엔티티 매핑 어노테이션

2-1-1. @Entity, @Table, @Column

- @Entity 어노테이션은 엔티티 클래스임을 지정하며 테이블과 매핑된다.
- @Table 어노테이션은 엔티티가 매핑될 테이블을 지정하고 생략시 엔티티 클래스 이름과 같은 테이블로 매핑된다. 대부분의 JPA Persistence 제공자들은 테이블 생성 기능을 제공한다.
- EMP 클래스를 MYEMP 테이블로 매핑하는 예

```
@Entity
@Table(name="MYEMP")
public class Emp ...

아래처럼 칼럼에 대해 고유 제약조건을 지정할 수 있다.
@Entity
@Table(name="MYEMP",
        uniqueConstraints= {@UniqueConstraint(columnNames={"EMPNO"})})
public class Emp ...
```

- @Column 어노테이션은 칼럼의 이름을 이용하여 지정된 필드나 속성을 테이블의 칼럼에 매핑 한다. 생략되면 속성과 같은 이름의 칼럼으로 매핑된다.

- empno 속성을 테이블의 emp_no 컬럼으로 매핑

```
@Column(name="EMP_NO")
protected Integer empno;
```

- 한 엔티티의 속성을 여러 테이블의 칼럼에 다중 매핑 하는 경우.(city 속성은 EMP 테이블 및 CUSTOMER_CITY 테이블에 동시에 매핑된다.)

```
@Entity
public class Emp {
    @Column(name="CITY", table="CUSTOMER_CITY")
    protected String city;
...
```

- insertable, updatable 속성 사용예

```
@Column(name="EMP_NO", insertable=false, updatable=false)
protected Integer empno;
```

2-1-2. 엔티티 속성 매핑(@Basic)

- 테이블의 단순타입 칼럼 매핑에 사용된다. (자바 원시데이터 타입, String, BigInteger, Date, byte[], Byte[], char[], Character[], Serializable 인터페이스를 구현한 여러 타입)
- 아규멘트 fetch는 EAGER(즉시로딩), LAZY(지연로딩) 지정 가능 하며 EAGER가 기본값이며 optional을 true, false 형태로 지정할 수 있다.
- @Basic의 optional 속성은 런타임중 DB에 저장 되기전에 체크되는 속성이며, @Column의 nullable 속성은 DB에 테이블 스키마가 생성되는 시점에 해당 칼럼에 대해 not null로 만들어 지도록 속성을 지정한다.

```
@Basic
protected String ename;

@Basic(fetch=LAZY)
protected String getEname() {   return ename;   }

//이름을 지정한 칼럼에 지연로딩을 원할 경우 아래처럼 기술하는 것도 가능하다.
@Basic(fetch=LAZY)
@Column(name="ename")
protected String ename;
```

2-1-3. 엔티티 속성 매핑(@Enumerated)

- @Enumerated 어노테이션은 열거형 데이터를 매핑 한다.
- 열거형 타입을 정의한다.
  - ✓ public enum CustomerType { GOLD, SILVER, BRONZE };
- 열거형 값은 ordinal이라 부르는 인덱스 값과 연동되고 처음 값인 GOLD가 0, SILVER가 1을 가진다.

```
아래의 경우 CustomerType.GOLD라는 값이 들어가면 0이 저장됨
@Enumerated(EnumType.ORDINAL)     -- default
CustomerType customerType;

CustomerType.SILVER 값이 들어가면 "SILVER"가 저장된다.
@Enumerated(EnumType.STRING)
CustomerType customerType;
```

2-1-4. 엔티티 속성 매핑(@Lob)

■ @Lob 어노테이션은 테이블의 CLOB, BLOB로 매핑 되는데 속성 타입이 String, char[] 이면 CLOB, 그 외에는 BLOB 로 매핑된다.

```
import javax.persistence.Entity;
import javax.persistence.Lob;

@Entity
public class Data {
        @Lob
        private Character[] charData;

        @Lob
        //필드로 직접 매핑되며 객체가 생성될 때 로드되는 것이 아니라 실제 접근할 때 로드된다.
        @Basic(fetch=FetchType.LAZY)
        private Byte[] byteData;

        public Character[] getCharData() {              return charData;  }
        public void setCharData(Character[] charData) {       this.charData = charData; }
        public Byte[] getByteData() {                   return byteData;  }
        public void setByteData(Byte[] byteData) {    this.byteData = byteData; }
}
```

