

## SQL OPTIMIZATION

구간	문제 번호	주제	난이도
1~5	128~132	옵티마이저와 실행계획 개요	☆☆ 초~중급
6~10	133~137	인덱스 구조와 활용 전략	☆☆☆ 중급
11~15	138~142	인덱스 성능과 조인 방식 비교	☆☆☆ 고급
16~19	143~146	조인 기법별 특징과 적용 조건	☆☆☆☆ 실무형

### 구간별 설명 (2줄 요약)

#### 1~5번: 옵티마이저와 실행계획 개요

CBO, RBO 등 옵티마이저의 개념과 SQL 실행 흐름을 이해합니다.  
실행계획의 구성 요소와 처리 절차를 시각적으로 분석하는 기초 구간입니다.

#### 6~10번: 인덱스 구조와 활용 전략

B-TREE, CLUSTERED, BITMAP 등 인덱스 유형을 비교합니다.  
인덱스의 장단점과 옵티마이저의 인덱스 선택 기준을 학습합니다.

#### 11~15번: 인덱스 성능과 조인 방식 비교

인덱스 컬럼 순서, 범위 조건, DML 부하 등 성능에 미치는 영향을 분석합니다.  
SQL 실행 조건에 따라 인덱스 효율성과 조인 방식이 어떻게 달라지는지 파악합니다.

#### 16~19번: 조인 기법별 특징과 적용 조건

Nested Loop, Hash Join, Sort Merge Join의 특징과 적용 조건을 비교합니다.  
선행 테이블 선택, 인덱스 유무, 데이터량에 따른 조인 전략을 실무 중심으로 익힙니다.

[문제 128]

아래에 해당하는 내용을 작성하시오.

테이블 및 인덱스 등의 통계 정보를 활용하여 SQL문을 실행하는데 소요될 처리시간 및 CPU, I/O 자원량 등을 계산하여 가장 효율적일 것으로 예상되는 실행계획을 선택하는 옵티마이저를 (        ) 라 한다.

[문제 128]

통계 정보를 활용해 실행계획을 선택하는 옵티마이저는?

✅ **정답:** CBO (Cost Based Optimizer) , 비용기반 옵티마이저

🐱 **쉬운 해설:**

CBO는 비용을 계산해서 가장 빠른 실행계획을 선택해요!

📖 **전문 해설:**

- CBO는 통계 정보 기반
- CPU, I/O, 처리 시간 등을 고려
- RBO는 규칙 기반으로 우선순위만 따짐

🧠 **기억법:**

CBO = Cost Based → 비용 계산

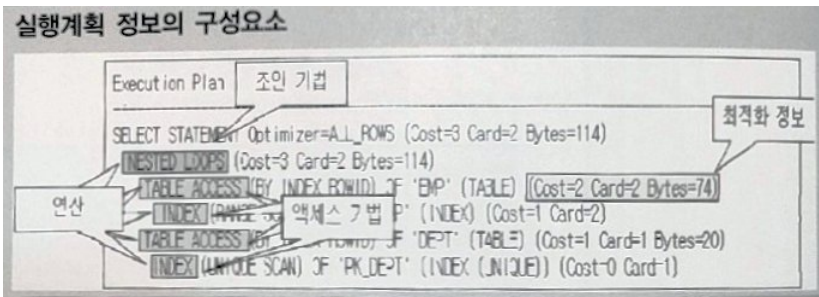
📌 **관련 암기카드**

- 📌 카드 70: 옵티마이저 종류
- 📌 카드 71: 실행계획 개요

[문제 129]

다음 중 실행계획을 통해서 알 수 있는 정보로 가장 부적절한 것은?

- ① 액세스 기법
- ② 질의 처리 예상 비용(Cost)
- ③ 조인 순서
- ④ 실제 처리 건수



✓ 정답: ④

🐱 쉬운 해설:

실행계획은 예측 정보만 보여줘요. 실제 처리 건수는 실행 후에만 알 수 있어요!

