

Chapter 05. SQL 최적화 기본 원리

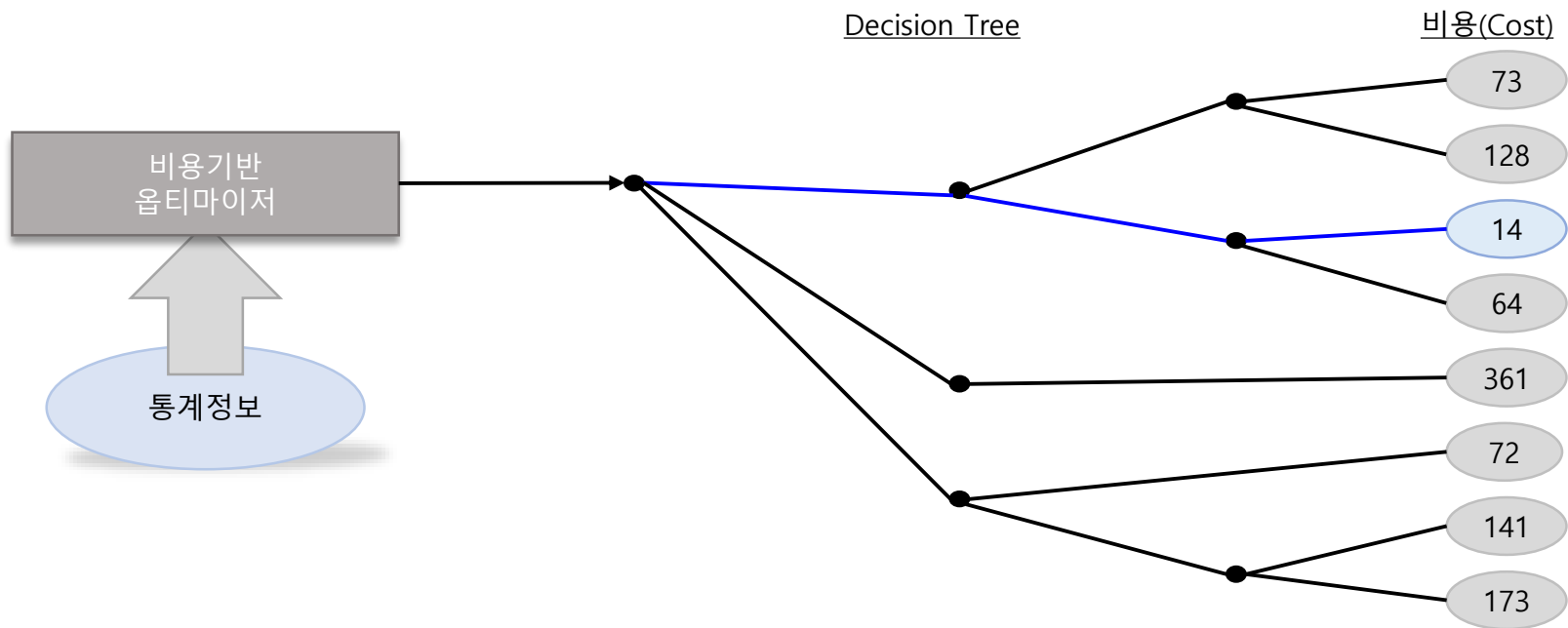
# 5-1. 옵티마이저와 실행계획

➤ 옵티마이저란

- ① 사용자가 질의한 SQL문에 대한 최적의 실행방법을 결정하는 역할을 수행한다. 이러한 최적의 실행방법을 실행 계획이라고 한다.
- ② 사용자의 요구사항을 만족하는 결과를 추출할 수 있는 다양한 실행 방법이 존재한다.
- ③ 다양한 실행 방법들을 도출하고 그중에서 최적의 실행방법을 결정하는 것이 옵티마이저의 역할이다.

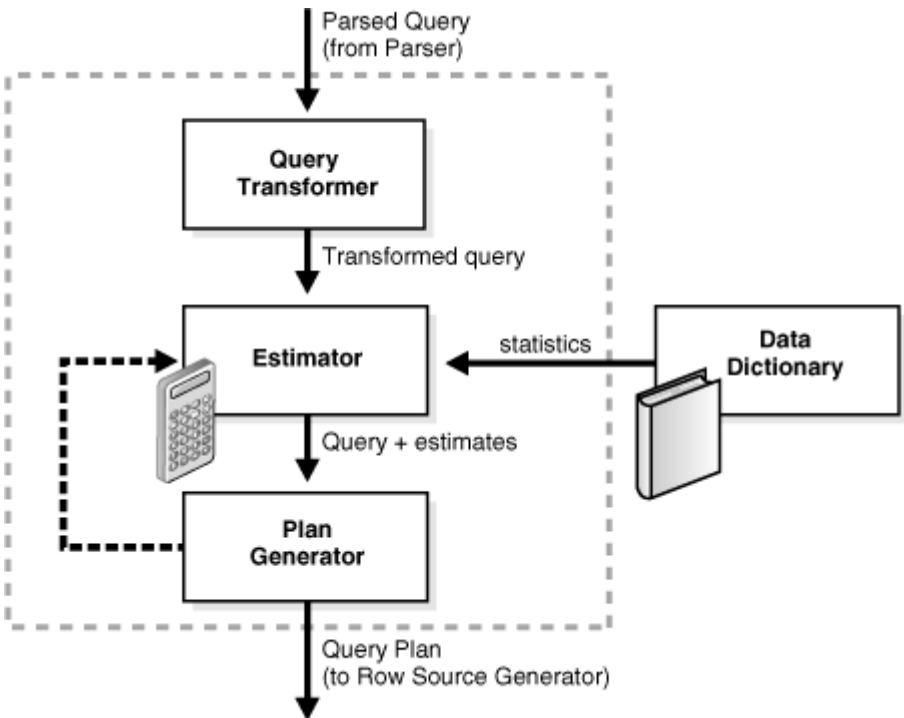
➤ 비용기반 옵티마이저

- ① SQL문을 처리하는데 비용이 가장 적게 드는 실행계획을 선택하는 방식이다. 비용이란 SQL문을 처리하는데 예상되는 시간 또는 자원을 의미한다.
- ② 테이블, 인덱스 등의 통계 정보와 시스템 통계정보를 이용하여 최적의 실행계획을 도출한다.
- ③ 인덱스를 사용하는 비용이 테이블 풀 스캔 비용보다 크다고 판단되면 테이블 풀 스캔을 유도 하게 된다.



➤ 옵티마이저의 구성 요소

구성 요소	설명
질의 변환기 (Query Transformer)	• 사용자가 작성한 SQL문을 처리하기에 보다 용이한 형태의 SQL문으로 <b>변환</b>
비용 예측기 (Estimator)	• 대안 계획 생성시에 의해서 생성된 대안 계획의 <b>비용을 예측</b> 하는 모듈 • 대안 계획의 정확한 비용을 측정하기 위해서 연산의 중간 집합의 크기 및 결과 집합의 크기, 분포도 등의 예측을 함, 보다 나은 예측을 위해서 <b>정확한 통계 정보가 필요</b> 함
대안계획생성기 (Plan Generator)	• 동일한 결과를 생성하는 <b>다양한 대안 계획을 생성</b> 하는 모듈 • 대안 계획은 연산의 적용 순서, 연산방법변경, 조인 순서 변경 등을 통해서 생성 • 동일한 결과를 생성하는 가능한 모든 대안 계획을 생성해야 보다 나은 최적화를 수행할 수 있음



SQL은 기본적으로 구조적이고 집합적이고 선언적인 질의 언어다.



결과 집합을 만드는 과정은 절차적이며, 각각의 프로시저를 만드는 역할을 옵티마이저가 담당한다.

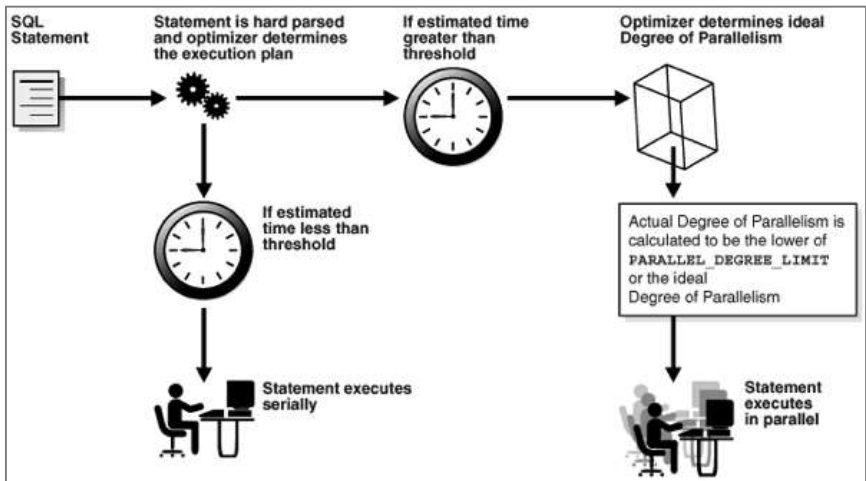
➤ 네비게이션과 옵티마이저

❖ 네비게이션



- ❖ 경로 탐색 과정에서 네비게이션이 착안하는 정보
  - GPS 위치정보, 지도, 주소 정보, 도로정보
  - 통행 요금, 구간별 평균/제한 속도, 실시간 교통 정보
  - 공사 구간이나 시위, 도로 행진, 기타 행사로 인한 임시 교통 통제 구간 정보
- ❖ 수많은 정보를 바탕으로 여러 개의 경로를 알아낸 후 최적의 경로로 안내해주는 역할을 함

