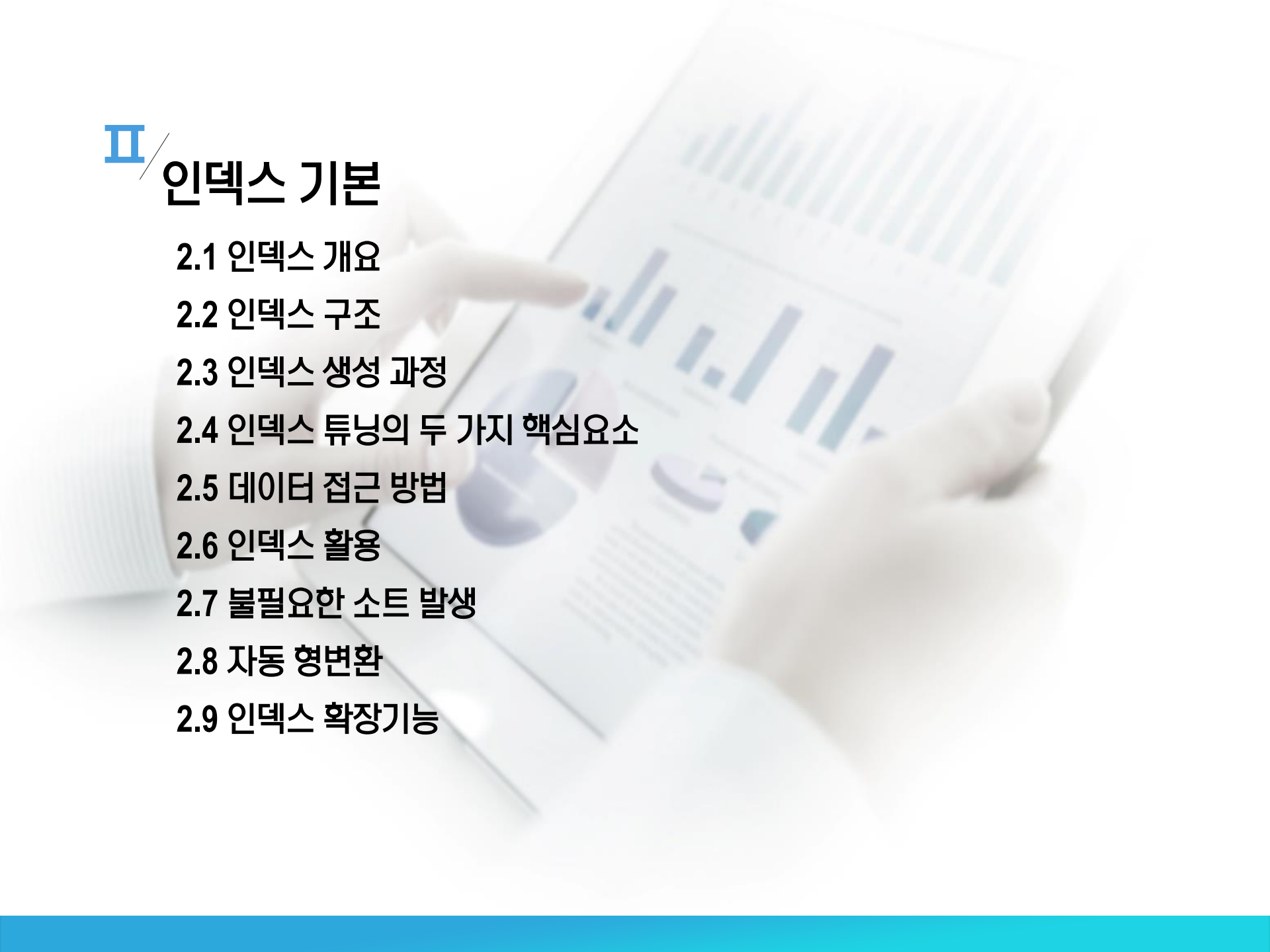
TIBERO 튜닝 기초교육

(2. 인덱스 기본)

YOON



II 인덱스 기본

2.1 인덱스 개요

2.2 인덱스 구조

2.3 인덱스 생성 과정

2.4 인덱스 튜닝의 두 가지 핵심요소

2.5 데이터 접근 방법

2.6 인덱스 활용

2.7 불필요한 소트 발생

2.8 자동 형변환

2.9 인덱스 확장기능

2.1 인덱스 개요

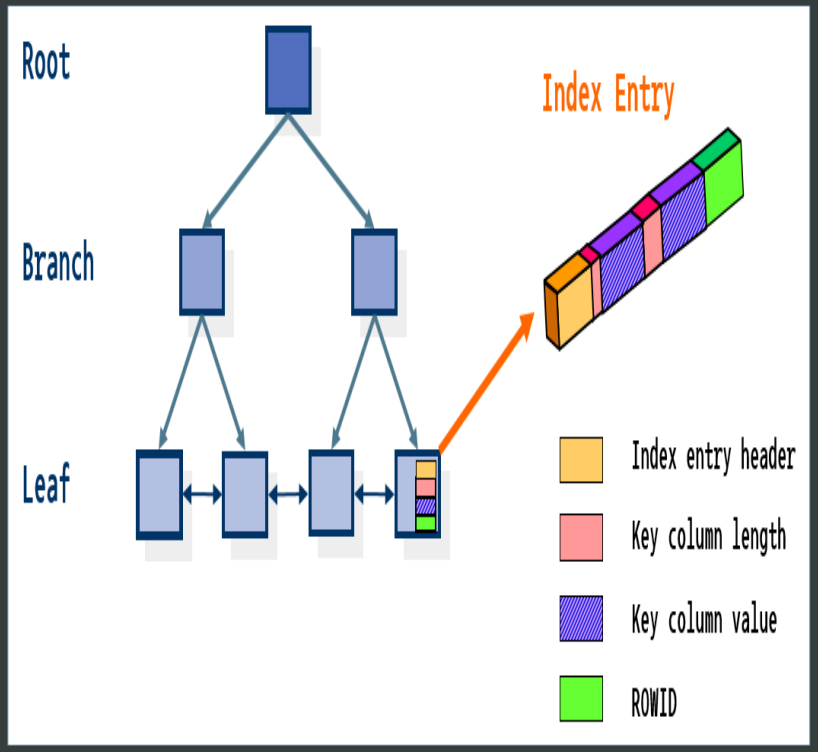
개요

- 데이터베이스 객체이며 저장 공간이 필요
  (오라클은 NOSEGMENT 사용 가능)
- 책의 색인(인덱스)이 정보를 빨리 찾을 수 있도록 도와주는 것과 마찬가지로 인덱스는 테이블 데이터에 대한 빠른 액세스 경로를 제공
- 튜닝의 가장 중요한 포인트
  (적절한 인덱스 선택, 인덱스 변경, 생성, 삭제...)
- 옵티마이저가 실행계획을 만드는 가장 중요한 판단 기준 (통계정보)
- 튜닝 시 가장 비용(COST)이 적게드는 방법(?)