📖 전문 해설:

- 실행계획은 예상 비용, 조인 순서, 액세스 방식 등 포함
- 실제 건수는 실행 후 통계에서 확인 가능

🧠 기억법:

실행계획 = 예측 정보

실제 건수 ≠ 실행계획

📌 관련 암기카드

- 📌 카드 71: 실행계획 개요

[문제 130]

아래 실행계획의 실행순서에 맞게 ㉠, ㉡, ㉢ 을 작성하시오.

- 1 NESTED LOOPS
  - 2 HASH JOIN
  - 3 TABLE ACCESS (FULL) TAB1
  - 4 TABLE ACCESS (FULL) TAB2
  - 5 TABLE ACCESS (BY ROWID) TAB3
  - 6 INDEX (UNIQUE SCAN) PK TAB3
- ㉠ → ㉡ → ㉢ → 6 → 5 → 1

✓ 정답: ㉠ 3 → ㉡ 4 → ㉢ 2

🐱 쉬운 해설:

실행은 항상 가장 안쪽부터 시작해요. FULL SCAN → JOIN → LOOP 순서예요!

📖 전문 해설:

- 3, 4: 테이블 전체 스캔
- 2: HASH JOIN
- 6 → 5: 인덱스 → ROWID
- 1: NESTED LOOP

## ✅ 실행계획 구조

```

1  NESTED LOOPS
2    HASH JOIN
3      TABLE ACCESS (FULL) TAB1
4      TABLE ACCESS (FULL) TAB2
5      TABLE ACCESS (BY ROWID) TAB3
6      INDEX (UNIQUE SCAN) PK TAB3

```

## 🧠 실행 순서 한 줄씩 설명

### 1 ① 3: TABLE ACCESS (FULL) TAB1

- 가장 먼저 TAB1 테이블을 **전체 스캔**합니다
- HASH JOIN의 첫 번째 입력으로 사용됨

### 2 ② 4: TABLE ACCESS (FULL) TAB2

- 다음으로 TAB2 테이블을 **전체 스캔**합니다
- HASH JOIN의 두 번째 입력으로 사용됨

### 3 ③ 2: HASH JOIN

- TAB1과 TAB2의 결과를 해시 조인합니다
- 이 결과는 NESTED LOOP의 외부 테이블 역할을 합니다

### 4 ④ 6: INDEX (UNIQUE SCAN) PK TAB3

- TAB3에 대해 **인덱스를 먼저 스캔**합니다
- PK 인덱스를 통해 해당 ROWID를 찾습니다

### 5 ⑤ 5: TABLE ACCESS (BY ROWID) TAB3

- 인덱스로 찾은 ROWID를 통해 TAB3의 실제 데이터를 읽습니다

### 6 ⑥ 1: NESTED LOOPS

- HASH JOIN 결과와 TAB3 데이터를 NESTED LOOP 방식으로 결합합니다

✅ 최종 실행 순서

번호	설명
㉟ 3	TAB1 전체 스캔
㉞ 4	TAB2 전체 스캔
㉟ 2	HASH JOIN 수행
6	TAB3 인덱스 스캔
5	TAB3 ROWID 접근
1	NESTED LOOP 결합

📌 핵심 요약

구성 요소	역할
TABLE ACCESS (FULL)	테이블 전체 스캔
HASH JOIN	두 테이블 조인
INDEX SCAN	인덱스를 통해 ROWID 찾기
TABLE ACCESS (BY ROWID)	인덱스로 찾은 행 읽기
NESTED LOOPS	조인 결과 반복 처리

🧠 기억하세요: 실행계획은 항상 가장 안쪽부터 시작해서 바깥으로 진행됩니다.

🧠 기억법:

실행계획은 안쪽부터 바깥으로

📌 관련 암기카드

- 📌 카드 71: 실행계획 개요

[문제 131]

다음 중 실행계획에 대한 설명으로 가장 부적절한 것은?

