DB 실행계획 분석

SQL문 작동순서.

◎ 소개

1. FROM
2. ON
3. JOIN
4. WHERE
5. GROUPT BY
6. HAVING
7. SELECT
8. ORDERBY

◎ 예제

```
--  SELECT문과 실행순서 비교 예제.
SELECT EMPLNUMB EMP
FROM HRPAYMBD
-- WHERE EMP = 'L0094'
-- GROUP BY EMP
-- ORDER BY EMP

// SELECT문에서 선언한 ALIAS EMP를 사용할 수 있는 절은
   ORDER BY절 밖에 없는 것을 확인할 수 있다.

-- GROUP BY , WHERE , HAVING 실행 순서 비교 예제.
SELECT EMPLNUMB , SUM(PROCAMNT)
FROM HRPAYMBD
-- WHERE SUM(PROCAMNT) > 1000000
 GROUP BY EMPLNUMB
-- HAVING SUM(PROCAMNT) > 1000000

// WHERE절에서는 GROUP BY가 선언되야 사용할 수 있는 그룹함수를 사용할 수 없지만
   HAVING절 에서는 그룹함수를 사용할 수 있다.

-- FROM절과 WHERE절의 실행순서 비교 예제
SELECT EMPLNUMB EMP
FROM HRPAYMBD A
WHERE A.EMPLNUMB = 'L0094'

// WHERE절에서는 FROM절에서 선언한 ALIAS를 사용할 수 있다.
```

쿼리실행 + 실행계획

◎ 소개

쿼리를 실행하고 실행계획을 실행하는 방법이다.

실행계획과 함께 출력된 결과물을 확인할 수 있다는 장점이 있다.

복잡한 쿼리의 경우 실제 실행하는데 많은 시간이 소요되는 단점이 있다.

◎ 방법

1. set autotrace on

   SQL 실제 수행 => SQL 실행결과, 실행계획 및 실행통계 출력

2. set autotrace on explain

   SQL 실제 수행 => SQL 실행결과, 실행계획 출력

3. set autotrace on statistics

   SQL 실제 수행 => SQL 실행결과, 실행통계 출력

4. set autotrace traceonly

   SQL 실제 수행 => 실행계획 및 실행통계 출력

5. set autotrace traceonly explain

   SQL 실제 수행 X => 실행계획 출력

6. set autotrace traceonly statistics

   SQL 실제 수행 => 실행통계 출력

* SQL Developer에서는 [F5] 버튼을 사용한다.

실행계획만.

◎ 소개

실행계획만 작동시키는 방법이다.

실제로 쿼리를 실행하지 않기 때문에 속도가 빠르다.

실행결과를 직접적으로 확인할 수 없다는 단점이 있다.

◎ 방법

아래의 내용이 한 묶음이다.

```
 -- 실행계획 저장하기.
EXPLAIN PLAN
       SET STATEMENT_ID = 'TEST1'
       INTO PLAN_TABLE
      FOR

 -- 실행할 쿼리문
   SELECT *
    FROM HREMPLMT ;

 -- 저장된 실행계획 보기.
   SELECT *
    FROM TABLE(DBMS_XPLAN.DISPLAY
                    ('PLAN_TABLE', 'TEST1', 'ALL'));
```

실행계획 해석하기.

◎ 소개

출력된 실행계획을 확인하는 방법.

◎ 방법

```
Execution Plan
----------------------------------------------------------

Plan hash value: 169057108

-------------------------------------------------------------------------------
| Id  | Operation                    | Name   | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------
|   0 | SELECT STATEMENT             |        |     3 |   114 |     2   (0)| 00:00:01 |
|*  1 |  TABLE ACCESS BY INDEX ROWID| EMP    |     3 |   114 |     2   (0)| 00:00:01 |
|*  2 |   INDEX RANGE SCAN           | PK_EMP |     3 |       |     1   (0)| 00:00:01 |
-------------------------------------------------------------------------------

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

   1 - filter("SAL">1400)
   2 - access("EMPNO">7800)
```

항목	의미
Id	Operation의 Id. Predicate Information에 정보가 있다면 Id와 연결된다.
Operation	수행되는 일
Name	Operation이 수행되는 테이블 혹은 인덱스 --> 오브젝트명
Rows	각 Operation이 끝났을 때 반환되는 예상치 건수
Bytes	Access 하는 byte 수 예상치
Cost(%CPU)	Operation의 비용 --> 누적치
Time	예상 수행 시간
access Predicate	block을 읽기 전에 어떤 방법으로 block을 읽을 것인지 결정. --> index ? full table scan ? 등
filter Predicate	block을 읽은 후 데이터를 필터링되어 사용되는 조건

```
-- 실행계획 분석 예제
SELECT *
  FROM HREMPLMT A
  LEFT JOIN HRPAYMBD B
    ON A.EMPLNUMB = B.EMPLNUMB
 WHERE A.EMPLNUMB = 'L0124'
   AND B.PAYMONTH = '2020-01'
```

인덱스 타는 조건.

