10. 쿼리 최적화의 기본사항

(1) 옵티마이저에 대한 이해

SQL 구문분석(Parse)과 최적화

- 사용자로부터 SQL을 전달받으면 SQL Parser가 구문분석을 진행합니다.
- SQL 옵티마이저가 SQL 최적화를 진행합니다.
  - SQL 옵티마이저는 사용자가 원하는 작업을 가장 효율적으로 수행할 수 있도록 최적의 데이터 액세스 경로를 선택해 주는 DBMS의 핵심엔진입니다.
  - 미리 수집한 시스템 및 오브젝트 통계정보를 바탕으로 다양한 실행경로를 생성해서 비교한 후 가장 효율적인 하나를 선택합니다.
- Row-Source Generation

옵티마이저의 종류

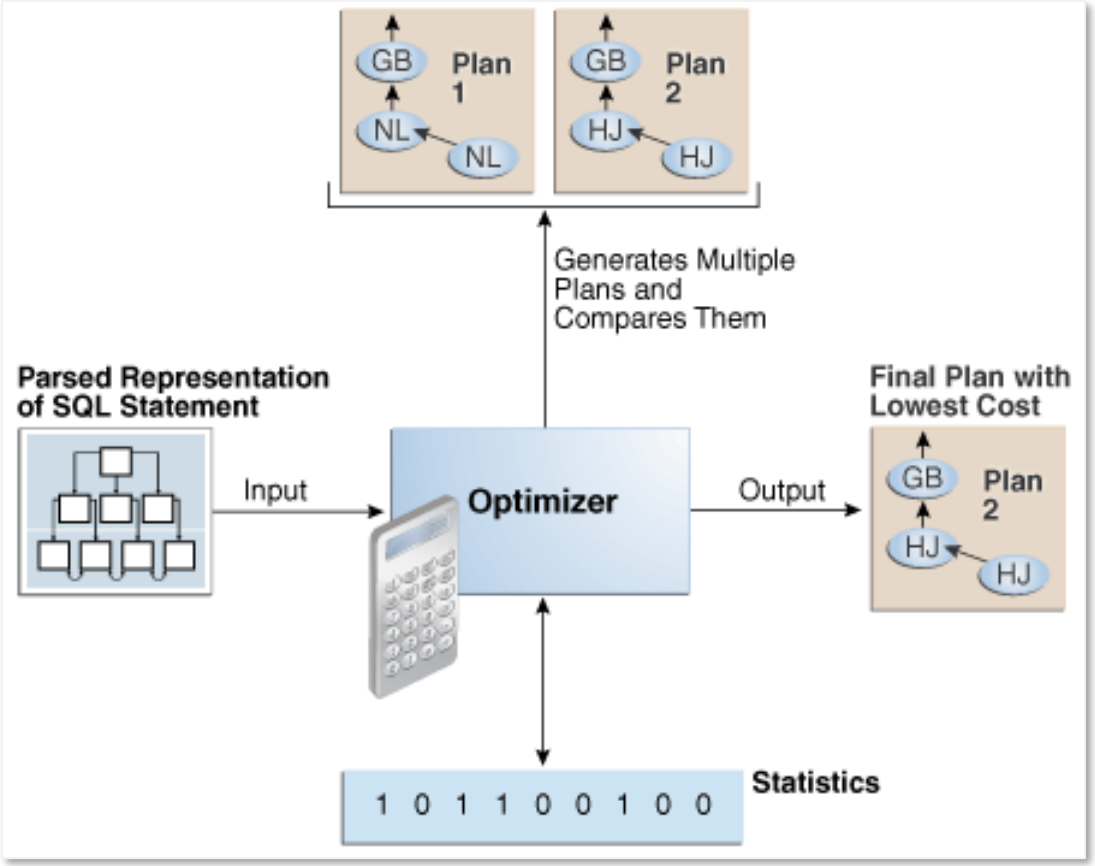
- 비용기반 옵티마이저(Cost-Based Optimizer, CBO)
  - 사용자 쿼리를 위해 후보군 실행 계획을 도출한 후 통계정보를 이용해 각 실행계획의 예상 비용을 산정한 결과 가장 낮은 비용의 실행계획을 선택합니다.
  - CBO는 데이터량, 컬럼 값의 수, 컬럼 값 분포, 인덱스 높이, 클러스터링 팩터 등의 통계정보를 사용합니다.
- 규칙기반 옵티마이저(Rule_Based Optimizer, RBO)
  - 각 엑세스 경로에 대한 우선 순위 규칙에 따라 실행계획을 만드는 옵티마이저 입니다.
  - 인덱스 구조, 연산자, 조건절 형태 등이 순위를 결정하는 주요 요소 입니다.
- Oracle 12c 이상에서는 RBO를 완전히 제거했습니다.

SQL 옵티마이저

- 쿼리 옵티마이저는 SQL 문이 요청된 데이터에 액세스하는 가장 효율적인 방법을 결정하는 내장 데이터베이스 소프트웨어입니다.
- 사용자가 구조화된 질의언어(SQL)로 결과집합 요구 시, 이를 생성하는데 필요한 처리경로를 옵티마이저가 자동으로 생성합니다.
- 옵티마이저가 생성한 SQL 처리경로를 실행계획(Execution Plan)이라고 합니다.
- 옵티마이저의 최적화 단계
  1) 사용자로부터 전달 받은 쿼리를 수행하는데 사용될 후보군인 실행 계획들을 찾는다.
  2) 데이터 딕셔너리에 미리 수집한 통계정보를 이용해서 각 실행 계획의 예상비용을 산정한다.
  3) 최저 비용(cost)을 나타내는 실행 계획을 선택한다.

실행계획 (Execution Plans)

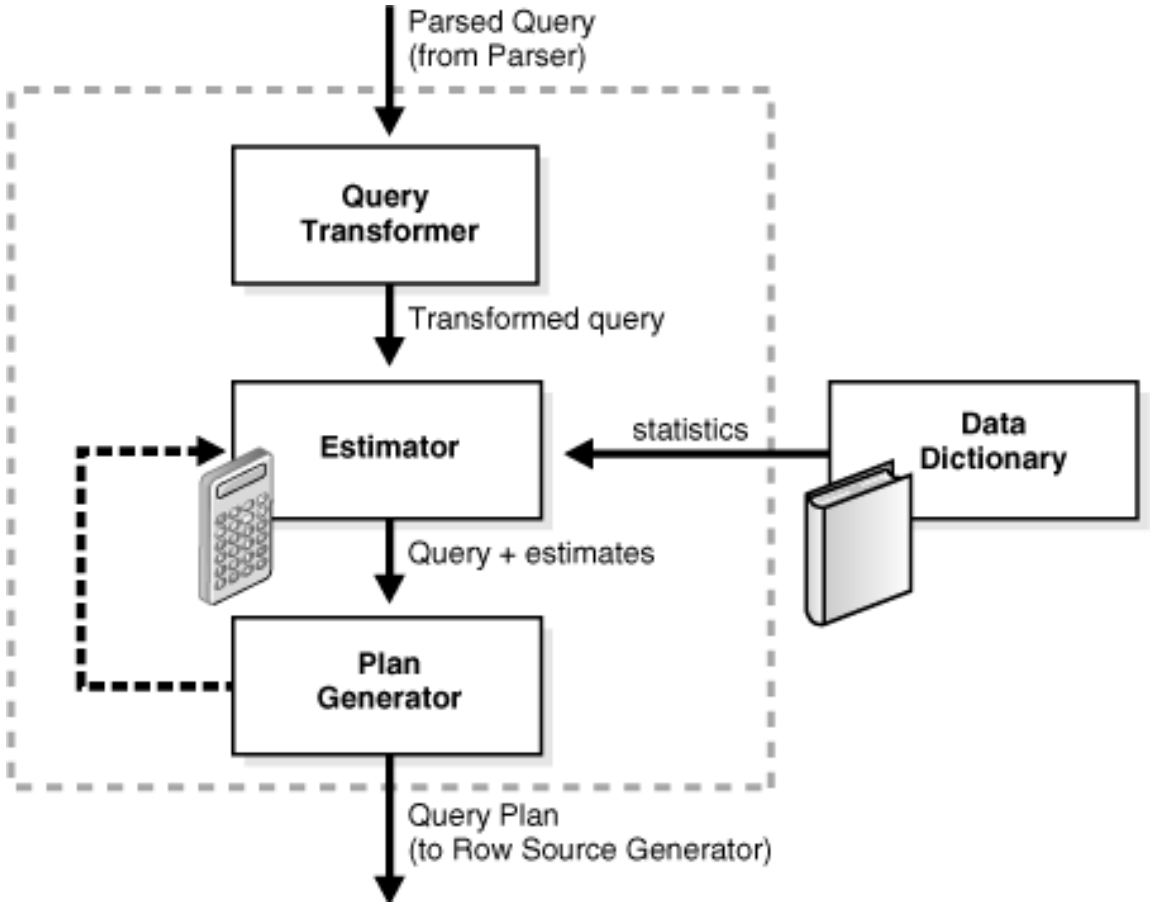
- 실행 계획은 SQL 문에 권장되는 실행 방법을 설명합니다.
- 실행 계획은 Oracle Database가 SQL 문을 실행하는 데 사용하는 단계의 조합을 보여줍니다.
  - 각 단계는 데이터베이스에서 데이터 행을 물리적으로 검색하거나 명령문을 실행하는 사용자를 위해 데이터 행을 준비합니다.
  - 0번 줄에 표시된 전체 계획과 각 개별 작업의 비용을 표시합니다.
  - 비용은 실행 계획이 계획을 비교할 수 있도록 표시하는 내부 단위입니다.



