실습 6. 인덱스(index) / 조인(join) / 실행계획(explain plan) / 힌트(hint)

6-1. 'Lein Bale' 사원의 사번(emp_no), 생년월일(birth_date), 성별(gender), 고용일(hire_date), 소속부서명(dept_name), 직급(title), 급여(salary) 조회

6-1-① 무엇이 잘못되었을까?

```
select *
from employees
where first_name||' '||last_name = 'Lein Bale' ;
```

6-1-② 인덱스 생성(last_name)

```
create index employees_idx1 on employees(last_name) ;
```

6-1-③ ①번 SQL 튜닝 (실행계획과 수행시간 비교)

```
select *
from employees
where first_name = 'Lein'
  and last_name = 'Bale' ;
```


실습 6. 인덱스(index) / 조인(join) / 실행계획(explain plan) / 힌트(hint)

6-2. 1999년도에 입사한 사원의 사번(emp_no), 생년월일(birth_date), 성별(gender), 고용일(hire_date), 소속부서명(dept_name), 직급(title), 급여(salary) 조회

6-2-① 답은 맞는거 같은데... 튜닝포인트는 없을까?

```
select a.emp_no
      , a.birth_date, a.gender, a.hire_date
      , c.dept_name, d.title, e.salary
from employees a, dept_emp b, departments c, titles d, salaries e
where to_char(a.hire_date,'yyyymmdd') between '19990101' and '19991231'
  and a.emp_no = b.emp_no and b.to_date = '99991231'
  and b.dept_no = c.dept_no
  and a.emp_no = d.emp_no and d.to_date = '99991231'
  and a.emp_no = e.emp_no and e.to_date = '99991231' ;
```

6-2-② 인덱스 생성(hire_date)

```
create index employees_idx2 on employees(hire_date) ;
```

실습 6. 인덱스(index) / 조인(join) / 실행계획(explain plan) / 힌트(hint)

6-2. 1999년도에 입사한 사원의 사번(emp_no), 생년월일(birth_date), 성별(gender), 고용일(hire_date), 소속부서명(dept_name), 직급(title), 급여(salary) 조회

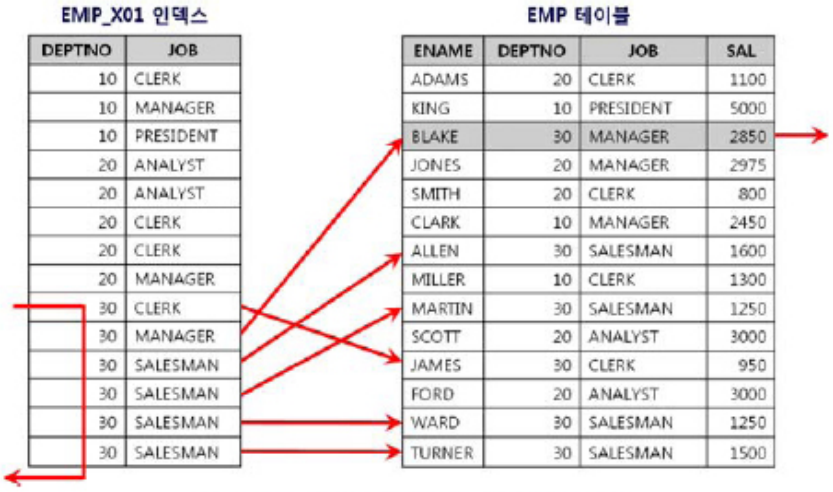
6-2-③ ①번 SQL 튜닝

```
select /*+ ordered use_nl(a b c d e) index(a employees_idx2) */ a.emp_no
     , a.birth_date, a.gender, a.hire_date
     , c.dept_name, d.title, e.salary
from employees a, dept_emp b, departments c, titles d, salaries e
where a.hire_date between to_date('19990101', 'yyyymmdd') and to_date('19991231', 'yyyymmdd')
  and a.emp_no = b.emp_no and b.to_date = '99991231'
  and b.dept_no = c.dept_no
  and a.emp_no = d.emp_no and d.to_date = '99991231'
  and a.emp_no = e.emp_no and e.to_date = '99991231' ;
```

인덱스 컬럼을 가공하지 말 것

개념 이해 5. 인덱스(Index)

- 목적 : 검색 속도 향상
- 구성 : 인덱스 컬럼(1개 이상) + ROWID (row의 위치정보)
- 특징 : 인덱스 컬럼 기준으로 정렬되어 있고, 물리적인 데이터를 가지는 데이터베이스 객체
- 종류 : B-tree 인덱스, Bitmap 인덱스, FBI(Function Based Index), Clustered Index 등
- 단점 : Insert, Update, Delete 시에 Index도 갱신해야 하므로 부하 가중

