12. 실행 계획 분석과 인덱스 최적화

(1) 액세스 경로 개념 및 테이블액세스 경로

액세스 경로 소개

- 액세스 경로는 쿼리가 행 소스에서 행을 검색하는 데 사용하는 기법입니다.
  - 행 소스는 실행 계획의 각 단계에서 반환되는 행 집합입니다.
  - 행 소스는 테이블, 뷰 또는 조인이나 그룹화 연산의 결과일 수 있습니다.
- 접근 경로와 같은 단항 연산은 하나의 데이터 원천(행 소스)으로부터 데이터를 가져오는 방식입니다.
  - 예를 들어, 전체 테이블 스캔은 단일 테이블로부터 모든 행을 순차적으로 읽어오는 연산입니다.
  - 반면 조인은 이진 연산이며 정확히 두 개의 행 소스에서 입력을 받습니다.
- 데이터베이스는 테이블과 인덱스 등 서로 다른 관계형 데이터 구조에 대해 서로 다른 액세스 경로를 사용합니다.

힙 구성 테이블 (Heap-Organized Table) 액세스 정보

- 기본적으로 오라클 테이블은 힙으로 구성됩니다.
  - 데이터베이스는 사용자가 지정한 순서가 아닌 가장 적합한 위치에 행을 배치합니다.
  - 행은 삽입된 순서대로 검색되지 않을 수 있습니다.
- 힙 테이블의 Full Table Scan
  - 전체 테이블 스캔 중에 데이터베이스는 포맷된 것으로 알려진 HWM까지의 모든 블록을 읽습니다.
  - 하이 워터 마크(HWM)는 세그먼트에서 데이터 블록이 포맷되지 않고 사용되지 않은 지점을 말합니다.
  - 데이터베이스는 하이 워트 마크 이후의 블록들은 포맷되지 않았기 때문에 Full Table Scan시 읽지 않습니다.
  - DELETE 작업은 HWM를 줄이지 못하므로 삭제가 많이 발생한 테이블의 Full Table Scan의 성능은 좋지 않을 수 있습니다.

Full Table Scan: Example

- 이 예제에서는 hr.employees 테이블을 스캔하여 $4,000 이상의 월급을 쿼리합니다.
  - salary 열에 인덱스가 없으므로 옵티마이저는 인덱스 범위 스캔을 사용할 수 없으므로 전체 테이블 스캔을 사용합니다.

```
SELECT salary  FROM   employees
WHERE  salary > 4000;
SELECT * FROM TABLE(DBMS_XPLAN.DISPLAY_CURSOR(NULL, NULL, 'ALLSTATS LAST ALL +NOTE'));
```

```
Plan hash value: 1445457117

-----------------------------------------------------------------------------
| Id  | Operation          | Name      | E-Rows |E-Bytes| Cost (%CPU)| E-Time   |
-----------------------------------------------------------------------------
|   0 | SELECT STATEMENT   |           |        |       |     3 (100)|          |
|*  1 |  TABLE ACCESS FULL | EMPLOYEES |     64 |   832 |     3   (0)| 00:00:01 |
-----------------------------------------------------------------------------
```

Full Table Scan이 발생하는 일반적인 이유

이유 (Reason)	설명 (Explanation)
1. 인덱스가 없음	인덱스가 존재하지 않으면 옵티마이저는 전체 테이블 스캔을 수행함.
2. 인덱스 컬럼에 함수 적용	WHERE TO_NUMBER(char_col)=1처럼 함수가 적용되면, 일반 인덱스는 사용할 수 없어 전체 스캔 발생. 함수 기반 인덱스를 따로 생성하지 않은 경우.
3. SELECT COUNT(*) 쿼리에서 인덱스된 컬럼이 NULL 포함	인덱스는 NULL 값을 포함하지 않기 때문에, 카운트 계산 시 전체 테이블 스캔이 필요함.
4. B-트리 인덱스의 선행 열을 사용하지 않음	인덱스가 (first_name, last_name)인 경우 WHERE last_name='KING' 쿼리는 인덱스 사용 불가.
5. 쿼리가 비선택적임 (Unselective)	대부분의 행을 반환하는 쿼리는 인덱스를 쓰는 것보다 전체 테이블 스캔이 효율적일 수 있음.
6. 테이블 통계가 오래됨 (Stale statistics)	테이블 크기나 상태가 바뀌었는데 통계가 업데이트되지 않으면 옵티마이저는 잘못된 판단을 하게 됨.
7. 테이블이 작음	테이블이 매우 작으면, 인덱스보다 전체 테이블 스캔이 비용 효율적일 수 있음.
8. 높은 병렬성 설정 (High Parallelism)	병렬 쿼리 처리를 위해 전체 테이블 스캔이 선호될 수 있음.
9. FULL 힌트 사용	FULL(table_alias) 힌트를 명시적으로 사용하면 옵티마이저는 전체 테이블 스캔을 수행함.

행 액세스를 위한 ROWID의 중요성

- 힙 구조 테이블의 모든 행은 해당 테이블에 고유한 ROWID를 가지며, 이는 행 조각의 물리적 주소에 해당합니다.
  - ROWID는 특정 파일, 블록 및 행 번호를 가리킵니다.
- 물리적 행 ID는 테이블 행에 대한 가장 빠른 액세스를 제공하여 데이터베이스가 단 한 번의 I/O만으로 행을 검색할 수 있도록 합니다.
- Oracle Database는 인덱스 생성을 위해 내부적으로 ROWID를 사용합니다.
  - 예를 들어, B-트리 인덱스의 각 키는 연관된 행의 주소를 가리키는 행 ID와 연결됩니다.

