Week 8

섹션 11.객체지향 쿼리 언어2_중급 문법

강의요약

JPQL

경로 표현식

.(점)을 찍어 객체 그래프를 탐색하는 것

```
SELECT m.username
 FROM Member m
   JOIN m.team t
   JOIN m.orders o
WHERE t.name = '팀A'
```

상태필드 : 단순히 값을 저장, 경로 탐색의 끝, 탐색X

단일 값 연관 필드 : 대상이 엔티티, 묵시적 내부 조인(inner join) 발생, 탐색 O

컬렉션 값 연관 필드 : 대상이 컬렉션, 묵시적 내부 조인 발생, 탐색X

명시적 조인, 묵시적 조인

실무에서는 가급적 명시적 조인을 사용해야한다

-> 묵시적 조인은 한눈에 파악하기 어려워 이후에 SQL 튜닝하기 까다롭기 때문.

강의요약

페치 조인(fetch Join)

JPQL에서 성능 최적화를 위해 제공하는 기능

연관된 엔티티나 컬렉션을 SQL 한 번에 함께 조회하는 기능

select m from Member m **join fetch** m.team

엔티티 페치 조인

컬렉션 페치 조인 : 일대다 관계, 데이터가 뻥튀기 된다, DISTINCT

페치 조인 한계

페치 조인 대상에는 별칭을 사용하지 않는다!

컬렉션을 페치 조인하면 페이징 API(setFirstResult, setMaxResults)를 사용할 수 없다.

여러 테이블을 조인해서 엔티티가 가진 모양이 아닌 전혀 다른 결과를 내야 하면,

페치 조인 보다는 **일반 조인**을 사용하고 필요 한 데이터들만 조회해서 **DTO로 반환**하는 것이 효과적

강의요약

다형성 쿼리

select i from Item i where type(i) IN (Book, Movie)

Type : 조회 대상을 특정 자식으로 한정

select i from Item i where treat(i as Book).author = ‘kim’

Treat : 상속 구조에서 부모 타입을 특정 자식 타입으로 다룰 때 사용

엔티티 직접 사용

JPQL에서 엔티티를 직접 사용하면 SQL에서 해당 엔티티의 기본키 값을 사용한다

select count(m) from Member m // 엔티티를 직접 사용

select count(m.id) as cnt from Member m // 엔티티의 아이디를 사용

-> 실행 SQL은 같다

강의요약

Named 쿼리
미리 정의해서 이름을 부여해두고 사용하는 JPQL
정적 쿼리만 가능, 애플리케이션 로딩 시점에 쿼리를 검증!!
@NamedQuery(name = “엔티티명.지정이름”, query= “쿼리”)
**Spring Data Jpa에서는 @Query()의 형태로 사용됨

벌크 연산
쿼리 한 번으로 여러 테이블 로우 변경(엔티티)
executeUpdate()의 결과는 영향 받은 엔티티 수 반환

```
em.createQuery(SqlString)
    .setParameter("stockAmount", 10)
    .executeUpdate();
```

주의점
벌크 연산은 영속성 컨텍스트를 무시하고 데이터베이스에 직접 쿼리하기 때문에
영속성 컨텍스트에 아무것도 없는 상태 -> 벌크 연산을 먼저 실행
영속성 컨텍스트에 값에 뭐가 있는 상태 -> 벌크 연산 수행 후 영속성 컨텍스트 초기화하는게 좋음

SQL 튜닝

SQL 튜닝이란? 최소한의 CPU, I/O, 메모리를 사용하여 최대한 빠른 시간내에 원하는 작업을 수행하도록 만드는 것

1. 가급적 WHERE 조건에서는 인덱스 컬럼을 모두 사용한다.
CONTRACT 테이블 : CONTRACT_NO, CONTRACT_TYPE 컬럼이 CON_NO_IDX 인덱스

SELECT *
FROM CONTRACT
WHERE **CONTRACT_NO** = '900000'
AND **CONTRACT_TYPE** = '1'

2. 인덱스 컬럼은 변형하여 사용하지 않도록 한다.
WHERE 조건에 인덱스 컬럼을 사용했고, 동등 연산자를 사용했다 하더라도
인덱스 컬럼에 변형을 가하게 되면 인덱스를 사용하지 못한다.

- 변형 하지 않았을때
SELECT * FROM CONTRACT
WHERE **CREATOR_ID** LIKE 'KKK%'

- 변형했을때
SELECT * FROM CONTRACT
WHERE **SUBSTR(CREATOR_ID, 1, 3)** = 'KKK'

SQL 튜닝

3. 그룹핑 쿼리를 사용할 경우 가급적 HAVING 보다는 WHERE 절에서 데이터를 필터링하라.

