

|  |
| --- |
| **HIBER NATE 5.x**  **시작하기** |

수익성의 성격을 띄지 않는다면

재배포, 수정 가능합니다.

[visualkhh@gmail.com](mailto:visualkhh@gmail.com)

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

(2016. 12. 21)

목 차

[ORM (Object Relational Mapping) 무엇인가? 5](#_Toc470167544)

[JPA (Java Persistence API) 무엇인가? 5](#_Toc470167545)

[HIBERNATE 무엇인가? 5](#_Toc470167546)

[왜 JPA를 쓰는가? 5](#_Toc470167547)

[1. 기존 SQL 중심적인 개발시 불편하다 5](#_Toc470167548)

[2. 객체-관계 간 모델 불일치 5](#_Toc470167549)

[3. 상속 불일치 6](#_Toc470167550)

[4. 관계와 연관 관계의 불일치 6](#_Toc470167551)

[장단점 6](#_Toc470167552)

[장점 6](#_Toc470167553)

[단점 6](#_Toc470167554)

[JPA, HIBERNATE Architecture 7](#_Toc470167555)

[엔티티 상태 및 생명주기 9](#_Toc470167556)

[비영속 상태 9](#_Toc470167557)

[영속 상태 9](#_Toc470167558)

[준영속 상태 9](#_Toc470167559)

[Hibernate 셋팅 11](#_Toc470167560)

[gradle 11](#_Toc470167561)

[hibernate.cfg.xml 13](#_Toc470167562)

[Session 14](#_Toc470167563)

[Session API 14](#_Toc470167564)

[SessionFactory 14](#_Toc470167565)

[Hibernate-provided BasicTypes 15](#_Toc470167566)

[Entity 18](#_Toc470167567)

[@Entity 18](#_Toc470167568)

[식별자 19](#_Toc470167569)

[@GeneratedValue 19](#_Toc470167570)

[복합 식별자 ID (KEY) 22](#_Toc470167571)

[키선언 첫번째 방법 (@Embeddable) 23](#_Toc470167572)

[키선언 두번째 방법 (@EmbeddedId) 24](#_Toc470167573)

[키선언 세번째 방법 (@IdClass) 25](#_Toc470167574)

[Join 조인 26](#_Toc470167575)

[연관 관계 26](#_Toc470167576)

[다중성 26](#_Toc470167577)

[방향성 26](#_Toc470167578)

[조인전에 먼저 알아야될 Cascade 26](#_Toc470167579)

[1:1 @OneToOne 27](#_Toc470167580)

[방향성을 갖자 31](#_Toc470167581)

[@OneToMany, @ManyToOne 32](#_Toc470167582)

[1:N, N:1, N:N 컬렉션 영속화 (List, Set, Map, Array[]) 32](#_Toc470167583)

[방향성 갖자 44](#_Toc470167584)

[조회하기 44](#_Toc470167585)

[@ManyToMany 47](#_Toc470167586)

[N:N연결하기 50](#_Toc470167587)

[캐싱 55](#_Toc470167588)

[1차 캐시 55](#_Toc470167589)

[2차 캐시 56](#_Toc470167590)

[cache usage 속성 56](#_Toc470167591)

[쿼리 캐시 57](#_Toc470167592)

[상속 전략 58](#_Toc470167593)

[@Table-per-Class 58](#_Toc470167594)

[@Table-per-Subclass 62](#_Toc470167595)

[@Table-per-Concrete-Class 63](#_Toc470167596)

[하이버네이트 질의어 64](#_Toc470167597)

[Query Class 사용하기 64](#_Toc470167598)

[@Embedded Objects 68](#_Toc470167599)

[@ElementCollection 69](#_Toc470167600)

[Pagination 페이지네이션 71](#_Toc470167601)

[Criteria 72](#_Toc470167602)

[네임드 쿼리 75](#_Toc470167603)

[네이티브 쿼리 77](#_Toc470167604)

[Groovy Template 이용하여 Dynamic Query 사용하기 78](#_Toc470167605)

ORM (Object Relational Mapping) 무엇인가?

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

때문에 ORM을 사용해 오브젝트와 RDB 사이에 객체지향적으로 다루기 위한 기술입니다.

JPA (Java Persistence API) 무엇인가?

ORM 전문가가 참여한 EJB 3.0 스펙 작업에서 기존 EJB ORM이던 Entity Bean을 JPA라고 바꾸고 JavaSE, JavaEE를 위한 영속성(persistence) 관리와 ORM을 위한 표준 기술입니다. JPA는 ORM 표준 기술로 Hibernate, OpenJPA, EclipseLink, TopLink Essentials과 같은 구현체가 있고 이에 표준 인터페이스가 바로 JPA입니다.

HIBERNATE 무엇인가?

Boss에서 개발한 ORM(Object Relational Mapping) 프레임워크 입니다.

장점

Hibernate는 특정 클래스에 매핑되어야 하는 데이터베이스의 테이블에 대한 관계 정의가 되어 있는 XML 파일의 메타데이터로 객체관계 매핑을 간단하게 수행시킵니다.

Hibernate를 사용하면 데이터베이스가 변경되더라도 SQL 스크립트를 수정하는등의 작업을 할 필요가 없습니다.

애플리케이션에서 사용되는 데이터베이스를 변경시키고자 한다면 설정파일의 dialect 프로퍼티를 수정함으로서 쉽게 처리할 수 있습니다.

Hibernate는 MySQL, Oracle, Sybase, Derby, PostgreSQL를 포함한 많은 데이터베이스를 지원하며 POJO기반의 모델과도 원활하게 동작합니다.

왜 JPA를 쓰는가?

1. 기존 SQL 중심적인 개발시 불편하다

* 쿼리가 변경되면 이에따른 프로그램 소스 DTO객체의 변경도 불가피하게 일어난다
* 데이터를 가져와 객체지향적으로 관계를 Mapping하는 일이 매번 일어난다.

!!SQL의존적인 개발이 이루어진다.

2. 객체-관계 간 모델 불일치

관계형 데이터베이스에는 로우와 컬럼의 2차원 형태로 데이터가 저장된다. 데이터 관계는 외래키foreign 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 불일치object-relational impedance mismatch라고 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 방법은없지만, 문제를 풀 수 있는 몇 가지 접근법이 있습니다. 이 접근법은 다양한 클래스-테이블class-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. (JPA를 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

Application

Hibernate

JPA

JDBC API

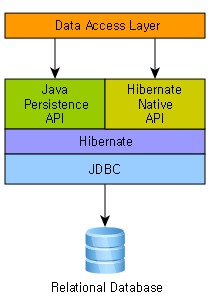
DB

Application

JPA 인터페이스

Hibernate, EclipseLink, DataNucleus..등

...









