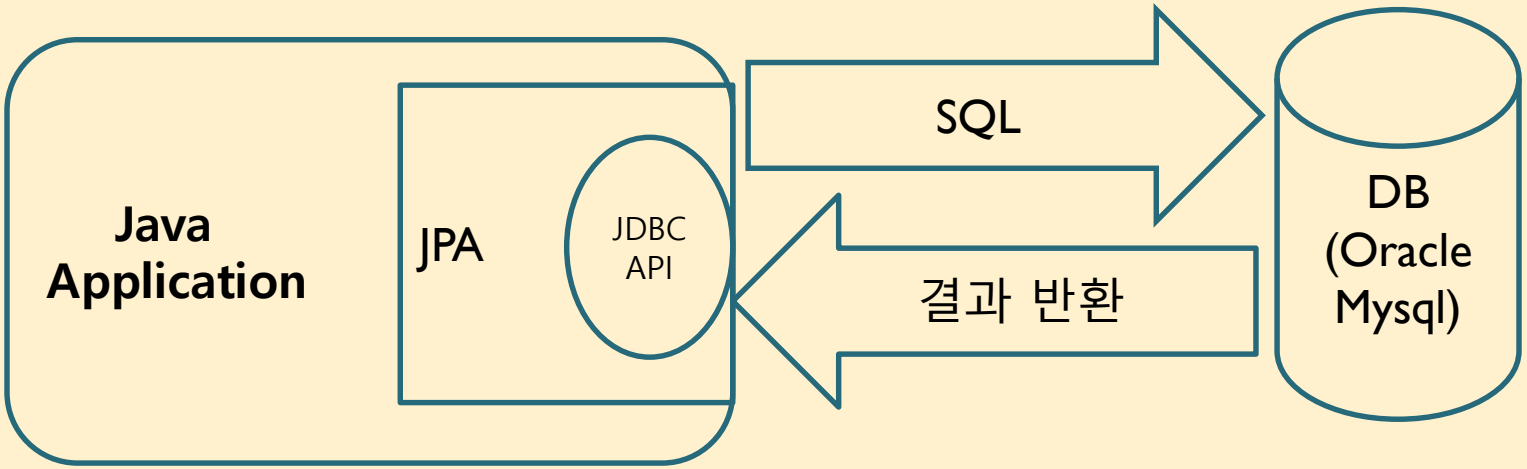
Spring Jpa01

강사 강태광

1. JPA 개념

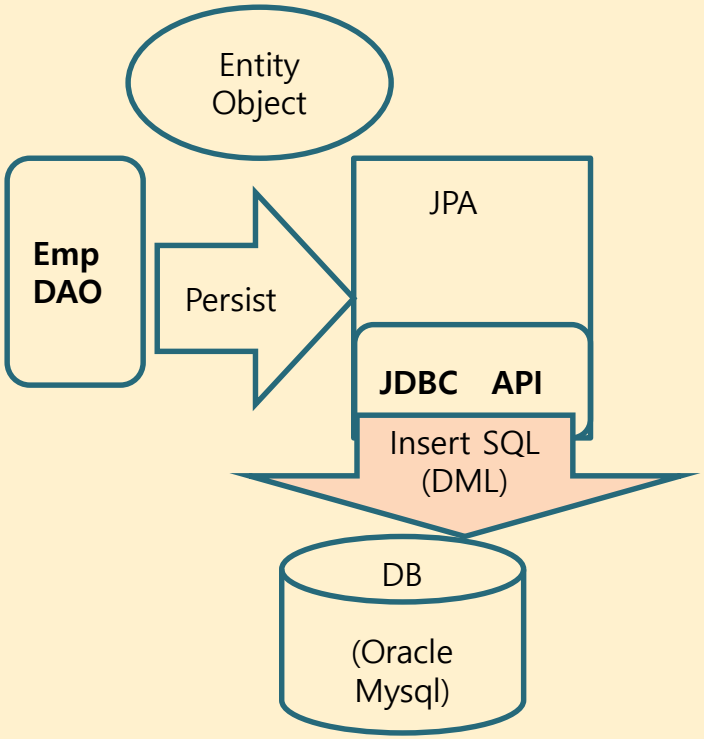
1.JPA(Java Persistence API)
 - 자바 진영의 ORM 기술 표준.

2. JPA 개념도
 - Java Program(일반적으로 DAO) 에서 JPA에 접근 (Entity)
 - JPA에서 일반적 SQL 자동 생성 (Query로 만들수도 있음)
 - JDBC API 통해 DML 작업 수행

