Oracle HINT

사용법

```
{SELECT | INSERT | UPDATE | DELETE} /*+ hint [text] [hint [text]] ... */
```

이러한 힌트의 사용은 SQL전체가 아닌 쓰여진 SQL 블럭에만 적용됩니다.

힌트의 종류별 분류

Optimization Goals and Approaches

ALL_ROWS

```
/*+ ALL_ROWS */
```

최소한의 자원을 사용하여 결과값의 전체를 추출하게 합니다. 가장 좋은 단위 처리량의 목표로 문 블록을 최적화하기 위해 cost-based 접근 방법을 선택합니다. (즉, 전체적인 최소의 자원 소비, 모든 레코드의 처리하는 시간의 최소활르 목적으로 최적화)

FIRST_ROWS

```
/*+ FIRST_ROWS(n) */
```

전체 결과값의 반환 대신 지정한 숫자만큼 로우의 결과값을 반환하는데 집중하게 유도합니다. 가장 좋은 응답 시간의 목표로 문 블록을 최적화하기 위해 cost-based 접근 방법을 선택합니다. (첫번째 레콛의 추출 시간을 최소화할 목적으로 최적화)

CHOOSE

```
/*+ CHOOSE */
```

최적자(optimizer)가 그 문에 의해 접근된 테이블을 위해 통계의 존재에 근거를 두는 SQL문을 위해 Rule-Based 접근 방법과 Cost-Based 접근 방법 사이에 섵개하게 됩니다.

RULE

```
/*+ RULE */
```

실행 계획을 Rule-Based 방식으로 실행하게 합니다. 해당 쿼리 블럭에 다른 힌트 또한 사용되었을 경우, 다른 힌트들은 사용되지 않습니다.

Acess Method Hints

AND_EQUAL

```
/*+ AND_EQUAL (table index index [index] [index] [index]) */
```

복수의 단일 컬럼을 스캔하여 머지 방식으로 처리하게 합니다.

CLUSTER

```
/*+ CLUSTER (table) */
```

지정 테이블의 클러스터 스캔을 유도합니다. 클러스터된 객체에만 사용할 수 있습니다.

FULL

```
/*+ FULL (table) */
```

지정한 테이블에 대해 풀 테이블 스캔을 유도합니다.

HASH

```
/*+ HASH (table) */
```

지정한 테이블에 대해 hash 스캔을 수행하도록 유도합니다. 클러스터 테이블 만을 대상으로 합니다.

INDEX

```
/*+ INDEX (table index [index] [index] ...) */
```

INDEX를 순차적으로 스캔

NO_INDEX

```
/*+  NO_INDEX (table [index] [index] ...) */
```

지정 테이블의 인덱스 사용을 방지합니다.

INDEX_ASC

INDEX를 내림차순으로 스캔.

INDEX_DESC

INDEX를 오름차순으로 스캔.

INDEX_COMBINE

```
/*+ INDEX_COMBINE */
```

해당 테이블에 Bitmap 인덱스의 존재 시,, Bitmap 인덱스를 통한 액세스를 유도합니다.

INDEX_FFS

```
/*+ INDEX_FFS (table [index] [index] ...) */
```

풀 테이블 스캔 대신에 빠른 풀 테이블 스캔의 실행을 유도합니다.

ROWID

```
/*+ ROWID (table) */
```

지정한 테이블의 스캔을 ROWID 방식으로 수행하게 합니다.

Join Order Hints

ORDERED

```
/*+ ORDERED */
```

FROM절에 나열된 테이블의 순서대로 조인 작업을 실행합니다.

STAR

```
/* STAR */
```

* Star 쿼리 계획이 사용 가능하다면, 실행하게 됩니다. * Star 쿼리 계획이란 가장 큰 테이블이 마지막 순서로 조인되며, 조인될 시 가장 큰 테이블 내의 Concatenated 인덱스에 대해 Nested Loop 조인 방식으로 실행되는 것을 말합니다. * 최소한 세개 이상의 테이블이 사용되며, 엑세스나 조인 방식에 충돌이 없어

야만 이 힌트는 사용됩니다.

Join Operation Hints

DRIVING_SITE

```
/*+ DRIVING_SITE (table) */
```

오라클이 선택한 SITE 대신, 지정한 SITE를 사용하여 쿼리를 실행합니다. Rule-Based와 Cost-Based, 두 모두 다 사용 가능합니다.

HASH_AJ

```
/*+ HASH+AJ */
```

EXISTS 구문 뒤에 오는 서브 쿼리에 사용된다. HASH*AJ* 는 *hash anti-join*, *SJ* 는 *semi-join*, *MERGE*SJ 은 sort merge semi-join 이며 NL_SJ 은 nested loop semi-join 이다.