엔티티 상태 및 생명주기

비영속 상태

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

영속 상태

현재 활성화된 Session과 연결된 퍼시스턴트 객체. 이 상태의 퍼시스턴트 객체는 고유성을 가지며, 프로퍼티 값의 변경이 Session을 통해 자동으로 데이터베이스에 반영된다.

준영속 상태

영속 상태의 퍼시스턴트 객체가 Session과 연결이 끊기면 준영속 상태가 된다. Hibernate의 관리를 받지는 않지만, 영속 데이터를 갖고 있다.

|  |  |
| --- | --- |
|  | http://cfs13.tistory.com/image/10/tistory/2008/12/08/20/54/493d0af9b086d |
| 2016-07-11_13-38-11 | |
| http://cfs13.tistory.com/image/12/tistory/2008/12/08/20/54/493d0af9b9002 | |

|  |
| --- |
| 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 scaner \*/  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 XML으로 설정  Configuration configuration = getHibernateConfigByCode();  //CODE BASE로 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 가상머신Standard 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 클래스factory class다. 이는 Thread-safe한 객체이므로 데이터가 의도하지 않게 바뀌는 것을 염려하지 않고, 여러 클래스에서사용해도 된다. Session 인스턴스가 생성될 때 매핑 정보도 함께 전달하므로 컴파

일된 형태로 모든 매핑 데이터를 가진다

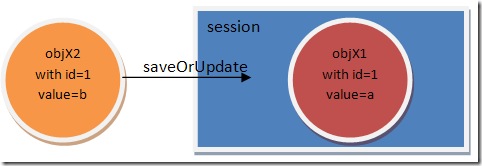
SessionFactory

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.

[](http://lh5.ggpht.com/_LFzrs9y4Am8/Sb9Tpi0bMvI/AAAAAAAAC4U/-TRotefhQmI/s1600-h/sessionObject%5b11%5d.jpg)

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.utility.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[] |
| MaterializedBlobType | BLOB | byte[] | materized\_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 name을 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.

@GenerateValue 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. 데  이터베이스에서 고유한 식별자를 제공하는 역할을 한다. |
| **GenerationType.SEQUENCE** | 몇몇 데이터베이스에서는 연속된 숫자에 관한 메커니즘을 제공하는데, 하이버  네이트에서는 일련번호를 사용한다. |
| **GenerationType.TABLE** | 다른 테이블의 고유한 컬럼 값에서 기본키를 생성하는데, 이 경우 TABLE 생성자  를 사용한다. 시퀀스seqeunce 방법에서는 strategy와 generator 속성을 모두 정의해야 한다. |

**GenerationType.AUTO**

|  |  |
| --- | --- |
| @Data @Entity public class UserGen\_AUTO {  @Id  @GeneratedValue(strategy = GenerationType.*AUTO*)  Integer seq;  String name;  String address;  Integer age; } | 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)  ) |
| 자동으로 hibernate\_sequence를 생성한다. | |
| ▼실행 | |
| 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(); |  |
| 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  (sequence\_name, next\_val)  values  (?,?)  update  USER\_ID\_TABLE  set  next\_val=?  where  next\_val=?  and sequence\_name=?  select  tbl.next\_val  from  USER\_ID\_TABLE tbl  where  tbl.sequence\_name=? for update    update  USER\_ID\_TABLE  set  next\_val=?  where  next\_val=?  and sequence\_name=?  insert  into  UserGen\_TABLE  (address, age, name, seq)  values  (?, ?, ?, ?)  13:35 TRACE o.h.type.descriptor.sql.BasicBinder - binding parameter [1] as [VARCHAR] - [null]  13:35 TRACE o.h.type.descriptor.sql.BasicBinder - binding parameter [2] as [INTEGER] - [null]  13:35 TRACE o.h.type.descriptor.sql.BasicBinder - binding parameter [3] as [VARCHAR] - [name]  13:35 TRACE o.h.type.descriptor.sql.BasicBinder - binding parameter [4] as [INTEGER] - [1] | | |

**IdentifierGenerator (커스텀 Generator)**

|  |
| --- |
| /\* 사용하는 방법  @GenericGenerator( name = "SEQ\_CLAS\_PART\_NUM", strategy = "com.nhis.ggij.api.cms.jpa.primary.SequenceIdentifierPrefixGenerator",  parameters = {@Parameter(name = "sequence\_name", value = "SEQ\_CLAS\_PART\_NUM"), @Parameter(name = "sequence\_prefix", value = "CP")}  )  @Id  @GeneratedValue(generator = "SEQ\_CLAS\_PART\_NUM", strategy = GenerationType.SEQUENCE)  @Column(name="CLAS\_PART\_NUM")  int clasPartNum; //NUMBER(22) 강습반 소속 번호 +  \*/  public class SequenceIdentifierPrefixGenerator implements IdentifierGenerator, Configurable {  Logger log = LoggerFactory.*getLogger*(this.getClass());  public static final String *SEQUENCE\_PREFIX* = "sequence\_prefix";   private String sequencePrefix;   private String sequenceCallSyntax;   @Override  public void configure(Type type, Properties params, ServiceRegistry serviceRegistry) throws MappingException {  final JdbcEnvironment jdbcEnvironment = serviceRegistry.getService(JdbcEnvironment.class);  final Dialect dialect = jdbcEnvironment.getDialect();   sequencePrefix = ConfigurationHelper.*getString*(  *SEQUENCE\_PREFIX*,  params,  "SEQ\_");   final String sequencePerEntitySuffix = ConfigurationHelper.*getString*(  SequenceStyleGenerator.*CONFIG\_SEQUENCE\_PER\_ENTITY\_SUFFIX*,  params,  SequenceStyleGenerator.*DEF\_SEQUENCE\_SUFFIX*);   final String defaultSequenceName = ConfigurationHelper.*getBoolean*(  SequenceStyleGenerator.*CONFIG\_PREFER\_SEQUENCE\_PER\_ENTITY*,  params,  false)  ? params.getProperty(*JPA\_ENTITY\_NAME*) + sequencePerEntitySuffix  : SequenceStyleGenerator.*DEF\_SEQUENCE\_NAME*;   sequenceCallSyntax = dialect.getSequenceNextValString(  ConfigurationHelper.*getString*(  SequenceStyleGenerator.*SEQUENCE\_PARAM*,  params,  defaultSequenceName));  }   @Override  public Serializable generate(SessionImplementor session, Object obj) {  if (obj instanceof Identifiable) {  Identifiable identifiable = (Identifiable) obj;  Serializable id = identifiable.getId();  if (id != null) {  return id;  }  }  //기본 하이버네이트 SQLQuery쓰면 중복 Casecad 처리된거 처리될때 StackoverFlowError뜬다 // long seqValue = ((Number) Session.class.cast(session) // .createSQLQuery(sequenceCallSyntax) // .uniqueResult()).longValue();  Connection connection = session.connection();  String result = null;  try {  PreparedStatement ps = connection.prepareStatement(sequenceCallSyntax);  log.debug(sequenceCallSyntax);  ResultSet rs = ps.executeQuery();  if (rs.next()) {  String seqValue = rs.getString(1);  result = sequencePrefix + seqValue;  }  }catch (Exception e){  throw new HibernateException("Unable to generate SequenceIdentifierPrefixGenerator Sequence",e);  }  return result; // return sequencePrefix + String.format("%011d%s", 0 ,seqValue);  } } |
| //회원정보 @Data @NoArgsConstructor @AllArgsConstructor @Entity public class TBGGIJ36Base implements Serializable{  @GenericGenerator( name = "SEQ\_USER\_NUM", strategy = "com.nhis.ggij.api.cms.jpa.primary.SequenceIdentifierGenerator",  parameters = {  @org.hibernate.annotations.Parameter(  name = "sequence\_name", value = "SEQ\_USER\_NUM"),  @org.hibernate.annotations.Parameter(  name = "sequence\_prefix", value = "UN"),  }  )   @Id  @GeneratedValue(generator = "SEQ\_USER\_NUM", strategy = GenerationType.*SEQUENCE*)  @Column(name="USER\_NUM")  String userNum; //VARCHAR2(20) 회원 번호 +  @Column(name="CNTR\_CD")  String cntrCd; //VARCHAR2(6) 센터 코드 +  @Column(name="HDQT\_AUTH\_YN") |
| Session s = primarySessionFactory.getCurrentSession(); Serializable r = s.save(tbggij36); s.flush(); |
| [VARCHAR] - [김미숙D]  [VARCHAR] - [777700]  [VARCHAR] - [UN20000000004]... |