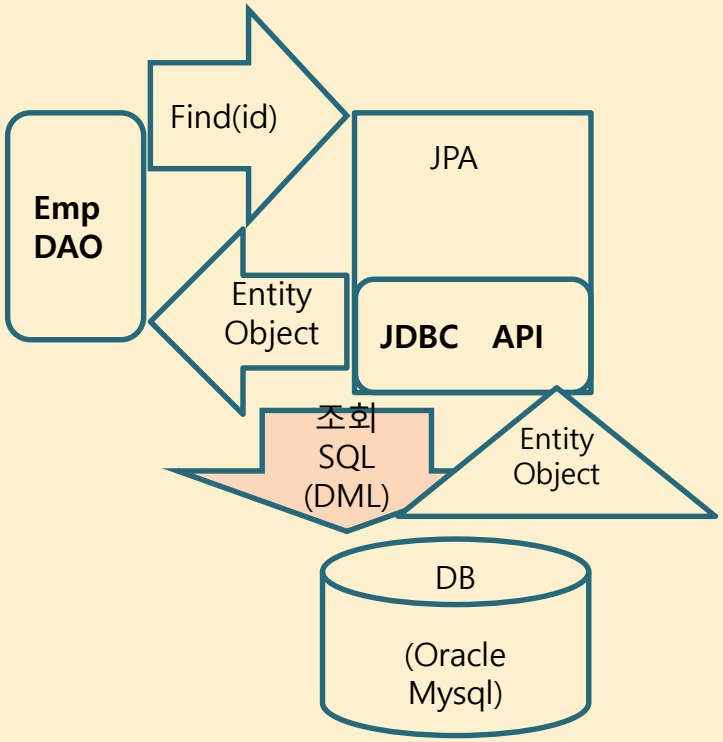


2. JPA 동작(저장 & 조회)

1. 저장
 - Entiry 분석
 - insert Emp SQL 자동 생성
 - JDBC API수행
 - DML 작업 수행

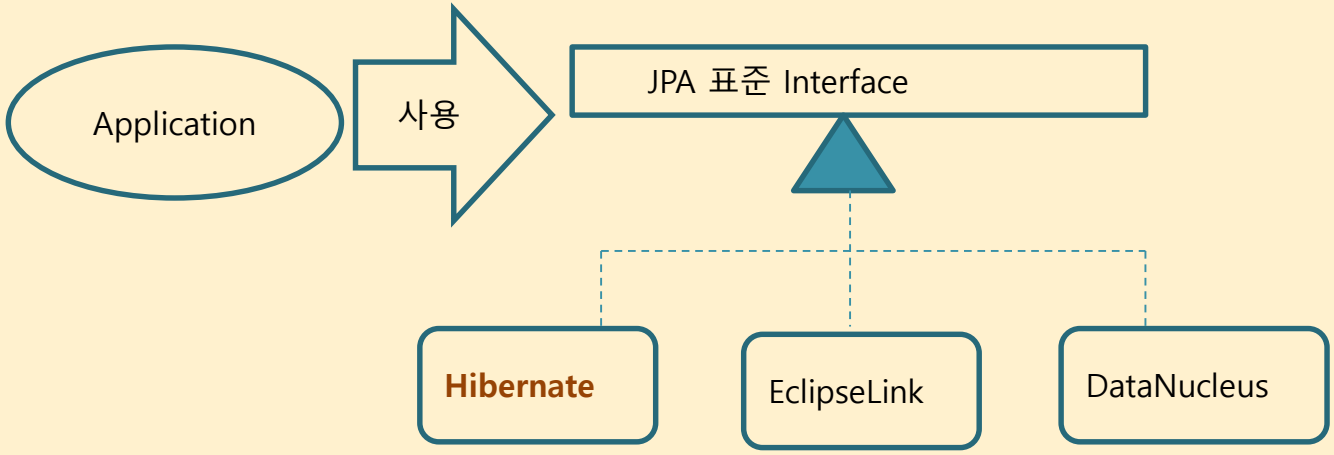


2. 조회
 - find안에 PK를 넘겨 줌
 - JDBC API수행
 - 조회된 객체를 전달



3. JPA 표준명세 와 특징

1) JPA 표준명세
- JPA는 Interface 모음
- JPA 2.1 표준 명세를 구현한 3가지 구현체
- **하이버네이트**, EclipseLink, DataNucleus
- JPA 2.0(JSR 317) 2009년 : 대부분의 ORM 기능을 포함, JPA Criteria 추가
- JPA 2.1(JSR 338) 2013년 : 스토어드 프로시저 접근, 컨버터(Converter), Entity 그래프 기능이 추가



2) JPA 특징
- SQL 중심적인 개발에서 객체 중심으로 개발
- 생산성
- 유지보수
- 성능
- 데이터 접근 추상화와 벤더 독립성
- 표준

3) SQL별로 표준을 지키지 않은 데이터베이스만의 고유한 기능, **데이터베이스 방언**
- JAP 구현체는 이러한 문제점을 해결하기 위하여 데이터베이스 방언 클래스를 제공. JPA를 사용하고있는 프로젝트의 개발자는 이에 대하여 사용하는 데이터베이스가 변경되더라도 데이터베이스 방언만 교체하면 쉽게 데이터베이스를 변경
- 예시

① 데이터 타입 : 가변문자 타입으로 MySQL은 VARCHAR, 오라클은 VARCHAR2를 사용.
② 다른 함수명 : 문자열을 자르는 함수로 예를 들면 SQL 표준은 SUBSTRING() 이지만, 오라클은 SUBSTR()을 사용.
③ 페이징 처리 : MySQL 은 LIMIT를 사용하지만 오라클은 ROWNUM을 사용

4. JPA CRUD

1) 저장: jpa.persist(emp)
Entitymanager를 통해 영속성 Context에 저장

2) 조회: Emp emp= jpa.find(empno)

