Oracle 실행 계획 분석 개요

Part 1. 학습 범위 설정

왜 이 질문이 중요한가

실행 계획은 DBA의 영역으로 인식되기 쉽다. Oracle 공식 문서만 해도 수백 페이지에 달하고, Wait Event, AWR Report, ASH 분석 등 깊이 들어가면 끝이 없다. 백엔드 엔지니어가 이 모든 것을 학습하는 것은 비효율적이다. 문제는 "어디까지 알아야 하는가"의 경계가 모호하다는 점이다.

이 경계를 설정하려면 먼저 "왜 실행 계획을 봐야 하는가"라는 목적을 명확히 해야 한다. 목적이 정해지면 그 목적을 달성하는 데 필요한 최소한의 지식 범위가 자연스럽게 도출된다.

백엔드 엔지니어가 실행 계획을 봐야 하는 상황

JPA/Hibernate를 사용하는 백엔드 엔지니어는 SQL을 직접 작성하지 않는다. `reviewStatisticsRepository.findAllByPropertyIdIn(ids)`를 호출하면 Hibernate가 SQL을 생성하고, JDBC 드라이버가 Oracle에 전송하고, Oracle이 실행한다. 이 과정에서 개발자가 통제할 수 있는 것은 Repository 메서드 설계와 엔티티 매핑뿐이다.

문제는 이 추상화된 구조에서 성능 문제가 발생했을 때 나타난다. API 응답 시간이 5초가 걸린다. 로그를 보니 DB 조회에서 대부분의 시간이 소요된다. 여기서 백엔드 엔지니어가 할 수 있는 질문은 두 가지다.

첫 번째 질문은 "Hibernate가 어떤 SQL을 생성했는가"다. 이것은 `show-sql=true` 설정으로 확인할 수 있다. N+1 쿼리가 발생하는지, IN 절이 어떻게 구성되는지 등을 파악한다.

두 번째 질문은 "Oracle이 그 SQL을 어떻게 처리했는가"다. 같은 SQL이라도 Oracle이 인덱스를 사용했는지, Full Scan을 했는지에 따라 성능이 천차만별이다. 이 두 번째 질문에 답하기 위해 실행 계획을 본다.

백엔드 엔지니어의 역할 경계

DBA는 Oracle 인스턴스 전체의 성능을 책임진다. Buffer Cache 크기 조정, Redo Log 설정, Wait Event 분석을 통한 시스템 레벨 튜닝이 DBA의 영역이다.

백엔드 엔지니어의 책임 범위는 자신이 작성한 코드가 유발하는 쿼리의 효율성이다. 시스템 전체가 아니라, 특정 API 또는 특정 서비스 메서드가 실행하는 쿼리가 효율적으로 동작하는지 확인하고, 비효율적이라면 코드를 수정하여 개선하는 것이 백엔드 엔지니어의 역할이다.

이 역할 범위에서 실행 계획은 진단 도구다. 의사가 환자를 치료하기 위해 X-ray를 읽듯이, 백엔드 엔지니어는 쿼리 성능 문제를 해결하기 위해 실행 계획을 읽는다. 의사가 방사선학 전문의 수준으로 X-ray를 분석할 필요는 없듯이, 백엔드 엔지니어도 DBA 수준으로 실행 계획을 분석할 필요는 없다. "이 쿼리가 왜 느린지 파악하고, 코드를 어떻게 수정해야 하는지 판단할 수 있는 수준"이 적정 범위다.

포트폴리오 목적에서의 실행 계획

피드백 문서에서 지적한 핵심은 "왜 그것이 병목이었는지를 명확하게 제시할 수 있어야 한다"는 것이다. 현재 포트폴리오 문서에서 "N+1 쿼리가 25번 실행되어서 느렸다"라고 주장하고 있지만, 이것만으로는 두 가지가 불명확하다.

첫째, 25번의 쿼리 각각이 Oracle 내부에서 어떻게 처리되었는가가 불명확하다. 각 쿼리가 인덱스를 타고 0.001초에 끝났다면, 25번 실행해도 총 0.025초다. 각 쿼리가 Full Scan으로 0.1초씩 걸렸다면, 총 2.5초다. 실행 계획 없이는 이 구분이 불가능하다.

둘째, Bulk Fetch에서 IN 절 OR 분해가 왜 비효율적인가가 불명확하다. "Oracle IN 절 1000개 제한으로 OR 분해가 발생했다"고 주장하지만, OR 분해가 실제로 어떤 실행 계획을 유발했고 그것이 왜 문제인지에 대한 근거가 없다.

실행 계획을 읽을 수 있으면 이 두 질문에 답할 수 있다. "N+1 쿼리 각각이 INDEX UNIQUE SCAN으로 처리되었지만, 25번의 네트워크 왕복과 Connection 점유가 누적되어 총 응답 시간이 증가했다"거나 "Bulk Fetch에서 CONCATENATION 실행 계획이 발생하여 Cost가 800으로 증가했다"와 같은 구체적 근거를 제시할 수 있다.