복합 식별자 ID (KEY)

복합 아이디composite-id 식별자 설정과 관련된 세 가지 방법

1. @Embededdable

2. @EmbeddedId

3. @IdClass

|  |
| --- |
| ID 값으로 사용되는 클래스에서 구현해야될것들   1. Default Constructor() 2. hashCode() 3. equals() 4. implements Serializable |
| **tip : Lombok lib사용한다면 위 내용을 자동으로 생성해준다.(** <https://projectlombok.org/> **)** |

키선언 첫번째 방법 (@Embeddable)

|  |  |  |
| --- | --- | --- |
| @Data @Embeddable public class UserPK\_Embededdable implements Serializable{  String name;  Integer number; } | | @Data @Entity public class UserInfo{  @Id  UserPK\_Embededdable id;  String address;  Integer age; } |
| create table UserInfo (  name varchar(255) not null,  number integer not n  ull,  address varchar(255),  age integer,  primary key (name, number)  ) | | |
| Session session = getSessionFactory().getCurrentSession(); session.beginTransaction();  UserPK\_Embededdable pk = new UserPK\_Embededdable(); pk.setName("name"); pk.setNumber(128);  UserInfo\_Embededdable user = new UserInfo\_Embededdable(); user.setId(pk); user.setAddress("korea"); user.setAge(53);  session.save(user); session.getTransaction().commit(); | insert  into  UserInfo\_Embededdable  (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] | |
| //select  UserPK\_Embededdable userPK = new UserPK\_Embededdable(); UserPK\_IdClass();  userPK.setName("name");  userPK.setNumber(128);  UserInfo\_Embededdable userBydb = session.get(UserInfo\_Embededdable.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\_Embededdable userinfo\_e0\_  where  userinfo\_e0\_.name=?  and userinfo\_e0\_.number=?  binding parameter [1] as [VARCHAR] - [name]  binding parameter [2] as [INTEGER] - [128] | |
| get::UserInfo\_Embededdable(id=UserPK\_Embededdable(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\_Embededdable 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] | |
| //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] |
| 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 관계assocation와 관계relationship에 대한 이해는 필수입니다.

연관 관계에서 반드시 기억할 두 가지는 **다중성multiplicity**과 **방향성directionality**입니다

다중성

|  |  |  |
| --- | --- | --- |
| 일대일 1:1 | 한 테이블에서 각 레코드는 반드시 다른 테이블 의  레코드 한 개와 관계가 있다. 반대의 경우도 마찬가지다.  다른 테이블의 레코드는 0일 수도 있다. | 자동차 한 대는 오직 한 개의 엔진만 가진다. |
| 일대다  또는  다대일  1:N, N:1 | 한 테이블에서 각 레코드는 다른 테이블의 0개  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  0개 또는 그 이상의 레코드와 관계가 있다. |  |

방향성

Car와 Engine의 관계에서 Car의 속성을 질의해서 Engine을 찾아낼수있습니다. car -> engin

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 Car에 대한 참조를 제공하 고, Car 객체에는

Owner에 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 entityManager.detach ()  entityManager.clear () |
| CascadeType.MERGE | 엔티티에 업데이트 된 데이터가 있으면이 작업이 관계에 반영됩니다 | 엔티티가 갱신되고 트랜잭션이 완료되거나, entityManager.merge () |
| CascadeType.PERSIST | 새로운 엔티티가 데이터베이스에 유지되면이 조치가 관계에 반영됩니다. | 트랜잭션이 끝나거나, entityManager.persist () |
| CascadeType.REFRESH | 엔티티가 데이터베이스와 동기화 된 데이터를 가질 때이 조치가 반영됩니다 | entityManager.refresh () |
| CascadeType.REMOVE | 엔터티가 데이터베이스에서 삭제되면 행동이 관계에 반영 될 것입니다. | 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 ALL을 사용한다)** | | |

@OneToOne (FetchType.EAGER 기본값)

|  |  |
| --- | --- |
| 1. FK지정 | |
| @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 FK를건다 | |
| ▼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] |
|  |

|  |
| --- |
| @Entity  class Emp {  @OneToOne(orphanRemoval=true)      private Addr addr;  }  **Emp 엔티티가 삭제될 때 참조가 끊어진 연관된 Addr 엔티티도 삭제하라는 의미이며 DB에서도 삭제되는데 참조(연결)가 끊어진 Addr 객체는 DB에서도 삭제된다는 뜻이다.**    @Entity  class Emp {  @OneToOne(cascade=CascadeType.REMOVE)      private Addr addr;  }  **Emp 엔티티가 삭제될 때 연관된 Addr 엔티티도 삭제하라는 의미이며 DB에서도 삭제된다.**    @Entity  class Emp {  @OneToMany(orphanRemoval=true)      private List<Addr> addr;  }  **addresses 컬렉션에서 Addr 객체가 제거되는 경우 DB에서도 삭제하라는 의미.**  **orphanRemoval**은 JPA2.0 이상에서 지원하는 것으로 ORM 스펙, JPA 레벨에서의 정의이고 On Delete Cascade는 DBMS 레벨에서 작동되며 하는 일은 같다. orphanRemoval은 @OneToMany 연관에서 부모 엔티티의 컬렉션 등에서 자식 엔티티가 삭제될 때 참조가 끊어지므로 DB 레벨에서도 삭제되고 @OneToOne연관에서 엔티티가 삭제될 때 연관된 엔티티가 참조가 끊어지므로 DB에서 삭제된다. 즉 참조, 연결이 끊어진(Disconnected된) 엔티티를 같이 삭제하라는 의미로 Owner 객체와 참조가 끊어진 객체들을 정리할 때 유용하다.  **cascade=CascadeType.REMOVE**는 연결이 끊어진다고 해서 자동 삭제되는 것은 아니고 명시적으로 연관 엔티티가 삭제될 때 같이 삭제하라는 영속성 전이와 관련된 옵션이다.   반면 **On Delete Cascade**는 DB레벨에서 부모 테이블의 레코드가 삭제될 때 자식레코드도 같이 삭제하라는 의미이다. |

