영속성 관리

엔티티 매니저(EntityManagerFactory) 팩토리와 엔티티 매니저(EntityManager)

엔티티 매니저는 엔티티를 저장, 수정, 삭제, 조회하는 등 엔티티와 관련된 모든 작업을 처리합니다. 이름 그대로, 엔티티를 관리하는 관리자 역할을 합니다. 개발자 입장에서는 엔티티 매니저를 엔티티를 저장하는 가상의 데이터베이스로 생각하시면 됩니다.

애플리케이션이 하나의 데이터베이스만 사용하는 경우, 일반적으로 **EntityManagerFactory**는 한 번만 생성됩니다.

```
EntityManagerFactory emf = Persistence.createEntityManagerFactory("jpabook");
```

위 코드는 `META-INF/persistence.xml` 파일에 있는 설정 정보를 바탕으로 **EntityManagerFactory**를 생성합니다. 이후에는 필요할 때마다 이 **EntityManagerFactory**에서 **EntityManager**를 생성하여 사용하면 됩니다.

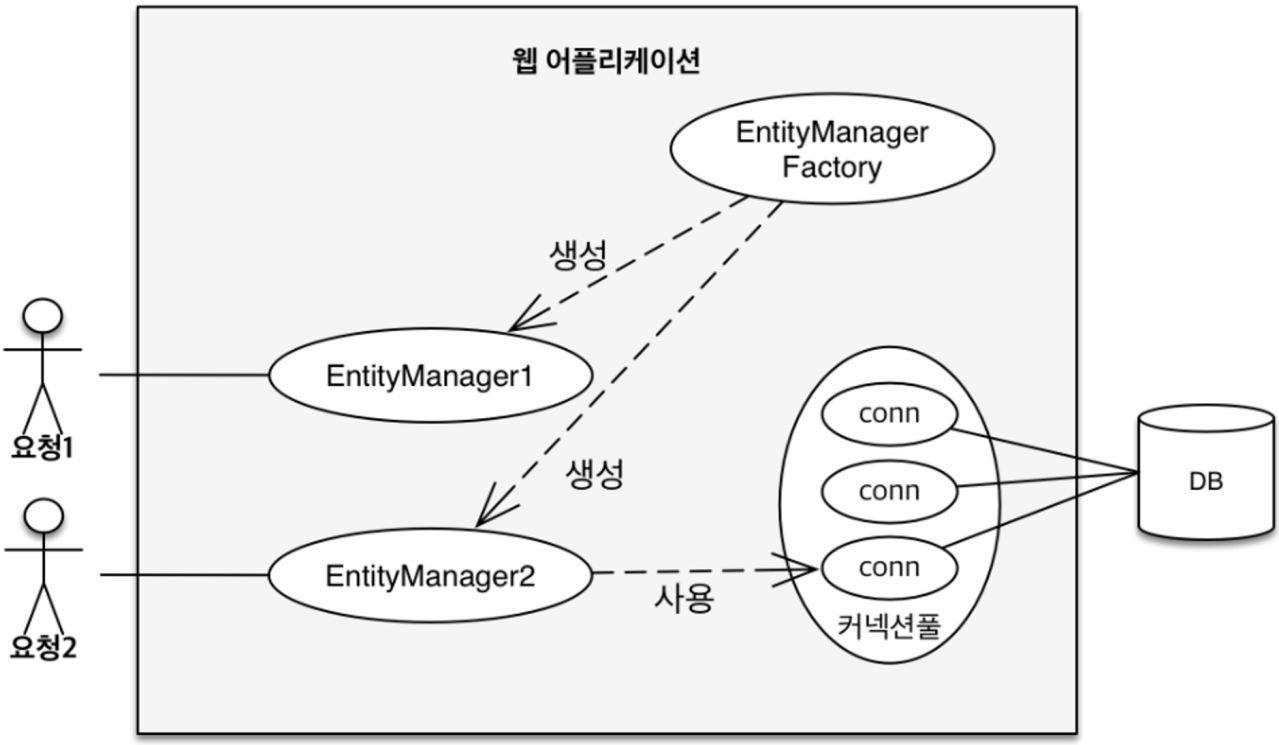
```
EntityManager em = emf.createEntityManager(); // 팩토리에서 엔티티 매니저 생성 (비용이 거의 들지 않음)
```