3) 수정: emp.setEname("변경할 이름")

4) 삭제: jpa.remove(emp)

5. JPA Performance(캐시)

1) 1차 cache 와 동일성
- 같은 트랜잭션 안에서는 같은 엔티티를 반환 - 약간의 조회 성능 향상
- DB Isolation Level이 Read Commit이어도 애플리케이션에서 Repeatable Read 보장

2) Cache 예시

```
// SQL 1번만 실행
String empno = "1000";
Emp emp1 = jpa.find(Emp.class, empno); //SQL
Emp emp2 = jpa.find(Emp.class, empno); //캐시
println(emp1 == emp2)   // true
```

6-1. JPA Transaction 지연

1) INSERT 쓰기 지연
① 트랜잭션을 커밋할 때까지 INSERT SQL을 모음
② JDBC BATCH SQL 기능을 사용해서 한번에 SQL 전송

2) Transaction 예시

```
// Transaction 시작
transaction.begin();
em.persist(emp1);
em.persist(emp2);
em.persist(emp3);
//여기까지 INSERT SQL을 데이터베이스에 보내지 않는다.
//커밋하는 순간 데이터베이스에 INSERT SQL을 모아서 보낸다.
transaction.commit();
```

6-2. JPA Transaction 지연

1) UPDATE 쓰기 지연
① UPDATE, DELETE로 인한 로우(ROW)락 시간 최소화
② 트랜잭션 커밋 시 UPDATE, DELETE SQL 실행하고, 바로 커밋

2) Transaction 예시

```
// Transaction 시작
transaction.begin();
changeEmp(empA);
deleteEmp(empB);
비즈니스_로직_수행(); //비즈니스 로직 수행 동안 DB 로우 락이 걸리지 않는다.
// 커밋하는 순간 데이터베이스에 UPDATE, DELETE SQL을 보낸다.
// 커밋하는 순간 변경 감지(Dirty Checking) 하여 Update SQL 생성 수행
transaction.commit(); // [트랜잭션] 커밋
```

3) 지연로딩(객체가 실제 사용될 때 로딩)

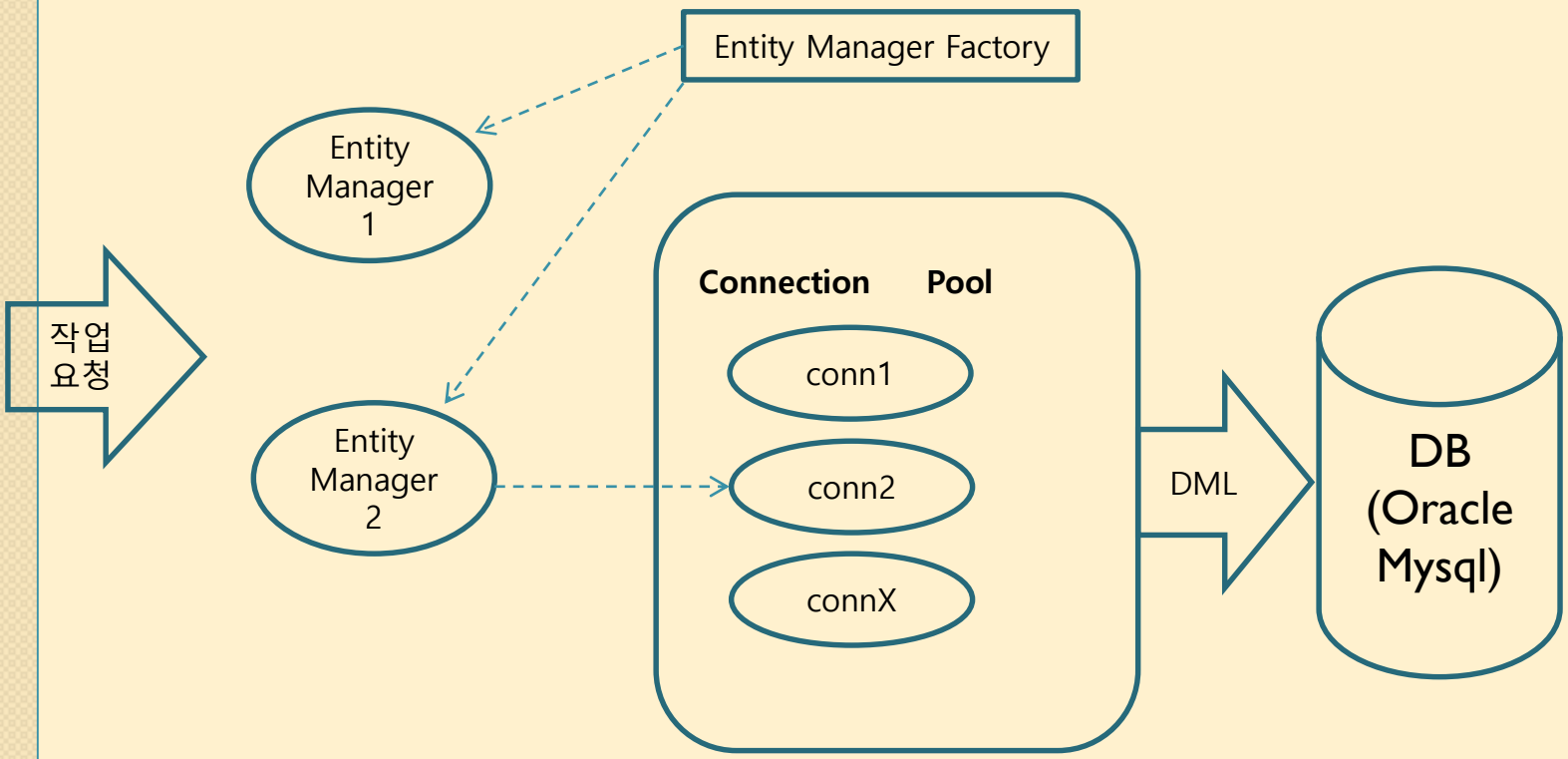
```
// SELECT * FROM MEMBER
Member member = memberDAO.find(memberId);
 Team team = member.getTeam();
// SELECT * FROM TEAM
 String teamName = team.getName();
```