❖ 옵티마이저



- ❖ SQL 실행 계획 수립 시 옵티마이저가 착안하는 정보
  - 테이블, 컬럼, 인덱스 구조에 관한 기본 정보
  - 오브젝트 통계(테이블 통계, 인덱스 통계, 히스토그램 통계)
  - 시스템 통계 : CPU속도, Single Block I/O 속도, Multi block I/O 속도
  - 옵티마이저 관련 파라미터
- ❖ 하나의 쿼리 수행 시 후보 군이 될만한 무수히 많은 실행계획들을 도출한 후 짧은 순간 각각의 효율성을 판단하여 최적의 실행계획을 수립하게 됨

➤ 실행계획 예시

- ❖ TB\_EMP 테이블의 인덱스 : PK\_TB\_EMP(EMP\_NO)
- ❖ TB\_DEPT 테이블의 인덱스 : PK\_TB\_DEPT(DEPT\_CD)
- ❖ TB\_EMP\_CERTI 테이블의 인덱스 : PK\_TB\_EMP\_CERTI(EMP\_NO,CERTI\_SN)

```
SELECT A.EMP_NO
      , A.EMP_NM
      , B.DEPT_CD
      , C.CERTI_CD
      , B.DEPT_NM
FROM TB_EMP A
     , TB_DEPT B
     , TB_EMP_CERTI C
WHERE B.DEPT_CD = '100004'
      AND A.DEPT_CD = B.DEPT_CD
      AND A.EMP_NO = C.EMP_NO
;
```

- ✓ 위에서 아래로, 바깥쪽에서 안쪽으로 읽는다.
- ✓ 실행계획에는 사용 객체, 조인 방법, 조인 순서, 액세스 스패턴등의 정보가 출력된다.

\*\*\*\*\*[Explain Plan Time: 2021/12/15 22:39:31]\*\*\*\*\*  
Execution Plan

```
0      SELECT STATEMENT Optimizer=ALL_ROWS (Cost=7 Card=12 Bytes=720)
1    0      HASH JOIN (Cost=7 Card=12 Bytes=720)
2      1      NESTED LOOPS (Cost=4 Card=4 Bytes=168)
3        2        TABLE ACCESS (BY INDEX ROWID) OF 'TB_DEPT' (TABLE) (Cost=1 Card=1 Bytes=17)
4          3          INDEX (UNIQUE SCAN) OF 'PK_TB_DEPT' (INDEX (UNIQUE)) (Cost=0 Card=1)
5            2            TABLE ACCESS (FULL) OF 'TB_EMP' (TABLE) (Cost=3 Card=4 Bytes=100)
6              1              TABLE ACCESS (FULL) OF 'TB_EMP_CERTI' (TABLE) (Cost=3 Card=123 Bytes=2K)
```

Predicate information (identified by operation id):

```
1 - access("A"."EMP_NO"="C"."EMP_NO")
4 - access("B"."DEPT_CD"='100004')
5 - filter("A"."DEPT_CD"='100004')
```

❖ Dbeaver에서  
[CTRL+SHIFT+E]를 누르면  
실행계획이 출력된다.

➤ 실행계획 예시 - 계속

DBeaver 21.3.0 - <orcl12cr2\_SQLD> Script-47

파일(F) 편집(E) 탐색(N) 검색(A) SQL 편집기 데이터베이스(D) 윈도우(W) 도움말(H)

SQL 커밋 롤백 None orcl12cr2\_SQLD SQL

<orcl12cr2\_SQLD> Script-48 \*<orcl12cr2\_SQLD> Script-47 X

1 SELECT A.EMP\_NO  
2 , A.EMP\_NM  
3 , B.DEPT\_CD  
4 , C.CERTI\_CD  
5 , B.DEPT\_NM  
6 FROM TB\_EMP A  
7 , TB\_DEPT B  
8 , TB\_EMP\_CERTI C  
9 WHERE B.DEPT\_CD = '100004'  
10 AND A.DEPT\_CD = B.DEPT\_CD  
11 AND A.EMP\_NO = C.EMP\_NO  
12 ;

❖ Dbeaver에서 [CTRL+SHIFT+E]  
를 누르면 실행계획이 출력된다.

Results 1 Execution plan - 1 X

Operation	Object	Optimizer	Cost	Cardinality	Bytes
SELECT STATEMENT		ALL_ROWS	7	12	720
HASH JOIN			7	12	720
NESTED LOOPS			4	4	168
TABLE ACCESS (BY INDEX ROWID)	TB_DEPT	ANALYZED	1	1	17
INDEX (UNIQUE SCAN)	PK_TB_DEPT	ANALYZED	0	1	0
TABLE ACCESS (FULL)	TB_EMP	ANALYZED	3	4	100
TABLE ACCESS (FULL)	TB_EMP_CERTI	ANALYZED	3	123	2,214

SELECT A.EMP\_NO  
A.EMP\_NM  
B.DEPT\_CD

KST ko 쓰기 가능 스마트 삽입 11 : 27 : 235 S...0

➤ 실행내역 확인

❖ 현재 접속한 세션의 통계 레벨을 ALL로 설정

```
ALTER SESSION SET STATISTICS_LEVEL = ALL;
```

❖ SQL문 실행

```
SELECT /* SELECT.TB_EMP.TB_DEPT.001 */
      A.EMP_NO
    , A.EMP_NM
    , B.DEPT_CD
    , B.DEPT_NM
  FROM TB_EMP A
    , TB_DEPT B
 WHERE B.DEPT_CD = '100004'
        AND B.DEPT_CD = A.DEPT_CD
;
```

❖ SQL문의 실행 정보 조회

```
SELECT
      SQL_ID
    , CHILD_NUMBER
    , SUBSTR(SQL_FULLTEXT, 1, 40)
  FROM V$SQL
 WHERE SQL_FULLTEXT LIKE '%SELECT.TB_EMP.TB_DEPT.001%'
        AND SQL_FULLTEXT NOT LIKE '%V$SQL%'
        AND SQL_FULLTEXT NOT LIKE 'EXPLAIN%'
;
```

SQL_ID	CHILD_NUMBER	SUBSTR(SQL_FULLTEXT,1,40)
gmt02v87zk47d	0	SELECT /* SELECT.TB_EMP.TB_DEPT.001 */

➤ 실행내역 확인

❖ 실행 내역 출력

```
SELECT * FROM TABLE(DBMS_XPLAN.DISPLAY_CURSOR('gmt02v87zk47d',0,'ALLSTATS LAST -ROWS'));
```

❖ 실행 내역 확인

|PLAN\_TABLE\_OUTPUT

|-----

|SQL\_ID gmt02v87zk47d, child number 1

|-----

|SELECT /\* SELECT.TB\_EMP.TB\_DEPT.001 \*/ A.EMP\_NO ,

|A.EMP\_NM , B.DEPT\_CD , B.DEPT\_NM FROM TB\_EMP A ,

|TB\_DEPT B WHERE B.DEPT\_CD = '100004' AND B.DEPT\_CD =

|A.DEPT\_CD

|

|Plan hash value: 1631502933

|

|-----

Id	Operation	Name	Starts	A-Rows	A-Time	Buffers
0	SELECT STATEMENT		1	4	00:00:00.01	9
1	NESTED LOOPS		1	4	00:00:00.01	9
2	TABLE ACCESS BY INDEX ROWID	TB_DEPT	1	1	00:00:00.01	2
* 3	INDEX UNIQUE SCAN	PK_TB_DEPT	1	1	00:00:00.01	1
* 4	TABLE ACCESS FULL	TB_EMP	1	4	00:00:00.01	7

|-----

|Predicate Information (identified by operation id):

|-----

| 3 - access("B"."DEPT\_CD"='100004')

| 4 - filter("A"."DEPT\_CD"='100004')

|

|Note

|-----

| - dynamic statistics used: dynamic sampling (level=2)



➤ 옵티마이저의 선택 - INDEX RANGE SCAN

❖ 인덱스 생성 후 SQL문 실행

```
CREATE INDEX IDX_TB_EMP_01 ON TB_EMP(DEPT_CD);
```

```
SELECT A.EMP_NO  
      , A.EMP_NM  
      , A.DEPT_CD  
FROM TB_EMP A  
WHERE A.DEPT_CD = '100004'  
;
```

\*\*\*\*\*[Explain Plan Time: 2021/12/15 19:47:28]\*\*\*\*\*  
Execution Plan

0	SELECT STATEMENT Optimizer=ALL_ROWS (Cost=2 Card=4 Bytes=388)
1	0 TABLE ACCESS (BY INDEX ROWID) OF 'TB_EMP' (TABLE) (Cost=2 Card=4 Bytes=388)
2	1 INDEX (RANGE SCAN) OF 'IDX_TB_EMP_01' (INDEX) (Cost=1 Card=4)

Predicate information (identified by operation id):

2	- access("A"."DEPT_CD"='100004')
---	----------------------------------



- ❖ 비용(Cost)은 쿼리를 수행하는 동안 발생 될 것으로 예상하는 I/O 횟수 또는 예상 소요시간을 표현한 값이다.
- ❖ 비용(Cost)은 언제나 지나 예상치일 뿐이다. 실행 경로를 선택하기 위해 옵티마이저가 통계 정보를 활용해서 계산해낸 값이다.