옵티마이저는 입력 SQL 문에 대해 두 가지 가능한 실행 계획을 생성하고, 통계를 사용하여 비용을 추정하고, 비용을 비교한 다음, 비용이 가장 낮은 계획을 선택합니다.

옵티마이저 구성요소

- 옵티마이저는 Query Transformer, Estimator 그리고 Plan Generator 세 가지로 구성됩니다.



Query Transformer (쿼리변환기)	옵티마이저는 더 나은 실행 계획을 생성할 수 있도록 쿼리 형식을 변경하는 것이 도움이 되는지 판단합니다.
Estimator (비용추정기)	옵티마이저는 데이터 사전의 통계를 기반으로 각 계획의 비용을 추정합니다.
Plan Generator (실행계획 생성기)	옵티마이저는 계획의 비용을 비교하고 가장 비용이 낮은 실행 계획을 선택하여 행 소스 생성기로 전달합니다.

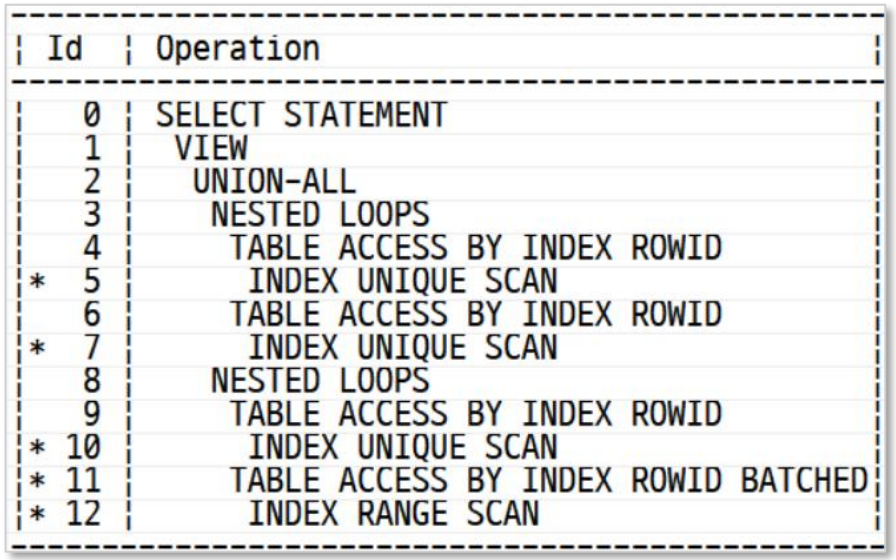
쿼리 변환

- 사용자가 작성한 SQL 문을 Oracle 옵티마이저가 더 효율적인 논리 구조로 내부적으로 재작성하는 과정입니다.
- 쿼리변환의 중요성
  - 사용자의 비효율적인 SQL을 자동으로 최적화
    - 예: OR 조건 → UNION ALL, WHERE EXISTS → JOIN 등
    - 개발자의 쿼리 재작성 없이 성능 개선 ➔ "자동 SQL 튜닝"의 기반
  - 서브쿼리, 뷰, 인라인 뷰 등 복잡한 SQL의 실행계획 생성을 가능하게 함
    - 복잡한 구조의 SQL을 옵티마이저가 이해하고 단순화하여 실행 불가능하거나 비효율적인 계획을 피할 수 있음
  - 인덱스 활용 가능성 증가
    - 예: 단일 OR 조건이면 인덱스 못 쓰는 경우, UNION ALL로 변환 후 각 분기에서 인덱스 활용 가능
  - 데이터웨어하우스 최적화 ➔ 분석시스템에서 성능저하 방지

쿼리변환 예시

- 다음 명령은 Oracle 옵티마이저 쿼리변환의 OR-expansion을 통해 인덱스 사용이 불가능한 쿼리를 각 테이블의 인덱스를 사용하는 쿼리로 변환합니다.

```
SELECT *
FROM   employees e, departments d
WHERE  (e.email='SSTILES' OR d.department_name='Treasury')
AND    e.department_id = d.department_id;
```



```
| Id  | Operation                              |
|   0 | SELECT STATEMENT                       |
|   1 |  VIEW                                  |
|   2 |   UNION-ALL                            |
|   3 |    NESTED LOOPS                        |
|   4 |     TABLE ACCESS BY INDEX ROWID        |
| * 5 |      INDEX UNIQUE SCAN                 |
|   6 |     TABLE ACCESS BY INDEX ROWID        |
| * 7 |      INDEX UNIQUE SCAN                 |
|   8 |    NESTED LOOPS                        |
|   9 |     TABLE ACCESS BY INDEX ROWID        |
| * 10|      INDEX UNIQUE SCAN                 |
| * 11|     TABLE ACCESS BY INDEX ROWID BATCHED|
| * 12|      INDEX RANGE SCAN                  |
```

```
SELECT *
FROM   employees e, departments d
WHERE  e.email = 'SSTILES'
AND    e.department_id = d.department_id
UNION ALL
SELECT *
FROM   employees e, departments d
WHERE  d.department_name = 'Treasury'
AND    e.department_id = d.department_id;
```

비용추정기(Estimator)

- 추정기는 주어진 실행 계획의 전체 비용을 결정하는 옵티마이저의 구성 요소입니다.
- 추정기는 비용을 결정하기 위해 세 가지 다른 측정값을 사용합니다.
  - Selectivity (선택성)
    - 쿼리가 행 집합에서 선택하는 행의 백분율로, 0은 행이 없음을 의미하고 1은 모든 행을 의미합니다.
    - 조건자는 선택성 값이 0에 가까울수록 선택성이 높아지고, 값이 1에 가까울수록 선택성이 낮아집니다.
  - Cardinality (기수)
    - 실행 계획에서 각 연산이 반환하는 행의 수입니다.
    - 추정기는 DBMS_STATS에서 수집한 테이블 통계에서 카디널리티를 도출하거나, 조건자(필터, 조인 등), DISTINCT 또는 GROUP BY 연산 등의 영향을 고려하여 카디널리티를 도출합니다.
  - Cost(비용)
    - 쿼리 옵티마이저는 디스크 I/O, CPU 사용량 및 메모리 사용량을 작업 단위로 사용하여 비용을 측정 합니다.

비용추정기(Estimator) : Selectivity

- 옵티마이저는 통계 사용 가능 여부에 따라 선택성을 추정합니다.
- 통계 사용 불가
  - 옵티마이저는 OPTIMIZER_DYNAMIC_SAMPLING 초기화 매개변수 값에 따라 동적 통계 또는 내부 기본값을 사용합니다.
- 사용 가능한 통계
  - 추정기는 통계를 사용하여 선택성을 추정합니다.
  - 열에 히스토그램이 있는 경우, 추정기는 고유 값의 개수 대신 히스토그램을 사용합니다.
  - 히스토그램은 열에 있는 다양한 값의 분포를 포착하므로, 특히 데이터 비대칭이 있는 열에 대해 더 나은 선택성 추정값을 제공합니다.