4) 즉시 로딩(JOIN SQL로 한번에 연관된 객체까지 미리 조회)

```
// SELECT M.*, T.*  FROM MEMBER  JOIN TEAM
Member member = memberDAO.find(memberId);
 Team team = member.getTeam();
 String teamName = team.getName();
```

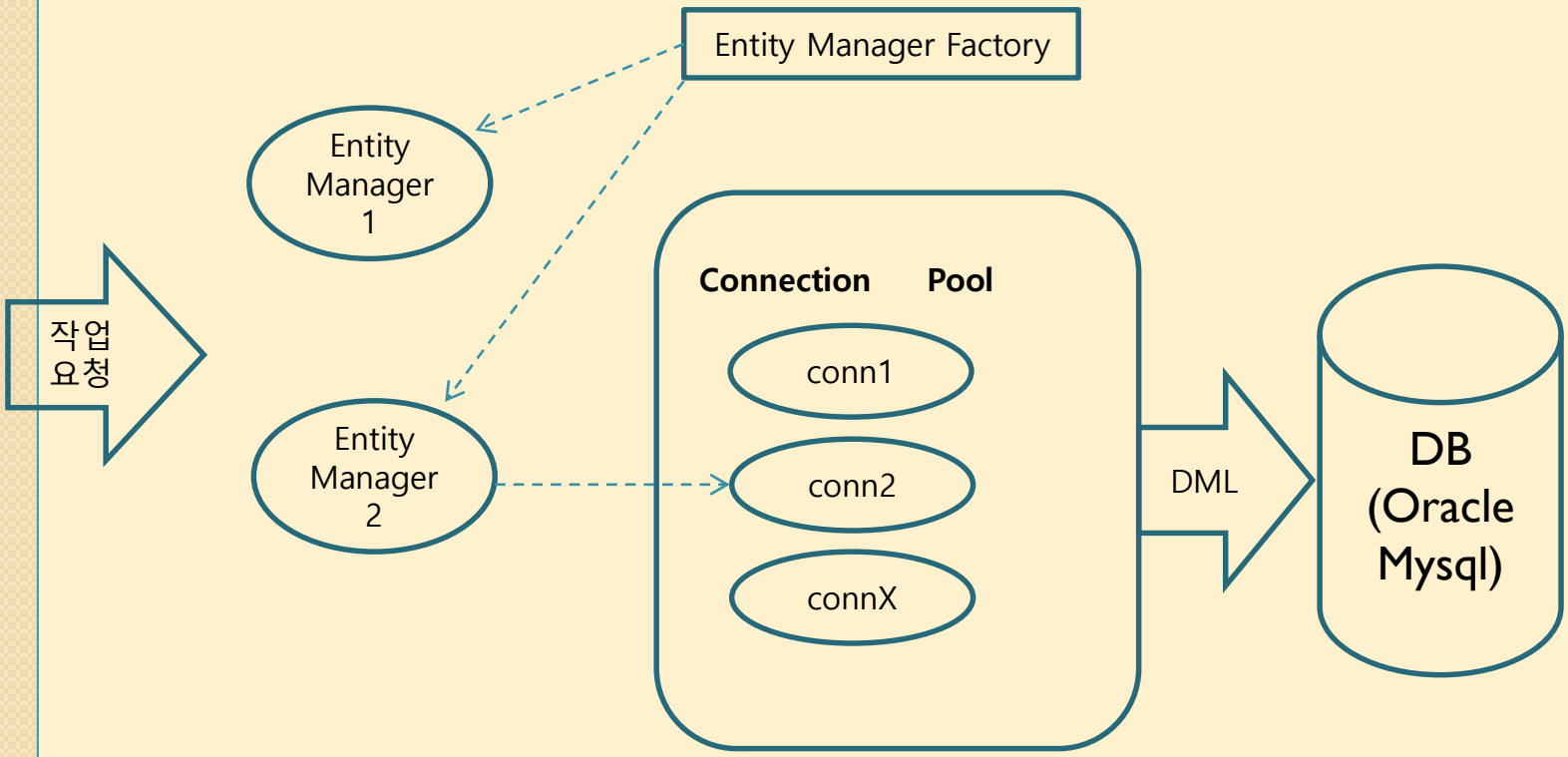
7. JPA 핵심 구성요소

1) JPA 핵심 구성요소 Entity Manager Factory 와 Entity Manager 개념도
   ① Entity Manager Factory는 각각의 Entity Manage를 관리
   ② Entity Manager는 **Connection Pool을 통해 DB와 연계**
   ③ 관련 SQL을 일부 자동으로 만들고 DML 작업 수행



