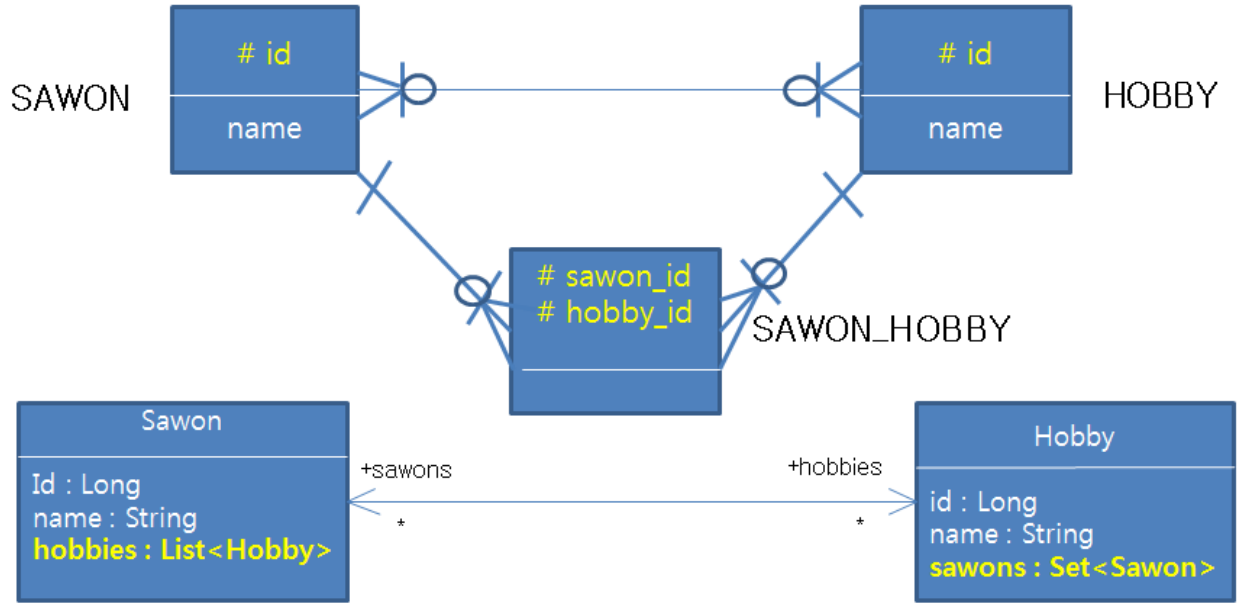
M : N 연관관계(다 : 다) 실습

- 사원(Sawon), 취미(Hobby)는 다 : 다 관계이다. 사원은 여러 취미를 가질 수 있고, 하나의 취미 역시 여러 사원에 할당 될 수 있기 때문이다. 보통 관계형 DB에서는 다 : 다 관계는 1 : 다, 다 : 1 로 나누어서 풀게 된다. 그러나 객체에서는 보통 2개의 객체로 다 : 다 관계를 만드는데, 사원에서 취미를 컬렉션에 넣어 접근 가능하고, 반대로 취미에서도 사원들을 컬렉션에 넣어 접근하면 양쪽에서 접근이 가능하다.
- 다 : 다 매핑을 위해 @ManyToMany 어노테이션을 사용한다.
- @ManyToMany 어노테이션에 mappedBy속성을 사용하여 연관관계의 주인을 지정한다. (mappedBy가 없는 곳이 Owning Side 이다)



[예제]

STS -> Spring Stater Project ,name : manytomany
다음화면에서 SQL -> JPA, MySQL 선택

http://ojc.asia/bbs/board.php?bo_table=LecSpring&wr_id=524
(마리아 DB 설치는 위 URL에서 참조)

src/main/resources/application.properties

```
spring.datasource.platform=mysql
spring.datasource.url=jdbc:mysql://localhost/manytomany?createDatabaseIfNotExist=true
```

```
spring.datasource.username=root
spring.datasource.password=1111
spring.datasource.driver-class-name=com.mysql.jdbc.Driver
spring.datasource.sql-script-encoding=UTF-8
spring.jpa.hibernate.ddl-auto=create
spring.jpa.show-sql=true
```

