



HIBERNATE

HIBER NATE 5.x 시작하기

수익성의 성격을 띄지 않는다면
재배포, 수정 가능합니다.

visualkhh@gmail.com

<https://github.com/visualkhh/book-hibernate>

(2016. 12. 21)

목 차

ORM (Object Relational Mapping) 무엇인가?	5
JPA (Java Persistence API) 무엇인가?	5
HIBERNATE 무엇인가?	5
왜 JPA 를 쓰는가?	5
1. 기존 SQL 중심적인 개발시 불편하다	5
2. 객체-관계 간 모델 불일치	5
3. 상속 불일치	5
4. 관계와 연관 관계의 불일치	6
장단점	6
장점	6
단점	6
JPA, HIBERNATE Architecture	7
엔티티 상태 및 생명주기	9
비영속 상태	9
영속 상태	9
준영속 상태	9
Hibernate 셋팅	11
gradle	11
hibernate.cfg.xml	13
Session	14
Session API	14
SessionFactory	14
Hibernate-provided BasicTypes	15
Entity	18
@Entity	18
식별자	19
@GeneratedValue	19
복합 식별자 ID (KEY)	22

키선언 첫번째 방법 (@Embeddable)	23
키선언 두번째 방법 (@EmbeddedId)	24
키선언 세번째 방법 (@IdClass)	25
Join 조인	26
연관 관계	26
다중성	26
방향성	26
조인전에 먼저 알아야될 Cascade	26
1:1 @OneToOne	27
방향성을 갖자	31
@OneToMany, @ManyToOne	32
1:N, N:1, N:N 컬렉션 영속화 (List, Set, Map, Array[])	32
방향성 갖자	44
조회하기	44
@ManyToMany	47
N:N 연결하기	50
캐싱	55
1 차 캐시	55
2 차 캐시	56
cache usage 속성	56
쿼리 캐시	57
상속 전략	58
@Table-per-Class	58
@Table-per-Subclass	62
@Table-per-Concrete-Class	63
하이버네이트 질의어	64
Query Class 사용하기	64
@Embedded Objects	68
@ElementCollection	69
Pagination 페이지네이션	71

Criteria	72
네임드 쿼리	75
네이티브 쿼리	77
Groovy Template 이용하여 Dynamic Query 사용하기	78

ORM (Object Relational Mapping) 무엇인가?

RDB 테이블을 객체지향적으로 사용하기 위한 기술입니다. RDB 은 객체지향적 (상속, 다형성, 레퍼런스, 오브젝트 등)으로 접근하기 쉽지 않습니다.

때문에 ORM 을 사용해 오브젝트와 RDB 사이에 객체지향적으로 다루기 위한 기술입니다.

JPA (Java Persistence API) 무엇인가?

ORM 전문가가 참여한 EJB 3.0 스펙 작업에서 기존 EJB ORM 이던 Entity Bean 을 JPA 라고 바꾸고 JavaSE, JavaEE 를 위한 영속성(persistence) 관리와 ORM 을 위한 표준 기술입니다. JPA 는 ORM 표준 기술로 Hibernate, OpenJPA, EclipseLink, TopLink Essentials 과 같은 구현체가 있고 이에 표준 인터페이스가 바로 JPA 입니다.

HIBERNATE 무엇인가?

Boss 에서 개발한 ORM(Object Relational Mapping) 프레임워크 입니다.

왜 JPA 를 쓰는가?

1. 기존 SQL 중심적인 개발시 불편하다

- 쿼리가 변경되면 이에따른 프로그램 소스 DTO 객체의 변경도 불가피하게 일어난다
- 데이터를 가져와 객체지향적으로 관계를 Mapping 하는 일이 매번 일어난다.

!!SQL 의존적인 개발이 이루어진다.

2. 객체-관계 간 모델 불일치

관계형 데이터베이스에는 로우와 컬럼의 2 차원 형태로 데이터가 저장된다. 데이터 관계는 외래키 foreign key 형태로 표현된다. 문제는 도메인 객체를 관계형 데이터 베이스로 저장할 때 발생한다. 애플리케이션의 객체는 로우와 컬럼 형태가 아니다. 도메인 객체는 객체의 상태를 속성(변수)으로 가지고 있다. 그래서 도메인 객체 그대로 관계형 데이터베이스에 저장할 수가 없다. 이러한 불일치를 객체-관계 간 임피던스 불일치 object-relational impedance mismatch 라고 합니다.

3. 상속 불일치

상속은 객체 세계에서는 지원하지만, 관계형 스키마에서는 지원하지 않는다. 상속은 모든 객체지향 언어, 특히 자바에서 바늘과 실처럼 뗄 수 없는 특징입니다. 안타깝게도 관계형 스키마에는 상속 개념이 없습니다. 회사에서 임원과 직원의 예를 들어보면, 임원 개인도 회사의 직원이죠. 이 관계를 데이터베이스에서 표현하는 것은 테이블 간 관계 수정이 필요해서 쉽지 않습니다. 상속 없이 현실 세계의 문제 상황을 표현하는 것은 매우 복잡한 일입니다. 그런데 데이터베이스는 상속 관계와 같은 형태를 알지 못하지요. 이것을 해결할 간단한 방법은 없지만, 문제를 풀 수 있는 몇 가지 접근법이 있습니다. 이 접근법은 다양한 클래스-테이블 class-to-table 전략을 사용합니다.

4. 관계와 연관 관계의 불일치

1. SQL 중심적인 개발의 문제점

- field 하나추가시 쿼리도 바꿔야하고 VO도 바꿔야되고 ...
- SQL에 의존적인 개발을 피하기 어렵다.
- 객체답게 모델링 할수록 매핑 작업만 늘어난다

장단점

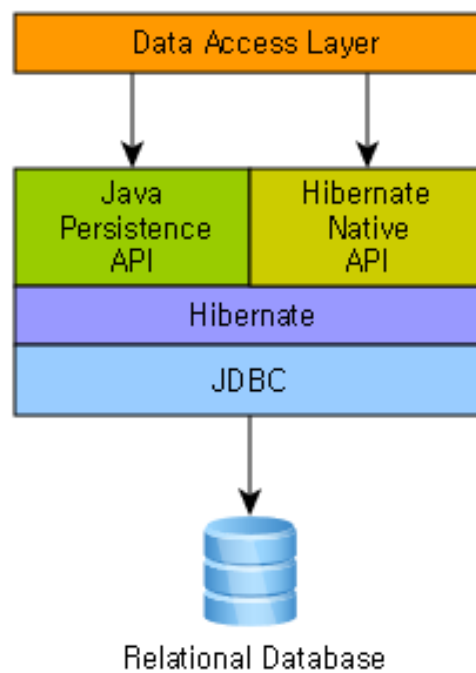
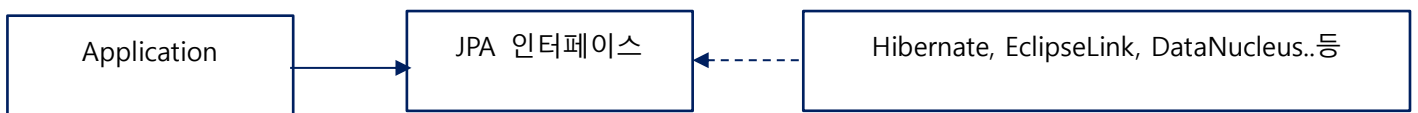
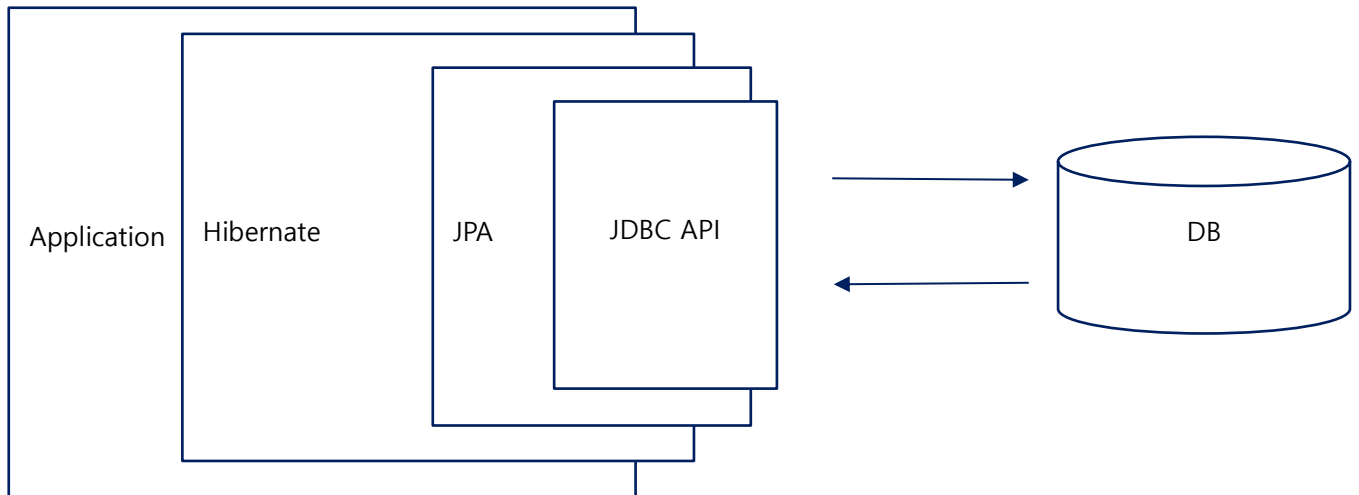
장점

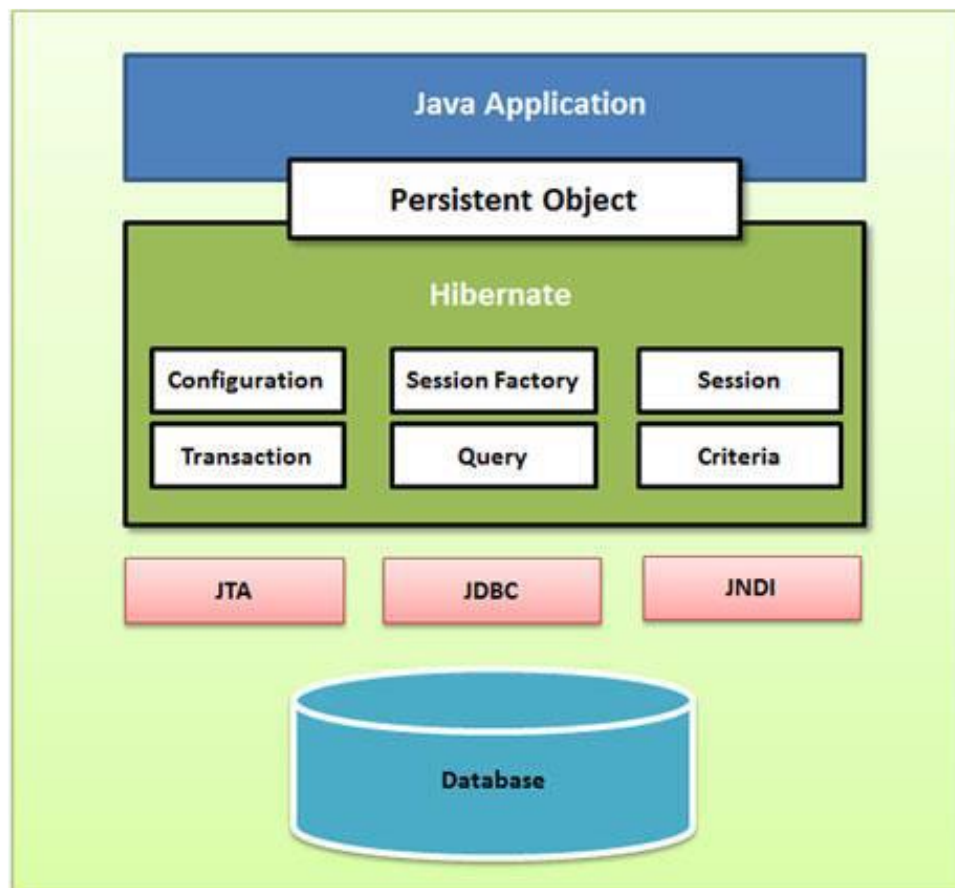
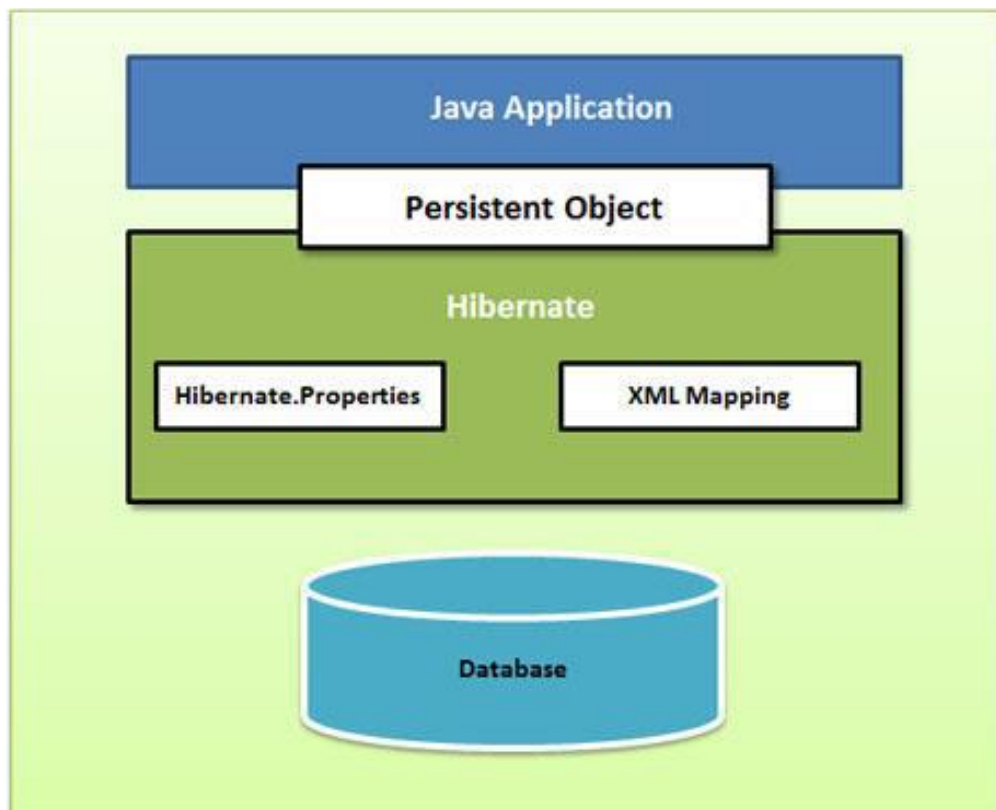
- 객체지향적으로 데이터를 관리할 수 있기 때문에 비즈니스 로직에 집중 할 수 있으며, 객체지향 개발이 가능합니다.
- 테이블 생성, 변경, 관리가 쉽다. (JPA 를 잘 이해하고 있는 경우) 로직을 쿼리에 집중하기 보다는 객체자체에 집중 할 수 있습니다.
- 빠른 개발이 가능합니다.

단점

- 어렵다. 장점을 더 극대화 하기 위해서 알아야 할게 많습니다.
- 잘 이해하고 사용하지 않으면 독이될수도 있습니다.
- 성능상 문제가 있을 수 있다.(이 문제 또한 잘 이해해야 해결이 가능합니다.

JPA, HIBERNATE Architecture





엔티티 상태 및 생명주기

비영속 상태

퍼시스턴트 객체를 처음 만들었을 때의 상태. 데이터베이스 테이블에 관련 데이터가 없으며, 연관된 Session 이 없다.

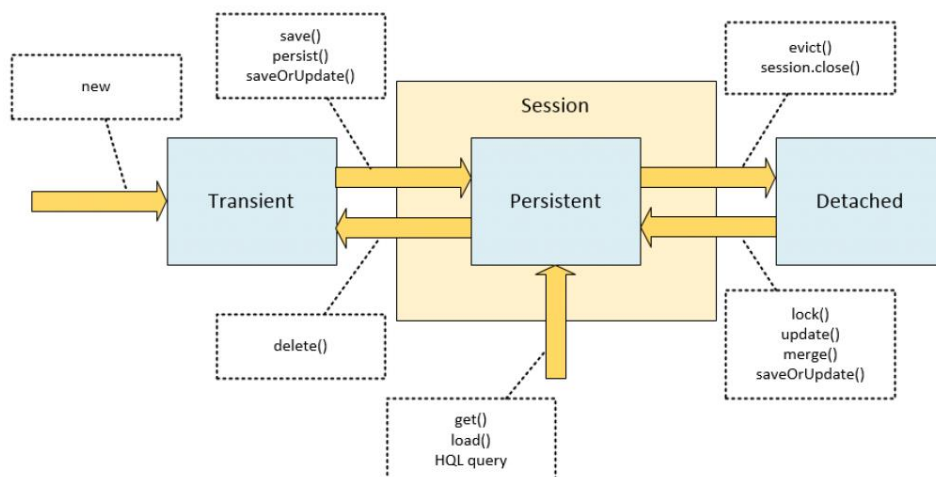
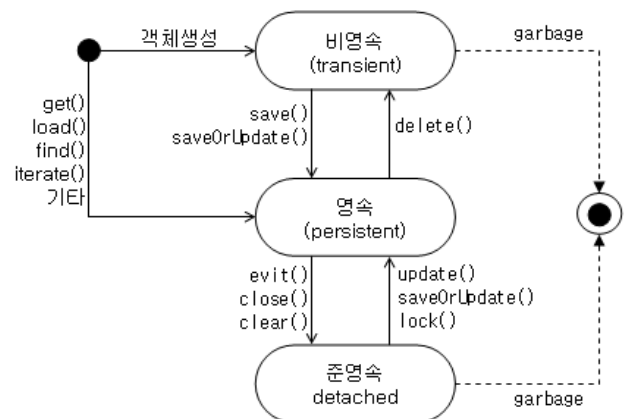
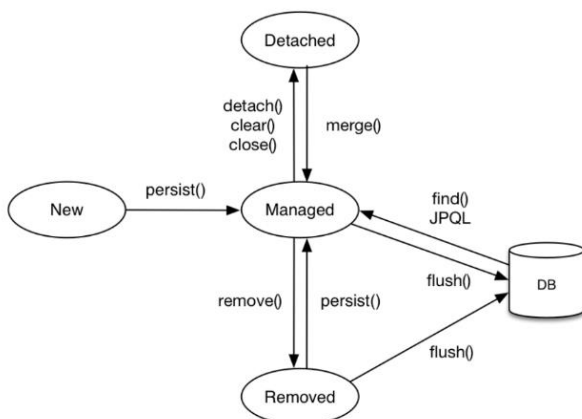
영속 상태

현재 활성화된 Session 과 연결된 퍼시스턴트 객체. 이 상태의 퍼시스턴트 객체는 고유성을 가지며, 프로퍼티 값의 변경이 Session 을 통해 자동으로 데이터베이스에 반영된다.

준영속 상태

영속 상태의 퍼시스턴트 객체가 Session 과 연결이 끊기면 준영속 상태가 된다. Hibernate 의 관리를 받지 않지만, 영속 데이터를 갖고 있다.

엔티티의 생명주기



		SomeObject obj = new SomeObject();
비명속		Session session = sessions.openSession();
		Transaction tx = session.beginTransaction();
		session.save(obj);
명속		tx.commit();
		session.close();
준명속		obj.getId();
		...