|  |  |  |
| --- | --- | --- |
| 1. PK를 FK지정 (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();   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(); | call next value for SEQ\_USER  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 as height3\_1\_0\_,  userinfo0\_.weight as weight4\_1\_0\_,  user1\_.USER\_SEQ as USER\_SEQ1\_0\_1\_,  user1\_.name as name2\_0\_1\_,  user1\_.password as 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 | |
|  | | |
| 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 ID로 해준다.)  또한 PrimaryKeyJoinColumn을 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 :** insert했을때 Entity는 관계설정이 되어있지 않아 로그가 찍히지 않는다 하지만 다시 SELECT해온 Entity는 방향성 설정이 되어 넘어온다 여기서 알수 있는것은 insert했을때 사용했던 Entity와 get해온 Entity는 다르다는것을 알수 있다. | |

@OneToMany, @ManyToOne

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

List

|  |  |
| --- | --- |
| ▼ **OneToMany조인하기 맵핑 테이블 만들어 조인하기 (FetchType.**LAZY **기본값)** | |
| @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 FKmnns1wp8mo27pxkbbwjl8nuun  foreign key (auths\_seq)  references Auth  alter table User\_Auth  add constraint FKn1dj4srm089e409sq8vkhogug  foreign key (User\_seq)  references User | |
|  |  |
| ▼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\_ADMIB");  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\_ADMIB]  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  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)); |
|  |
| 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))); | | |
|  | | |
| ▼**바로 OneToMany조인하기(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; } | | |
|  |  | | |
| ▼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. | | | |

|  |  |  |
| --- | --- | --- |
| ▼PK를 FK지정 (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\_ADMIB"); 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> | | |

|  |  |  |  |
| --- | --- | --- | --- |
| ▼PK를 FK지정 (맵핑 테이블 없음) GeneratedValue값으로 KEY값 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: "FKOU1XVFC0SWVOFR46KFD8KHY1N: 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 설정했기떄문에 각각 시퀀스 값으로 들어간다 여기서 FK제약조건이 걸려 오류가 난것이다. | | | |
| 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", nullable=false,updatable=false,insertable=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> | | | |
| //강습반소속회원정보 @Data @NoArgsConstructor @Entity @Table(name="TBGGIJ01") public class Tbggij01 extends Tbggij01Base {  @ManyToOne(cascade = CascadeType.*ALL*) @JoinColumn(name = "CNTR\_CD", referencedColumnName = "CNTR\_CD", nullable = false, updatable = false, insertable = false)  Tbggij02 tbggij02; }  joinColumn의 name은 현재Entity객체의 컬럼 referencedColumnName은 상대방 부모 테이블 컬럼 | | | |

**여러컬럼을 조인하기**

|  |
| --- |
| @OneToMany(cascade = CascadeType.*ALL*) @JoinColumns({  @JoinColumn(name = "RSCH\_KD\_CD", referencedColumnName = "RSCH\_KD\_CD"),  @JoinColumn(name = "REG\_SEQ\_NUM", referencedColumnName = "REG\_SEQ\_NUM"), }) List<TBGGIJ11> tbggij11s; |

방향성 갖자

|  |
| --- |
| 집고 넘어가기 1 : @MonyToOne 쪽에 @JoinColums 를 사용하여 자기자신 테이블 컬럼명을 지정할수도있다.  @ManyToOne(cascade = CascadeType.*ALL*) @JoinColumn(name="USER\_SEQ", nullable=false,updatable=false,insertable=false) User user; |

**조회조건 바로입력하기**

|  |
| --- |
| //메뉴 권한정보 @Data @NoArgsConstructor @AllArgsConstructor @Entity public class Tbggij26 extends Tbggij26Base{  @OneToOne(cascade=CascadeType.*ALL*, fetch = FetchType.*LAZY*)  @JoinColumn(name = "MENU\_NUM", nullable=false, updatable=false, insertable=false)  @WhereJoinTable(clause = "USE\_YN = 'Y'")  Tbggij05 tbggij05; } |

조회하기

|  |
| --- |
| **지연로딩 FetchType.LAZY (@oneToMany기본값)** |
| @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 (FetchType.EAGER 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*)  @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(); |  |
|  | |
| 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:N연결하기

|  |  |
| --- | --- |
| @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 FKxjfywgcu7nyi1lfdq0elefqx  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 inser와 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] | |

**optional**

조인에서 자식이 존재를 optional하게 설정해줄수 있다.

|  |
| --- |
| @JsonInclude(JsonInclude.Include.*NON\_NULL*) @OneToOne(optional = false, fetch = FetchType.*LAZY*) @JoinColumn(name = "USER\_NUM", referencedColumnName = "USER\_NUM", insertable = false, updatable = false) Tbggij33 tbggij33; //비상연락망 |
| 위에서 fetch모드가 LAZY이고, USER\_NUM이 PK이다 이런상황에서 optional=false를 해주지 않는다면  PK로 조인이 걸려있으므로 무조건 로딩을 다시한다 (여러번 쿼리가 날라간다) 이에 방지를 해주기위하여 optional=false를 해줘야한다.  의미하는걸로는 자식이 있을수도 있고 없을수도 있다 option이false 이라는것이다.즉 없을수도 있다는것이다. |

