Sql 튜닝

성능저하

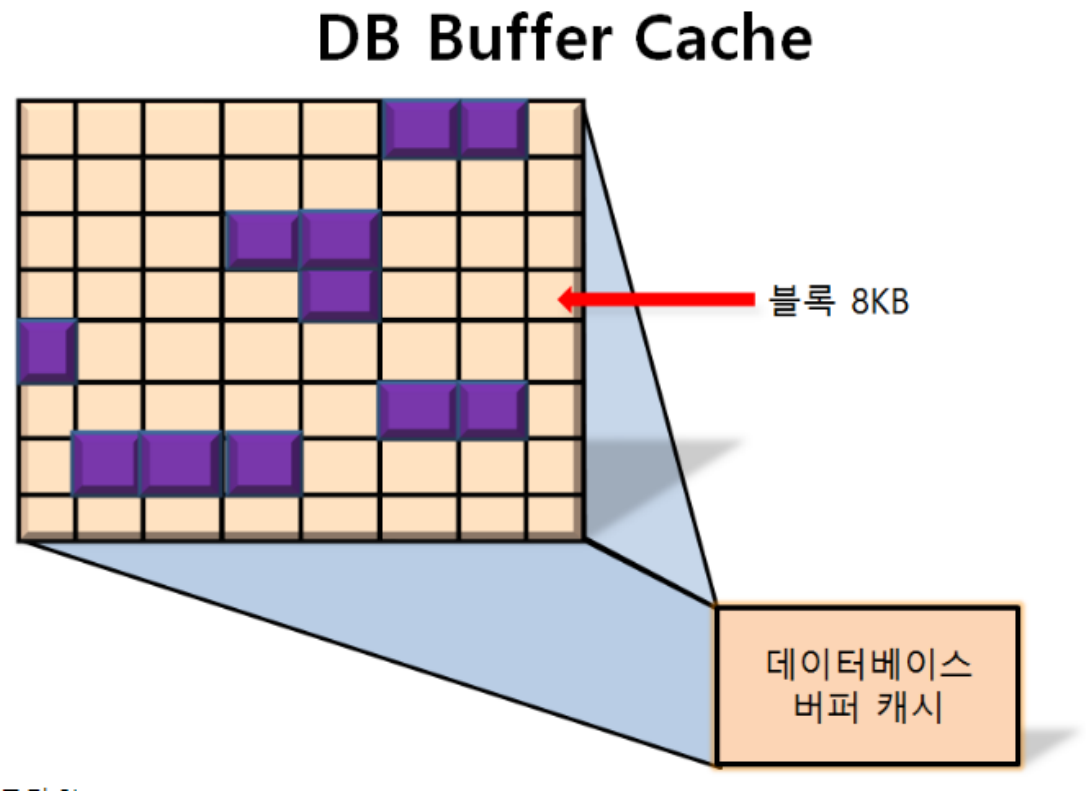
- **잘못된 옵티마이저 통계정보**
- **누락된 엑세스 구조**
- **최적 상태가 아닌 실행계획 상태**
- **잘못된 sql (데이터 타입 불일치)**
- **잘못 설계된 인덱스**