출처

https://docs.jboss.org/hibernate/orm/current/userguide/html_single/Hibernate_User_Guide.html

<http://www.baeldung.com/hibernate-save-persist-update-merge-saveorupdate>

<http://hibernate.org/search/documentation/getting-started/>

http://docs.jboss.org/hibernate/orm/5.2/userguide/html_single/Hibernate_User_Guide.html

<https://docs.jboss.org/hibernate/orm/3.3/reference/en-US/html/session-configuration.html>

<http://docs.jboss.org/hibernate/orm/>

http://www.slideshare.net/zipkyh/spring-datajpa?next_slideshow=1

<http://www.javajigi.net/pages/viewpage.action?pageId=5924>

<http://javacan.tistory.com>

Hibernate 셋팅

gradle

```
group 'com.khh'
version '1.0-SNAPSHOT'

apply plugin: 'java'

sourceCompatibility = 1.5

repositories {
    mavenCentral()
}

dependencies {
    /* slf4j */
    compile(group: 'org.slf4j', name: 'slf4j-api', version: '1.7.21')

    /* hibernate */
    compile(group: 'org.hibernate', name: 'hibernate-core', version: '5.2.5.Final')

    /* lombok */
    compile(group: 'org.projectlombok', name: 'lombok', version: '1.16.8')

    /* groovy */
    compile group: 'org.codehaus.groovy', name: 'groovy-all', version: '2.4.6'

    /* class scanner */
    compile (group: 'net.sf.corn', name: 'corn-cps', version: '1.1.7')

    /* h2 database */
    compile(group: 'com.h2database', name: 'h2', version: '1.4.193')

    /* logback-classic */
    compile(group: 'ch.qos.logback', name: 'logback-classic', version: '1.1.8')

    testCompile(group: 'junit', name: 'junit', version: '4.11')
}
```

build.gradle

Hibernate Setting (Properties Set)

```
package com.khh.hibernate.c0;
import lombok.Data;
import lombok.extern.slf4j.Slf4j;
import org.h2.tools.Server;
import org.hibernate.SessionFactory;
import org.hibernate.boot.registry.StandardServiceRegistryBuilder;
import org.hibernate.cfg.Configuration;
import org.hibernate.cfg.Environment;
import java.sql.SQLException;
@Data
@Slf4j
public class HibernateSetting implements Runnable{

    Server server = null;
    SessionFactory sessionFactory = null;

    public HibernateSetting() throws SQLException {
        //H2 Database Start
        server = Server.createWebServer("-web", "-webAllowOthers", "-webPort", "8088").start();

        //외부 XML 으로 설정
        Configuration configuration = getHibernateConfigByCode();
        //CODE BASE 로 설정
        //Configuration configuration = getHibernateConfigByCode();

        //Build
        StandardServiceRegistryBuilder serviceRegistryBuilder = new
StandardServiceRegistryBuilder().applySettings(configuration.getProperties());
        sessionFactory = configuration.buildSessionFactory(serviceRegistryBuilder.build());
        configuration.buildSessionFactory(serviceRegistryBuilder.build());
    }

    public Configuration getHibernateConfigByXML() {
        Configuration configuration = new Configuration();
        configuration.configure("hibernate.cfg.xml");
        return configuration;
    }

    public Configuration getHibernateConfigByCode() {
        Configuration configuration = new Configuration();
        configuration.setProperty(Environment.DIALECT, "org.hibernate.dialect.H2Dialect");
        configuration.setProperty(Environment.DRIVER, "org.h2.Driver");
        configuration.setProperty(Environment.URL, "jdbc:h2:mem:test");
        configuration.setProperty(Environment.USER, "sa");
        configuration.setProperty(Environment.POOL_SIZE, "55");
        configuration.setProperty(Environment.STATEMENT_BATCH_SIZE, "30");
        configuration.setProperty(Environment.AUTOCOMMIT, "true");
        configuration.setProperty(Environment.SHOW_SQL, "true");
        configuration.setProperty(Environment.FORMAT_SQL, "true");
        configuration.setProperty(Environment.CURRENT_SESSION_CONTEXT_CLASS, "thread");
        configuration.setProperty(Environment.HBM2DDL_AUTO, "create-drop");
        return configuration;
    }

    public void run() {
        //log.debug("start");
    }

    public static void main(String[] arg) throws SQLException {
        new HibernateSetting().run();
    }
}
```

매개변수로 표준 가상머신 Standard VM 인수 형식을 사용할수 있다

-Dhibernate.connection.url=jdbc:derby:memory:JH;create=true

-Dhibernate.username=mk

hibernate.cfg.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hibernate-configuration SYSTEM "http://www.hibernate.org/dtd/hibernate-configuration-3.0.dtd">

<hibernate-configuration>
  <session-factory>
    <!--
      http://www.tutorialspoint.com/hibernate/hibernate_configuration.htm
      https://docs.jboss.org/hibernate/orm/3.3/reference/en-US/html/session-configuration.html
    -->
    <property name="hibernate.dialect">org.hibernate.dialect.H2Dialect</property>
    <property name="hibernate.connection.driver_class">org.h2.Driver</property>
    <!--
      <property name="hibernate.default_schema">user</property>
    -->

    <!-- Assume test is the database name -->
    <property name="hibernate.connection.url">jdbc:h2:mem:test</property>
    <property name="hibernate.connection.username">sa</property>
    <property name="hibernate.connection.password"></property>

    <!-- JDBC connection pool -->
    <property name="hibernate.connection.pool_size">55</property>
    <property name="hibernate.jdbc.batch_size">30</property>
    <property name="hibernate.connection.autocommit">true</property>

    <!--
      <property name="cache.provider_class">org.hibernate.cache.NoCacheProvider</property>
    -->
    <property name="hibernate.show_sql">true</property>
    <property name="hibernate.format_sql">true</property>
    <!--
      <property name="hibernate.use_sql_comments">true</property>
    -->
    <!-- current_session ( e.g. jta | thread | managed | custom.Class ) -->
    <property name="hibernate.current_session_context_class">thread</property>

    <!-- hbm2ddl.auto ( e.g. validate | update | create | create-drop ) -->
    <property name="hbm2ddl.auto">create-drop</property>
    <!-- List of XML mapping files -->
    <!--
      <mapping resource="Employee.hbm.xml" />
    -->

    <!-- Infinispan 캐시 공급자 지정 -->
    <property name="hibernate.cache.provider_class">
      org.hibernate.cache.infinispan.InfinispanRegionFactory
    </property>
  </session-factory>
</hibernate-configuration>
```

Session

Session API

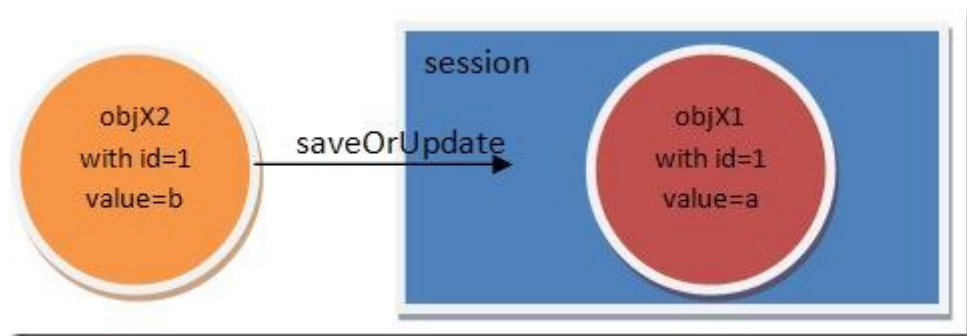
org.hibernate.SessionFactory 클래스에서 제공하는 SessionFactory는 Session 인스턴스를 생성하는 팩토리 클래스 factory class다. 이는 Thread-safe한 객체이므로 데이터가 의도하지 않게 바뀌는 것을 염려하지 않고, 여러 클래스에서사용해도 된다. Session 인스턴스가 생성될 때 매핑 정보도 함께 전달하므로 컴파일된 형태로 모든 매핑 데이터를 가진다

SessionFactory

캐시의 두 번째 수준을 관리한다. 이는 애플리케이션을 구성하는 모든 컴포넌트에도 해당된다. 전역global 캐시는 데이터베이스에서 이미 가져온동일한 결과를 여러 애플리케이션에서 요청하는 경우 사용된다.

SessionFactory가 데이터베이스 연결을 위한 열쇠 꾸러미를 가지고 있다면,

Session은 데이터베이스 접속과 데이터 이동이 이루어지는 열쇠 자체다. Session은 싱글 스레드single-threaded 객체이므로 여러 컴퍼넌트에 같이 선언되어 사용해서는 안 된다. 하나의 작업 단위를 뜻한다. 팩토리에서 세션을 가져오려면 factory.getCurrentSession() 메소드를 사용한다. 세션 객체를 얻으면 한 개의 트랜잭션 안에서 데이터베이스 작업을 수행한다. 세션과 트랜잭션은 밀접한 관련이 있다. 다음은 세션과 트랜잭션의 생애 주기를 보여주고 있다.



Hibernate-provided BasicTypes

Table 1. Standard BasicTypes

Hibernate type (org.hibernate.type package)	JDBC type	Java type	BasicTypeRegistry key(s)
StringType	VARCHAR	java.lang.String	string, java.lang.String
MaterializedClob	CLOB	java.lang.String	materialized_clob
TextType	LONGVARCHAR	java.lang.String	text
CharacterType	CHAR	char, java.lang.Character	char, java.lang.Character
BooleanType	BIT	boolean, java.lang.Boolean	boolean, java.lang.Boolean
NumericBooleanType	INTEGER, 0 is false, 1 is true	boolean, java.lang.Boolean	numeric_boolean
YesNoType	CHAR, 'N'/'n' is false, 'Y'/'y' is true. The uppercase value is written to the database.	boolean, java.lang.Boolean	yes_no
TrueFalseType	CHAR, 'F'/'f' is false, 'T'/'t' is true. The uppercase value is written to the database.	boolean, java.lang.Boolean	true_false
ByteType	TINYINT	byte, java.lang.Byte	byte, java.lang.Byte
ShortType	SMALLINT	short, java.lang.Short	short, java.lang.Short
IntegerTypes	INTEGER	int, java.lang.Integer	int, java.lang.Integer
LongType	BIGINT	long, java.lang.Long	long, java.lang.Long
FloatType	FLOAT	float, java.lang.Float	float, java.lang.Float
DoubleType	DOUBLE	double, java.lang.Double	double, java.lang.Double
BigIntegerType	NUMERIC	java.math.BigInteger	big_integer, java.math.BigInteger
BigDecimalType	NUMERIC	java.math.BigDecimal	big_decimal, java.math.BigDecimal
TimestampType	TIMESTAMP	java.sql.Timestamp	timestamp, java.sql.Timestamp
TimeType	TIME	java.sql.Time	time, java.sql.Time
DateType	DATE	java.sql.Date	date, java.sql.Date
CalendarType	TIMESTAMP	java.util.Calendar	calendar, java.util.Calendar
CalendarDateType	DATE	java.util.Calendar	calendar_date
CalendarTimeType	TIME	java.util.Calendar	calendar_time
CurrencyType	java.util.Currency	VARCHAR	currency, java.util.Currency
LocaleType	VARCHAR	java.util.Locale	locale, java.util.Locale
TimeZoneType	VARCHAR, using the TimeZone ID	java.util.TimeZone	timezone, java.util.TimeZone
URLType	VARCHAR	java.net.URL	url, java.net.URL
ClassType	VARCHAR (class FQN)	java.lang.Class	class, java.lang.Class
BlobType	BLOB	java.sql.Blob	blog, java.sql.Blob
ClobType	CLOB	java.sql.Clob	clob, java.sql.Clob
BinaryType	VARBINARY	byte[]	binary, byte[]

Table 1. Standard BasicTypes

Hibernate type (org.hibernate.type package)	JDBC type	Java type	BasicTypeRegistry key(s)
MaterializedBlobType	BLOB	byte[]	materialized_blob
ImageType	LONGVARBINARY	byte[]	image
WrapperBinaryType	VARBINARY	java.lang.Byte[]	wrapper-binary, Byte[], java.lang.Byte[]
CharArrayType	VARCHAR	char[]	characters, char[]
CharacterArrayType	VARCHAR	java.lang.Character[]	wrapper-characters, Character[], java.lang.Character[]
UUIDBinaryType	BINARY	java.util.UUID	uuid-binary, java.util.UUID
UUIDCharType	CHAR, can also read VARCHAR	java.util.UUID	uuid-char
PostgresUUIDType	PostgreSQL UUID, through Types#OTHER, which complies to the PostgreSQL JDBC driver definition	java.util.UUID	pg-uuid
SerializableType	VARBINARY	implementors of java.lang.Serializable	Unlike the other value types, multiple instances of this type are registered. It is registered once under java.io.Serializable, and registered under the specific java.io.Serializable implementation class names.
StringNvarcharType	NVARCHAR	java.lang.String	nstring
NTextType	LONGNVARCHAR	java.lang.String	ntext
NClobType	NCLOB	java.sql.NClob	nclob, java.sql.NClob
MaterializedNClobType	NCLOB	java.lang.String	materialized_nclob
PrimitiveCharacterArrayNClobType	NCHAR	char[]	N/A
CharacterNCharType	NCHAR	java.lang.Character	ncharacter
CharacterArrayNClobType	NCLOB	java.lang.Character[]	N/A

Table 2. Java 8 BasicTypes

Hibernate type (org.hibernate.type package)	JDBC type	Java type	BasicTypeRegistry key(s)
DurationType	BIGINT	java.time.Duration	Duration, java.time.Duration
InstantType	TIMESTAMP	java.time.Instant	Instant, java.time.Instant
LocalDateTimeType	TIMESTAMP	java.time.LocalDateTime	LocalDateTime, java.time.LocalDateTime
LocalDateType	DATE	java.time.LocalDate	LocalDate, java.time.LocalDate
LocalTimeType	TIME	java.time.LocalTime	LocalTime, java.time.LocalTime
OffsetDateTimeType	TIMESTAMP	java.time.OffsetDateTime	OffsetDateTime, java.time.OffsetDateTime
OffsetTimeType	TIME	java.time.OffsetTime	OffsetTime, java.time.OffsetTime
OffsetTimeType	TIMESTAMP	java.time.ZonedDateTime	ZonedDateTime, java.time.ZonedDateTime

Table 3. Hibernate Spatial BasicTypes

Hibernate type (org.hibernate.spatial package)	JDBC type	Java type	BasicTypeRegistry key(s)
JTSGeometryType	depends on the dialect	com.vividsolutions.jts.geom.Geometry	jts_geometry, or the classname of Geometry or any of its subclasses
GeolatteGeometryType	depends on the dialect	org.geolatte.geom.Geometry	geolatte_geometry, or the classname of Geometry or any of its subclasses

Entity

클래스를 영속화하려면 먼저 엔티티로 정의해야 하는데, @Entity 어노테이션으로 정의할 수 있다
또한 xml으로도 지정가능합니다.

@Entity

```
package com.khh.hibernate.c1.entity;

import lombok.Data;
import lombok.extern.slf4j.Slf4j;

import javax.persistence.Entity;
import javax.persistence.Id;

@Data
@Entity
public class UserNormal {
    @Id
    Integer seq;
    String name;
    String address;
    Integer age;
}
```

```
drop table UserNormal if exists

create table UserNormal (
    seq integer not null,
    address varchar(255),
    age integer,
    name varchar(255),
    primary key (seq)
)
```

기본값으로 Entity 클래스 이름이 테이블 이름으로 적용되고 필드 이름이 컬럼 이름으로 생성된다.

Table name, Column name 을 직접 지정할수 있다.

```
package com.khh.hibernate.c1.entity;

import lombok.Data;
import lombok.extern.slf4j.Slf4j;

import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.Id;
import javax.persistence.Table;

@Data
@Entity
@Table(name = "USER")
public class UserNaming {
    @Id
    @Column(name = "JUMIN_NUMBER")
    Integer seq;
    @Column(name = "USER_NAME")
    String name;
    @Column(name = "USER_ADDR")
    String address;
    @Column(name = "USER_AGE")
    Integer age;
}
```

```
drop table UserNormal if exists

create table USER (
    JUMIN_NUMBER integer not null,
    USER_ADDR varchar(255),
    USER_AGE integer,
    USER_NAME varchar(255),
    primary key (JUMIN_NUMBER)
)
```

식별자

각 객체는 유일한 식별자를 가지고 데이터베이스에 영속화되어야 한다. 이때 식별자를 자동으로 생성하는 다양한 방법을 활용할 수 있다.

@GeneratedValue

어노테이션을 추가하면 요구 사항에 맞춘 다른 생성 방법을 설정할 수 있다.

@GeneratedValue 어노테이션은 strategy와 generator 두 가지 속성이 있는데,

strategy 속성은 사용할 식별자 생성 타입을 가리키고 generator속성은 식별자를 생성할 메소드를 정의한다.

다음 코드는 ID 생성을 위한 IDENTITY 방법을 보여준다

기본값 : GenerationType.AUTO

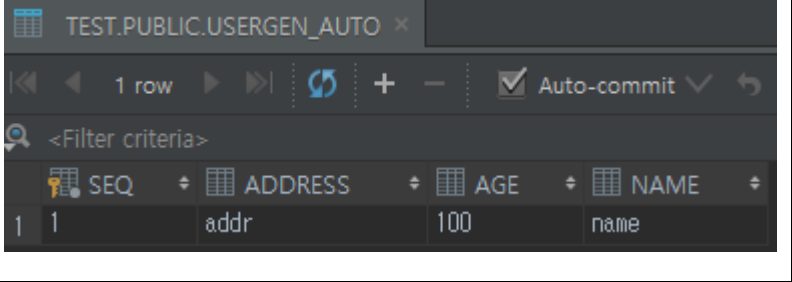
GenerationType.AUTO	기본 방법으로 다른 데이터베이스 간에도 이용할 수 있다. 하이버네이트에서는 데이터베이스를 기반으로 적절한 ID를 선택한다.
GenerationType.IDENTITY	이 설정은 몇몇 데이터베이스에서 제공하는 identity 함수를 기반으로 한다. 데이터베이스에서 고유한 식별자를 제공하는 역할을 한다.
GenerationType.SEQUENCE	몇몇 데이터베이스에서는 연속된 숫자에 관한 메커니즘을 제공하는데, 하이버네이트에서는 일련번호를 사용한다.
GenerationType.TABLE	다른 테이블의 고유한 컬럼 값에서 기본키를 생성하는데, 이 경우 TABLE 생성자를 사용한다. 시퀀스sequence 방법에서는 strategy와 generator 속성을 모두 정의해야 한다.

GenerationType.AUTO

<pre>@Data @Entity public class UserGen_AUTO { @Id @GeneratedValue(strategy = GenerationType.AUTO) Integer seq; String name; String address; Integer age; }</pre>	<pre>create sequence hibernate_sequence start with 1 increment by 1 create table UserGen_AUTO (seq integer not null, address varchar(255), age integer, name varchar(255), primary key (seq))</pre>
---	---

자동으로 hibernate_sequence를 생성한다.

▼실행

<pre>Session session = getSessionFactory().getCurrentSession(); session.beginTransaction(); UserGen_AUTO user = new UserGen_AUTO(); user.setName("name"); user.setAddress("addr"); user.setAge(100); session.save(user); session.getTransaction().commit();</pre>	
---	--

call next value for hibernate_sequence

insert
into

UserGen_AUTO

(address, age, name, seq)

values

(?, ?, ?, ?)

binding parameter [1] as [VARCHAR] - [addr]

binding parameter [2] as [INTEGER] - [100]

binding parameter [3] as [VARCHAR] - [name]

binding parameter [4] as [INTEGER] - [1]

GenerationType.IDENTITY

```
@Data
@Entity
public class UserGen_IDENTITY {
    @Id
    @GeneratedValue(strategy =
GenerationType.IDENTITY)
    Integer seq;
    String name;
    String address;
    Integer age;
}
```

```
create table UserGen_IDENTITY (
    seq integer generated by default as identity,
    address varchar(255),
    age integer,
    name varchar(255),
    primary key (seq)
)
```

▼실행

```
Session session = getSessionFactory().getCurrentSession();
session.beginTransaction();
UserGen_IDENTITY user = new UserGen_IDENTITY();
user.setName("name");
user.setAddress("addr");
user.setAge(100);
session.save(user);
session.getTransaction().commit();
```

```
insert
into
    UserGen_IDENTITY
(seq, address, age, name)
values
    (null, ?, ?, ?)
binding parameter [1] as [VARCHAR] - [addr]
binding parameter [2] as [INTEGER] - [100]
binding parameter [3] as [VARCHAR] - [name]
```

GenerationType.SEQUENCE

```
@Data
@Entity
@SequenceGenerator(name = UserGen_SEQUENCE.SEQUENCE_GEN_NAME,
sequenceName = "EMP_SEQ_GEN")
public class UserGen_SEQUENCE {
    public static final String SEQUENCE_GEN_NAME = "empSeqGen";
    @Id
    @GeneratedValue (strategy = GenerationType.SEQUENCE,
generator = SEQUENCE_GEN_NAME)
    Integer seq;
    String name;
    String address;
    Integer age;
}
```

```
create sequence EMP_SEQ_GEN start with 1
increment by 50
```

```
create table UserGen_SEQUENCE (
    seq integer not null,
    address varchar(255),
    age integer,
    name varchar(255),
    primary key (seq)
)
```

▼실행

```

Session session = getSessionFactory().getCurrentSession();
session.beginTransaction();
UserGen_SEQUENCE user = new UserGen_SEQUENCE();
user.setName("name");
user.setAddress("addr");
user.setAge(100);
session.save(user);
session.getTransaction().commit();

```

call next value for EMP_SEQ_GEN

```

insert
into
    UserGen_SEQUENCE
(address, age, name, seq)
values
    (?, ?, ?, ?)
15:18 TRACE o.h.type.descriptor.sql.BasicBinder - binding parameter [1] as [VARCHAR] - [addr]
15:18 TRACE o.h.type.descriptor.sql.BasicBinder - binding parameter [2] as [INTEGER] - [100]
15:18 TRACE o.h.type.descriptor.sql.BasicBinder - binding parameter [3] as [VARCHAR] - [name]
15:18 TRACE o.h.type.descriptor.sql.BasicBinder - binding parameter [4] as [INTEGER] - [1]

```

GenerationType.TABLE

```

@Data
@Entity
@TableGenerator(name = UserGen_TABLE.TABLE_GEN_NAME, table =
"USER_ID_TABLE")
public class UserGen_TABLE {
    public static final String TABLE_GEN_NAME =
"empTableGen";
    @Id
    @GeneratedValue(strategy = GenerationType.TABLE,
generator = TABLE_GEN_NAME)
    Integer seq;
    String name;
    String address;
    Integer age;
}

```

```

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

create table UserGen_TABLE (
    seq integer not null,
    address varchar(255),
    age integer,
    name varchar(255),
    primary key (seq)
)

```

▼실행

```

Session session = getSessionFactory().getCurrentSession();
session.beginTransaction();

UserGen_TABLE user = new UserGen_TABLE();
user.setName("name");
session.save(user);

session.getTransaction().commit();

```

```

select
    tbl.next_val
from
    USER_ID_TABLE tbl
where
    tbl.sequence_name=? for update

```