- 저장 형태 : B*Tree
- 종류
  ① NORMAL
  ② FBI(Function Based Index)
  ③ IOT(Index Organized Table)
- 주요 속성
  ① 연결된 (다중 열) 인덱스에서 중복 항목을 제거한 선택적 압축(티베로 미지원 ?)
  - Index Compress(Index Key Compression)
  ② 역방향을 통한 특정 블록 경합 해소 가능
  ③ 오름 차순, 내림 차순
- 기타 인덱스

2.2 인덱스 구조

B*Tree



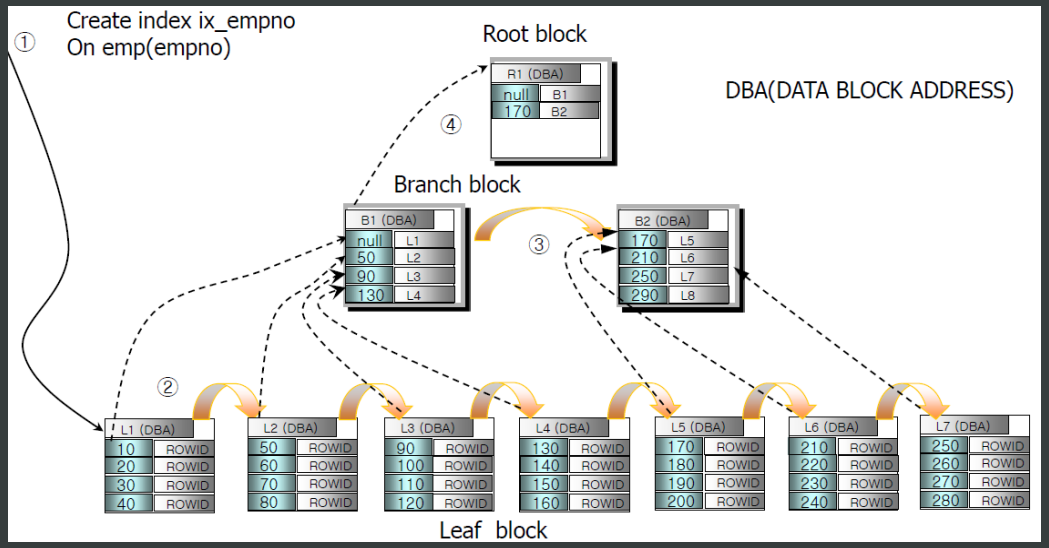
BLevel : 2 / Depth : 3

- 가장 범용적으로 사용되는 인덱스 자료구조
- Balance Tree 자료구조에 의해 Root Block ↔ Leaf Block의 Bleve / Depth가 동일 (한쪽에 치우쳐진 상태가 아닌 항상 일정하게 유지)
- Leaf Block의 값은 정렬되어 있음
  (Leaf 간에는 Double Linked List로 되어있음)
- 최하위 블록은 인덱스의 모든 값 및 연관된 데이터 세그먼트에 있는 행을 가리키는 ROWID를 포함

2.3 인덱스 생성 과정

B*Tree

Temp Segment
10
20
30
40
50
60
70
80
90
100
110
120
130
…
280



PCTFREE

1. 데이터 블록 Row 데이터의 길이가 늘어날 것을 대비하기 위한 여유 공간 설정 값 (기본 값 : 10%)
2. 이미 데이터 블록에 들어있는 데이터의 수정이 발생했을 때 일어날 수 있는 Row Migration 문제를 방지하기 위함

TIBERO

- _INDEX_PCTFREE : 5 (?)
- _TD_DEFAULT_PCTFREE : 10

① Temp Segment 데이터 정렬

② 인덱스 세그먼트의 Leaf Block에 기록, Leaf Block이 PCTFREE에 도달

③ 새로운 Leaf Block과 Branch Block 생성 / Leaf Block이 추가될 때마다 Branch Block에 Row를 추가 작업 계속 (Branch Block이 PCTFREE에 도달), 새로운 Branch Block 생성

④ Root Block 생성

2.4 인덱스 튜닝의 두 가지 핵심요소

1. 인덱스 스캔 효율화 튜닝

이름	시력	학년-반-번호
강수지	1.5	4학년 3반 37번
김철수	0.5	3학년 2반 13번
...	...	...
이영희	1.5	6학년 4반 19번
...	...	...
홍길동	1.0	2학년 6반 24번
홍길동	1.5	5학년 1반 16번
홍길동	2.0	1학년 5반 15번
...	...	...

시력	이름	학년-반-번호
0.5	김철수	3학년 2반 13번
...	...	...
1.0	홍길동	5학년 1반 16번
1.5	강수지	4학년 3반 37번
1.5	이영희	6학년 4반 19번
1.5	홍길동	1학년 5반 15번
1.5	...	...
2.0	홍길동	2학년 6반 24번
...	...	...

- 학생명부에서 시력이 1.0 ~ 1.5인 홍길동 학생을 찾는 경우의 효율성 비교
- 인덱스 구성 및 조건 컬럼 차이("=, Between" vs "Between, =")에 따른 인덱스 스캔 비교

2.4 인덱스 튜닝의 두 가지 핵심요소

2. 랜덤 액세스 최소화

이름	시력	학년-반-번호
강수지	1.5	4학년 3반 37번
김철수	0.5	3학년 2반 13번
...	...	...
이영희	1.5	6학년 4반 19번
...	...	...
홍길동	1.0	2학년 6반 24번
홍길동	1.5	5학년 1반 16번
홍길동	2.0	1학년 5반 15번
...	...	...