- ① 실행계획은 SQL 처리를 위한 실행 절차와 방법을 표현한 것이다.
- ② 실행계획은 조인 방법, 조인 순서, 액세스 기법 등이 표현된다.
- ③ 동일 SQL문에 대해 실행계획이 다르면 실행 결과도 달라질 수 있다.
- ④ CBO(Cost Based Optimizer)의 실행계획에는 단계별 예상 비용 및 건수 등이 표시된다.

✅ 정답: ③

🐻 쉬운 해설:

실행계획이 달라도 결과는 같아요! 다만 성능이 달라질 뿐이에요.

📖 전문 해설:

- 실행계획은 SQL 실행 방법
- 결과는 동일하지만 성능은 달라짐
- CBO는 비용 기반으로 계획 수립

🧠 기억법:

실행계획 ≠ 결과 변경

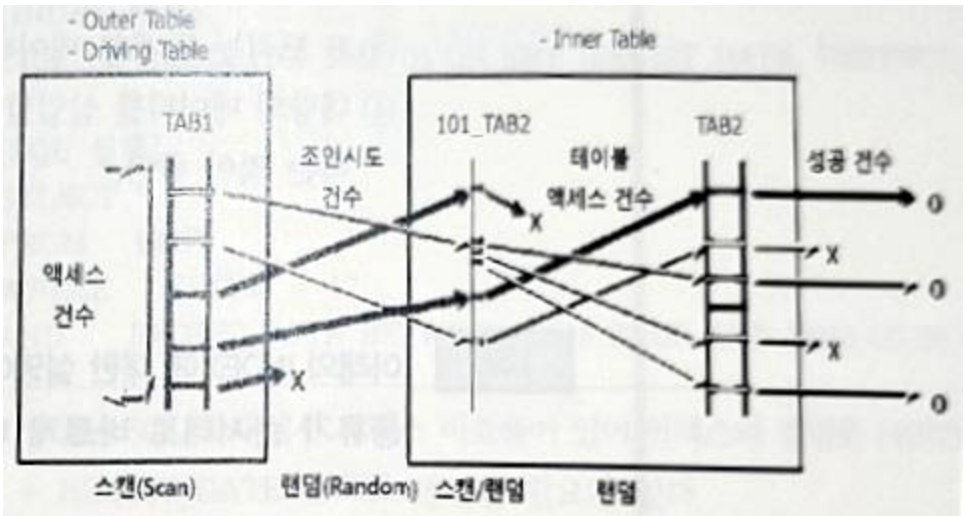
실행계획 = 성능 영향

📌 관련 암기카드

- 📌 카드 70: 옵티마이저
- 📌 카드 71: 실행계획

[문제 132]

다음 중 아래와 같은 SQL 처리 흐름도(Access Flow Diagram)에 대한 설명으로 가장 적절한 것을 2개 고르시오.



`` ① SQL의 실행 시간을 알 수 있다. ② 인덱스 스캔, 테이블 전체 스캔 등과 같은 액세스 기법이 표현된다. ③ 실행계획과는 무관하다. ④ SQL의 내부적인 처리 절차를 시각적으로 표현해준다. `` \*\* ☒ 정답: \*\* ②, ④

### 🐱 쉬운 해설:

흐름도는 SQL 내부 처리 절차를 시각적으로 보여줘요. 인덱스 스캔도 표현돼요!

### 📖 전문 해설:

- Access Flow Diagram은 SQL 처리 절차 시각화
- 인덱스 스캔, 테이블 스캔 등 표현
- 실행 시간은 포함되지 않음

보기 번호	설명	적절성
①	실행 시간은 알 수 없음	<input checked="" type="checkbox"/>
②	액세스 기법 표현 가능	<input checked="" type="checkbox"/>
③	실행계획과 관련 있음	<input checked="" type="checkbox"/>
④	내부 처리 절차 시각화	<input checked="" type="checkbox"/>

### 🧠 기억법:

Access Flow = 내부 처리 흐름

### 📌 관련 암기카드

- 📌 카드 71: 실행계획
- 📌 카드 72: SQL 처리 흐름

[문제 133]

다음 중 옵티마이저와 실행계획에 대한 설명으로 부적절한 것을 2개 고르시오.