학습 범위 요약

알아야 할 것: 내가 작성한 코드가 유발하는 쿼리가 Oracle에서 효율적으로 처리되는지 판단할 수 있는 수준

알 필요 없는 것: Oracle 인스턴스 튜닝, 시스템 레벨 Wait Event 분석, AWR Report 심층 분석, 실행 계획을 강제하는 힌트 최적화, 통계 정보 직접 조회 및 분석

Part 2. 핵심 학습 내용 (백엔드 엔지니어 필수)

이 섹션은 백엔드 엔지니어가 반드시 알아야 할 내용이다. 포트폴리오에서 "왜 병목이었는지"를 설명하려면 이 수준의 지식이 필요하다.

2.1 실행 계획 조회 방법

실행 계획을 조회하는 방법을 알아야 한다. SQL Developer의 "계획 설명" 탭을 사용하는 것으로 충분하다. 포트폴리오에 첨부할 때는 쿼리와 함께 실행 계획 캡처를 포함하면 된다.

더 상세한 정보가 필요할 때는 `GATHER_PLAN_STATISTICS` 힌트와 `DBMS_XPLAN.DISPLAY_CURSOR`를 사용한다. 이것은 Optimizer의 추정치(E-Rows)와 실제 결과(A-Rows)를 비교하여 "통계 정보가 정확한가"를 판단할 때 사용한다.

2.2 테이블 접근 방식 해석

Oracle이 테이블에서 데이터를 가져오는 방식이 효율적인지 판단할 수 있어야 한다.

TABLE ACCESS FULL이 나타나면 "전체 테이블을 읽고 있다"는 의미다. 테이블이 작으면 문제없지만, 대용량 테이블에서 소수의 행만 필요한데 Full Scan이 발생하면 비효율적이다. 이 경우 인덱스가 없거나, 있어도 Optimizer가 사용하지 않기로 결정한 것이다.

INDEX RANGE SCAN 또는 **INDEX UNIQUE SCAN**이 나타나면 "인덱스를 사용하여 필요한 데이터 위치를 직접 찾아가고 있다"는 의미다. 일반적으로 효율적이다.

TABLE ACCESS BY INDEX ROWID가 INDEX SCAN 다음에 나타나면 "인덱스에서 찾은 위치로 테이블에 접근하고 있다"는 의미다. 인덱스만으로 조회 컬럼을 충족하지 못할 때 발생한다.

백엔드 엔지니어로서 판단해야 할 것은 "현재 쿼리의 조건과 테이블 규모를 고려할 때, 이 접근 방식이 합리적인가"다. 52,020행 테이블에서 295행을 조회하는데 Full Scan이 발생하면, 인덱스가 없거나 통계가 오래된 것일 수 있다. 이것을 확인하고 DBA에게 인덱스 생성 또는 통계 갱신을 요청하거나, 개발 환경에서 직접 조치할 수 있다.

2.3 Wherehouse 프로젝트 특화 패턴

포트폴리오의 Wherehouse 프로젝트에서 다루는 성능 문제에 특화된 패턴 두 가지를 알아야 한다.

INLIST ITERATOR: IN 절을 효율적으로 처리할 때 나타난다. 이 Operation이 나타나면 Oracle이 IN 절의 각 값에 대해 인덱스를 사용하여 접근하고 있다는 의미다. 이것이 나타나면 IN 절 처리가 효율적이다.

CONCATENATION: IN 절이 OR로 분해되어 여러 쿼리 블록으로 나뉘었을 때 나타난다. Wherehouse 프로젝트에서 Bulk Fetch가 비효율적이었던 이유가 바로 이것이다. IN 절에 1000개 초과의 값이 포함되면 Hibernate가 OR로 분해하고, Oracle이 CONCATENATION 실행 계획을 수립한다. 이 패턴이 나타나면 IN 절을 1000개 이하로 분할하는 Chunk 방식이 해결책이 된다.

Part 3. 배경 지식 (실행 계획 이해를 위한 개념)

이 섹션은 실행 계획의 숫자들이 무엇을 의미하는지 이해하기 위한 배경 지식이다. 직접 조회하거나 분석할 필요는 없지만, 실행 계획을 해석할 때 이 개념들을 알면 도움이 된다.

3.1 Cost-Based Optimizer의 작동 원리

Oracle은 SQL 문장을 받으면 가능한 여러 실행 계획 후보를 생성한 뒤, 각 후보의 "비용(Cost)"을 계산하여 가장 낮은 비용의 계획을 선택한다. 이것이 Cost-Based Optimizer(CBO)의 핵심 동작이다.

여기서 "비용"이란 해당 실행 계획을 수행할 때 예상되는 I/O 횟수와 CPU 사용량의 가중 합산값이다. Optimizer는 이 비용을 계산하기 위해 반드시 다음 질문들에 답해야 한다:

- 이 테이블에 행이 몇 개 있는가?
- 이 조건을 만족하는 행이 몇 개일 것으로 예상되는가?
- 이 인덱스를 사용하면 몇 번의 블록 I/O가 발생하는가?
- 테이블을 전체 스캔하면 몇 개의 블록을 읽어야 하는가?