EntityManagerFactory는 이름 그대로 **EntityManager**를 생성하는 "공장" 역할을 합니다. 하지만 이 공장을 생성하는 데는 큰 비용이 듭니다. 따라서 **EntityManagerFactory**는 애플리케이션 전체에서 하나만 생성하고, 모든 곳에서 공유하는 방식으로 설계됩니다. 반면, **EntityManager**를 생성하는 것은 비용이 거의 들지 않으므로 필요할 때마다 쉽게 생성할 수 있습니다.

중요한 점은 **EntityManagerFactory**는 **여러 스레드가 동시에 접근해도 안전**하다는 것입니다. 따라서 여러 스레드에서 공유해서 사용할 수 있습니다. 반면, **EntityManager**는 **스레드 간에 공유하면 동시성 문제**가 발생할 수 있습니다. 즉, **EntityManager**는 한 번 생성된 이후에는 하나의 스레드에서만 사용해야 하며, 여러 스레드에서 동시에 접근하지 않도록 주의해야 합니다.

이 요약으로 보면, **EntityManagerFactory**는 비용이 크지만 한 번만 생성하여 애플리케이션 전반에 걸쳐 재사용되고, **EntityManager**는 가볍고 필요한 만큼 생성하여 각각의 스레드가 독립적으로 사용해야 하는 객체입니

다.



위 그림에서 볼 수 있듯이, 하나의 **EntityManagerFactory**에서 여러 개의 **EntityManager**가 생성되었습니다. 이 중 **EntityManager1**은 아직 데이터베이스 커넥션을 사용하지 않고 있습니다. 이는 **EntityManager**가 실제로 데이터베이스 연결을 필요로 하는 시점까지 커넥션을 얻지 않기 때문입니다. 즉, **엔티티 매니저는 실제 데이터베이스와 상호작용하기 전까지는 커넥션을 요청하지 않습니다**. 반면, **EntityManager2**는 현재 커넥션을 사용 중이며, 이는 보통 트랜잭션을 시작할 때 커넥션을 획득하기 때문입니다.

JPA 구현체(예: **Hibernate**)는 **EntityManagerFactory**를 생성할 때 **커넥션 풀(connection pool)**도 함께 생성합니다. 이 커넥션 풀은 `persistence.xml` 파일에 정의된 데이터베이스 접속 정보를 기반으로 설정됩니다. 커넥션 풀은 데이터베이스와의 연결을 효율적으로 관리하여, 애플리케이션이 필요할 때 빠르게 연결을 사용할 수 있도록 도와줍니다.

이 방식은 주로 **J2SE(Java 2 Standard Edition)** 환경에서 사용됩니다. 그러나 **J2EE(Java 2 Enterprise Edition)** 환경이나 스프링 프레임워크와 같은 엔터프라이즈 환경에서는 상황이 조금 다릅니다. 이러한 환경에서는 해당 애플리케이션 서버나 컨테이너가 제공하는 **데이터 소스(DataSource)**를 사용하여 커넥션을 관리합니다. 즉, 데이터베이스 연결에 대한 세부 사항을 애플리케이션이 직접 처리하지 않고, 컨테이너가 제공하는 데이터 소스를 통해 안전하고 효율적으로 커넥션을 사용할 수 있습니다.

정리하자면:

1. **EntityManager**는 실제로 필요한 시점까지 데이터베이스 커넥션을 요청하지 않습니다.
2. 일반적으로 트랜잭션이 시작될 때 커넥션을 획득합니다.
3. JPA 구현체(예: Hibernate)는 **EntityManagerFactory**를 생성할 때 **커넥션 풀**도 함께 생성하며, 이를 통해 성능을 최적화합니다.
4. **J2SE** 환경에서는 `persistence.xml`의 설정을 기반으로 커넥션이 관리되며, **J2EE** 환경에서는 컨테이너가 제공하는 **DataSource**를 사용하여 커넥션을 관리합니다.