LEADING

```
/*+ LEADING (table) */
```

* 테이블 간의 조인 시에 지정한 테이블을 먼저 수행하도록 유도합니다. * 두 개 이상의 LEADING 힌트의 사용 시, 힌트 자체가 사용되어 지지 않습니다. * ORDERED 힌트와 더불어 사용시, LEADING 힌트는 사용되지 않습니다.

USE_HASH

```
/*+ USE_HASH (table [table]...) */
```

Hash 조인 방식으로 각 테이블을 조인하게 합니다.

USE_MERGE

```
/*+ USE_MERGE (table [table]...) */
```

Sort-Merge방식으로 각 테이블을 조인하게 합니다.

USE_NL

```
/*+ USE_NL (table [table]...) */
```

Nested-Loop방식으로 각 테이블을 조인하게 합니다.

Parallel Execution Hints

PARALLEL

```
/*+ PARALLEL (table, n) */
```

* SELECT, INSERT 시 여러개의 프로세스로 수행

NOPARALLEL

```
/*+ NOPARALLEL(table) */
```

* 지정한 테이블의 병렬 처리를 방지합니다. * 테이블의 지정된 PARALLEL 값에 대해서 우선권을 가집니다. * 중첩 테이블에 대해서는 병렬 처리를 할 수 없습니다.

PARALLEL_INDEX

```
/*+ PARALLEL_INDEX (table [[indxe] [,i ndex]...] [[,n |, DEFAULT |,] [, n | DEFAULT]]) */
```

파티션 인덱스의 인덱스 범위 스캔 작업의 병렬 처리에 할당될 서버 프로세스의 갯수를 지정합니다.

PQ_DISTRIBUTE

```
/* PQ_DISTRIBUTE (table [,] outer_distribution, inner_distribution) */
```

병렬 조인 시, Producer 프로세스와 Consumer 프로세스 간의 데이터 전달 방식을 지정합니다.

NOPARALLEL_INDEX

```
/* NOPARALLEL_INDEX (table [index] [index] ...) */
```

인덱스 스캔 작업의 병렬 처리를 방지합니다. 인덱스에 지정된 PARALLEL 값에 우선권을 가집니다.

Query Transformation Hints

EXPAND_GSET_TO_UNION

```
/* EXPAND_GSET_TO_UNION */
```

* GROUP BY GROUPING SET 혹은 GROUP BY ROLLUP 등과 같은 구문을 포함하는 쿼리에 사용할 수 있습니다. * 이 힌트는 기존의 쿼리를 개별적인 그룹 생성 후, UNION ALL 방식으로 실행되게 유도합니다.

FACT

```
/*+ FACT (table) */
```

스타변형 구문에서 사용되며 해당 테이블이 FACT 테이블로 사용되게 유도합니다.

NO_FACT

```
/*+ NO_FACT (table) */
```

Star 변형 시, 해당 테이블의 FACT 테이블로서의 사용을 방지합니다.

MERGE

```
/*+ MERGE (table) */
```

* VIEW MERGING 수행 * 각 쿼리의 결과값을 머지합니다. * 해당 쿼리 내에 GROUP BY 절의 사용이나 SELECT 구문에 DISTINCT가 사용되었을 시, 머지의 실행이 가능할 경우에만 힌트가 실행됩니다. * IN과 서브 쿼리의 사용 시, 서브 쿼리와 상위 쿼리 간의 상호 관계가 없을 시에만 머지의 실행이 가능합니다. * 이 힌트는 Cost-based가 아닙니다. 따라서 액세스하는 실행 쿼리 블럭에 MERGE 힌트가 반드시 명시되어야만 합니다. 그렇지 않을 경우 옵티마이저는 다른 실행 계획을 수립합니다.

NO_EXPAND

```
/* NO_EXPAND */
```

* 실행 쿼리 내에 OR 나 WHERE 절의 IN이 사용되었을 시, Cost-Based 옵티마이저가 쿼리 처리를 위해 OR를 사용한 확장을 사용하는 것을 방지합니다. * 일반적으로 옵티마이저는 위와 같은 경우 OR - 확장의 가격이 확장을 사용하지 않는 것보다 적을 시, 확장 방식으로 수행합니다.

NO_MERGE

```
/*+ NO_MERGE(table) */
```

* 머지 처리 방식의 사용을 방지합니다. * VIEW MERGING 수행못하게 함

REWRITE

```
/*+ REWRITE [([materialized_view] [materialized_view]...)] */
```

- 실행 계획의 가격에 상관없이 Materialized View를 사용하여 쿼리 재생성을 하도록 합니다.
- Materialized View를 지정할 시, 지정한 Materialized View 의 가격에 상관없이 무조건 쿼리 재생성을 실행합니다.
- Materialized View를 지정하지 않을 시, 오라클은 사용 가능한 모든 Materialized View를 참조하여 그 중 가장 가격이 낮은 Materialized View를 사용하여 쿼리 재생성을 합니다.
- Materialized View를 지정하지 않는 힌트의 사용이 권장됩니다.