Table Access by Rowid

- ROWID를 지정하여 행을 찾는 것은 행의 정확한 위치를 지정하기 때문에 단일 행을 검색하는 가장 빠른 방법입니다.
- 옵티마이저가 ROWID를 통한 테이블 액세스를 선택하는 경우
  - 대부분의 경우 데이터베이스는 하나 이상의 인덱스 스캔 후 ROWID로 테이블에 액세스합니다.
  - 인덱스에 필요한 모든 열이 포함되어 있는 경우 ROWID를 통한 액세스가 발생하지 않을 수 있습니다.
- ROWID를 통한 테이블 액세스 작동 방식

  1) WHERE 절 또는 하나 이상의 인덱스에 대한 인덱스 스캔을 통해 선택된 행의 ROWID를 가져옵니다.
  2) ROWID를 기반으로 테이블에서 선택된 각 행을 찾습니다.

Table Access by Rowid: Example

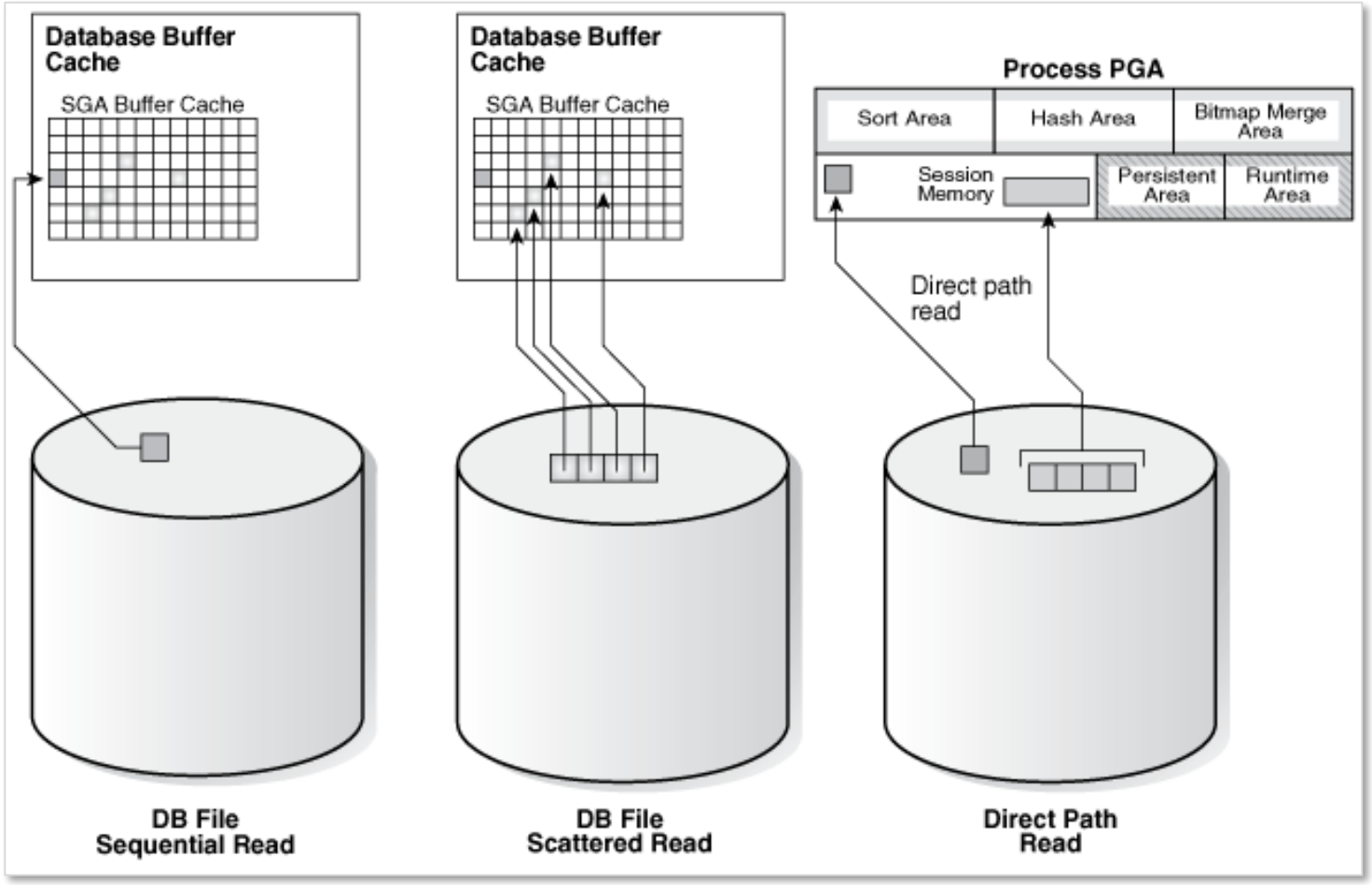
- 다음 실행 계획의 Step 2는 hr.employees 테이블에 있는 emp_emp_id_pk 인덱스에 대한 범위 스캔(range scan)을 나타냅니다.
  - 데이터베이스는 인덱스에서 얻은 ROWID를 사용해 해당하는 employees 테이블의 행을 찾아서 읽어옵니다.

```
SELECT * FROM   employees
WHERE  employee_id > 205;
SELECT * FROM TABLE(DBMS_XPLAN.DISPLAY_CURSOR(NULL, NULL, 'ALLSTATS LAST ALL +NOTE'));
```

```
Plan hash value: 2147325939

-----------------------------------------------------------------------------------------------
| Id  | Operation                           | Name         | E-Rows |E-Bytes| Cost (%CPU)| E-Time   |
-----------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT                    |              |        |       |     2 (100)|          |
|   1 |  TABLE ACCESS BY INDEX ROWID BATCHED| EMPLOYEES    |      1 |    69 |     2   (0)| 00:00:01 |
|*  2 |   INDEX RANGE SCAN                  | SYS_C008451  |      1 |       |     1   (0)| 00:00:01 |
-----------------------------------------------------------------------------------------------
```