시력	이름	학년-반-번호
0.5	김철수	3학년 2반 13번
...	...	...
1.0	홍길동	5학년 1반 16번
1.5	강수지	4학년 3반 37번
1.5	이영희	6학년 4반 19번
1.5	홍길동	1학년 5반 15번
1.5	...	...
2.0	홍길동	2학년 6반 24번
...	...	...

- 학생명부에서 시력이 1.0 ~ 1.5인 홍길동 학생을 찾는 경우의 효율성 비교
- 인덱스 컬럼 구성의 효율성(선택도, NDV) 및 교실 방문 회수 차이 비교

2.5 데이터 접근 방법

테이블과 인덱스

구분	액세스 경로
테이블	• Full Table Scan(FTS) • Rowid Scan • Sample Table Scan (Oracle, Tibero 실행계획 차이)
인덱스	• Index Unique Scan • Index Range Scan(Desc) • Index Full Scan • Index Fast Full Scan • Index Skip Scan

- 테이블을 통한 데이터 접근은 테이블 전체를 스캔하는 것을 의미함
- 인덱스를 통한 데이터 접근은 Leaf Block에서 스캔 시작점을 찾아 거기서부터 스탠하다가 중간에 멈출는 것을 의미함(Range Scan)
- 스캔의 시작점을 찾을 수 없고 멈출 수도 없는 경우는 Leaf Block 전체를 스캔해야 함(Index Range Scan을 하지 못하고 Index Full Scan으로 처리해야함)

2.6 인덱스 활용

1. 소트 연산 생략

장비번호	변경일자	변경순번
...	...	...
B	20180505	031583
C	20180316	000001
C	20180316	000002
C	...	...
C	20180316	131576
C	20180316	131577
C	20180428	000001
C	20180428	000002
...	...	...

인덱스 구성

```
SELECT *
FROM   상태변경이력
WHERE  장비번호 = 'C'
  AND  변경일자 = '20180316'

Execution Plan
----------------------------------------------------------
0      SELECT STATEMENT Optimizer=ALL_ROWS (Cost=85 Card=81 Bytes=5K)
1    0   TABLE ACCESS (BY INDEX ROWID) OF '상태변경이력' (TABLE) (Cost=85 ···)
2    1     INDEX (RANGE SCAN) OF '상태변경이력_PK' (INDEX (UNIQUE)) (Cost=3 ···)
```

```
SELECT *
FROM   상태변경이력
WHERE  장비번호 = 'C'
  AND  변경일자 = '20180316'
ORDER BY 변경순번

Execution Plan
----------------------------------------------------------
0      SELECT STATEMENT Optimizer=ALL_ROWS (Cost=85 Card=81 Bytes=5K)
1    0   TABLE ACCESS (BY INDEX ROWID) OF '상태변경이력' (TABLE) (Cost=85 ···)
2    1     INDEX (RANGE SCAN) OF '상태변경이력_PK' (INDEX (UNIQUE)) (Cost=3 ···)
```

```
SELECT *
FROM   상태변경이력
WHERE  장비번호 = 'C'
  AND  변경일자 = '20180316'
ORDER BY 변경순번 DESC

Execution Plan
----------------------------------------------------------
0      SELECT STATEMENT Optimizer=ALL_ROWS (Cost=85 Card=81 Bytes=5K)
1    0   TABLE ACCESS (BY INDEX ROWID) OF '상태변경이력' (TABLE) (Cost=85 ··· )
2    1     INDEX (RANGE SCAN DESCENDING) OF '상태변경이력_PK' (INDEX (UNIQUE))
```

2.6 인덱스 활용

1. 소트 연산 생략 (계속)

- 인덱스는 항상 정렬 상태를 유지하므로 order by, group by를 위한 소트 연산을 생략할 수 있음.

- 상황에 따라 조건절에 사용되지 않은 컬럼이더라도 소트 연산을 대체할 목적으로 인덱스 구성에 포함시킴으로써 성능개선을 도모할 수 있음

- 주요 규칙
  ① 인덱스를 이용해 소트 연산을 대체하려면 인덱스 컬럼 구성과 같은 순서로 누락 없이 (뒤쪽 컬럼이 누락되는 것은 상관없음) Order By절에 기술해 주어야 함
  ② 단, 인덱스 구성 컬럼이 조건절에서 '=' 연산자로 비교된다면, 그 컬럼은 Order By절에서 누락되거나 인덱스와 다른 순서로 기술하더라도 상관 없음
  ③ Order By 절에 기술한 순서가 인덱스 순서와 일치하기만 한다면 조건절에서 어떤 연산자(부정형, 범위형)로 비교되더라도 정렬작업이 생략될 수 있고 그리고 이런 규칙은 Group By절에도 똑같이 적용됨

2.6 인덱스 활용

1. 소트 연산 생략 (계속)

```
create table t nologging as
select rownum a, rownum b, rownum c, rownum d, rownum e
  from dual
connect by level <= 100000;

create index t_idx on t(a, b, c, d);
```

아래 쿼리들의 수행된 실행 계획을 보면 SORT가 없음

```
select * from t where a = 1           order by a, b, c;
select * from t where a = 1 and b = 1 order by c, d;
select * from t where a = 1 and c = 1 order by b, d;
select * from t where a = 1 and b = 1 order by a, b, c, d;
```

```
-------------------------------------------------------------
| ID  | Operation             | Name  | Cards   | Rows    |
-------------------------------------------------------------
|   1 | TABLE ACCESS (ROWID)  | T     |       1 |       1 |
|   2 |  FILTER               |       |       1 |       1 |
|   3 |   INDEX (RANGE SCAN)  | T_IDX |       1 |       1 |
-------------------------------------------------------------
```

또한, 아래처럼 Order By절에 기술한 순서가 인덱스 순서와 일치하기만 한다면 조건절에서 어떤 연산자로 비교되더라도 정렬작업이 생략될 수 있음
(옵티마이저가 항상 그런 선택을 한다는 뜻은 아닙니다.)

```
-------------------------------------------------------------------
    INDEX RANGE SCAN(A)    |       INDEX FULL SCAN(B)
-------------------------------------------------------------------
 select * from t           | select /*+ index(t, t_idx) */ *
  where a between 1 and 2  |   from t
    and b not in (1, 2)    |  where b between 2 and 3
    and c between 2 and 3  |  order by a, b, c, d;
  order by a, b, c, d;     |
                           |
 select * from t           |
  where a between 1 and 2  |
    and c between 2 and 3  |
  order by a, b, c;        |
                           |
 select * from t           |
  where a between 1 and 2  |
    and b <> 3             |
  order by a, b, c;        |
-------------------------------------------------------------------
```