- ① SQL 처리 흐름도는 성능적인 측면의 표현은 고려하지 않는다.
- ② 규칙기반 옵티마이저에서 제일 높은 우선순위는 행에 대한 고유 주소를 사용하는 방법이다.
- ③ SQL 처리 흐름도는 인덱스 스캔 및 전체 테이블 스캔 등의 액세스 기법을 표현할 수 있다.
- ④ 인덱스 범위 스캔은 항상 여러 건의 결과가 반환된다.

✅ 정답: ①, ④

🐱 쉬운 해설:

- ① SQL 흐름도도 성능 표현 가능해요!
- ④ 인덱스 범위 스캔은 1건만 나올 수도 있어요!

📖 전문 해설:

- ① SQL 흐름도는 액세스 기법과 성능 흐름을 표현함
- ④ 인덱스 범위 스캔은 조건에 따라 0건, 1건도 가능
- ② RBO는 ROWID 액세스를 가장 우선시함
- ③ 흐름도는 인덱스 스캔 등 표현 가능

🧠 기억법:

범위 스캔 = 결과 다양  
흐름도 = 성능 흐름 포함

📌 관련 암기카드

- 📌 카드 70: 옵티마이저
- 📌 카드 71: 실행계획
- 📌 카드 72: SQL 흐름도

[문제 134]

다음 중 관계형 데이터베이스의 인덱스(Index)에 대한 설명으로 가장 적절한 것은?

- ① 기본 인덱스(Primary Key Index)에 중복된 키 값들이 나타날 수 있다.
- ② 기본 인덱스에 널 값(Null Value)들이 나타날 수 없다.
- ③ 보조 인덱스(Secondary Index)에는 고유한 키 값들만 나타날 수 있다.
- ④ 자주 변경되는 속성은 인덱스를 정의할 좋은 후보이다.

✅ 정답: ②



### 🐱 쉬운 해설:

기본 인덱스(PK)는 NULL을 허용하지 않아요!

### 📖 전문 해설:

- PK 인덱스는 NULL 불가
- 중복 불가
- 보조 인덱스는 중복 가능
- 자주 변경되는 컬럼은 인덱스에 부적합

### 🧠 기억법:

PK = NULL 불가, 중복 불가

### 📁 관련 암기카드

- 📁 카드 73: 인덱스 구조
- 📁 카드 74: 인덱스 특징

[문제 135]

다음 중 관계형 데이터베이스 인덱스(Index)에 대한 설명으로 가장 부적절한 것을 2개 고르시오.

- ① 테이블의 전체 데이터를 읽는 경우는 인덱스가 거의 불필요하다.
- ② 인덱스는 조회, 삽입, 삭제, 갱신 연산의 속도를 향상시킨다.
- ③ B 트리는 관계형 데이터베이스의 주요 인덱스 구조이다.
- ④ 대량의 데이터를 삽입할 때는 모든 인덱스를 생성하고 데이터를 입력하는 것이 좋다.

✅ 정답: ②, ④

### 🐱 쉬운 해설:

- ② 인덱스는 DML 성능을 저하시킬 수도 있어요!  
④ 대량 삽입 시 인덱스는 나중에 생성하는 게 좋아요!

### 📖 전문 해설:



- ① 전체 데이터 읽을 땐 인덱스 불필요
- ② DML 성능은 저하됨
- ③ B-TREE는 대표 인덱스 구조
- ④ 대량 삽입 시 인덱스는 나중에 생성

### 🧠 기억법:

DML + 인덱스 = 성능 저하

대량 삽입 → 인덱스 나중에

## 관련 암기카드

-  카드 74: 인덱스 특징
-  카드 75: 인덱스 성능

[문제 136]

아래의 INDEX에 대한 설명에서 (가),(나),(다)에 들어갈 인덱스 종류가 순서대로 바르게 나열된 것은?

