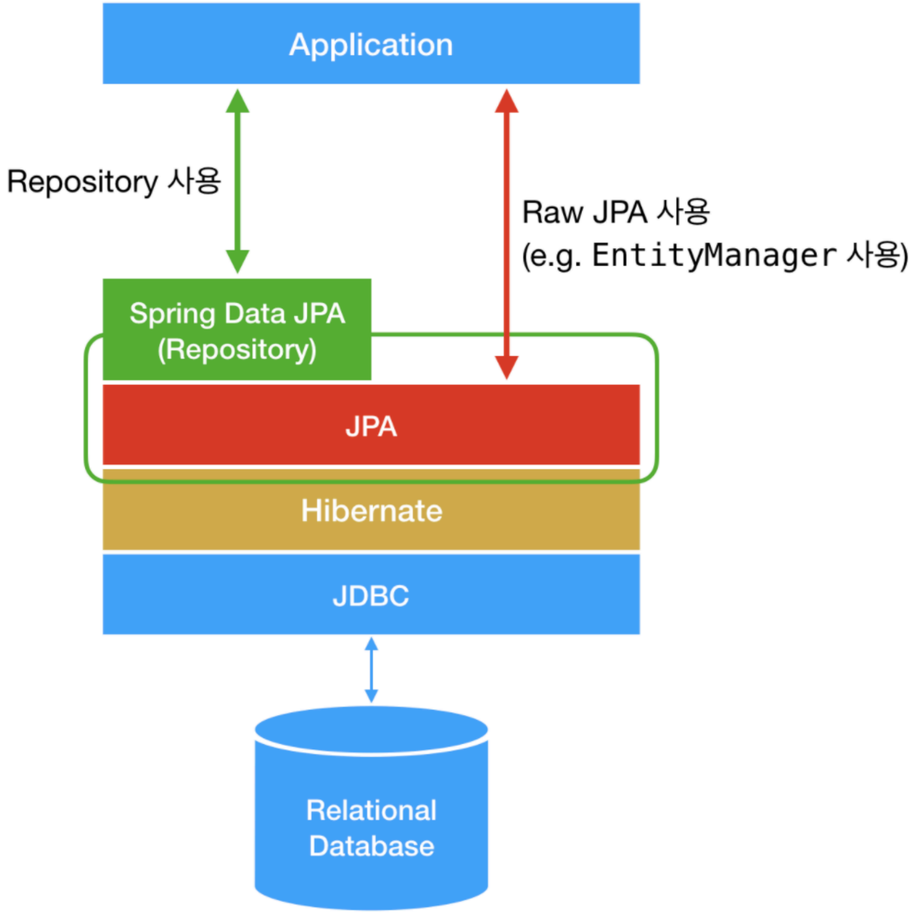
Spring Data JPA

Spring Data JPA



JDBC: 자바에서 데이터베이스에 접속할 수 있도록 도와주는 API

JPA: 자바 ORM 기술에 대한 표준 명세서로 Java에서 제공하는 API
- 자바에서 관계형 데이터베이스를 사용하는 방식을 정의한 인터페이스
- JPA는 말 그대로 명세서 인터페이스다.
- 그러므로 구현 되어 있는게 아니다.

Hibernate: JPA의 구현체이다.
- 즉 JPA를 사용하기 위해서는 Hibernate를 반드시 사용할 필요가 없다
- Hibernate의 작동방식이 마음에 들지 않는다면
- DataNucleus, EclipseLink 등 다른 JPA 구현체를 사용해도 좋다

Spring Data JPA: Spring에서 제공하는 모듈로 JPA를 더 쉽게 사용할 수 있다
- Repository라는 인터페이스를 제공함으로써 이는 가능하다.
- Repository 인터페이스에 정해진 규칙대로 메소드를 입력하면 Spring이 알아서
- 해당 메소드에 적합한 쿼리를 날려주는 구현체를 만들어준다
- 즉 내부적으로는 JPA를 사용하고 있는것

ORM (Object-Relation Mapping)

ORM은 자바 애플리케이션 객체를 SQL 데이터베이스의 테이블에 자동으로 영속화 해주는 기술이다.

```java
@Entity
@Getter @Setter @EqualsAndHashCode(of = "id")
@Builder @AllArgsConstructor @NoArgsConstructor
public class Account {

    @Id @GeneratedValue
    private Long id;

    @Column(unique = true)
    private String email;

    @Column(unique = true)
    private String nickname;

    private String password;

    private boolean emailVerified;

    private String emailCheckToken;

    private LocalDateTime emailCheckTokenGeneratedAt;

    private LocalDateTime joinedAt;
```