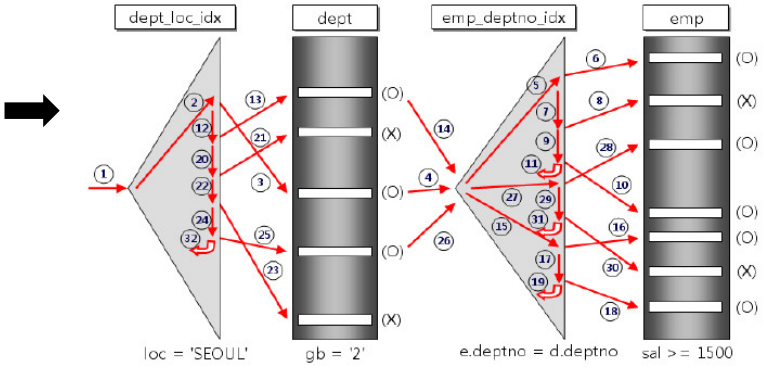


개념 이해 6. 조인(Join) : How-to-do 관점

Nested loop join

- 2개 이상의 테이블에서 하나의 집합을 기준으로 순차적으로 상대방 Row를 결합하여 원하는 결과를 조합하는 방식
- 먼저 선행 테이블의 처리 범위를 하나씩 액세스하면서 추출된 값으로 연결할 테이블을 조인한다
- 특징
  - ✓ 좁은 범위에 유리한 성능을 보여줌
  - ✓ 순차적으로 처리하며, Random Access 위주
  - ✓ 후행 테이블(Driven)에는 조인을 위한 인덱스 생성 필요

```
select *
from dept a, emp b
where a.loc = 'Seoul'
  and a.gb = '2'
  and a.dept_no = b.dept_no
  and b.sal >= 1500 ;
```

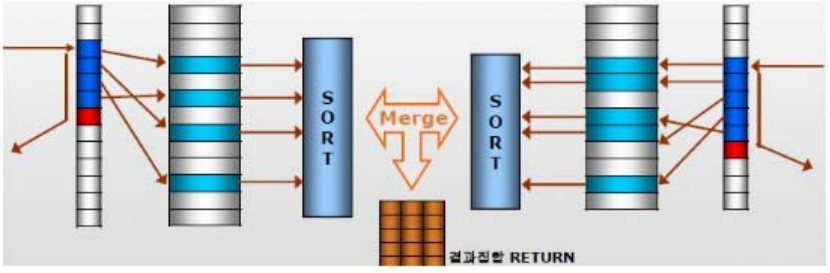


* 출처: https://needjarvis.tistory.com/162 [자비스가 필요해]

개념 이해 6. 조인(Join) : How-to-do 관점

Sort-Merge join

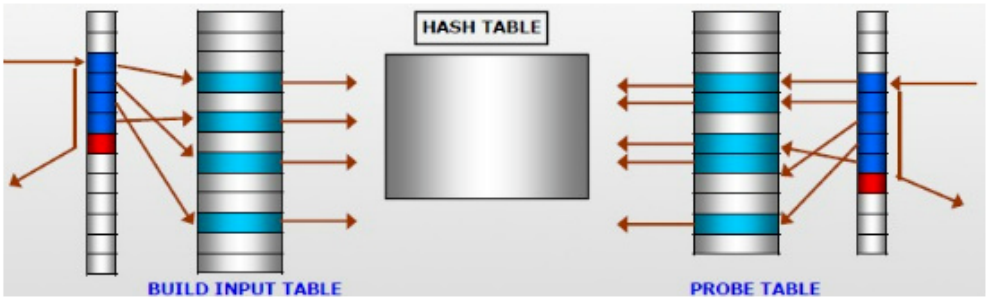
- 조인의 대상범위가 넓을 경우 발생하는 Random Access를 줄이기 위한 경우나 연결고리에 마땅한 인덱스가 존재하지 않을 경우 해결하기 위한 조인 방안
- 양쪽 테이블의 처리범위를 각자 Access하여 정렬한 결과를 차례로 Scan하면서 연결고리의 조건으로 Merge하는 방식
- 특징
  - ✓ 연결을 위해 랜덤 액세스를 하지 않고 스캔을 하면서 수행
  - ✓ Nested Loop Join처럼 선행집합 개념이 없음
  - ✓ 정렬을 위한 영역(Sort Area Size)에 따라 효율에 큰 차이 발생
  - ✓ 조인 연산자가 '='이 아닌 경우 nested loop 조인보다 유리한 경우가 많음



* 출처: https://needjarvis.tistory.com/162 [자비스가 필요해]