Chapter 05. SQL 최적화 기본 원리

# 5-2. 인덱스 기본

## ➤ 인덱스 란?

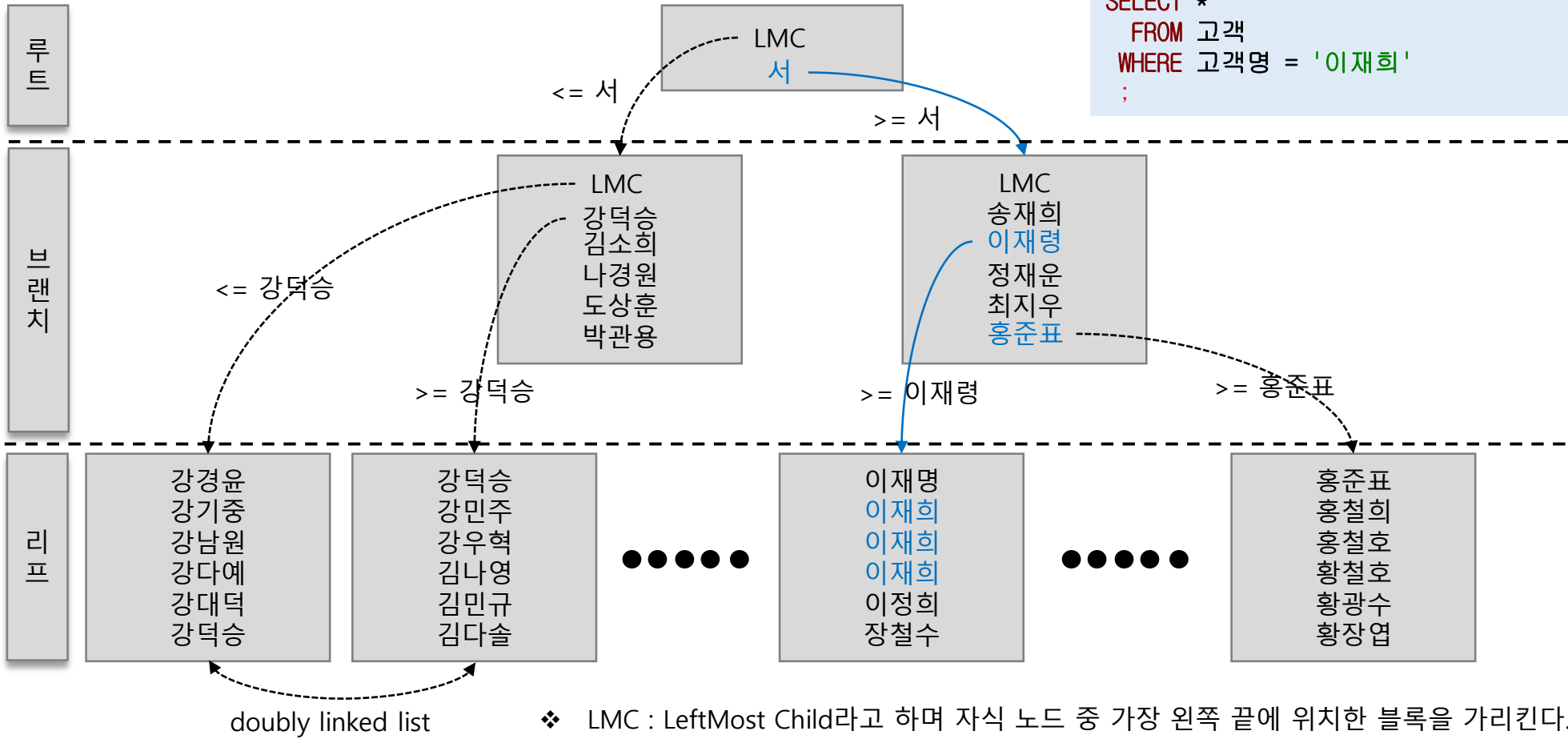
- ① 인덱스는 원하는 데이터를 쉽게 찾을 수 있도록 돕는 책의 찾아보기와 유사한 개념이다.
- ② 검색조건에 부합하는 데이터를 효과적으로(빠르게) 검색할 수 있도록 돕는다.
- ③ 한 테이블은 0개~N개의 인덱스를 가질 수 있다.
- ④ 한 테이블에 과도하게 많은 인덱스가 존재하면 INSERT, UPDATE, DELETE와 같은 DML작업 시 부하가 발생한다.

## ➤ B\*Tree 인덱스란?

- ① DBMS에서 널리 사용되는 가장 일반적인 인덱스이다.
- ② 루트 블록, 브랜치 블록, 리프 블록으로 구성된다.
- ③ 가장 상위에 존재하는 블록이 루트 블록이고 브랜치 블록은 분기를 목적으로 하는 블록이다.
- ④ 리프 블록은 트리의 가장 아래 단계에 존재하는 블록이다.
- ⑤ 리프 블록은 인덱스를 구성하는 칼럼의 데이터와 해당 데이터를 가지고 있는 행의 위치를 가리키는 레코드 식별자 인 ROWID로 구성되어 있다.

➤ B\*Tree 인덱스 구조

```
CREATE INDEX 고객_N1 ON 고객(고객명);  
  
SELECT *  
FROM 고객  
WHERE 고객명 = '이재희';  
;
```



➤ 인덱스 구조 상세

- ① 루트와 브랜치 블록에 있는 각 레코드는 하위 블록에 대한 주소 값을 갖는다. 키 값은 하위 블록에 저장된 키 값의 범위를 나타낸다.
- ② LMC가 가리키는 주소로 찾아간 블록에는 키 값을 가진 첫번째 레코드보다 작거나 같은 레코드가 저장돼 있다.
- ③ 리프 블록에 저장된 각 레코드는 키 값 순으로 정렬돼 있을 뿐만 아니라 테이블 레코드를 가리키는 주소값 즉 Rowid를 갖는다.
- ④ 인덱스 키 값이 같으면 Rowid순으로 정렬된다.
- ⑤ 인덱스를 스캔하는 이유는 검색조건을 만족하는 소량의 데이터를 빨리 찾고 거기서 Rowid를 얻기 위해서이다.

➤ ROWID의 구성

항목	구성
ROWID	데이터 블록 주소 + 로우 번호
데이터 블록 주소	데이터 파일 번호 + 블록 번호
블록 번호	데이터 파일 내에서 부여한 상대적 순번
로우번호	블록 내 순번

➤ 인덱스 스캔 효율화

❖시력이 1.0~1.5인 "이정민" 학생을 찾는 경우

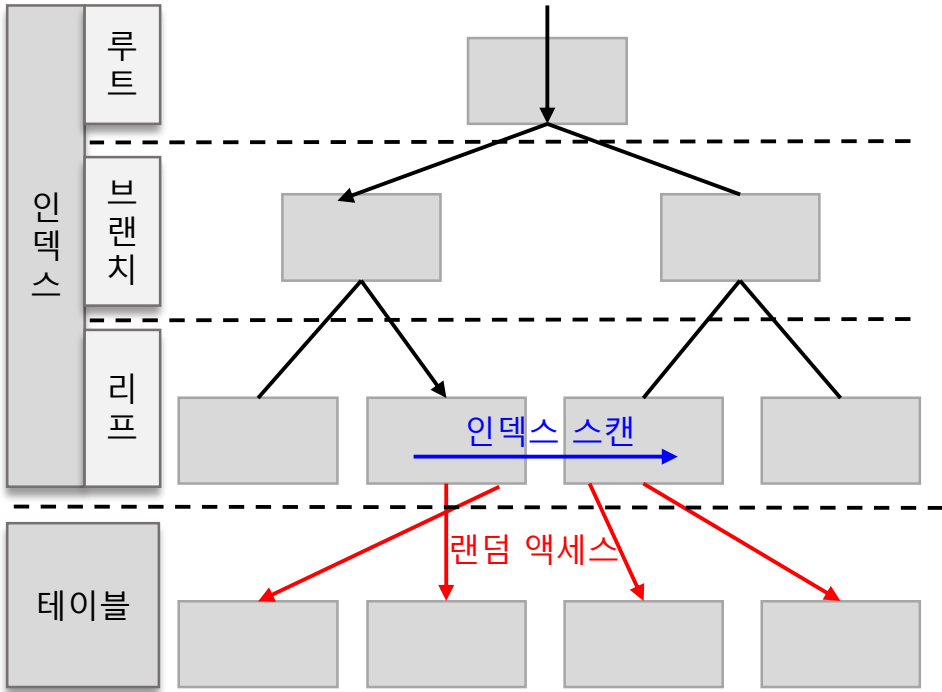
시력	이름	학년-반-번호
0.5	김지성	3학년 2반 13번
...	...	...
1.0	이정민	5학년 1반 16번
1.5	이경오	4학년 3반 37번
1.5	남경희	6학년 4반 19번
1.5	이정민	1학년 5반 15번
1.5	...	...
2.0	이정민	2학년 6반

✓ "시력+이름+학  
년-반-번호" 로  
이루어진 인덱스  
는 스캔 비효율  
이 존재함

✓ "이름+시력+학년-  
반-번호"로 이루어  
진 인덱스는 스캔  
효율이 좋음

이름	시력	학년-반-번호
이경오	1.5	4학년 3반 37번
김지성	0.5	3학년 2반 13번
...	...	...
남경희	1.5	6학년 4반 19번
...	...	...
이정민	1.0	5학년 1반 16번
이정민	1.5	1학년 5반 15번
이정민	2.0	2학년 6반 24번

➤ 랜덤 액세스 최소화



- ❖ 인덱스 스캔 후 추가 정보를 가져오기 위해 Table Random Access를 수행한다.
- ❖ 인덱스 스캔 비효율이 발생하면 덩달아 Table Random Access의 횟수도 많아지게 된다.
- ❖ 해당 작업은 DBMS 성능 부하의 주 요인이 되며 SQL튜닝은 곧 Random I/O와의 전쟁이라 할 수 있다.

➤ 인덱스를 탄다 VS 인덱스를 안탄다

where 생년월일 between '20070101' and '20070131'

리프

20060216	20060505	20060815	20061225	20070109	20070126	20070314	20070515	20070518	20070606

인덱스를 탄다!