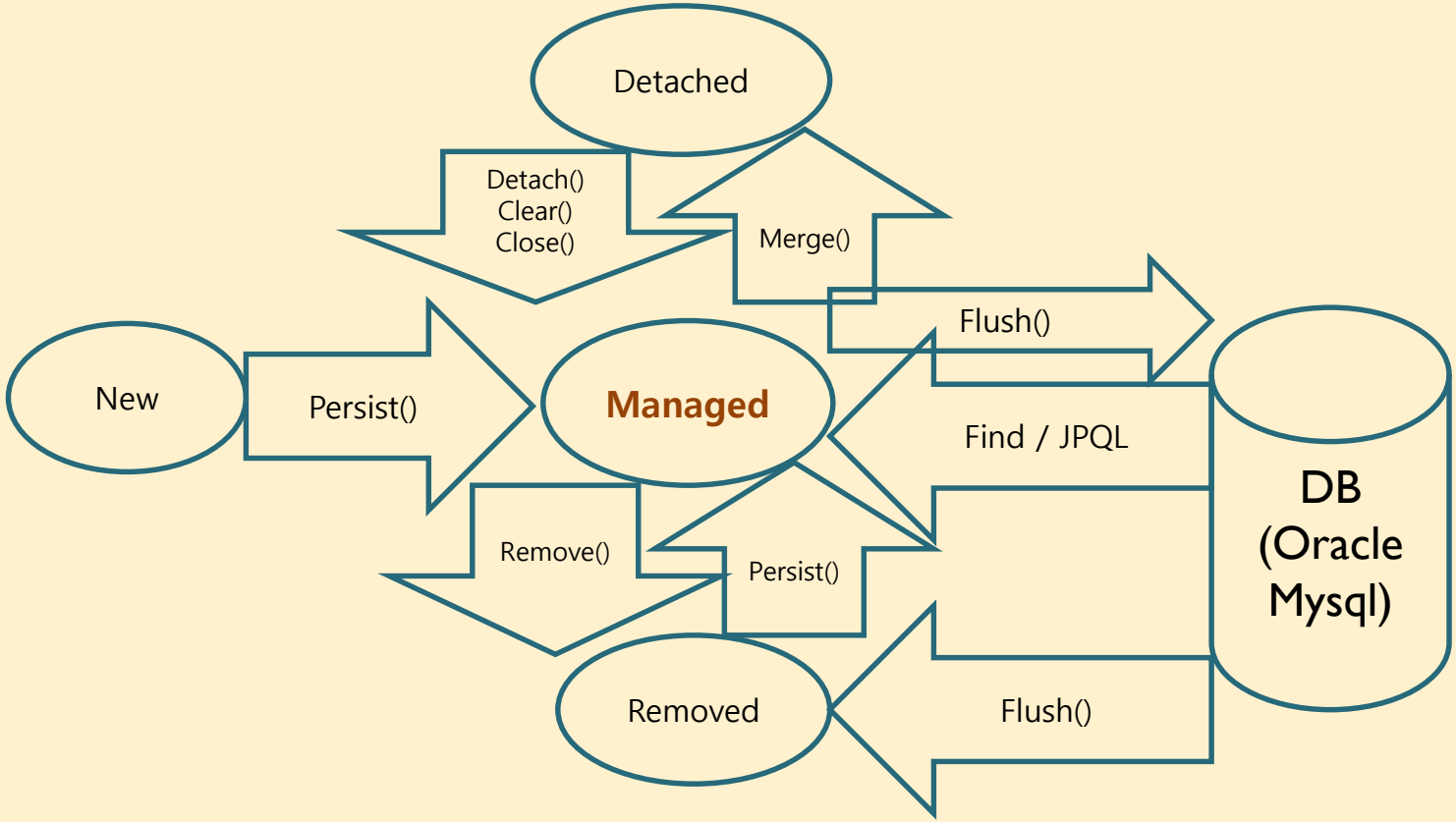
7. JPA 핵심 구성요소

1) JPA 핵심 구성요소 Entity Manager Factory 와 Entity Manager 개념도
   ① Entity Manager Factory는 각각의 Entity Manage를 관리
   ② Entity Manager는 **Connection Pool을 통해 DB와 연계**
   ③ 관련 SQL을 일부 자동으로 만들고 DML 작업 수행



8-1. JPA Entity 생명주기

1) Entity 생명주기
   ① 비영속 (new/transient) : 영속성 컨텍스트와 전혀 관계가 없는 새로운 상태
   ② 영속 (managed) : 영속성 컨텍스트에 관리되는 상태
   ③ 준영속 (detached) : 영속성 컨텍스트에 저장되었다가 분리된 상태
   ④ 삭제 (removed) : 삭제된 상태