비용추정기(Estimator) : Cardinality

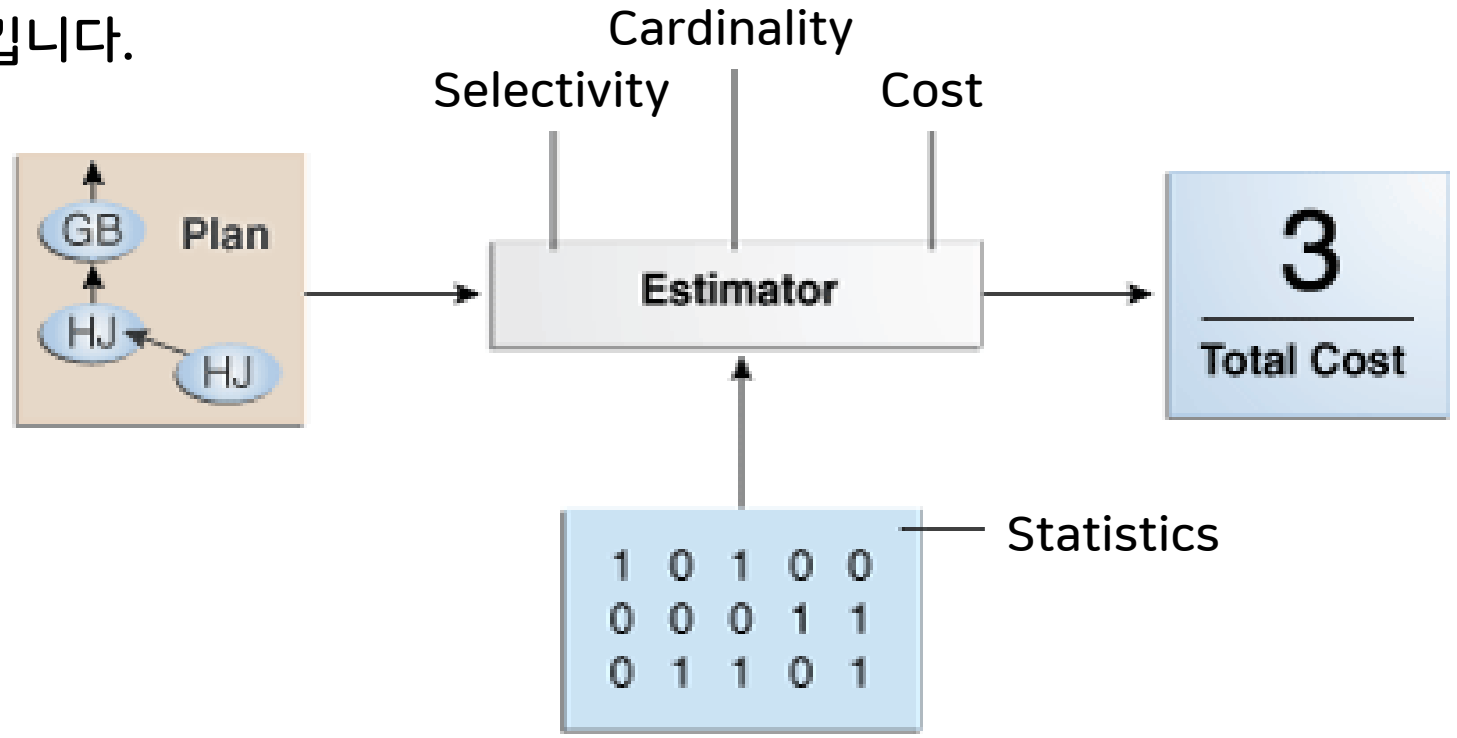
- 카디널리티는 실행 계획에서 각 작업이 반환하는 행 수입니다.
  - 전체 테이블 스캔에서 반환되는 행 수에 대한 옵티마이저 추정값이 100이면 이 작업의 카디널리티 추정값도 100입니다.
  - employees 테이블에는 107개의 행이 있고 salary 열의 고유 값 개수는 58개이면 옵티마이저는 107/58=1.84 공식을 사용하여 결과 집합의 카디널리티를 2로 추정합니다.
  - 카디널리티 추정값은 실행 계획의 행 열에 나타납니다.
- 카디널리티는 옵티마이저가 정렬 또는 조인 비용을 결정할 때 중요합니다.
  - 예를 들어, employees 테이블과 departments 테이블의 중첩 루프 조인에서 employees 테이블의 행 개수는 데이터베이스가 departments 테이블을 탐색해야 하는 빈도를 결정합니다.

비용추정기(Estimator): Cost(비용)

- 옵티마이저 비용 모델은 쿼리가 사용할 것으로 예상되는 머신 리소스를 고려합니다.
- 비용은 계획의 예상 리소스 사용량을 나타내는 내부 수치 측정값입니다.
- 비용은 옵티마이저 환경의 쿼리에 따라 달라지며, 옵티마이저는 비용을 추정하기 위해 다음과 같은 요소를 고려합니다.
  - 예상 I/O, CPU 및 메모리를 포함한 시스템 리소스
  - 예상 반환 행 수(카디널리티)
  - 초기 데이터 세트의 크기
  - 데이터 분포
  - 액세스 구조

통계를 사용한 비용 추정

- 다음 그림에서 볼 수 있듯이, 통계를 사용할 수 있는 경우 추정기는 해당 통계를 사용하여 측정값을 계산합니다.
- 통계는 측정값의 정확도를 향상시킵니다.