```

insert
into
    USER_ID_TABLE

```

<pre> (sequence_name, next_val) values (?,?) </pre>
<pre> update USER_ID_TABLE set next_val=? where next_val=? and sequence_name=? </pre>
<pre> select tbl.next_val from USER_ID_TABLE tbl where tbl.sequence_name=? for update </pre>
<pre> update USER_ID_TABLE set next_val=? where next_val=? and sequence_name=? </pre>
<pre> insert into UserGen_TABLE (address, age, name, seq) values (?, ?, ?, ?) </pre> <p>13:35 TRACE o.h.type.descriptor.sql.BasicBinder - binding parameter [1] as [VARCHAR] - [null]</p> <p>13:35 TRACE o.h.type.descriptor.sql.BasicBinder - binding parameter [2] as [INTEGER] - [null]</p> <p>13:35 TRACE o.h.type.descriptor.sql.BasicBinder - binding parameter [3] as [VARCHAR] - [name]</p> <p>13:35 TRACE o.h.type.descriptor.sql.BasicBinder - binding parameter [4] as [INTEGER] - [1]</p>

복합 식별자 ID (KEY)

복합 아이디composite-id 식별자 설정과 관련된 세 가지 방법

1. @Embeddable
2. @EmbeddedId
3. @IdClass

ID 값으로 사용되는 클래스에서 구현해야될것들

1. Default Constructor()
2. hashCode()
3. equals()
4. implements Serializable

tip : Lombok lib 사용한다면 위 내용을 자동으로 생성해준다.(<https://projectlombok.org/>)

키선언 첫번째 방법 (@Embeddable)

```
@Data
@Embeddable
public class UserPK_Embeddable implements
Serializable{
    String name;
    Integer number;
}
```

```
@Data
@Entity
public class UserInfo{
    @Id
    UserPK_Embeddable id;
    String address;
    Integer age;
}
```

```
create table UserInfo (
    name varchar(255) not null,
    number integer not null,
    address varchar(255),
    age integer,
    primary key (name, number)
)
```

```
Session session =
getSessionFactory().getCurrentSession();
session.beginTransaction();

UserPK_Embeddable pk = new
UserPK_Embeddable();
pk.setName("name");
pk.setNumber(128);

UserInfo_Embeddable user = new
UserInfo_Embeddable();
user.setId(pk);
user.setAddress("korea");
user.setAge(53);

session.save(user);
session.getTransaction().commit();
```

```
insert
into
    UserInfo_Embeddable
    (address, age, name, number)
values
    (?, ?, ?, ?)
binding parameter [1] as [VARCHAR] - [korea]
binding parameter [2] as [INTEGER] - [53]
binding parameter [3] as [VARCHAR] - [name]
binding parameter [4] as [INTEGER] - [128]
```

TEST.PUBLIC.USERINFO_EMBEDDEDDABLE

1 row

<

```
//select
    UserPK_Embeddable userPK = new
UserPK_Embeddable();
UserPK_IdClass();
    userPK.setName("name");
    userPK.setNumber(128);
    UserInfo_Embeddable userBydb =
session.get(UserInfo_Embeddable.class, userPK
);
    log.info("get::"+userBydb.toString());
```

```
select
    userinfo_e0_.name as name1_1_0_,
    userinfo_e0_.number as number2_1_0_,
    userinfo_e0_.address as address3_1_0_,
    userinfo_e0_.age as age4_1_0_
from
    UserInfo_Embeddable userinfo_e0_
where
    userinfo_e0_.name=?
    and userinfo_e0_.number=?
binding parameter [1] as [VARCHAR] - [name]
binding parameter [2] as [INTEGER] - [128]
```

get::UserInfo_Embeddable(id=UserPK_Embeddable(name=name, number=128), address=korea, age=53)

tip : @Embeddable 을 선언하지 않고 @Id 로 사용한다면 해당클래스의 toString()값이 들어간다

키선언 두번째 방법 (@EmbeddedId)

```
@Data
public class UserPK_EmbeddedId implements
Serializable{
    String name;
    Integer number;
}
```

```
@Data
@Entity
public class UserInfo_EmbeddedId {
    @EmbeddedId
    UserPK_EmbeddedId id;
    String address;
    Integer age;
}
```

```
create table UserInfo_EmbeddedId (
    name varchar(255) not null,
    number integer not null,
    address varchar(255),
    age integer,
    primary key (name, number)
)
```

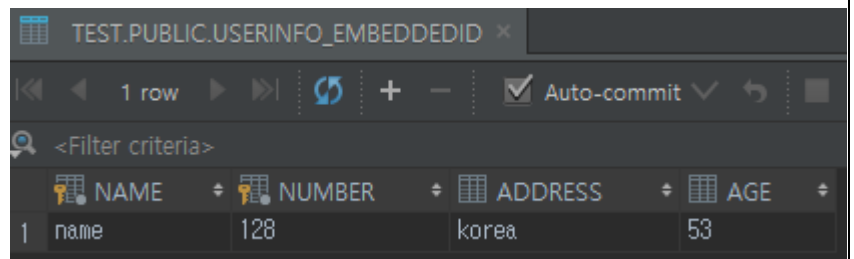
```
Session session =
getSessionFactory().getCurrentSession();
session.beginTransaction();

UserPK_EmbeddedId pk = new
UserPK_EmbeddedId();
pk.setName("name");
pk.setNumber(128);

UserInfo_EmbeddedId user = new
UserInfo_EmbeddedId();
user.setId(pk);
user.setAddress("korea");
user.setAge(53);

session.save(user);
session.getTransaction().commit();
```

```
insert
into
    UserInfo_EmbeddedId
    (address, age, name, number)
values
    (?, ?, ?, ?)
binding parameter [1] as [VARCHAR] - [korea]
binding parameter [2] as [INTEGER] - [53]
binding parameter [3] as [VARCHAR] - [name]
binding parameter [4] as [INTEGER] - [128]
```



NAME	NUMBER	ADDRESS	AGE
name	128	korea	53

```
//select
UserPK_EmbeddedId userPK = new
UserPK_EmbeddedId();
userPK.setName("name");
userPK.setNumber(128);
UserInfo_EmbeddedId userBydb =
session.get(UserInfo_EmbeddedId.class, userPK);
log.info("get::"+userBydb.toString());
```

```
select
    userinfo_e0_.name as name1_0_0_,
    userinfo_e0_.number as number2_0_0_,
    userinfo_e0_.address as address3_0_0_,
    userinfo_e0_.age as age4_0_0_
from
    UserInfo_EmbeddedId userinfo_e0_
where
    userinfo_e0_.name=?
    and userinfo_e0_.number=?
binding parameter [1] as [VARCHAR] - [name]
binding parameter [2] as [INTEGER] - [128]
```

```
get::UserInfo_EmbeddedId(id=UserPK_EmbeddedId(name=name, number=128), address=korea, age=53)
```


키선언 세번째 방법 (@IdClass)

```
@Data
public class UserPK_IdClass implements
Serializable{
    String name;
    Integer number;
}
```

```
@Data
@Entity
@IdClass(UserPK_IdClass.class)
public class UserInfo_IdClass {
    @Id
    String name;
    @Id
    Integer number;
    String address;
    Integer age;
}
```

```
create table UserInfo_IdClass (
    name varchar(255) not null,
    number integer not null,
    address varchar(255),
    age integer,
    primary key (name, number)
)
```

▼실행

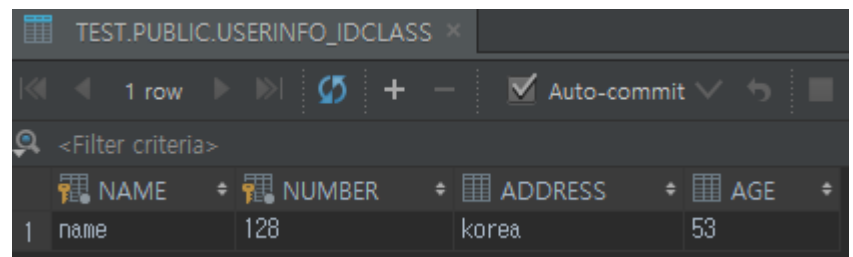
```
Session session =
getSessionFactory().getCurrentSession();
session.beginTransaction();

UserInfo_IdClass pk = new
UserInfo_IdClass();
pk.setName("name");
pk.setNumber(128);

UserInfo_IdClass user = pk;
user.setAddress("korea");
user.setAge(53);

session.save(user);
session.getTransaction().commit();
```

```
insert
into
    UserInfo_IdClass
    (address, age, name, number)
values
    (?, ?, ?, ?)
binding parameter [1] as [VARCHAR] - [korea]
binding parameter [2] as [INTEGER] - [53]
binding parameter [3] as [VARCHAR] - [name]
binding parameter [4] as [INTEGER] - [128]
```



	NAME	NUMBER	ADDRESS	AGE
1	name	128	korea	53

```
UserPK_IdClass userPK = new
UserPK_IdClass();
userPK.setName("name");
userPK.setNumber(128);
UserInfo_IdClass userBydb =
session.get(UserInfo_IdClass.class, userPK);
log.info("get::"+userBydb.toString());
```

```
select
    userinfo_i0.name as name1_2_0_,
    userinfo_i0.number as number2_2_0_,
    userinfo_i0.address as address3_2_0_,
    userinfo_i0.age as age4_2_0_
from
    UserInfo_IdClass userinfo_i0_
where
    userinfo_i0_.name=?
    and userinfo_i0_.number=?
binding parameter [1] as [VARCHAR] - [name]
binding parameter [2] as [INTEGER] - [128]
```

```
get::UserInfo_IdClass(name=name, number=128, address=korea, age=53)
```

Join 조인

연관 관계

객체 영속화 세상에서는 연관 관계 association 와 관계 relationship 에 대한 이해는 필수입니다.

연관 관계에서 반드시 기억할 두 가지는 **다중성 multiplicity** 과 **방향성 directionality** 입니다

다중성

일대일 1:1	한 테이블에서 각 레코드는 반드시 다른 테이블 의 레코드 한 개와 관계가 있다. 반대의 경우도 마찬가지다. 다른 테이블의 레코드는 0 일 수도 있다.	자동차 한 대는 오직 한 개의 엔진만 가진다.
일대다 또는 다대일 1:N, N:1	한 테이블에서 각 레코드는 다른 테이블의 0 개 또는 그 이상의 레코드와 관계가 있다.	영화 한 편은 많은 배우를 가진다(일대다) 배우 한 명은 여러 작품에서 연기할 수 있다 (다대일).
N:N	양쪽 테이블 모두 각 레코드가 다른 쪽 테이블의 0 개 또는 그 이상의 레코드와 관계가 있다.	

방향성

Car 와 Engine 의 관계에서 Car 의 속성을 질의해서 Engine 을 찾아낼수있습니다. car -> engine

Car 클래스와 Owner 클래스의 경우 주어진 Car 객체로 자동차의 주인이 누구인지 알 수 있으며,

Owner 객체로 차주의 자동차가 무엇인지 알 수 있습니다

양방향성 연관 관계를 유지할 수 있도록 Owner 객체에 Car 에 대한 참조를 제공하 고, Car 객체에는

Owner 에 대한 참조를 제공하고 있습니다.

조인전에 먼저 알아야될 Cascade

부모 객체와 자식 객체의 종속성 설정 : 하이버네이트에서는 "부모"객체가 실행되면 "자식" 혹은 "의존" 객체까지 전이되는 연산을 cascade 어트리뷰트로 처리할 수 있다. 이 기능은 모든 종류의 컬렉션과 연관에 적용된다.

타입	행위	언제
CascadeType.DETACH	엔티티가 Persistence Context에서 제거되면 (엔티티가 분리 될 것입니다)이 작업은 관계에 반영됩니다.	Finished Persistence Context 또는 명령 : entityManager.detach () entityManager.clear ()
CascadeType.MERGE	엔티티에 업데이트 된 데이터가 있으면이 작업이 관계에 반영됩니다	엔티티가 갱신되고 트랜잭션이 완료되거나 명령 : entityManager.merge ()

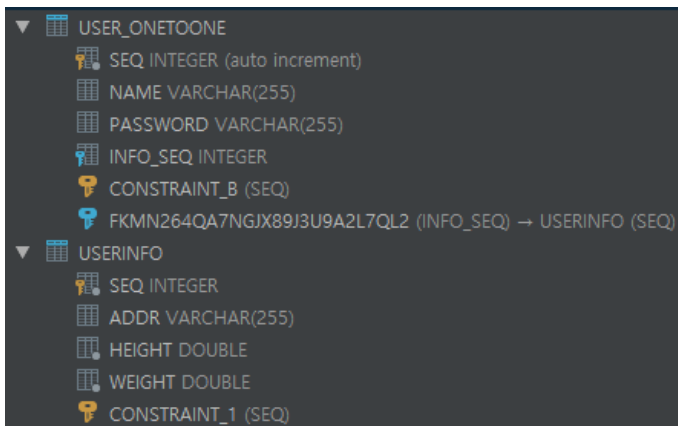
CascadeType.PERSIST	새로운 엔티티가 데이터베이스에 유지되면 조치가 관계에 반영됩니다.	트랜잭션이 끝나거나 명령 : entityManager.persist ()
CascadeType.REFRESH	엔티티가 데이터베이스와 동기화 된 데이터를 가질 때 이 조치가 반영됩니다	명령 : entityManager.refresh ()
CascadeType.REMOVE	엔티티가 데이터베이스에서 삭제되면 행동이 관계에 반영 될 것입니다.	명령 : entityManager.remove ()
CascadeType.ALL	위의 조치 중 하나가 JPA 또는 명령에 의해 호출 될 때,이 조치는 관계에 반영됩니다.	위에 설명 된 명령이나 행동.
쉽게 말하기 : 객체 상태 전이 타입 (보통 ALL 을 사용한다)		

1:1 @OneToOne

1. FK 지정

```
@Data
@Entity
public class User_OneToOne {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    Integer seq;
    String name;
    String password;
    @OneToOne(cascade = CascadeType.ALL)
    UserInfo info;
}
```

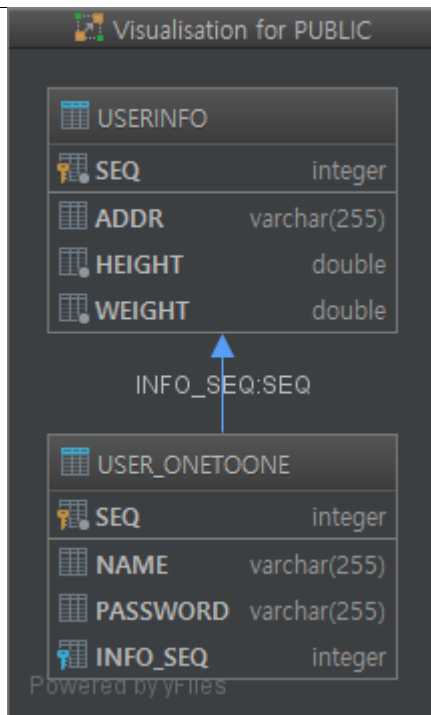
```
@Data
@Entity
public class UserInfo {
    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    Integer seq;
    String addr;
    double weight;
    double height;
}
```



```
create table UserInfo (
    seq integer not null,
    addr varchar(255),
    height double not null,
    weight double not null,
    primary key (seq)
)
```

```
create table User_OneToOne (
    seq integer generated by default as
identity,
    name varchar(255),
    password varchar(255),
    info_seq integer,
    primary key (seq)
)
```

```
alter table User_OneToOne
add                                     constraint
FKmn264qa7ngjx89j3u9a2l7ql2
foreign key (info_seq)
references UserInfo
```



집고 넘어가기 : 자동으로 User_OneToOne Table(부모테이블)에 INFO_SEQ컬럼 생기고 FK를 건다

▼실행

```

Session session =
getSessionFactory().getCurrentSession();
session.beginTransaction();

UserInfo info = new UserInfo();
info.setAddr("addr");
info.setHeight(180);
info.setWeight(70);

User_OneToOne user = new User_OneToOne();
user.setName("name");
user.setPassword("pwd");
user.setInfo(info);

session.save(user);

session.getTransaction().commit();
  
```

```

insert
into
  UserInfo
  (addr, height, weight, seq)
values
  (?, ?, ?, ?)
binding parameter [1] as [VARCHAR] - [addr]
binding parameter [2] as [DOUBLE] - [180.0]
binding parameter [3] as [DOUBLE] - [70.0]
binding parameter [4] as [INTEGER] - [1]
  
```

```

insert
into
  User_OneToOne
  (seq, info_seq, name, password)
values
  (null, ?, ?, ?)
binding parameter [1] as [INTEGER] - [1]
binding parameter [2] as [VARCHAR] - [name]
binding parameter [3] as [VARCHAR] - [pwd]
  
```

SELECT * FROM USER_ONETOONE;

SEQ	NAME	PASSWORD	INFO_SEQ
1	name	pwd	1

SELECT * FROM USERINFO;

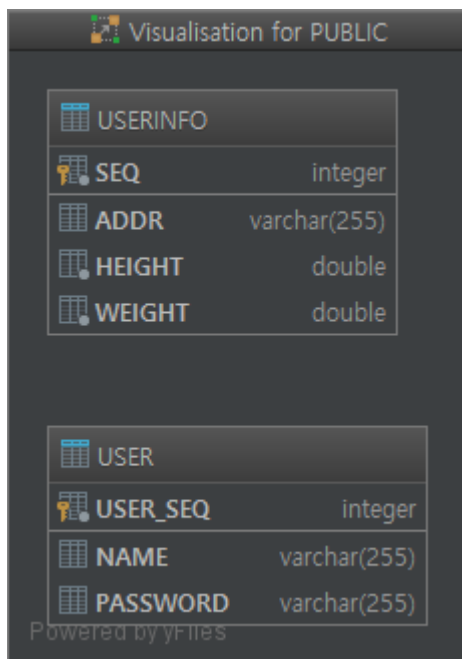
SEQ	ADDR	HEIGHT	WEIGHT
1	addr	180.0	70.0

1. PK 를 FK 지정 (아주 중요함)

```
@Data
@Entity
@SequenceGenerator(name = User.SEQ_NAME, sequenceName = User.SEQ_NAME, initialValue = 100, allocationSize = 1)
public class User {
    public static final String SEQ_NAME = "SEQ_USER";
    @Id
    @GeneratedValue(strategy = GenerationType.SEQUENCE, generator = SEQ_NAME)
    @Column(name = "USER_SEQ")
    Integer seq;
    String name;
    String password;
    @OneToOne(cascade = CascadeType.ALL)
    @JoinColumn(name = "USER_SEQ", referencedColumnName="SEQ")
    // @PrimaryKeyJoinColumn(name = "USER_SEQ", referencedColumnName="SEQ")
    UserInfo info;
}
```

```
@Data
@Entity
@GenericGenerator(name = "generator", strategy = "foreign", parameters = @Parameter(name = "property", value = "user"))
public class UserInfo {
    @Id
    @Column(name = "SEQ")
    @GeneratedValue(generator = "generator")
    Integer seq;
    String addr;
    double weight;
    double height;

    @OneToOne(cascade = CascadeType.ALL, mappedBy = "info")
    User user;
}
```



```
create table User (
    USER_SEQ integer not null,
    name varchar(255),
    password varchar(255),
    primary key (USER_SEQ)
)
```

```
create table UserInfo (
    SEQ integer not null,
    addr varchar(255),
    height double not null,
    weight double not null,
    primary key (SEQ)
)
```

▼ 실행

```
Session session = getSessionFactory().getCurrentSession();
session.beginTransaction();
```

call next value for SEQ_USER

```

UserInfo info = new UserInfo();
info.setAddr("addr");
info.setHeight(180);
info.setWeight(70);

User user = new User();
user.setName("name");
user.setPassword("pwd");

//서로의 관계를 맺어준다.
user.setInfo(info);
info.setUser(user);

session.save(user);
session.flush();
session.clear();

UserInfo userInfoBydb = session.get(UserInfo.class, 100);
if(null!=userInfoBydb && null!=userInfoBydb.getUser())
    log.debug("get(DB) Entity ----> "
+userInfoBydb.getUser().getName());

session.getTransaction().commit();

```

```

Sequence value obtained: 100
insert
into
    User
    (name, password, USER_SEQ)
values
    (?, ?, ?)
[1] as [VARCHAR] - [name]
[2] as [VARCHAR] - [pwd]
[3] as [INTEGER] - [100]

```

```

insert
into
    UserInfo
    (addr, height, weight, SEQ)
values
    (?, ?, ?, ?)
[1] as [VARCHAR] - [addr]
[2] as [DOUBLE] - [180.0]
[3] as [DOUBLE] - [70.0]
[4] as [INTEGER] - [100]

```

```

select
    userinfo0_.SEQ as SEQ1_1_0_,
    userinfo0_.addr as addr2_1_0_,
    userinfo0_.height
height3_1_0_,
    userinfo0_.weight
weight4_1_0_,
    user1_.USER_SEQ
USER_SEQ1_0_1_,
    user1_.name as name2_0_1_,
    user1_.password
password3_0_1_
from
    UserInfo userinfo0_
left outer join
    User user1_
on
    userinfo0_.SEQ=user1_.USER_SEQ
where
    userinfo0_.SEQ=?
[1] as [INTEGER] - [100]
get(DB) Entity ----> name

```

TEST.PUBLIC.USER			
1 row			
USER_SEQ	NAME	PASSWORD	
1 100	name	pwd	

TEST.PUBLIC.USERINFO				
1 row				
SEQ	ADDR	HEIGHT	WEIGHT	
1 100	addr	180	70	

자동으로 부모 테이블의 PK값을 가지고 자신의 FK쪽 값으로 사용한다

```

@GenericGenerator(name = "generator", strategy = "foreign", parameters = @Parameter(name = "property", value =
"user"))
....
@GeneratedValue(generator = "generator")
Integer seq;

```

value값은 부모Entity객체 필드명

집고 넘어가기 : @JoinColumn(name = "USER_SEQ", referencedColumnName="SEQ")

name = 자기 자신 테이블의 FK대상 컬럼명

referencedColumnName = 자식테이블 컬럼명 (안적으면 자동으로 자식테이블 ID 로 해준다.)

또한 PrimaryKeyJoinColumn 을 쓸수도 있다