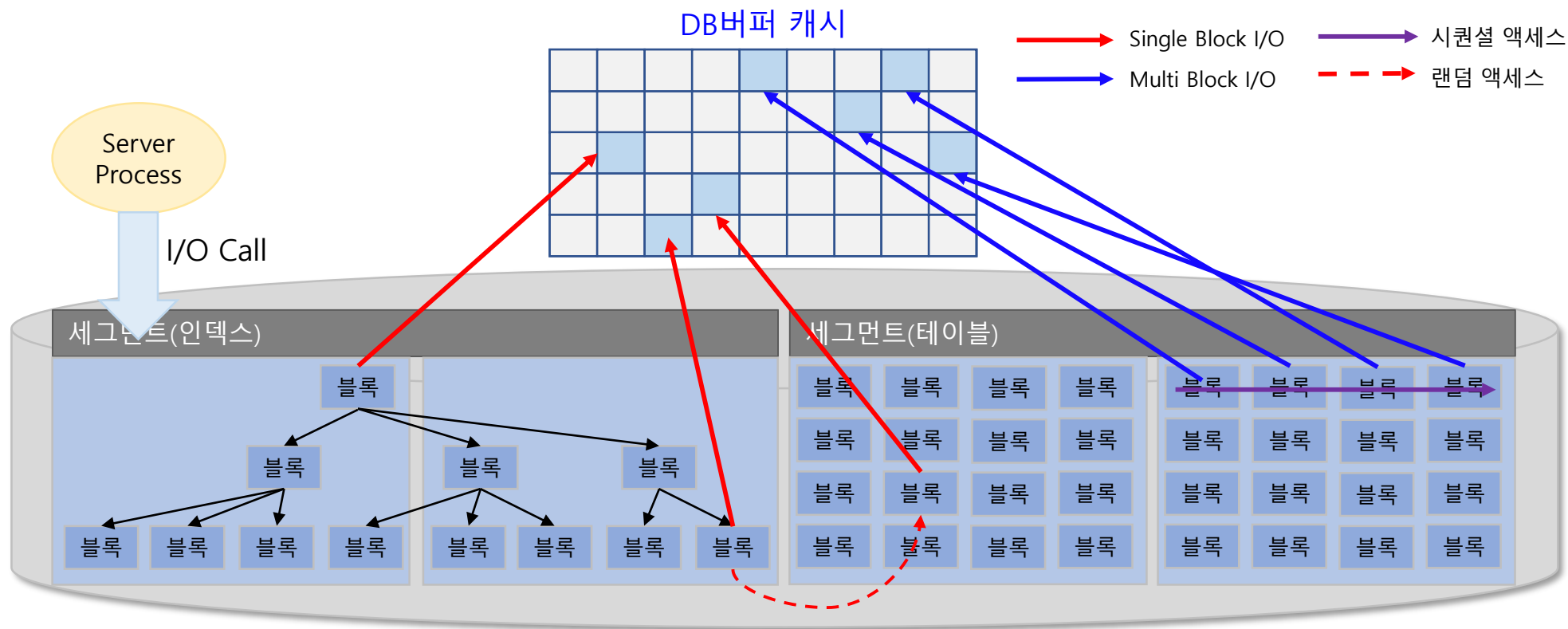
where substr(생년월일, 5, 2) = '05'

리프

20060216	20060505	20060815	20061225	20070109	20070126	20070314	20070515	20070518	20070606

인덱스를 제대로 안탄다!

➤ Single Block I/O vs Multi Block I/O



- ❖ Single Block I/O는 인덱스 루트 블록을 읽을 때, 인덱스 루트 블록에서 얻은 주소로 브랜치 블록을 읽을 때, 인덱스 브랜치 블록에서 읽은 주소로 리프 블록을 읽을 때, 인덱스 리프블록에서 읽은 주소로 테이블 블록을 읽을 때 발생
- ❖ Multi Block I/O는 캐시에서 찾지 못한 특정 블록을 읽으려고 I/O Call을 할 때 **디스크 상에 그 블록과 인접한 블록들을 한꺼번에 읽어 캐시에 미리 적재** 하는 것이다. (DB\_FILE\_MULTIBLOCK\_READ\_COUNT)



➤ 테이블 풀 스캔과 인덱스 스캔

스캔 유형	설명
테이블 풀 스캔	<ul style="list-style-type: none"><li>• 테이블에 존재하는 모든 데이터를 읽어가면서 조건에 맞으면 결과로 추출하고 조건에 맞지 않으면 버리는 방식</li><li>• HIGH WATER MARK는 테이블에 데이터가 쓰여졌던 블록 상의 최상위 위치로써 테이블 풀 스캔 시는 HWM까지의 블록에 있는 모든 데이터를 읽어야 하기 때문에 시간이 오래 걸릴 수 있다.</li><li>• 테이블 풀 스캔으로 읽은 블록은 재 사용성이 낮다고 보고 메모리 버퍼 캐시에서 금방 제거될 수 있도록 관리한다. (LRU 리스트에 끝으로 가서 금방 제거된다.)</li><li>• 옵티마이저가 테이블 풀 스캔을 선택하는 경우<ul style="list-style-type: none"><li>- SQL문에 조건이 존재하지 않는 경우</li><li>- SQL문의 조건을 기준으로 사용 가능한 인덱스가 없는 경우</li><li>- 옵티마이저의 판단으로 테이블 풀 스캔이 유리하다고 판단하는 경우</li><li>- 전체 테이블 스캔을 하도록 강제로 힌트를 지정한 경우</li></ul></li></ul>
인덱스 스캔	<ul style="list-style-type: none"><li>• 인덱스 스캔은 인덱스를 구성하는 칼럼의 값을 기반으로 데이터를 추출하는 액세스 기법</li><li>• 인덱스 리프 블록은 인덱스를 구성하는 칼럼과 ROWID로 구성</li><li>• 인덱스의 리프 블록을 읽으면 인덱스 구성 칼럼의 값과 ROWID를 알 수 있음</li><li>• 즉 인덱스를 읽어서 대상 ROWID를 찾으면 해당 ROWID로 다시 테이블을 찾아 가야함 (Table Random Access발생)</li><li>• 하지만 SQL문에서 필요로 하는 칼럼이 모두 인덱스 구성칼럼이라면 테이블을 찾아갈 필요 없음</li><li>• 일반적으로 인덱스 스캔을 통해 데이터를 추출하면 해당 결과는 인덱스의 칼럼의 순서로 정렬된 상태로 반환됨</li><li>• 인덱스 스캔으로 읽은 블록은 테이블 풀 스캔에 의해서 읽은 블록에 비해 버퍼 캐시에 더욱 더 오랫동안 남아 있다.</li></ul>

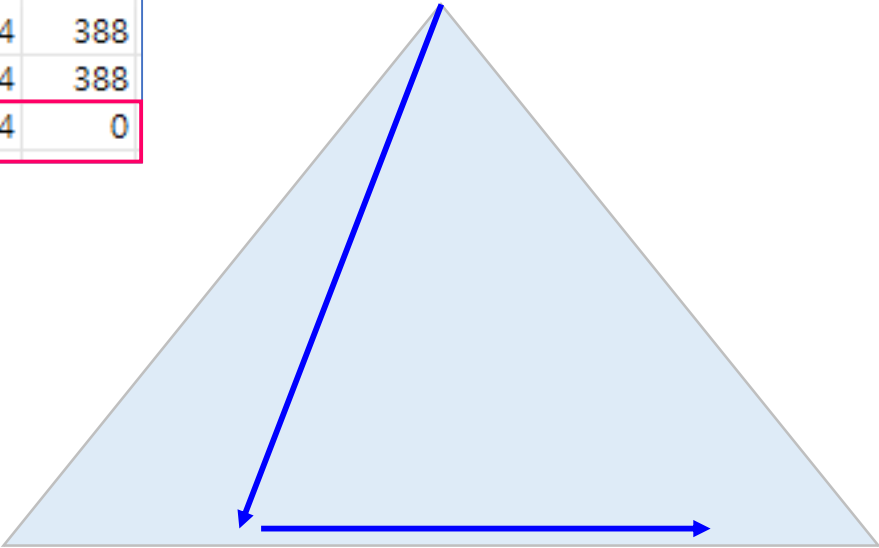
➤ 인덱스 범위 스캔

- ① 인덱스를 이용하여 **한건 이상의 데이터를 추출**하는 방식
- ② 인덱스 스캔으로 특정 범위를 스캔하면서 대상 레코드를 하나하나 리턴하는 방식임

```
CREATE INDEX IDX_TB_EMP_01 ON TB_EMP(DEPT_CD);
```

```
SELECT A.EMP_NO  
      , A.EMP_NM  
      , A.DEPT_CD  
FROM TB_EMP A  
WHERE A.DEPT_CD = '100004'  
;
```

Operation	Object	Optimizer	Cost	Cardinality	Bytes
▼ SELECT STATEMENT		ALL_ROWS	2	4	388
▼ TABLE ACCESS (BY INDEX ROWID)	TB_EMP		2	4	388
INDEX (RANGE SCAN)		ANALYZED	1	4	0