Account 객체

id	bio	email	email_check_token	email_check_token_generated_at
115	<null>	yjm9505160@gmail.com	8e95c86f-afd5-4d62-9a28-5dc33b472e91	2020-08-11 20:02:51.140873
102	안녕하세요 저는 개발자입니다	jeongmin@gmail.com	4d191d2e-ac7b-4f8d-9b61-712094445c74	2020-08-10 21:29:08.149198
183	<null>	serious_yeon@naver.com	891be4bb-56fd-4bb3-bbb6-51436a605355	2020-08-17 10:43:23.869307
184	<null>	tmdgh0221@gmail.com	9cbe81d9-dccc-4dee-9883-98654e2f549c	2020-08-17 20:23:47.396525
186	<null>	mintmarshmalloc@gmail.com	a310d186-5a5a-49ad-887f-71bc37b2f264	2020-08-20 12:28:31.667527
188	<null>	khy3231@gmail.com	4ec2a9f0-1079-4af9-ad4d-b5d74133364c	2020-08-21 08:21:15.437809
220	<null>	dlwoabsdk@naver.com	c7d3ae52-0009-4698-8471-0a2c32784e6d	2020-08-24 07:55:32.335590
214	안녕하세요 저는 기획자입니다	seriousyeon@gmail.com	074b36bd-e57e-4654-9bee-0954c49be9a9	2020-08-21 23:58:41.192499
189	코딩의 지배자, 누룽이	cwpink@naver.com	750e7c92-a116-4072-8283-df0cf09bc038	2020-08-21 08:21:45.310598
116	안녕하세요 저는 백엔드 개발	joonpark0221@gmail.com	ac7dcc01-9e74-46cb-b4ec-f7a81b563f56	2020-08-11 20:02:55.337356
185	안녕하세요 저는 블록체인 개	junho918918@gmail.com	3288273a-220b-47a6-85f1-bdae28b8b46f	2020-08-20 05:07:29.003611

Account 테이블

ORM (Object-Relation Mapping)

```
try(
        Connection connection = DriverManager.getConnection(url, username, password)) {

        String sql = "INSERT INTO ACCOUNT VALUES(1, 'keesun', 'pass');";

        try(PreparedStatement statement = connection.prepareStatement(sql)) {

        statement.execute();
        }
    }
```

JDBC (Java Database Connectivity)

- 자바에서 데이터베이스에 접속할 수 있도록 하는 API

ORM (Object-Relation Mapping)

```
// Account라는 테이블이 데이터베이스에 있다

Account account = new Account(“keesun”, “pass”);

accountRepository.save(account);
```