```
@PrimaryKeyJoinColumn(name = "USER_SEQ", referencedColumnName="SEQ")
```

방향성을 갖자

소유자 찾기

```
@Data
@Entity
@SequenceGenerator(name = User.SEQ_NAME, sequenceName =
User.SEQ_NAME, initialValue = 100, allocationSize = 1)
public class User {
    public static final String SEQ_NAME = "SEQ_USER";
    @Id
    @GeneratedValue(strategy =
GenerationType.SEQUENCE, generator = SEQ_NAME)
    @Column(name = "USER_SEQ")
    Integer seq;
    String name;
    String password;
    @OneToOne(cascade = CascadeType.ALL)
    UserInfo info;
}
```

```
@Data
@Entity
@GenericGenerator(name = "generator", strategy =
"foreign", parameters = @Parameter(name = "property",
value = "user"))
public class UserInfo {
    @Id
    @Column(name = "SEQ")
    @GeneratedValue(generator = "generator")
    Integer seq;
    String addr;
    double weight;
    double height;

    @OneToOne(cascade = CascadeType.ALL, mappedBy = "info")
    User user;
}
```

UserInfo Entity (자식테이블) 에서 부모Entity에서 선언된 자식Entity의 필드명을 적으면된다

@OneToOne(mappedBy = "code feildName")

집고 넘어가기 : insert 했을때 Entity 는 관계설정이 되어있지 않아 로그가 찍히지 않는다 하지만 다시 SELECT 해온 Entity 는 방향성 설정이 되어 넘어온다 여기서 알수 있는것은 insert 했을때 사용했던 Entity 와 get 해온 Entity 는 다르다는것을 알수 있다.

@OneToMany, @ManyToOne

1:N, N:1, N:N 컬렉션 영속화 (List, Set, Map, Array[])

List

▼ OneToMany 조인하기 맵핑 테이블 만들어 조인하기

```
@Data
@Entity
public class User {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    Integer seq;
    String name;
    String password;

    @OneToMany(cascade = CascadeType.ALL)
    List<Auth> auths = null;
}
```

```
@Data
@Entity
@SequenceGenerator(name = Auth.SEQUENCE_NAME, sequenceName = Auth.SEQUENCE_NAME, initialValue = 100)
public class Auth implements Serializable {
    public static final String SEQUENCE_NAME = "AUTH_SEQ";
    @Id
    @GeneratedValue(strategy = GenerationType.SEQUENCE, generator = SEQUENCE_NAME)
    Integer seq;
    String auth;
    Date expiry;
}
```

```
create table Auth (
    seq integer not null,
    auth varchar(255),
    expiry timestamp,
    primary key (seq)
)
```

```
create table User (
    seq integer generated by default as identity,
    name varchar(255),
    password varchar(255),
    primary key (seq)
)
```

-- 새로운 맵핑 테이블이 생긴다

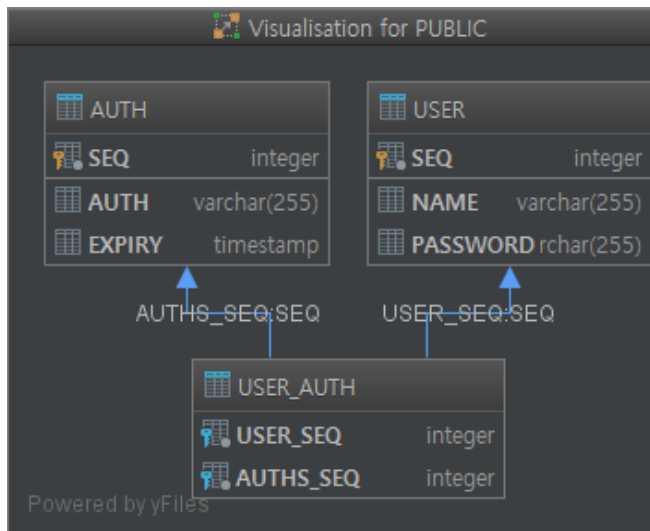
```
create table User_Auth (
    User_seq integer not null,
    auths_seq integer not null
)
```

```
alter table User_Auth
    add constraint UK_n1upof81pc3xys7s3xrovghof unique (auths_seq)
```

```
alter table User_Auth
    add constraint FKmnns1wp8mo27pxkbbwj18nuun
    foreign key (auths_seq)
    references Auth
```



```
alter table User_Auth
add constraint FK1dj4srm089e409sq8vkhogug
foreign key (User_seq)
references User
```



```
▼ AUTH
  SEQ INTEGER (auto increment)
  AUTH VARCHAR(255)
  EXPIRY TIMESTAMP
  CONSTRAINT_1 (SEQ)

▼ USER
  SEQ INTEGER (auto increment)
  NAME VARCHAR(255)
  PASSWORD VARCHAR(255)
  CONSTRAINT_2 (SEQ)

▼ USER_AUTH
  USER_SEQ INTEGER
  AUTHS_SEQ INTEGER
  FKMNNS1WP8MO27PXKBBWJL8NUUN (AUTHS_SEQ) → AUTH (SEQ)
  FKN1DJ4SRM089E409SQ8VKHOGUG (USER_SEQ) → USER (SEQ)
```

▼ 맵핑 테이블 이름바꾸기

```
@Data
@Entity
public class User {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    Integer seq;
    String name;
    String password;

    @OneToMany(cascade = CascadeType.ALL)
    @JoinTable(name = "USER_AUTH_MAPPING")
    List<Auth> auths = null;
}
```

▼ 실행

```
Session session =
getSessionFactory().getCurrentSession();
session.beginTransaction();

Auth auth1 = new Auth();
auth1.setAuth("ROLE_ADMIN");
auth1.setExpiry(new Date());
Auth auth2 = new Auth();
auth2.setAuth("ROLE_USER");
auth2.setExpiry(new Date());

User user = new User();
user.setAuths(Arrays.asList(auth1,auth2));
user.setName("name");
user.setPassword("pwd");

session.save(user);
session.getTransaction().commit();
```

```
insert
into
    Auth
    (auth, expiry, seq)
values
    (?, ?, ?)
binding parameter [1] as [VARCHAR] - [ROLE_ADMIN]
binding parameter [2] as [TIMESTAMP] - [Sat Dec 17
14:27:21 KST 2016]
binding parameter [3] as [INTEGER] - [100]
```

```
insert
into
    Auth
    (auth, expiry, seq)
values
    (?, ?, ?)
binding parameter [1] as [VARCHAR] - [ROLE_USER]
binding parameter [2] as [TIMESTAMP] - [Sat Dec 17
14:27:21 KST 2016]
binding parameter [3] as [INTEGER] - [101]
```

```
insert
```

TEST.PUBLIC.USER			
1 row			
SEQ	NAME	PASSWORD	
1	name	pwd	

TEST.PUBLIC.AUTH			
2 rows			
SEQ	AUTH	EXPIRY	
1	100	ROLE_ADMIN	2016-12-17 14:27:21.966000000
2	101	ROLE_USER	2016-12-17 14:27:21.966000000

TEST.PUBLIC.USER_AUTH		
2 rows		
USER_SEQ	AUTHS_SEQ	
1	1	100
2	1	101

```

into
    User_Auth
    (User_seq, auths_seq)
values
    (?, ?)
binding parameter [1] as [INTEGER] - [1]
binding parameter [2] as [INTEGER] - [100]

```

```

insert
into
    User_Auth
    (User_seq, auths_seq)
values
    (?, ?)
binding parameter [1] as [INTEGER] - [1]
binding parameter [2] as [INTEGER] - [101]

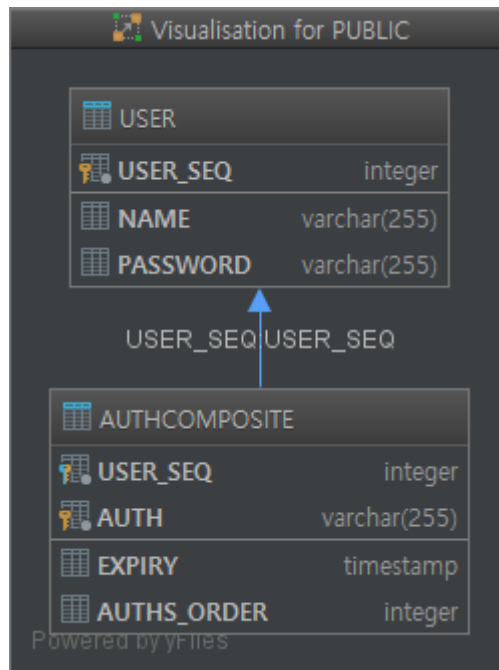
```

Array

```
@Data
@Entity
@SequenceGenerator(name = User.SEQUENCE_NAME, sequenceName = User.SEQUENCE_NAME, initialValue = 100, allocationSize = 1)
public class User {
    public static final String SEQUENCE_NAME = "USER_SEQ";
    @Id
    @GeneratedValue(strategy = GenerationType.SEQUENCE, generator = SEQUENCE_NAME)
    @Column(name = "USER_SEQ")
    Integer seq;
    String name;
    String password;

    @OneToMany(cascade = CascadeType.ALL)
    @JoinColumn(name = "USER_SEQ", referencedColumnName = "USER_SEQ")
    @OrderColumn
    private AuthComposite[] auths;
}
```

```
user.setAuths(Stream.of(authc1, authc2).peek(it->{it.setUser(user);}).toArray(AuthComposite[]::new));
```



TEST.PUBLIC.AUTHCOMPOSITE				
2 rows				
<Filter criteria>				
	USER_SEQ	AUTH	EXPIRY	AUTHS_ORDER
1	100	ROLE_ADMIN	2016-12-22 10:34:03.689000000	0
2	100	ROLE_USER	2016-12-22 10:34:03.689000000	1

집고 넘어가기 : Array사용할때에는 @OrderColumn으로 선언해야된다.
@OrderColumn은 Index값이 들어가도록 하는것입니다.

Set

```
@Data
@Entity
@SequenceGenerator(name = User.SEQUENCE_NAME, sequenceName = User.SEQUENCE_NAME, initialValue = 100, allocationSize = 1)
public class User {
    public static final String SEQUENCE_NAME = "USER_SEQ";
    @Id
    @GeneratedValue(strategy = GenerationType.SEQUENCE, generator = SEQUENCE_NAME)
    @Column(name = "USER_SEQ")
    Integer seq;
    String name;
    String password;

    @OneToMany(cascade = CascadeType.ALL)
    @JoinColumn(name = "USER_SEQ", referencedColumnName = "USER_SEQ")
    private Set<AuthComposite> auths;
}
```

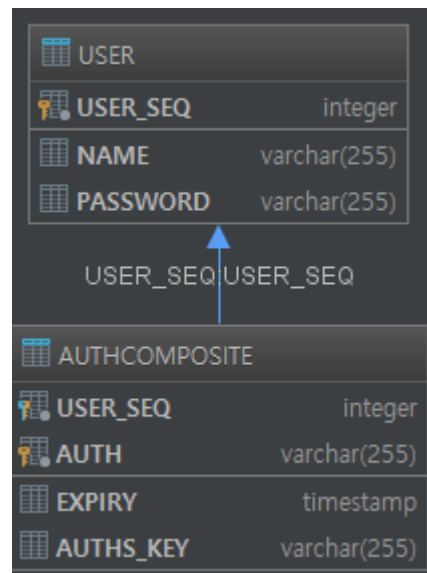
```
user.setAuths(Stream.of(authc1, authc2).peek(it->{it.setUser(user);}).collect(Collectors.toSet()));
```

Map

```
@Data
@Entity
@SequenceGenerator(name = User.SEQUENCE_NAME, sequenceName = User.SEQUENCE_NAME, initialValue = 100, allocationSize = 1)
public class User {
    public static final String SEQUENCE_NAME = "USER_SEQ";
    @Id
    @GeneratedValue(strategy = GenerationType.SEQUENCE, generator = SEQUENCE_NAME)
    @Column(name = "USER_SEQ")
    Integer seq;
    String name;
    String password;

    @OneToMany(cascade = CascadeType.ALL)
    @JoinColumn(name = "USER_SEQ", referencedColumnName = "USER_SEQ")
    private Map<String, AuthComposite> auths;
}
```

```
user.setAuths(Stream.of(authc1, authc2).peek(it->{it.setUser(user);}).collect(Collectors.toMap(c-> c.getAuth(), c->c)));
```



TEST.PUBLIC.AUTHCOMPOSITE				
2 rows				
<Filter criteria>				
	USER_SEQ	AUTH	EXPIRY	AUTHS_KEY
1	100	ROLE_USER	2016-12-22 10:53:45.271000000	ROLE_USER
2	100	ROLE_ADMIN	2016-12-22 10:53:45.271000000	ROLE_ADMIN

▼바로 OneToMany 조인하기(맵핑테이블 없음) (자식테이블 인조식별자 + 유니크 인덱스)

```
@Data
@Entity
public class User {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    Integer seq;
    String name;
    String password;

    @OneToMany(cascade = CascadeType.ALL)
    @JoinColumn
    List<Auth> auths = null;
}
```

```
@Data
@Entity
@SequenceGenerator(name = Auth.SEQUENCE_NAME, sequenceName =
Auth.SEQUENCE_NAME, initialValue = 100)
public class Auth implements Serializable {
    public static final String SEQUENCE_NAME = "AUTH_SEQ";
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    Integer seq;
    String auth;
    Date expiry;
}
```

TEST.PUBLIC.USER x

1 row

<Filter criteria>

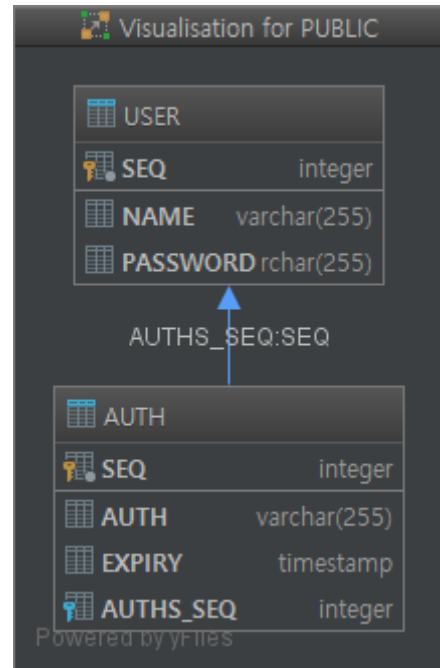
	SEQ	NAME	PASSWORD
1	1	name	pwd

TEST.PUBLIC.AUTH x

2 rows

<Filter criteria>

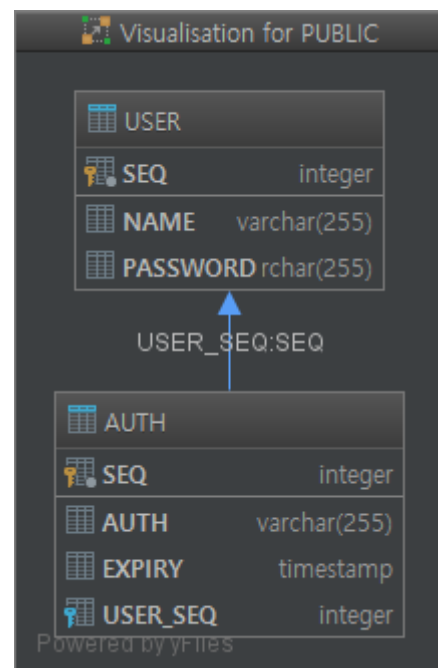
	SEQ	AUTH	EXPIRY	AUTHS_SEQ
1	1	ROLE_ADMIN	2016-12-17 14:58:10.457000000	1
2	2	ROLE_USER	2016-12-17 14:58:10.457000000	1



▼자식테이블 FK 컬럼 명바꾸기

```
Data
@Entity
public class User {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    Integer seq;
    String name;
    String password;

    @OneToMany(cascade = CascadeType.ALL)
    @JoinColumn(name = "USER_SEQ")
    List<Auth> auths = null;
}
```



집고 넘어가기 : OneToOne에서는 자기자신 컬럼을 가르켰지만 OneToMany에서는 자식컬럼을 말한다.

▼PK 를 FK 지정 (맵핑 테이블 없음)

```
@Data
@Entity
@SequenceGenerator(name = User.SEQUENCE_NAME,
sequenceName = User.SEQUENCE_NAME, initialValue = 100)
public class User {
    public static final String SEQUENCE_NAME =
"USER_SEQ";
    @Id
    Integer seq;
    String name;
    String password;

    @OneToMany(cascade = CascadeType.ALL)
    @JoinColumn(name = "USER_SEQ",
referencedColumnName = "SEQ")
    List<Auth> auths = null;
}
```

```
@Data
@Entity
@SequenceGenerator(name = Auth.SEQUENCE_NAME, sequenceName =
Auth.SEQUENCE_NAME, initialValue = 100)
public class Auth implements Serializable {
    public static final String SEQUENCE_NAME = "AUTH_SEQ";
    @Id
    @Column(name = "USER_SEQ")
    Integer seq;
    @Id
    String auth;
    Date expiry;
}
```

▼실행

```
Session session = getSessionFactory().getCurrentSession();
session.beginTransaction();

Auth auth1 = new Auth();
auth1.setAuth("ROLE_ADMIN");
auth1.setExpiry(new Date());
Auth auth2 = new Auth();
auth2.setAuth("ROLE_USER");
auth2.setExpiry(new Date());

User user = new User();
user.setSeq(50);
user.setAuths(Stream.of(auth1, auth2).peek(it -
> it.setSeq(user.getSeq())).collect(Collectors.toList()));
user.setName("name");
user.setPassword("pwd");
session.save(user);
session.getTransaction().commit();
```



집고 넘어가기 : (PK를 FK지정) (맵핑 테이블 없음) 는 좀 문제가 있다. 수동으로 KEY값을 맵핑시켜줘다.

자동으로 맵핑할수 있는 방법도 있다. (아래에서 알아보자)

```
@GenericGenerator(name = "generator", strategy = "foreign", parameters = @Parameter(name = "property", value
= "user"))
```

<http://howtodoinjava.com/hibernate/hibernate-one-to-many-mapping-using-annotations/>

https://en.wikibooks.org/wiki/Java_Persistence/OneToMany

<http://www.beingjavaguys.com/2013/09/hibernate-one-to-many-mapping.html>

<http://levelup.lishman.com/hibernate/associations/one-to-many.php>

▼PK 를 FK 지정 (맵핑 테이블 없음) GeneratedValue 값으로 KEY 값 처리시 문제.

