큰 차이점은 FULL TABLE SCAN이 Multi Block Scan을 한다는 점이며, INDEX FULL SCAN은 리프 블록을 일일이 방문하기 때문에 Single Block Read를 할 수 밖에 없음
  따라서, 때로는 FULL TABLE SCAN의 성능이 더 좋을 때가 있음



Day 17. 인덱스 이해하기

■ INDEX FAST FULL SCAN 이해하기

- ▶ INDEX FULL SCAN보다 빠름
- ▶ 인덱스 데이터 전체를 탐색하면서 더 빠르게 결과를 얻을 수 있는 이유는, Multi Block Read가 가능하기 때문임
- ▶ 하지만 이로 인해 정렬된 상태로 데이터를 받아 올 수 없다는 제한이 있음
- ▶ INDEX FULL SCAN은 데이터의 논리적인 순서로 결과를 얻지만 INDEX FAST FULL SCAN은 데이터의 물리적인 저장 구조의 순서대로 받아오므로 Multi Block Read가 가능함



Day 17. 인덱스 이해하기

■ 인덱스 사용에 대한 결론

▶ 인덱스는 기본적으로 최적의 INDEX RANGE SCAN을 목표로 설계해야 하며, 수행 횟수가 적은 SQL을 위해 인덱스를 추가 하는 것이 비효율적 일 때 나머지 스캔 방식을 차선책으로 활용하는 전략이 바람직함

Day 17. 인덱스 이해하기

■ 결합 인덱스의 효율 비교하기 (출고일자 + 출고차수 + 출고유형구분)

출고일자	출고차수	출고유형구분
…	…	…
9/1	O 001	O M21
9/1	O 001	O M21
9/1	O 001	X M22
9/1	X 002	M21
…	…	…
9/2	O 001	O M21
9/2	O 001	O M21
9/2	O 001	X M22
9/2	X 002	M21
…	…	…
9/3	O 001	O M21
9/3	O 001	O M21
9/3	O 001	X M22
9/3	X 002	M21
…	…	…
…	…	…
9/30	O 001	O M21
9/30	O 001	O M21
9/30	O 001	X M22
9/30	X 002	M21
…	…	…

```
SELECT *
  FROM LO_OUT_M
 WHERE OUTBOUND_DATE  BETWEEN '2019-09-01' AND '2019-09-30'
   AND OUTBOUND_BATCH = '001'
   AND OUT_TYPE_DIV   = 'M21'
```

- [테이블 전체] 건수 → 2,823,592
- [9/1 ~ 9/30] 건수 → 160,383
- [9/1 ~ 9/30] & [001] 건수 → 217
- [9/1 ~ 9/30] & [001] & [M21] 건수 → 213

Day 17. 인덱스 이해하기

■ 결합 인덱스의 효율 비교하기 (출고차수 + 출고유형구분 + 출고일자)

출고차수	출고유형 구분	출고일자
001	M11	…
001	M11	9/1
001	M11	9/2
001	M11	9/2
001	M11	…
001	M11	9/30
001	M11	9/30
001	…	…
001	M21	…
001	M21	9/1
001	M21	9/2
001	M21	9/2
001	M21	…
001	M21	9/30
001	M21	9/30
…	…	…
002	M11	…
002	M11	9/1
002	M11	9/2
002	M11	9/2
002	M11	…
…	…	…

```
SELECT *
  FROM LO_OUT_M
 WHERE OUTBOUND_DATE  BETWEEN '2019-09-01' AND '2019-09-30'
   AND OUTBOUND_BATCH = '001'
   AND OUT_TYPE_DIV   = 'M21'
```

- [테이블 전체] 건수 → 2,823,592
- [001] & [M21] & [9/1 ~ 9/30] 건수 → 213

Day 17. 인덱스 이해하기

실전문제① ▶ 실행속도 향상시키기				
《테이블》	■ LO_OUT_M(출고주문)	■ LO_OUT_D(출고주문상세)		
《조건》	■ OUTBOUND_DATE(출고일자) ■ OUTBOUND_BATCH(출고차수) ■ ORDER_QTY(출고수량)	▶ 2019년 1월 1일 ~ 2019년 9월 30일 ▶ 046 ▶ 100 이상		
《정렬》	■			
《특징》	■ 실행속도를 향상시킬 수 있는 방안 제시하기			

결과 ▼ 총 건수 : 1건

SUM_QTY
1556

실전문제② ▶ 실행시간 체크 및 실행속도 향상시키기				
《테이블》	■ LO_OUT_D(출고주문상세)	■ CS_CODE(상용코드)		
《조건》	■ CODE_GRP(그룹코드)	▶ LDIV03		
《정렬》	■			
《특징》	■ LO_OUT_D 테이블 전체 데이터에서 OUT_TYPE_DIV_D 컬럼에 존재하는 유니크한 값을 출력하기 ■ 현재 490만 건 정도의 데이터가 존재 → 1억 건 이상이 존재해도 가장 빨리 결과를 도출할 수 있는 SQL 작성하기 ■ 먼저, 해결 방안부터 제시하기			

결과 ▼ 총 건수 : 12(13)건

CODE_CD
M11
M12
M13
M14
M15
M17
M21
M22
M23
M24

실전문제③ ▶ 성능 좋은 SQL 작성하기				
《테이블》	■ LO_OUT_D(출고주문상세)	■ CS_CDOE(상용코드)		
《조건》	■ CODE_GRP	▶ LDIV03		
《정렬》	■			
《특징》	■ 화면 상단의 콤보박스에 출고 테이블상의 OUT_TYPE_DIV_D 값을 모두 표시하기 위한 SQL 작성하기 ■ 명칭도 함께 표시하기			

결과 ▼ 총 건수 : 12(13)건

OUT_TYPE_DIV_D	CODE_NM
M11	[상온]기획
M12	[상온]DPS
M13	[상온]DAS
M14	[상온]개별박스
M15	[상온]원상품
M17	[상온]대량 기획
M21	[저온]기획
M22	[저온]DPS

Thank you !

ASETEC Location http://www.asetec.co.kr

본사. 경기도 성남시 분당구 성남대로 331번길 8, 킨스타워 2201호 TEL.031.609.7000 FAX.031.609.7009
부산. 부산광역시 해운대구 센텀동로 99 TEL.051.506.6352 FAX.051.504.8794