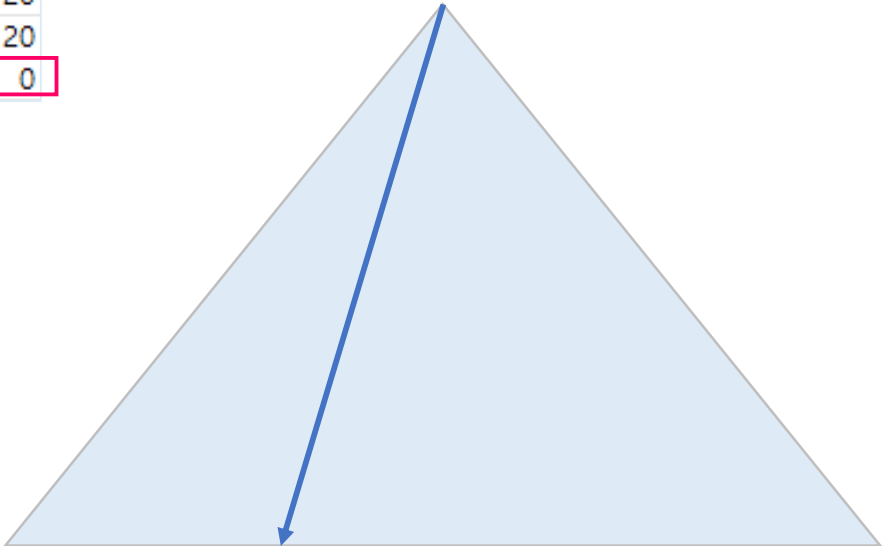
Index Range Scan

➤ 인덱스 유일 스캔

- ① 인덱스를 사용하여 단 하나의 데이터를 추출하는 방식
- ② 유일 인덱스는 중복 레코드를 허용하지 않음
- ③ 유일 인덱스는 반드시 '='조건으로 조회 해야 됨(그렇게 할 수 밖에 없음)

```
CREATE TABLE TB_EMP_TMP AS SELECT * FROM TB_EMP;  
CREATE UNIQUE INDEX IDX_TB_EMP_TMP_01 ON TB_EMP_TMP(EMP_NO);  
  
SELECT *  
FROM TB_EMP_TMP  
WHERE EMP_NO = '1000000013';
```

Operation	Object	Optimizer	Cost	Cardinality	Bytes
▼ SELECT STATEMENT		ALL_ROWS	1	1	120
▼ TABLE ACCESS (BY INDEX ROWID)	TB_EMP_TMP	ANALYZED	1	1	120
INDEX (UNIQUE SCAN)	IDX_TB_EMP_TMP_01	ANALYZED	0	1	0



Index Unique Scan

➤ 인덱스 전체 스캔

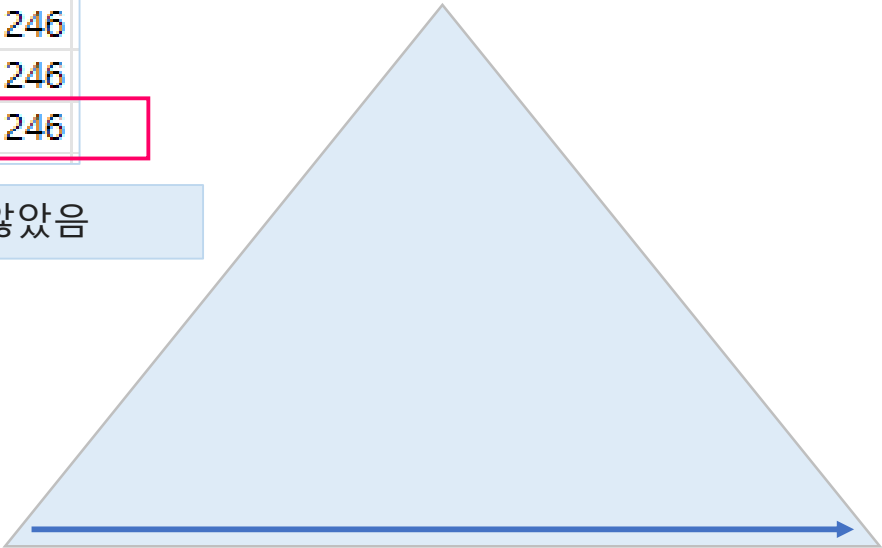
- ① 인덱스를 처음부터 끝까지 전체를 읽으면서 조건에 맞는 데이터를 추출함
- ② 데이터를 추출 시 리프 블록에 있는 ROWID로 테이블의 레코드를 찾아가서 조건에 부합하는지 판단하고 조건에 부합되면 해당 행을 리턴 함

```
CREATE INDEX IDX_TB_EMP_02 ON TB_EMP(FINAL_EDU_SE_CD, TEL_NO);
```

```
SELECT A.FINAL_EDU_SE_CD, A.TEL_NO  
FROM TB_EMP A  
WHERE A.TEL_NO <> '010-9'  
ORDER BY A.TEL_NO  
;
```

Operation	Object	Optimizer	Cost	Cardinality	Bytes
▼ SELECT STATEMENT		ALL_ROWS	2	3	246
▼ SORT (ORDER BY)			2	3	246
INDEX (FULL SCAN)	IDX_TB_EMP_02	ANALYZED	1	3	246

❖ 위 SQL문은 인덱스 구성 컬럼만을 SELECT하기 때문에 테이블까지 가지 않았음



Index Full Scan

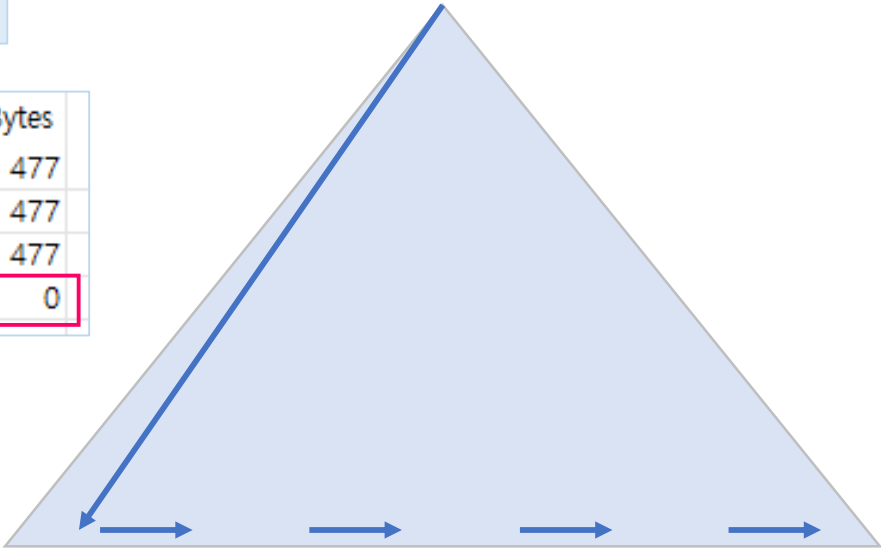
➤ 인덱스 스킵 스캔

- ① 인덱스 선두 컬럼이 조건절에 없어도 인덱스를 활용하는 스캔 방식이다.
- ② 조건절에 빠진 인덱스 선두 컬럼의 Distinct Value의 개수가 적고, 후행 컬럼의 Distinct Value의 개수가 많을 때 유용
- ③ 루트 또는 브랜치에서 읽은 컬럼 값 정보를 이용해 조건절에 부합하는 레코드를 포함할 가능성이 있는 리프 블록만 액세스한다.

```
CREATE INDEX IDX_TB_EMP_02 ON TB_EMP(FINAL_EDU_SE_CD, TEL_NO);

SELECT /*+ INDEX_SS(A IDX_TB_EMP_02) */
      A.FINAL_EDU_SE_CD, A.TEL_NO, A.EMP_NM
FROM   TB_EMP A
WHERE  A.TEL_NO > '010-9'
ORDER BY A.TEL_NO, A.EMP_NM
;
```

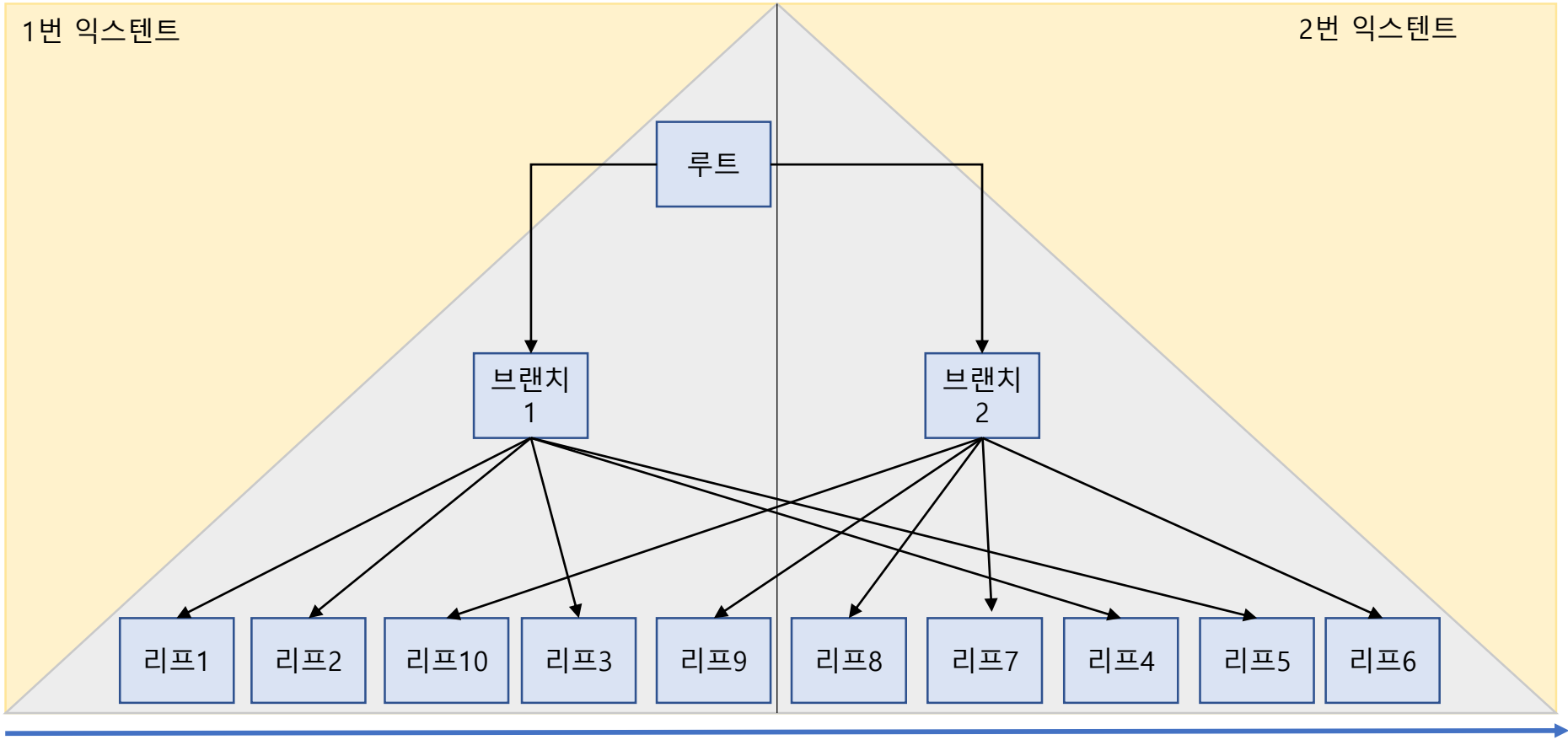
Operation	Object	Optimizer	Cost	Cardinality	Bytes
SELECT STATEMENT		ALL_ROWS	3	3	477
SORT (ORDER BY)			3	3	477
TABLE ACCESS (BY INDEX ROWID) TB_EMP	TB_EMP		2	3	477
INDEX (SKIP SCAN)		ANALYZED	1	1	0