이 질문들에 대한 답이 바로 통계 정보다. 통계 정보가 없거나 부정확하면 Optimizer는 잘못된 가정에 기반하여 비효율적인 실행 계획을 선택한다.

3.2 선택도(Selectivity)와 Cardinality

선택도는 **전체 행 중에서 특정 조건을 만족하는 행의 비율**이다. 값의 범위는 0에서 1 사이(또는 0%~100%)다. Optimizer가 실행 계획의 비용을 계산하려면 "이 WHERE 조건을 적용하면 몇 개의 행이 반환될 것인가"를 알아야 한다. 이 예상 반환 행 수를 **Cardinality**라고 하고, 다음 공식으로 계산한다:

```
Cardinality = NUM_ROWS × Selectivity
```

예를 들어 NUM_ROWS가 10,000이고 특정 조건의 선택도가 1%라면:

```
Cardinality = 10,000 × 0.01 = 100행
```

Optimizer는 해당 조건을 만족하는 행이 100개일 것으로 추정한다.

3.3 Optimizer가 선택도를 계산하는 방법

선택도 계산은 조건의 유형에 따라 다르다. 가장 단순한 등호 조건(`WHERE column = value`)의 경우:

```
Selectivity = 1 / NUM_DISTINCT
```

NUM_DISTINCT는 해당 컬럼의 고유 값 개수다. 예를 들어 `STATUS` 컬럼에 'ACTIVE', 'INACTIVE', 'PENDING' 세 가지 값만 존재한다면:

```
Selectivity = 1 / 3 = 33.3%
```

`WHERE STATUS = 'ACTIVE'` 조건을 적용하면 전체 행의 약 33%가 반환될 것으로 추정한다. 이것이 "균등 분포 가정(Uniform Distribution Assumption)"이다. Optimizer는 특별한 통계가 없으면 각 값이 균등하게 분포한다고 가정한다.

3.4 Wherehouse 프로젝트에서의 선택도 계산 예시

REVIEW_STATISTICS 테이블에서 `WHERE PROPERTY_ID IN (...)` 조건을 생각해보자. PROPERTY_ID가 Primary Key라면 DISTINCT_KEYS = NUM_ROWS다. 각 PROPERTY_ID는 고유하므로:

```
단일 값 선택도 = 1 / NUM_ROWS
```

NUM_ROWS가 52,020이면 단일 PROPERTY_ID의 선택도는 약 0.002%(1/52,020)다. IN 절에 300개의 ID가 있다면:

```
전체 선택도 = 300 / 52,020 ≈ 0.58%
예상 Cardinality = 52,020 × 0.58% ≈ 300행
```

3.5 선택도가 실행 계획 선택에 미치는 영향

선택도가 낮으면(반환 행이 적으면) 인덱스 사용이 유리하다. 선택도가 높으면(반환 행이 많으면) Full Scan이 유리할 수 있다. Optimizer의 판단 기준을 단순화하면:

- **낮은 선택도(예: 0.1%)**: 전체 테이블 중 극소수만 필요 → 인덱스로 정확한 위치를 찾아가는 것이 효율적

- **높은 선택도(예: 30%)**: 전체 테이블의 상당 부분이 필요 → 어차피 많이 읽어야 하므로 순차적 Full Scan이 랜덤 I/O보다 효율적

Optimizer가 선택도를 실제보다 높게 추정하면 불필요하게 Full Scan을 선택하고, 낮게 추정하면 비효율적인 인덱스 접근을 선택한다. 이것이 통계 정보가 정확해야 하는 이유다.

3.6 통계 부재가 실행 계획에 미치는 영향

통계가 수집되지 않은 테이블에 대해 Oracle은 "Dynamic Sampling"이라는 기법으로 실행 시점에 샘플 데이터를 읽어 통계를 추정한다. 그러나 이 추정은 정확도가 낮고, 특히 데이터 분포가 불균등한 경우 심각한 오판을 유발한다. 통계가 부정확하면 Optimizer는 선택도를 잘못 계산하고, 그 결과 비효율적인 실행 계획을 선택한다.

Part 4. 심화 참고 자료 (DBA 영역)

이 섹션은 백엔드 엔지니어가 직접 수행할 필요가 없는 DBA 영역의 내용이다. "Optimizer가 왜 이런 결정을 내렸는가"를 깊이 분석해야 할 때 참고할 수 있지만, 포트폴리오 목적에서는 이 수준까지 다룰 필요가 없다. 실행 계획에서 문제가 발견되면 DBA에게 "이 쿼리 Full Scan 타는데 확인 부탁드립니다"라고 요청하는 것이 적절한 역할 분담이다.

4.1 테이블 통계 정보

테이블 통계란 무엇인가

테이블 통계는 Oracle이 특정 테이블의 물리적/논리적 특성을 수치화한 메타데이터다. Optimizer는 이 정보를 읽어서 "이 테이블을 Full Scan하면 비용이 얼마인가", "이 조건의 선택도(Selectivity)가 얼마인가"를 계산한다.