NOREWRITE

```
/*+ NOREWRITE */
```

- 해당 쿼리 블럭의 쿼리 재생성의 실행을 방지합니다.
- QUERY*REWRITE*ENABLE 파리미터에 대해 우선권을 가집니다.
- NOREWIRTE 힌트의 사용시, Function-Based 인덱스의 사용이 금지됩니다.

USE_CONCAT

```
/*+ USE_CONCAT */
```

- WHERE 절의 OR 조인을 UNION ALL로 변경하여 수행하게 합니다.
- 일반적으로 이러한 변경은 결과값의 병합 수행의 가격이 수행하지 않을 시의 가격 보다 낮을 때에만 실행됩니다.

Other Hints

APPEND 혹은 NOAPPEND

```
/*+ APPEND */
```

- 직렬 모드 데이터베이스에서 Direct INSERT를 실행하게 합니다.
- Enterprise Edition 이 아닌 데이터베이스의 기본 모드는 직렬 모드입니다. 이러한 직렬 모드 데이터 베이스에서의 INSERT 작업은 Conventional 를 기본값으로 하고 병렬 처리 시에는 Direct INSERT를 기본값으로 합니다.

CACHE 혹은 NOCACHE

```
/*+ CACHE (table) */
```

풀 테이블 스캔의 사용 시, 테이블에서 일거온 블럭을 버퍼의 LRU 리스트의 MRU 쪽에 위치시킵니다. 작은 테이블의 사용시 유용합니다.

CURSOR_SHARING_EXACT

```
/*+ CURSOR_SHARING_EXACT */
```

- 바인드 변수 값의 교체를 불가능하게 합니다.
- 기본적으로 CURSOR_SHARING 파라미터를 사용하여, 안전하다고 판단될 시 SQL 내의 바인드 변수 값을 교체할 수 있게 되어 있습니다.

DYNAMIC_SAMPLING

```
/*+ DYNAMIC_SAMPLING ([table] n) */
```

- 해당 객체의 Selectivity 와 Cardinality 에 대한 보다 자세한 정보를 자동으로 생성시켜 실행합니다.
- 값은 0부터 10까지 지정할 수 있으며, 높을 수록 보다 자세한 정보를 생성하게 됩니다. 테이블에 해당 값을 지정하지 않았을 경우, 기본 값은 CURSOR 레벨의 값이 쓰여집니다.

UNNEST

```
/*+ UNNEST */
```

- 서브 쿼리 블럭에 대해 인증성 만을 검사하게 합니다.
- 인증이 되었다면 그 이상의 검증 작업없이 서브쿼리에 대한 UNNESTING의 설정을 가능하게 합니다.

NO_UNNEST

```
/*+ NO_UNNEST */
```

- 해당 서브 쿼리 블럭의 UNNESTING 설정의 사용을 방지합니다.

ORDERED_PREDICATES

```
/*+ ORDERED_PREDICATE */
```

- 옵티마이저에 의한 조인 관계의 Cost를 산출하기 위해 미리 정해둔 조인 관계별 실행 순서의 사용을 방지합니다.
- 인덱스 키를 사용한 조인 관계들은 제외됩니다.
- 이 힌트는 쿼리의 WHERE절에 사용합십시오.

참고

Nested Loop

- 테이블의 인덱스끼리 inner-outer 루프를 형성하여 결과를 쿼리하는 방식입니다.

- 제일 많은 유형의 실행계획입니다.

Sort Merge

- 쿼리의 결과가 많은 양의 데이터를 읽는 경우, 테이블들을 각각 full-scan하여 같은 키값을 갖는 데이터끼리 조인하여 실행합니다.
- Sort-Merge 방식은 많은 메모리와 디스크 I/O를 필요로 하기 때문에, sqlplus를 실행하여 주체의 메모리/CPU/디스크 스펙에 많은 영향을 받습니다.

Hash Join

- 한 테이블은 매우 많은 Row를 갖고, 다른 한 테이블은 매우 적은 Row를 가질 때, 해쉬 알고리즘에 의해 큰 테이블을 여러개의 버켓으로 나누어 쿼리를 수행하는 방식입니다. 작은 테이블은 인덱스를 태우는 것보다 full-scan을 하는 것이 유리할 때 사용됩니다.

Ref

- http://theone79.tistory.com/entry/%EC%98%A4%EB%9D%BC%ED%81%B4-hint-%EC%82%AC%EC%9A%A9%EB%B2%95
- http://annehouse.tistory.com/413
- http://blog.naver.com/kyumi0705/20130933394

관련 문서

- OPEN
- OPEN