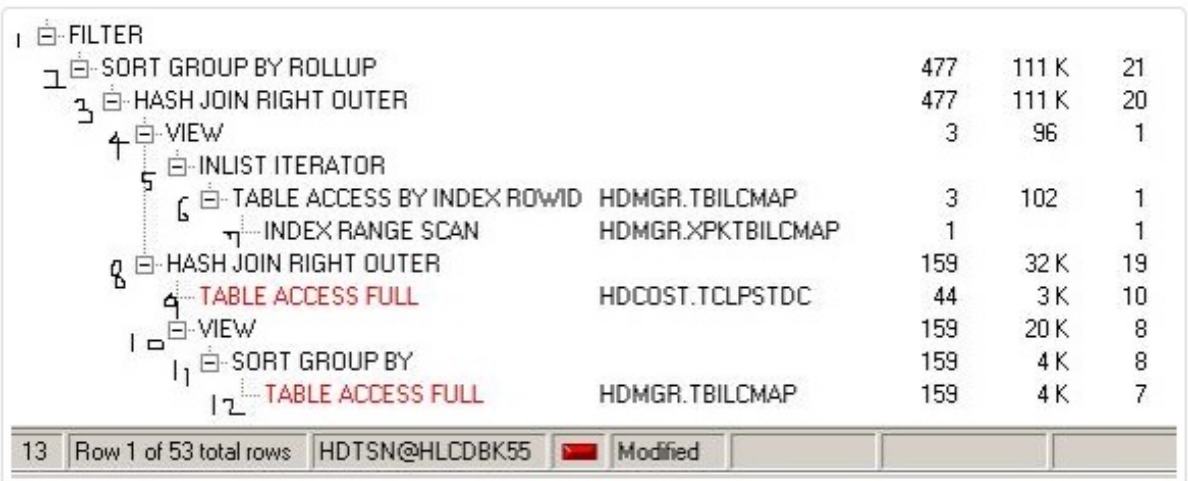
실행계획 읽는 법

1. 위에서 아래로
2. 안에서 밖으로 (안에서 밖으로를 우선으로 두어 가장 안쪽에 있는 구문을 처음 읽는것이라고 착각해서는 안됨)



by finecomp [2009.12.15 16:37:34]

플랜보는 법을 외우려고 하지마세요...;
8의 hash조인이 되려면 9의 집합을 build-in으로하고 10의 view를 탐색하여 조인한다는 계획입니다.
10의 view는 11 group by가 끝나기 전엔 알 수 없고 group by는 12의 데이터를 scan하기전엔 절대 알 수 없습니다.
해서 당연히 9>12>11>10>8 순으로 scan하고 hash로 연결을 완성하겠단 계획입니다.

1. Plan 읽는 법

- Plan은 SQL을 실행하기 전에 Optimizer에 의해 선택된 최적의 실행 경로 및 계산되어진 예상 Cost를 보여줍니다.

```
**************************[Explain Plan Time: 2016/02/21 14:01:15]**************************
Execution Plan
------------------------------------------------------------
    0       SELECT STATEMENT Optimizer=ALL_ROWS (Cost=5 Card=5 Bytes=325)
    1     0   SORT (ORDER BY) (Cost=9 Card=5 Bytes=325)
    2     1     UNION-ALL
    3     2       COUNT (STOPKEY)
    4     3         VIEW (Cost=5 Card=1 Bytes=65)
    5     4           SORT (ORDER BY STOPKEY) (Card=1 Bytes=14)
    6     5             FILTER
    7     6               TABLE ACCESS (FULL) OF 'BBS' (TABLE) (Cost=3 Card=10 Bytes=1
    8     2       VIEW (Cost=4 Card=4 Bytes=260)
    9     8         SORT (ORDER BY) (Cost=4 Card=4 Bytes=56)
   10     9           COUNT (STOPKEY)
   11    10             TABLE ACCESS (FULL) OF 'BBS' (TABLE) (Cost=3 Card=5 Bytes=70)
------------------------------------------------------------

Predicate information (identified by operation id):
------------------------------------------------------------
    3 - filter(ROWNUM<=4)
    5 - filter(ROWNUM<=4)
    6 - filter(NULL IS NOT NULL)
    7 - filter("NUM"<10)
   10 - filter(ROWNUM<=4)
   11 - filter("NUM">10)
------------------------------------------------------------
```

1.1 실행순서 (access path)

- sibling 사이에서는 먼저 나온 것을 먼저 처리
- child가 있는 경우 child부터 다 처리하고 parent 처리하기

이 두가지만 기억하면 됩니다.

위 Plan을 기준으로 처리 순서는 다음과 같습니다.
(0 ~ 11까지 있는 왼쪽의 Index를 사용하겠습니다.)

```
7 -> 6 -> 5 -> 4 -> 3 -> 11 -> 10 -> 9 -> 8 -> 2 -> 1 -> 0
```

- (4) UNION-ALL 아래에 2개의 child(3,8)가 있습니다.
- 이 둘중 위에 있는 (3)부터 처리를 해야하는데 (3)은 child가 있으므로 가장 안쪽부터 처리합니다.
- (3)의 child를 다 처리한 후에 자신의 sibling인 (8)을 처리해야하는데, (8)도 child가 있으므로 안쪽부터 처리합니다.
- (3, 8)이 모두 처리된 후에 (2)부터 쭉 처리하면 됩니다.