2.6 인덱스 활용

1. 소트 연산 생략 (계속)

- 아래의 SQL은 인덱스로 소트오퍼레이션 대체 불가한 경우의 예제

- Order By에 기술된 컬럼의 순서가 인덱스 순서와 맞지 않거나, 조건절의 인덱스 컬럼이 ORDER BY절에 생략된 경우임

```
select * from t where a = 1 order by c, b, a; -- order by b, c

select * from t where a = 1 order by c; -- order by b, c

select * from t where a = 1 and b between 2 and 3 order by c, d;  -- order by b, c

select * from t where a = 1 and b between 1 and 2 order by a, c, b; -- order by b, c
```

```
-------------------------------------------------------------------------------------------------------------
| ID  | Operation              | Name  | Cost (%CPU)  | Cards   | Rows    | Elaps. Time  | CR Gets  | Starts   |
-------------------------------------------------------------------------------------------------------------
|   1 | ORDER BY (SORT)        |       |    4     (0)|       1 |       1 |00:00:00.0000 |       0  |       1  |
|   2 |  TABLE ACCESS (ROWID)  | T     |    4     (0)|       1 |       1 |00:00:00.0000 |       1  |       1  |
|   3 |   INDEX (RANGE SCAN)   | T_IDX |    3     (0)|       1 |       1 |00:00:00.0000 |       3  |       1  |
-------------------------------------------------------------------------------------------------------------
```

2.6 인덱스 활용

2. 오라클 기준의 트레이스 예제

인덱스 구성 - author, title

[Order By 사용안함]

```
select * from book
 where author = 'zrjthoiqpy';
```

10053 Trace 결과

****

Order-by elimination (OBYE)

****

OBYE: OBYE bypassed: no order by to eliminate.

인덱스 구성 - author, title

[Order By 사용]

```
select * from book
 where author = 'zrjthoiqpy'
 order by title ;
```

10053 Trace 결과

****

Order-by elimination (OBYE)

****

OBYE: OBYE performed.

2.7 불필요한 소트 발생

1. ORDER BY 절에서 컬럼 가공

```
select /*+ index(h(cust_no, chg_dt)) */
       h.cust_no, h.chg_dt, h.odue_amt
  from cust_odue_hist  h
 where cust_no in (1, 2)
 order by cust_no || chg_dt
```

[Oracle]

```
Execution Plan
----------------------------------------------------------
SELECT STATEMENT Optimizer=ALL_ROWS
  SORT (ORDER BY)
    INLIST ITERATOR
      TABLE ACCESS (BY INDEX ROWID) OF 'CUST_ODUE_HIST' (TABLE)
        INDEX (RANGE SCAN) OF 'CUST_ODUE_HIST_IDX01' (INDEX)
```

[Tibero]

```
Execution Plan
----------------------------------------------------------
STATEMENTS
  ORDER BY (SORT)
    TABLE ACCESS (ROWID) OF 'CUST_ODUE_HIST' (TABLE)
      INLIST ITERATOR
        INDEX (RANGE SCAN) OF 'CUST_ODUE_HIST_IDX01' (INDEX)
```

2.7 불필요한 소트 발생

1. ORDER BY 절에서 컬럼 가공 (계속)

```
select to_char(ord_no, 'fm0000') as ord_no
     , ord_qty
     , ord_amt
  from orders o
 where ord_dt = '20220509'
   and ord_no >= 10
 order by ord_no;
```

① order by ord_no ▶ order by o.ord_no

from 절에 사용된 테이블 Alias를 이용하여 order by에 기술하는 방안

② (select list) as ord_no ▶ fm_ord_no

select list에 사용된 최종 컬럼명을 원래 컬럼과 다른 이름으로 기술하는 방안

[Oracle]

```
Execution Plan
----------------------------------------------------------
SELECT STATEMENT Optimizer=ALL_ROWS
  SORT (ORDER BY)
    TABLE ACCESS (BY INDEX ROWID) OF 'ORDERS' (TABLE)
      INDEX (RANGE SCAN) OF 'ORDERS_IX1' (INDEX)
----------------------------------------------------------
```

[Tibero]

```
Execution Plan
----------------------------------------------------------
STATEMENTS
  ORDER BY (SORT) (Cost=3 Card=20)
    TABLE ACCESS (ROWID) OF 'ORDERS' (TABLE)
      INDEX (RANGE SCAN) OF 'ORDERS_IX1' (INDEX)
----------------------------------------------------------
```

2.7 불필요한 소트 발생

2. SELECT LIST에서 컬럼 가공

[컬럼 가공 없음]

```
select /*+ index(h(cust_no, chg_dt)) */ min(chg_dt)
  from cust_odue_hist  h
 where cust_no = 7369;
```

```
-------------------------------------------------------------------------------------------
| ID  | Operation                           | Name                | Rows    | CR Gets  |
-------------------------------------------------------------------------------------------
|   1 | COLUMN PROJECTION                   |                     |       1 |       0  |
|   2 |  SORT AGGR                          |                     |       1 |       0  |
|   3 |   COUNT (STOP NODE) (STOP LIMIT 2)  |                     |       1 |       0  |
|   4 |    INDEX (RANGE SCAN)               | CUST_ODUE_HIST_IDX01 |     295 |       6  |
-------------------------------------------------------------------------------------------
```

[컬럼 가공]

```
select /*+ index(h(cust_no, chg_dt)) */ min(to_date(chg_dt))
  from cust_odue_hist  h
 where cust_no = 7369;
```

```
-----------------------------------------------------------------------------
| ID  | Operation             | Name                | Rows    | CR Gets  |
-----------------------------------------------------------------------------
|   1 | COLUMN PROJECTION     |                     |       1 |       0  |
|   2 |  SORT AGGR            |                     |       1 |       0  |
|   3 |   INDEX (RANGE SCAN)  | CUST_ODUE_HIST_IDX01 |  100000 |     344  |
-----------------------------------------------------------------------------
```