demo.model.Sawon.java

```
package demo.model;

import java.util.List;
import java.util.Set;

import javax.persistence.CascadeType;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.Id;
import javax.persistence.JoinColumn;
import javax.persistence.JoinTable;
import javax.persistence.ManyToMany;
import javax.persistence.OrderColumn;

@Entity
public class Sawon {
	private Long id;
	private String name;
	private List<Hobby> hobbies;

	public Sawon(String name, List<Hobby> hobby) {
		this.name = name;
		this.hobbies = hobby;
	}

	public Sawon(Long id, String name, List<Hobby> hobbies) {
		super();
		this.id = id;
		this.name = name;
		this.hobbies = hobbies;
```

```
	}

	public Sawon() {
	}

	@Id
	@GeneratedValue
	public Long getId() {
		return id;
	}

	public void setId(Long id) {
		this.id = id;
	}

	public String getName() {
		return name;
	}

	public void setName(String name) {
		this.name = name;
	}

	@ManyToMany(cascade = {CascadeType.MERGE, CascadeType.PERSIST,
CascadeType.REFRESH}) @JoinTable(name = "sawon_hobby",
	joinColumns = @JoinColumn(name = "sawon_id", referencedColumnName = "id") ,
	inverseJoinColumns = @JoinColumn(name = "hobby_id", referencedColumnName =
"id") )
	@OrderColumn(name="hobby_order")  //Sawon_Hobby에 hobby_order칼럼 생성
	public List<Hobby> getHobbies() {
		return hobbies;
	}

	public void setHobbies(List<Hobby> hobbies) {
		this.hobbies = hobbies;
	}

	@Override
	public String toString() {
```

```
                String result = String.format("Sawon [id=%d, name='%s']%n", id, name);
                if (hobbies != null) {
                        for (Hobby hobby : hobbies) {
                                result += String.format("Hobby[id=%d, name='%s']%n",
hobby.getId(), hobby.getName());
                        }
                }
                return result;
        }

}
```

demo.model.Hobby.java

```
@Entity
public class Hobby {
        private Long id;
        private String name;
        private Set<Sawon> sawons;
        public Hobby() {}
        public Hobby(String name) {
                this.name = name;
        }

        public Hobby(Long id, String name) {
                this.id = id;   this.name = name;
        }
        public Hobby(Long id, String name, Set<Sawon> sawons) {
                this.id = id;
                this.name = name;
                this.sawons = sawons;
        }

        @Id   @GeneratedValue
        public Long getId() {   return id; }
        public void setId(Long id) {   this.id = id;   }
        public String getName() {   return name;   }
        public void setName(String name)
        {
            this.name = name;
```

```
        }

        @ManyToMany(mappedBy = "hobbies")
        public Set<Sawon> getSawons() {
                return sawons;
        }

        public void setSawons(Set<Sawon> sawons) {
            this.sawons = sawons;
        }
}
```

demo.repository.SawonRepository.java

```
package demo.repository;
import org.springframework.data.jpa.repository.JpaRepository;
import demo.model.Sawon;
public interface SawonRepository extends JpaRepository<Sawon, Long>{
}
```

demo.repository.HobbyRepository.java

```
package demo.repository;
import org.springframework.data.jpa.repository.JpaRepository;
import demo.model.Hobby;
public interface HobbyRepository extends JpaRepository<Hobby, Long>{
}
```

demo.ManytomanyApplication.java

```
package demo;

import java.util.ArrayList;
import java.util.HashSet;
import java.util.List;

import javax.persistence.EntityManager;
import javax.persistence.PersistenceContext;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
```