그룹핑 쿼리 처리순서는 **WHERE 조건이 먼저 처리되므로**
가급적 필터링 할 대상은 WHERE 조건에서 처리할 수 있게 쿼리를 작성하도록 한다.
HAVING 절은 이미 WHERE 절에서 처리된 로우들을 대상으로 조건을 감시하기 때문에 좋은 성능을 발휘하기가 힘들다.

4. DISTINCT는 가급적 사용하지 않는다.
DISTINCT는 키워드 내부적으로 정렬 작업을 수반하기 때문에 꼭 필요한 경우가 아니라면 사용하지 않는다.

5. IN, NOT IN 대신 EXISTS 와 NOT EXISTS를 사용하라

IN 사용

```
SELECT A.*
FROM CONTRACT A, CONTRACTOR B
WHERE A.CONTRACT_NO = B.CONTRACT_NO
AND B.CONTRACT_NO IN ('1111', '2222', '3333');
```

EXISTS 사용

```
SELECT *
FROM CONTRACT A
WHERE EXISTS ( SELECT 1
               FROM CONTRACTOR B
              WHERE A.CONTRACT_NO = B.CONTRACT_NO
                AND B.CONTRACT_NO IN ('1111', '2222', '3333') );
```

Index Scan

1. Full table scan

- 테이블에 속한 블록 전체를 읽어서 사용자가 원하는 데이터를 찾음
- 인덱스가 없거나 테이블의 통계정보를 수집할 때 Full table scan을 사용한다
- 힌트 : /*+ FULL(테이블명 인덱스명) */

2. Index range scan

- 인덱스의 일부분만 범위 스캔해서 DATA를 스캔하는 방법
- 인덱스 컬럼이 선두에 가공되지 않은 상태로 조건절에 있어야 Index range scan이 가능
- 소량의 데이터를 찾을 때 주로 이용
- 힌트 : /*+ INDEX(테이블명 인덱스명) */ (기본적으로 사용하는 hint는 Index range scan 실행.)

Index Scan

3. Index full scan

- 인덱스를 full 로 스캔 , 대용량 테이블이라 Table Full Scan에 대한 부담이 너무 크면 Index full scan을 활용할 수 있음
- 힌트 : /*+ INDEX_FS(테이블명 인덱스명) */

4. Index fast full scan

- 인덱스를 full 로 스캔하는데 multi block I/O가 가능해서 Index full scan 보다 더 속도가 빠르다.
- 힌트 : /*+ INDEX_FFS(테이블명 인덱스명) */

**

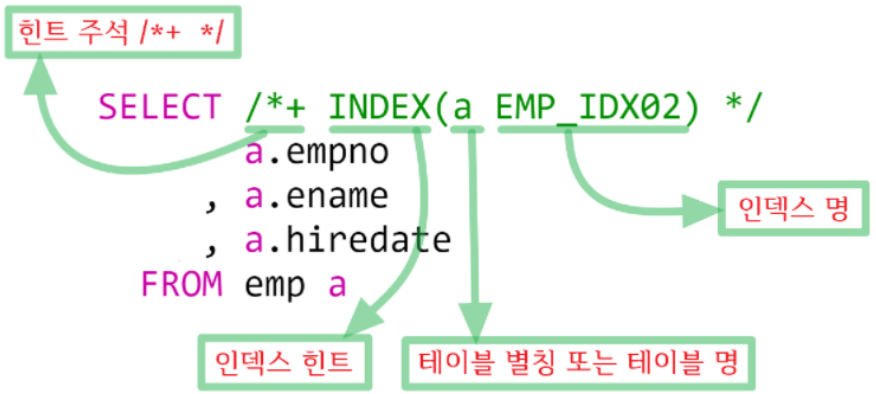
Single Block I/O : 한번에 한블록씩 가지고 오는것

Multi Block I/O : 한번에 여러 블록을 가지고 오는것

SQL Hint

SQL 튜닝의 핵심 부분으로 일종의 **지시 구문**.

SQL문 실행을 위한 데이터를 스캐닝하는 경로, 조인하는 방법 등을 알려주기 위해 SQL 사용자가 SQL 구문에 작성하는 것을 뜻함.
DB가 항상 최적의 실행 경로를 만들어 내기는 불가능하기 때문에 **직접 최적의 실행 경로를 작성해 주는 것**이다.



힌트, 인덱스, 조인의 개념을 정확히 알고 사용하지 않은 무분별한 힌트의 사용은
!!오히려 성능의 저하를 초래!!
최적의 경로를 알고 있는 경우 상황에 따라 적절하게 사용하여야 한다.