Index Skip Scan

➤ 인덱스 고속 전체 스캔

- ① Index Fast Full Scan은 물리적으로 디스크에 저장된 순서대로 인덱스 리프 블록들을 Multi Block I/O 방식으로 읽어 들인다. 또한 병렬 인덱스 스캔도 가능하다.



인덱스 리프 노드가 갖는 연결 구조를 무시한 채 데이터를 읽기 때문에 인덱스 정렬 순서의 보장을 하지 못한다.

➤ 인덱스 고속 전체 스캔 - 계속

```
CREATE INDEX IDX_TB_EMP_02 ON TB_EMP(FINAL_EDU_SE_CD, TEL_NO);

SELECT /*+ INDEX_FFS(A IDX_TB_EMP_02) */
      A.FINAL_EDU_SE_CD, A.TEL_NO
FROM   TB_EMP A
WHERE  A.TEL_NO > '010-9'
ORDER BY A.TEL_NO
;
```

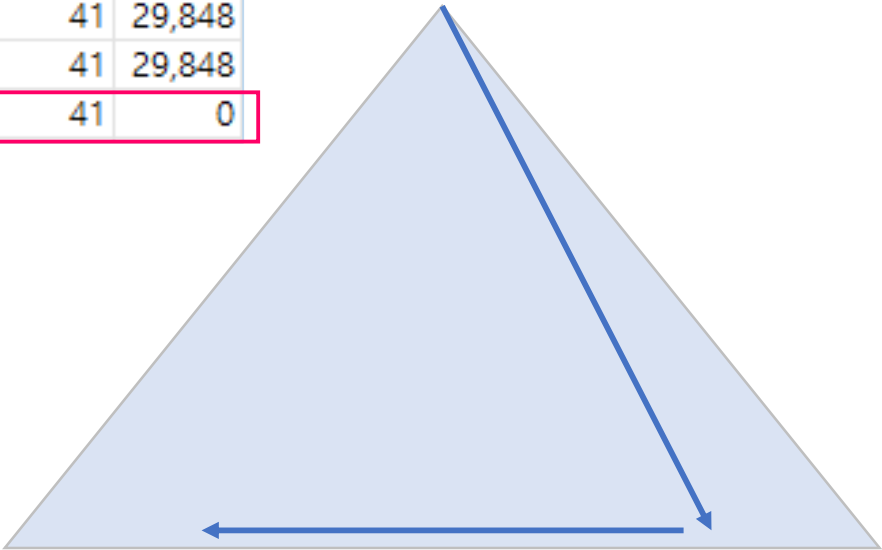
Operation	Object	Optimizer	Cost	Cardinality	Bytes
▼ SELECT STATEMENT		ALL_ROWS	3	3	246
▼ SORT (ORDER BY)			3	3	246
INDEX (FAST FULL SCAN)	IDX_TB_EMP_02	ANALYZED	2	3	246

➤ 인덱스 역순 범위 스캔

- ① 인덱스 리프 블록은 Doubly Linked List방식으로 저장되어 있음
- ② 즉 이 성질을 이용하여 인덱스를 역순으로(거꾸로) 읽을 수 있음
- ③ 인덱스를 뒤에서부터 앞쪽으로 스캔하기 때문에 내림차순으로 정렬된 결과 집합을 얻을 수 있다. (스캔 순서를 제외하고는 Range Scan과 동일함)

```
SELECT *  
FROM TB_EMP A  
WHERE A.EMP_NO > '0'  
ORDER BY A.EMP_NO DESC
```

Operation	Object	Optimizer	Cost	Cardinality	Bytes
▼ SELECT STATEMENT		ALL_ROWS	1	41	29,848
▼ TABLE ACCESS (BY INDEX ROWID)	<a href="#">TB_EMP</a>		1	41	29,848
INDEX (RANGE SCAN DESCENDING)	<a href="#">PK_TB_EMP</a>	ANALYZED	1	41	0



Index Range Scan Descending



➤ 테이블 스캔 VS 인덱스 스캔

테이블 풀 스캔	인덱스 스캔
항상 이용 가능	인덱스가 존재해야만 이용가능
한번에 여러 개의 BLOCK을 읽음	한번에 한 개에 블록만을읽음
많은 데이터를 조회 시 성능 상 유리	극히 일부분의 데이터를 조회 시 유리
Table Random Access부하 없음	Table Random Access에 의한 부하가 발생됨
읽었던 블록을 반복해서 읽는 경우 없음	읽었던 블록을 반복해서 읽는 비효율 발생(논리적인 블록 I/O의 개수도 많아짐)

- ❖ 인덱스 스캔은 생각 했던 것보다 훨씬 부하가 큰 작업이다.
- ❖ 즉 반드시 인덱스 스캔이 테이블 풀 스캔보다 성능이 좋다고 생각하는 것은 금물이다.
- ❖ 데이터 건수가 많은 테이블에서 **소량의 데이터를 스캔 할 때** 사용해야 한다.
- ❖ 즉 인덱스 스캔 효율을 높여서 절대적인 논리적 I/O를 줄이는 노력을 해야한다.

Chapter 05. SQL 최적화 기본 원리

# 5-3. 조인 수행 원리

## ➤ 조인이란?

- ① 조인이란 **두개 이상의 테이블을 하나의 집합으로 만드는 연산**이다.
- ② SQL문의 FROM절에 두개 이상의 테이블 혹은 집합이 존재할 경우 조인이 수행된다.
- ③ 조인은 3개 이상의 테이블을 조인한다고 하더라도 **특정 시점에 2개의 테이블 단위로 조인**이 된다.
- ④ A, B, C 집합을 조인한다면 **A, B조인 후 해당 결과 집합을 C와 조인** 하는 방식이다.
- ⑤ **각각의 조인 단계에서는 서로 다른 조인 기법(방법)이 사용**될 수 있다.
- ⑥ 즉 A, B조인 시에는 NL조인을 수행하고 A, B조인의 결과와 C를 조인 시에는 해시 조인이 수행될 수 있다.

➤ NL 조인

```
SELECT *
FROM TB_EMP A
, TB_DEPT B
WHERE A.EMP_NO = '1000000001'
AND A.DEPT_CD = B.DEPT_CD
;
```

NL 조인은 작은 집합이 Driving되어야 하고 Inner 테이블의 인덱스 스캔이 매우 중요하다.

Id	Operation	Name	Starts	A-Rows	A-Time	Buffers
0	SELECT STATEMENT		1	1	00:00:00.01	4
1	NESTED LOOPS		1	1	00:00:00.01	4
2	TABLE ACCESS BY INDEX ROWID	TB_EMP	1	1	00:00:00.01	2
* 3	INDEX UNIQUE SCAN	PK_TB_EMP	1	1	00:00:00.01	1
4	TABLE ACCESS BY INDEX ROWID	TB_DEPT	1	1	00:00:00.01	2
* 5	INDEX UNIQUE SCAN	PK_TB_DEPT	1	1	00:00:00.01	1

- ❖ RANDOM 액세스 위주(인덱스구성이 완벽 해도 대량 데이터 조인 시 불리)
- ❖ 한 레코드 씩 순차 진행(부분 범위 처리를 유도해야 효율적 수행)
- ❖ DRIVING 테이블 처리 범위에 의해 전체 성능이 결정됨
- ❖ 인덱스 유무, 인덱스 구성에 크게 영향 받음
- ❖ 소량의 데이터를 처리하거나 부분범위처리가 가능한 OLTP 환경에 적합

➤ 소트 머지 조인

```
SELECT /* SELECT.TB_EMP_TB_DEPT.002 */  
      /*+ USE_MERGE(A B) */  
      *  
FROM TB_EMP A  
     , TB_DEPT B  
WHERE A.DEPT_CD = B.DEPT_CD  
;
```

정렬 작업을 생략할 수 있는 인덱스가 존재하는 경우 사용!