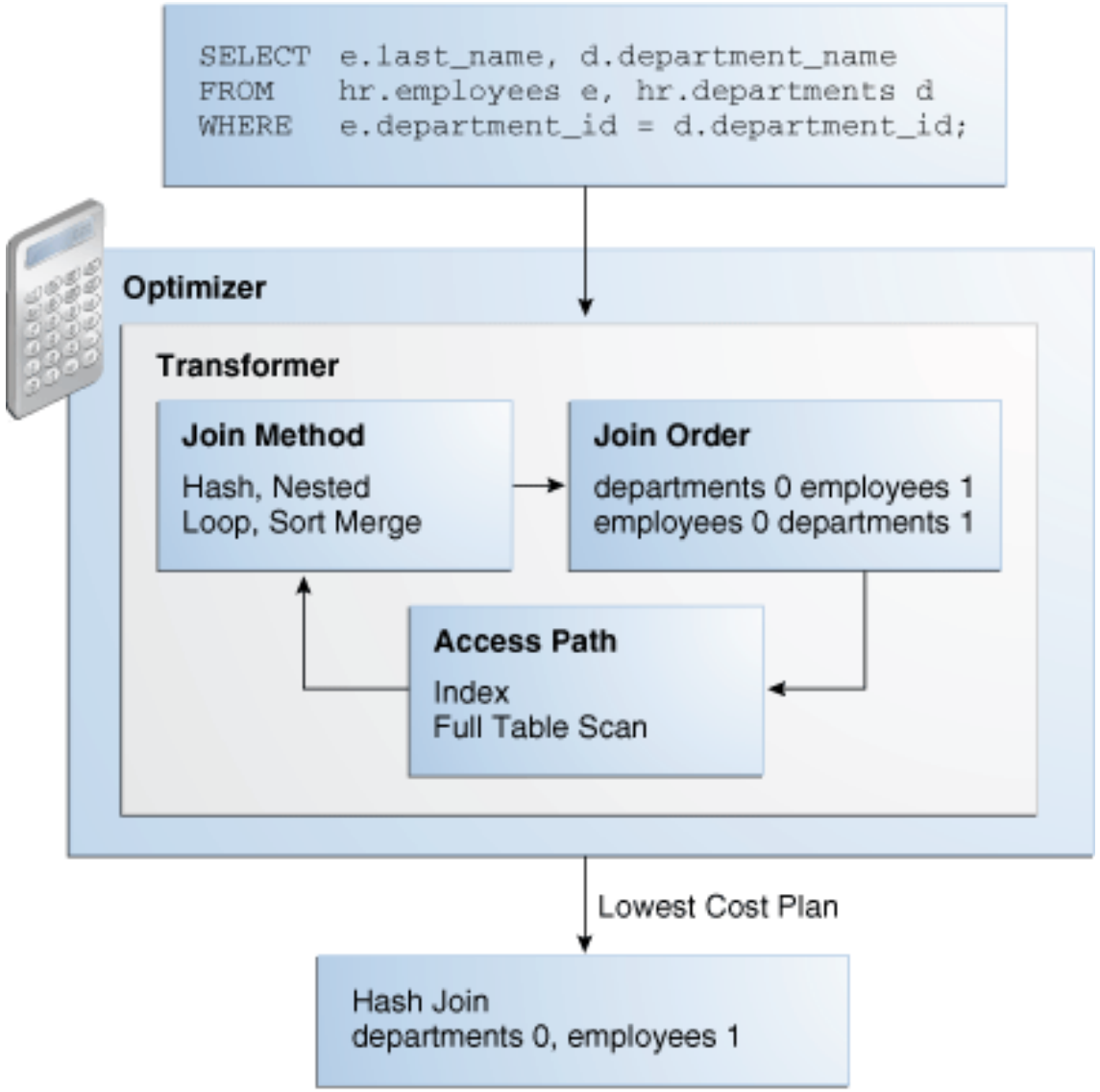
Cost Example

- 다음 쿼리의 경우, 추정기는 선택성, 추정된 카디널리티(총 10개 행의 결과), 그리고 비용 측정값을 사용하여 총 비용 추정값 3을 생성합니다.

```
-------------------------------------------------------------------------------
|Id| Operation                    |Name          |Rows|Bytes|Cost %CPU|Time|
-------------------------------------------------------------------------------
| 0| SELECT STATEMENT             |              |10|250|3 (0)|00:00:01|
| 1|  NESTED LOOPS                |              |  |   |     |        |
| 2|   NESTED LOOPS               |              |10|250|3 (0)|00:00:01|
|*3|    TABLE ACCESS FULL         |DEPARTMENTS   | 1|  7|2 (0)|00:00:01|
|*4|    INDEX RANGE SCAN          |EMP_DEPARTMENT_IX|10|  |0 (0)|00:00:01|
| 5|   TABLE ACCESS BY INDEX ROWID|EMPLOYEES     |10|180|1 (0)|00:00:01|
-------------------------------------------------------------------------------
```

Plan Generator(계획생성기)

- 플랜 생성기는 다양한 액세스 경로, 조인 방법 및 조인 순서를 시도하여 쿼리 블록에 대한 다양한 플랜을 탐색합니다.
- 데이터베이스가 동일한 결과를 생성하는 데 사용할 수 있는 다양한 플랜 가운데 옵티마이저는 가장 낮은 비용을 가진 플랜을 선택합니다.
- 그림은 옵티마이저가 입력 쿼리에 대해 다양한 플랜을 테스트하는 모습을 보여줍니다.



옵티마이저 모드

- Oracle 옵티마이저는 SQL 문에 대한 최적의 실행 계획을 선택하기 위해 동작하며, 옵티마이저 모드(Optimizer Mode)는 이 실행 계획을 선택하는 전략에 영향을 줍니다.
- ALL_ROWS : 전체 결과 집합을 빠르게 처리하는 계획을 선택 (배치 처리용)
  - 쿼리 최종 결과집합을 끝까지 읽는 것을 전제로, 시스템 리소스(I/O, CPU, 메모리 등)를 가장 적게 사용하는 실행 계획을 선택합니다.
  - Oracle을 포함해 대부분 DBMS의 기본 옵티마이저 모드는 전체 처리속도 최적화에 맞춰져 있습니다.
- FIRST_ROWS_n : 처음 n 개의 행을 빠르게 반환하는 계획을 선택 (인터랙티브 처리용)
  - 최초 응답속도에 최적화 된 모드입니다.
  - 사용자가 전체 결과집합 중 처음 n개 로우만 읽고 멈추는 것을 전제로 가장 빠른 응답 속도를 낼 수 있는 실행계획을 선택합니다. (n은 1, 10, 100, 1000 중 가능)

(2) 통계 수집

옵티마이저와 객체통계

- 옵티마이저는 객체의 통계 정보를 기반으로 실행계획을 결정합니다.
- 통계 정보가 없다면 옵티마이저는 부정확한 추정을 통해 비효율적인 실행계획을 생성할 수 있습니다.
- 객체 통계는 테이블 통계, 인덱스 통계, 컬럼 통계(히스토그램 포함) 입니다.
- 통계 수집 방법
  - ANALYZE 명령
  - DBMS_STATS 패키지
- 새 통계를 수집하면 데이터 딕셔너리에 있는 기존 통계를 덮어쓰고 공유 풀에서 관련된 실행 계획을 비워집니다.

ANALYZE 명령

- ANALYZE 명령으로 객체에 대한 통계를 수집합니다.
  - 자주 수정되는 테이블에 대한 새 통계를 주기적으로 수집하여 옵티마이저가 최신 정보를 사용하도록 합니다.
- 구문

```
ANALYZE {TABLE table_name ¦ INDEX index_name}
{COMPUTE STATISTICS ¦ ESTIMATE STATISTICS[SAMPLE size {ROW ¦ PERCENT}] ¦ DELETE STATISTICS}
```

  - COMPUTE STATISTICS: 대상 오브젝트에 대한 가장 정확한 통계 정보를 계산합니다.
  - ESTIMATE STATISTICS [SAMPLE size {ROW | PERCENT}]: 대상 오브젝트에 대한 통계 정보를 추정합니다. SAMPLE 절을 사용하여 샘플링 비율 또는 행 수를 지정할 수 있습니다.
  - DELETE STATISTICS : 통계 정보를 삭제합니다.

ANALYZE 예제

- 테이블 통계

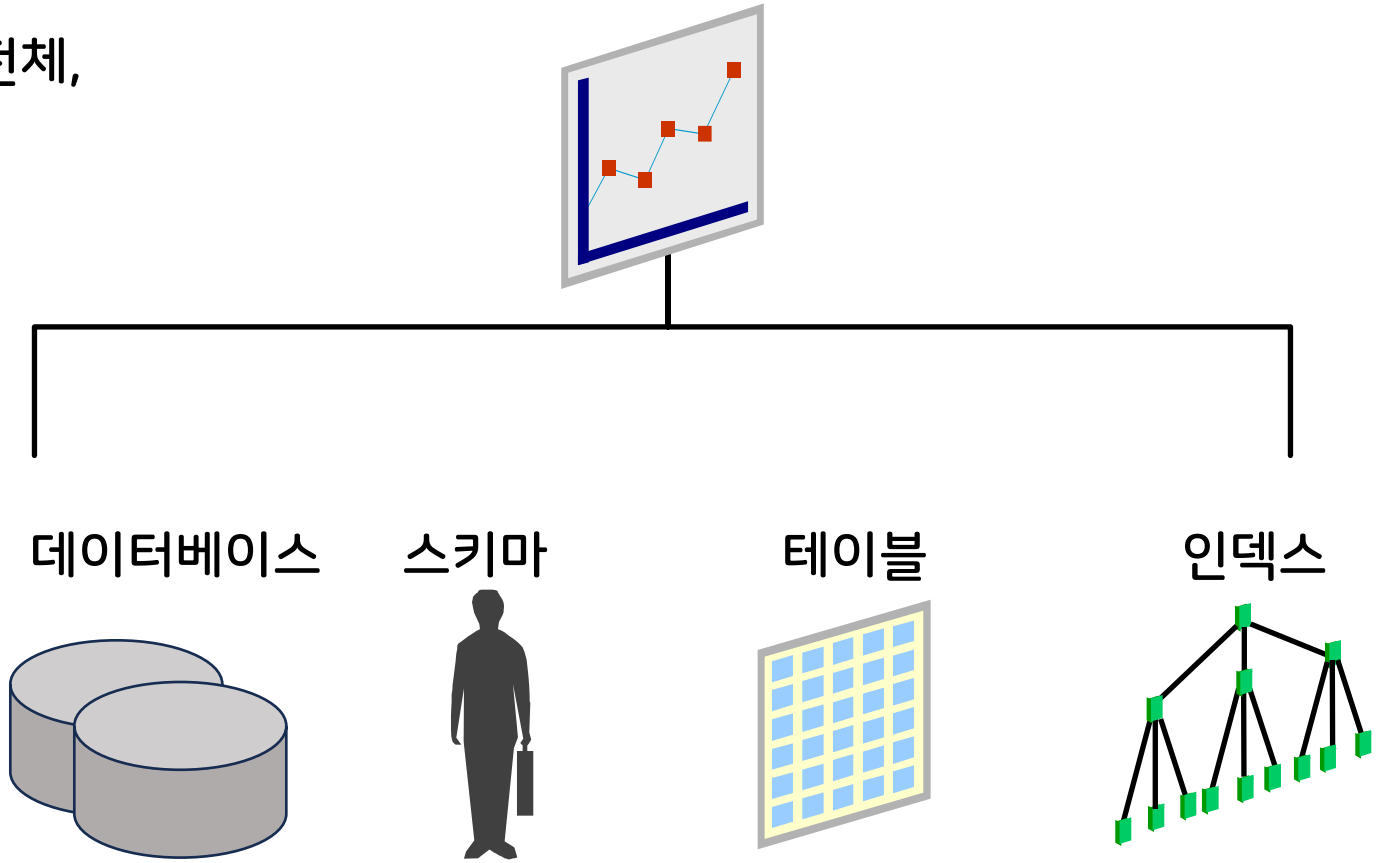
```
ANALYZE TABLE employees COMPUTE STATISTICS;
ANALYZE TABLE employees DELETE STATISTICS;
```

- 인덱스 통계

```
ANALYZE INDEX emp_last_name_idx COMPUTE STATISTICS;
ANALYZE TABLE emp_last_name_idx DELETE STATISTICS;
```

DBMS_STATS 패키지

- 오라클 옵티마이저가 정확한 실행계획을 생성하기 위해 ANALYZE 명령 대신 사용하는 객체 통계 (Statistics) 수집 패키지입니다.
- DBMS_STATS는 테이블, 인덱스, 스키마 전체, 데이터베이스 전체 등 다양한 범위에서 통계를 수집할 수 있습니다.