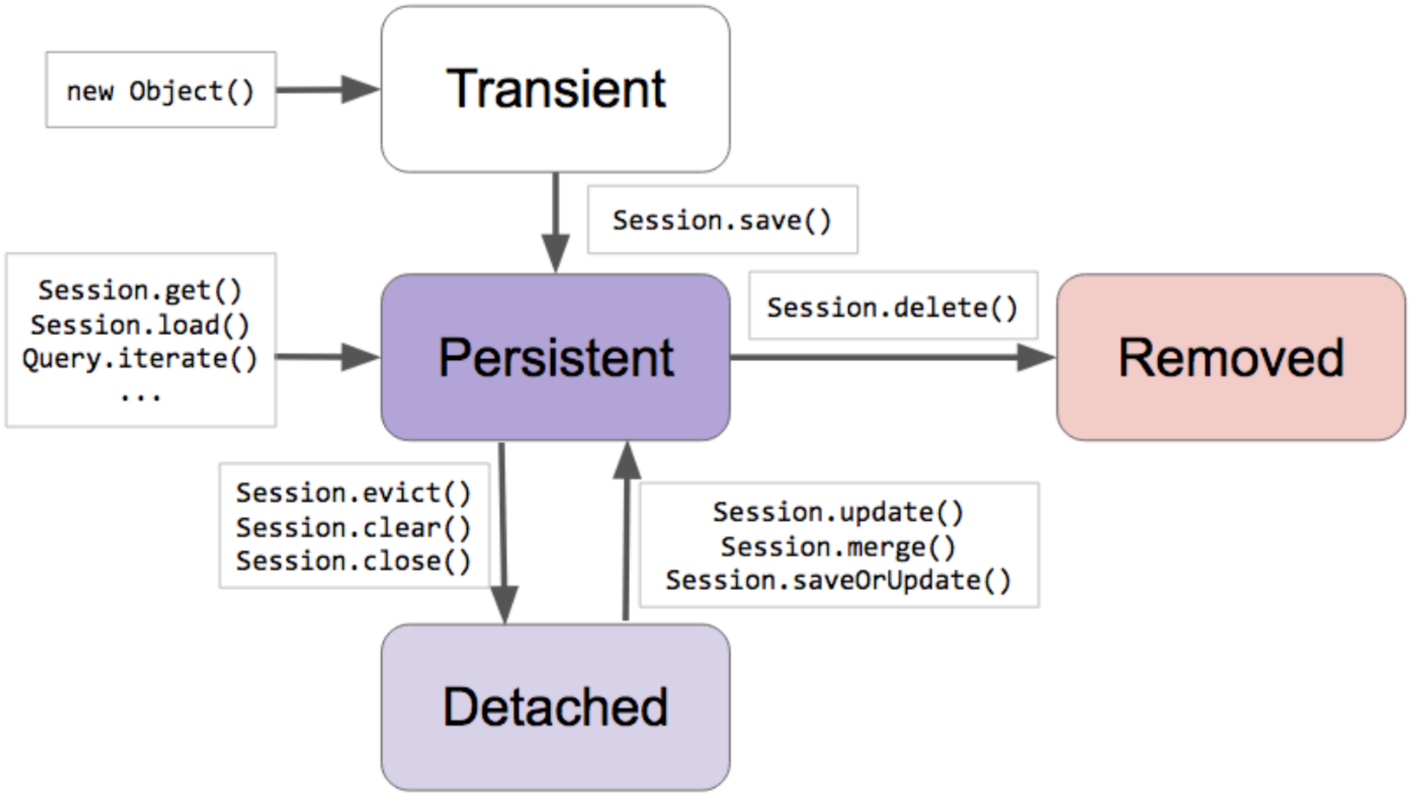
ORM (Object-Relation Mapping)

ORM (Object-Relation Mapping)

JDBC 대신 ORM을 쓰는 이유는?

- 객체 지향 프로그래밍의 장점을 사용할 수 있다
- 각종 디자인 패턴을 사용할 수 있다
- 코드를 재사용할 수 있다
- 테스트하기도 쉽다
- 반복되는 코드를 줄일 수 있다

JPA에서 알고가야 할 것



JPA에서 알고가야 할 것

Entity의 상태들

- Transient: JPA가 모르는 상태
  - ex) 객체를 처음 만든 경우 하지만 저장은 하지 않은 경우

- Persistent: JPA가 관리하는 상태
  - DB에 저장을 할 경우
  - save() 저장을 했다고 해서 바로 즉시 Insert 쿼리가 발생해 DB에 저장되는건 아니다
  - Persistent 상태가 되고나서 한 흐름(Transaction)이 끝날 때마다 모아서 DB에 Sync한다.
  - Persistent 상태가되면 다양한 기능을 제공해준다.
    - Persistent 상태가 되면 1차 캐시 기능을 제공해준다.
      - 캐시가 되어있기 때문에 객체를 조회할 경우 Select 쿼리가 발생하지 않는다
      - DB에 접근하지 않고 Persistent 객체를 관리하는 Context에서 바로 돌려준다

    - Persistent 상태가 되면 Dirty Checking과 WriteBehind 기능을 제공해준다.
      - save()를 하면 Persistent 상태가 된다고 했다.
      - 이 상태에서 객체 데이터를 변경한 경우 알아서 Update Query가 발생한다.
      - Dirty Checking은 변경상태를 계속 모니터링 해준다 라는 뜻이다.
      - WriteBehind는 객체 상태 변화를 최대한 늦게 DB에 적용한다 라는 뜻이다
  - 한 흐름(Transaction)이 끝나면 Persistent 상태는 Detached 상태로 바뀌게 된다.
    - Detached: JPA가 더이상 관리하지 않는 상태

Spring Data JPA

실제로 Spring을 사용하면서 데이터베이스에 접근해야 하는 경우가 생기면 Spring Data JPA를 사용합니다

사용 방법은 되게 단순하다.

```
public interface AccountRepository extends JpaRepository <Account, Long>{

}
```

JpaRepository에 공통으로 적용할 수 있는 기능들은 모두 들어가 있다.