View

@SubSelect

|  |
| --- |
| //강습반소속회원정보 @Data @NoArgsConstructor @Entity   @Subselect("SELECT tbggij01.\* FROM TBGGIJ01 tbggij01, (select user\_num, max(tbggij01\_in.clas\_part\_num) as clas\_part\_num from TBGGIJ01 tbggij01\_in GROUP BY tbggij01\_in.USER\_NUM) tbggij01\_sub WHERE tbggij01.user\_num = tbggij01\_sub.user\_num and tbggij01.clas\_part\_num = tbggij01\_sub.clas\_part\_num") public class TBGGIJ01MaxClasPartNum implements EntityBase, Serializable {  @Column(name="CLAS\_PART\_NUM")  String clasPartNum; //NUMBER(22) 강습반 소속 번호 +  @Column(name="CNTR\_CD")  String cntrCd; //VARCHAR2(20) 센터 코드 +  @Id  @Column(name="USER\_NUM")  String userNum; //VARCHAR2(20) 회원 번호 +  @Column(name="CLAS\_NUM")  String clasNum; //VARCHAR2(20) 강습반 일련번호 +  @Column(name="REG\_EMP\_NUM")  String regEmpNum; //VARCHAR2(20) 등록 직원 아이디 +  @Column(name="REG\_DTTM")  String regDttm; //VARCHAR2(14) 등록 일시 +  @Column(name="UPD\_EMP\_NUM")  String updEmpNum; //VARCHAR2(20) 수정 직원 아이디 +  @Column(name="UPD\_DTTM")  String updDttm; //VARCHAR2(14) 수정 일시 +   @ManyToOne(cascade=CascadeType.*ALL*)  @JoinColumn(name = "CLAS\_NUM", referencedColumnName = "CLAS\_NUM", updatable=false, insertable=false)  TBGGIJ02 tbggij02; } |
| **SELECT** this\_.USER\_NUM **AS** USER\_NUM1\_0\_2\_, ...  this\_.USER\_NM **AS** USER\_NM34\_0\_2\_,  this\_.ZIPCD **AS** ZIPCD35\_0\_2\_,  tbggij02x2\_.CLAS\_NUM **AS** CLAS\_NUM1\_2\_0\_, ....  tbggij32s1\_.UPD\_EMP\_NUM **AS** UPD\_EMP\_NUM5\_32\_1\_,  tbggij32s1\_.USER\_NUM **AS** USER\_NUM6\_32\_1\_ **FROM** (**SELECT** DETAIL\_ADDR,  STAT\_CD, ...  PHONE\_NUM,  BIRTHDAY,  B.CLAS\_NUM, ...  B.REG\_EMP\_NUM,  **FROM** TBGGIJ36 **A**, TBGGIJ18 B  **WHERE A**.USER\_NUM = B.USER\_NUM) this\_ **INNER JOIN** TBGGIJ02 tbggij02x2\_  **ON** this\_.CLAS\_NUM = tbggij02x2\_.CLAS\_NUM  **LEFT OUTER JOIN** TBGGIJ32 tbggij32s1\_ **ON** this\_.USER\_NUM = tbggij32s1\_.USER\_NUM **WHERE** this\_.CNTR\_CD = ? |

캐싱

1차 캐시

Session 객체와 관련하여 트랜잭션이 보장되는 캐시다. 이는 세션의 수명이 유지되는 동안이나

컨버세이션conversation 내에서만 가능하다. 1차 캐시는 하이버네이트 프레임워크에서 기본으로 제공한다.

|  |
| --- |
| //팩토리 가져오기  SessionFactory factory = d.getSessionFactory();    //세션가져오기  Session session = factory.getCurrentSession();  //트랜젝션 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);  //트랜젝션 커밋  session.getTransaction().commit();    //트랜젝션이 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\_5wrx276k6vlwxjehoqh6h1rwa 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=?  Hibernate: update T\_AUTH set USER\_SEQ=? where FNC\_ID=?  Exception in thread "main" org.hibernate.SessionException: Session is closed! |
| 두 번째로 User 객체를 불렀을 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. |

2차 캐시

SessionFactory 클래스를 이용하여 전역에서 사용할 수 있다.

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

하이버네이트는 EhCache와 InfiniSpan 같은 오픈소스 캐시 라이브러리를 지원한다.

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

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

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

다음은 JBoss의 InifiniSpan을 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. 매  핑 정의 파일의 class 태그 안에 cache 속성을 확인해 보자. |

cache usage 속성

|  |  |
| --- | --- |
| transactional | 이 전략은 트랜잭션이 가능한 캐시를 지원하는 캐시 공급자를 위해 제공된다. 모  든 캐시 공급자가 트랜잭션이 가능한 캐싱 상품을 가지고 있지 않다는 점에 유의  하자. |
| read-only | 갱신할 필요가 없는 영속화 객체에 자주 접근한다면 read-only를 선택한다. 데  이터베이스를 거의 변경하지 않거나 전혀 변경하지 않는다면 이 옵션으로 성능  이 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 true로 지정한다.

코드에 한 가지를 더 추가해야 하는데, Query.setCacheable() 메소드를 호출해서 Query의

cacheable 속성을 true로 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  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] | | | | | | | |
|  | | | | | | | |
| **집고 넘어가기 : DTYPE 저것은 무엇일까 왜 들어가는걸까? 하이버네이트에서 자동으로 어떤 클래스에의해 INSERT가 되었는지 출처를 알수 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; } | | | | | | |  |
| ▼DTYPE컬럼을 값을 각각 클래스 별로 다르게 지정해보자 @DiscriminatorValue | | | | | | | |
| @Data @Entity @DiscriminatorValue(value="BANK") public class UserBankPrivacy extends User{  String password; } | @Data @Entity @DiscriminatorValue(value="BODY") public class UserBodyPrivacy extends User{  Integer age;  double weight;  double height; } | | | | @Data @Entity @DiscriminatorValue(value="OFFICE") public class UserOfficePrivacy extends User{  String address; } | | |
|  | | | | | | | |
| ▼DTYPE컬럼을 없에보자 (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)  )  또는  **@MappedSuperclass**  public class Parent {  // ...  }  @Entity  public class ParentEntity extends Parent {  // no code at all here  }  @Entity  public class Child extends Parent {  // additional fields and methods here  } | | | |
| ▼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 InheritanceType에는 TABLE\_PER\_CLASS와 JOINED 전략이 있다. Table-per-class 전략을 이용할  때 InheritanceType.TABLE\_PER\_CLASS 값으로 잘못 지정할 수도 있다. 반드시SINGLE\_TABLE만 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(); | |  |
|  | | |
|  | | |
| 집고 넘어가기 : 부모 Entity의 TABLE을 기준으로 자식 테이블 내용이 들어간다 userBank Entity를 넣을때 보면 부모Entity의 name값을 넣지 않아도 부모 시퀀스가 들어가고 난뒤 자식Entity의 테이블에 내용이 들어간다  부모SEQ ID컬럼과 자식들의 SEQ컬럼은 자동 생성된다.(부모 ID Column 이름으로) | | |

@Table-per-Concrete-Class

부모 클래스의 모든 속성이 자식 클래스와 관련된 테이블에 복사된다. (자주 쓰이진 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; } |
|  | | |