조회 쿼리의 각 컬럼 의미

```
SELECT table_name
     , num_rows
     , blocks
     , avg_row_len
     , last_analyzed
FROM user_tables
WHERE table_name = 'REVIEW_STATISTICS';
```

NUM_ROWS는 테이블의 총 행 수 추정치다. Optimizer가 Cardinality를 계산할 때 기준이 되는 값이다.

BLOCKS는 테이블 데이터가 저장된 데이터 블록의 수다. Oracle의 I/O 단위는 블록(기본 8KB)이므로, Full Table Scan의 비용은 이 BLOCKS 값에 직접 비례한다. BLOCKS가 10이면 Full Scan 시 10번의 블록 I/O가 발생하고, BLOCKS가 10,000이면 10,000번의 I/O가 발생한다.

AVG_ROW_LEN은 한 행의 평균 바이트 크기다. 이 값과 블록 크기를 조합하면 한 블록에 몇 개의 행이 저장되는지 추정할 수 있다.

LAST_ANALYZED는 통계가 마지막으로 수집된 시점이다. 이 값이 NULL이면 통계가 한 번도 수집되지 않은 것이고, 오래된 날짜라면 현재 데이터 분포와 통계가 일치하지 않을 수 있다.

4.2 인덱스 통계 정보

인덱스 통계란 무엇인가

인덱스 통계는 특정 인덱스의 구조적 특성과 데이터 분포를 수치화한 메타데이터다. Optimizer는 이 정보를 읽어서 "이 인덱스를 사용하면 비용이 얼마인가"를 계산한다.

조회 쿼리의 각 컬럼 의미

```
SELECT index_name
     , num_rows
     , distinct_keys
     , clustering_factor
     , last_analyzed
FROM user_indexes
WHERE table_name = 'REVIEW_STATISTICS';
```

NUM_ROWS는 인덱스에 포함된 엔트리 수다. 일반적으로 테이블의 NUM_ROWS와 동일하지만, NULL 값이 있는 컬럼의 인덱스는 NULL 행이 제외되므로 차이가 발생할 수 있다.

DISTINCT_KEYS는 인덱스 키 값의 고유한 개수다. Primary Key 인덱스의 경우 NUM_ROWS와 동일해야 한다. 이 값은 선택도 계산에 사용된다. 선택도 공식 `Selectivity = 1 / NUM_DISTINCT`에서 NUM_DISTINCT가 바로 이 DISTINCT_KEYS에 해당한다.

CLUSTERING_FACTOR는 인덱스 사용 시 테이블 블록 I/O 패턴을 예측하는 핵심 지표다. 이 값이 인덱스 통계에서 가장 중요하다.

CLUSTERING_FACTOR의 의미

CLUSTERING_FACTOR의 의미를 정확히 이해하려면 인덱스와 테이블의 물리적 관계를 알아야 한다. 인덱스는 키 값 순서로 정렬되어 있지만, 테이블의 행들은 삽입 순서대로 저장된다. 인덱스를 통해 테이블에 접근할 때, 인덱스의 인접한 엔트리들이 테이블의 서로 다른 블록을 가리키면 블록 I/O가 많이 발생한다.

CLUSTERING_FACTOR가 테이블의 블록 수(BLOCKS)에 가까우면 인덱스 순서와 테이블 저장 순서가 유사하여 인덱스 Range Scan이 효율적이다. 반대로 CLUSTERING_FACTOR가 테이블의 행 수(NUM_ROWS)에 가까우면 인덱스의 각 엔트리가 서로 다른 블록을 가리켜서, 인덱스를 사용할수록 랜덤 I/O가 증가한다.

Optimizer는 CLUSTERING_FACTOR를 기반으로 "인덱스 Range Scan 후 테이블 접근" 비용을 계산한다. CLUSTERING_FACTOR가 높으면 이 비용이 높게 산정되어 Full Table Scan이 선택될 수 있다.

인덱스 통계와 실행 계획 선택의 관계

IN 절에 300개의 값이 있을 때, Optimizer가 인덱스 사용 비용을 계산하는 공식:

```
인덱스 사용 비용 ≈ (조회할 행 수) × (CLUSTERING_FACTOR / NUM_ROWS) + 인덱스 블록 I/O
```

만약 CLUSTERING_FACTOR가 높고, 조회할 행 수가 많다면, 인덱스 사용 비용이 Full Scan 비용보다 높게 계산되어 Full Scan이 선택된다. 통계 정보를 확인하면 이 계산 과정을 추적할 수 있다.

4.3 Optimizer Hint

힌트란 무엇인가

힌트는 개발자가 Optimizer의 실행 계획 결정에 직접 개입하는 지시문이다. Optimizer가 통계 정보를 기반으로 자동 선택한 계획이 실제로는 비효율적일 때, 힌트를 통해 특정 실행 방식을 강제할 수 있다.

힌트는 SQL 문장의 SELECT, INSERT, UPDATE, DELETE 키워드 직후에 `/*+ 힌트명 */` 형식으로 작성한다.