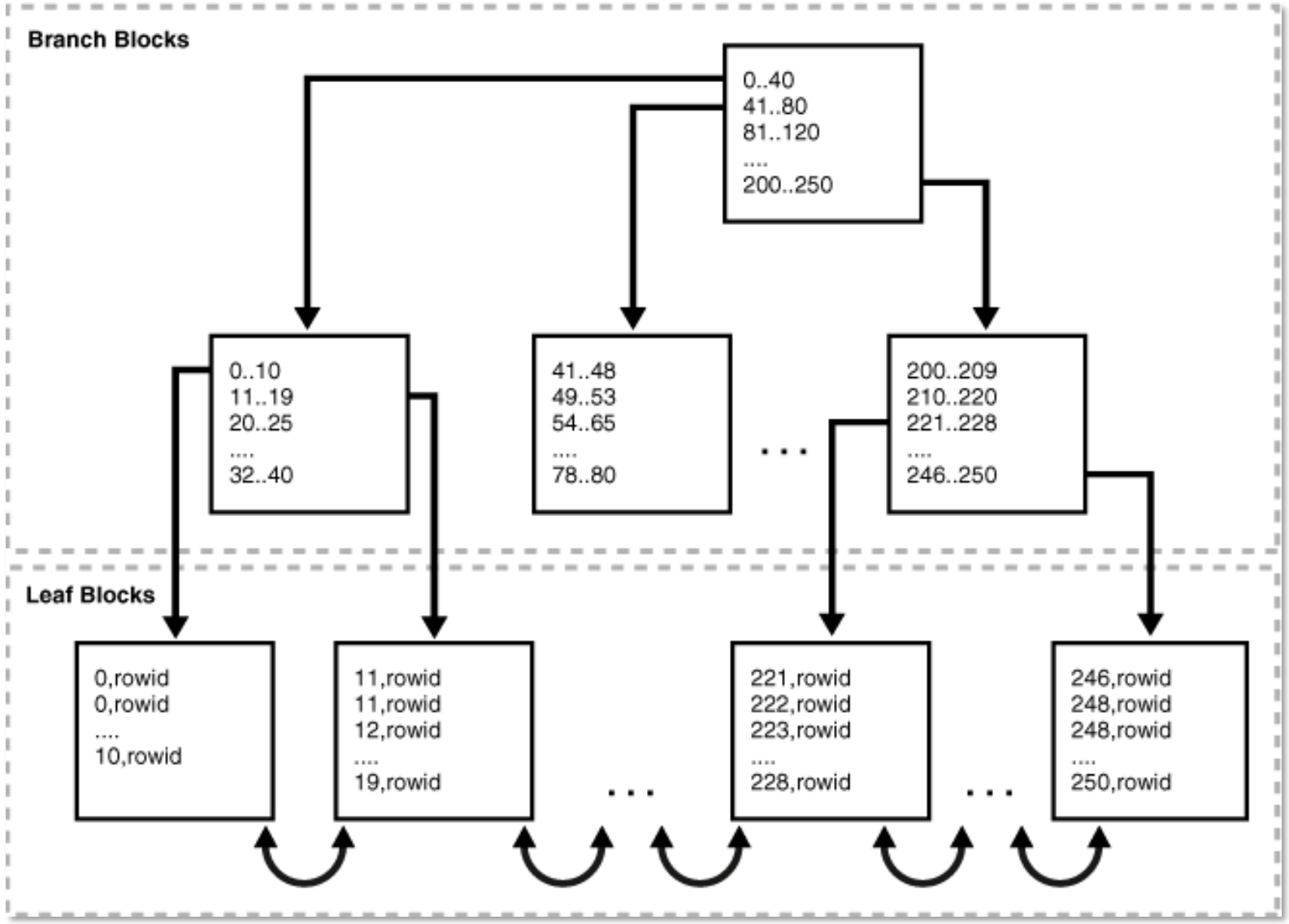
Direct Path Read

- Direct Path Read에서 데이터베이스는 SGA를 완전히 우회하여 디스크에서 PGA로 버퍼를 직접 읽습니다.
- 이 그림은 버퍼를 SGA에 저장하는 분산 읽기 및 순차 읽기와 직접 경로 읽기를 비교합니다.



(2) B-Tree Index Access Path

B-Tree Index Structure



B-Tree 인덱스 저장 방식이 인덱스 스캔에 미치는 영향

- 인덱스 블록 본문은 테이블 행과 마찬가지로 인덱스 항목을 힙에 저장하므로 블록 본문의 인덱스 항목은 키 순서대로 저장되지 않습니다.
  - 예를 들어, 값 10이 테이블에 삽입되면 키가 10인 인덱스 항목이 인덱스 블록의 맨 아래에 삽입된 후 테이블에 0이 삽입되면 키 0에 대한 인덱스 항목이 키 10에 대한 항목 위에 삽입될 수 있습니다.
- 인덱스 블록 내에서 행 헤더는 레코드를 키 순서대로 저장합니다.
  - 인덱스 스캔은 행 헤더를 읽어 범위 스캔의 시작 및 종료 위치를 결정할 수 있으므로 블록의 모든 항목을 읽을 필요가 없습니다.