8-2. JPA Entity 생명주기별 상태

```
1) Entity 생명주기 비영속 (new/transient)
// 객체를 생성한 상태(비영속)
Member member = new Member();
member.setId("member1");
member.setUsername("회원1");

2) Entity 생명주기 영속 (managed)
//객체를 생성한 상태(비영속)
Member member = new Member();
member.setId("member1");
member.setUsername("회원1");
EntityManager em = emf.createEntityManager();
em.getTransaction().begin();
//객체를 저장한 상태(영속)
em.persist(member);


3) Entity 생명주기 준영속 (detached)
//회원 엔티티를 영속성 컨텍스트에서 분리, 준영속 상태
em.detach(member);

4) ) Entity 생명주기 삭제 (removed)
//객체를 삭제한 상태(삭제)
em.remove(member);
```

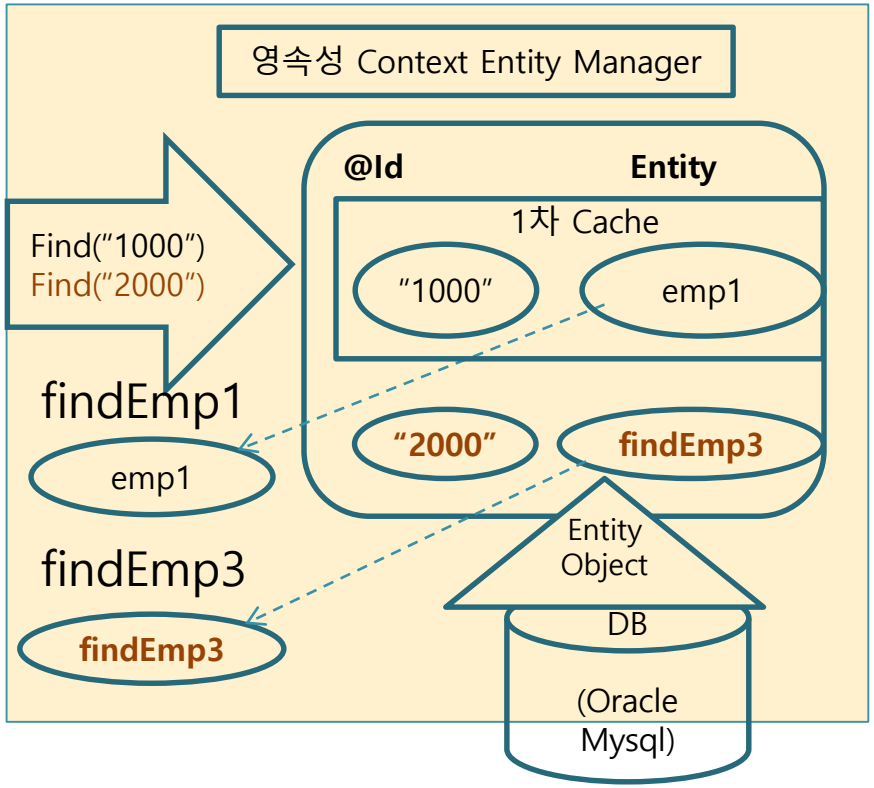
9. JPA 영속성 Context

1) 엔티티 조회, 1차 캐시
// 엔티티를 생성한 상태(비영속)
Emp emp1= new Emp();
emp.setIEmpno("1000");
Emp.setEname("양만춘");

//엔티티를 영속, 1차 캐시에 저장됨
em.persist(emp);

//1차 캐시에서 조회
Emp findEmp1 = em.find(Emp.class, "1000");

// 데이터베이스 에서 조회
Emp findEmp3 = em.find(Emp.class, "2000");



10. JPA Flush

1) Flush 개념
 - 영속성 Context의 변경내용을 데이터베이스에 반영
2) Flush 발생 시점
 • 변경 감지
 • 수정된 엔티티 쓰기 지연 SQL 저장소에 등록
 • 쓰기 지연 SQL 저장소의 쿼리를 데이터베이스에 전송(등록, 수정, 삭제 쿼리)
3) Flush 발생 방법
 • em.flush() - 직접 호출
 • 트랜잭션 commit - Flush 자동 호출
 • JPQL Query 실행 - Flush 자동 호출
4) Flush Mode Option
 • FlushModeType.AUTO : 커밋이나 쿼리를 실행할 때 Flush (기본값)
 • FlushModeType.COMMIT : 커밋할 때만 Flush
5) Flush 적용후 고려사항
 • 영속성 Context 를 비우지 않음
 • 영속성 Context 의 변경내용을 데이터베이스에 동기화
 • 트랜잭션이라는 작업 단위가 중요 -> 커밋 직전에만 동기화하면 됨
6) 준영속 상태로 만드는 method
 • em.detach(entity) : 특정 Entity 만 준영속 상태로 전환
 • em.clear() : 영속성 Context 를 완전히 초기화
 • em.close() : 영속성 Context 를 종료