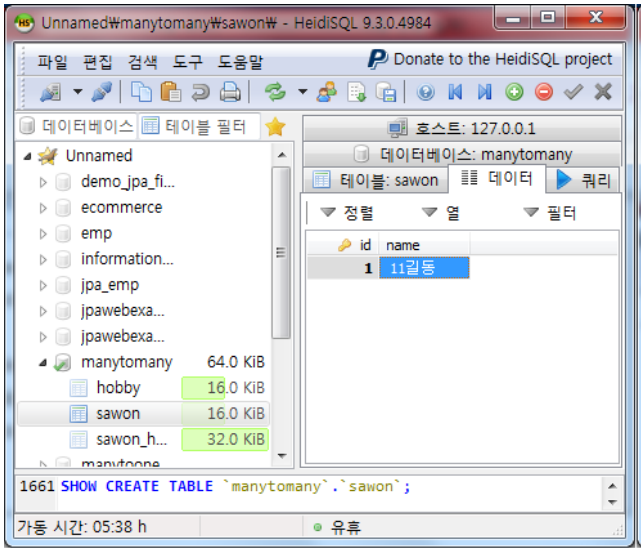
```
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.CommandLineRunner;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.transaction.annotation.Transactional;

import demo.model.Hobby;
import demo.model.Sawon;
import demo.repository.HobbyRepository;
import demo.repository.SawonRepository;

@SpringBootApplication
public class ManytomanyApplication implements CommandLineRunner {

	private static final Logger logger =
				LoggerFactory.getLogger(ManytomanyApplication.class);

	public static void main(String[] args) {
		SpringApplication.run(ManytomanyApplication.class, args);
	}


	@Autowired
    SawonRepository sawonRepository;

	@Autowired
    HobbyRepository hobbyRepository;

	@Transactional
	public void run(String...args) {
		save();			//저장
		sawonRepository.flush();
		deleteSawon();		//삭제
		sawonRepository.flush();
		updateSawon();		//수정
		sawonRepository.flush();
		deleteSawonHobby();	//삭제
	}
```

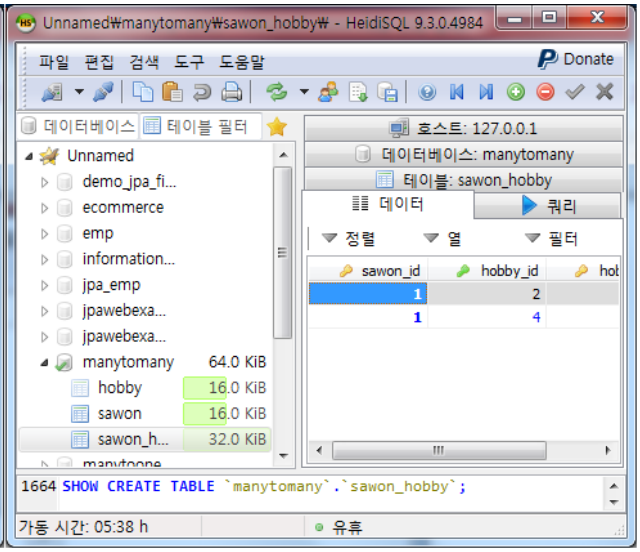
```
	void save() {
		Sawon s1 = new Sawon("1길동", new ArrayList() {
			{
				add(new Hobby("취미1"));
				add(new Hobby("취미2"));

			}
		});

		Sawon s2 = new Sawon("2길동", new ArrayList() {
			{
				add(new Hobby("취미3"));

			}
		});

		//-------------------------------------------------------------
		// Sawon 테이블에는 [1, 1길동],[2, 2길동] 이 있고
		// Hobby 테이블에는 [1, 취미1],[2, 취미2],[3, 취미3]
		// Sawon_Hobbby 테이블에는 트랜잭션 종료시점(커밋)시점에
		// [1, 1], [1, 2], [2, 3] 형태로 데이터가 삽입된다. 로그를 확인하자.
		// 만약 트랜잭션 도중에 Sawon 데이터가 삭제된다면 트랜잭션 커밋시점에
		// Sawon_Hobby 테이블에 insert 하므로 SQL로그는 확인되지 않는다.
		// 다른 트랜잭션이라면 insert 로그가 남는다.
		// insert into sawon (name) values (?)
		// insert into hobby (name) values (?)
		// insert into hobby (name) values (?)
		// insert into sawon (name) values (?)
		// insert into hobby (name) values (?)
		//-------------------------------------------------------------
		sawonRepository.save(new HashSet<Sawon>() {
			{
				add(s1);
				add(s2);
			}
		});

		// select sawon0_.id as id1_1_, sawon0_.name as name2_1_ from sawon
sawon0_
```