Unique and Nonunique Indexes

- 비고유 인덱스에서 데이터베이스는 키를 고유하게 만들기 위해 키에 추가 열로 ROWID를 저장합니다.
  - 다음 그림과 같이 비고유 인덱스의 첫 번째 인덱스 키는 단순히 0이 아니라 0, ROWID 쌍입니다.
  - 데이터베이스는 인덱스 키 값을 기준으로 데이터를 정렬한 다음 ROWID를 오름차순으로 정렬합니다.
  - 예를 들어 항목은 다음과 같이 정렬됩니다.

```
0,AAAPvCAAFAAAAFaAAa
0,AAAPvCAAFAAAAFaAAg
0,AAAPvCAAFAAAAFaAAl
2,AAAPvCAAFAAAAFaAAm
```

- 고유 인덱스에서 인덱스 키는 ROWID를 포함하지 않습니다.
  - 데이터베이스는 0, 1, 2 등과 같이 인덱스 키 값을 기준으로만 데이터를 정렬합니다.

B-Tree Indexes and Nulls

- 단일 열 B-트리 인덱스는 Null을저장하지 않으며, 이는 옵티마이저가 액세스 경로를 선택하는 방식에 중요합니다.
- 다음 예제는 옵티마이저가 hr.employees 테이블의 모든 부서 ID에 대한 쿼리에 대해 전체 테이블 스캔을 선택하는 것을 보여줍니다.

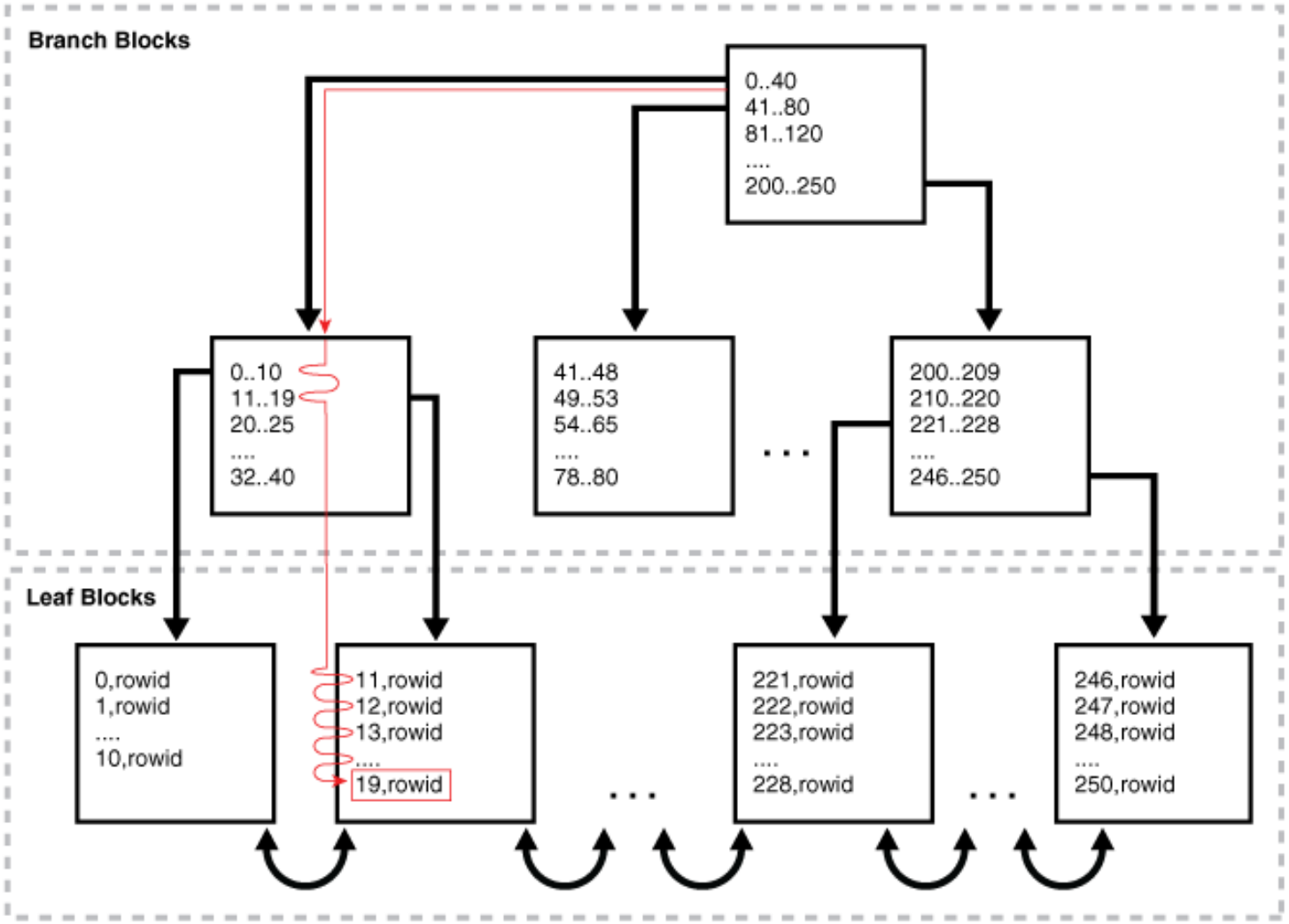
```
SQL> EXPLAIN PLAN FOR SELECT department_id FROM employees;

PLAN_TABLE_OUTPUT
------------------------------------------------------------------------------
Plan hash value: 3476115102

------------------------------------------------------------------------------
|Id | Operation         | Name      | Rows| Bytes | Cost (%CPU)| Time     |
------------------------------------------------------------------------------
| 0 | SELECT STATEMENT  |           | 107 |   321 |     2   (0)| 00:00:01 |
| 1 |  TABLE ACCESS FULL| EMPLOYEES | 107 |   321 |     2   (0)| 00:00:01 |
------------------------------------------------------------------------------
```