개념 이해 6. 조인(Join) : How-to-do 관점

- **Hash join**
  - 해싱 함수(Hashing Function) 기법을 활용하여 조인을 수행하는 방식(해싱 함수는 직접적인 연결을 담당하는 것이 아니라 연결될 대상을 특정 지역(partition)에 모아두는 역할만을 담당
  - 해시값을 이용하여 테이블을 조인하는 방식
  - Sort-Merge 조인은 소트의 부하가 많이 발생하여, Sort 대신 해쉬값을 이용하는 조인
  - 특징
    - ✓ 대용량 처리의 선결조건인 랜덤 액세스와 정렬에 대한 부담을 해결할 수 있는 대안
    - ✓ parallel processing을 이용한 hash 조인은 대용량 데이터를 처리하기 위한 최적의 솔루션
    - ✓ 2개의 조인 테이블 중 small rowset을 가지고 hash table 생성



* 출처: https://needjarvis.tistory.com/162 [자비스가 필요해]

개념 이해 7. 실행계획(Explain plan)

- **Explain plan (또는 Execution plan)**
  - 옵티마이져가 SQL문을 실행하기 위해 수행한 일련의 동작(OPERATIONS)을 트리형식으로 표현한 것
  - 사용자로 하여금 어떻게 SQL을 수행할 것인지를 보여주고 SQL 성능 판단을 위한 정보 제공
- **실행계획이 포함하는 정보**
  - 쿼리문이 참조한 테이블들의 순서
  - 쿼리문이 참조한 테이블들의 접근방법(ACESS PATH)
  - 조인에 의해 영향받는 테이블들의 조인방법(JOIN MEHTOD)
  - 데이터 조작방법(filter, sort, or aggregation, etc)
- **실행계획 상에 나타나는 성능에 악영향을 주는 요소**
  - Full scans : 의도하지 않은 Full scans
  - Unselective range scans : 100건을 조회하기 위해 백만건을 스캔하는 경우
  - Late predicate filters : 미리 처리범위를 좁히지 못하는 경우
  - Wrong join order : 잘못된 조인순서는 처리범위를 증가시킨다.
  - Late filter operations : 필터로 버릴 것이 있다면 조인 전에 필터하는 것이 좋다.

개념 이해 7. 실행계획(Explain plan)

실행계획 예

```
select *
from employees
where first_name = 'Lein'
  and last_name = 'Bale' ;
```

➡

인덱스 EMPLOYEES_IDX1을 통하여 LAST_NAME='Bale'조건으로 필터링된 row 각각에 대하여 EMPLOYEES 테이블에서 FIRST_NAME='Lein'조건을 체크하여 만족하는 row 출력

OPERATION	OBJECT_NAME	OPTIONS	CARDINALITY	COST
SELECT STATEMENT			1	178
TABLE ACCESS	EMPLOYEES	BY INDEX ROWID	1	178
Filter Predicates				
FIRST_NAME='Lein'				
INDEX	EMPLOYEES_IDX1	RANGE SCAN	183	3
Access Predicates				
LAST_NAME='Bale'				

```
Other XML
  {info}
    info type="db_version"
      11.2.0.2
    info type="parse_schema"
    "DA"
    info type="plan_hash"
      1147874131
    info type="plan_hash_2"
      3431891562
    {hint}
      INDEX_RS_ASC(@"SEL$1" "A"@"SEL$1" ("EMPLOYEES","LAST_NAME"))
      OUTLINE_LEAF(@"SEL$1")
      ALL_ROWS
      DB_VERSION('11.2.0.2')
      OPTIMIZER_FEATURES_ENABLE('11.2.0.2')
      IGNORE_OPTIM_EMBEDDED_HINTS
```


개념 이해 8. 힌트(Hint)

Oracle hint

- SQL에 포함되어 쓰여져 Optimizer의 실행 계획을 원하는 대로 유도하는 일종의 지시문
- 오라클 Optiomizer라고 해서 항상 최선의 실행 계획을 수립할 수는 없으므로 테이블이나 인덱스의 잘못된 실행 계획을 개발자가 직접 바꿀 수 있도록 도와주는 줌
- 힌트를 사용하게 되면 액세스 경로, 조인의 순서, Optiomizer의 목표(Goal)를 변경 가능하며 SQL 문장 내에 "/*+ 힌트내용 */" 형태로 사용되어짐 (*와 +는 띄어쓰기 없이 표현되어야 함에 주의)