DBMS_STATS의 사용

- DBMS_STATS는 주로 다음의 프로시저를 사용합니다.
  - GATHER_TABLE_STATS: 테이블 단위
  - GATHER_SCHEMA_STATS: 스키마 전체
  - GATHER_DATABASE_STATS: 데이터베이스 전체
  - GATHER_INDEX_STATS: 인덱스 단위
  - DELETE_*_STATS : 통계삭제
- 예제

```
EXEC dbms_stats.gather_table_stats ('c##hr', 'employees');
EXEC dbms_stats.gather_schema_stats('c##hr');
EXEC dbms_stats.delete_schema_stats('c##hr');
```

테이블 통계

- 다음은 DBA_TABLES, ALL_TABLES, USER_TABLES을 통해 볼 수 있는 수집된 테이블 통계입니다.

딕셔너리 열 이름	수집 항목 설명
NUM_ROWS	테이블의 총 레코드 수
BLOCKS	테이블이 실제로 사용하는 데이터 블록 수
EMPTY_BLOCKS	빈(사용되지 않은) 블록 수
AVG_ROW_LEN	평균 행 길이 (바이트)
AVG_SPACE	사용 가능한 평균 공간 (바이트 단위)
LAST_ANALYZED	마지막으로 통계 수집한 날짜

인덱스 통계

- 다음은 DBA_INDEXES, ALL_INDEXES, USER_INDEXES 뷰에서 확인할 수 있는 인덱스 통계 정보 입니다.

딕셔너리 열 이름	인덱스 통계 항목 설명
BLEVEL	루트에서 리프까지의 트리 높이 (B-Tree 인덱스 레벨)
LEAF_BLOCKS	최하위 블록(리프블록) 수
DISTINCT_KEYS	구분 키 수 (인덱스 내의 고유 키 값)
AVG_LEAF_BLOCKS_PER_KEY	하나의 키 값에 매핑되는 리프 블록 수의 평균
AVG_DATA_BLOCKS_PER_KEY	하나의 키 값에 매핑되는 테이블 데이터 블록 수의 평균
NUM_ROWS	인덱스 항목(엔트리) 수
CLUSTERING_FACTOR	인덱스 키 순서와 테이블 데이터의 정렬 정도
LAST_ANALYZED	마지막으로 통계가 수집된 날짜

테이블과 인덱스 통계 보기

- 테이블 통계

```
select num_rows, blocks, empty_blocks, avg_space,avg_row_len, sample_size
from   user_tables
where  table_name = 'EMPLOYEES';
```

- 인덱스 통계

```
select uniqueness, blevel ,leaf_blocks, distinct_keys, clustering_factor
from   user_indexes
where  index_name = 'PRODUCTS_PK';
```

컬럼 통계

- 컬럼통계는 테이블 통계 수집때 함께 수집되며 DBA_TAB_COL_STATISTICS, ALL_TAB_COL_STATISTICS, USER_TAB_COL_STATISTICS 뷰에서 확인할 수 있습니다

컬럼 통계 항목	설명
NUM_DISTINCT	해당 컬럼에서 나타나는 고유값의 개수
DENSITY	선택도(Selectivity)의 역수로, 낮을수록 인덱스 활용에 유리
AVG_COL_LEN	해당 컬럼의 평균 데이터 길이 (바이트 단위)
LOW_VALUE	컬럼에서 옵티마이저가 파악한 가장 작은 값 (RAW 형식으로 저장됨)
HIGH_VALUE	컬럼에서 옵티마이저가 파악한 가장 큰 값 (RAW 형식으로 저장됨)
NUM_NULLS	컬럼에 저장된 NULL 값의 개수

컬럼 통계와 히스토그램의 필요성

- Oracle 옵티마이저는 SQL 실행계획을 선택할 때 컬럼 통계를 기반으로 선택도(Selectivity) 를 계산합니다.
- 기본 통계 항목:
  - NUM_DISTINCT: 고유 값의 수
  - DENSITY: 선택도 추정에 사용
  - AVG_COL_LEN: 평균 데이터 길이
  - NUM_NULLS: NULL 값 개수
- 일반적인 선택도는 “조건을 만족하는 행 수 / 전체 행 수” 입니다.
- 컬럼 값의 분포가 불균형한 경우, 평균적인 통계로는 정확한 선택도 예측이 어려울 수 있으며, 히스토그램(Histogram) 이 사용되어야 합니다.

히스토그램의 개념과 유형

- 히스토그램은 컬럼 값의 분포를 더 정밀하게 표현하는 통계입니다.
  - 특정 값이 자주 등장하거나 (예: 'KOREA') 극소수의 값만 조건으로 자주 조회될 경우 일반 통계로는 선택도 계산이 부정확할 수 있습니다.
- Oracle 히스토그램 유형

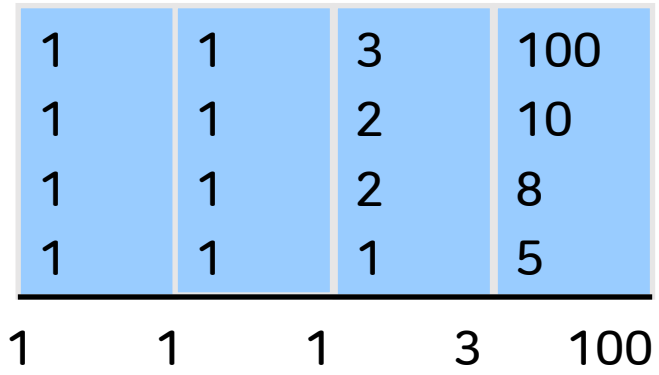
히스토그램 유형	설명
HEIGHT BALANCED(도수분포)	구간마다 동일한 row 수(값 별로 빈도수 저장)
FREQUENCY(높이균형)	각 버킷의 높이가 동일하도록 데이터 분포 관리
TOP-FREQUENCY(상위도수분포)	많은 레코드를 가진 상위 n개 값에 대한 빈도수 저장(12c 이상)
HYBRID	도수분포와 높이 균형 히스토그램의 특성 결합(12c 이상)

너비균등 vs 높이 균등 히스토그램

너비 균등

높이 균등 히스토그램이 데이터 분포를 더 잘 보여 줍니다.

높이 균등



히스토그램 통계: 예제

- PRODUCTS 테이블과 PROD_LIST_PRICE 열에 대한 통계를 생성합니다. 최대 버킷 수는 50개입니다.

```
analyze table products compute statistics
for table for columns prod_list_price
size 50;
```

- 버킷 수를 지정하지 않고 동일한 열에 대한 통계를 다시 계산합니다.

```
analyze table products compute statistics
for columns prod_list_price;
```

히스토그램 통계: 예제

- PRODUCTS 테이블과 PROD_LIST_PRICE 열에 대한 통계를 생성합니다.

  DBMS_STATS.GATHER_TABLE_STATS를 사용하여 최대 버킷 수는 50개로 생성합니다.

```
EXECUTE DBMS_STATS.GATHER_TABLE_STATS (‘c##sh','PRODUCTS’
            , METHOD_OPT=>'FOR COLUMNS SIZE 50 prod_list_price');
```

  - 테이블 통계 수집 시 method_opt 파라미터를 지정합니다.
  - SIZE 키워드는 히스토그램에 대한 최대 버킷 수를 선 언합니다.
  - DBMS_STATS 패키지를 사용하여 SIZE를 “auto”로 지정하면 데이터베이스가 히스토그램을 필요로 하는 열을 자동으로 결정합니다.