Index Unique Scans

- Index Unique Scan 은 최대 1개의 행 ID를 반환합니다.
- 옵티마이저는 데이터베이스는 쿼리 조건자가 WHERE prod_id=10과 같은 등호 연산자를 사용하여 고유 인덱스 키의 모든 열을 참조하는 경우에만 Unique Scan을 수행합니다.
- INDEX 힌트를 사용하면 사용할 인덱스를 지정할 수 있지만 특정 유형의 인덱스 액세스 경로는 지정할 수 없습니다.
- Index Unique Scans 작동 방식
  - 지정된 키를 순서대로 인덱스에서 검색하다가 첫 번째 레코드를 찾는 즉시 처리를 중단합니다.
  - 데이터베이스는 인덱스 항목에서 ROWID를 가져온 다음 ROWID로 지정된 행을 검색합니다.

Index Unique Scans 작동 방식

- 이 그림은 명령문이 prod_id 열에서 기본 키 인덱스를 가진 제품 ID 19에 대한 레코드를 요청하는 Index Unique Scan을보여줍니다.



Index Range Scans

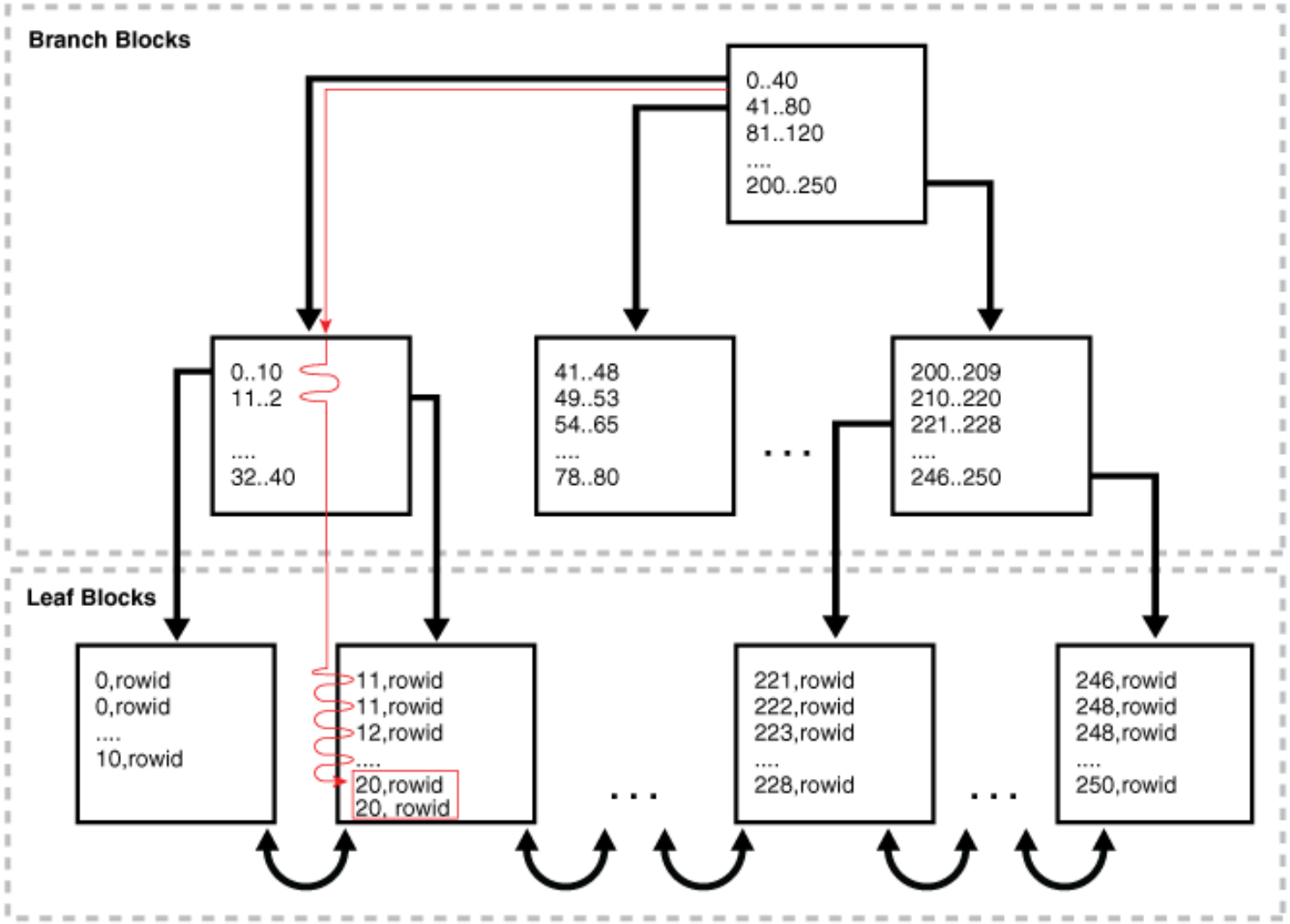
- 인덱스 범위 스캔은 값에 대한 순서 있는 스캔입니다.
  - 스캔의 범위는 양쪽 모두 제한되거나 한쪽 또는 양쪽 모두 제한되지 않을 수 있습니다.
  - 옵티마이저는 일반적으로 선택도가 높은 쿼리에 대해 범위 스캔을 선택합니다.
  - 기본적으로 데이터베이스는 인덱스를 오름차순으로 저장합니다.
- 옵티마이저는 인덱스의 선행 열이 하나 이상 조건에 지정된 경우 인덱스 범위 스캔을 고려합니다.
  - 예를들어 department_id = :id, department_id < :id, department_id > :id 또는 department_id > :low AND department_id < :hi 와 같은 경우 입니다.
- 옵티마이저가 전체 테이블 스캔이나 다른 인덱스를 선택하는 경우, 이 액세스 경로를 강제로 적용하기 위해 힌트를 사용할 수 있습니다.
  - 옵티마이저가 특정 인덱스를 사용하도록 INDEX(tbl_alias ix_name) 및 INDEX_DESC(tbl_alias ix_name) 힌트를 사용합니다.