2-1-5. 엔티티 속성 매핑(@Temporal)

■ @Temporal 어노테이션은 날짜타입 필드(java.util.Date, java.util.Calendar)로 매핑 되는데 지정하지 않는 경우에는 TIMESTAMP가 기본이다. DB는 서로 다른 형태로 날짜 타입을 지원하는데 TemporalType.DATE(day, month, year 저장, 시간은 저장안됨), TemporalType.TIME(time만 저장), TemporalType.TIMESTAMP(time, day, month, year 저장) 타입이 있다. 자바 Date 클래스는 기본적으로 날짜 및 시간을 포함한다. 자바 Date 타입 속성에 @Temporal 어노테이션을 쓰면 @Temporal(TemporalType.TIMESTAMP)로 지정한다.

```
@Temporal(TemporalType.DATE)
@Column(name="LAST_UPDATE_TIME")
private Date lastUpdateTime;
```

위 예문의 경우 칼럼의 값에는 시/분/초 값은 저장되지 않는다. (예를 들면 2016-10-10과 같은 형태)

2-1-6. 엔티티 속성 매핑(@Transient)

- @Transient 어노테이션은 테이블의 칼럼에 매핑되지 않는 속성을 지정하며 임시로 값을 저장하는 용도로 사용된다. 기본적으로 @Entity 애노테이션이 붙은 클래스의 모든 속성은 모두 테이블의 칼럼(필드)으로 생성되는데 어떤 속성을 테이블의 칼럼으로 만들고 싶지 않다면 이 어노테이션을 해당 속성 또는 속성의 getter 메소드에 사용하면 된다.
- 보통 회원 가입시 비밀번호 확인을 위해 비밀번호를 한번 더 입력 받는데 이속성은 DB 테이블의 칼럼으로 매핑할 필요가 없을 것이다. 자바 객체 직렬화에서도 Transient 키워드는 직렬화에서 해당 속성을 빼 달라는 의미로 사용된다.

```
@Entity
public class Customer {
    protected Integer id;
    protected String pwd;
    @Transient
    protected String confirmPwd;
}
```

2-1-7. 엔티티 속성 매핑(@Access)

- @Access 어노테이션은 프로퍼티에 접근하는 방식을 정하는데, JPA Annocation은 클래스의 멤버 필드(AccessType.Field) 또는 Get 메소드(AccessType.Property)에 부여할 수 있으며 동시에 부여할 수 없고 반드시 한쪽에만 부여해야 한다. JPA 명세에는 속성(필드)과 Get 메소드 중 어디를 디폴트로 할지 정해져 있지 않으며 명세를 구현한 벤더에 따라 다르다. 보통은 @Id 어노테이션이 부여된 곳을 기준으로 하므로 @Id 어노테이션을 필드에 부여했으면 필드를 기준으로 영속화가 이루어지고 메소드에 부여했으면 Get 메소드를 기준으로 영속화가 이루어진다.
- 명시적으로 @Access(AccessType.FIELD/AccessType.PROPERTY)를 이용해 지정 가능하다.
- AccessType.FIELD인 경우 영속화 과정에서 필드에 데이터를 설정하거나 읽어 올때 메소드를 통하지 않고 직접 필드에 접근해서 읽어오기 때문에 Getter 메소드에 별도의 로직이 존재하는 경우 동작하지 않는다. AccessType이 PROPERTY인 경우는 그 반대로 동작한다.
- 필드명과 Get 메소드 이름이 다른 경우, 예를 들면 월급을 제외한 다른 값들은 멤버 필드를

통해 엑세스 하지만 월급의 경우 별도의 로직을 통해 변환된 값을 데이터 베이스에 저장해야 하는 경우에는 @Transient 어노테이션을 멤버 필드에 부여해 영속화에서 제외시키고 대신 Get 메소드를 통해 접근하도록 설정해 변환된 값이 저장되도록 할 수 있다.

```
// sal 필드는 영속화 제외
@Transient
private Long sal;
// Get메소드를 리턴하는 값을 값을 DB에 저장하도록 AccessType을 설정
@Access(AccessType.PROPERTY)
private Long getTotalSal() {            return this.sal*2;      }
private void setTotalSal(Long sal) {            this.sal = sal;      }
```