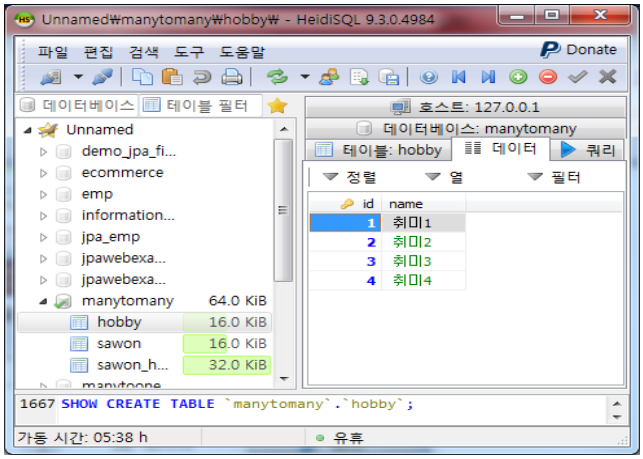
```
		//Sawon[id=1,name='1길동']Hobby[id=1,name='취미1']Hobby[id=2,name='취미2']
		// Sawon [id=2, name='2길동'] Hobby[id=3, name='취미3']
		for(Sawon s : sawonRepository.findAll()) {
			logger.info(s.toString());
		}

		// insert into sawon_hobby (sawon_id, hobby_order, hobby_id) values (?, ?, ?)
		// insert into sawon_hobby (sawon_id, hobby_order, hobby_id) values (?, ?, ?)
		// insert into sawon_hobby (sawon_id, hobby_order, hobby_id) values (?, ?, ?)
	}

	void deleteSawon() {
		//---------------------------------------------------------------------
		// 2번사원 검색 후 삭제
		// Sawon 및 Sawon_Hobby에서 2번 사원의 데이터가 삭제되고
		// 만약 CascadeType.DELETE라면 Hobby 테이블에서도 2번사원 취미[3, 취미3]은
		// 1번사원에 없으므로 삭제된다. (다른 사원이 사용하는 취미라면 삭제안됨)
		// 현재 CascadeType.DELETE는 빠져있다. 그러므로 Hobby에서 "취미3"은 삭제안됨
		// delete from sawon_hobby where sawon_id=?
		// delete from sawon where id=?
		//---------------------------------------------------------------------
		Sawon s2 = sawonRepository.findOne(2L);
		logger.info("삭제될 사원 => " + s2.toString());
		sawonRepository.delete(s2);
	}

	void updateSawon() {
		//-----------------------------------------------------
		// 1번 사원 검색 후 수정,먼저 영속성컨텍스트에서 1번사원을 로딩
		// "취미4"는 Hobby 테이블에 [4, 취미4]로 저장되고
		// 1번사원의 이름을 "11길동"으로 수정 후
		// Sawon_Hobby 테이블에 [1, 1, 4]가 insert 된다.
		// insert into hobby (name) values (?)
		// update sawon set name=? where id=?
		// insert into sawon_hobby (sawon_id, hobby_order, hobby_id) values (?, ?, ?)
		//-----------------------------------------------------
```

```
                Sawon s1 = sawonRepository.findOne(1L);
                logger.info("수정될 사원 => " +s1.toString());
                s1.getHobbies().add(new Hobby("취미4"));
                s1.setName("11길동");
                sawonRepository.save(s1);
        }

        void deleteSawonHobby() {
                //-------------------------------------------------------------
                // Hobby에서 [1, 취미1]을 검색 후 로딩
                // 아래처럼 삭제하는 경우 Sawon_Hobby에서 "1번사원"의 "취미1"만 삭제된다.
                // Hobby 테이블에서는 "취미1"이 삭제되지 않는다.
                // delete from sawon_hobby where sawon_id=? and hobby_order=?
                // update sawon_hobby set hobby_id=? where sawon_id=? and hobby_order=?
                // update sawon_hobby set hobby_id=? where sawon_id=? and hobby_order=?
                //-------------------------------------------------------------
                Sawon s1 = sawonRepository.findOne(1L);
                Hobby h1 = hobbyRepository.findOne(1L);

                if (s1.getHobbies().contains(h1))
                    s1.getHobbies().remove(h1);
                sawonRepository.save(s1);
        }
}
```

데이터 확인하기







[실행 결과]

alter table sawon_hobby drop foreign key FK_p9uwrr1nddt9pgue7xqmey4cr

alter table sawon_hobby drop foreign key FK_qqk3c8ippcim47nrhxyislbtl

drop table if exists hobby

drop table if exists sawon

drop table if exists sawon_hobby

create table hobby (id bigint not null auto_increment, name varchar(255), primary key (id))

create table sawon (id bigint not null auto_increment, name varchar(255), primary key (id))