옵티마이저

- 비유:

```
올바른  길을  안내하는  네비게이션
```

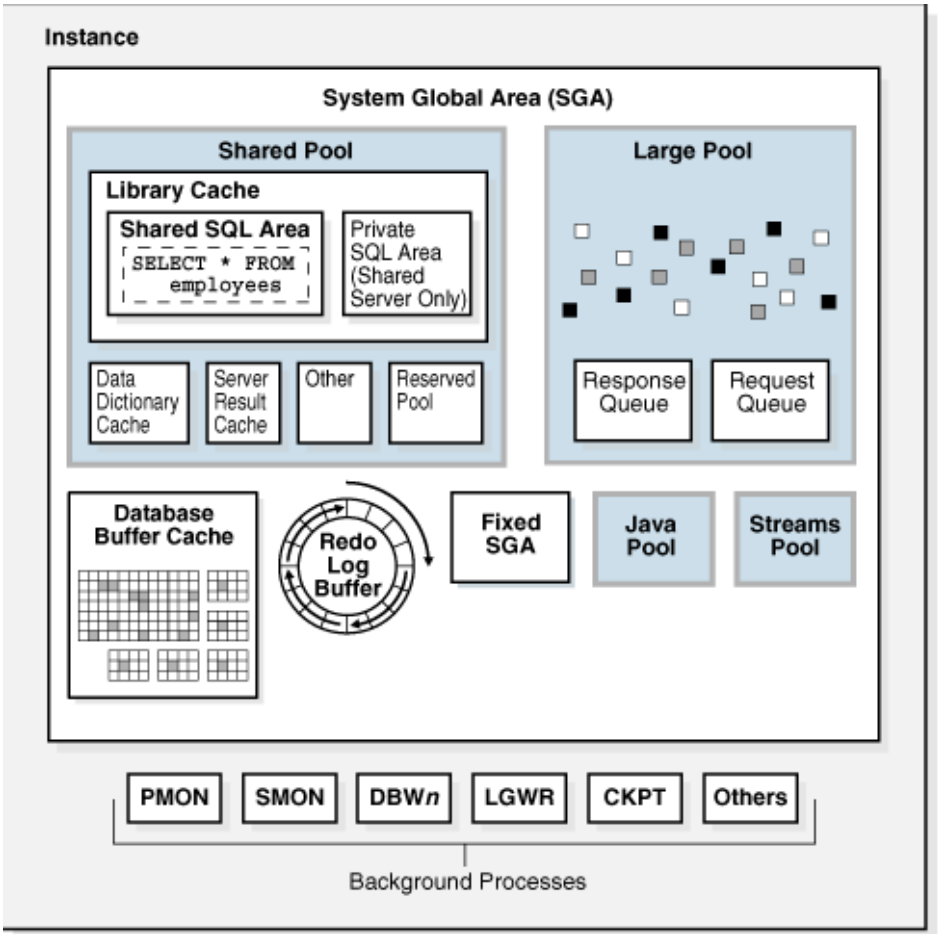
- 데이터 딕셔너리를 읽은후 → 실행계획 플랜 생성 → 실행
- 데이터파일을 읽어 전송한다.
- Parse - Excute - Peach



sql 이 실행되면 버퍼캐쉬에 한 번 캐쉬하고 재활용되게 한다.

캐쉬에서 못찾을 경우 디스크에서 가지고 와야 한다.

데이터베이스의 엔진은 최대한 디스크에 방문하지 않도록 설계한다.



- 자주 사용하는 sql은 라이브러리 캐쉬에 저장되게 한다.
- 디스크에 가게되면 시스템 사용량을 많이 차치하기 때문에
- 버퍼 캐쉬의 경우 데이터블록을 메모리에서 저장하는 반면
- 라이브러리 캐쉬의 경우 sql의 결과를 저장하게 됨.
- 공백 띄어쓰기가 조금만달라도 다른sql로 바라보기에 관례를 만들고 지켜야 한다.

LRU

시간	1	2	3	4	5	6	7	8	9	10	11	12
참조값	0	1	2	3	4	1	3	2	3	4	0	1
페이지	0	0	0	3	3	3	3	3	3	3	3	1
크기 : 3		1	1	1	4	4	4	2	2	2	0	0
			2	2	2	1	1	1	1	4	4	4
페이지 부재	0	0	0	0	0	0		0		0	0	0

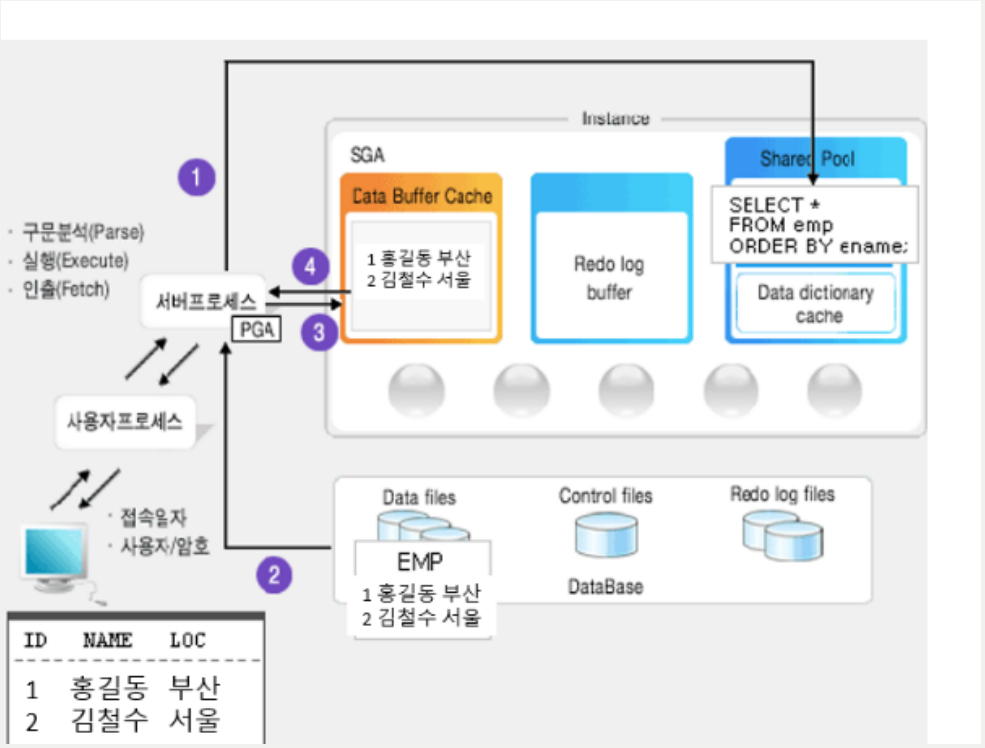
- 자주 사용되지 않는 데이터를 삭제하고
- 자주 사용하는것을 우선순위로 둔다.