Index Range Scan 작동방식

- 일반적으로 스캔 알고리즘은 다음과 같습니다.
  1) 루트 블록을 읽습니다.
  2) 브랜치 블록을 읽습니다.
  3) 모든 데이터를 검색할 때까지 다음 단계를 번갈아 수행합니다.
     - 리프 블록을 읽어 행 ID를 가져옵니다.
     - 테이블 블록을 읽어 행을 검색합니다.
- 이와 같이 인덱스를 스캔하기 위해 데이터베이스는 리프 블록을 앞뒤로 이동합니다.
  - 예를 들어, 20에서 40 사이의 ID를 스캔하는 경우 키 값이 20 이상인 가장 낮은 첫 번째 리프 블록을 찾습니다.
  - 스캔은 리프 노드의 연결 리스트를 가로로 진행하여 40보다 큰 값을 찾은 후 중지합니다.

Index Range Scan 작동방식 : Example

- 다음 그림은 오름차순을 사용하는 인덱스 범위 스캔을 보여줍니다.
  - 명령문은 고유하지 않은 인덱스를 가진 department_id 열의 값이 20인 직원 레코드를 요청합니다.
  - 이 예에서는 20 부서에 대한 인덱스 항목이 2개 있습니다.

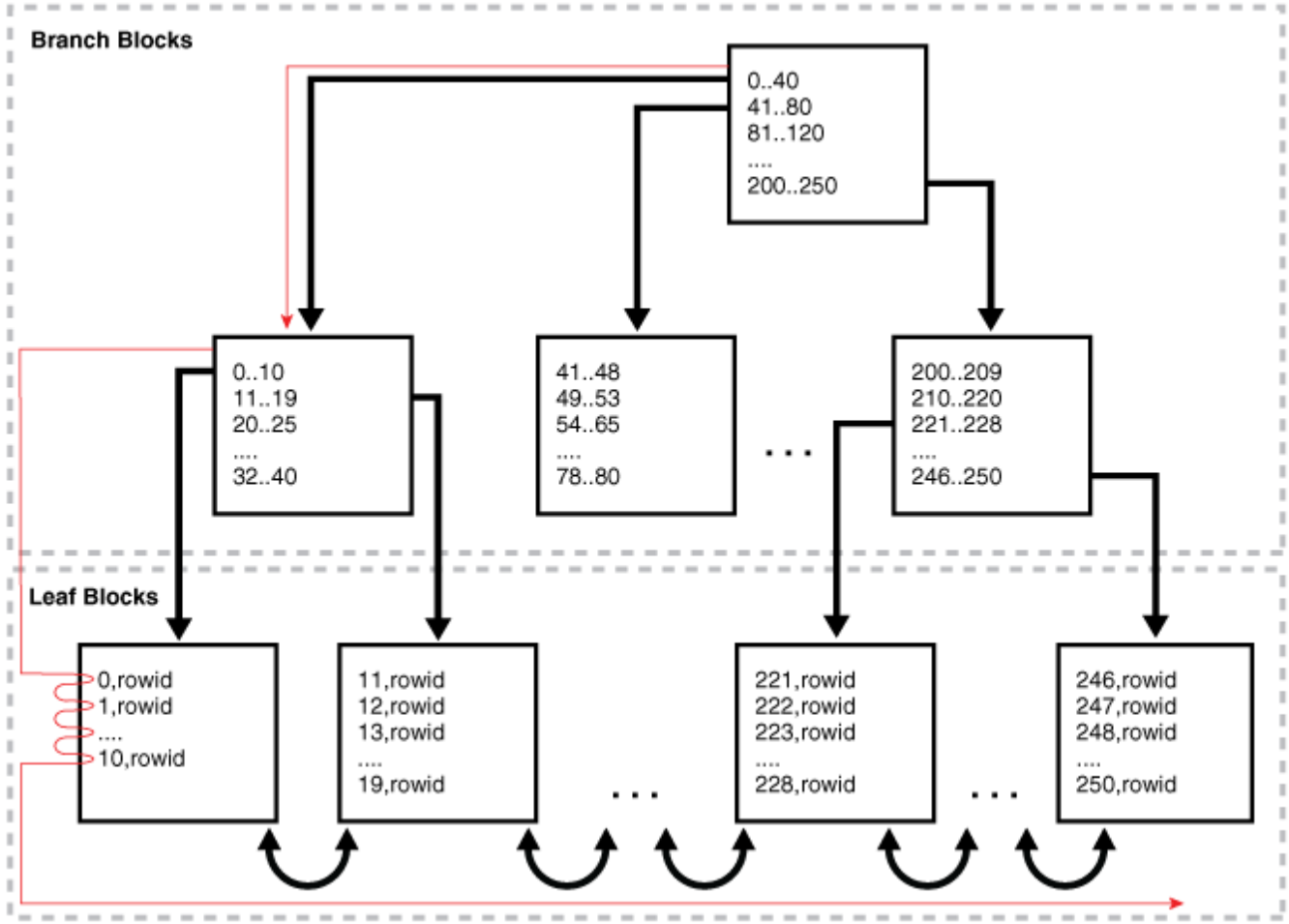


Index Full Scan

- 인덱스 전체 스캔은 전체 인덱스를 순서대로 읽습니다.
  - 인덱스의 데이터가 인덱스 키순으로 정렬되어 있으므로 인덱스 전체 스캔을 사용하면 별도의 정렬 작업이 필요 없습니다.
- 옵티마이저는 다양한 상황에서 인덱스 전체 스캔을 고려합니다.
  - 술어가 인덱스의 열을 참조합니다. 이 열이 선행 열일 필요는 없습니다.
  - 술어가 지정되지 않았지만 다음 조건이 모두 충족되는 경우
    - 테이블과 쿼리의 모든 열이 인덱스에 있습니다.
    - 인덱싱된 열 중 적어도 하나가 null이 아닙니다.
  - 쿼리에 인덱스된 null이 허용되지 않는 열에 대한 ORDER BY가 포함되어 있습니다.

Index Full Scan 작동방식

- 데이터베이스는 루트 블록을 읽은 다음 인덱스의 왼쪽을 따라 아래로(내림차순 전체 스캔을 수행하는 경우 오른쪽으로) 탐색하여 리프 블록에 도달할 때까지 이동합니다.
- 데이터베이스가 리프 블록에 도달하면, 인덱스 하단을 따라 한 번에 한 블록씩 정렬된 순서대로 스캔합니다.
- 이 그림은 부서 ID별로 정렬된 부서 레코드를 요청하는 명령문의 Index Full Scan 방식을 보여줍니다.



Index Fast Full Scans

- Index Fast Full Scan은 인덱스 자체를 테이블처럼 사용하여 디스크에 있는 인덱스 블록을 정렬되지 않은 순서대로 읽습니다.
  - 이 스캔은 인덱스를 사용하여 테이블을 탐색하는 것이 아니라, 테이블 대신 인덱스를 읽습니다.
- 옵티마이저는 쿼리가 인덱스의 속성에만 액세스할 때 이 방식을 고려합니다.