Spring Data JPA - JpaRepository

```java
@NoRepositoryBean
public interface JpaRepository<T, ID> extends PagingAndSortingRepository<T, ID>, QueryByExampleExecutor<T> {
    List<T> findAll();

    List<T> findAll(Sort var1);

    List<T> findAllById(Iterable<ID> var1);

    <S extends T> List<S> saveAll(Iterable<S> var1);

    void flush();

    <S extends T> S saveAndFlush(S var1);

    void deleteInBatch(Iterable<T> var1);

    void deleteAllInBatch();

    T getOne(ID var1);

    <S extends T> List<S> findAll(Example<S> var1);

    <S extends T> List<S> findAll(Example<S> var1, Sort var2);
}
```

Spring Data JPA - JpaRepository

```java
@NoRepositoryBean
public interface PagingAndSortingRepository<T, ID> extends CrudRepository<T, ID> {
    Iterable<T> findAll(Sort var1);

    Page<T> findAll(Pageable var1);
}
```

Spring Data JPA - JpaRepository

```java
@NoRepositoryBean
public interface CrudRepository<T, ID> extends Repository<T, ID> {
    <S extends T> S save(S var1);

    <S extends T> Iterable<S> saveAll(Iterable<S> var1);

    Optional<T> findById(ID var1);

    boolean existsById(ID var1);

    Iterable<T> findAll();

    Iterable<T> findAllById(Iterable<ID> var1);

    long count();

    void deleteById(ID var1);

    void delete(T var1);

    void deleteAll(Iterable<? extends T> var1);

    void deleteAll();
}
```

Spring Data JPA

결국엔 내부적으로 JPA로 변환이 된다

단순한 CRUD를 만들지 않아도 되니까 핵심 비즈니스 로직을 개발하는데 집중 할 수 있다.

공통으로 만들 수 없는 단순한 기능들은 이름만 지어주면 그것대로 기능이 만들어진다.
- example

Query에 포함할 수 있는 키워드들이 정리되어 있는 Jpa Reference
https://docs.spring.io/spring-data/jpa/docs/1.10.1.RELEASE/reference/html/#jpa.sample-app.finders.strategies

단순한 기능들은 Spring Data JPA를 통해서 바로 금방 만들 수 있다

복잡한 기능들을 만들려면 QueryDSL을 해야한다
- example

Spring Data JPA - EntityGraph

쿼리 메소드마다 연관관계의 Fetch 모드를 설정할 수 있다

Example

Post (게시물) - Comment (댓글) = 1 : N 관계
- OneToMany (Post 입장)
- ManyToOne (Comment 입장)

Comment 하나를 조회하면 Comment와 연관되어 있는 Post 까지 가지고오게 된다
Post 하나를 조회하면 Comment들을 가지고 오지 않는다.

Why?
- ManyToOne 인 경우에 Database에 값을 가져올 때 FetchType이 기본적으로 Eager이 되어 있어서
- OneToMany 인 경우에는 Database에 값을 가져올 때 FetchType이 기본적으로 Lazy로 되어있어서

Eager vs Lazy
- Eager는 한번 조회를 할 때 다 가지고 오는것을 말하고
- Lazy는 필요한 시점에 (접근을 할 때) 조회를 한다.
- 당장 필요 없는 경우라면 Lazy로 해서 성능을 높이는게 좋다.

Spring Data JPA - Transaction

Spring Data JPA가 제공하는 Repository의 모든 메소드에는 기본적으로 @Transaction이 적용되어 있다.

RuntimeException과 Error가 발생하면 Rollback을 하지만 Checked Exception인 경우에는 그렇지않다
- Checked Exception인 경우는 Try Catch를 통해 반드시 처리해야하는 Exception을 말한다
- 대표적으로는 IO Exception
- Checked Exception인 경우에는 복구 전략을 추가하는게 가능하다 그러므로 그런 작업을 했을 수 있으니까 롤백하지 않는다.

특정 Exception에 대한 Rollback 추가하는게 가능
- rollbackFor
- rollbackForClassName

ReadOnly
- Entity(JPA가 관리하는 객체)를 조회만 하는 경우라면 읽기 전용으로 사용해서 성능을 개선할 수 있다
- 어떻게 개선하는가?
  - Entity들은 JPA의 Persistent Context들이 관리한다.
  - Persistent의 상태일 때는 다양한 기능들이 제공된다 (Dirty Checking 등)
  - 이런 기능들은 메모리를 과다하게 사용할 수도 있다
  - 메모리를 과다하게 사용하면 성능이 급격하게 느려지겠죠?
  - 조회만 한다면 이런 기능들은 다 필요가 없으니까 최적화 하는게 가능하다.