컬럼에 대한 인덱스가 존재해야 한다.

◎ 소개

WHERE절에서 사용할 컬럼에 대한 인덱스가 존재해야 한다.

◎ 예제

```
 -- 인덱스가 있음
SELECT *
FROM HREMPLMT
WHERE EMPLNUMB = 'L0094';

// PK(기본키)의 경우 자동으로 인덱스를 생성해준다.


 -- 인덱스가 없음
SELECT *
FROM HREMPLMT
WHERE DEPTCODE = '610';

// PK(기본키)가 아닌 경우 기본적으로 인덱스가 없다.
// 이런 경우 직접 인덱스를 생성해줘야 한다.
// 하지만 인덱스를 생성하게 되면 테이블이 변하거나 수정될 때마다
   인덱스 또한 수정되기 때문에 불필요한 컬럼은 인덱스를 생성하지 않는 것이 좋다.

// 인덱스의 생성은
   읽기의 효율  VS  쓰기의 효율
```

WHERE절에서 컬럼을 수정하지 않아야 한다.

◎ 소개

WHERE절에서 컬럼을 수정해서 사용하면 인덱스를 타지 않는다.

◎ 예제

```
-- 컬럼을 가공 안한 경우
SELECT *
FROM HRPAYMBD
WHERE PAYMONTH  = '2020-01'

// 컬럼을 가공하지 않으면 정상적으로 인덱스를 탄다.

 -- 컬럼을 가공한 경우
SELECT *
FROM HRPAYMBD
WHERE SUBSTR(PAYMONTH , 1 , 7 ) = '2020-01'

// 조건은 같지만 컬럼을 가공하면 인덱스를 타지 못한다.
```

컬럼이 암시적(Implicit) 형변환을 하면 안된다.

◎ 소개

예기치 못하게 WHERE절 컬럼의 자료형이 변환되는 경우를 뜻하는 것으로 보임.

인덱스의 종류.

Index Unique Scan.

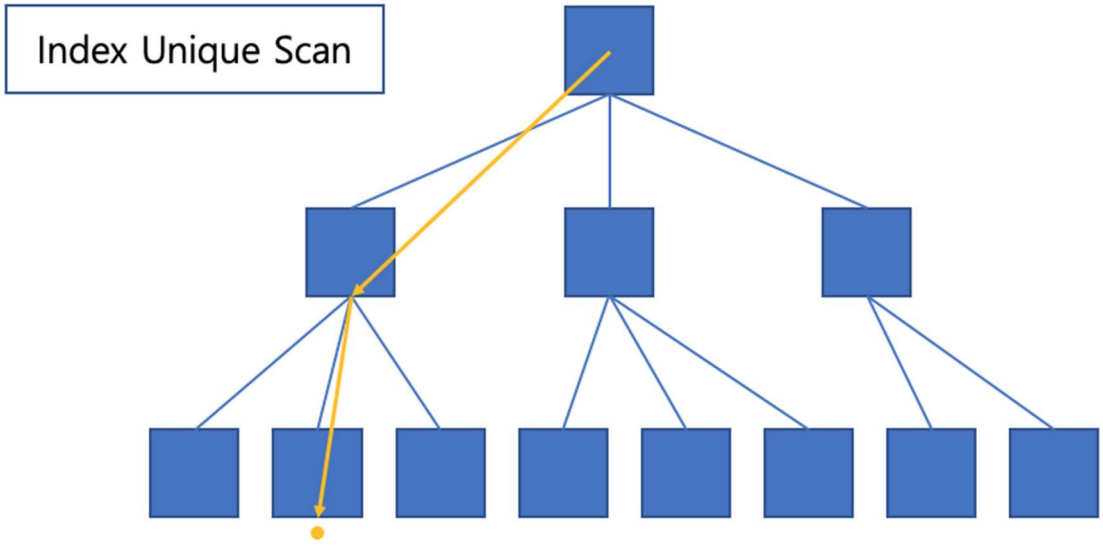
◎ 소개

결과로 하나의 건이 반환될 때 사용되는 Scan방식이다.

칼럼이 유일한 값으로 구성되어야 합니다. 예를들어 Id값.

Primary Key에 기본적으로 생성되는 인덱스입니다.