1.2 예상 성능지표 (Cost-based Optimizer Mode에서만 표시)

- Cost : Cost 예상 지수. 클수록 성능상 (CPU 점유, Disk I/O, 수행시간 등...) 안좋다는 의미입니다.
- Card : (Computed Cardinality) : ~~CBO상 계산된 예상되는 return row~~ 입니다.
- Bytes : ~~return row의~~ byte수 입니다.

access된 row 수

access된

1.3 Predicate information

각 단계별 filter 조건이 어떻게 적용되었다는 정보를 보여줍니다.

Cost=633K : 633,000 비용발생(논리적 비용 = IO + MEM + CPU + NET + …)

Card=42M : 42,000,000건(접근하는 레코드 수)

Bytes=15G : 15,000,000,000(42,000,000 * 1 ROW의 총 길이)

Cost는 비용을 의미하는데 해당 쿼리가 동작되었을 때 소요하는 비용을 말한다. 비용이 크면 클수록 오라클이 많은 일을 하고 있다고 생각하면 된다. 즉 무거운 쿼리인 것이다. 여기서 비용은 물리적 비용이 아니라, 논리적 비용을 의미한다. 논리적 비용이란 직접적이고 구체적인 수치에 의해 명확하게 알 수 있는 비용이 아니다. 오라클 옵티마이저가 산출한 비용에 대해 우리는 왜 그러한 비용값이 계산되었는지 이해할 필요까지는 없다. 단지 플랜 비교 시 비교 기준 값으로는 삼을 수 있다.

Card(Cardinality)는 쿼리 조건에 맞는 레코드 건수를 의미한다. 참고로 우리는 K는 10의 3승을 의미하고, M은 10의 6승을 의미하고, G는 10의 9승을 의미함을 이미 알고 있다. 위의 플랜에서 Card 값은 42M이므로 고객 테이블의 데이터 건수가 4,200만임을 알 수 있다.

Bytes는 쿼리 실행,시 발생하는 네트워크 트레픽을 의미한다. 즉 I/O 발생량이다. 1Row를 구성하는 컬럼의 길이 총 합을 구한 후 Card 값을 곱하면 된다. 결국 위의 플랜에서는 15,000,000,000 Byte라는 어마어마한 네트워크 트레픽이 발생함을 알 수 있다.

Id, Operation, Name

이 부분은 우리가 흔히 봐 왔던 플랜 정보다. 자원에 대한 접근 순서와 접근 방법을 나타낸다. 참고적으로 접근 순서를 변경할 수 있는 힌트절은 ORDERED, LEADING이 있다. 또한 접근 방법을 변경할 수 있는 힌트절은 USE_NL, USE_HASH, USE_MERGE가 있다.

Starts

오퍼레이션을 수행한 횟수를 의미한다. Starts * E-Rows 의 값이 A-Rows 값과 비슷하다면, 통계정보의 예측 Row 수와 실제 실행 결과에 따른 실제 Row 수가 유사함을 알 수 있다. 만약 값에 큰 차이가 있다면 통계정보가 실제의 정보를 제대로 반영하지 못했다고 생각할 수 있다. 이로 인해 오라클의 Optimizer가 잘못된 실행 계획을 수립할 수도 있음을 염두에 둬야 한다.

E-Rows (Estimated Rows)

통계정보에 근거한 예측 Row 수를 의미한다. 통계정보를 갱신할수록 값이 매번 다를 수 있으며, 대부분의 DB 운영에서는 통계정보를 수시로 갱신하지 않으므로 해당 값에 큰 의미를 둘 필요는 없다. 하지만 E-Rows 값과 A-Rows 값이 현격하게 차이가 있다면 오라클이 잘못된 실행 계획을 세울 수도 있음을 인지해야 하며 통계정보 생성을 검토해 보아야 한다.

A-Rows (Actual Rows)

쿼리 실행 결과에 따른 실제 Row 수를 의미한다. 우리는 A-Rows 에서 중요한 여러 정보를 추정 할 수 있다. 이에 대한 부분은 이번 연재에서 계속 설명이 이어진다(이번 호 문제 부분에서).

A-Time (Actual Elapsed Time)

쿼리 실행 결과에 따른 실제 수행 시간을 의미한다. 하지만 실행 시점의 여러 상황이 늘 가변적이고 또한 메모리에 올라온 Block의 수에 따라서 수행 시간이 달라지므로 해당 값에 큰 의미를 둘 필요는 없다.

Buffers (Logical Reads)

논리적인 Get Block 수를 의미한다. 해당 값은 오라클 옵티마이저가 일한 총량을 의미하므로, 튜닝을 진행할 때 필자가 가장 중요하게 생각하는 요소 중 하나다.

Reads (Physical Reads)

물리적인 Get Block 수를 의미한다. 동일한 쿼??음이 아닌 경우에는 값이 0인 것을 보면 알 수 있듯이 메모리에서 읽어온 Block은 제외된다. 해당 값에 큰 의미를 둘 필요는 없다.

위의 헤더에서 튜닝 시 가장 중요하게 활용되는 부분은 Buffers와 A-Rows다. Buffers 값을 통해서 Get Block의 총량을 알 수 있고, A-Rows를 통해 플랜 단계별로 실제 Row 수를 알 수 있기 때문이다.