2.8 자동 형변환

1. 변환 규칙

- 숫자형과 문자형이 만나면 숫자형 컬럼 기준으로 문자형 컬럼을 변환
- 날짜형과 문자형이 만나면 날짜형 컬럼 기준으로 문자형 컬럼을 변환
- LIKE 연산자 사용시에는 문자형 컬럼 기준으로 숫자형 컬럼이 변환됨
- 자동 형변환이 작동되면 편리할 수도 있겠지만 이 기능으로 인해 성능과 애플리케이션 품질에 종종 문제가 생김

- 인덱스의 성능과 관련해서 자동 형변환의 기능에 의존하지 말고, 인덱스 컬럼 기준으로 반대편 컬럼 또는 값을 정확히 형변환해 주어야 함
- 일부 개발자는 형변환 함수를 생략함으로 성능개선을 할 수 있다고 잘못알고 있지만 SQL의 성능은 I/O 블록을 줄임으로서 결정됨
- 형변환 함수를 생략해도 옵티마이저가 자동으로 생성

```
Predicate Information (identified by operation id):
---------------------------------------------------

   1 - filter(TO_NUMBER("V_DEPTNO")=20)
```

2.8 자동 형변환

1. 변환 규칙 (계속 - 조건절 상수 / 타입 불일치)

[Oracle]

```
-------------------------------------------------------------------------------
| Id  | Operation         | Name   | Starts | A-Rows |   A-Time   | Buffers |
-------------------------------------------------------------------------------
|   0 | SELECT STATEMENT  |        |      1 |     30 |00:00:00.01 |      19 |
|*  1 |  TABLE ACCESS FULL| ORDERS |      1 |     30 |00:00:00.01 |      19 |
-------------------------------------------------------------------------------

Predicate Information (identified by operation id):
---------------------------------------------------
   1 - filter(TO_NUMBER("ORD_DT")=20220509)
```

```
select /*+ index(o(ord_dt)) */
       ord_no
     , ord_qty
     , ord_amt
  from orders o
 where ord_dt = 20220509;
```

[Tibero]

```
-----------------------------------------------------------------------------------------------
| ID  | Operation             | Name       | Rows    | Elaps. Time  | CR Gets  | Starts   |
-----------------------------------------------------------------------------------------------
|   1 | TABLE ACCESS (ROWID)  | ORDERS     |      30 |00:00:00.0001 |        1 |        1 |
|   2 |  FILTER               |            |      30 |00:00:00.0013 |        0 |        1 |
|   3 |   INDEX (FULL)        | ORDERS_IX1 |    4500 |00:00:00.0002 |       18 |        1 |
-----------------------------------------------------------------------------------------------

Predicate Information
-----------------------------------------------------------------------------------------------
    2 - filter: ("O"."ORD_DT" = 20220509) (0.005)
```

2.8 자동 형변환

1. 변환 규칙 (계속 - 조건절 변수 / 타입 불일치)

[Oracle]

```
-------------------------------------------------------------------------------
| Id  | Operation         | Name   | Starts | A-Rows |   A-Time   | Buffers |
-------------------------------------------------------------------------------
|   0 | SELECT STATEMENT  |        |      1 |     30 |00:00:00.01 |      19 |
|*  1 |  TABLE ACCESS FULL| ORDERS |      1 |     30 |00:00:00.01 |      19 |
-------------------------------------------------------------------------------

Peeked Binds (identified by position):
--------------------------------------
   1 - (NUMBER): 20220509

Predicate Information (identified by operation id):
---------------------------------------------------
   1 - filter(TO_NUMBER("ORD_DT")=:V_ORD_DT)
```

```
-----------------------------------------------------------------------------------------------
| Id  | Operation                   | Name       | Starts | A-Rows |   A-Time   | Buffers |
-----------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT            |            |      1 |     30 |00:00:00.01 |       6 |
|   1 |  TABLE ACCESS BY INDEX ROWID| ORDERS     |      1 |     30 |00:00:00.01 |       6 |
|*  2 |   INDEX RANGE SCAN          | ORDERS_IX1 |      1 |     30 |00:00:00.01 |       4 |
-----------------------------------------------------------------------------------------------

Peeked Binds (identified by position):
--------------------------------------
   1 - (VARCHAR2(30), CSID=846): '20220509'

Predicate Information (identified by operation id):
---------------------------------------------------
   2 - access("ORD_DT"=:V_ORD_DT)
```

[Tibero]

```
-----------------------------------------------------------------------------------------------
| ID  | Operation             | Name       | Rows    | Elaps. Time  | CR Gets  | Starts   |
-----------------------------------------------------------------------------------------------
|   1 | TABLE ACCESS (ROWID)  | ORDERS     |      30 |00:00:00.0001 |       1  |       1  |
|   2 |  INDEX (RANGE SCAN)   | ORDERS_IX1 |      30 |00:00:00.0016 |      18  |       1  |
-----------------------------------------------------------------------------------------------

Predicate Information
---------------------------------------------------------------------------------------------
    2 - access: ("O"."ORD_DT" = :0) (0.005)
```

```
-----------------------------------------------------------------------------------------------
| ID  | Operation             | Name       | Rows    | Elaps. Time  | CR Gets  | Starts   |
-----------------------------------------------------------------------------------------------
|   1 | TABLE ACCESS (ROWID)  | ORDERS     |      30 |00:00:00.0001 |       1  |       1  |
|   2 |  INDEX (RANGE SCAN)   | ORDERS_IX1 |      30 |00:00:00.0001 |       2  |       1  |
-----------------------------------------------------------------------------------------------

Predicate Information
---------------------------------------------------------------------------------------------
    2 - access: ("O"."ORD_DT" = :0) (0.005)
```

티베로의 경우 ACCESS 처리되는 것처럼 보이지만 일량의 차이가 있음

2.8 자동 형변환

1. 변환 규칙 (계속 - 조건절 변수 / 타입 불일치)