조건 검색이 단일(=)인 경우 사용하게됩니다.



◎ 예제

```
 -- Index Unique Scan 예제 1
SELECT *
FROM HREMPLMT
WHERE EMPLNUMB = 'L0094';

// EMPLNUMB 유일한 값을 가지기 때문에 Index Unique Scan을 타는 것을 알 수 있다.

 -- Index Unique Scan 예제 2
SELECT *
FROM HREMPLMT
WHERE EMPLNUMB IN ( 'L0094' , 'L0095' , 'L0096' );
```

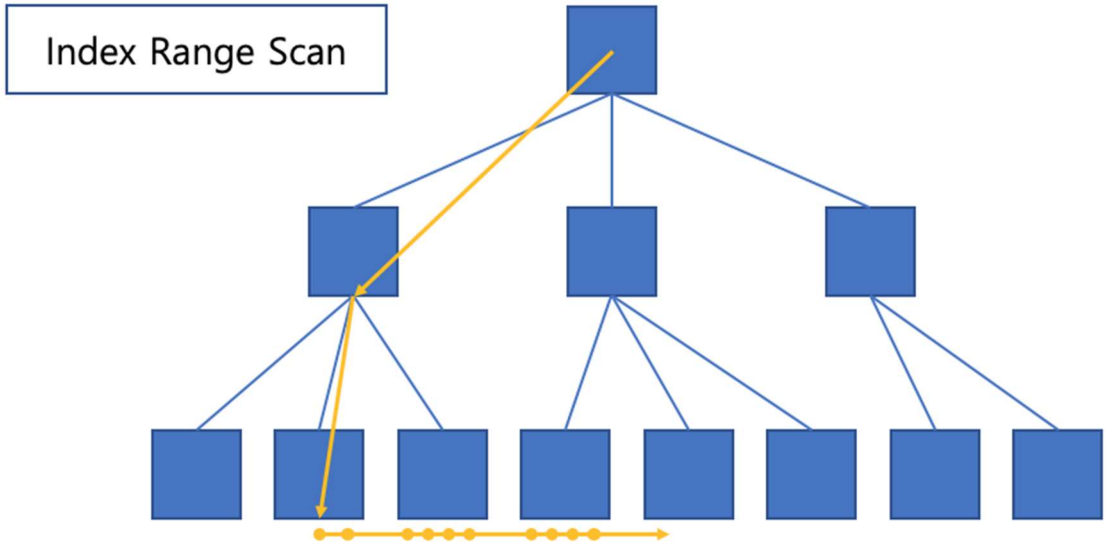
Index Range Scan

◎ 소개

Index Range Scan은 가장 일반적인 Index의 탐색 방식이다.
Index를 수직 탐색 후 필요한 범위까지만 탐색은 하는 방식이다.
인덱스의 데이터는 정렬된 상태로 저장이 되므로
범위에 대한 탐색이 필요할때 효율적으로 데이터를 탐색할 수 있다.

리프 블록에서 다음 리프블록의 정보를 갖고 있어
다시 브랜치 블록을 읽을 필요없이 다음 데이터를 바로 읽을 수 있습니다.

이런 특징을 잘 활용하면 정렬작업을 생략하거나
최대값, 최솟값을 찾는데 유용하게 활용될 수 있습니다.



◎ 예제

```
-- Index Range Scan 예제 1
SELECT *
FROM HRPAYMBD
WHERE PAYMONTH  = '2020-01'

// PAYMONTH의 값은 유일하지 않기 때문에
  Index Range Scan이 실행된다.


-- Index Range Scan 예제 2
SELECT *
FROM HRPAYMBD
WHERE PAYMONTH  BETWEEN '2020-01' AND '2020-02'
```

```
ORDER BY PAYMONTH , EMPLNUMB;
```

// Index를 타는 경우 데이터가 정렬되어 있기 때문에
   따로 정렬을 실행하지 않는 것을 알 수 있다.

```
-- Index Range Scan 예제 3
SELECT *
FROM HRPAYMBD
WHERE SUBSTR(PAYMONTH , 1 , 7)  BETWEEN '2020-01' AND '2020-02'
ORDER BY PAYMONTH
```

// Index를 타지 않는 경우 정렬 작업을 수행한다.

```
-- Index Range Scan 예제 4
SELECT *
FROM HRPAYMBD
WHERE PAYMONTH  BETWEEN '2020-01' AND '2020-02'
ORDER BY PAYMONTH DESC , EMPLNUMB DESC ;
```