(가)인덱스는 브랜치 블록과 리프 블록으로 구성되며, 브랜치 블록은 분기를 목적으로 하고 리프블록은 인덱스를 구성하는 컬럼의 값으로 정렬된다. 일반적으로 OLTP 시스템 환경에서 가장 많이 사용된다.

(나) 인덱스는 인덱스의 리프 페이지가 곧 데이터 페이지이며, 리프 페이지의 모든 데이터는 인덱스 키 컬럼 순으로 물리적으로 정렬되어 저장된다.

(다) 인덱스는 시스템에서 사용될 질의를 시스템 구현 시에 모두 알 수 없는 경우인 DW 및 AD-HOC 질의 환경을 위해서 설계되었으며, 하나의 인덱스 키 엔트리가 많은 행에 대한 포인터를 저장하고 있는 구조이다.

- ① B-TREE 인덱스, BITMAP 인덱스, CLUSTERED 인덱스
- ② B-TREE 인덱스, CLUSTERED 인덱스, BITMAP 인덱스
- ③ BITMAP 인덱스, CLUSTERED 인덱스, REVERSE KEY 인덱스
- ④ BITMAP 인덱스, REVERSE KEY 인덱스, CLUSTERED 인덱스

 **정답:** ② B-TREE, CLUSTERED, BITMAP

## 쉬운 해설:

B-TREE는 OLTP, CLUSTERED는 물리적 정렬, BITMAP은 DW에 좋아요!

## 전문 해설:

- B-TREE: 브랜치 + 리프 구조, OLTP에 적합
- CLUSTERED: 리프 = 데이터 페이지, 물리적 정렬
- BITMAP: 다건 포인터, DW/AD-HOC 질의에 적합


## 기억법:

B-TREE = OLTP

CLUSTERED = 정렬

BITMAP = DW

## 관련 암기카드

-  카드 73: 인덱스 구조

- 📇 카드 74: 인덱스 특징

[문제 137]

다음 중 인덱스에 대한 설명으로 가장 적절한 것을 2개 고르시오.

- ① 인덱스는 인덱스 구성 칼럼으로 항상 오름차순으로 정렬된다.
- ② 비용기반 옵티마이저는 인덱스 스캔이 항상 유리하다고 판단한다.
- ③ 규칙기반 옵티마이저는 적절한 인덱스가 존재하면 항상 인덱스를 사용 하려고 한다.
- ④ 인덱스 범위 스캔은 결과가 없으면 한 건도 반환하지 않을 수 있다.

✅ 정답: ③, ④

🐼 쉬운 해설:

- ③ RBO는 인덱스 있으면 무조건 써요!
- ④ 범위 스캔은 결과가 없을 수도 있어요!

📖 전문 해설:

- ① 인덱스는 정렬 방향 다양
- ② CBO는 비용에 따라 판단
- ③ RBO는 인덱스 우선
- ④ 범위 조건에 따라 결과 없음 가능

🧠 기억법:

RBO = 인덱스 우선

범위 스캔 = 결과 다양

📇 관련 암기카드

- 📇 카드 70: 옵티마이저
- 📇 카드 74: 인덱스 특징

[문제 138]

다음 중 아래의 내용에 대한 설명으로 가장 적절한 것을 2개 고르시오.

### [INDEX 생성]

아래

```
CREATE INDEX IDX_EMP_01 ON EMP (REGIST_DATE, DEPTNO);
```

### [SQL 실행]

```
SELECT *
```

```
FROM EMP
```

```
WHERE DEPTNO = 47
```

```
AND REGIST_DATE BETWEEN '2015.02.01' AND '2015.02.28':
```

- ① 실행된 SQL에 대해서 인덱스 비효율이 있어 인덱스의 컬럼을 DEPTNO + REGIST\_DATE 순으로 변경할 필요가 있다.
- ② IDX\_EMP\_01 인덱스를 이용하여 DEPTNO = 47 조건을 효율적으로 탐색할 수 있다.
- ③ REGIST\_DATE 컬럼에 대한 조건을 범위 검색이 아닌 동등 검색 조건으로 변경하면 IDX\_EMP\_01 인덱스를 효율적으로 활용할 수 있다.
- ④ IDX\_EMP\_01 인덱스는 테이블 내의 대량 데이터를 탐색할 때 매우 유용하게 활용될 수 있는 인덱스 형식이다.