11-1. JPA Entity Mapping

1. 객체와 테이블 매핑
   1) @Entity
      ① @Entity가 붙은 클래스는 JPA가 관리, 엔티티라 함
      ② JPA를 사용해서 테이블과 매핑할 클래스는 @Entity 필수
      ③ 기본 생성자 필수(파라미터가 없는 public 또는 protected 생성자)
      ④ final 클래스, enum, interface, inner 클래스 사용하지 말것
      ⑤ 저장할 필드에 final 사용 하지 말것
   2) @Entity 속성 name
      - JPA에서 사용할 엔티티 이름을 지정
      - 기본값: 클래스 이름을 그대로 사용(예: Emp)
      - 같은 클래스 이름이 없으면 가급적 기본값을 사용
   3) @Table은 엔티티와 매핑할 테이블 지정

항목	속성
name	매핑할 테이블 이름, 기본값: 엔티티 이름을 사용
catalog	데이터베이스 catalog 매핑
schema	데이터베이스 schema 매핑
uniqueConstraints	DDL 생성 시에 유니크 제약 조건 생성

11-2. JPA Database Schema 자동 생성

1) Database Schema 자동 생성
  ① DDL을 애플리케이션 실행 시점에 자동 생성
  ② 테이블 중심 에서 객체 중심으로
  ③ 데이터베이스 방언을 활용해서 데이터베이스에 맞는 적절한 DDL 생성
  ④ 이렇게 생성된 DDL은 개발 장비에서만 사용하고 운영장비에서 사용하지 말것 권장
  ⑤ 생성된 DDL은 운영서버에서는 사용하지 않거나, 적절히 다듬 은 후 사용
2) Database Schema 자동 생성 속성(hibernate.hbm2ddl.auto)

항목	속성	고려사항
create	기존테이블 삭제 후 다시 생성 (DROP + CREATE)	운영DB에는 사용 안됨, 개발 초기 는 create 또는 update
create-drop	create와 같으나 종료시점에 테이블 DROP	운영DB에는 사용 안됨, 테스트 서버는 update 또는 validate
update	변경분만 반영()	운영DB에는 사용 안됨, validate 또는 none
validate	엔티티와 테이블이 정상 매핑되었는지만 확인	스테이징과 운영 서버에서 사용
none	사용하지 않음	스테이징과 운영 서버에서 사용

3) DDL 생성 기능 예시
  ① 제약조건 추가: 회원 이름은 필수, 10자 초과할수 없다
    @Column(nullable = false, length = 10)
  ② 유니크 제약조건 추가
    @Table(uniqueConstraints = {@UniqueConstraint(name = "NAME_AGE_UNIQUE", columnNames = {"NAME", "AGE"})})
  ③ DDL 생성 기능은 DDL을 자동 생성할 때만 사용되고 JPA의 실행 로직에는 영향을 주지 않음

11-3. JPA Database Field와 Column mapping

1) Database Schema Mapping Annotation 속성(hibernate.hbm2ddl.auto)

항목	속성
@Column	컬럼 매핑
@Temporal	날짜 타입 매핑
@Enumerated	enum 타입 매핑
@Lob	BLOB, CLOB 매핑
@Transient	특정 필드를 컬럼에 매핑하지 않음(매핑 무시)

2) Database Schema Mapping Annotation **@Column속성 상세** (hibernate.hbm2ddl.auto)

항목	속성	기본값
name	필드와 매핑할 테이블의 컬럼 이름	객체의 필드 이름
insertable, updatable	등록, 변경 가능 여부	TRUE
nullable(DDL)	null 값의 허용 여부를 설정. false로 설정하면 DDL 생성 시에 not null 제약조건이 생성	
unique(DDL)	@Table의 uniqueConstraints와 같지만 한 Column에 간단히 unique 제약조건 걸 때 사용.	
columnDefinition (DDL)	데이터베이스 컬럼 정보를 직접 줄 수 있다. ex) varchar(100) default 'EMPTY'	
length(DDL)	문자 길이 제약조건, String 타입에만 사용	255
precision, scale(DDL)	BigDecimal 타입에서 사용한다(BigInteger도 사용할 수 있다). precision은 소수점을 포함한 전체 자릿수를, scale은 소수의 자릿수 . 참고로 double, float 타입에는 적용되지 않음. 아주 큰 숫자나 정밀한 소수를 다루어야 할 때만 사용.	precision=19, scale=2

11-4. JPA Database Field와 Column mapping

3) Database Schema Mapping Annotation 속성 상세 (hibernate.hbm2ddl.auto)

항목	속성	기본값
@Enumerated	- 자바 enum 타입을 매핑할 때 사용 • EnumType.ORDINAL: enum 순서를 데이터베이스에 저장 • EnumType.STRING: enum 이름을 데이터베이스에 저장	EnumType.ORDINAL (이것 사용하지 말것 권장)
@Temporal	- 날짜 타입(java.util.Date, java.util.Calendar)을 매핑할 때 사용 - LocalDate, LocalDateTime을 사용할 때는 생략 가능(최신 하이버네이트 지원) - 예시 • TemporalType.DATE: 날짜, 데이터베이스 date 타입과 매핑 (예: 2013–10–11) • TemporalType.TIME: 시간, 데이터베이스 time 타입과 매핑 (예: 11:11:11) • TemporalType.TIMESTAMP: 날짜와 시간, 데이터베이 스 timestamp 타입과 매핑(예: 2013–10–11 11:11:11)	
@Lob	데이터베이스 BLOB, CLOB 타입과 매핑 • @Lob에는 지정할 수 있는 속성이 없다. • 매핑하는 필드 타입이 문자면 CLOB 매핑, 나머지는 BLOB 매핑 • CLOB: String, char[], java.sql.CLOB • BLOB: byte[], java.sql. BLOB	
@Transient	필드 매핑하지 않음 • 데이터베이스에 저장하지 않음, 조회하지 않음 • 주로 메모리상에서만 임시로 어떤 값을 보관하고 싶을 때 사용 • @Transient private Integer temp;	