```
@Data
@Entity
@SequenceGenerator(name = User.SEQUENCE_NAME, sequenceName = User.SEQUENCE_NAME, initialValue = 100)
public class User {
    public static final String SEQUENCE_NAME = "USER_SEQ";
    @Id
    @GeneratedValue(strategy = GenerationType.SEQUENCE, generator = SEQUENCE_NAME)
```

```

@Column(name = "USER_SEQ")
Integer seq;
String name;
String password;
@OneToMany(cascade = CascadeType.ALL)
@JoinColumn(name = "USER_SEQ", referencedColumnName = "USER_SEQ")
private List<AuthComposite> auths;
}

```

```

@Data
@Entity
@SequenceGenerator(name = AuthComposite.SEQUENCE_NAME, sequenceName = AuthComposite.SEQUENCE_NAME, initialValue = 50)
public class AuthComposite implements Serializable{
    public static final String SEQUENCE_NAME = "AUTHCOMPOSITE_SEQ";

    @GeneratedValue(strategy = GenerationType.SEQUENCE, generator = SEQUENCE_NAME)
    @Id
    @Column(name = "USER_SEQ", nullable = false)
    Integer seq;
    @Id
    @Column(nullable = false)
    String auth;

    Date expiry;
}

```

```

insert
into
    User
    (name, password, USER_SEQ)
values
    (?, ?, ?)
12:08:37 TRACE : binding parameter [1] as [VARCHAR] - [name]
12:08:37 TRACE : binding parameter [2] as [VARCHAR] - [pwd]
12:08:37 TRACE : binding parameter [3] as [INTEGER] - [100]

```

```

insert
into
    AuthComposite
    (expiry, USER_SEQ, auth)
values
    (?, ?, ?)
12:08:37 TRACE : binding parameter [1] as [TIMESTAMP] - [Sun Dec 18 12:08:36 KST 2016]
12:08:37 TRACE : binding parameter [2] as [INTEGER] - [50]
12:08:37 TRACE : binding parameter [3] as [VARCHAR] - [ROLE_ADMIN]

```

```

insert
into
    AuthComposite
    (expiry, USER_SEQ, auth)
values
    (?, ?, ?)
12:08:37 TRACE : binding parameter [1] as [TIMESTAMP] - [Sun Dec 18 12:08:36 KST 2016]
12:08:37 TRACE : binding parameter [2] as [INTEGER] - [51]
12:08:37 TRACE : binding parameter [3] as [VARCHAR] - [ROLE_USER]

```

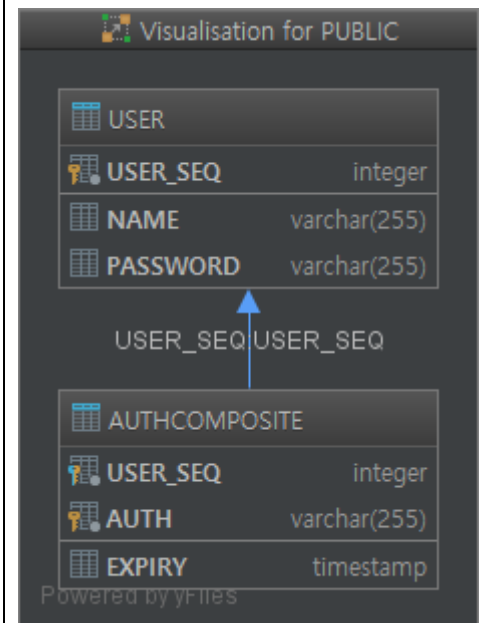
ERROR : HHH000315: Exception executing batch [org.h2.jdbc.JdbcBatchUpdateException: Referential integrity constraint violation: "FKOU1XVFC0SWV0FR46KFD8KHY1N: PUBLIC.AUTHCOMPOSITE FOREIGN KEY(USER_SEQ) REFERENCES PUBLIC.USER(USER_SEQ) (51)"; SQL statement:


```

AuthComposite authc1 = new AuthComposite();
authc1.setAuth("ROLE_ADMIN");
authc1.setExpiry(new Date());
AuthComposite authc2 = new AuthComposite();
authc2.setAuth("ROLE_USER");
authc2.setExpiry(new Date());

User user = new User();
user.setAuths(Stream.of(authc1, authc2).collect(Collectors.toList()));
user.setName("name");
user.setPassword("pwd");
log.debug("---->" + user.getSeq());
session.save(user);

```



집고 넘어가기 : 부모Entity와 자식Entity에 관계가 맺어있지만 각각클래스의 ID값에 GeneratedValue 설정했기때문에 각각 시퀀스 값으로 들어간다 여기서 FK제약조건이 걸려 오류가 난것이다.

(부모는 ID에만 GeneratedValue 사용 하고 자식테이블은 사용하지 말어라.)

save 할때 어떻게 해야될까?

1. 이렇게 해보자

부모Entity는 ID는 GeneratedValue 자식Entity은 GeneratedValue사용하지 않고 자동으로 넘어가도록 기대해보자

```

@Data
@Entity
public class AuthComposite implements Serializable{
    @Id
    @Column(name = "USER_SEQ", nullable = false)
    Integer seq;
    @Id
    @Column(nullable = false)
    String auth;
    Date expiry;
}

```

```

insert
into
    User
    (name, password, USER_SEQ)
values
    (?, ?, ?)

```

```

parameter [1] as [VARCHAR] - [name]
parameter [2] as [VARCHAR] - [pwd]
parameter [3] as [INTEGER] - [100]

```

```

insert
into
    AuthComposite
    (expiry, USER_SEQ, auth)
values
    (?, ?, ?)

```

```

parameter [1] as [TIMESTAMP] - [Sun Dec 18 13:10:36 KST 2016]
parameter [2] as [INTEGER] - [null]
parameter [3] as [VARCHAR] - [ROLE_ADMIN]

```

```

insert
into
    AuthComposite
    (expiry, USER_SEQ, auth)
values
    (?, ?, ?)

```

```

parameter [1] as [TIMESTAMP] - [Sun Dec 18 13:10:36 KST 2016]
parameter [2] as [INTEGER] - [null]
parameter [3] as [VARCHAR] - [ROLE_USER]

```

기대는 했지만 자동으로 넘어가지 않는다. 그저 NULL값이 들어간다 정직하다.. 오류난다

2. 이렇게 해보자 (나눠서 save하자)

부모Entity만 save 선행후~ 부모Entity에서 추출된 ID (SEQUENCE) 값을 가지고

자식Entity ID값으로 지정 그뒤 자식Entity를 save해보자

```
AuthComposite authc1 = new AuthComposite();
authc1.setAuth("ROLE_ADMIN");
authc1.setExpiry(new Date());
AuthComposite authc2 = new AuthComposite();
authc2.setAuth("ROLE_USER");
authc2.setExpiry(new Date());
User user = new User();
user.setAuths(Stream.of(authc1,authc2).collect(Collectors.toList()));
user.setName("name");
user.setPassword("pwd");
log.debug("---->" + user.getSeq());
session.save(user);
log.debug("---->" + user.getSeq());
user.getAuths().forEach(it->it.setSeq(user.getSeq()));
```

---->null

call next value for USER_SEQ

Sequence value obtained: 100

HHH000387: ResultSet's statement was not registered

Generated identifier: 100, using strategy: org.hibernate.id.enhanced.SequenceStyleGenerator

---->100

```
insert
  into
    User
    (name, password, USER_SEQ)
  values
    (?, ?, ?)
parameter [1] as [VARCHAR] - [name]
parameter [2] as [VARCHAR] - [pwd]
parameter [3] as [INTEGER] - [100]
```

```
insert
  into
    AuthComposite
    (expiry, USER_SEQ, auth)
  values
    (?, ?, ?)
parameter [1] as [TIMESTAMP] - [Sun Dec 18 13:21:49 KST 2016]
parameter [2] as [INTEGER] - [100]
parameter [3] as [VARCHAR] - [ROLE_ADMIN]
```

```
insert
  into
    AuthComposite
    (expiry, USER_SEQ, auth)
  values
    (?, ?, ?)
parameter [1] as [TIMESTAMP] - [Sun Dec 18 13:21:49 KST 2016]
parameter [2] as [INTEGER] - [100]
parameter [3] as [VARCHAR] - [ROLE_USER]
```

집고 넘어가기 1 : 부모Entity만 save후 자식Entity List를 넣은뒤 다시 save했다

여기서 알수 있듯 save 를 타게되면 바로 시퀀스 값을 가져와 바인딩 시켜주니 그것을 가지고 자식 셋팅후 commit을 하면 그때 가서 자식 Entity에 바인딩된 ID값으로 insert된다.

집고 넘어가기 2 : PK를 PK복합키로만 구성하는 방법과, 인조식별자+유니크 인덱스로 구성하는것은 아직도 많은 논쟁있다. 하지만 어느정도 대세는 인조식별자+유니크 를 두는것으로 많이 기울었습니다. <http://okky.kr/article/257331>

3. 이렇게 해보자 (자식Entity에 ID값을 부모Entity의 ID값으로 자동 save하자) 베스트!

```
@Data
@Entity
@SequenceGenerator(name = User.SEQUENCE_NAME,
sequenceName = User.SEQUENCE_NAME, initialValue = 100,
allocationSize = 1)
public class User {
    public static final String SEQUENCE_NAME =
"USER_SEQ";
    @Id
    @GeneratedValue(strategy = GenerationType.SEQUENCE,
generator = SEQUENCE_NAME)
    @Column(name = "USER_SEQ")
    Integer seq;
    String name;
    String password;

    @OneToMany(cascade = CascadeType.ALL)
    @JoinColumn(name = "USER_SEQ", referencedColumnName =
"USER_SEQ")
    private List<AuthComposite> auths;
}
```

```
@Data
@Entity
@GenericGenerator(name = "generator", strategy = "foreign",
parameters = @Parameter(name = "property", value = "user"))
public class AuthComposite implements Serializable{
    @Id
    @Column(name = "USER_SEQ", nullable = false)
    @GeneratedValue(generator = "generator")
    Integer seq;
    @Id
    @Column(nullable = false)
    String auth;

    Date expiry;

    @ManyToOne(cascade = CascadeType.ALL)
    @JoinColumn(name="USER_SEQ",
null table=false,updatable=false,insert table=false)
    User user;
}
```

```
AuthComposite authc1 = new AuthComposite();
authc1.setAuth("ROLE_ADMIN");
authc1.setExpiry(new Date());
AuthComposite authc2 = new AuthComposite();
authc2.setAuth("ROLE_USER");
authc2.setExpiry(new Date());

User user = new User();
user.setAuths(Stream.of(authc1,authc2).peek(it->{it.setUser(user);}).collect(Collectors.toList()));
user.setName("name");
user.setPassword("pwd");
Serializable id = session.save(user);
```

```
insert
into
    User
    (name, password, USER_SEQ)
values
    (?, ?, ?)
10:44:27 TRACE : binding parameter [1] as [VARCHAR] - [name]
10:44:27 TRACE : binding parameter [2] as [VARCHAR] - [pwd]
10:44:27 TRACE : binding parameter [3] as [INTEGER] - [100]
10:44:27 DEBUG : Executing batch size: 1
10:44:27 DEBUG :
insert
into
    AuthComposite
    (expiry, USER_SEQ, auth)
values
    (?, ?, ?)
10:44:27 TRACE : binding parameter [1] as [TIMESTAMP] - [Wed Dec 21 10:44:27 KST 2016]
```

```

10:44:27 TRACE : binding parameter [2] as [INTEGER] - [100]
10:44:27 TRACE : binding parameter [3] as [VARCHAR] - [ROLE_ADMIN]
10:44:27 DEBUG : Reusing batch statement
10:44:27 DEBUG :
    insert
    into
        AuthComposite
        (expiry, USER_SEQ, auth)
    values
        (?, ?, ?)
10:44:27 TRACE : binding parameter [1] as [TIMESTAMP] - [Wed Dec 21 10:44:27 KST 2016]
10:44:27 TRACE : binding parameter [2] as [INTEGER] - [100]
10:44:27 TRACE : binding parameter [3] as [VARCHAR] - [ROLE_USER]

```

객체간 관계만 맺어주면 (변수 값셋팅)

자동으로 부모Entity값을 가지고 자기자신ID값으로 사용하는걸 알수 있다.

<https://www.mkyong.com/hibernate/hibernate-one-to-one-relationship-example-annotation/>

<http://www.codejava.net/frameworks/hibernate/hibernate-one-to-one-association-on-primary-key-annotations-example>

방향성 갖자

잡고 넘어가기 1 : @ManyToOne 쪽에 @JoinColumn 를 사용하여 자기자신 테이블 컬럼명을 지정할수도있다.

```

@ManyToOne(cascade = CascadeType.ALL)
@JoinColumn(name="USER_SEQ", nullable=false,updatable=false,insertable=false)
User user;

```

조회하기

지연로딩 FetchType.LAZY (기본값)

```

@Data
@Entity
@SequenceGenerator(name = User.SEQUENCE_NAME, sequenceName = User.SEQUENCE_NAME, initialValue = 100)
public class User {
    public static final String SEQUENCE_NAME = "USER_SEQ";
    @Id
    @GeneratedValue(strategy = GenerationType.SEQUENCE, generator = SEQUENCE_NAME)
    Integer seq;
    String name;
    String password;

    @OneToMany(cascade = CascadeType.ALL, fetch = FetchType.LAZY)
    @JoinColumn(name = "USER_SEQ")
    List<Auth> auths = null;
}

```

```

Session session = getSessionFactory().getCurrentSession();
session.beginTransaction();
/////Fetch
User userBydb = session.get(User.class,100); //부모테이블만 조회
try {Thread.sleep(5000);} catch (InterruptedException e) {e.printStackTrace();}
//아래에서 사용할때 자식테이블 조회
userBydb.getAuths().forEach(it->log.info("get(DB)+"it.getUser().getName()));

session.getTransaction().commit();

```

```
18:04:14 DEBUG :
select
    user0_.seq as seq1_1_0_,
    user0_.name as name2_1_0_,
    user0_.password as password3_1_0_
from
    User user0_
where
    user0_.seq=?
18:04:14 TRACE : binding parameter [1] as [INTEGER] - [100]
```

```
18:04:19 DEBUG :
select
    auths0_.USER_SEQ as USER_SEQ4_0_0_,
    auths0_.AUTH_SEQ as AUTH_SEQ1_0_0_,
    auths0_.auth as auth2_0_0_,
    auths0_.AUTH_SEQ as AUTH_SEQ1_0_1_,
    auths0_.auth as auth2_0_1_,
    auths0_.expiry as expiry3_0_1_,
    auths0_.user_seq as user_seq4_0_1_,
    user1_.seq as seq1_1_2_,
    user1_.name as name2_1_2_,
    user1_.password as password3_1_2_
from
    Auth auths0_
left outer join
    User user1_
        on auths0_.user_seq=user1_.seq
where
    auths0_.USER_SEQ=?
18:04:19 TRACE : binding parameter [1] as [INTEGER] - [100]
```

즉시로딩 FetchType.EAGER

```
@Data
@Entity
@SequenceGenerator(name = User.SEQUENCE_NAME, sequenceName = User.SEQUENCE_NAME, initialValue = 100)
public class User {
    public static final String SEQUENCE_NAME = "USER_SEQ";
    @Id
    @GeneratedValue(strategy = GenerationType.SEQUENCE, generator = SEQUENCE_NAME)
    Integer seq;
    String name;
    String password;

    @OneToMany(cascade = CascadeType.ALL, fetch = FetchType.EAGER)
    @JoinColumn(name = "USER_SEQ")
    List<Auth> auths = null;
}
```

즉시 조인 쿼리가 날라간다.

```
select
    user0_.seq as seq1_1_0_,
    user0_.name as name2_1_0_,
    user0_.password as password3_1_0_,
```

```
    auths1_.USER_SEQ as USER_SEQ4_0_1_,
    auths1_.AUTH_SEQ as AUTH_SEQ1_0_1_,
    auths1_.auth as auth2_0_1_,
    auths1_.AUTH_SEQ as AUTH_SEQ1_0_2_,
    auths1_.auth as auth2_0_2_,
    auths1_.expiry as expiry3_0_2_,
    auths1_.user_seq as user_seq4_0_2_,
    user2_.seq as seq1_1_3_,
    user2_.name as name2_1_3_,
    user2_.password as password3_1_3_
from
    User user0_
left outer join
    Auth auths1_
        on user0_.seq=auths1_.USER_SEQ
left outer join
    User user2_
        on auths1_.user_seq=user2_.seq
where
    user0_.seq=?
18:09:42 TRACE : binding parameter [1] as [INTEGER] - [100]
```

@ManyToMany

```
@Data
@Entity
@SequenceGenerator(name = User.SEQUENCE_NAME,
sequenceName = User.SEQUENCE_NAME, initialValue = 100)
public class User {
    public static final String SEQUENCE_NAME =
"USER_SEQ";
    @Id
    @GeneratedValue(strategy =
GenerationType.SEQUENCE, generator = SEQUENCE_NAME)
    @Column(name = "USER_SEQ")
    Integer seq;
    String name;

    @ManyToMany(cascade = CascadeType.ALL)
    @JoinColumn(name = "USER_SEQ",
referencedColumnName = "SHOP_SEQ", nullable = false)
    List<CoffeShop> shops = null;
}
```

```
@Data
@Entity
@IdClass(CoffeShop.class)
@SequenceGenerator(name = CoffeShop.SEQUENCE_NAME,
sequenceName = CoffeShop.SEQUENCE_NAME, initialValue = 100)
public class CoffeShop implements Serializable {
    public static final String SEQUENCE_NAME = "SHOP_SEQ";
    @Id
    @Column(name = "SHOP_SEQ")
    @GeneratedValue(strategy = GenerationType.SEQUENCE,
generator = SEQUENCE_NAME)
    Integer seq;

    String number;
    Date open;
    Date close;

    @ManyToMany(cascade = CascadeType.ALL, mappedBy = "shops")
    List<User> users = null;
}
```

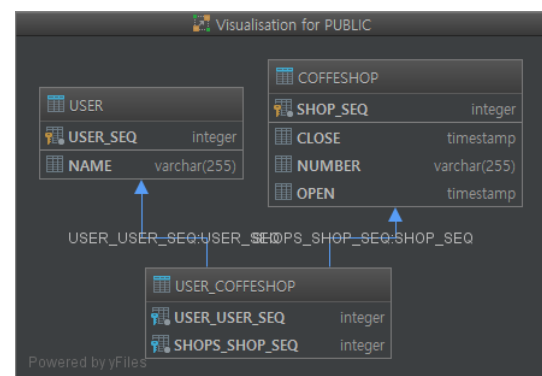
```
Session session =
getSessionFactory().getCurrentSession();
session.beginTransaction();

User user1 = new User();
user1.setName("name1");
User user2 = new User();
user2.setName("name2");

CoffeShop shop1 = new CoffeShop();
shop1.setName("shop1");
CoffeShop shop2 = new CoffeShop();
shop2.setName("shop2");

user1.setShops(Arrays.asList(shop1,shop2));
user2.setShops(Arrays.asList(shop1,shop2));

session.save(user1);
session.save(user2);
session.getTransaction().commit();
```



TEST.PUBLIC.COFFESHOP

2 rows Tab-se...d (TSV) View Query

<Filter criteria>

	SHOP_SEQ	CLOSE	NAME	OPEN
1	100	<null>	shop1	<null>
2	101	<null>	shop2	<null>

TEST.PUBLIC.USER_COFFESHOP

Tab-se...d (TSV) View Query

<Filter criteria>

	USERS_U...	SHOPS_S...
1	100	100
2	100	101
3	101	100
4	101	101

TEST.PUBLIC.USER

2 rows View Query

<Filter criteria>

	USER_SEQ	NAME
1	100	name1
2	101	name2

```
insert
into
    User
    (name, USER_SEQ)
values
```

(?, ?)

15:53:09 TRACE : binding parameter [1] as [VARCHAR] - [name1]

15:53:09 TRACE : binding parameter [2] as [INTEGER] - [100]

```
insert
into
    CoffeShop
    (close, name, open, SHOP_SEQ)
```

```
values
    (?, ?, ?, ?)
```

15:53:09 TRACE : binding parameter [1] as [TIMESTAMP] - [null]

15:53:09 TRACE : binding parameter [2] as [VARCHAR] - [shop1]

15:53:09 TRACE : binding parameter [3] as [TIMESTAMP] - [null]

15:53:09 TRACE : binding parameter [4] as [INTEGER] - [100]

```
insert
into
    CoffeShop
    (close, name, open, SHOP_SEQ)
```

```
values
    (?, ?, ?, ?)
```

15:53:09 TRACE : binding parameter [1] as [TIMESTAMP] - [null]

15:53:09 TRACE : binding parameter [2] as [VARCHAR] - [shop2]

15:53:09 TRACE : binding parameter [3] as [TIMESTAMP] - [null]

15:53:09 TRACE : binding parameter [4] as [INTEGER] - [101]

```
insert
into
    User
    (name, USER_SEQ)
```

```
values
    (?, ?)
```

15:53:09 TRACE : binding parameter [1] as [VARCHAR] - [name2]

15:53:09 TRACE : binding parameter [2] as [INTEGER] - [101]

15:53:09 DEBUG : Executing batch size: 1

```
insert
into
    User_CoffeShop
    (users_USER_SEQ, shops_SHOP_SEQ)
```

```
values
    (?, ?)
```

15:53:09 TRACE : binding parameter [1] as [INTEGER] - [100]

15:53:09 TRACE : binding parameter [2] as [INTEGER] - [100]

```
insert
into
    User_CoffeShop
    (users_USER_SEQ, shops_SHOP_SEQ)
```

```
values
    (?, ?)
```

15:53:09 TRACE : binding parameter [1] as [INTEGER] - [100]

15:53:09 TRACE : binding parameter [2] as [INTEGER] - [101]

15:53:09 DEBUG : Done inserting collection: 2 rows inserted

15:53:09 DEBUG : Inserting collection: [com.khh.hibernate.c2.join.manytomany.entity.User.shops#101]

```
insert
into
```



```
User_CoffeShop
(users_USER_SEQ, shops_SHOP_SEQ)
values
(?, ?)
15:53:09 TRACE : binding parameter [1] as [INTEGER] - [101]
15:53:09 TRACE : binding parameter [2] as [INTEGER] - [100]
```

```
insert
into
User_CoffeShop
(users_USER_SEQ, shops_SHOP_SEQ)
values
(?, ?)
15:53:09 TRACE : binding parameter [1] as [INTEGER] - [101]
15:53:09 TRACE : binding parameter [2] as [INTEGER] - [101]
```

```
CoffeShop shopBydb = session.get(CoffeShop.class, 100);
shopBydb.getUsers().forEach(it->log.debug("-->" + it.getName()));
session.getTransaction().commit();
```

```
select
users0_.shops_SHOP_SEQ as shops_SH2_2_0_,
users0_.users_USER_SEQ as users_US1_2_0_,
user1_.USER_SEQ as USER_SEQ1_1_1_,
user1_.name as name2_1_1_
from
User_CoffeShop users0_
inner join
User user1_
on users0_.users_USER_SEQ=user1_.USER_SEQ
where
users0_.shops_SHOP_SEQ=?
15:59:43 TRACE : binding parameter [1] as [INTEGER] - [100]
15:59:43 DEBUG : -->name1
15:59:43 DEBUG : -->name2
```

N:N 연결하기

```
@Data
@Entity
@SequenceGenerator(name = User.SEQUENCE_NAME, sequenceName = User.SEQUENCE_NAME, initialValue = 100)
public class User {
    public static final String SEQUENCE_NAME = "USER_SEQ";
    @Id
    @GeneratedValue(strategy = GenerationType.SEQUENCE, generator = SEQUENCE_NAME)
    @Column(name = "USER_SEQ")
    Integer seq;
    String name;
    @OneToMany(cascade = CascadeType.ALL, mappedBy = "user")
    List<UserShop> shops = null;
}
```

```
@Entity
@SequenceGenerator(name = CoffeShop.SEQUENCE_NAME, sequenceName = CoffeShop.SEQUENCE_NAME, initialValue = 100)
public class CoffeShop implements Serializable {
    public static final String SEQUENCE_NAME = "SHOP_SEQ";
    @Id
    @Column(name = "SHOP_SEQ")
    @GeneratedValue(strategy = GenerationType.SEQUENCE, generator = SEQUENCE_NAME)
    Integer seq;
    String name;
    Date open;
    Date close;
    @OneToMany(cascade = CascadeType.ALL, mappedBy = "shop")
    List<UserShop> shops = null;
}
```