히스토그램 정보 보기

```
SELECT table_name, column_name, endpoint_number, endpoint_value,
       endpoint_repeat_count, histogram_type
FROM user_histograms
WHERE table_name = ‘PRODUCTS' AND column_name = 'PROD_LIST_PRICE '
ORDER BY endpoint_number;
```

컬럼명	설명
ENDPOINT_NUMBER	히스토그램 버킷 번호
ENDPOINT_VALUE	정렬 기준값 (RAW 포맷, 내부 코드화됨)
ENDPOINT_REPEAT_COUNT	동일 값이 얼마나 반복되는지
HISTOGRAM_TYPE	히스토그램 유형

(3) SQL 진단 도구

실행 계획 표시

- 옵티마이저가 예상한 계획을 선택하는지 확인하거나, 테이블에 인덱스를 생성했을 때의 영향을 파악하기위해 실행 계획을 확인합니다.
- 실행 계획을 표시하는 가장 일반적으로 사용되는 도구는 다음과 같습니다.
  - EXPLAIN PLAN
    - 실제로 SQL 문을 실행하지 않고도 옵티마이저가 SQL 문을 실행하는 데 사용할 실행 계획을 볼 수 있습니다.
  - AUTOTRACE
    - SQL*Plus의 이 AUTOTRACE명령은 쿼리 성능에 대한 실행 계획과 디스크 읽기 및 메모리 읽기와 같은 통계를 생성합니다.
  - V$SQL_PLAN및 관련 뷰
    - 실행된 SQL 문과 공유 풀에 있는 실행 계획에 대한 정보가 포함되어 있습니다.
  - DBMS_XPLAN
    - DBMS_XPLAN 패키지 메서드를 사용하면 EXPLAIN PLAN 명령과 V$SQL_PLAN 쿼리로 생성된 실행 계획을 표시합니다.

EXPLAIN PLAN

- EXPLAIN PLAN 을 사용하면 Oracle은 내부적으로 이 실행계획을 PLAN_TABLE이라는 테이블에 기록합니다.
- EXPLAIN PLAN 명령은 실제로 명령문을 실행하지는 않습니다.

```
EXPLAIN PLAN FOR
SELECT employee_id, last_name, first_name, department_name
from employees e, departments d
WHERE e.department_id = d.department_id
and last_name like 'T%'
ORDER BY last_name;
```

PLAN_TABLE 조회

- 다음 쿼리로 PLAN_TABLE에 저장된 실행 계획을 볼 수 있습니다.

```
SELECT * FROM TABLE(DBMS_XPLAN.DISPLAY(NULL, NULL, 'TIPYCAL'));
```

Id	Operation	Name	Rows	Bytes	Cost (%CPU)	Time
0	SELECT STATEMENT		5	185	5 (20)	00:00:01
1	SORT ORDER BY		5	185	5 (20)	00:00:01
* 2	HASH JOIN		5	185	4 (0)	00:00:01
3	TABLE ACCESS BY INDEX ROWID BATCHED	EMPLOYEES	5	105	2 (0)	00:00:01
* 4	INDEX RANGE SCAN	EMP_NAME_IX	5		1 (0)	00:00:01
5	VIEW	index$_join$_002	27	432	2 (0)	00:00:01
* 6	HASH JOIN					
7	INDEX FAST FULL SCAN	DEPARTMENT_NAME_UK	27	432	1 (0)	00:00:01
8	INDEX FAST FULL SCAN	DEPT_ID_PK	27	432	1 (0)	00:00:01

FORMAT 문자열	BASIC	가장 간단한 정보만 출력 (id, operation, object_name 등)
	TYPICAL	기본적인 실행 계획 + 예상 로우 수, 비용 등 (기본값)
	SERIAL	병렬 정보 생략하고 직렬 실행 전용 포맷으로 출력
	ALL	실행 계획에 포함될 수 있는 모든 항목 출력
	ADVANCED	옵티마이저 힌트 정보, Plan hash, Outline 정보까지 포함

실행 계획의 내용

- 실행 계획은 데이터베이스가 SQL 문을 실행하기 위해 수행하는 작업의 순서입니다.
- 행 소스 트리는 실행 계획의 핵심으로 다음 정보를 보여줍니다.
  - 명령문에서 참조하는 테이블의 조인 순서
  - 명령문에 언급된 각 테이블에 대한 액세스 경로
  - 명령문의 조인 연산에 영향을 받는 테이블의 조인 방법
  - 필터, 정렬 또는 집계와 같은 데이터 연산
- 행 소스 트리 외에도 계획 테이블에는 다음 정보가 포함됩니다.
  - 최적화(예: 각 연산의 비용 및 카디널리티)
  - 파티셔닝(예: 액세스되는 파티션 집합)
  - 병렬 실행(예: 조인 입력의 분산 방법)

V$SQL과 V$SQL_PLAN

- V$SQL_PLAN뷰는 최근에 실행된 커서에 대한 실행 계획을 검사하는 방법을 제공합니다.
- 특정 SQL의 HASH_VALUE와 ADDRESS는 V$SQL에서 확인가능합니다.

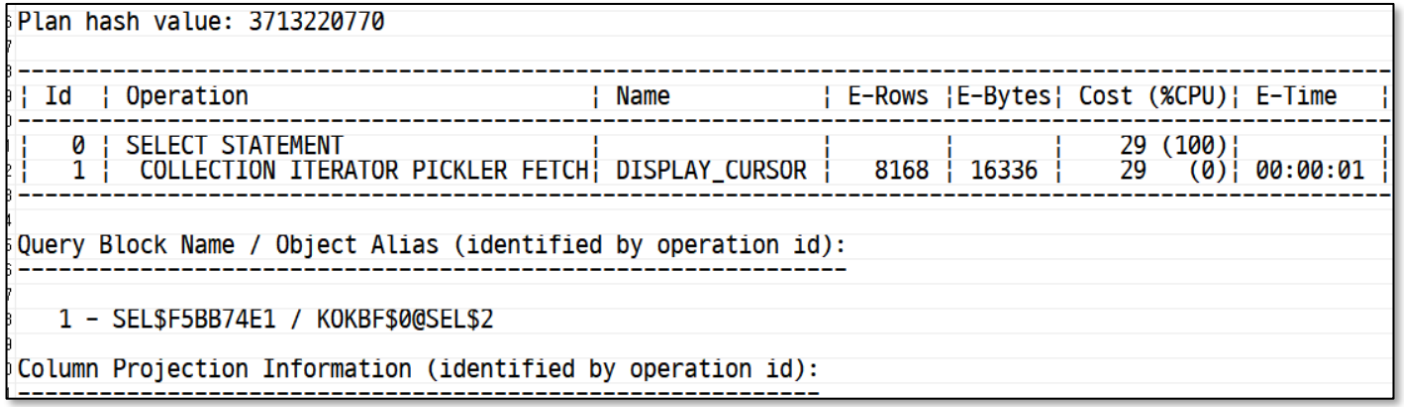
```
SELECT id, lpad (' ', depth) || operation operation, options , object_name ,
       optimizer, cost
FROM V$SQL_PLAN
WHERE hash_value = 1700806449
AND address    = '00007FF9A7402188'
START WITH id = 0
CONNECT BY
       (     prior id           = parent_id
         AND prior hash_value   = hash_value
         AND prior child_number = child_number
       )
ORDER SIBLINGS BY id, position;
```

DBMS_XPLAN

- 다음 명령은 Oracle에서 실제로 실행된 SQL의 실행 계획과 통계 정보를 확인할 수 있습니다.

```
SELECT * FROM TABLE(
  DBMS_XPLAN.DISPLAY_CURSOR(NULL, NULL, 'ALLSTATS LAST ALL +NOTE')
);
```

  - DBMS_XPLAN.DISPLAY_CURSOR: 실제 실행된 SQL 문에 대한 실행계획을 보여주는 Oracle 패키지 함수
  - NULL, NULL: 현재 세션에서 가장 최근에 실행한 SQL을 대상으로 지정
  - 'ALLSTATS LAST +NOTE': 실행계획과 함께 실제 통계 정보와 옵티마이저 메모도 함께 출력



```
Plan hash value: 3713220770

-----------------------------------------------------------------------------------------------------
| Id  | Operation                          | Name           | E-Rows |E-Bytes| Cost (%CPU)| E-Time   |
-----------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT                   |                |        |       |    29 (100)|          |
|   1 |  COLLECTION ITERATOR PICKLER FETCH | DISPLAY_CURSOR |   8168 | 16336 |    29   (0)| 00:00:01 |
-----------------------------------------------------------------------------------------------------

Query Block Name / Object Alias (identified by operation id):
-------------------------------------------------------------

   1 - SEL$F5BB74E1 / KOKBF$0@SEL$2

Column Projection Information (identified by operation id):
-----------------------------------------------------------
```