**Entity 상속 처리**

**@MappedSuperclass**

|  |  |
| --- | --- |
| @Data @NoArgsConstructor @AllArgsConstructor @MappedSuperclass public class TBGGIJ36Base implements EntityBase, Serializable{  @GenericGenerator( name = "SEQ\_USER\_NUM", strategy = "com.nhis.ggij.api.cms.jpa.primary.SequenceIdentifierGenerator",  parameters = {@Parameter(name = "sequence\_name", value = "SEQ\_USER\_NUM"), @Parameter(name = "sequence\_prefix", value = "UN")}  )   @Id  @GeneratedValue(generator = "SEQ\_USER\_NUM", strategy = GenerationType.*SEQUENCE*)  @Column(name="USER\_NUM")  String userNum; //VARCHAR2(20) 회원 번호 +  ...  @PrePersist @PreUpdate  public void prepersistOrUpdate() {  String dt = DateUtil.*getDate*("yyyyMMddHHmmss");  setRegDttm(dt);  setUpdDttm(dt);  }  } | //회원정보 @Data @NoArgsConstructor @Entity @Table(name="TBGGIJ36") public class TBGGIJ36Origin extends TBGGIJ36Base{} |
| //회원정보 @Data @NoArgsConstructor @Entity @Table(name="TBGGIJ36") public class TBGGIJ36 extends TBGGIJ36Base{  //무산소운동결과정보 join  @OneToMany(cascade = CascadeType.*ALL*) @JoinColumn(name = "USER\_NUM", referencedColumnName = "USER\_NUM")  List<TBGGIJ45> tbggij45s; } |
|  | |

**@GeneratedValue를 자식 Entity쪽에 위임하기**

|  |
| --- |
| @Data @AllArgsConstructor @NoArgsConstructor @MappedSuperclass @IdClass(T29Id.class) public class T29Base extends T29BaseBody implements Serializable,T29K {  @GeneratedValue(generator = "SEQ\_EMP\_NUM", strategy = GenerationType.*SEQUENCE*)  @Id  @Column(name="EMP\_NUM")  String empNum; //VARCHAR2(7) 직원 번호 + |
| @IdClass(T29Id.class) @GenericGenerator(  name = "SEQ\_EMP\_NUM", strategy = "com.omnicns.java.db.hibernate.id.SequenceIdentifierPrefixGenerator",  parameters = {@org.hibernate.annotations.Parameter(name = "sequence\_name", value = "SEQ\_CLAS\_PART\_NUM"), @org.hibernate.annotations.Parameter(name = "sequence\_prefix", value = "ES")}  )public class T29 extends T29Base implements java.io.Serializable{  ...  class위쪽에다가 지정해줘야한다. |

**@AttributeOverride 처리**

|  |
| --- |
| @Data @AllArgsConstructor @NoArgsConstructor @MappedSuperclass @IdClass(T29Id.class) public class T29Base extends T29BaseBody implements Serializable,T29K {  @Id  String empNum; //VARCHAR2(7) 직원 번호 +  String cntrCd; //VARCHAR2(6) 센터 코드 + |
| @Entity @Table(name = "TBGGIJ29") @AttributeOverrides({  @AttributeOverride(name = "empNum", column = @Column(name="EMP\_NUM")),  @AttributeOverride(name = "cntrCd", column = @Column(name="CNTR\_CD")),  @AttributeOverride(name = "grpCd", column = @Column(name="GRP\_CD")),  @AttributeOverride(name = "empId", column = @Column(name="EMP\_ID")), }) public class T29 extends T29Base implements java.io.Serializable{ |
| Entity자식에서 컬럼맵핑을 다시 재정의 한다 |
|  |

**@Id 재정의**

|  |
| --- |
| Entity자식에서 @Id 재정의를 하기위하여 @IdClass를 Entity에 다시 선언해야한다. |

하이버네이트 질의어

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

HQL은 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("guro1");  session.save(office);   user = new User();  user.setName("nanme2");  user.setPassword("pwd2");  session.save(user);  office = new Office();  office.setSeq(user.getSeq());  office.setAddr("guro2");  session.save(office);  session.flush();  session.clear();   Query query = session.createQuery("from User");  List list = query.getResultList();  list.forEach(it->*log*.debug("-->"+it.toString()));   session.getTransaction().commit(); | |  |
| 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;  } }  필드를받을수 있는 생성자를 생성한다 | 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  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"); | | |

**Entity 넘어온 컬럼값만 업데이트 하는 쿼리만들기**

|  |
| --- |
| public Serializable userUpdate(String userNum, Map<String, Object> user) {  Session s = primarySessionFactory.getCurrentSession();  String query = "update Tbggij36Single tbggij36 set ";  query += user.entrySet().stream().filter(fit->null!=fit.getValue()&&!"null".equals(fit.getValue())).map(it->"tbggij36."+it.getKey()+" = :"+it.getKey()).collect(Collectors.*joining*(" , "));  query += " where tbggij36.userNum = :userNum";  Query q = s.createQuery(query);  q.setParameter("userNum",userNum);  user.entrySet().stream().forEach(it->q.setParameter(it.getKey(),it.getValue()));  Serializable r = q.executeUpdate();  s.flush();  return r; } |

@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)  ) |  |
| ▼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] | |

@ElementCollection

Enum사용하기 @Enumerated

별도의 Entity를 만들지 않고 Collection을 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 |  | |
| ▼테이블명, FK컬럼 명바꾸기. | | |
| @ElementCollection @CollectionTable(name="CAR\_SET",joinColumns=@JoinColumn(name="CAR\_SET\_SEQ"))  Set<Car> cars\_set;   @ElementCollection @CollectionTable(name="CAR\_ENUM",joinColumns=@JoinColumn(name="CAR\_ENUM\_SEQ"))  @Enumerated(EnumType.*STRING*)  List<CAR\_TYPE> cars\_enum; |  | |

@Temproal

|  |
| --- |
| @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 페이지네이션Pagination 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); //30번째 로우부터 query.setMaxResults(5); //5개를 가져와라 List<User> users = query.getResultList(); users.stream().forEach(it->*log*.debug("-->"+it));  *log*.debug("-----------");  query = session.createQuery("from User"); query.setFirstResult(40); //40번째 로우부터 query.setMaxResults(5); //5개를 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 limt를 걸지 rownum을걸지 이것또한 hibernate가 알아서 해준다 |

Criteria

하이버네이트에서는 criteria를 도입하여 필터링의 또 다른 방법을 제공합니다.