✅ 정답: ①, ③

### 🐱 쉬운 해설:

- ① 인덱스 컬럼 순서가 조건과 맞아야 해요!
- ③ 범위 조건보다 동등 조건이 인덱스에 더 좋아요!

### 📖 전문 해설:

- IDX\_EMP\_01: REGIST\_DATE, DEPTNO
- WHERE절에서 DEPTNO만 동등 조건이면 인덱스 비효율
- REGIST\_DATE를 동등 조건으로 바꾸면 효율 증가

### 🧠 기억법:

인덱스 순서 = 조건 순서와 맞춰야  
동등 조건 > 범위 조건

### 📌 관련 암기카드

- 📌 카드 75: 인덱스 성능
- 📌 카드 76: 인덱스 활용 전략

[문제 139]

다음 중 인덱스에 대한 설명으로 가장 부적절한 것은?

- ① 인덱스의 목적은 조회 성능을 최적화하는 것이다.
- ② Insert, Update, Delete 등의 DML 처리 성능을 저하시킬 수도 있다.
- ③ B-트리 인덱스는 일치 및 범위 검색에 적절한 구조이다.
- ④ 인덱스 액세스는 테이블 전체 스캔보다 항상 유리하다.

✅ 정답: ④

🐱 쉬운 해설:

인덱스가 항상 유리한 건 아니예요! 전체 스캔이 더 빠를 때도 있어요.

📖 전문 해설:

- 인덱스는 조회 성능 향상 목적
- DML 시 인덱스 유지 비용 발생
- B-트리는 일치/범위 검색에 적합
- 소량 데이터나 전건 조회 시 전체 스캔이 더 효율적일 수 있음

🧠 기억법:

인덱스 ≠ 항상 빠름

조건에 따라 전체 스캔이 유리할 수도 있음

📁 관련 암기카드

- 📇 카드 74: 인덱스 특징
- 📇 카드 75: 인덱스 성능

[문제 140]

다음 중 아래에서 인덱스에 대한 설명으로 가장 올바른 것만 묶은 것은?

- 가) 인덱스는 데이터 조회 목적에는 효과적이지만, INSERT, UPDATE, DELETE 작업에는 오히려 많은 부하를 줄 수도 있다.
- 나) 인덱스를 이용한 데이터 조회는 대부분의 경우 테이블 전체 스캔보다 빠르다.
- 다) SQL Server의 클러스터형 인덱스는 ORACLE의 IOT와 매우 유사하다.
- 라) 인덱스는 INSERT와 DELETE 작업과는 다르게 UPDATE 작업에는 부하가 없을 수도 있다.
- 마) 인덱스를 활용하여 데이터를 조회할 때 인덱스를 구성하는 컬럼들의 순서는 SQL 실행 성능과 관계가 없다.

- ① 가, 나, 다
- ② 가, 다, 라
- ③ 다, 라, 마
- ④ 가, 다, 마

✅ 정답: ① (가, 나, 다)

### 🐼 쉬운 해설:

인덱스는 조회엔 좋지만 DML엔 부하가 생겨요. 클러스터형 인덱스는 IOT와 비슷해요!

### 📖 전문 해설:

- 가: DML 시 인덱스 유지 비용 발생
- 나: 조건 검색 시 인덱스가 일반적으로 빠름
- 다: SQL Server의 클러스터형 인덱스는 Oracle의 IOT와 유사한 구조

### 🧠 기억법:

조회 = 인덱스 유리

DML = 인덱스 부하

### 📁 관련 암기카드

- 📇 카드 74: 인덱스 특징
- 📇 카드 76: 인덱스 활용 전략

[문제 141]

아래의 옵티마이저와 실행계획 대한 설명 중에서 옳은 것을 모두 묶은 것은?