[Tibero]

```
-- 타입 일치
-----------------------------------------------------------------------------------------------
| ID  | Operation             | Name       | Rows    | Elaps. Time  | CR Gets  | Starts   |
-----------------------------------------------------------------------------------------------
|   1 | TABLE ACCESS (ROWID)  | ORDERS     |       1 |00:00:00.0000 |       1  |       1  |
|   2 |  INDEX (UNIQUE SCAN)  | ORDERS_UX1 |       1 |00:00:00.0000 |       2  |       1  |
-----------------------------------------------------------------------------------------------

Predicate Information
-----------------------------------------------------------------------------------------------
    2 - access: ("ORDERS"."ORD_DT" = :0) AND ("ORDERS"."ORD_NO" = 1) (0.005 * 0.016)

-- 타입 불일치
-----------------------------------------------------------------------------------------------
| ID  | Operation             | Name       | Rows    | Elaps. Time  | CR Gets  | Starts   |
-----------------------------------------------------------------------------------------------
|   1 | TABLE ACCESS (ROWID)  | ORDERS     |       1 |00:00:00.0000 |       1  |       1  |
|   2 |  INDEX (UNIQUE SCAN)  | ORDERS_UX1 |       1 |00:00:00.0014 |      15  |       1  |
-----------------------------------------------------------------------------------------------

Predicate Information
-----------------------------------------------------------------------------------------------
    2 - access: ("ORDERS"."ORD_DT" = :0) AND ("ORDERS"."ORD_NO" = 1) (0.005 * 0.016)
```

티베로의 경우 ACCESS 처리되는 것처럼 보이지만 일량의 차이가 있음

2.8 자동 형변환

2. Decode(a, b, c, d) 변환규칙

① 'a = b'이면 c를 반환하고 아니면 d를 출력

이때 출력된 값의 Data Type은 세번째 인자인 c에 의해서 결정됨

따라서 c가 문자형이고 d가 숫자형이면 내부적으로 d가 문자형으로 변환

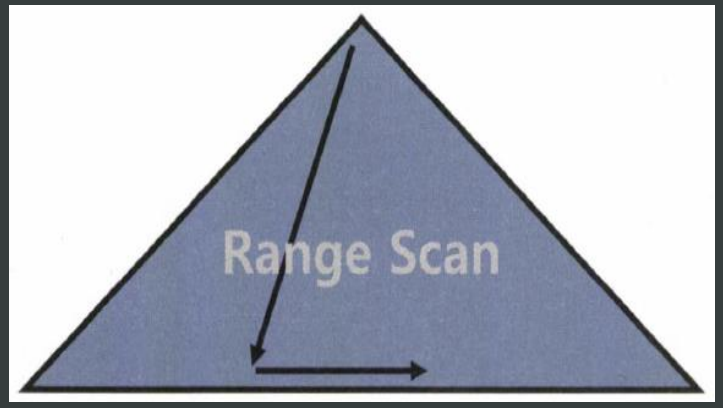
② c 인자가 null 값이면 varchar로 취급

```
max(decode(job, 'PRESIDENT', null, sal))             : sal이 문자로 취급, to_char(sal)로 변환
max(decode(job, 'PRESIDENT', null, to_char(sal)))    : 위와 결과 동일
max(decode(job, 'PRESIDENT', to_number(null), sal)) : c 인자를 to_number로 처리함
max(decode(job, 'PRESIDENT', null, to_number(sal))) : 강제적으로 d 출력값을 to_number로 처리함
```

```
select round(avg(sal)) avg_sal
     , min(sal)        min_sal
     , max(sal)        max_sal1
     , max(decode(job, 'PRESIDENT', null, sal))               max_sal2
     , max(decode(job, 'PRESIDENT', null, to_char(sal)))      max_sal3
     , max(decode(job, 'PRESIDENT', to_number(null), sal))    max_sal4
     , max(decode(job, 'PRESIDENT', null, to_number(sal)))    max_sal5
     , max(case when job = 'PRESIDENT' then null else sal end) max_sal6
  from emp;

AVG_SAL MIN_SAL MAX_SAL1  MAX_SAL2 MAX_SAL3 MAX_SAL4 MAX_SAL5 MAX_SAL6
------- ------- --------- -------- -------- -------- -------- --------
   2073     800      5000 950      950          3000      950     3000
```

2.9 인덱스 확장기능

1. Index Range Scan



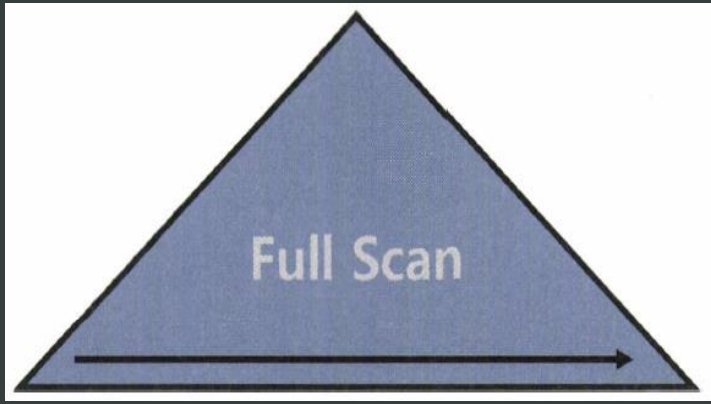

```
select * from emp where deptno = 20;

Execution Plan
----------------------------------------------------------
   0      STATEMENTS
   2    0   TABLE ACCESS (ROWID) OF 'EMP' (TABLE)
   3    1     INDEX (RANGE SCAN) OF 'EMP_X01' (INDEX)
```

- B*Tree 인덱스의 가장 일반적이고 정상적인 형태의 액세스 방식
- Root Block에서 Leaf Block까지 수직적으로 탐색한 후 필요한 범위(Range)만 스캔
- 수직 + 수평

- Index Range Scan을 하려면 선두 컬럼을 가공하지 않은 상태로 조건절에 사용해야 함
- 단순 실행계획만 보고 판단하면 안됨 (성능 OK?)
- Index Range Scan의 성능은 인덱스의 스캔 범위, 테이블 액세스 횟수를 얼마냐 줄일 수 있는냐가 관건