Id	Operation	Name	Starts	A-Rows	A-Time	Buffers	OMem	lMem	Used-Mem
0	SELECT STATEMENT		1	41	00:00:00.01	11			
1	MERGE JOIN		1	41	00:00:00.01	11			
2	TABLE ACCESS BY INDEX ROWID	TB_DEPT	1	14	00:00:00.01	4			
3	INDEX FULL SCAN	PK_TB_DEPT	1	14	00:00:00.01	2			
* 4	SORT JOIN		14	41	00:00:00.01	7	9216	9216	8192 (0)
5	TABLE ACCESS FULL	TB_EMP	1	41	00:00:00.01	7			

- ❖ 실시간 인덱스 생성 : 양쪽 집합을 정렬한 다음에는 NL 조인과 같은 오퍼레이션
- ❖ 인덱스 유무에 영향을 받지 않음 : 미리 정렬된 인덱스가 있으면 좀 더 빠르게 수행할 수는 있음
- ❖ 양쪽 집합을 개별적으로 읽고 나서 조인 : 조인 컬럼에 인덱스가 없는 상황에서 두 테이블을 독립적으로 읽어 조인 대상 집합을 줄일 수 있을 때 아주 유리
- ❖ 스캔(Scan) 위주의 액세스 방식 : 양쪽 소스 집합에서 정렬 대상 레코드를 찾는 작업은 인덱스를 이용 Random 액세스 방식으로 처리될 수 있음

➤ 해시 조인

```
SELECT /* SELECT.TB_EMP_TB_DEPT.003 */  
      /*+ USE_HASH(A B) */  
      *  
FROM TB_EMP A  
     , TB_DEPT B  
WHERE A.EMP_NO = '1000000001'  
      AND A.DEPT_CD = B.DEPT_CD  
;
```

작은 집합을 build Input  
으로 하고 큰 집합을  
probe input으로 하는  
것이 중요

Id	Operation	Name	Starts	A-Rows	A-Time	Buffers	OMem	lMem	Used-Mem
0	SELECT STATEMENT		1	1	00:00:00.01	10			
* 1	HASH JOIN		1	1	00:00:00.01	10	833K	833K	435K (0)
2	TABLE ACCESS BY INDEX ROWID	TB_EMP	1	1	00:00:00.01	2			
* 3	INDEX UNIQUE SCAN	PK_TB_EMP	1	1	00:00:00.01	1			
4	TABLE ACCESS FULL	TB_DEPT	1	14	00:00:00.01	8			

- ❖ 대량의 데이터 처리가 필요하고 쿼리 수행 시간이 오래 걸리는 대용량 테이블을 조인할 때(배치 프로그램, DW, OLAP성 쿼리) 사용
- ❖ NL 조인처럼 Random 액세스 부하 없음
- ❖ 소트 머지 조인처럼 정렬 부하 없음
- ❖ 해시 테이블을 생성하는 비용에 따라서 Build Input이 (Hash Area에 담을 수 있을 정도로 충분히) 작을 때라야 효과적

➤ 조인 기법 비교

NL Join	Sort Merge Join	Hash Join
순차적 Random Access Join 조건이 중요함 Join 방향성 부분 범위 처리 가능 ...	동시적 전체 범위 처리(일부분 제외) Join 조건 무관 Join 무 방향성 Sort 부하 PGA 과다 사용 우려 ...	Hash Area Size중요 등치 조인이 매우 중요 대량 범위 처리 유리 배치, SP처리에 유리 ...
OLTP	-	OLAP

- ❖ 집합적인 사고를 바탕으로 적절한 Join Method 선정이 중요하다.
- ❖ 시스템의 특성을 참고하여 적절한 Join Method 선정이 중요하다.

➤ 조인 순서의 중요성

항목	설명	비고
First Table	<ul style="list-style-type: none"><li>• 두개의 Table을 조인 할 경우 먼저 처리되는 테이블을 의미한다.</li><li>• WHERE절에 상수/바인드 변수 조건이 존재하는 것이 성능상 유리하다.</li></ul>	Outer Table Driving Table Build Input
Second Table	<ul style="list-style-type: none"><li>• 두개의 테이블을 조인 할 경우 뒤에 처리되는 테이블을 의미한다.</li><li>• First Table로 부터 입력 값을 받아서 처리하게 된다.</li><li>• 조인 조건의 여부 및 성질이 조인 조건이 성능에 영향을 미친다.</li><li>• 조인 조건 및 상수/바인드 변수 조건에 인덱스 존재 여부가 매우 중요하다. (NL Join에 경우)</li></ul>	Inner Table Driven Table Probe Input
최적화된 Join Order	<ul style="list-style-type: none"><li>• First Table이 Second Table에 비해서 작은 집합 이어야 성능상 유리하다. (NL, Hash Join에 경우)</li></ul>	



Chapter 05. SQL 최적화 기본 원리

# 5-4. 연습문제

문제1. 전체 사원이 1000명인 회사에서 아래의 SQL문을 수행하려고 한다. 다음 중 옵티마이저가 예상하는 건수로 가장 적절한 것은?

```
SELECT * FROM 사원 WHERE 직급 = '부장' AND 부서 = '영업';
```

직급은 {부장, 과장, 대리, 사원}이 존재한다.  
부서는 {인사, 총무, 회계, 마케팅, 영업}이 존재한다.

- ① 250
- ② 200
- ③ 111.1
- ④ 50

문제2. 다음 중 NL조인의 특징을 설명한 것으로 가장 부적절한 것은?

- ① 조인을 한 로우 씩 차례대로 진행하므로 부분 범위 처리에 자주 사용된다.
- ② 먼저 액세스한 테이블의 전체 범위에 따라 전체 비용이 결정된다.
- ③ INNER쪽 조인 칼럼에 대한 인덱스 전략이 중요하지만, 조인 칼럼이 모두 포함되지 않은 인덱스라도 사용가능하다.
- ④ OUTER쪽은 반드시 인덱스를 타야 효율적이다.

문제1. 전체 사원이 1000명인 회사에서 아래의 SQL문을 수행하려고 한다. 다음 중 옵티마이저가 예상하는 건수로 가장 적절한 것은?

```
SELECT * FROM 사원 WHERE 직급 = '부장' AND 부서 = '영업';
```

직급은 {부장, 과장, 대리, 사원}이 존재한다.  
부서는 {인사, 총무, 회계, 마케팅, 영업}이 존재한다.

- ① 250
- ② 200
- ③ 111.1
- ④ 50

문제2. 다음 중 NL조인의 특징을 설명한 것으로 가장 부적절한 것은?

- ① 조인을 한 로우 씩 차례대로 진행하므로 부분 범위 처리에 자주 사용된다.
- ② 먼저 액세스한 테이블의 전체 범위에 따라 전체 비용이 결정된다.
- ③ INNER쪽 조인 칼럼에 대한 인덱스 전략이 중요하지만, 조인 칼럼이 모두 포함되지 않은 인덱스라도 사용가능하다.
- ④ OUTER쪽은 반드시 인덱스를 타야 효율적이다.

문제3. 아래는 NL조인을 유도한 SQL문의 실행계획이다. 테이블의 조인 순서와 조인 횟수가 올바르게 짝지어진 것은 무엇인가?

Id	Operation	Name	Starts	A-Rows	A-Time	Buffers
0	SELECT STATEMENT		1	41	00:00:00.01	53
1	NESTED LOOPS		1	41	00:00:00.01	53
2	NESTED LOOPS		1	41	00:00:00.01	12
3	TABLE ACCESS FULL	TB_EMP	1	41	00:00:00.01	8
* 4	INDEX UNIQUE SCAN	PK_TB_DEPT	41	41	00:00:00.01	4
5	TABLE ACCESS BY INDEX ROWID	TB_DEPT	41	41	00:00:00.01	41

- ① 조인순서 : TB\_EMP -> TB\_DEPT 조인횟수 : 1
- ② 조인순서 : TB\_EMP -> TB\_DEPT 조인횟수 : 41
- ③ 조인순서 : TB\_DEPT -> TB\_EMP 조인횟수 : 1
- ④ 조인순서 : TB\_DEPT -> TB\_EMP 조인횟수 : 41

문제3. 아래는 NL조인을 유도한 SQL문의 실행계획이다. 테이블의 조인 순서와 조인 횟수가 올바르게 짝지어진 것은 무엇인가?

Id	Operation	Name	Starts	A-Rows	A-Time	Buffers
0	SELECT STATEMENT		1	41	00:00:00.01	53
1	NESTED LOOPS		1	41	00:00:00.01	53
2	NESTED LOOPS		1	41	00:00:00.01	12
3	TABLE ACCESS FULL	TB_EMP	1	41	00:00:00.01	8
* 4	INDEX UNIQUE SCAN	PK_TB_DEPT	41	41	00:00:00.01	4
5	TABLE ACCESS BY INDEX ROWID	TB_DEPT	41	41	00:00:00.01	41

- ① 조인순서 : TB\_EMP -> TB\_DEPT 조인횟수 : 1
- ② 조인순서 : TB\_EMP -> TB\_DEPT 조인횟수 : 41
- ③ 조인순서 : TB\_DEPT -> TB\_EMP 조인횟수 : 1
- ④ 조인순서 : TB\_DEPT -> TB\_EMP 조인횟수 : 41

## 문제4. 다음 중 아래 SQL문과 인덱스 구성에 대한 설명으로 옳바르지 않는 것은?

```
SELECT *  
FROM TB_TEMP_1  
WHERE COL_1 = '1'  
      AND COL_2 LIKE '1%'  
      AND COL_3 IN ('1', '5')  
      AND COL_4 BETWEEN '1' AND '10'  
;
```

인덱스 구성 : COL\_1 + COL2 + COL3  
모든 칼럼은 VARCHAR2 타입이라고 가정한다.

- ① COL\_1 조건이 없었다면 정상적인 인덱스 범위 스캔을 할 수 없었을 것이다.
- ② COL\_2 조건은 인덱스의 스캔 범위를 줄이는데 기여한다.
- ③ COL\_3 조건은 인덱스의 스캔 범위를 줄이는데 기여한다.
- ④ COL\_4 조건은 인덱스의 스캔 범위를 줄이는데 기여하지 못하지만 테이블 필터 처리를 줄여준다.