Criteria와 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 | |

Criteria에서 EAGER로 list가져올때 복수개의 Entity가 나올수 있다

이에따른 조치로는 2가지 방법이 있다

1.

|  |
| --- |
| @OneToMany  @Fetch(FetchMode.SELECT) //http://stackoverflow.com/questions/19431021/hibernate-and-criteria-that-return-the-same-items-multiple-times  private Collection<Authority> authorities; |

2.

|  |
| --- |
| Criteria crit = currentSession.createCriteria(User.class);  crit.setResultTransformer(Criteria.DISTINCT\_ROOT\_ENTITY);//조인 쿼리가 날라간다 |

강제로 지연로등 사용 하는 방법 : crit.setFetchMode("authorities", FetchMode.JOIN); //강제로 지연로딩 사용안함

distinct

|  |
| --- |
| criteria.setProjection(Projections.distinct(Projections.property("id")));  criteria.setProjection(Projections.distinct(Projections.id()));  criteria.setResultTransformer(Transformers.aliasToBean(UserStateTransformer.class));  criteria.setResultTransformer(Criteria.DISTINCT\_ROOT\_ENTITY);  criteria.setProjection(  Projections.distinct(  Projections.projectionList()  .add(Projections.id())  .add(Projections.property("the property that you want to ordered by"))  )  ); |

조인

|  |
| --- |
| Session s = primarySessionFactory.getCurrentSession(); Criteria crit = s.createCriteria(TBGGIJ02.class,"tbggij02"); crit.createCriteria("tbggij02.tbggij18","tbggij18", JoinType.*LEFT\_OUTER\_JOIN*); crit.createCriteria("tbggij02.tbggij29","tbggij29", JoinType.*LEFT\_OUTER\_JOIN*); return crit.list(); |

Order By

|  |
| --- |
| crit.addOrder(Order.*desc*("tbggij18") |

Group By

|  |
| --- |
| Criteria crit = s.createCriteria(TBGGIJ38.class); crit.add(Restrictions.*eq*("cntrCd", cntrCd)); crit .setProjection( Projections.*projectionList*()  .add( Projections.*count*("userNum"), "count")  .add( Projections.*groupProperty*("atndDt"), "atndDt" ) );  crit.setResultTransformer(Criteria.*ALIAS\_TO\_ENTITY\_MAP*); |

ResultTransformer

|  |
| --- |
| crit.setResultTransformer(Criteria.*ALIAS\_TO\_ENTITY\_MAP*); |

네임드 쿼리

클래스 레벨에서 엔티티의 쿼리를 사용하기 위해 @NamedQuery 어노테이션를 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->*log*.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->*log*.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>  <https://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"); | |

Query

|  |
| --- |
| Query q = s.createQuery("SELECT a.userNum, a.userNm, SUM(b.exTime)\*COUNT(c.exResultNum) as val from TBGGIJ36 a left join a.tbggij44s as b left join a.tbggij45s as c WHERE a.cntrCd=:cntrCd GROUP BY a.userNum, a.userNm order by (SUM(b.exTime)\*COUNT(c.exResultNum)) DESC"); |

Query Fect 처리

Query q = s.createQuery("SELECT a from TBGGIJ36 a left join fetch a.tbggij44s as b

left join fetch a.tbggij45s as c ");

**ResultTransformer**

|  |
| --- |
| Query q = s.createQuery("select a from TBGGIJ02 as a , TBGGIJ01 as b where a.clasSeq = b.clasSeq");  q.setResultTransformer(Criteria.*DISTINCT\_ROOT\_ENTITY*);  q.setResultTransformer(Transformers.*ALIAS\_TO\_ENTITY\_MAP*); Map<String,?> o = (Map<String, ?>) q.uniqueResult();  ModelMapper modelMapper = new ModelMapper(); TBGGIJ01Base orderDTO = modelMapper.map(o, TBGGIJ01Base.class); |

네이티브 쿼리

네이티브 쿼리를 실행하는 기능도 제공한다

HQL이 사용하기 간편한데 왜 네이티브 SQL을 ???

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->*log*.debug("-->"+it.getName()));  session.getTransaction().commit(); |
| SELECT  \*  FROM  USER |
| ▼ XML맵핑 |
| <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> |

**ResultTransformer**

|  |
| --- |
| .createSQLQuery( "select e.id as id,e.first\_name as firstName,e.password as password from xxxxxx")  .addScalar("id",StandardBasicTypes.INTEGER )  .addScalar("firstName",StandardBasicTypes.STRING )  .addScalar("password",StandardBasicTypes.STRING )  .setResultTransformer(Transformers.aliasToBean(Employee.class))  .list(); |

**인텁셉터 처럼 사용할수 있다**

|  |
| --- |
| @PrePersist public void prepersist() {  System.*out*.println("prePersist()"); }   @PreUpdate public void preupdate() {  System.*out*.println("PreUpdate."); }  @PostLoad public void postload() {  System.*out*.println("postload."); }  **------**  @PrePersist @PreUpdate public void prepersistOrUpdate() {  String dt = DateUtil.*getDate*("yyyyMMdd");  setRegDttm(dt);  setUpdDttm(dt); } |

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->*log*.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("];");  *log*.debug("Groovy Template-->"+t);  engine.put("template", template);  return (String)engine.eval(t + " template.make(binding).toString();"); } |