```
@Data
public class UserShopPK implements Serializable {
    @Column(name = "USER_SEQ")
    Integer user_seq;
    @Column(name = "SHOP_SEQ")
    Integer shop_seq;
    public UserShopPK() {}

    public UserShopPK(Integer user_seq, Integer shop_seq){
        this.user_seq = user_seq;
        this.shop_seq = shop_seq;
    }
}
```

```

@Data
@Entity
@IdClass(UserShopPK.class)
public class UserShop{
    @Id
    Integer user_seq;
    @Id
    Integer shop_seq;

    @ManyToOne(cascade = CascadeType.ALL)
    @JoinColumn(name="USER_SEQ", referencedColumnName = "USER_SEQ", insertable = false, updatable = false)
    private User user;

    @ManyToOne(cascade = CascadeType.ALL)
    @JoinColumn(name="SHOP_SEQ",referencedColumnName = "SHOP_SEQ", insertable = false, updatable = false)
    private CoffeShop shop;

    @Temporal(TemporalType.TIMESTAMP)
    // @ColumnDefault("now()")
    @Version
    Date reg;
}

```

```

create table CoffeShop (
    SHOP_SEQ integer not null,
    close timestamp,
    name varchar(255),
    open timestamp,
    primary key (SHOP_SEQ)
)

```

```

create table User (
    USER_SEQ integer not null,
    name varchar(255),
    primary key (USER_SEQ)
)

```

```

create table UserShop (
    SHOP_SEQ integer not null,
    USER_SEQ integer not null,
    reg timestamp,
    primary key (SHOP_SEQ, USER_SEQ)
)

```

```

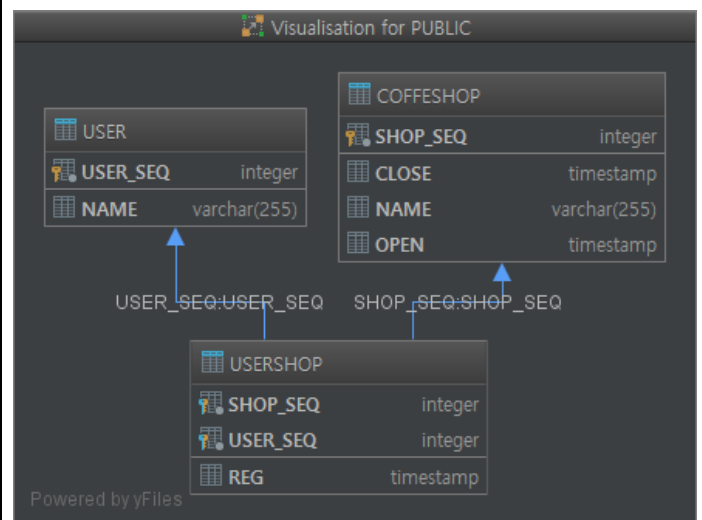
alter table UserShop
    add constraint FKxjfywgc7nyi1lfdq0elefqx
    foreign key (SHOP_SEQ)
    references CoffeShop

```

```

alter table UserShop
    add constraint FK7t3g73sa7nsvtdikuq8lon55
    foreign key (USER_SEQ)
    references User

```



▼실행

```

Session session = getSessionFactory().getCurrentSession();
    session.beginTransaction();

    User user1 = new User();
    user1.setName("name1");
    User user2 = new User();
    user2.setName("name2");

    CoffeShop shop1 = new CoffeShop();
    shop1.setName("shop1");
    CoffeShop shop2 = new CoffeShop();
    shop2.setName("shop2");

    session.save(user1);
    session.save(user2);
    session.save(shop1);
    session.save(shop2);

    UserShop userShop = new UserShop();
    userShop.setUser_seq(user1.getSeq());
    userShop.setShop_seq(shop1.getSeq());
    session.save(userShop);

    userShop = new UserShop();
    userShop.setUser_seq(user1.getSeq());
    userShop.setShop_seq(shop2.getSeq());
    session.save(userShop);

    session.flush();
    session.clear();

    UserShop byDB = session.get(UserShop.class, new UserShopPK(100, 101));
    log.debug(byDB.getUser().getName()+"<—>" + byDB.getShop().getName());

    session.getTransaction().commit();

```

```

insert
into
    User
    (name, USER_SEQ)
values
    (?, ?)
16:55:03 TRACE : binding parameter [1] as [VARCHAR] - [name1]
16:55:03 TRACE : binding parameter [2] as [INTEGER] - [100]
16:55:03 DEBUG : Reusing batch statement
16:55:03 DEBUG :
    insert
    into
        User
        (name, USER_SEQ)
    values
        (?, ?)
16:55:03 TRACE : binding parameter [1] as [VARCHAR] - [name2]
16:55:03 TRACE : binding parameter [2] as [INTEGER] - [101]
16:55:03 DEBUG : Executing batch size: 2
16:55:03 DEBUG :
    insert
    into
        CoffeShop
        (close, name, open, SHOP_SEQ)

```

```

values
    (?, ?, ?, ?)
16:55:03 TRACE : binding parameter [1] as [TIMESTAMP] - [null]
16:55:03 TRACE : binding parameter [2] as [VARCHAR] - [shop1]
16:55:03 TRACE : binding parameter [3] as [TIMESTAMP] - [null]
16:55:03 TRACE : binding parameter [4] as [INTEGER] - [100]
16:55:03 DEBUG : Reusing batch statement
16:55:03 DEBUG :
    insert
    into
        CoffeShop
        (close, name, open, SHOP_SEQ)
values
    (?, ?, ?, ?)
16:55:03 TRACE : binding parameter [1] as [TIMESTAMP] - [null]
16:55:03 TRACE : binding parameter [2] as [VARCHAR] - [shop2]
16:55:03 TRACE : binding parameter [3] as [TIMESTAMP] - [null]
16:55:03 TRACE : binding parameter [4] as [INTEGER] - [101]
16:55:03 DEBUG : Executing batch size: 2
16:55:03 DEBUG :
    insert
    into
        UserShop
        (reg, SHOP_SEQ, USER_SEQ)
values
    (?, ?, ?)
16:55:03 TRACE : binding parameter [1] as [TIMESTAMP] - [2016-12-18 16:55:03.706]
16:55:03 TRACE : binding parameter [2] as [INTEGER] - [100]
16:55:03 TRACE : binding parameter [3] as [INTEGER] - [100]
16:55:03 DEBUG : Reusing batch statement
16:55:03 DEBUG :
    insert
    into
        UserShop
        (reg, SHOP_SEQ, USER_SEQ)
values
    (?, ?, ?)
16:55:03 TRACE : binding parameter [1] as [TIMESTAMP] - [2016-12-18 16:55:03.73]
16:55:03 TRACE : binding parameter [2] as [INTEGER] - [101]
16:55:03 TRACE : binding parameter [3] as [INTEGER] - [100]
16:55:03 DEBUG : Executing batch size: 2
16:55:03 DEBUG :
    select
        usershop0_.SHOP_SEQ as SHOP_SEQ1_2_0_,
        usershop0_.USER_SEQ as USER_SEQ2_2_0_,
        usershop0_.reg as reg3_2_0_,
        coffeshop1_.SHOP_SEQ as SHOP_SEQ1_0_1_,
        coffeshop1_.close as close2_0_1_,
        coffeshop1_.name as name3_0_1_,
        coffeshop1_.open as open4_0_1_,
        user2_.USER_SEQ as USER_SEQ1_1_2_,
        user2_.name as name2_1_2_
    from
        UserShop usershop0_
    left outer join
        CoffeShop coffeshop1_
            on usershop0_.SHOP_SEQ=coffeshop1_.SHOP_SEQ
    left outer join
        User user2_

```

```

        on usershop0_.USER_SEQ=user2_.USER_SEQ
    where
        usershop0_.SHOP_SEQ=?
        and usershop0_.USER_SEQ=?
16:55:03 TRACE : binding parameter [1] as [INTEGER] - [101]
16:55:03 TRACE : binding parameter [2] as [INTEGER] - [100]

```

집고 넘어가기 : 여기서 알수 있듯. ManyToOne쪽에서
 @JoinColumn(name="SHOP_SEQ",referencedColumnName = "SHOP_SEQ", insertable = false, updatable = false)
 name은 자기자신 테이블의 컬럼명이다 (FK)
 관계테이블은 insert와 update가 이루어지면 안되므로 false를 주었다

▼ 실행

```
User userBydb = session.get(User.class,100);
```

```

select
    user0_.USER_SEQ as USER_SEQ1_1_0_,
    user0_.name as name2_1_0_
from
    User user0_
where
    user0_.USER_SEQ=?
09:55:20 TRACE : binding parameter [1] as [INTEGER] - [100]

```

```

select
    shops0_.USER_SEQ as USER_SEQ2_2_0_,
    shops0_.SHOP_SEQ as SHOP_SEQ1_2_0_,
    shops0_.SHOP_SEQ as SHOP_SEQ1_2_1_,
    shops0_.USER_SEQ as USER_SEQ2_2_1_,
    shops0_.reg as reg3_2_1_,
    coffeshop1_.SHOP_SEQ as SHOP_SEQ1_0_2_,
    coffeshop1_.close as close2_0_2_,
    coffeshop1_.name as name3_0_2_,
    coffeshop1_.open as open4_0_2_
from
    UserShop shops0_
left outer join
    COFFESHOP coffeshop1_
        on shops0_.SHOP_SEQ=coffeshop1_.SHOP_SEQ
where
    shops0_.USER_SEQ=?
09:55:20 TRACE : binding parameter [1] as [INTEGER] - [100]

```

캐싱

1 차 캐시

Session 객체와 관련하여 트랜잭션이 보장되는 캐시다. 이는 세션의 수명이 유지되는 동안이나 컨버세이션 conversation 내에서만 가능하다. 1 차 캐시는 하이버네이트 프레임워크에서 기본으로 제공한다.

```
//팩토리 가져오기
SessionFactory factory = d.getSessionFactory();

//세션가져오기
Session session = factory.getCurrentSession();
//트랜잭션 시작
session.beginTransaction();

User user = new User("admin","admin_pwd");
Set<Auth> auths = new HashSet<Auth>();
auths.add(new Auth("fName1","fCode","fVal"));
auths.add(new Auth("fName2","fCode","fVal"));
auths.add(new Auth("fName3","fCode","fVal"));
user.setAuths(auths);
session.save(user);

user = new User("user","user_pwd");
auths = new HashSet<Auth>();
auths.add(new Auth("fName1_1","fCode","fVal"));
auths.add(new Auth("fName1_2","fCode","fVal"));
auths.add(new Auth("fName1_3","fCode","fVal"));
user.setAuths(auths);
session.save(user);
//1차 캐쉬부분 읽어오기
User load_user = (User) session.load(User.class,new Integer(50));
System.out.println(load_user);
load_user.setId("admin_after");
session.save(user);
//트랜잭션 커밋
session.getTransaction().commit();

//트랜잭션이 닫혔기때문에 아래 부분은 오류남
User after_load_user = (User) session.load(User.class,new Integer(50));
System.out.println(after_load_user);
```

```
Hibernate: drop table T_AUTH if exists
Hibernate: drop table T_USER if exists
Hibernate: drop table T_USER_PRIVACY if exists
Hibernate: drop sequence SEQ_USER
Hibernate: create table T_AUTH (FNC_ID integer generated by default as identity (start with 1), fncCode varchar(255), fncName varchar(255), fncValue varchar(255), user_SEQ integer, primary key (FNC_ID))
Hibernate: create table T_USER (SEQ integer not null, ID varchar(255), PWD varchar(255), primary key (SEQ))
Hibernate: create table T_USER_PRIVACY (SEQ integer generated by default as identity (start with 1), AGE integer, NAME varchar(255), primary key (SEQ))
Hibernate: alter table T_AUTH add constraint FK_5wrx276k6vlwxjehogh6h1rwa foreign key (user_SEQ) references T_USER
Hibernate: create sequence SEQ_USER start with 1
Hibernate: call next value for SEQ_USER
Hibernate: insert into T_USER (ID, PWD, SEQ) values (?, ?, ?)
Hibernate: insert into T_AUTH (FNC_ID, fncCode, fncName, fncValue, user_SEQ) values (default, ?, ?, ?, ?)
```

```

Hibernate: insert into T_AUTH (FNC_ID, fncCode, fncName, fncValue, user_SEQ) values
(default, ?, ?, ?, ?)
Hibernate: insert into T_AUTH (FNC_ID, fncCode, fncName, fncValue, user_SEQ) values
(default, ?, ?, ?, ?)
Hibernate: insert into T_USER (ID, PWD, SEQ) values (?, ?, ?)
Hibernate: insert into T_AUTH (FNC_ID, fncCode, fncName, fncValue, user_SEQ) values
(default, ?, ?, ?, ?)
Hibernate: insert into T_AUTH (FNC_ID, fncCode, fncName, fncValue, user_SEQ) values
(default, ?, ?, ?, ?)
Hibernate: insert into T_AUTH (FNC_ID, fncCode, fncName, fncValue, user_SEQ) values
(default, ?, ?, ?, ?)
User(seq=50, id=admin, password=admin_pwd, auths=[Auth(fncId=1, fncName=fName2, fncCode=fCode,
fncValue=fVal, user=null), Auth(fncId=2, fncName=fName3, fncCode=fCode, fncValue=fVal, user=null),
Auth(fncId=3, fncName=fName1, fncCode=fCode, fncValue=fVal, user=null)])
Hibernate: update T_USER set ID=?, PWD=? where SEQ=?
Hibernate: update T_AUTH set USER_SEQ=? where FNC_ID=?
Hibernate: update T_AUTH set USER_SEQ=? where FNC_ID=?
Hibernate: update T_AUTH set USER_SEQ=? where FNC_ID=?
Hibernate: update T_AUTH set USER_SEQ=? where FNC_ID=?
Hibernate: update T_AUTH set USER_SEQ=? where FNC_ID=?
Exception in thread "main" org.hibernate.SessionException: Session is closed!

```

두 번째로 User 객체를 불러올 때(load) 세션 자체가 가진 캐시에서 해당 객체를 조회한다. 이렇게 데이터베이스를 이용하는 네트워 크 라운드트립roundtrip을 피한다. 세션 캐시는 클래스 타입과 함께 명시되므로 이미 존재하는 인스턴스를 오버라이드할 때 좀 더 주의해야 한다.

2 차 캐시

SessionFactory 클래스를 이용하여 전역에서 사용할 수 있다.

그래서 2 차 캐시 안에 있는 어떤 데이터라도 애플리케이션 전체에서 사용이 가능하다.

하이버네이트는 EhCache 와 InfiniSpan 같은 오픈소스 캐시 라이브러리를 지원한다.

사용자 정의 캐시 라이브러리를 사용하려면 org.hibernate.cache.spi.CacheProvider 인터페이스

관련 라이브러리를 구현하면 된다. 하이버네이트는 기본 옵션으로 EhCache 를 2 차 캐시 공급자로 사용한다.

캐시 공급자를 연결하려면 hibernate.cache.provider 클래스 속성에 캐시 공급자를 명시한다.

다음은 JBoss 의 InifiniSpan 을 캐시 공급자로 연결하는 예다.

```

<hibernate-configuration>
  <session-factory>
    ....
    <!-- Infinispan 캐시 공급자 지정 -->
    <property name="hibernate.cache.provider_class">
      org.hibernate.cache.infinispan.InfinispanRegionFactory
    </property>
  </session-factory>
</hibernate-configuration>

```

다양한 캐시 속성을 이용하여 캐싱할 클래스별로 캐싱 정책을 지정할 수 있다. 매핑 정의 파일의 class 태그 안에 cache 속성을 확인해 보자.

cache usage 속성

transactional	이 전략은 트랜잭션이 가능한 캐시를 지원하는 캐시 공급자를 위해 제공된다. 모든 캐시 공급자가 트랜잭션이 가능한 캐싱 상품을 가지고 있지 않다는 점에 유의하자.
---------------	---

read-only	갱신할 필요가 없는 영속화 객체에 자주 접근한다면 read-only 를 선택한다. 데이터베이스를 거의 변경하지 않거나 전혀 변경하지 않는다면 이 옵션으로 성능이 크게 향상할 것이다.
read-write	객체를 데이터베이스에서 읽거나 데이터베이스에 쓸 때 이 방법을 사용한다.
nonstrict-read-write	객체를 그다지 자주 갱신하지 않을 때 사용한다.
이 옵션을 전역에서 사용하려면 설정 파일에서 hibernate.cache.default_cache_concurrency_strategy 속성을 설정한다.	

쿼리 캐시

객체 뿐만 아니라 쿼리도 캐싱할 수도 있다. 특정 쿼리를 빈번하게 사용한다면 캐싱하는 것을 추천한다.

이 기능을 사용하려면 hibernate.cache.use_query_cache 속성을 true 로 지정한다.

코드에 한 가지를 더 추가해야 하는데, Query.setCacheable() 메소드를 호출해서 Query의 cacheable 속성을 true 로 지정해야한다.

상속 전략

Entity 상속으로 처리하는 방법이 3가지 있다.

1. Table-per-Class 전략
2. Table-per-Subclass 전략
3. Table-per-Concrete-Class 전략

@Table-per-Class

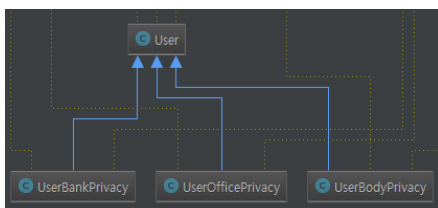
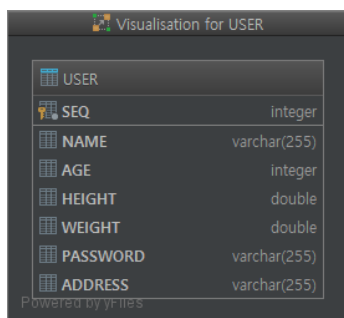
기본값 @Inheritance(strategy=InheritanceType.SINGLE_TABLE)

```
@Data
@Entity
//@Inheritance(strategy=InheritanceType.SINGLE_TABLE)
@SequenceGenerator(name = User.SEQ_NAME, sequenceName = User.SEQ_NAME, initialValue = 50, allocationSize = 1)
public class User {
    public static final String SEQ_NAME = "SEQ_USER";
    @Id
    @GeneratedValue(strategy = GenerationType.SEQUENCE, generator = SEQ_NAME)
    Integer seq;
    String name;
}
```

```
@Data
@Entity
public class UserBankPrivacy extends User {
    String password;
}
```

```
@Data
@Entity
public class UserBodyPrivacy
extends User {
    Integer age;
    double weight;
    double height;
}
```

```
@Data
@Entity
public class UserOfficePrivacy
extends User {
    String address;
}
```



```
create table USER (
    seq integer not null,
    name varchar(255),
    age integer,
    height double,
    weight double,
    password varchar(255),
    address varchar(255),
    primary key (seq)
)
```

▼실행

```
insert t
into
    User
(name, address, DTYPE, seq)
```

```

values
    (?, ?, 'UserOfficePrivacy', ?)
16:31:06 TRACE : binding parameter [1] as [VARCHAR] - [name1]
16:31:06 TRACE : binding parameter [2] as [VARCHAR] - [addr1]
16:31:06 TRACE : binding parameter [3] as [INTEGER] - [50]

```

```

insert
into
    User
    (name, age, height, weight, DTYPE, seq)
values
    (?, ?, ?, ?, 'UserBodyPrivacy', ?)
16:31:06 TRACE : binding parameter [1] as [VARCHAR] - [name2]
16:31:06 TRACE : binding parameter [2] as [INTEGER] - [2]
16:31:06 TRACE : binding parameter [3] as [DOUBLE] - [2.0]
16:31:06 TRACE : binding parameter [4] as [DOUBLE] - [2.0]
16:31:06 TRACE : binding parameter [5] as [INTEGER] - [51]

```

```

16:31:06 DEBUG :
insert
into
    User
    (name, password, DTYPE, seq)
values
    (?, ?, 'UserBankPrivacy', ?)
16:31:06 TRACE : binding parameter [1] as [VARCHAR] - [name2]
16:31:06 TRACE : binding parameter [2] as [VARCHAR] - [pwd]
16:31:06 TRACE : binding parameter [3] as [INTEGER] - [52]

```

	DTYPE	SEQ	NAME	AGE	HEIGHT	WEIGHT	PASSWORD	ADDRESS
1	UserOfficePrivacy	50	name1	<null>	<null>	<null>	<null>	addr1
2	UserBodyPrivacy	51	name2	2	2	2	<null>	<null>
3	UserBankPrivacy	52	name2	<null>	<null>	<null>	pwd	<null>

집고 넘어가기 : DTYPE 저것은 무엇일까 왜 들어가는걸까? 하이버네이트에서 자동으로 어떤 클래스에 의해 INSERT가 되었는지 출처를 알 수 있도록 DTYPE이라는 컬럼을 만들어 넣어버렸다. 문제를 해결해보자

▼DTYPE컬럼명을 변경해보자 @DiscriminatorColumn