힌트의 설계적 의의

힌트는 Optimizer의 판단을 재정의하는 도구다. 이것이 필요한 이유는 Optimizer가 완벽하지 않기 때문이다. 통계가 부정확하거나, 데이터 분포가 특수하거나, 쿼리 패턴이 Optimizer의 모델과 맞지 않으면 잘못된 계획이 선택된다. 이때 힌트로 올바른 계획을 강제할 수 있다.

그러나 힌트 사용에는 주의가 필요하다. 힌트를 사용하면 데이터 분포가 변해도 실행 계획이 고정된다. 현재는 인덱스 사용이 효율적이더라도, 데이터가 증가하면 Full Scan이 더 나을 수 있다. 힌트가 박힌 코드는 이런 변화에 적응하지 못한다.

따라서 힌트는 "왜 Optimizer가 이 계획을 선택했는가"를 분석하는 진단 도구로 먼저 사용하고, 근본 원인(통계 부재, 인덱스 설계 오류 등)을 해결한 후에는 제거하는 것이 원칙이다.

INDEX 힌트의 동작

```
SELECT /*+ INDEX(r PK_REVIEW_STATISTICS) */
       PROPERTY_ID, AVG_RATING, ...
FROM REVIEW_STATISTICS r
WHERE PROPERTY_ID IN (...);
```

`/*+ INDEX(테이블별칭 인덱스명) */` 힌트는 Optimizer에게 "비용 계산 결과와 무관하게 이 인덱스를 사용하라"고 지시한다. 이 힌트를 적용한 실행 계획을 힌트 없는 계획과 비교하면:

1. **비용 확인**: 인덱스 사용 시 실제 Cost가 얼마인지 확인할 수 있다. 이 Cost 값은 Optimizer가 산정한 예상 비용이며, Full Scan의 Cost와 직접 비교 가능하다.
2. **의사결정 추적**: Optimizer가 왜 인덱스를 선택하지 않았는지 비용 차이로 파악할 수 있다. 만약 INDEX 힌트 적용 시 Cost가 Full Scan보다 높다면, Optimizer의 판단이 비용 모델 관점에서는 합리적이었음을 의미한다.
3. **실측 검증**: 인덱스 사용이 실제로 더 빠른지 실행 시간으로 검증할 수 있다. Cost는 추정치이므로 실제 실행 시간과 다를 수 있다. 실측 결과가 Cost 예측과 다르다면 통계 정보의 부정확성을 의심해야 한다.

4.4 학습하지 않아도 되는 것

DBA 영역에 해당하는 다음 내용은 백엔드 엔지니어 포트폴리오 목적에서 학습 우선순위가 낮다.

Wait Event 심층 분석: "db file sequential read 대기 시간이 증가했다"는 분석은 DBA 영역이다. 백엔드 엔지니어는 "이 쿼리가 Full Scan을 하고 있다" 수준까지만 파악하면 된다. 그 Full Scan이 디스크 I/O를 유발하는지, Buffer Cache에서 처리되는지는 DBA가 판단할 문제다.

힌트를 사용한 실행 계획 강제: INDEX 힌트, FULL 힌트 등을 사용하여 Optimizer의 결정을 재정의하는 것은 위험하다. 데이터 분포가 변하면 힌트가 오히려 성능을 악화시킬 수 있다. 백엔드 엔지니어는 힌트를 "진단 목적"으로만 사용하고, 프로덕션 코드에 힌트를 삽입하는 것은 DBA와 협의 후 결정해야 한다.

AWR Report, ASH 분석: 시스템 전체의 성능을 분석하는 도구다. 특정 쿼리의 문제를 파악하는 데는 실행 계획만으로 충분하다.

Part 5. 학습 자료

핵심 문서 (우선 학습)

Reading Execution Plans https://docs.oracle.com/database/121/TGSQL/tgsql_interp.htm

이 문서가 가장 적합하다. 실행 계획의 구조, 각 Operation의 의미, 실행 순서 해석 방법을 설명한다. 백엔드 엔지니어가 알아야 할 범위가 대부분 여기에 있다.

보조 문서

Generating and Displaying Execution Plans https://docs.oracle.com/en/database/oracle/oracle-database/19/tgsql/generating-and-displaying-execution-plans.html

실행 계획을 조회하는 방법(EXPLAIN PLAN, DBMS_XPLAN 등)을 설명한다. "어떻게 실행 계획을 뽑는가"에 대한 문서다.

학습 진행 방식 제안

위 "Reading Execution Plans" 문서를 읽으면서 이해가 안 되거나 Wherehouse 프로젝트에 어떻게 적용해야 할지 모르겠는 부분이 있으면 질문하라. 문서의 내용을 기반으로 맥락 있는 설명을 제공하겠다.

문서가 영문이라 읽기 부담스러우면, 특정 섹션을 지정하면 해당 섹션의 핵심 내용을 Wherehouse 프로젝트 맥락에서 설명할 수 있다.

Part 6. 결론

백엔드 엔지니어로서 Wherehouse 프로젝트 포트폴리오 목적에 맞는 실행 계획 학습 범위는 다음과 같다.