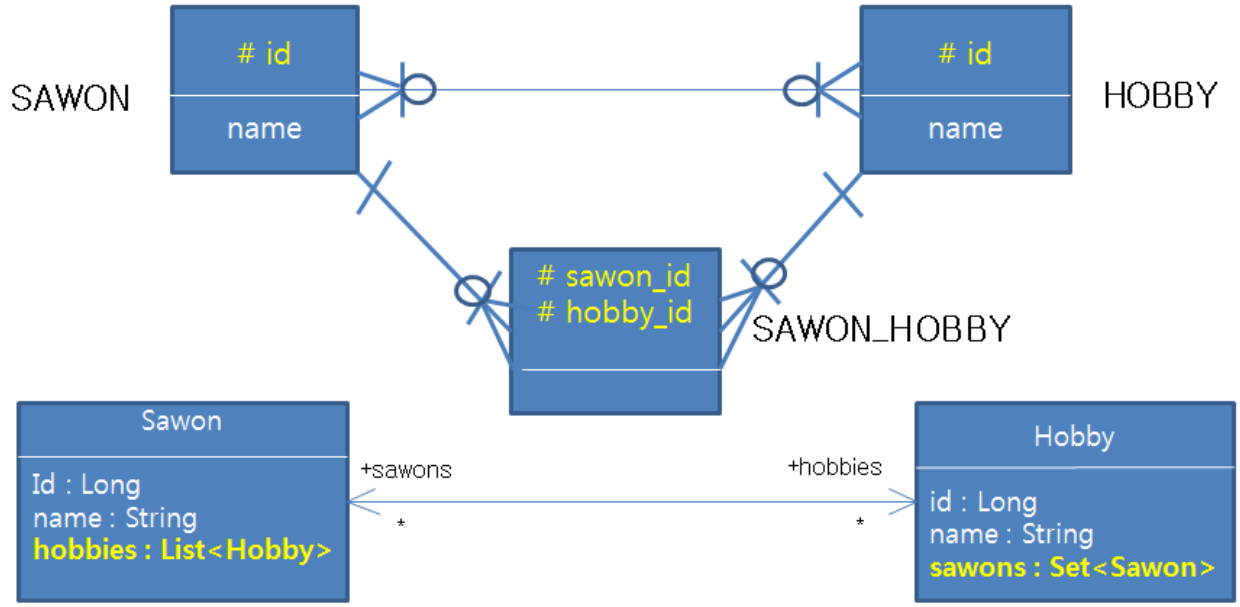
create table sawon_hobby (sawon_id bigint not null, hobby_id bigint not null, **hobby_order** integer not null, primary key (sawon_id, hobby_order))

alter table sawon_hobby add constraint FK_p9uwrr1nddt9pgue7xqmey4cr foreign key (hobby_id) references hobby (id)

alter table sawon_hobby add constraint FK_qqk3c8ippcim47nrhxyislbtl foreign key (sawon_id) references sawon (id)

```
insert into sawon (name) values (?)
insert into hobby (name) values (?)
insert into hobby (name) values (?)
insert into sawon (name) values (?)
insert into hobby (name) values (?)

select sawon0_.id as id1_1_, sawon0_.name as name2_1_ from sawon sawon0_
Sawon [id=1, name='1길동'] Hobby[id=1, name='취미1'] Hobby[id=2, name='취미2']
Sawon [id=2, name='2길동'] Hobby[id=3, name='취미3']

insert into sawon_hobby (sawon_id, hobby_order, hobby_id) values (?, ?, ?)
insert into sawon_hobby (sawon_id, hobby_order, hobby_id) values (?, ?, ?)
insert into sawon_hobby (sawon_id, hobby_order, hobby_id) values (?, ?, ?)

삭제될 사원 => Sawon [id=2, name='2길동']
Hobby[id=3, name='취미3']
delete from sawon_hobby where sawon_id=?
delete from sawon where id=?

수정될 사원 => Sawon [id=1, name='1길동']
Hobby[id=1, name='취미1']
Hobby[id=2, name='취미2']

insert into hobby (name) values (?)
update sawon set name=? where id=?
insert into sawon_hobby (sawon_id, hobby_order, hobby_id) values (?, ?, ?)

delete from sawon_hobby where sawon_id=? and hobby_order=?
update sawon_hobby set hobby_id=? where sawon_id=? and hobby_order=?
update sawon_hobby set hobby_id=? where sawon_id=? and hobby_order=?
```