## 문제4. 다음 중 아래 SQL문과 인덱스 구성에 대한 설명으로 옳바르지 않는 것은?

```
SELECT *  
FROM TB_TEMP_1  
WHERE COL_1 = '1'  
      AND COL_2 LIKE '1%'  
      AND COL_3 IN ('1', '5')  
      AND COL_4 BETWEEN '1' AND '10'  
;
```

인덱스 구성 : COL\_1 + COL2 + COL3  
모든 칼럼은 VARCHAR2 타입이라고 가정한다.

- ① COL\_1 조건이 없었다면 정상적인 인덱스 범위 스캔을 할 수 없었을 것이다.
- ② COL\_2 조건은 인덱스의 스캔 범위를 줄이는데 기여한다.
- ③ COL\_3 조건은 인덱스의 스캔 범위를 줄이는데 기여한다.
- ④ COL\_4 조건은 인덱스의 스캔 범위를 줄이는데 기여하지 못하지만 테이블 필터 처리를 줄여준다.

### 문제5. 인덱스에 대한 설명으로 가장 적절한 것은?

- ① PRIMARY KEY 인덱스에 중복된 칼럼이 나타날 수 있다.
- ② PRIMARY KEY 인덱스에 널 값이 허용된다.
- ③ 자주 변경되는 칼럼은 인덱스로 관리하는게 성능 상 유리하다.
- ④ PRIMARY KEY 인덱스가 아니고 선두 칼럼이 아니라면 널이 허용된다.

### 문제6. 다음 중 옵티마이저에 대한 설명으로 틀린 것은?

- ① SQL문의 결과 집합이 같아도 수행 속도는 차이가 날수 있다.
- ② 옵티마이저에 의해서 내부실행계획이 정해지므로 결과 집합에 차이가 날수 있다.
- ③ 사용자가 제시하는 SQL문은 요구 사항 일 뿐이며 내부 연산 처리는 옵티마이저가 담당한다.
- ④ 사용자가 부여한 조건은 실행 계획에 영향을 준다.

### 문제7. 옵티마이저에 대한 설명으로 적절하지 않은 것은?

- ① 옵티마이저는 질의에 대한 실행계획을 생성한다.
- ② 옵티마이저는 비용 계산을 위해서 다양한 통계정보를 이용한다.
- ③ 옵티마이저는 통계 정보가 없다면 실행계획을 생성하지 못한다.
- ④ 옵티마이저는 인덱스가 존재해도 인덱스 스캔을 할지 말지 결정한다.



### 문제5. 인덱스에 대한 설명으로 가장 적절한 것은?

- ① PRIMARY KEY 인덱스에 중복된 칼럼이 나타날 수 있다.
- ② PRIMARY KEY 인덱스에 널 값이 허용된다.
- ③ 자주 변경되는 칼럼은 인덱스로 관리하는게 성능 상 유리하다.
- ④ PRIMARY KEY 인덱스가 아니고 선두 칼럼이 아니라면 널이 허용된다.

### 문제6. 다음 중 옵티마이저에 대한 설명으로 틀린 것은?

- ① SQL문의 결과 집합이 같아도 수행 속도는 차이가 날수 있다.
- ② 옵티마이저에 의해서 내부실행계획이 정해지므로 결과 집합에 차이가 날수 있다.
- ③ 사용자가 제시하는 SQL문은 요구 사항 일 뿐이며 내부 연산 처리는 옵티마이저가 담당한다.
- ④ 사용자가 부여한 조건은 실행 계획에 영향을 준다.

### 문제7. 옵티마이저에 대한 설명으로 적절하지 않은 것은?

- ① 옵티마이저는 질의에 대한 실행계획을 생성한다.
- ② 옵티마이저는 비용 계산을 위해서 다양한 통계정보를 이용한다.
- ③ 옵티마이저는 통계 정보가 없다면 실행계획을 생성하지 못한다.
- ④ 옵티마이저는 인덱스가 존재해도 인덱스 스캔을 할지 말지 결정한다.

### 문제8. 실행 계획에 대한 설명으로 적절하지 않은 것은?

- ① 실행계획은 SQL문의 처리를 위한 절차와 방법이 표현된다.
- ② 실행계획은 액세스 기법, 조인 순서, 조인방법등으로 구성된다.
- ③ 실행계획이 다르면 결과 집합도 달라질 수 있다.
- ④ 실행 계획의 예상 실행 비용이 크더라도 실제로는 작을 수 있다.

### 문제9. 조인에 대한 설명으로 적절하지 않은 것은?

- ① FROM절에 존재하는 모든 테이블을 동시에 조인한다.
- ② NL조인은 중첩 반복문과 유사한 형식이다.
- ③ NL조인은 선행 테이블에 조건을 만족하는 건수 만큼 반복 수행된다.
- ④ 해시 조인은 작은 테이블을 선행테이블로 하는게 성능상 유리하다.

### 문제10. 다음 중 해시 조인에 대한 특징 중 올바른 것은?

- ① 해시함수의 알고리즘을 사용자가 직접 제어 할 수 있다.
- ② 오라클 외부에 해시 평선을 사용한다.
- ③ 조인 조건의 인덱스 유무의 영향을 받지 않지만 소팅 작업이 필요하다.
- ④ 선행 테이블과 후행 테이블의 결과 집합의 차이가 큰 경우 성능 상 유리하다.

### 문제8. 실행 계획에 대한 설명으로 적절하지 않은 것은?

- ① 실행계획은 SQL문의 처리를 위한 절차와 방법이 표현된다.
- ② 실행계획은 액세스 기법, 조인 순서, 조인방법등으로 구성된다.
- ③ 실행계획이 다르면 결과 집합도 달라질 수 있다.
- ④ 실행 계획의 예상 실행 비용이 크더라도 실제로는 작을 수 있다.

### 문제9. 조인에 대한 설명으로 적절하지 않은 것은?

- ① FROM절에 존재하는 모든 테이블을 동시에 조인한다.
- ② NL조인은 중첩 반복문과 유사한 형식이다.
- ③ NL조인은 선행 테이블에 조건을 만족하는 건수 만큼 반복 수행된다.
- ④ 해시 조인은 작은 테이블을 선행테이블로 하는게 성능상 유리하다.

### 문제10. 다음 중 해시 조인에 대한 특징 중 올바른 것은?

- ① 해시함수의 알고리즘을 사용자가 직접 제어 할 수 있다.
- ② 오라클 외부에 해시 평선을 사용한다.
- ③ 조인 조건의 인덱스 유무의 영향을 받지 않지만 소팅 작업이 필요하다.
- ④ 선행 테이블과 후행 테이블의 결과 집합의 차이가 큰 경우 성능 상 유리하다.

문제11. 다음 중 해시 조인에 대한 설명으로 부적절한 것은 무엇인가?

- ① 각 테이블에 반드시 인덱스가 필요한 것은 아니다.
- ② 일반적으로 작은 테이블을 메모리에 올리는 선행 테이블로 사용한다.
- ③ 해시 함수의 특성상 NOT EQUAL JOIN도 가능하다.
- ④ 조인을 위해 사전 소트작업이 필요하지 않다.

문제12. 아래 실행 계획에 대한 설명으로 올바른 것은 무엇인가?

```
Execution Plan
-----
0      SELECT STATEMENT Optimizer=ALL_ROWS (Cost=5 Card=10 Bytes=7K)
1    0      HASH JOIN (Cost=5 Card=10 Bytes=7K)
2    1        TABLE ACCESS (FULL) OF 'TB_DEPT' (TABLE) (Cost=3 Card=14 Bytes=322)
3    1        VIEW (Cost=2 Card=10 Bytes=7K)
4    3          COUNT (STOPKEY)
5    4          TABLE ACCESS (FULL) OF 'TB_EMP' (TABLE) (Cost=2 Card=10 Bytes=1K)
-----
```

- ① TB\_EMP테이블과 TB\_DEPT테이블은 해시 조인 아우터로 연산이 수행되고 있다.
- ② TB\_EMP테이블이 선행 테이블로써 메모리에 저장된다.
- ③ 별도의 조건 혹은 인덱스가 없어서 테이블 풀 스캔을 하고있어 성능에 치명적이다.
- ④ EMP테이블을 조회 시 ROWNUM을 통한 부분범위처리가 작동 중이다.

문제11. 다음 중 해시 조인에 대한 설명으로 부적절한 것은 무엇인가?

- ① 각 테이블에 반드시 인덱스가 필요한 것은 아니다.
- ② 일반적으로 작은 테이블을 메모리에 올리는 선행 테이블로 사용한다.
- ③ 해시 함수의 특성상 NOT EQUAL JOIN도 가능하다.
- ④ 조인을 위해 사전 소트작업이 필요하지 않다.

문제12. 아래 실행 계획에 대한 설명으로 올바른 것은 무엇인가?

```
Execution Plan
-----
0      SELECT STATEMENT Optimizer=ALL_ROWS (Cost=5 Card=10 Bytes=7K)
1    0      HASH JOIN (Cost=5 Card=10 Bytes=7K)
2    1        TABLE ACCESS (FULL) OF 'TB_DEPT' (TABLE) (Cost=3 Card=14 Bytes=322)
3    1        VIEW (Cost=2 Card=10 Bytes=7K)
4    3          COUNT (STOPKEY)
5    4          TABLE ACCESS (FULL) OF 'TB_EMP' (TABLE) (Cost=2 Card=10 Bytes=1K)
-----
```

- ① TB\_EMP테이블과 TB\_DEPT테이블은 해시 조인 아우터로 연산이 수행되고 있다.
- ② TB\_EMP테이블이 선행 테이블로써 메모리에 저장된다.
- ③ 별도의 조건 혹은 인덱스가 없어서 테이블 풀 스캔을 하고있어 성능에 치명적이다.
- ④ EMP테이블을 조회 시 ROWNUM을 통한 부분범위처리가 작동 중이다.