> "내가 작성한 코드가 유발하는 쿼리의 실행 계획을 조회하고, TABLE ACCESS FULL / INDEX SCAN / INLIST ITERATOR / CONCATENATION 패턴을 식별하여, 현재 쿼리가 효율적으로 처리되고 있는지 판단할 수 있는 수준"

이 수준을 달성하면 피드백에서 요구한 "왜 그것이 병목이었는지", "어떤 데이터로 확인했는지"에 대해 실행 계획 캡처와 함께 구체적 근거를 제시할 수 있다.

Part 7. Oracle 공식 문서 학습 범위 분석

Oracle 공식 문서 "Reading Execution Plans" (https://docs.oracle.com/database/121/TGSQL/tgsql_interp.htm)의 각 섹션을 Wherehouse 프로젝트 맥락에서 분석하여 학습 필요 여부를 판단한 내용이다.

7.1 필수 학습 섹션

섹션	학습 목적
7.1 Reading Execution Plans: Basic	TABLE ACCESS FULL vs INDEX SCAN 구분, 기본 실행 계획 구조 이해
7.2.5.5 INLIST ITERATOR	IN 절이 Oracle에서 효율적으로 처리될 때 나타나는 Operation 패턴
Table 7-1 PLAN_TABLE Columns	COST, CARDINALITY, ACCESS_PREDICATES 컬럼 해석
Table 7-3 OPERATION and OPTIONS Values	OPERATION/OPTIONS 조합별 의미 (CONCATENATION, INDEX RANGE SCAN 등)

Section 7.1 - Reading Execution Plans: Basic

기본적인 실행 계획 해석 방법을 다룬다. Example 7-1과 7-2가 핵심이다.

Example 7-1: TABLE ACCESS FULL이 발생하는 조건. WHERE 조건의 컬럼에 인덱스가 없거나, 인덱스가 있어도 `LIKE '%...'` 처럼 인덱스를 활용할 수 없는 조건일 때 발생한다.

Example 7-2: INDEX RANGE SCAN이 발생하는 조건. 인덱스 컬럼에 대한 범위 조건이 주어질 때 발생한다.

Wherehouse 프로젝트에서 REVIEW_STATISTICS 테이블의 IN 절 쿼리가 INDEX를 사용하는지 TABLE ACCESS FULL을 하는지 판단하는 기초가 된다.

Section 7.2.5.5 - Examples of INLIST ITERATOR and EXPLAIN PLAN

IN 절이 Oracle에서 처리되는 방식을 설명하는 핵심 섹션이다. 문서에서 발췌한 실행 계획 패턴:

```
OPERATION          OPTIONS           OBJECT_NAME
----------------   ---------------   --------------
SELECT STATEMENT
INLIST ITERATOR
TABLE ACCESS       BY ROWID          EMP
INDEX              RANGE SCAN        EMP_EMPNO
```

문서 원문: "The INLIST ITERATOR operation iterates over the next operation in the plan for each value in the IN-list predicate."

INLIST ITERATOR는 IN 절의 각 값에 대해 인덱스를 효율적으로 순회한다는 의미다. 이 Operation이 나타나면 IN 절 처리가 효율적임을 의미한다.

Wherehouse 프로젝트에서 Chunk 방식(1000개 단위 분할)으로 쿼리를 실행했을 때 이 패턴이 나타나는지 확인이 필요하다.

Table 7-3 - OPERATION and OPTIONS Values

핵심 Operation 선별:

OPERATION	OPTIONS	의미	Wherehouse 관련성
INLIST ITERATOR	-	IN 절의 각 값에 대해 다음 operation을 반복 실행	핵심 - Chunk 방식에서 나타나야 함
CONCATENATION	-	여러 row set의 union-all을 반환	핵심 - Bulk Fetch에서 OR 분해 시 나타남
INDEX	RANGE SCAN	하나 이상의 rowid를 인덱스에서 조회 (오름차순)	핵심 - PK 인덱스 사용 여부 확인
INDEX	UNIQUE SCAN	단일 rowid를 인덱스에서 조회	참고
TABLE ACCESS	FULL	테이블 전체 행 조회	주의 - 이것이 나타나면 비효율
TABLE ACCESS	BY INDEX ROWID	인덱스로 찾은 rowid로 테이블 접근	핵심 - 인덱스 사용 시 나타남

Table 7-1 - PLAN_TABLE Columns

Cost 해석에 필요한 컬럼:

Column	의미	포트폴리오 활용
COST	Optimizer가 추정한 실행 비용 (상대값)	Bulk vs Chunk의 Cost 비교
CARDINALITY (Rows)	예상 반환 행 수	실제 반환 행과 비교하여 통계 정확성 판단
BYTES	예상 데이터 크기	참고용
ACCESS_PREDICATES	인덱스 접근에 사용된 조건	IN 절이 어떻게 적용되는지 확인
FILTER_PREDICATES	행 필터링에 사용된 조건	추가 필터 조건 확인