M : N(다 : 다) 관계에서 List 컬렉션에 @OrderColumn 사용하기



위 실습의 마지막 실행메소드(deleteSawonHobby)의 로그를 보면 delete 쿼리 하나에 update 쿼리 2개가 실행됨을 알 수 있다.

```
delete from sawon_hobby where sawon_id=? and hobby_order=?
update sawon_hobby set hobby_id=? where sawon_id=? and hobby_order=?
update sawon_hobby set hobby_id=? where sawon_id=? and hobby_order=?
```

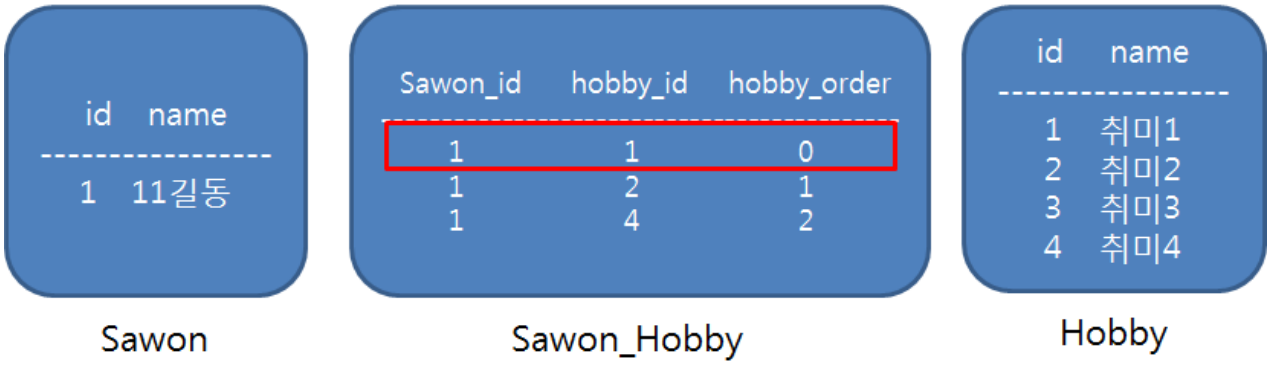
자바코드는 Sawon쪽의 hobbies List컬렉션에서 1번취미를 제거하는 내용이다. 영속성 전이로 인해 Sawon 컬렉션에서 제거될 때 Sawon_Hobby에서도 1번사원의 "취미1"도 삭제되는데 쿼리문은 위처럼 3개가 동작된다.

```
//deleteSawonHoby() 메소드의 내용
Sawon s1 = sawonRepository.findOne(1L);
Hobby h1 = hobbyRepository.findOne(1L);

if (s1.getHobbies().contains(h1))
    s1.getHobbies().remove(h1);
sawonRepository.save(s1);

//Sawon 테이블쪽의 컬렉션 필드 매핑은 다음과 같다.
@ManyToMany(cascade = {CascadeType.MERGE, CascadeType.PERSIST,
```

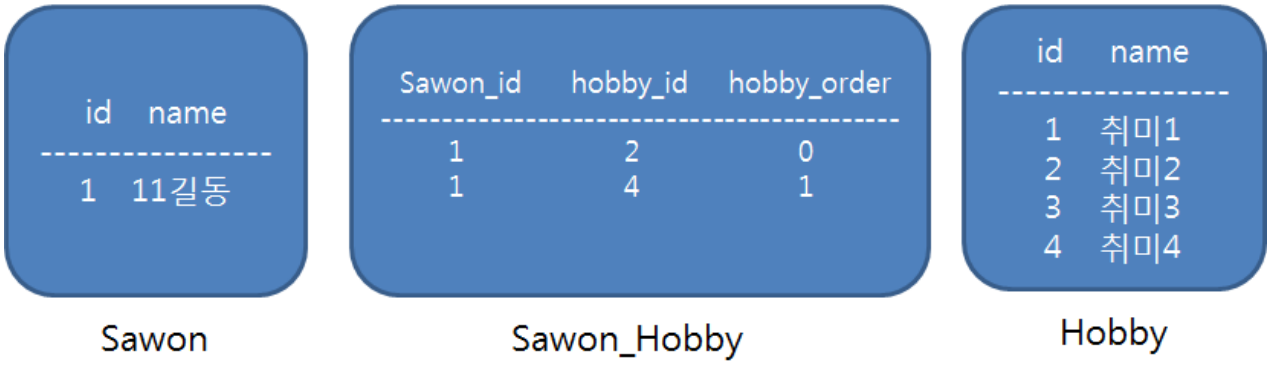
```
CascadeType.REFRESH})
@JoinTable(name = "sawon_hobby",
   joinColumns = @JoinColumn(name = "sawon_id", referencedColumnName = "id") ,
   inverseJoinColumns = @JoinColumn(name = "hobby_id", referencedColumnName = "id") )
@OrderColumn(name="hobby_order")   //Sawon_Hobby에 hobby_order칼럼 생성
public List<Hobby> getHobbies() {
            return hobbies;
}
```