처리구조

구분	설명
파싱 트리 생성	SQL 문을 이루는 개별 구성요소를 분석해서 파싱 트리 생성
Syntax 체크	문법적 오류가 없는지 확인 (ex. 사용할 수 없는 키워드를 사용했거나 순서가 바르지 않거나 누락된 키워드가 있는지 확인)
Semantic 체크	의미상 오류가 없는지 확인 (ex. 존재하지 않는 테이블 또는 컬럼을 사용했는지, 사용한 오브젝트에 대한 권한이 있는지 확인)

Select문 처리과정

- 구문 분석: excution plan : 내부적으로 하루에 한 번 갱신되는 통계정보를 근거로 가장 cost가 늦은 루트로 계획을 실행한다. 결가를 처리후 shared pool 에 구문을 저장
- 실행: DISK/I/O를 최대한 줄이기 위해 sga내의 DB 버퍼 캐쉬에서 있는지를 확인하고 디스크로 가게된다.
- 인출: 사용자에게 결과를 보고한다.



- 찾으려는 데이터가 각각 다 다른 블럭에서 찾아야할경우
- 다량의 IO 가 발생하게 된다.

오라클 데이터 저장단위

- 데이터블록 - 익스텐드 -세그먼트 - 테이블스페이스 순이다.

- 데이터블록 크기 조회

```
select * from gv$parameter where name like 'block%';
```

실행계획 읽는 순서

각각의 스탭에서 그 간계에서 어떤 명령이 수행되었고, 총몇건의데이터가 처리되었는지
시간, 코스트를 계산

위에서 아래로 내려가며 제일 먼저 읽을 스탭을 찾는다
내려가는 과정에서 같은 들여쓰기가 존재하면 위→ 아래순으로 읽는다
읽고자하는 스탭보다 들여쓰기가 된 하위 스탭이 존재시 가장 안쪽으로
들여쓰기 된 스텝을 시작으로 한 단계식 상위 스탭으로 읽어 나옴.

SQL 플랜해석

full tablescan: 전체 데이터블록을 읽는 방식

uniaue scan : 하나의 인덱스만 읽음

index range scan : 인덱스의 범위를 읽음

nasted loops : 무한 반복문이라고 생각하면 될듯.

실행계획 읽기

- 테이블 액세스 프로레스와 그 테이블의 인덱스를 액세스 하는 프로세스는 하나의 딘
- 여러 문장중에서 들여쓰기가 많이 되어있는 문장이 먼저 실행
- 들여쓰기가 같은 동일 레벨이라면 위에 있는 문장이 먼저실행
- 하위 노드를 가진 노드의 경우 하위 노드가 먼저 실행

```
select /*+ gather_plan statistics */* from emp where deptn
select * from table( DBMS_XPLAN.DISPLAY_CURSOR(null,null,'
```

룰기반

- 통계정보를 사용하지 않는다.
- 절대적으로 index 정보를 활용한다.
- 사전에 정의된 규칙에 따라 실행한다.
- 정해진 식으로 수행된다고 보면된다.

비용기반 옵티마이저

- 가장 빠르게 실해오디는 방법의 cost(비용을) 계산하여 실행
- 처리방법,순서를 우선적으로 정한다.
- 비용 산출을 위해 통계정보 필수
- 통계정보를 토대로 실행되게 한다.

실습

```
select  /*+ gather_plan_statistics */ *
    from emps e ,depts d
    where e.dept_no = d.dept_no
        and e.dept_no = 30;
        ----------------------------
select * from table( DBMS_XPLAN.DISPLAY_CURSOR(null,null,'allsta

---------------------------------------------------------------
Plan hash value: 4284654339

-------------------------------------------------------------------------
| Id  | Operation                     | Name     | Starts | E-Row
-------------------------------------------------------------------------
|   0 | SELECT STATEMENT              |          |      1 |
|   1 |  NESTED LOOPS                 |          |      1 |
|   2 |   TABLE ACCESS BY INDEX ROWID| DEPTS    |      1 |
|*  3 |    INDEX UNIQUE SCAN          | PK_DEPTS |      1 |
|*  4 |   TABLE ACCESS FULL           | EMPS     |      1 |
-------------------------------------------------------------------------
```