7.2 학습 불필요 섹션

섹션	제외 근거
7.2.1 Adaptive Query Plans	단일 테이블 IN 절 조회로 조인이 없음. Adaptive Plan은 조인 방식 전환이 핵심이므로 해당 없음
7.2.2 Parallel Execution	테이블에 PARALLEL 속성 미설정, 쿼리에 PARALLEL 힌트 미사용. 직렬 실행만 수행
7.2.3 Bitmap Indexes	PK_REVIEW_STATISTICS는 B-tree Index. Bitmap Index 미존재
7.2.4 Result Cache	result_cache 힌트 미사용. 쿼리 구조별 DBMS 처리 효율 측정이 목적이므로 캐시 사용 시 측정 왜곡
7.2.5.1~7.2.5.4 Partitioned Objects	REVIEW_STATISTICS 테이블 파티셔닝 안 됨. 52,020행으로 파티셔닝 불필요 규모
7.2.6 Domain Indexes	Oracle Text, Spatial 등 특수 기능 미사용. 단순 동등 비교(IN 절)만 수행

Section 7.2.1 - Reading Adaptive Query Plans

해당 섹션이 다루는 기술: Adaptive Query Plan은 Oracle 12c부터 도입된 기능으로, Optimizer가 실행 시점에 수집한 통계를 기반으로 실행 계획을 동적으로 변경하는 메커니즘이다. 문서의 예시는 NESTED LOOPS JOIN으로 시작했다가 런타임에 HASH JOIN으로 전환되는 케이스를 다룬다.

적용 조건: Adaptive Plan이 작동하려면 복잡한 조인 쿼리여야 한다. 문서의 예시는 orders, order_items, product_information 세 테이블을 조인하는 쿼리다. Optimizer의 Cardinality 추정이 실제와 크게 다를 때 발생한다. 문서에서 "the difference between the estimated and actual number of rows is significant"라고 명시한다.

Wherehouse 프로젝트 현재 상태: REVIEW_STATISTICS 테이블에 대한 쿼리는 단일 테이블 조회다.

```
SELECT PROPERTY_ID, AVG_RATING, LAST_CALCED,
       NEGATIVE_KEYWORD_COUNT, POSITIVE_KEYWORD_COUNT, REVIEW_COUNT
FROM REVIEW_STATISTICS
WHERE PROPERTY_ID IN (...)
```

조인이 없다. Adaptive Plan은 조인 방식(Nested Loops vs Hash Join)을 런타임에 전환하는 것이 핵심인데, 조인 자체가 없으므로 이 메커니즘이 개입할 여지가 없다. 통계 정보가 최신 상태이며 NUM_ROWS가 52,020으로 정확하여 Cardinality 추정 오차가 발생할 가능성이 낮다.

Section 7.2.2 - Viewing Parallel Execution with EXPLAIN PLAN

해당 섹션이 다루는 기술: Parallel Query는 하나의 SQL 문장을 여러 Parallel Execution Server(PX Server)가 동시에 처리하는 기능이다. 문서의 예시에서 PX COORDINATOR, PX SEND, PX RECEIVE, PX BLOCK ITERATOR 등의 Operation이 등장한다.

적용 조건: Parallel Query가 작동하려면 테이블에 PARALLEL 속성이 설정되어야 한다.

```
ALTER TABLE emp2 PARALLEL 2;
```

또는 쿼리에 PARALLEL 힌트가 포함되어야 한다.

```
SELECT /*+ PARALLEL(t, 4) */ * FROM table t;
```

Oracle Enterprise Edition 라이선스가 필요하다. Standard Edition에서는 Parallel Query가 제한된다.

Wherehouse 프로젝트 현재 상태: 로컬 개발 환경에서 테이블 생성 시 PARALLEL 속성을 명시적으로 설정하지 않았다면 기본값은 1(직렬 실행)이다. 테스트 쿼리에 PARALLEL 힌트를 사용하지 않았다. 직렬 실행이므로 PX 관련 Operation이 실행 계획에 나타나지 않는다.

Section 7.2.3 - Viewing Bitmap Indexes with EXPLAIN PLAN

해당 섹션이 다루는 기술: Bitmap Index는 B-tree Index와 다른 구조로, 각 키 값에 대해 비트맵을 저장한다. 주로 Cardinality가 낮은 컬럼(예: 성별, 상태코드)에 사용된다. 문서의 예시에서 BITMAP CONVERSION, BITMAP OR, BITMAP MINUS, BITMAP MERGE 등의 Operation이 등장한다.

적용 조건: Bitmap Index가 실행 계획에 나타나려면 해당 테이블에 Bitmap Index가 생성되어 있어야 한다.

```
CREATE BITMAP INDEX idx_status ON table(status);
```

Wherehouse 프로젝트 현재 상태: PK_REVIEW_STATISTICS는 Primary Key 제약조건에 의해 자동 생성된 인덱스다. Oracle에서 PK 인덱스는 기본적으로 B-tree Unique Index다. Bitmap Index가 아니다. PROPERTY_ID는 32자리 해시값으로 52,020개의 고유 값을 가진다 (DISTINCT_KEYS = NUM_ROWS). High Cardinality 컬럼에는 Bitmap Index가 적합하지 않다.