Hint의 종류

Optimization Goals and Approaches	Acess Method Hints	Join Order Hints	Join Operation Hints	Parallel Execution Hints	Query Transformation Hints	Other Hints
• ALL_ROWS 혹은 FIRST_ROWS • CHOOSE • RULE	• AND_EQUAL • CLUSTER • FULL • HASH • INDEX 혹은 NO_INDEX • NDEX_ASC 혹은 INDEX_DESC • INDEX_COMBINE • INDEX_FFS • ROWID	• ORDERED • STAR	• DRIVING_SITE • HASH_AJ, MERGE_AJ 혹은 NL_AJ • LEADING • USE_HASH 혹은 USE_MERGE • USE_NL	• PARALLEL 혹은 NOPARALLEL • PARALLEL_INDEX • PQ_DISTRIBUTE • NOPARALLEL_INDEX	• EXPAND_GSET_TO_UNION • FACT 혹은 NOFACT • MERGE • NO_EXPAND • NO_MERGE • REWIRTE 혹은 NOREWRITE • STAR_TRANSFORMATION • USE_CONCAT	• APPEND 혹은 NOAPPEND • CACHE 혹은 NOCACHE • CURSOR_SHARED_EXACT • DYNAMIC_SAMPLING • NESTED_TABLE_GET_REFS • UNNEST 혹은 NO_UNNEST • ORDERED_PREDICATES

* 출처: https://luckys.tistory.com/342 [Lucky's...]

실습 7. 재귀적 관계(Recursive Relationship)

7-1. sample table 생성

```
CREATE TABLE EMP
       (EMPNO NUMBER(4) CONSTRAINT EMP_PK PRIMARY KEY,
        ENAME VARCHAR2(10),
       JOB VARCHAR2(9),
       MGR NUMBER(4),
       HIREDATE DATE,
       SAL NUMBER(7,2),
       COMM NUMBER(7,2),
       DEPTNO NUMBER(2));
```

7-2. sample data 생성

```
INSERT INTO EMP VALUES
(7369,'SMITH','CLERK',7902,to_date('17-12-1980','dd-mm-yyyy'),800,NULL,20);
INSERT INTO EMP VALUES
(7499,'ALLEN','SALESMAN',7698,to_date('20-2-1981','dd-mm-yyyy'),1600,300,30);
…
```


실습 7. 재귀적 관계(Recursive Relationship)

7-3. Emp 테이블에서 사원 간 상하관계 출력

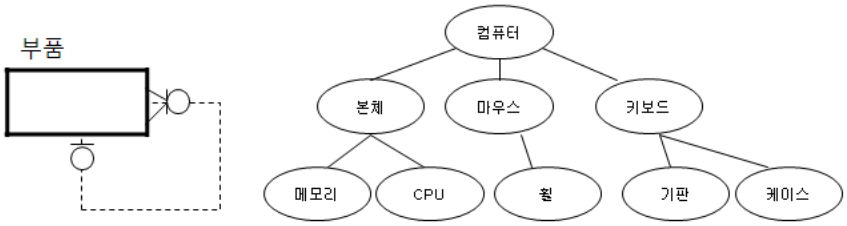
```
select lpad(' ', 2*level)||ename||'('||job||')' as "조직도"
from emp
start with mgr is null
connect by prior empno = mgr ;
```

출력 결과

```
조직도
KING(PRESIDENT)
  JONES(MANAGER)
    FORD(ANALYST)
      SMITH(CLERK)
  BLAKE(MANAGER)
    ALLEN(SALESMAN)
    WARD(SALESMAN)
    MARTIN(SALESMAN)
    TURNER(SALESMAN)
    JAMES(CLERK)
  CLARK(MANAGER)
    MILLER(CLERK)
```

개념 이해 9. 재귀적 관계(Recursive relationship)

- **재귀적 관계 (또는 순환관계)**
  - 하나의 엔티티 내에서 엔티티와 엔티티가 관계를 맺고 있는 형태
  - 부서, 부품, 메뉴 등과 같이 계층 구조 형태를 표현할 때 유용



부품은 다른 부품으로 조립된다.

- **START WITH ... CONNECT BY PRIOR ...**
  - START WITH 절 : 계층 구조의 데이터를 읽어나가는데 있어 시작점을 지정
  - CONNECT BY 절은 다음에 읽을 자식 데이터를 지정
    - ✓ 'PRIOR 자식컬럼 = 부모컬럼' 형태 : 부모 -> 자식 방향으로 내려가는 순방향 조회 시 사용
    - ✓ 'PRIOR 부모컬럼 = 자식컬럼' 형태 : 자식 -> 부모 방향으로 올라가는 역방향 조회 시 사용
  - NOCYCLE : 사이클이 발생한 이후의 데이터는 읽지 않습니다.
  - ORDER SIBLINGS BY 절 : 동일한 LEVEL의 데이터 사이에서 정렬을 합니다.