2.9 인덱스 확장기능

2. Index Full Scan



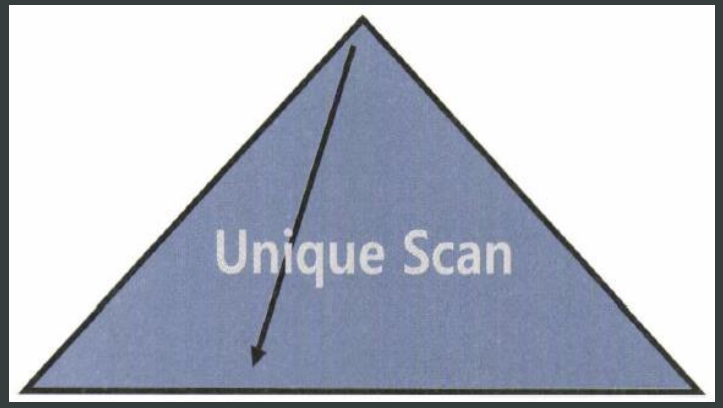
- 수직적 탐색없이 Leaf Block을 처음부터 끝까지 수평적으로 탐색하는 방식
- 대개 데이터 검색을 위한 최적의 인덱스가 없을 때 차선으로 선택됨
- 수평

```
create index emp_ename_sal_idx on emp(ename, sal);
select * from emp where sal > 2000 order by ename;

Execution Plan
----------------------------------------------------------
   0      STATEMENTS
   2    0   TABLE ACCESS (ROWID) OF 'EMP' (TABLE)
   3    1     FILTER
   4    2       INDEX (FULL) OF 'EMP_ENAME_SAL_IDX' (INDEX)
```

- 인덱스 선두 컬럼인 ename이 조건절에 없으므로 Index Range Scan은 불가능
- 뒤쪽이긴 하지만 sal 컬럼이 인덱스에 포함되어 있으므로 Index Full Scan을 통해 sal이 2000보다 큰 레코드를 찾음

2.9 인덱스 확장기능

3. Index Unique Scan



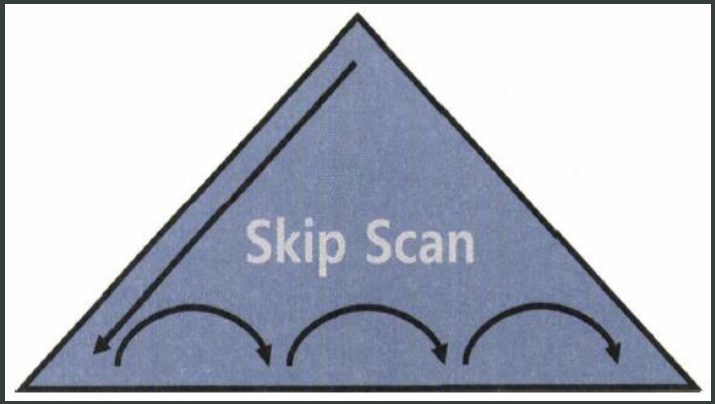

```
select * from emp where empno = 7788;

Execution Plan
----------------------------------------------------------
   0      STATEMENTS
   2    0   TABLE ACCESS (ROWID) OF 'EMP' (TABLE)
   3    1     INDEX (UNIQUE SCAN) OF 'PK_EMP' (INDEX)
```

- 수직적 탐색만으로 데이터를 찾는 스캔 방식
- Unique Index를 “=“ 조건으로 탐색할 경우 작동
- 수직

- Unique Index라고 해도 범위검색 조건(Between, Like, 부등호)으로 검색할 때는 Index Range Scan으로 처리됨
- Unique Composite Index에 대해 일부 컬럼만으로 검색할 때도 Index Range Scan으로 나타남

2.9 인덱스 확장기능

4. Index Skip Scan




```
create index emp_job_ename_idx on emp(job, ename);
select * from emp e where ename = 'CLERK';

Execution Plan
----------------------------------------------------------
   0      STATEMENTS
   2    0   TABLE ACCESS (ROWID) OF 'EMP' (TABLE)
   3    1     INDEX (SKIP SCAN) OF 'EMP_JOB_ENAME_IDX' (INDEX)
```

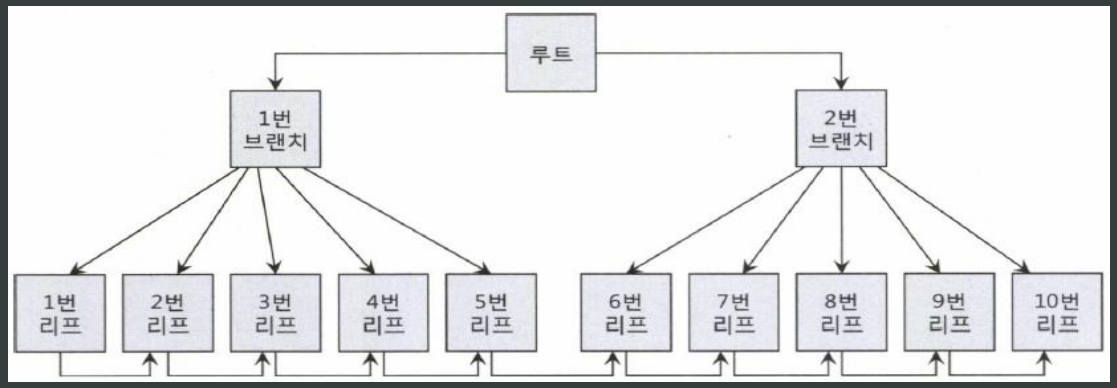
- 인덱스의 선두 컬럼이 조건절에 없어도 인덱스를 활용하는 스캔 방식
- 조건절에 빠진 인덱스 선두 컬럼의 Distinct Value 개수가 적고 후행 컬럼의 Distinct 개수가 많을 때 유용
- 수직 + 수평

- Index Skip Scan은 Root / Branch Block에서 읽은 컬럼 값 정보를 이용해 조건절에 부합하는 레코드를 포함할 가능성이 있는 Leaf Block만 골라서 액세스하는 방식
- 인덱스의 구성 컬럼에서 중간 컬럼에 대한 조건절이 없는 경우에도 Skip Scan을 사용할 수 있음