Section 7.2.4 - Viewing Result Cache with EXPLAIN PLAN

해당 섹션이 다루는 기술: Result Cache는 쿼리 결과를 메모리에 캐싱하여 동일 쿼리 재실행 시 캐시된 결과를 반환하는 기능이다. 실행 계획에 RESULT CACHE Operation이 나타난다.

적용 조건: Result Cache가 작동하려면 쿼리에 `result_cache` 힌트가 포함되어야 한다.

```
SELECT /*+ result_cache */ deptno, avg(sal) FROM emp GROUP BY deptno;
```

또는 RESULT_CACHE_MODE 파라미터가 FORCE로 설정되어야 한다.

Wherehouse 프로젝트 현재 상태: 테스트 쿼리에 `result_cache` 힌트가 없다. Wherehouse 프로젝트의 성능 테스트 목적은 "동일 쿼리 반복 실행의 캐시 효과"가 아니라 "쿼리 구조(N+1 vs Bulk vs Chunk)에 따른 DBMS 처리 효율성 차이"를 측정하는 것이다. Result Cache를 사용하면 오히려 측정이 왜곡된다.

Section 7.2.5 - Viewing Partitioned Objects with EXPLAIN PLAN (7.2.5.5 제외)

해당 섹션이 다루는 기술: 테이블 파티셔닝은 대용량 테이블을 물리적으로 분할하여 저장하는 기법이다. Range Partition, Hash Partition, Composite Partition 등이 있다. 실행 계획에

PARTITION RANGE, PARTITION HASH, PARTITION_START, PARTITION_STOP 등이 나타난다. Partition Pruning(불필요한 파티션 제외)이 핵심 최적화 포인트다.

적용 조건: 파티션 관련 실행 계획이 나타나려면 테이블이 파티셔닝되어 있어야 한다.

```
CREATE TABLE emp_range PARTITION BY RANGE(hire_date) (
  PARTITION emp_p1 VALUES LESS THAN (TO_DATE('1-JAN-1992','DD-MON-YYYY')),
  ...
);
```

Wherehouse 프로젝트 현재 상태: JPA/Hibernate로 생성한 테이블은 기본적으로 파티셔닝되지 않는다. @Table 어노테이션에 파티션 설정이 없다. 52,020행은 파티셔닝이 필요한 규모가 아니다. 일반적으로 수백만 행 이상의 테이블에서 파티셔닝의 이점이 나타난다. 단, Section 7.2.5.5 (INLIST ITERATOR)는 예외로, 파티션과 무관하게 IN 절 처리 방식을 설명하므로 필수 학습 대상이다.

Section 7.2.6 - Domain Indexes and EXPLAIN PLAN

해당 섹션이 다루는 기술: Domain Index는 Oracle에서 사용자 정의 인덱스 타입을 생성할 수 있는 기능이다. 주로 Oracle Text(전문 검색), Oracle Spatial(공간 데이터) 등에서 사용된다. 실행 계획에 DOMAIN INDEX Operation이 나타난다.

문서의 예시:

```
SELECT * FROM emp WHERE CONTAINS(resume, 'Oracle') = 1
```

CONTAINS는 Oracle Text의 전문 검색 연산자이며, 이를 지원하기 위해 Domain Index가 필요하다.

적용 조건: Domain Index가 실행 계획에 나타나려면 Oracle Text, Oracle Spatial 등 특수 기능을 사용해야 하며, 해당 기능을 위한 Domain Index가 생성되어 있어야 한다.

```
CREATE INDEX emp_resume ON emp(resume) INDEXTYPE IS CTXSYS.CONTEXT;
```

Wherehouse 프로젝트 현재 상태: REVIEW_STATISTICS 테이블의 쿼리는 전문 검색(CONTAINS), 공간 쿼리(SDO_WITHIN_DISTANCE) 등 특수 연산자를 사용하지 않는다. 단순 동등 비교(IN 절)만 사용한다. PK_REVIEW_STATISTICS의 index_type은 NORMAL(B-tree)이다. Domain Index가 아니다.

7.3 즉시 실습 권장 사항

문서의 7.1 섹션 학습 후, 다음 쿼리를 SQL Developer에서 실행하여 실제 실행 계획을 확인한다:

```
-- 295개 IN 절 쿼리의 실행 계획 확인
EXPLAIN PLAN FOR
SELECT PROPERTY_ID, AVG_RATING, LAST_CALCED,
       NEGATIVE_KEYWORD_COUNT, POSITIVE_KEYWORD_COUNT, REVIEW_COUNT
FROM REVIEW_STATISTICS
WHERE PROPERTY_ID IN ('e03d65e70b49ab5bb17258be672024c2', ... /* 295개 */);

SELECT * FROM TABLE(DBMS_XPLAN.DISPLAY);
```

실행 계획에서 INLIST ITERATOR가 나타나는지, 아니면 다른 패턴이 나타나는지 확인하면 문서의 내용을 실제 프로젝트에 즉시 연결할 수 있다.