열 이름	설명	
Id	실행 계획에서 각 단계의 고유 식별자	
Operation	각 단계에서 수행되는 작업의 유형	
Name	작업이 수행되는 데이터베이스 객체의 이름	
Starts	해당 작업이 시작된 횟수	

열 이름	설명	
E-Rows	옵티마이저가 예측한 결과 집합의 행 수	
A-Rows	실제로 처리된 행 수	
A-Time	해당 단계가 실제로 소요된 시간 (밀리초 단위)	
Buffers	해당 단계에서 사용된 데이터 블록의 수	

클러스터링 팩터

- 데이터베이스에서 특정 클럼을 기준으로 같은 값을 갖는 데이터가 서로 모여있는 정도
- 좋다고 표현한다면 정렬 순서와 테이블 정렬 순서가 서로 비슷하다는 의미
- clustering_factor 수치가 테이블 블록이 가까울수록 데이터가 잘 정렬되 있음을 의미
- 레코드 개수(num_rows) 가까울 수록 흩어져 있음을 의미

힌트

sql 블록의 첫 키워드 바로 뒤에 입력

- 각 블록에는 하나의 hint 주석만 있어야 하나 하나의 hint 주석은 여려개의 힌트 포함가능
- hint는 해당 블록에만 적용
- 문장에 alias를사용하는 경우 힌트는 그 alias를 무조건 참조해야 함.

실습

- create 인덱스명 on table(칼럼)
- 인덱스의 역방향으로 수행

```
select  /*+INDEX_desc(e ix_emp_on) gather_plan_statistics*/ *
    from emp e
    where  e.deptno = 10;
select * from table( DBMS_XPLAN.DISPLAY_CURSOR(null,null,'allsta
```

- 정렬 시

```
Plan hash value: 1793559192

-------------------------------------------------------------------------
| Id  | Operation                                | Name      | Starts
-------------------------------------------------------------------------
|   0 | SELECT STATEMENT                         |           |      1
|   1 |  TABLE ACCESS BY INDEX ROWID BATCHED| EMP       |      1
|*  2 |   INDEX RANGE SCAN DESCENDING        | IX_EMP_ON |      1
-------------------------------------------------------------------------

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

- dept가 선두로 정렬되게 끔

```
-- 실행 계획을 출력
SELECT * FROM TABLE(DBMS_XPLAN.DISPLAY_CURSOR(NULL, NULL, 'ALLST
# orderd
select  /* orderd gather_plan_statistics*/ *
from emp e , dept d
where e.deptno = d.deptno
;

# reading
select  /* reading(b) gather_plan_statistics*/ *
from emp e , dept d
where e.deptno = d.deptno
;
-- 실행 계획을 출력
SELECT * FROM TABLE(DBMS_XPLAN.DISPLAY_CURSOR(NULL, NULL, 'ALLST
-- reading

-------------------------------------------------------------------------
| Id  | Operation          | Name | E-Rows |  OMem |  1Mem | Use
-------------------------------------------------------------------------
|   0 | SELECT STATEMENT   |      |        |       |       |
```

```
|*  1 |  HASH JOIN          |      |     14 |  1399K|  1399K| 107
|   2 |   TABLE ACCESS FULL| DEPT |      4 |       |       |
|   3 |   TABLE ACCESS FULL| EMP  |     14 |       |       |
---------------------------------------------------------------------


---------------------------------------------------------------------
| Id  | Operation           | Name | E-Rows |  OMem |  1Mem | Use
---------------------------------------------------------------------
|   0 | SELECT STATEMENT    |      |        |       |       |
|*  1 |  HASH JOIN          |      |     14 |  1399K|  1399K|  97
|   2 |   TABLE ACCESS FULL| DEPT |      4 |       |       |
|   3 |   TABLE ACCESS FULL| EMP  |     14 |       |       |
---------------------------------------------------------------------
```

CACHE 힌트

- table full scan 으로 읽혀진 테이블들을 buffer cache lru list 의 most recentily used 쪽에 둔다.
- 디폴트로 수행되는 방식이다.
- 쉽게 생각하면 LRU 메모리의 앞단에 둔다고 생각하면 된다.

```
select /* full(a) cache(a) */
empno,ename
from emp a;
```

Nocache

- TABLE full scan 으로 읽혀진 테이블 들을 buffer cache lru list의 least recently used 쪽에 둔다.
- 디폴트로 수행되는 방식이다.
- LRU 메모리의 뒷단에두어 빠른시간안에 사라진다.

```
select /* full(a) nocache(a) */
empno,ename

from emp a;
```