- INDEX_FFS(테이블 이름 인덱스 이름) 힌트는 고속 전체 인덱스 스캔을 강제로 실행합니다.
- Index Fast Full Scans 작동 방식
  - 데이터베이스는 다중 블록 I/O를 사용하여 루트 블록과 모든 리프 및 브랜치 블록을 읽습니다.
  - 데이터베이스는 브랜치 및 루트 블록을 무시하고 리프 블록의 인덱스 항목을 읽습니다.

NDEX FULL SCAN vs INDEX FFS(INDEX FAST FULL SCAN)

항목	INDEX FULL SCAN	INDEX FAST FULL SCAN (INDEX FFS)
목적	인덱스 키의 순서에 따라 전체 인덱스를 읽음	인덱스를 빠르게 전체 읽기, 정렬 무시
사용 조건	정렬이 필요하거나, 선두 키가 존재하는 범위 조건	SELECT에 ORDER BY 없음 + 인덱스로 커버 가능할 때
ROWID 접근	개별적으로 ROWID 따라 테이블 접근	테이블 접근 안 하거나 최소화 (커버링 인덱스인 경우)
정렬	정렬된 상태로 인덱스 읽기	정렬 무시, 병렬 가능
사용 예	ORDER BY가 있거나, MIN, MAX	SELECT count(*), GROUP BY, WHERE dept_id = 10 같은 단순 집계
효율성	느림 (순차 접근, I/O 많음)	빠름 (멀티블록 읽기, 병렬 가능)

Index Skip Scans

- 인덱스 스킵 스캔은 복합 인덱스의 초기 열이 쿼리에서 "건너뛰어지거나" 지정되지 않을 때 발생합니다.
- 종종 Index Skip Scan이 테이블 블록 스캔이나 전체 인덱스 스캔을 수행하는 것보다 빠를 수 있습니다.
- 옵티마이저가 인덱스 스킵 스캔을 고려하는 경우
  - 복합 인덱스의 선행 열이 쿼리 조건자에 지정되지 않은 경우
    - 예를 들어, 쿼리 조건자가 cust_gender 열을 참조하지 않고 복합 인덱스 키가 (cust_gender,cust_email)인 경우입니다.
  - 인덱스의 비선행 키에는 여러 개의 고유 값이 존재하고 선행 키에는 비교적 적은 고유 값이 존재하는 경우
    - 예를 들어, 복합 인덱스 키가 (cust_gender,cust_email)인 경우, cust_gender 열에는 두 개의 고유 값만 있지만 cust_email에는 수천 개의 고유 값이 있습니다.

Index Skip Scans 작동 방식

- 인덱스 스킵 스캔은 복합 인덱스를 논리적으로 더 작은 하위 인덱스로 분할합니다.
- 인덱스의 선행 열에 있는 고유 값의 개수에 따라 논리적 하위 인덱스의 개수가 결정됩니다.
- 개수가 적을수록 옵티마이저가 생성해야 하는 논리적 하위 인덱스의 개수가 줄어들고 스캔 효율성이 높아집니다.
- 스캔은 각 논리적 인덱스를 개별적으로 읽고, 선행 열이 아닌 열의 필터 조건을 충족하지 않는 인덱스 블록을 "건너뜁니다".