**스프링 데이터 JPA 레퍼런스 번역**

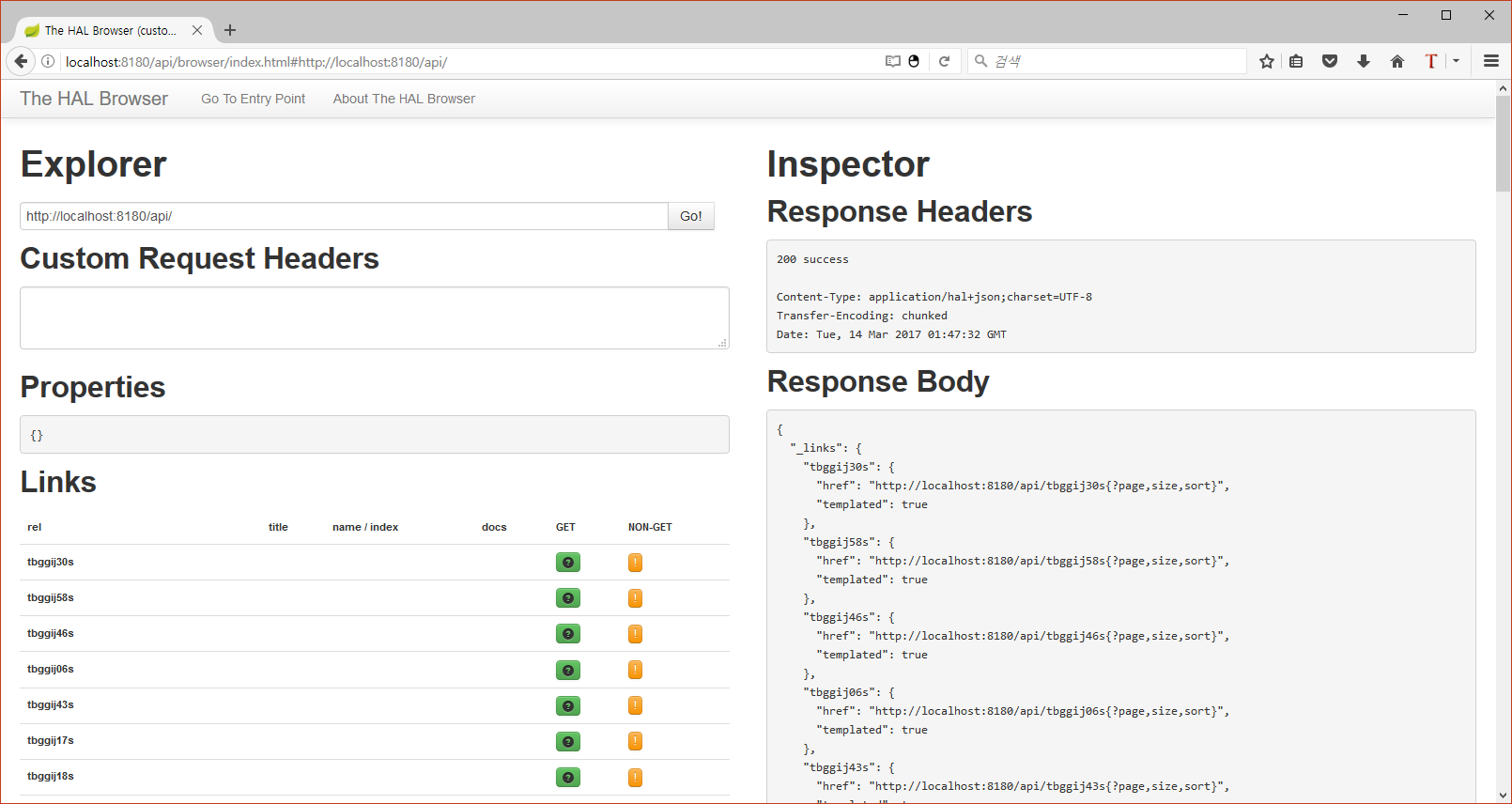
<http://arahansa.github.io/docs_spring/jpa.html#jpa.entity-graph>

|  |
| --- |
| @Repository @RepositoryRestResource public interface Tbggij15Repository extends JpaRepository<Tbggij15, String> {  } |

**스프링REST사용하기**

|  |
| --- |
| //http://docs.spring.io/spring-data/rest/docs/current/reference/html/#\_the\_hal\_browser //배포할때 아래 hal browser 주석처리 해주세요 같이 배포되면 안됩니다. compile 'org.springframework.data:spring-data-rest-hal-browser' //tools compile('org.springframework.boot:spring-boot-starter-data-rest') //rest http method to automatic generation |

**hal Browser는 Rest를 테스트 및 현재 상황을 명확하게 알수 있게하는 dev tool이다.**

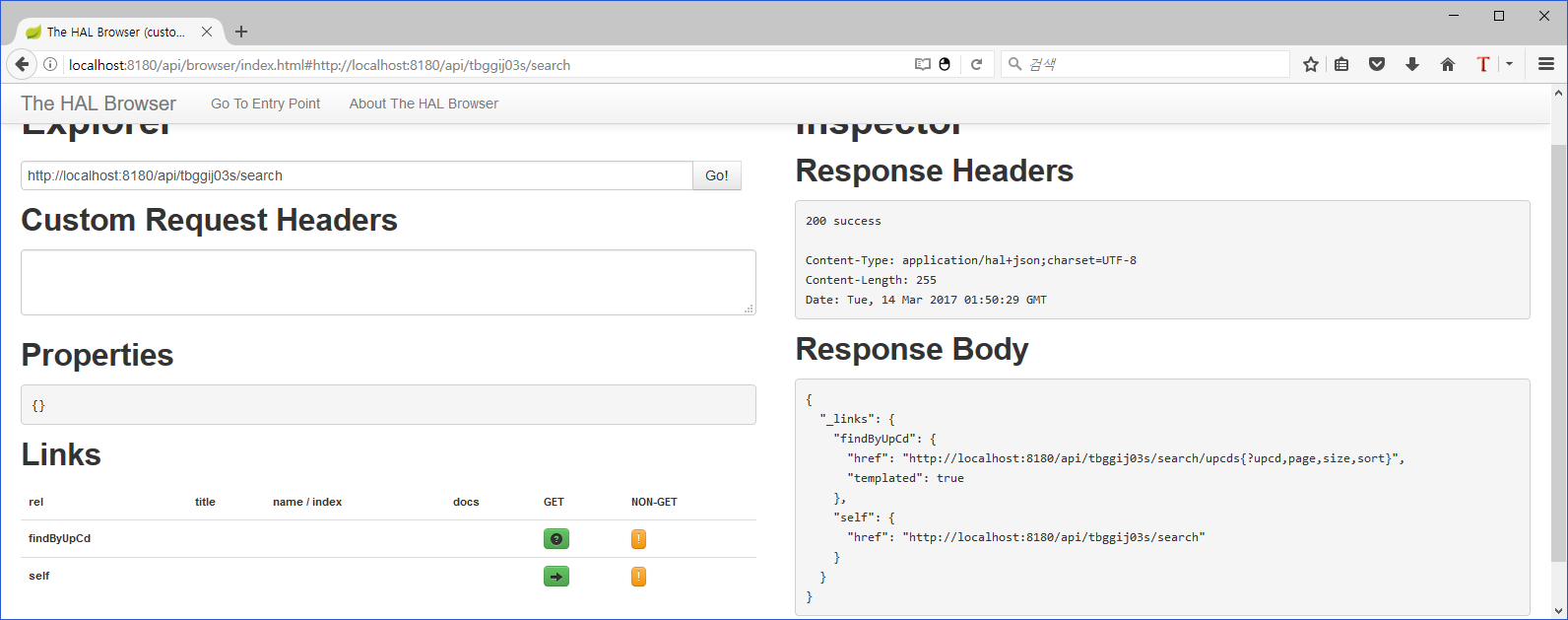


**Rest**

|  |
| --- |
| @Repository @RepositoryRestResource public interface Tbggij03Repository extends JpaRepository<Tbggij03, String> {   @RestResource(path = "upcds")  Page<Tbggij03> findByUpCd(@Param("upcd") String upcd, Pageable pageable);  } |

**위처럼 mapping을 시켜주면 tbggij03s/search/upcds uri로 처리하면된다.**

**search는 기본적으로 붙는 것이다.**



**기본적으로 Entity 이름명으로 Entitiys 로 uri가 지정되는데. 이것을 바꾸고자 할때에는**

|  |
| --- |
| @Repository @RepositoryRestResource(path = "users") public interface UserRepository extends JpaRepository<User, String> {  } |

**처럼 하면된다.**