Sawon 엔티티의 getHobbies() 메소드위에 @OrderColumn 어노테이션이 있으므로 List 컬렉션에 순서를 위한 칼럼(hobby_order)이 Sawon_Hobby에 생성된다. deleteSawonHobby() 메소드 실행전/후 Sawon, Sawon_Hobby, Hobby 테이블의 내용은 다음과 같다.

deleteSawonHobby() 메소드 실행 전



deleteSawonHobby() 메소드 실행 후



Sawon쪽의 hobbies List 컬렉션에서 1번취미를 제거하는데 왜 3개의 쿼리문이 실행될까?

흔히 생각할 때 1번사원의 1번취미 데이터 위 그림에서 붉은 박스 부분이 삭제되고 hobyy_order

값을 변경하여 List 컬렉션내의 순서를 변경할 것 같지만 하이버네이트에서는 List 컬렉션이라서 작동방식이 좀 다른 것 같다. 리스트 구조라 한건이 제거되면 그 이후 데이터 개수만큼 쿼리가 실행되며 아래와 같은 쿼리가 될 것이다.

```
delete from sawon_hobby where sawon_id=1 and hobby_order=2
update sawon_hobby set hobby_id=2 where sawon_id=1 and hobby_order=0
update sawon_hobby set hobby_id=4 where sawon_id=1 and hobby_order=1
```

결국 List라는 자료구조 특성상 맨앞 한 개가 빠지면서 리스트에서의 순서는 고정된 상태에서 이후의
데이터가 한칸씩 앞으로 밀리니 위와 같이 쿼리가 실행되는 것이다. hobby_order는 내부적으로 고정
된 값으로 두고 0번째가 hobby_id 2가 되고 1번째가 hobby_id 4가 되는 것이다.

데이터 건수가 많다면 좋지 않은 상황이 나올 수 있으니 특별한 경우가 아니라면 사용을 자제해야
할 것 같다.

Sawon의 getHobbies() 메소드에서 @OrderColumn(name="hobby_order") 어노테이션을 생략하고 실행하면 Sawon_Hobby 테이블의 List 컬렉션을 위한 hobby_order 칼럼이 생성되지 않으며 불 필요한 update가 일어나지 않지만 더 심각한 일이 일어난다. Sawon_Hobby가 변경되면 해당 사원데이터를 전부 지우고, 다시 입력하는 방법을 취한다.

deleteSawonHobby() 메소드의 실행결과는 다음과 같다.

```
delete from sawon_hobby where sawon_id=?
insert into sawon_hobby (sawon_id, hobby_id) values (?, ?)
insert into sawon_hobby (sawon_id, hobby_id) values (?, ?)
insert into sawon_hobby (sawon_id, hobby_id) values (?, ?)
```

updateSawon() 메소드에서 실행된 쿼리는 다음과 같다.

[자바코드]

```
Sawon s1 = sawonRepository.findOne(1L);
logger.info("수정될 사원 => " +s1.toString());
s1.getHobbies().add(new Hobby("취미4"));
s1.setName("11길동");
sawonRepository.save(s1);
```

[SQL 실행로그]

```
Hibernate: update sawon set name=? where id=?
Hibernate: delete from sawon_hobby where sawon_id=?
Hibernate: insert into sawon_hobby (sawon_id, hobby_id) values (?, ?)
Hibernate: insert into sawon_hobby (sawon_id, hobby_id) values (?, ?)
Hibernate: insert into sawon_hobby (sawon_id, hobby_id) values (?, ?)
```

컬렉션에 List를 사용하는 것은 충분히 고려 후 사용해야 할 것 같다!