영속성 컨텍스트(persistence context)

> "영속성 컨텍스트"라는 단어를 다른 자료와 함께 볼 때 혼란이 오지 않도록, 앞으로는 "persistence context"라고 부르겠습니다.

JPA에서 가장 중요한 개념 중 하나는 **Persistence Context**입니다. 이 개념은 "엔티티를 영구적으로 저장하고 관리하는 환경"을 의미합니다. **Persistence Context**는 엔티티가 데이터베이스와 연결되기 전까지 임시로 머물러 있는 메모리 공간이라고 생각할 수 있습니다.

예를 들어, **EntityManager**로 엔티티를 저장하거나 조회하면, **EntityManager**는 해당 엔티티를 **Persistence Context**에 보관하고 그 상태를 관리하게 됩니다.

```
em.persist(member);
```

이 코드를 단순히 "회원 엔티티를 저장한다"고 표현할 수 있지만, 좀 더 정확하게 말하자면, `persist()` 메서드는 **EntityManager**를 사용하여 **회원 엔티티를 Persistence Context에 저장**하는 역할을 합니다. 이 과정에서 데이터베이스에 바로 저장되는 것이 아니라, 우선 **Persistence Context**에 엔티티가 등록되고 관리됩니다. 실제 데이터베이스에 저장되는 시점은 트랜잭션이 커밋될 때 발생합니다.

`em.persist(member)` 메서드의 동작 과정:

1. **엔티티 등록**: `em.persist(member)`가 호출되면, `member` 엔티티는 **Persistence Context**에 등록됩니다. 이제 이 엔티티는 **Persistence** 상태가 되어, **JPA**가 해당 엔티티를 관리하게 됩니다.
2. **변경 사항 추적**: **Persistence Context**는 엔티티의 상태 변화를 추적합니다. 만약 `member` 엔티티의 필드 값이 변경되면, **Persistence Context**는 그 변경 사항을 인식하고, 나중에 트랜잭션이 커밋될 때 해당 변경 사항을 데이터베이스에 반영합니다.
3. **데이터베이스 반영 시점**: `persist()`가 호출되었다고 해서 곧바로 데이터베이스에 저장되는 것은 아닙니다. 실제로 데이터베이스에 반영되는 시점은 **트랜잭션이 커밋(commit)**될 때입니다. 이때 **Persistence Context**에 있는 변경 사항이 모두 데이터베이스에 반영됩니다.

Persistence Context의 특성:

- **1차 캐시 역할**: **Persistence Context**는 **1차 캐시**로 동작합니다. 즉, 동일한 트랜잭션 내에서 같은 엔티티를 여러 번 조회하더라도 데이터베이스에 다시 접근하지 않고, **Persistence Context**에 저장된 엔티티를 사용합니다.
- **변경 감지(Dirty Checking)**: **Persistence Context**는 엔티티의 상태를 추적하면서, 변경된 사항이 있을 경우 트랜잭션이 커밋될 때 자동으로 그 변경 사항을 데이터베이스에 반영합니다.
- **지연 쓰기(Write Behind)**: **Persistence Context**는 트랜잭션이 끝날 때까지 데이터를 데이터베이스에 바로 쓰지 않고 지연시킵니다. 이로 인해 여러 변경 작업이 일어나더라도 성능이 최적화될 수 있습니다.

Persistence Context는 논리적인 개념

Persistence Context는 논리적인 개념으로, 실제 눈에 보이는 물리적인 객체는 아닙니다. **EntityManager**가 생성될 때 **Persistence Context**가 함께 만들어지며, **EntityManager**를 통해 **Persistence Context**에 접근하고 관리할 수 있습니다. 엔티티 매니저는 **Persistence Context**를 조작하는 역할을 하며, 이로 인해 엔티티의 상태 관리, 변경 사항 추적 등이 가능합니다.

이러한 방식으로 **JPA**는 데이터베이스와의 직접적인 상호작용을 추상화하고, 개발자가 엔티티의 상태를 관리하기 쉽게 만들어 줍니다. `em.persist()`는 **Persistence Context**에 엔티티를 등록하고, 이 엔티티의 생명 주기를 **JPA**가 관리하도록 해주는 중요한 메서드입니다.