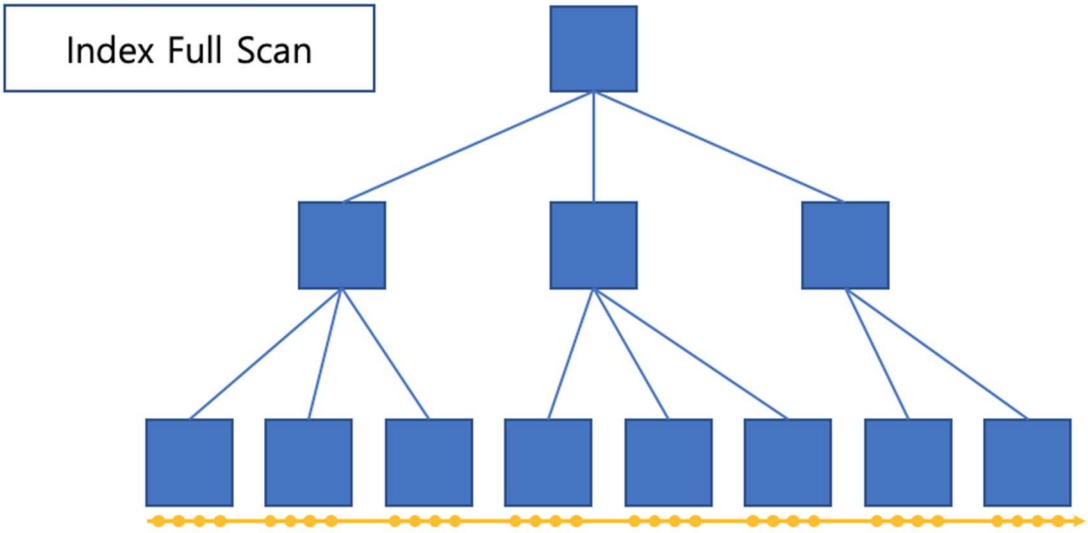
// 반대방향인 DESC로 정렬하게 되면
   실행계획에서 DESCENDING으로 실행하는 것을 볼 수 있다.
// 이는 오름차순으로 정렬되어 있는 인덱스를
   앞에서부터가 아닌 뒤에서부터 탐색하여 결과를 반환하는 것이다.

Index Full Scan

◎ 소개

Index Full Scan은 첫번째 리프블록까지 수직적 탐색 후, 인덱스 전체를 탐색하는 방법입니다.
주로 테이블에서 Table Full Scan의 부담이 크거나 정렬작업을 생략하기 위해
테이블 전체를 탐색하는 것보다 Index를 사용하는 것이 유리합니다.

그런데 이 방법은 생성된 인덱스가 없어 사용된 방법으로
Index Range Scan으로 유도하는 것이 좋습니다.



◎ 예제

```
-- Index Full Scan 예제1
SELECT EMPLNUMB
FROM HREMPLMT

-- Index Full Scan 예제
SELECT EMPLNUMB , KORENAME
FROM HREMPLMT
```

Index Fast Full Scan.

◎ 소개

Index Fast Full Scan은 말 그대로 Index Full Scan보다 빠르다.

전부를 탐색하면서 더 빠르게 결과를 얻을 수 있는 이유는 Multi-Block Read가 가능하기 때문이다.

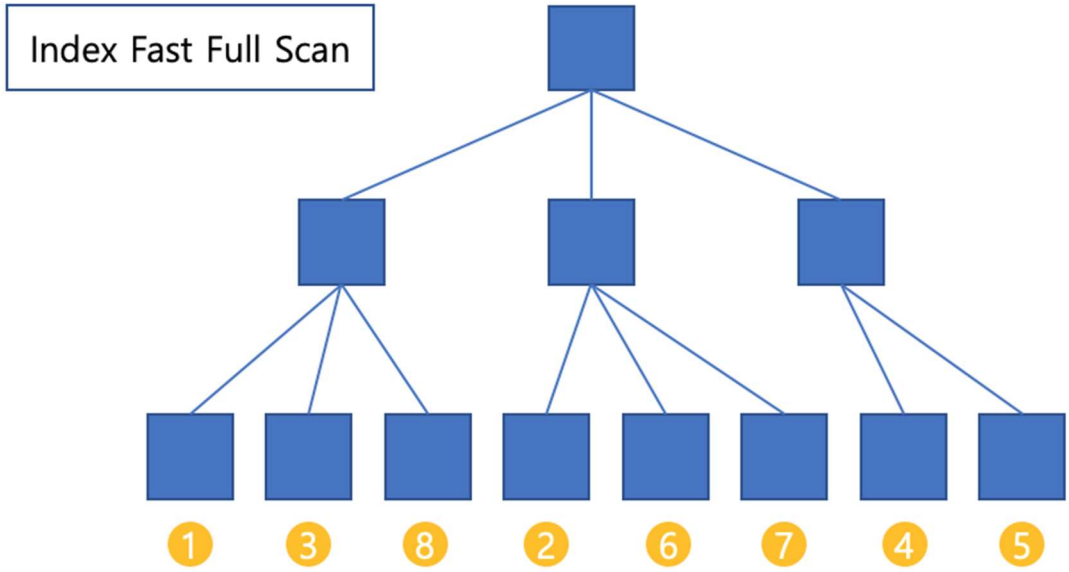
하지만 이로 인해 정렬된 상태로 데이터를 받을 수 없다는 제한이 있다.