실시간 SQL 모니터링

- Oracle 데이터베이스의 실시간 SQL 모니터링 기능을 사용하면 SQL 문이 실행되는 동안 성능을 모니터링 할 수 있습니다.
- 기본적으로 SQL 모니터링은 문이 병렬로 실행되거나 단일 실행에서 CPU 또는 I/O 시간이 5초 이상 소모되면 자동으로 시작됩니다.
- 다음 뷰를 사용하면 모니터링 중인 실행에 대한 자세한 정보를 얻을 수 있습니다.
  - V$ACTIVE_SESSION_HISTORY
  - V$SESSION
  - V$SESSION_LONGOPS
  - V$SQL
  - V$SQL_PLAN

모니터링 정보 확인

- V$SQL_MONITOR 뷰를 통해 실시간 모니터링 중인 SQL의 목록 및 상태를 봅니다.

```
SELECT sql_id, status, sql_text, elapsed_time, cpu_time
  FROM v$sql_monitor
 WHERE sql_text LIKE 'SELECT /*+ MONITOR%';
```

- DBMS_SQLTUNE.REPORT_SQL_MONITO 함수는 SQL의 상세 실행계획, 단계별 자원 사용량 등을 리포트 형식으로 제공합니다.

```
SELECT * FROM table(DBMS_SQLTUNE.REPORT_SQL_MONITOR(
    sql_id      => 'your_sql_id',
    type        => 'HTML',
    report_level => 'ALL'));
```

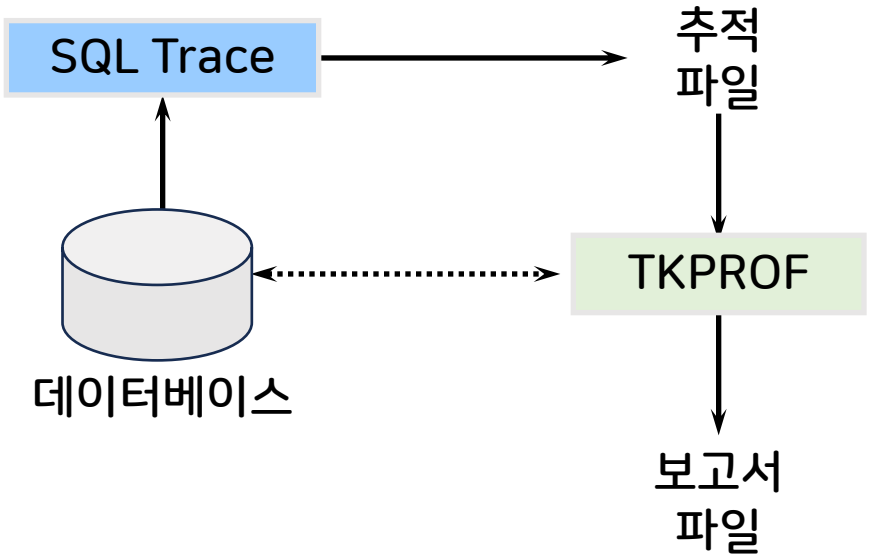
애플리케이션 추적

- 추적 파일은 SQL 성능 문제를 진단하는 데 유용할 수 있습니다.
  - SQL 추적 파일은 개별 SQL 문에 대한 성능 정보(구문 분석 횟수, 물리적 및 논리적 읽기 횟수, 라이브러리 캐시 누락 횟수 등)를 제공합니다.
  - EXPLAIN PLAN과 AUTOTRACE 결과로 문제점을 찾을 수 없는 경우 유용합니다.
- Oracle Database는 추적 파일을 분석하기 위한 다음과 같은 명령줄 도구를 제공합니다.
  - TKPROF
    - 이 유틸리티는 SQL Trace 기능에서 생성된 추적 파일을 입력으로 받아 형식화된 출력 파일(리포트)을 생성합니다.
  - trcsess
    - 이 유틸리티는 세션 ID, 클라이언트 ID, 서비스 ID 등의 기준에 따라 여러 추적 파일의 추적 출력을 통합합니다.

SQL Trace 기능

- 다음의 단계로 SQL Trace 툴을 사용합니다.

  1) 인스턴스 또는 세션 레벨에서 SQL_TRACE 파라미터를 설정합니다.
  2) 추적을 설정합니다.
  3) 응용 프로그램을 실행하고 완료되면 추적을 해제합니다.
  4) SQL Trace로 생성된 추적 파일에 대해 TKPROF로 형식을 지정합니다.
  5) 출력을 해석하고 필요하면 SQL 문을 튜닝합니다.



SQL Trace 설정

- 인스턴스인 경우 다음 파라미터를 설정합니다.

```
SQL_TRACE = TRUE
```

- 현재 혹은 임의의 세션인 경우:

```
ALTER SESSION SET sql_trace = true;
EXECUTE dbms_session.set_sql_trace(true);
EXECUTE dbms_system.set_sql_trace_in_session(session_id, serial_id, true);
```

- 튜닝을 완료한 후 TRUE 대신 FALSE를 사용하여 SQL Trace를 해제합니다.

추적 파일의 형식 지정

- 추적파일의 위치 조회

```
SELECT value FROM v$diag_info
WHERE name = 'Default Trace File';
```

- TKPROF 명령 형식

```
OS> tkprof tracefile outputfile [options]
```

- TKPROF 명령 예제:

```
OS> tkprof ora_901.trc run1.txt
OS> tkprof ora_901.trc run2.txt sys=no sort=execpu print=3
```

TKPROF 명령의 출력

- SQL 문의 텍스트 및 명령문의 추적 통계를 표시합니다.
  - 명령문의 추적 통계는 세 가지 SQL 처리 단계(Parse – Execute - Fetch)로 구분됩니다.
- 추적 통계의 7가지 범주

항목	설명
Count	프로시저(파싱, 실행, Fetch)가 수행된 횟수
CPU	SQL 실행 중 사용된 CPU 처리 시간 (초)
Elapsed	SQL의 전체 실행 시간 (CPU + 대기 시간 포함, 초)
Disk	물리적으로 디스크에서 블록을 읽은 횟수
Query	일관성 유지(CR)를 위한 논리적 버퍼 읽기 횟수
Current	현재 블록 수정을 위한 논리적 버퍼 읽기 횟수
Rows	SQL 수행으로 인출되거나 처리된 행(row)의 수

TKPROF 명령의 출력

- TKPROF 출력에는 다음 항목도 포함됩니다.
  - Recursive SQL 문
  - 라이브러리 캐시 실패
  - 마지막 구문 분석 사용자 ID
  - 실행 계획
  - 최적기 모드 또는 힌트

```
Trace file: xe_ora_49752.trc
Sort options: default

********************************************************************************
count    = number of times OCI procedure was executed
cpu      = cpu time in seconds executing
elapsed  = elapsed time in seconds executing
disk     = number of physical reads of buffers from disk
query    = number of buffers gotten for consistent read
current  = number of buffers gotten in current mode (usually for update)
rows     = number of rows processed by the fetch or execute call
********************************************************************************

SELECT employee_id, last_name, department_name
FROM employees JOIN departments
USING (department_id)

call     count       cpu    elapsed       disk      query    current        rows
------- ------  -------- ---------- ---------- ---------- ----------  ----------
Parse        1      0.01       0.01          0          0          0           0
Execute      1      0.00       0.00          0          0          0           0
Fetch        1      0.00       0.00          0         11          0          50
------- ------  -------- ---------- ---------- ---------- ----------  ----------
total        3      0.01       0.01          0         11          0          50

Misses in library cache during parse: 1
Optimizer mode: ALL_ROWS
Parsing user id: 105
********************************************************************************
```

TKPROF 출력 예제: 인덱스가 없는 경우

```
...
select cust_first_name, cust_last_name, cust_city, cust_state_province
from customers
where cust_last_name = 'Smith'

call     count       cpu    elapsed       disk      query    current        rows
------- ------  -------- ---------- ---------- ---------- ----------  ----------
Parse        3    801.14     800.00          0          0          1           0
Execute      3      0.00       0.00          0          0          0           0
Fetch       21   1802.59    9510.00         48       4835         10         234
------- ------  -------- ---------- ---------- ---------- ----------  ----------
total       27   2603.73   10310.00         48       4835         11         234

Misses in library cache during parse: 3
Optimizer goal: CHOOSE
Parsing user id: 44

Rows     Row Source Operation
-------  ---------------------------------------------------
     78  TABLE ACCESS FULL CUSTOMERS
...
```