- 가) ORACLE의 규칙기반 옵티마이저에서 가장 우선 순위가 높은 규칙은 Single row by rowid 액세스 기법이다.
- 나) 비용기반 옵티마이저는 테이블, 인덱스, 컬럼 등 객체의 통계정보를 사용하여 실행계획을 수립하므로 통계정보가 변경되면 SQL의 실행계획이 달라질 수 있다.
- 다) ORACLE의 실행계획에 나타나는 기본적인 Join 기법으로는 NL Join, Hash Join, Sort Merge Join 등이 있다.
- 라) 다양한 Join 기법 중 NL Join은 DW 등에서 데이터를 집계하는 업무에 많이 사용된다.

- ① 가, 다
- ② 가, 나, 다
- ③ 나, 다
- ④ 나. 다. 라

✅ 정답: ② (가, 나, 다)

🐱 쉬운 해설:

RBO는 ROWID를 가장 우선시하고, CBO는 통계 기반이에요. Oracle은 3가지 Join을 기본으로 써요!

📖 전문 해설:

- 가: RBO는 ROWID 액세스를 최우선
- 나: CBO는 통계정보 기반으로 비용 계산
- 다: Oracle은 NL, Hash, Sort Merge Join을 기본으로 사용

🧠 기억법:

RBO = 규칙 기반, ROWID 우선

CBO = 비용 기반, 통계 활용

📇 관련 암기카드

- 📇 카드 70: 옵티마이저
- 📇 카드 71: 실행계획
- 📇 카드 77: Join 기법

[문제 142]

다음 중 Nested Loop Join에 대한 설명으로 가장 부적절한 것은?

- ① 조인 칼럼에 적당한 인덱스가 있어서 자연조인(Natural join)이 효율적일 때 유용하다.
- ② Driving Table의 조인 데이터 양이 큰 영향을 주는 조인 방식이다.
- ③ 소트 머지 조인(Sort Merge Join)하기에 두 테이블이 너무 커서 소트(Sort) 부하가 심할 때 유용하다.
- ④ 유니크 인덱스를 활용하여 수행시간이 적게 걸리는 소량 테이블을 온라인 조회하는 경우 유용하다.

✅ 정답: ③

🐱 쉬운 해설:

Sort Merge Join이 부하가 클 때 Nested Loop Join을 쓰는 건 아니에요!

📖 전문 해설:

- NL Join은 인덱스가 있을 때 유리
- 소량 데이터에 적합
- Sort Merge Join은 대량 데이터에 적합
- NL Join은 Sort 부하 대체용이 아님

## 🧠 기억법:

NL Join = 소량, 인덱스

Sort Merge = 대량, 정렬

## 📌 관련 암기카드

- 📌 카드 77: Join 기법
- 📌 카드 78: Nested Loop 특징

[문제 143]

다음 중 아래와 같은 SQL에서 나타날 수 있는 Join 기법으로 가장 적절한 것은?

[DEPT 테이블 INDEX 정보]

PK\_DEPT : DEPTNO

[EMP 테이블 INDEX 정보]

PK\_EMP : EMPNO

IDX\_EMP\_01 : DEPTNO

[SQL]

SELECT \*

FROM DEPT D

WHERE D.DEPTNO = 'A001'

AND EXISTS (SELECT 'X' FROM EMP E WHERE D.DEPTNO = E.DEPTNO)

- ① HASH ANTI JOIN
- ② HASH SEMI JOIN
- ③ NESTED LOOP ANTI JOIN
- ④ NESTED LOOP SEMI JOIN

✅ 정답: ④ NESTED LOOP SEMI JOIN

## 🐻 쉬운 해설:

EXISTS는 SEMI JOIN이에요. 조건 만족 여부만 확인하니까요!

## 📖 전문 해설:

- EXISTS → SEMI JOIN
- DEPT가 드라이빙 테이블
- EMP에 조건 만족하는 행이 있는지만 확인
- NESTED LOOP 방식으로 처리



## 🧠 기억법:

EXISTS = SEMI JOIN

NOT EXISTS = ANTI JOIN

## 📌 관련 암기카드

- 📌 카드 77: Join 기법
- 📌 카드 78: Join 유형 비교

[문제 144]

다음 중 SMJ(Sort Merge Join)에 대한 설명으로 가장 부적절한 것은?