2.9 인덱스 확장기능

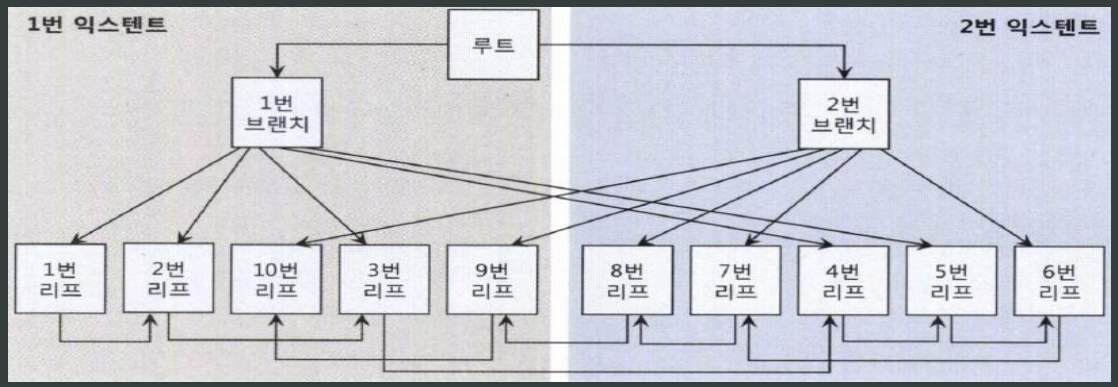
5. Index Fast Full Scan (Oracle vs. Tibero)

- Index Fast Full Scan은 Index Full Scan 보다 빠름 (이유는 논리적인 인덱스 트리 구조를 무시하고 인덱스 세그먼트 전체를 Multiblock I/O 방식으로 스캔하기 때문임)



Index Full Scan

루트 → 1번 브랜치 → 1 → 2 → 3 → 4 → 5 → 6 → 7 → 8 → 9 → 10번 순으로 블록을 읽음



Index Fast Full Scan

1번 익스텐트

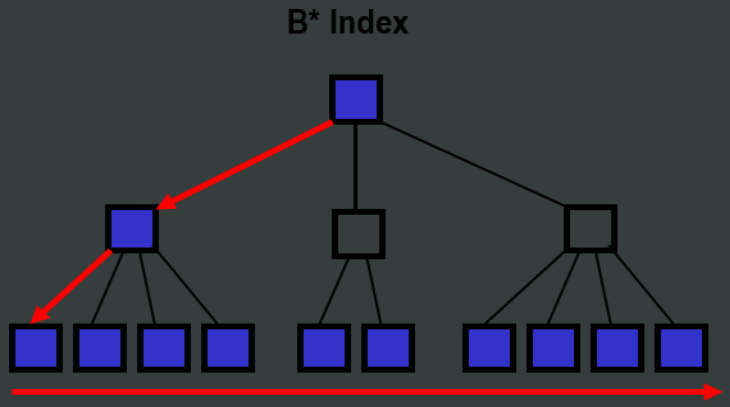
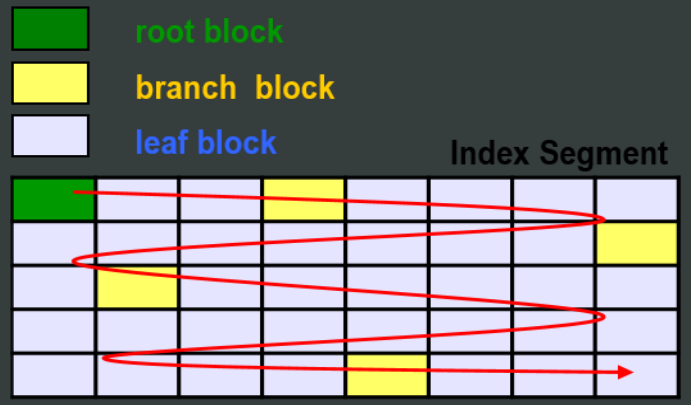
1 → 2 → 10 → 3 → 9

2번 익스텐트

8 → 7 → 4 → 5 → 6

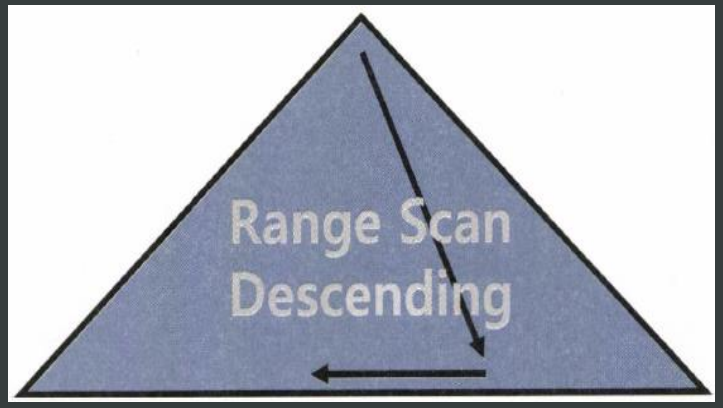
2.9 인덱스 확장기능

5. Index Fast Full Scan (계속)

Index Full Scan	Index Fast Full Scan
• 인덱스 구조를 따라 스캔 • 결과집합 순서 보장 • Single Block I/O • (파티션 안된 경우) 병렬스캔 불가 • 인덱스에 포함되지 않은 컬럼 조회시에도 사용 가능	• 세그먼트 전체를 스캔 • 결과집합 순서 보장 안 됨 • Multiblock I/O • 병렬스캔 가능 • 인덱스에 포함된 컬럼으로만 조회할 때 사용 가능 (오라클과 티베로는 차이가 있음, 별도 문서 참조)
	

2.9 인덱스 확장기능

6. Index Range Scan Descending



- Index Range Scan과 기본적으로 동일 방식
- 인덱스의 뒤쪽부터 스캔하기 때문에 정렬된 결과집합이 다름
- 수직 + 수평

```
create index emp_x02 on emp(deptno, sal);

select deptno, dname, loc
     , (select max(sal) from emp
         where deptno = d.deptno) as max_sal
  from dept d;

Execution Plan
----------------------------------------------------------
   0      STATEMENTS
   2    0   TABLE ACCESS (ROWID) OF 'DEPT' (TABLE)
   3    1     INDEX (FAST FULL SCAN) OF 'PK_DEPT' (INDEX)
   4    1     CACHE
   5    3       SORT AGGR
   6    4         COUNT (STOP NODE) (STOP LIMIT 2)
   7    5           INDEX (RANGE SCAN) DESCENDING OF 'EMP_X02' (INDEX)
```

- Max 값을 구하고자 할 때 해당 컬럼에 인덱스가 있으면 한 건만 읽고 멈추는 실행계획이 자동으로 수립됨
- "Min/Max" 알고리즘 vs. "RowNum <= 1"