```

@DiscriminatorColumn(name="CLASS_TYPE", discriminatorType=DiscriminatorType.STRING)
@SequenceGenerator(name = User.SEQ_NAME, sequenceName = User.SEQ_NAME, initialValue
= 50, allocationSize = 1)
public class User {
    public static final String SEQ_NAME = "SEQ_USER";
    @Id
    @GeneratedValue(strategy = GenerationType.SEQUENCE, generator = SEQ_NAME)
    Integer seq;
    String name;
}

```

Column	Data Type
SEQ	integer
CLASS_TYPE	varchar(31)
NAME	varchar(255)
AGE	integer
HEIGHT	double
WEIGHT	double
PASSWORD	varchar(255)
ADDRESS	varchar(255)

▼DTYPE컬럼을 값을 각각 클래스 별로 다르게 지정해보자 @DiscriminatorValue

<pre> @Data @Entity @DiscriminatorValue(value="BANK") public class UserBankPrivacy extends User{ String password; </pre>	<pre> @Data @Entity @DiscriminatorValue(value="BODY") public class UserBodyPrivacy extends User{ Integer age; </pre>	<pre> @Data @Entity @DiscriminatorValue(value="OFFICE") public class UserOfficePrivacy extends User{ String address; </pre>
--	--	---

```
}
double weight;
double height;
}
```

TEST.PUBLIC.USER								
<Filter criteria>								
	CLASS_TYPE	SEQ	NAME	AGE	HEIGHT	WEIGHT	PASSWORD	ADDRESS
1	OFFICE	50	name1	<null>	<null>	<null>	<null>	addr1
2	BODY	51	name2	2	2	2	<null>	<null>
3	BANK	52	name2	<null>	<null>	<null>	pwd	<null>

▼DTYPE컬럼을 없애보자 (사용하지 않기) @DiscriminatorFormula

```
@Data
@Entity
@Inheritance(strategy=InheritanceType.SINGLE_TABLE)
@DiscriminatorColumn(name="CLASS_TYPE",
discriminatorType=DiscriminatorType.STRING)
@DiscriminatorFormula("...")
@DiscriminatorValue(value="USER")
@SequenceGenerator(name = User.SEQ_NAME, sequenceName =
User.SEQ_NAME, initialValue = 50, allocationSize = 1)
public class User {
    public static final String SEQ_NAME = "SEQ_USER";
    @Id
    @GeneratedValue(strategy = GenerationType.SEQUENCE, generator =
SEQ_NAME)
    Integer seq;
    String name;
}
```

```
create table User (
    seq integer not null,
    name varchar(255),
    age integer,
    height double,
    weight double,
    password varchar(255),
    address varchar(255),
    primary key (seq)
)
```

▼실행

```
Session session = getSessionFactory().getCurrentSession();
session.beginTransaction();

UserOfficePrivacy userOffice = new UserOfficePrivacy();
userOffice.setName("name1");
userOffice.setAddress("addr1");
session.save(userOffice);

UserBodyPrivacy userBody = new UserBodyPrivacy();
userBody.setName("name2");
userBody.setAge(2);
userBody.setWeight(2.0);
userBody.setHeight(2.0);
session.save(userBody);

UserBankPrivacy userBank = new UserBankPrivacy();
userBank.setName("name2");
userBank.setPassword("pwd");
session.save(userBank);

session.getTransaction().commit();
```

```
insert
into
    User
(name, address, seq)
values
    (?, ?, ?)
```

```
16:40:56 TRACE : binding parameter [1] as [VARCHAR] - [name1]
16:40:56 TRACE : binding parameter [2] as [VARCHAR] - [addr1]
```

16:40:56 TRACE : binding parameter [3] as [INTEGER] - [50]

```
insert
into
  User
  (name, age, height, weight, seq)
values
  (?, ?, ?, ?, ?)
```

16:40:56 TRACE : binding parameter [1] as [VARCHAR] - [name2]

16:40:56 TRACE : binding parameter [2] as [INTEGER] - [2]

16:40:56 TRACE : binding parameter [3] as [DOUBLE] - [2.0]

16:40:56 TRACE : binding parameter [4] as [DOUBLE] - [2.0]

16:40:56 TRACE : binding parameter [5] as [INTEGER] - [51]

```
insert
into
  User
  (name, password, seq)
values
  (?, ?, ?)
```

16:40:56 TRACE : binding parameter [1] as [VARCHAR] - [name2]

16:40:56 TRACE : binding parameter [2] as [VARCHAR] - [pwd]

16:40:56 TRACE : binding parameter [3] as [INTEGER] - [52]

집고 넘어가기 : @Inheritance 어노테이션을 엔티티에 명시함으로써 Table-per-class 상속 전략을 정의한다. 이 어노테이션의 경우 strategy 변수를 통해 전략을 지정하는데, 여기서는 SINGLE_TABLE 전략을 사용한다.

그 외에도 InheritanceType 에는 TABLE_PER_CLASS 와 JOINED 전략이 있다. Table-per-class 전략을 이용할 때 InheritanceType.TABLE_PER_CLASS 값으로 잘못 지정할 수도 있다. 반드시 SINGLE_TABLE 만 사용하자.

Table-perclass

전략에서는 하위 클래스와 관련된 컬럼에는 NOT NULL 제약사항을 선언하지못한다는 점에 주의하자.

@Table-per-Subclass

Table-per-class 전략에서 구별자 컬럼을 이용하여 구분할 수있었지만

하나의 테이블을 이용하는 대신에 분리된 테이블을 사용하는 방법도 제공한다. @Table-per-subclass

```
@Inheritance(strategy=InheritanceType.JOINED)
```

```
@Data
@Entity
@Inheritance(strategy=InheritanceType.JOINED)
@DiscriminatorValue(value="USER")
@SequenceGenerator(name = UserSubClass.SEQ_NAME, sequenceName = UserSubClass.SEQ_NAME, initialValue = 50,
allocationSize = 1)
public class UserSubClass {
    public static final String SEQ_NAME = "SEQ_USER";
    @Id
    @GeneratedValue(strategy = GenerationType.SEQUENCE, generator = SEQ_NAME)
    Integer seq;
    String name;
}
```

```
@Data
@Entity
public class UserBankPrivacy extends UserSubClass{
    String password;
}
```

```
@Data
@Entity
public class UserBodyPrivacy extends UserSubClass{
    Integer age;
    double weight;
    double height;
}
```

```
@Data
@Entity
public class UserOfficePrivacy extends UserSubClass{
    String address;
}
```

▼실행

```
Session session = getSessionFactory().getCurrentSession();
session.beginTransaction();
```

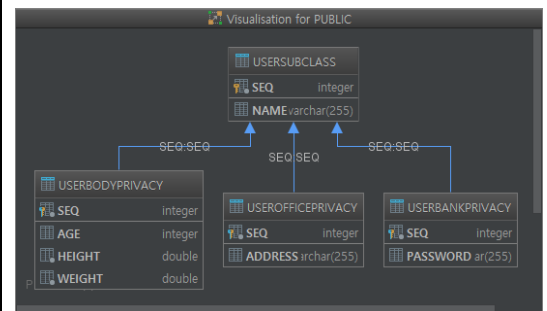
```
UserSubClass user = new UserSubClass();
user.setName("nameOffice");
session.save(user);
```

```
UserOfficePrivacy userOffice = new UserOfficePrivacy();
userOffice.setName("nameOffice");
userOffice.setAddress("addr");
session.save(userOffice);
```

```
UserBodyPrivacy userBody = new UserBodyPrivacy();
userBody.setName("nameBody");
userBody.setAge(2);
userBody.setWeight(2.0);
userBody.setHeight(2.0);
session.save(userBody);
```

```
UserBankPrivacy userBank = new UserBankPrivacy();
// userBank.setName("nameBank");
userBank.setPassword("pwd");
session.save(userBank);
```

```
session.getTransaction().commit();
```



TEST.PUBLIC.USERSUBCLASS	TEST.PUBLIC.USERBANKPRIVACY	TEST.PUBLIC.USERBODYPRIVACY	TEST.PUBLIC.USEROFFICEPRIVACY
Tab-se...d (TSV) View Query	Tab-se...d (TSV) View Query	Tab-se...d (TSV) View Query	Tab-se...d (TSV) View Query
<Filter criteria>	<Filter criteria>	<Filter criteria>	<Filter criteria>
SEQ NAME	PASSWORD SEQ	AGE HEIGHT WEIGHT SEQ	ADDRESS SEQ
1 50 nameOffice	1 pwd 53	1 2 2 2 52	1 addr 51
2 51 nameOffice			
3 52 nameBody			
4 53 <null>			

집고 넘어가기 : 부모 Entity 의 TABLE 을 기준으로 자식 테이블 내용이 들어간다 userBank Entity 를 넣을때 보면 부모 Entity 의 name 값을 넣지 않아도 부모 시퀀스가 들어가고 난뒤 자식 Entity 의 테이블에 내용이 들어간다

부모 SEQ ID 컬럼과 자식들의 SEQ 컬럼은 자동 생성된다.(부모 ID Column 이름으로)

@Table-per-Concrete-Class

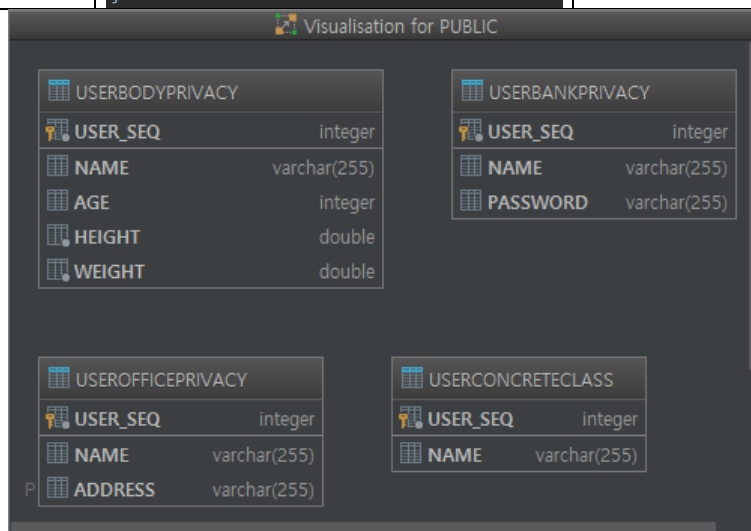
부모 클래스의 모든 속성이 자식 클래스와 관련된 테이블에 복사된다. (자주 쓰이지 않는다)

```
@Data
@Entity
@Inheritance(strategy=InheritanceType.TABLE_PER_CLASS)
@DiscriminatorValue(value="USER")
@SequenceGenerator(name = UserConcreteClass.SEQ_NAME, sequenceName = UserConcreteClass.SEQ_NAME, initialValue = 50, allocationSize = 1)
public class UserConcreteClass {
    public static final String SEQ_NAME = "SEQ_USER";
    @Id
    @GeneratedValue(strategy = GenerationType.SEQUENCE, generator = SEQ_NAME)
    @Column(name = "USER_SEQ")
    Integer seq;
    String name;
}
```

```
@Data
@Entity
public class UserBankPrivacy extends UserConcreteClass {
    String password;
}
```

```
@Data
@Entity
public class UserBodyPrivacy extends UserConcreteClass {
    Integer age;
    double weight;
    double height;
}
```

```
@Data
@Entity
public class UserOfficePrivacy extends UserConcreteClass {
    String address;
}
```



TEST.PUBLIC.USERCONCRETECLASS			TEST.PUBLIC.USEROFFICEPRIVACY		
>> Tab-se...d (TSV) View Query			<< << >> Tab-se...d (TSV) View Query		
<Filter criteria>			<Filter criteria>		
USER_SEQ	NAME		USER_SEQ	NAME	ADDRESS
1 50	nameOff ice		1 51	nameOff ice	addr

TEST.PUBLIC.USERBANKPRIVACY ×				TEST.PUBLIC.USERBODYPRIVACY ×					
1 row				1 row					
<Filter criteria>				<Filter criteria>					
USER_SEQ	NAME	PASSWORD		USER_SEQ	NAME	AGE	HEIGHT	WEIGHT	
1 53	<null>	pwd		1 52	nameBody	2	2	2	

하이버네이트 질의어

HQL(Hibernate Query Language)에서는 WHERE, ORDER BY, AVG, MAX 등을 SQL 처럼 사용할수 있습니다.

HQL 은 객체(Entity)를 사용합니다. 테이블을 나타내는 자리에 엔티티 객체 클래스명을 사용해야 합니다.

Query Class 사용하기

```
@Data
@Entity
@SequenceGenerator(name = User.SEQ_NAME, sequenceName = User.SEQ_NAME,
allocationSize = 1)
public class User {
    public static final String SEQ_NAME = "SEQ_USER";
    @Id
    @GeneratedValue(strategy = GenerationType.SEQUENCE)
    @Column(name = "USER_SEQ")
    Integer seq;
    String name;
    String password;
}
```

```
@Data
@Entity
public class Office {
    @Id
    @Column(name = "OFFICE_SEQ")
    Integer seq;
    String addr;
}
```

▼실행

```
Session session = getSessionFactory().getCurrentSession();
session.beginTransaction();

User user = new User();
user.setName("nanme1");
user.setPassword("pwd1");
session.save(user);

Office office = new Office();
office.setSeq(user.getSeq());
office.setAddr("gurol");
session.save(office);

user = new User();
user.setName("nanme2");
user.setPassword("pwd2");
session.save(user);
office = new Office();
office.setSeq(user.getSeq());
office.setAddr("gurol2");
session.save(office);
session.flush();
session.clear();

Query query = session.createQuery("from User");
List list = query.getResultList();
list.forEach(it->/og.debug("-->"+it.toString()));

session.getTransaction().commit();
```

Visualisation for PUBLIC

USER	
USER_SEQ	integer
NAME	varchar(255)
PASSWORD	varchar(255)

OFFICE	
OFFICE_SEQ	integer
ADDR	varchar(255)

Powered by yFiles

select


```

        user0_.USER_SEQ as USER_SEQ1_1_,
        user0_.name as name2_1_,
        user0_.password as password3_1_
    from
        User user0_

```

```

11:05:18 DEBUG : -->User(seq=1, name=nanme1, password=pwd1)
11:05:18 DEBUG : -->User(seq=2, name=nanme2, password=pwd2)

```

▼실행 (where)

```
Query query = session.createQuery("from User as A where A.seq = 1");
```

```

select
    user0_.USER_SEQ as USER_SEQ1_1_,
    user0_.name as name2_1_,
    user0_.password as password3_1_
from
    User user0_
where
    user0_.USER_SEQ=1

```

```
-->User(seq=1, name=nanme1, password=pwd1)
```

▼실행 (join) (모든객체 받아오기) Object[] 로받아온다.

```

Query query = session.createQuery("from User as A, Office as B where A.seq = 1 AND A.seq = B.seq");
List<Object[]> list = query.getResultList();
list.forEach(it->{
    for (Object entity : it) {
        log.debug("-->" + entity.toString());
    }
});

```

```

select
    user0_.USER_SEQ as USER_SEQ1_1_0_,
    office1_.OFFICE_SEQ as OFFICE_S1_0_1_,
    user0_.name as name2_1_0_,
    user0_.password as password3_1_0_,
    office1_.addr as addr2_0_1_
from
    User user0_ cross
join
    Office office1_
where
    user0_.USER_SEQ=1
    and user0_.USER_SEQ=office1_.OFFICE_SEQ

```

```

11:10:44 DEBUG : -->User(seq=1, name=nanme1, password=pwd1)
11:10:44 DEBUG : -->Office(seq=1, addr=guro1)

```

▼실행 (join) (특정객체 받아오기)

```

@Data
@Entity
public class Office {
    @Id
    @Column(name = "OFFICE_SEQ")
    Integer seq;
    String addr;
    public Office() {
    }
    public Office(Integer seq, String addr){
        this.seq = seq;
        this.addr = addr;
    }
}

```

```

Query query = session.createQuery("SELECT new Office (B.seq,B.addr)
from User as A, Office as B where A.seq = 1 AND A.seq = B.seq");
List<Office> list = query.getResultList();
list.forEach(it->log.debug("-->" + it.toString()));

```

```

select
    office1_.OFFICE_SEQ as col_0_0_,
    office1_.addr as col_1_0_
from
    User user0_ cross
join

```

```
}  
}
```

필드를받을수 있는 생성자를 생성한다

```
Office office1_  
where  
    user0_.USER_SEQ=1  
    and user0_.USER_SEQ=office1_.OFFICE_SEQ
```

11:15:27 DEBUG : -->Office(seq=1, addr=guro1)

▼실행 (그룹함수)

```
Query query = session.createQuery("SELECT MAX(A.seq) from User as A");  
Object result = query.getSingleResult();  
log.debug("-->" + result);
```

```
select  
    max(user0_.USER_SEQ) as col_0_0_  
from  
    User user0_
```

-->2

▼실행 (그룹함수 복합)

```
Query query = session.createQuery("SELECT MAX(A.seq), MIN(A.seq), AVG(A.seq), COUNT(*) from User as A");  
Object[] results = (Object[]) query.getSingleResult();  
for (Object entity : results) {  
    log.debug("-->" + entity.toString());  
}
```

```
select  
    max(user0_.USER_SEQ) as col_0_0_,  
    min(user0_.USER_SEQ) as col_1_0_,  
    avg(cast(user0_.USER_SEQ as double)) as col_2_0_,  
    count(*) as col_3_0_  
from  
    User user0_
```

11:27:35 DEBUG : -->2
11:27:35 DEBUG : -->1
11:27:35 DEBUG : -->1.5
11:27:35 DEBUG : -->2

▼HSQL에 변수 바인딩하기 ? 처리

```
Query query = session.createQuery("from User as A where A.seq = ?");  
query.setParameter(0, new Integer(1));
```

▼HSQL에 변수 바인딩하기 치환이름 처리

```
Query query = session.createQuery("from User as A where A.seq = :seq");  
query.setParameter("seq", new Integer(1));
```

▼HSQL에 변수 바인딩하기 in조건 list 처리

```
Query query = session.createQuery("from User as A where A.seq in :seq");  
query.setParameter("seq", Arrays.asList(new Integer(1), new Integer(2), new Integer(3), new Integer(4), new Integer(5)));
```

```
select  
    user0_.USER_SEQ as USER_SEQ1_1_,  
    user0_.name as name2_1_,  
    user0_.password as password3_1_  
from  
    User user0_  
where  
    user0_.USER_SEQ in (
```

?, ?, ?, ?, ?

)

15:13:42 TRACE : binding parameter [1] as [INTEGER] - [1]
15:13:42 TRACE : binding parameter [2] as [INTEGER] - [2]
15:13:42 TRACE : binding parameter [3] as [INTEGER] - [3]
15:13:42 TRACE : binding parameter [4] as [INTEGER] - [4]
15:13:42 TRACE : binding parameter [5] as [INTEGER] - [5]

▼단일결과 가져올때

```
query.getSingleResult();
```

▼복수결과 가져올때

```
query.getResultList();
```

▼DELETE문

```
session.createQuery("delete UserBio as bio where height=:height");
```

@Embedded Objects

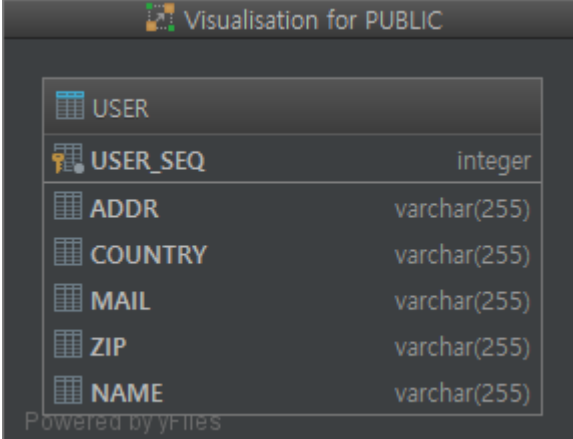
object를 하나의 Entity에 속해있는것처럼 할수 있는 방법입니다.

```
@Data
@Entity
@SequenceGenerator(name = User.SEQ_NAME,
sequenceName = User.SEQ_NAME, allocationSize = 1)
public class User {
    public static final String SEQ_NAME = "SEQ_USER";
    @Id
    @GeneratedValue(strategy =
GenerationType.SEQUENCE)
    @Column(name = "USER_SEQ")
    Integer seq;
    String name;

    @Embedded
    Address addr;
}
```

```
@Data
@Embeddable
public class Address {
    String mail;
    String zip;
    String country;
    String addr;
}
```

```
create table User (
    USER_SEQ integer not null,
    addr varchar(255),
    country varchar(255),
    mail varchar(255),
    zip varchar(255),
    name varchar(255),
    primary key (USER_SEQ)
)
```



Visualisation for PUBLIC

USER	
USER_SEQ	integer
ADDR	varchar(255)
COUNTRY	varchar(255)
MAIL	varchar(255)
ZIP	varchar(255)
NAME	varchar(255)

Powered by yFiles

▼실행

```
Session session = getSessionFactory().getCurrentSession();
session.beginTransaction();

Address addr = new Address();
addr.setAddr("address");
addr.setCountry("KOREA");
addr.setMail("visualkhh@gmail.com");
addr.setZip("08-08");

User user = new User();
user.setName("name");
user.setAddr(addr);

session.save(user);

session.getTransaction().commit();
```

```
insert
into
User
(addr, country, mail, zip, name, USER_SEQ)
values
(?, ?, ?, ?, ?, ?)
11:36:54 TRACE : binding parameter [1] as [VARCHAR] - [address]
11:36:54 TRACE : binding parameter [2] as [VARCHAR] - [KOREA]
11:36:54 TRACE : binding parameter [3] as [VARCHAR] - [visualkhh@gmail.com]
11:36:54 TRACE : binding parameter [4] as [VARCHAR] - [08-08]
```

11:36:54 TRACE : binding parameter [5] as [VARCHAR] - [name]
11:36:54 TRACE : binding parameter [6] as [INTEGER] - [1]

TEST.PUBLIC.USER x						
1 row						
<Filter criteria>						
USER_SEQ	ADDR	COUNTRY	MAIL	ZIP	NAME	
1	1	address	KOREA	visua1khh@gmail.com	08-08	name

@ElementCollection

Enum 사용하기 @Enumerated

별도의 Entity 를 만들지 않고 Collection 을 테이블로 사용하자.