- ① 조인 칼럼에 적당한 인덱스가 없어서 NL 조인(Nested Loops)가 비효율적일 때 사용할 수 있다
- ② Driving Table의 개념이 중요하지 않은 조인 방식이다
- ③ 조인 조건의 인덱스의 유무에 영향 받지 않는다
- ④ EQUI(=) 조인 조건에서만 동작한다.

## ✅ 정답: ④

## 🐼 쉬운 해설:

Sort Merge Join은 동등 조건뿐 아니라 범위 조건에서도 사용할 수 있어요!

## 📖 전문 해설:

- SMJ는 인덱스 없어도 사용 가능
- Driving Table 개념 없음
- 범위 조건도 사용 가능
- EQUI 조건만 가능하다는 건 오해

## 🧠 기억법:

SMJ = 정렬 기반, 범위 조건 가능

## 📌 관련 암기카드

- 📌 카드 77: Join 기법
- 📌 카드 79: Sort Merge 특징

[문제 145]

해싱(Hashing) 기법을 이용하여 조인을 하는 해시조인(Hash join)은 한쪽 테이블이 주 메모리의 가용 메모리에 담길 정도로 충분히 작고 해시 키 속성에 중복 값이 적을 때 효과적이다. 다음 중 해시조인이 더 효과적일 수 있는 조건에 대한 설명으로 가장 부적절한 것은?

- ① 조인 컬럼에 적당한 인덱스가 없어서 자연조인(Natural join)이 비효율적일 때
- ② 자연조인(Natural join)시 드라이빙(driving) 집합 쪽으로 조인 액세스량이 많아 Random 액세스 부하가 심할 때
- ③ 소트 머지 조인(Sort Merge Join)을 하기에는 두 테이블이 너무 커서 소트(Sort) 부하가 심할 때
- ④ 유니크 인덱스를 활용하여 수행시간이 적게 걸리는 소량 테이블을 온라인 조회하는 경우

✔ 정답: ④

🐻 쉬운 해설:

Hash Join은 소량 테이블 조회에는 적합하지 않아요. 그건 Nested Loop Join이 더 좋아요!

📖 전문 해설:

- Hash Join은 대량 데이터에 적합
- 인덱스 없을 때 유리
- 소량 데이터는 NL Join이 더 효율적
- 해시 키 중복 적을수록 효과적

🧠 기억법:

Hash Join = 대량, 인덱스 없음

NL Join = 소량, 인덱스 있음

📌 관련 암기카드

- 📌 카드 77: Join 기법
- 📌 카드 80: Hash Join 특징

[문제 146]

다음 중 Join 기법에 대한 설명으로 가장 적절한 것은?

- ① NL Join은 선택도가 낮은(결과 행의 수가 적은) 테이블이 선행 테이블로 선택되는 것이 일반적으로 유리하다.
- ② Sort Merge Join은 동등 Join(Equi Join)에서만 사용할 수 있으므로 제약이 존재한다.
- ③ Hash Join은 결과 행의 수가 큰 테이블을 선행 테이블로 사용하는 것이 성능에 유리하다.
- ④ Hash Join은 Sort Merge Join보다 항상 우수한 성능을 보장한다.

✔ 정답: ①

### 쉬운 해설:

NL Join은 결과가 적은 테이블을 먼저 처리하는 게 유리해요!



### 전문 해설:

- NL Join: 선택도 낮은 테이블을 선행 테이블로
- SMJ: 정렬 기반, Driving Table 개념 없음
- Hash Join: 선행 테이블은 메모리에 적재 가능해야 함
- Hash Join ≠ 항상 SMJ보다 빠름

### 기억법:

NL Join = 선택도 낮은 테이블 먼저

### 관련 암기카드

-  카드 77: Join 기법
-  카드 78: Join 전략