(3) 인덱스 튜닝 방법론

테이블 액세스 최소화

- 테이블 랜덤 액세스
  - 인덱스를 사용하지 않으면 테이블의 블록을 무작위로 읽게 되어 디스크 I/O가 증가합니다. 특히 대량 데이터에서는 치명적인 성능 저하로 이어질 수 있습니다.
- 인덱스 클러스터링 팩터
  - 인덱스 키 순서와 실제 데이터 저장 순서의 일치 정도를 나타내며, 값이 낮을수록 인덱스 효율이 높습니다.
  - 비효율적인 클러스터링 팩터는 인덱스를 타고 테이블을 읽는 비용을 증가시킵니다.
- 인덱스 손익 분기점
  - 테이블에서 반환되는 데이터의 비율이 일정 수준을 넘으면 풀 테이블 스캔이 더 유리합니다.
  - 이 분기점을 판단해 인덱스 사용 여부를 결정합니다.
- 인덱스 컬럼 추가
  - 자주 조회되는 컬럼을 인덱스에 포함시키면 인덱스만으로도 데이터를 조회할 수 있어 테이블 액세스를 줄일 수 있습니다.
  - 이를 커버링 인덱스라고 부릅니다.

테이블 액세스 최소화

- 인덱스만 읽고 처리
  - 쿼리가 SELECT 절과 WHERE 절에 모두 인덱스에 포함된 컬럼만 사용하는 경우, 테이블을 조회하지 않고 인덱스만으로 결과를 반환할 수 있습니다.
- 인덱스 구조의 테이블
  - 인덱스 조직 테이블(IOT)은 기본 테이블이 인덱스 구조로 저장되어 있어, PK 기반 액세스에서 매우 빠른 성능을 제공합니다.
  - 다만 범용적이지 않으며 설계 시 신중해야 합니다.
- 클러스터 테이블
  - 조인이 자주 발생하는 두 개 이상의 테이블을 물리적으로 인접하게 배치하여 I/O를 줄이고 연결 성능을 향상시키는 방식입니다.
  - 정적인 마스터-디테일 구조에 적합합니다.

부분 범위처리 활용

- 부분범위처리
  - 전체 결과를 한 번에 처리하지 않고 일정 단위로 나누어 처리함으로써 응답 지연을 줄이고 자원을 효율적으로 사용할 수 있습니다.
- 부분범위처리의 구현
  - ROWNUM, OFFSET-FETCH, 윈도우 함수 등을 이용해 일부 범위만 추출하여 처리합니다.
  - 페이징 쿼리나 최신 N건 조회 등에 사용됩니다.
- OLTP 환경에서 부분범위 처리에 의한 성능 개선원리
  - OLTP 시스템에서는 빠른 응답이 중요하므로, 전체 데이터가 아닌 필요한 만큼만 우선 처리하면 사용자 체감 성능이 개선됩니다.
  - 또한 락 경쟁을 줄일 수 있습니다.

인덱스 스캔 효율화

- 인덱스 탐색
  - 인덱스는 B-tree 구조로 되어 있어 루트부터 리프 노드까지 탐색을 수행하며, 이 과정은 정렬된 방식으로 매우 빠르게 이뤄집니다.
- 인덱스 스캔 효율성
  - 효율적인 스캔을 위해선 조건절이 인덱스의 선두 컬럼을 포함해야 하며, 범위 조건보다는 동등 조건이 유리합니다.
- 액세스 조건과 필터 조건
  - 액세스 조건은 인덱스 탐색에 직접 사용되는 조건이고, 필터 조건은 탐색 이후 걸러지는 조건입니다.
  - 인덱스가 액세스 조건을 많이 포함할수록 효율이 높습니다.

인덱스 스캔 효율화

- 비교연산자 종류와 컬럼 순서에 따른 군집성
  - 인덱스는 선두 컬럼부터 차례로 활용되기 때문에 컬럼 순서가 중요합니다.
  - 또한 =, BETWEEN, LIKE 등의 연산자에 따라 인덱스 활용도가 달라집니다.
- 범위검색 조건 남용 시의 비효율성
  - LIKE나 BETWEEN과 같은 범위 조건은 리프 노드의 많은 범위를 탐색하게 하여 불필요한 I/O를 초래합니다.
  - 특히 와일드카드 앞단 위치에 따라 인덱스 미사용 가능성이 있습니다.
- 함수 호출 부하 해소를 위한 인덱스 구성
  - WHERE절에서 자주 사용하는 함수나 표현식이 있다면, 함수 기반 인덱스를 사용하여 인덱스가 적용되도록 할 수 있습니다.
  - 단, 모든 DBMS에서 지원하는 것은 아닙니다.

인덱스 설계

- 인덱스 설계의 어려움
  - 다양한 쿼리 패턴, DML 빈도, 데이터 분포 등을 모두 고려해야 하므로 단순한 규칙만으로는 설계가 어렵습니다.
- 스캔 효율성 이외의 판단 기준
  - INSERT/UPDATE/DELETE 성능, 디스크 사용량, 인덱스 유지 비용 등도 함께 고려해야 전체적인 시스템 효율을 높일 수 있습니다.
- 공식을 초월한 전략적 설계 방안
  - 실행계획 분석과 경험을 바탕으로, 특정 업무에 맞는 비정형적 설계가 오히려 성능을 높이는 경우도 존재합니다.
- 중복 인덱스 제거
  - 동일한 컬럼 구성의 인덱스 또는 거의 유사한 인덱스가 여러 개 존재하면, 오히려 옵티마이저가 혼란을 겪고 리소스 낭비가 발생할 수 있으므로 주기적으로 정리해야 합니다.


Thank you