```
@Data
@Entity
@SequenceGenerator(name = User.SEQ_NAME, sequenceName = User.SEQ_NAME,
allocationSize = 1)
public class User {
    public static final String SEQ_NAME = "SEQ_USER";

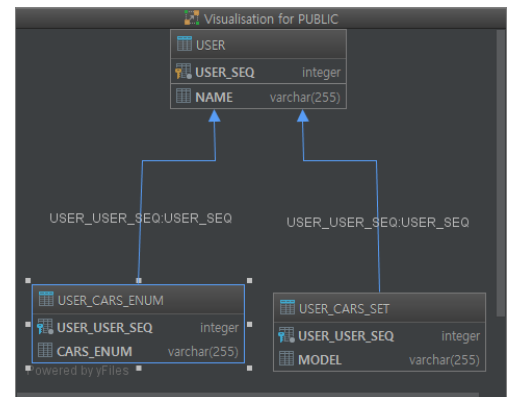
    public enum CAR_TYPE{
        ENUM_MODEL_1,
        ENUM_MODEL_2,
        ENUM_MODEL_3,
        ENUM_MODEL_4
    }

    @Id
    @GeneratedValue(strategy = GenerationType.SEQUENCE)
    @Column(name = "USER_SEQ")
    Integer seq;
    String name;

    @ElementCollection
    Set<Car> cars_set;

    @ElementCollection
    @Enumerated(EnumType.STRING)
    List<CAR_TYPE> cars_enum;
}
```

```
@Data
@Embeddable
public class Car {
    String model;
    public Car(){}
    public Car(String
model){this.model=model;}
}
```



▼실행

```
Session session = getSessionFactory().getCurrentSession();
session.beginTransaction();

User user = new User();
user.setName("name");
user.setCars_set(new HashSet<Car>((Arrays.asList(new Car("MODEL_2"), new Car("MODEL_1")))));
user.setCars_enum(Arrays.asList(User.CAR_TYPE.ENUM_MODEL_1, User.CAR_TYPE.ENUM_MODEL_2, User.CAR_TYPE.ENUM_MODEL_3));

session.save(user);

session.getTransaction().commit();
```

```

create table User (
    USER_SEQ integer not null,
    name varchar(255),
    primary key (USER_SEQ)
)
14:28:27 DEBUG :

create table User_cars_enum (
    User_USER_SEQ integer not null,
    cars_enum varchar(255)
)
14:28:27 DEBUG :

create table User_cars_set (
    User_USER_SEQ integer not null,
    model varchar(255)
)
14:28:27 DEBUG :

alter table User_cars_enum
    add constraint FKnyajuee7ju7fuxvjm01owpi8
    foreign key (User_USER_SEQ)
    references User
14:28:27 DEBUG :

alter table User_cars_set
    add constraint FK90sgwbja3qhfr5hntsu0sfjmh
    foreign key (User_USER_SEQ)
    references User

```

TEST.PUBLIC.USER x	
1 row	
<Filter criteria>	
USER_SEQ	NAME
1	name

TEST.PUBLIC.USER_CARS_ENUM x	
3 rows	
<Filter criteria>	
USER_USER_SEQ	CARS_ENUM
1	ENUM_MODEL_1
2	ENUM_MODEL_2
3	ENUM_MODEL_3

TEST.PUBLIC.USER_CARS_SET x	
2 rows	
<Filter criteria>	
USER_USER_SEQ	MODEL
1	MODEL_2
2	MODEL_1

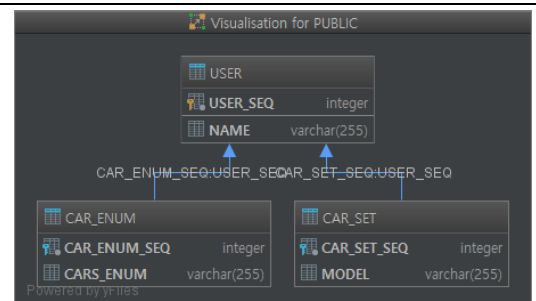
▼테이블명, FK컬럼 명바꾸기.

```

@ElementCollection
@CollectionTable(name="CAR_SET", joinColumns=@JoinColumn(name="CAR_SEQ", referencedColumnName="USER_SEQ"))
Set<Car> cars_set;

@ElementCollection
@CollectionTable(name="CAR_ENUM", joinColumns=@JoinColumn(name="CAR_ENUM_SEQ", referencedColumnName="USER_SEQ"))
@Enumerated(EnumType.STRING)
List<CAR_TYPE> cars_enum;

```



@Temporal

```
@Temporal(value= TemporalType.DATE)
private Date onlyDate;

@Temporal(value=TemporalType.TIME)
private Date onlyTime;

@Temporal(value=TemporalType.TIMESTAMP)
private Date dateAndTime;
```

@Enumerated

```
@Enumerated
private Enum vaules;

@Enumerated(value=EnumType.STRING)
private Enum vauleAsString;

@Enumerated(value=EnumType.ORDINAL)
private Enum vauleAsNumber;
```

Pagination 페이지네이션

몇 개의 레코드만 가져오려면 `setMaxResults()` 메소드에 한계치와 함께 호출함으 로써 페이지네이션 Pagination 기능을 사용할 수 있다.

```
Session session = getSessionFactory().getCurrentSession();
session.beginTransaction();

IntStream.range(1, 100).forEach(it->{
    session.save(new User("name"+it));
});

session.flush();
session.clear();

Query query = session.createQuery("from User");
query.setFirstResult(30); //30 번째 로우부터
query.setMaxResults(5); //5 개를 가져와라
List<User> users = query.getResultList();
users.stream().forEach(it->/log.debug("-->" + it));

/log.debug("-----");

query = session.createQuery("from User");
query.setFirstResult(40); //40 번째 로우부터
query.setMaxResults(5); //5 개를 가져와라
users = query.getResultList();
users.stream().forEach(it->/log.debug("-->" + it));

session.getTransaction().commit();
```

```

select
    user0_.USER_SEQ as USER_SEQ1_0_,
    user0_.name as name2_0_
from
    User user0_ limit ? offset ?
15:01:19 DEBUG : -->User(seq=31, name=name31)
15:01:19 DEBUG : -->User(seq=32, name=name32)
15:01:19 DEBUG : -->User(seq=33, name=name33)
15:01:19 DEBUG : -->User(seq=34, name=name34)
15:01:19 DEBUG : -->User(seq=35, name=name35)
15:01:19 DEBUG : -----
select
    user0_.USER_SEQ as USER_SEQ1_0_,
    user0_.name as name2_0_
from
    User user0_ limit ? offset ?
15:01:19 DEBUG : -->User(seq=41, name=name41)
15:01:19 DEBUG : -->User(seq=42, name=name42)
15:01:19 DEBUG : -->User(seq=43, name=name43)
15:01:19 DEBUG : -->User(seq=44, name=name44)
15:01:19 DEBUG : -->User(seq=45, name=name45)

```

집고 넘어가기 : 페이징 처리에 아주 유용하게 사용할수 있다. 각각 DB종류 에따라 limit를 걸지 rownum을걸지 이것또한 hibernate가 알아서 해준다

Criteria

하이버네이트에서는 criteria 를 도입하여 필터링의 또 다른 방법을 제공합니다.

Criteria 와 Restrictions 클래스를 이용하여 좀더 편하게 필터링을 해보자구요~

https://docs.jboss.org/hibernate/orm/5.0/userguide/html_single/chapters/query/criteria/Criteria.html

```

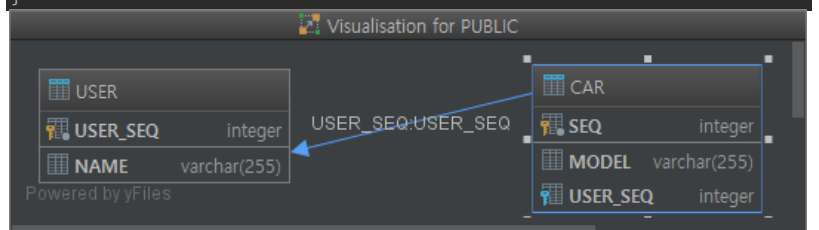
@Data
@Entity
@SequenceGenerator(name = User.SEQ_NAME,
sequenceName = User.SEQ_NAME, allocationSize = 1)
public class User {
    public static final String SEQ_NAME =
"SEQ_USER";
    @Id
    @GeneratedValue(strategy =
GenerationType.SEQUENCE)
    @Column(name = "USER_SEQ")
    Integer seq;
    String name;
    public User() {}
    public User(String name, List<Car> cars)
{ this.name=name; this.cars=cars; }
    @OneToMany(cascade = CascadeType.ALL)
    @JoinColumn(name = "USER_SEQ")
    List<Car> cars;
}

```

```

@Data
@Entity
@SequenceGenerator(name = Car.SEQ_NAME, sequenceName =
Car.SEQ_NAME, allocationSize = 1)
public class Car {
    public static final String SEQ_NAME = "SEQ_CAR";
    @Id
    @GeneratedValue(strategy = GenerationType.SEQUENCE, generator =
SEQ_NAME)
    Integer seq;
    String model;
    public Car() {}
    public Car(String model) { this.model=model; }
}

```



```

Session session = getSessionFactory().getCurrentSession();
session.beginTransaction();

IntStream.range(1,5).forEach(it->{
    User user = new User("name_"+it, Arrays.asList(new Car("CAR_"+it), new Car("TRUCK_"+it)));

```



```

        session.save(user);
    });

    session.flush();
    session.clear();

    // Create CriteriaBuilder
    CriteriaBuilder builder = session.getCriteriaBuilder();

    // Create CriteriaQuery
    CriteriaQuery<User> criteria = builder.createQuery(User.class);
    Root<User> studentRoot = criteria.from(User.class);
    criteria.select(studentRoot).where(builder.equal(studentRoot.get("seq"), 1));

    TypedQuery<User> query = session.createQuery(criteria);
    List<User> contacts = query.getResultList();

    session.getTransaction().commit();

```

```

select
    user0_.USER_SEQ as USER_SEQ1_1_,
    user0_.name as name2_1_
from
    User user0_
where
    user0_.USER_SEQ=1

```

```

select
    cars0_.USER_SEQ as USER_SEQ3_0_0_,
    cars0_.seq as seq1_0_0_,
    cars0_.seq as seq1_0_1_,
    cars0_.model as model2_0_1_
from
    Car cars0_
where
    cars0_.USER_SEQ=?
16:19:28 TRACE : binding parameter [1] as [INTEGER] - [1]

```

▼ 여러 조건걸기 and

```

// Create CriteriaBuilder
CriteriaBuilder builder = session.getCriteriaBuilder();

// Create CriteriaQuery
CriteriaQuery<User> criteria = builder.createQuery(User.class);
Root<User> studentRoot = criteria.from(User.class);

Predicate w1 = builder.equal(studentRoot.get("seq"), 1);
Predicate w2 = builder.equal(studentRoot.get("name"), "name_1");
criteria.select(studentRoot).where(w1, w2);

TypedQuery<User> query = session.createQuery(criteria);
List<User> contacts = query.getResultList();

```

```

select
    user0_.USER_SEQ as USER_SEQ1_1_,
    user0_.name as name2_1_
from
    User user0_
where

```

```
user0_.USER_SEQ=1
and user0_.name=?
16:23:00 TRACE : binding parameter [1] as [VARCHAR] - [name_1]
```

▼ 여러 조건걸기 or

```
Predicate w1 = builder.equal(studentRoot.get("seq"), 1);
Predicate w2 = builder.equal(studentRoot.get("name"), "name_1");
Predicate w3 = builder.or( w1,w2);
```

```
select
    user0_.USER_SEQ as USER_SEQ1_1_,
    user0_.name as name2_1_
from
    User user0_
where
    user0_.USER_SEQ=1
    or user0_.name=?
17:13:09 TRACE : binding parameter [1] as [VARCHAR] - [name_1]
```

▼ group by

```
Predicate w1 = builder.equal(studentRoot.get("seq"), 1);
Predicate w2 = builder.equal(studentRoot.get("name"), "name_1");
Predicate w3 = builder.or( w1,w2);

criteria.select(studentRoot).where(w3).groupBy(studentRoot.get("name"));
```

```
select
    user0_.USER_SEQ as USER_SEQ1_1_,
    user0_.name as name2_1_
from
    User user0_
where
    user0_.USER_SEQ=1
    or user0_.name=?
group by
    user0_.name
```

▼ like

```
Predicate w3 = builder.like(studentRoot.get("name"), "%name_1%");
//or user0_.name like ?
```

▼ between

```
Predicate w4 = builder.between(studentRoot.get("seq"), 1,100);
// or user0_.USER_SEQ between 1 and 100
```

네임드 쿼리

클래스 레벨에서 엔티티의 쿼리를 사용하기 위해 `@NamedQuery` 어노테이션을 이용하거나 매핑 파일에 선언할 수 있다.

```
@Data
@Entity
@SequenceGenerator(name = User.SEQ_NAME, sequenceName = User.SEQ_NAME, allocationSize = 1)
@NamedQueries(
    value = {
        @NamedQuery(name = User.NQ_FIND_ALL, query = "from User"),
        @NamedQuery(name = User.NQ_FIND_BY_USERNAME, query = "from User where name=:name")
    }
)

public class User {
    public static final String SEQ_NAME = "SEQ_USER";
    public static final String NQ_FIND_ALL = "USER_findAll";
    public static final String NQ_FIND_BY_USERNAME = "USER_findByUsername";

    @Id
    @GeneratedValue(strategy = GenerationType.SEQUENCE)
    @Column(name = "USER_SEQ")
    Integer seq;
    String name;
    public User() {}
    public User(String name, List<Car> cars) {this.name=name;this.cars=cars;}
    @OneToMany(cascade = CascadeType.ALL, fetch = FetchType.EAGER)
    @JoinColumn(name = "USER_SEQ")
    List<Car> cars;
}
```

```
@Data
@Entity
@SequenceGenerator(name = Car.SEQ_NAME, sequenceName = Car.SEQ_NAME, allocationSize = 1)
public class Car {
    public static final String SEQ_NAME = "SEQ_CAR";
    @Id
    @GeneratedValue(strategy = GenerationType.SEQUENCE, generator = SEQ_NAME)
    Integer seq;
    String model;
    public Car() {}
    public Car(String model) {this.model=model;}
}
```

▼실행 FindAll

```
Session session = getSessionFactory().getCurrentSession();
session.beginTransaction();

IntStream.range(1,5).forEach(it->{
    User user = new User("name_"+it, Arrays.asList(new Car("CAR_"+it),new Car("TRUCK_"+it)));
    session.save(user);
});

session.flush();
session.clear();

Query query = session.getNamedQuery(User.NQ_FIND_ALL);
List<User> users = query.getResultList();
users.stream().forEach(it->/og.debug("-->"+it.getName()));

session.getTransaction().commit();
```

select

```

        user0_.USER_SEQ as USER_SEQ1_1_,
        user0_.name as name2_1_
    from
        User user0_

```

▼실행 Parameter

```

Query query = session.getNamedQuery(User.NQ_FIND_BY_USERNAME);
query.setParameter("name", "name_4");
List<User> users = query.getResultList();
users.stream().forEach(it->/og.debug("-->" + it.getName()));

```

```

select
    user0_.USER_SEQ as USER_SEQ1_1_,
    user0_.name as name2_1_
from
    User user0_
where
    user0_.name=?
11:53:13 TRACE : binding parameter [1] as [VARCHAR] - [name_4]

```

집고 넘어가기 : NamedQuery 하나만 지정하려면

```

@NamedQuery(name = User.NQ_FIND_ALL, query = "from User"),
public class User {

```

<http://www.journaldev.com/3451/hibernate-named-query-example-namedquery>

<http://www.mkyong.com/hibernate/hibernate-named-query-examples/>

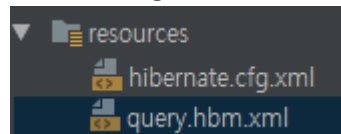
xml 맵핑 하여 사용하기

```

<hibernate-configuration>
    <session-factory>
        ....
        <!--List of XML mapping files -->
        <mapping resource="query.hbm.xml" />
    </session-factory>
</hibernate-configuration>

```

hibernate.cfg.xml



```

<hibernate-mapping>
    <query name="MNQ_USER_FINDALL">from User</query>
    <query name="MNQ_USER_FINDBYUSERNAME">
        <![CDATA[from User where name = :id]]>
    </query>
    <sql-query name="MSQ_USER_FINDALL">
        <![CDATA[select * from USER]]>
        <return alias="e"
class="com.khh.hibernate.c7.entity.User" />
    </sql-query>
</hibernate-mapping>

```

query.hbm.xml

```

Query query = session.getNamedQuery("MNQ_USER_FINDALL");

```

네이티브 쿼리

네이티브 쿼리를 실행하는 기능도 제공한다

HQL 이 사용하기 간편한데 왜 네이티브 SQL 을 ???

데이터베이스 벤더의 특정 함수 또는 생성에 의존적인 쿼리문이 있을 때 (예 : 통계 업무)

네이티브 쿼리를 사용할수밖에 없다

```
@Data
@Entity
@SequenceGenerator(name = User.SEQ_NAME, sequenceName = User.SEQ_NAME, allocationSize = 1)
@NamedNativeQueries(
    value = {
        @NamedNativeQuery(name = User.SQ_FIND_ALL, query = "SELECT * FROM USER" , resultClass = User.class),
        @NamedNativeQuery(name = User.SQ_FIND_BY_USERNAME, query = "SELECT * FROM USER WHERE NAME=:name")
    }
)

public class User {
    public static final String SEQ_NAME = "SEQ_USER";
    public static final String SQ_FIND_ALL = "USER_S_findAll";
    public static final String SQ_FIND_BY_USERNAME = "USER_S_findByUsername";

    @Id
    @GeneratedValue(strategy = GenerationType.SEQUENCE)
    @Column(name = "USER_SEQ")
    Integer seq;
    String name;
    public User() {}
    public User(String name, List<Car> cars) {this.name=name;this.cars=cars;}
    @OneToMany(cascade = CascadeType.ALL, fetch = FetchType.EAGER)
    @JoinColumn(name = "USER_SEQ")
    List<Car> cars;
}
```

```
@Data
@Entity
@SequenceGenerator(name = Car.SEQ_NAME, sequenceName = Car.SEQ_NAME, allocationSize = 1)
public class Car {
    public static final String SEQ_NAME = "SEQ_CAR";
    @Id
    @GeneratedValue(strategy = GenerationType.SEQUENCE, generator = SEQ_NAME)
    Integer seq;
    String model;
    public Car() {}
    public Car(String model) {this.model=model;}
}
```

▼실행 FindAll

```
Session session = getSessionFactory().getCurrentSession();
session.beginTransaction();

IntStream.range(1,5).forEach(it->{
    User user = new User("name_"+it, Arrays.asList(new Car("CAR_"+it),new Car("TRUCK_"+it)));
    session.save(user);
});

session.flush();
session.clear();

NativeQuery query = session.getNamedNativeQuery(User.SQ_FIND_ALL);
List<User> users = query.getResultList();
```

```
users.stream().forEach(it->/og.debug("-->"+it.getName()));
session.getTransaction().commit();
```

```
SELECT
    *
FROM
    USER
```

▼ XML 맵핑

```
<hibernate-mapping>
  <sql-query name="MSQ_USER_FINDALL">
    <![CDATA[select * from USER]]>
    <return alias="e" class="com.khh.hibernate.c7.entity.User" />
  </sql-query>
</hibernate-mapping>
```

Groovy Template 이용하여 Dynamic Query 사용하기

```
<hibernate-mapping>
  <sql-query name="MSQ_USER">
    <![CDATA[
      select * from USER
      <%if(null != name && name.length()>0 ) {%>
        WHERE name = :name
      <%}%>
    ]]>
  </sql-query>
</hibernate-mapping>
```

```
//groovy template dynamic query
NativeQuery query = session.getNamedNativeQuery("MSQ_USER");
String queryStr = query.getQueryString();
Map<String, Object> param = new HashMap<>();
param.put("name", "name_2");
queryStr = makeGQuery(queryStr, param);
NativeQuery dynamicQuery = session.createNativeQuery(queryStr);
param.entrySet().stream().forEach(it->dynamicQuery.setParameter(it.getKey(), it.getValue()));
query = dynamicQuery;
List<Object[]> users = query.getResultList();
users.stream().forEach(it->/og.debug("-->"+it[1]));
```

```
public String makeGQuery(String template, Map<String, Object> o) throws ScriptException {
    ScriptEngineManager factory = new ScriptEngineManager();
    ScriptEngine engine = factory.getEngineByName("groovy");
    StringBuffer t = new StringBuffer();
    t.append("def engine = new groovy.text.SimpleTemplateEngine(false);");
    t.append("  def template = engine.createTemplate(template);");
    t.append("  def binding = [");
    String commaSeparatedNumbers = (String)o.entrySet().stream().map((at) -> {
        Object val = at.getValue();
        if(at.getValue() != null && String.class.isAssignableFrom(val.getClass())) {
            val = "'" + val.toString() + "'";
        }

        return val == null?(String)at.getKey() + ":null":(String)at.getKey() + ":" + val.toString() + ",";
    }).collect(Collectors.joining(", "));
    t.append(commaSeparatedNumbers);
    t.append("];");
    /og.debug("Groovy Template-->"+t);
    engine.put("template", template);
    return (String)engine.eval(t + " template.make(binding).toString()");
}
```