Index Full Scan은 데이터의 논리적인 순서로 결과를 얻지만

Index Fast Full Scan은 데이터의 물리적 저장 구조의 순서로 받아오고

그래서 Multi-Block Read가 가능합니다.

Multi-Block Read : 인접한 블록들을 같이 읽어 메모리에 적재



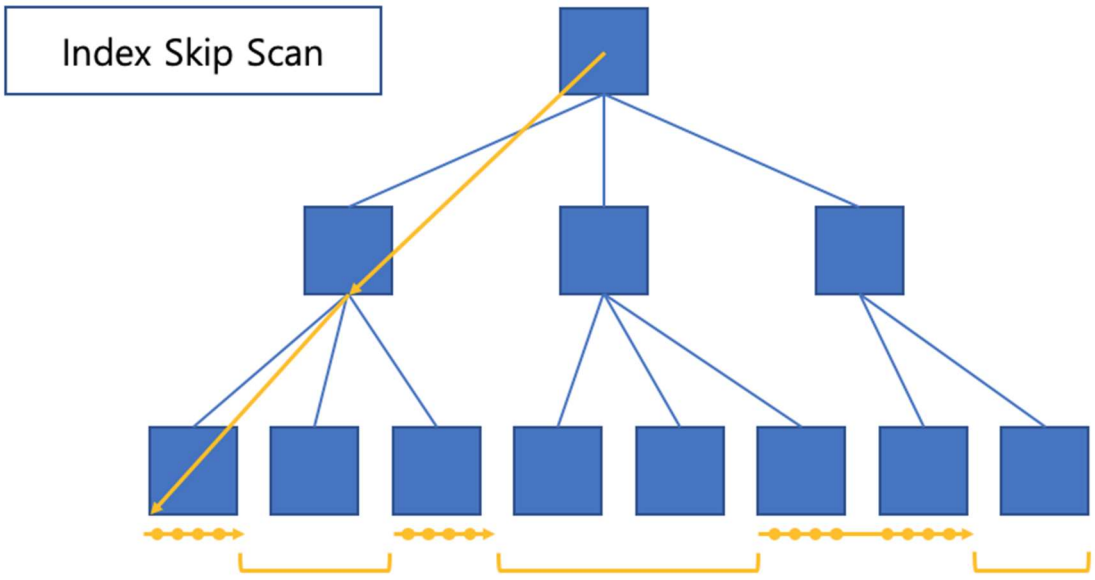
◎ 예제

없음.

Index Skip Scan

◎ 소개

Index Skip Scan은 Multi Column Index에서 후행 칼럼을 조건절에 사용할 때 활용할 수 있습니다.
성별처럼 선두 칼럼의 고유 값의 개수가 적고 후행 칼럼의 고유 값이 많을 때 효과적입니다.
예를 들어 부서가 10,20,30,40 인 회사가 있는데 직원이 1000명입니다.
각각 500,100,200,200 명의 직원이 있습니다.
우리가 알고 싶은 내용은 급여가 5000 ~ 7000 사이인 직원입니다.
생성된 인덱는 (부서번호,급여)로 결합된 인덱스가 있습니다.
만약 Index Skip Scan을 사용하면 10번 부서에서 5000 ~ 7000인 직원을 찾고
그 이후 10번 부서의 데이터는 확인할 필요가 없기 때문에
20번 부서로 넘어가서 똑같이 반복합니다.
이런식으로 필요없는 부분에 대해서 건너뛴다면 Full Table Scan보다 성능이 좋을 수 있습니다.



◎ 예제

```
-- Index Skip Scan 예제
SELECT /*+index_ss(HRPAYMBD PK_HRPAYMBD)*/   *
FROM HRPAYMBD
WHERE PROCAMNT BETWEEN 500000 AND 1000000
```

Index Min / Max Scan

Index Join