11-5. JPA Database Field와 Column mapping

1. Database Schema Mapping Annotation 기본 Key Mapping

1) 기본 키 Mapping 방법

① 직접 할당: @Id만 사용

② 자동 생성(@GeneratedValue)

- IDENTITY: 데이터베이스에 위임, MYSQL
- SEQUENCE: 데이터베이스 시퀀스 오브젝트 사용, ORACLE
- @SequenceGenerator 필요
- TABLE: 키 생성용 테이블 사용, 모든 DB에서 사용
- @TableGenerator 필요
- AUTO: 방언에 따라 자동 지정, 기본값

2) IDENTITY 상세

① 전략

- 기본 키 생성을 데이터베이스에 위임
- 주로 MySQL, PostgreSQL, SQL Server, DB2에서 사용(예: MySQL의 AUTO_ INCREMENT)
- JPA는 보통 트랜잭션 커밋 시점에 INSERT SQL 실행
- AUTO_ INCREMENT는 데이터베이스에 INSERT SQL을 실행한 이후에 ID 값을 알 수 있음
- IDENTITY 전략은 em.persist() 시점에 즉시 INSERT SQL 실행하고 DB에서 식별자를 조회

② mapping 예시

```
@Entity
public class Member {
@Id
@GeneratedValue(strategy = GenerationType.IDENTITY)
private Long id;
```

11-5. JPA Database Field와 Column mapping

1. Database Schema Mapping Annotation 기본 Key Mapping

3) SEQUENCE 상세

① 전략

- 데이터베이스 시퀀스는 유일한 값을 순서대로 생성하는 특별한 데이터베이스 오브젝트(예: 오라클 시퀀스)
- 오라클, PostgreSQL, DB2, H2 데이터베이스에서 사용

② mapping 예시

```
@Entity
@SequenceGenerator(
 name = "MEMBER_SEQ_GENERATOR",
 sequenceName = "MEMBER_SEQ", //매핑할 데이터베이스 시퀀스 이름
 initialValue = 1, allocationSize = 1)
public class Member {
 @Id
 @GeneratedValue(strategy = GenerationType.SEQUENCE, generator = "MEMBER_SEQ_GENERATOR")
 private Long id;
```

③ @SequenceGenerator 상세 속성

속성	내 용	기본값
name	식별자 생성기 이름	필수
sequenceName	- 데이터베이스에 등록되어 있는 시퀀스 이름	hibernate_sequence
initialValue	DDL 생성 시에만 사용됨, 시퀀스 DDL을 생성할 때 처음 시작하는수를 지정	1
allocationSize	시퀀스 한 번 호출에 증가하는 수(성능 최적화에 사용됨) 데이터베이스 시퀀스 값이 하나씩 증가하도록 설정되어 있으면 이 값을 반드시 1로 설정	50
catalog, schema	데이터베이스 catalog, schema 이름	

11-6. JPA Database Field와 Column mapping

1. Database Schema Mapping Annotation 기본 Key Mapping

3) TABLE 상세

① 전략

- 키 생성 전용 테이블을 하나 만들어서 데이터베이스 시퀀스를 흉내내는 전략
- 장점: 모든 데이터베이스에 적용 가능
- 단점: 성능

② mapping 예시

```
create table MY_Tab_SEQUENCES (
        sequence_name varchar(255) not null,
        next_val bigint,
        primary key ( sequence_name )
)

@Entity
@TableGenerator(
        name = "MEMBER_SEQ_GEN",
        table = " MY_Tab_SEQUENCES ",
        pkColumnValue = "MEMBER_SEQ", allocationSize = 1)
public class Member {
 @Id
 @GeneratedValue(strategy = GenerationType.TABLE, generator = "MEMBER_SEQ_GEN")
 private Long id;
```

③ @TableGenerator 상세 속성 – 계속

11-7. JPA Database Field와 Column mapping

1. Database Schema Mapping Annotation 기본 Key Mapping

③ @TableGenerator 상세 속성

속성	내 용	기본값
name	식별자 생성기 이름	필수
table	키생성 테이블명	hibernate_sequence
pkColumnName	시퀀스 컬럼명	sequence_name
valueColumnNa	시퀀스 값 컬럼명	next_val
pkColumnValue	키로 사용할 값 이름	엔티티 이름
initialValue	초기 값, 마지막으로 생성된 값이 기준	0
allocationSize	시퀀스 한 번 호출에 증가하는 수(성능 최적화에 사용됨)	50
catalog, schema	데이터베이스 catalog, schema 이름	
uniqueConstraint s(DDL)	유니크 제약 조건을 지정할 수 있음	