TKPROF 출력 예제: 고유 인덱스가 있는 경우

```
select cust_first_name, cust_last_name, cust_city, cust_state_province
from customers
where cust_last_name = 'Smith'

call     count       cpu    elapsed       disk      query    current        rows
------- ------  -------- ---------- ---------- ---------- ----------  ----------
Parse        1      0.00       0.00          0          0          0           0
Execute      1      0.00       0.00          0          0          0           0
Fetch        7    100.14    1500.00          2         87          0          78
------- ------  -------- ---------- ---------- ---------- ----------  ----------
total        9    100.14    1500.00          2         87          0          78

Misses in library cache during parse: 1
Optimizer goal: CHOOSE
Parsing user id: 44

Rows     Row Source Operation
-------  ---------------------------------------------------
     78  TABLE ACCESS BY INDEX ROWID CUSTOMERS
     78   INDEX RANGE SCAN (object id 32172)
```

(4) Optimizer Hints

Optimizer Hints

- 힌트는 SQL 문의 주석을 통해 옵티마이저에 전달되는 명령입니다.
- 테스트 또는 개발 환경에서 힌트는 특정 액세스 경로의 성능을 테스트하는 데 유용합니다.
  - 예를 들어, 특정 인덱스가 특정 쿼리에 대해 더 선택적이라는 것을 알 수 있는 경우, 다음 예와 같이 힌트를 사용하여 옵티마이저가 더 나은 실행 계획을 사용하도록 지시할 수 있습니다.

```
SELECT /*+ INDEX (employees emp_department_ix) */ employee_id, department_id
FROM   employees
WHERE  department_id > 50;
```

  - 오타, 잘못된 인수, 충돌하는 힌트, 그리고 변환으로 인해 유효하지 않게 된 힌트 등으로 인해 데이터베이스에서 힌트를 사용하지 않는 경우가 있습니다.

힌트 규칙과 권장 사항

- 힌트 규칙
  - 명령문 블록에서 첫 SQL 키워드 바로 뒤에 힌트를 입력합니다.
  - 각 명령문 블록에는 단 하나의 힌트 주석만 있어야 하지만, 주석에는 여러 개의 힌트가 포함될 수 있습니다.
  - 힌트는 해당 힌트가 나타나는 명령문 블록에만 적용됩니다.
  - 명령문이 별칭을 사용하는 경우 힌트는 테이블 이름이 아닌 별칭을 참조해야 합니다.
- 권장 사항
  - 힌트를 유지 관리하기 위해서는 많은 로드가 발생하므로 힌트를 주의 깊게 사용하십시오.
  - 하드 코딩된 힌트의 유효성이 감소할 경우 이 힌트가 성능에 주는 영향을 알고 있어야 합니다.

최적기 힌트 예제

- 인덱스를 사용하도록 권장하는 힌트가 포함된 예제 입니다.

```
update --+ INDEX(p PROD_CATAGORY_IDX) products p
set   p.prod_min_price =
          (select
           (pr.prod_list_price*.95)
           from products pr
           where p.prod_id = pr.prod_id)
where p.prod_category = 'Men'
and   p.prod_status = 'available, on stock';
```

```
Execution Plan
----------------------------------------------------------
   0      UPDATE STATEMENT Optimizer=CHOOSE
          (Cost=2 Card=3 Bytes=195)
   1    0   UPDATE OF 'PRODUCTS'
   2    1     TABLE ACCESS (BY INDEX ROWID) OF 'PRODUCTS'
          (Cost=2 Card =3 Bytes=195)
   3    2       INDEX (RANGE SCAN) OF 'PROD_CATEGORY_IDX'
          (NON-UNIQUE) Cost=1 Card=3)
   4    1     TABLE ACCESS (BY INDEX ROWID) OF 'PRODUCTS'
          (Cost=2 Card=1 Bytes=26)
   5    4       INDEX (UNIQUE SCAN) OF 'PRODUCTS_PK' (UNIQUE)
          (Cost=1 Card=291)
```

힌트 범주

- 다음을 명령문 레벨에서 힌트로 지정할 수 있습니다.
  - 최적화 모드에 대한 힌트
    - ALL_ROWS, FIRST_ROWS_n
  - 액세스 경로 방식에 대한 힌트
  - 병렬 실행에 대한 힌트
  - 조인 순서 및 작업에 대한 힌트

자주 사용하는 힌트 목록

분류	힌트	설명
최적화 목표	ALL_ROWS	전체 처리속도 최적화
	FIRST_ROWS_n	최초 n건 응답속도 최적화
액세스 방식	FULL	Table Full Scan 유도
	INDEX	Index Scan 유도
	INDEX_DESC	Index 역순으로 스캔 유도
	INDEX_FFS	Index Fast Full Scan 유도
	INDEX_SS	Index Skip Scan 유도
조인 순서	ORDERED	FROM 절에 나열된 순서대로 조인
	LEADING	LEADING 힌트 괄호에 기술한 순서대로 조인 ex) LEADING(T1 T2) - T1, T2 순서대로 조인
	SWAP_JOIN_INPUTS	해지 조인시, BUILD INPUT을 명시적으로 선택 ex) SWAP_JOIN_INPUTS(T1)

자주 사용하는 힌트 목록

분류	힌트	설명
조인방식	USE_NL	NL 조인 유도
	USE_MERGE	Sort Merge 조인 유도
	USE_HASH	Hash 조인 유도
	NL_SJ	NL 세미조인 유도
	MERGE_SJ	Sort Merge 세미조인 유도
	HASH_SJ	Hash 세미조인 유도
서브쿼리 팩토링	MATERIALIZE	WITH 문으로 정의한 집합을 물리적으로 생성하도록 유도 ex) WITH /*+ MATERIALIZE */ T AS (SELECT ---)
	INLINE	WITH 문으로 정의한 집합을 물리적으로 생성하지 않고 INLINE 처리하도록 유도 ex) WITH /*+ INLINE */ T AS (SELECT ---)

자주 사용하는 힌트 목록

분류	힌트	설명
쿼리 변환	MERGE	VIEW 머징 유도
	NO_MERGE	VIEW 머징 방지
	UNNEST	서브쿼리 Unnesting 유도
	NO_UNNEST	서브쿼리 Unnesting 방지
	PUSH_PRED	조인조건 Pushdown 유도
	NO_PUSH_PRED	조인조건 Pushdown 방지
	USE_CONCAT	OR 또는 IN-List 조건을 OR-Expansion 으로 유도
	NO_EXPAND	OR 또는 IN-List 조건에 대한 OR-Expansion 방지

자주 사용하는 힌트 목록

분류	힌트	설명
병렬 처리	PARALLEL	테이블 스캔 또는 DML 을 병렬방식으로 처리하도록 유도 ex) PARLLEL(T1 2) PARALLEL(T2 2)
	PARALLEL_INDEX	인덱스 스캔을 병렬방식으로 처리하도록 유도
	PQ_DISTRIBUTE	병렬 수행 시 데이터 분배 방식 경정 ex) PQ_DISTRIBUTE(T1 HASH HASH)
기타	APPEND	Direct-Path Insert 유도
	DRIVING_SITE	DB Link Remote 쿼리에 대한 최적화 및 실행 주체 지정 (Local 또는 Remote)
	PUSH_SUBQ	서브쿼리를 가급적 빨리 필터링하도록 유도
	NO_PUSH_SUBQ	서브쿼리를 가급적 늦게 필터링하도록 유도

기본 엑세스 경로 힌트 예제

- employees 테이블에 대해 인덱스를 사용하지 않고 전체 테이블을 스캔하도록 명시합니다.
  - 실행 계획에서 TABLE ACCESS FULL 로 표시됩니다.

```
SELECT /*+ FULL(emp) */ employee_id, last_name
FROM employees emp
WHERE department_id = 50;
```

- 다음은 특정 인덱스를 사용하도록 강제하는 힌트 입니다.
  - 실행 계획에서 INDEX RANGE SCAN 혹은 INDEX UNIQUE SCAN 으로 나타납니다.

```
SELECT /*+ INDEX(emp emp_deptid_ix) */ employee_id, last_name
FROM employees emp
WHERE department_id = 50;
```


Thank you