Spring Data JPA - Transaction

Transaction Isolate Level을 바꿀 수 있다.

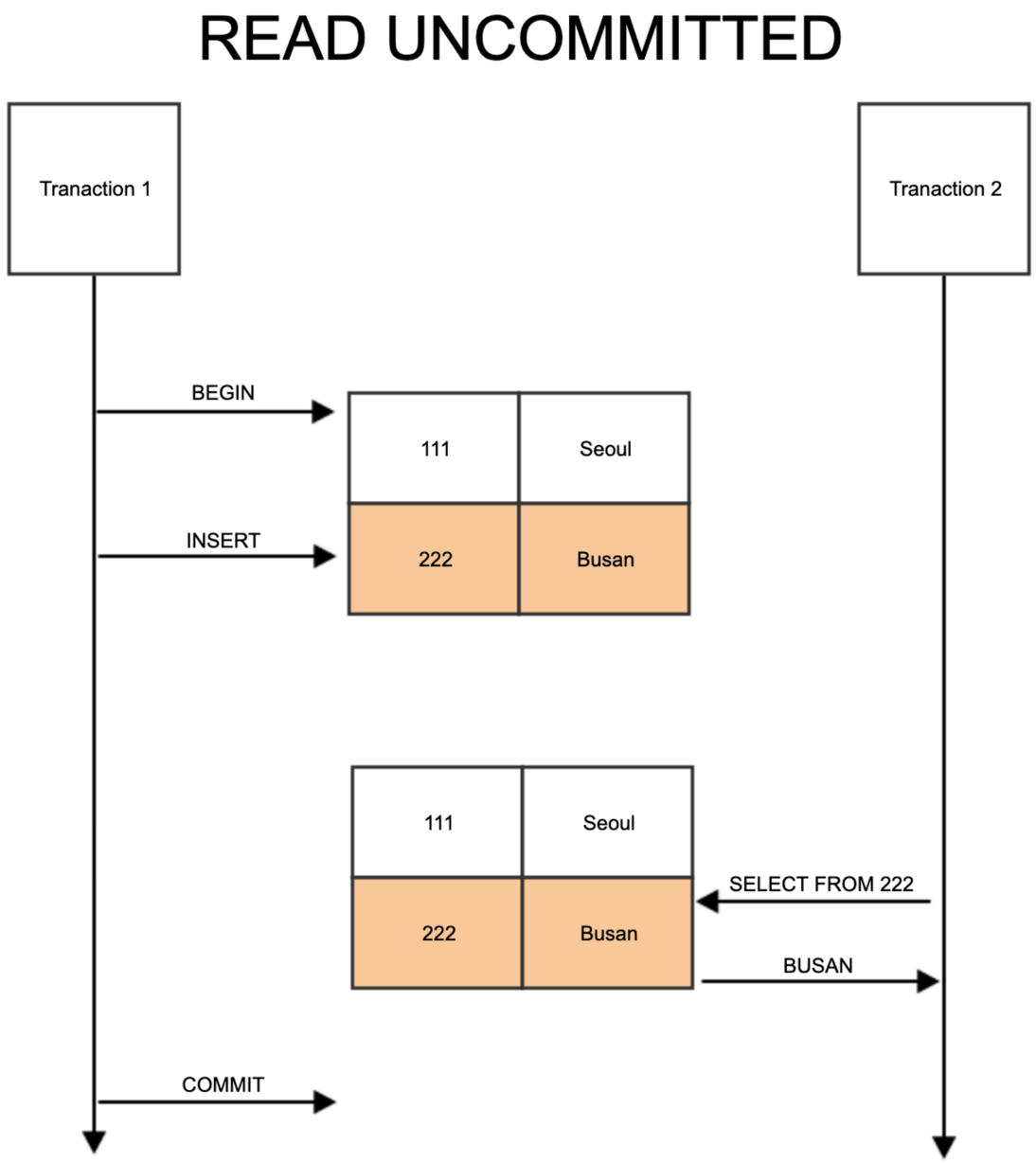
Isolate Level

- 여러 사용자가 동일한 데이터를 동시에 수정하려고 할 때 발생하는 상황에 대해서 어떻게 처리할 건지를 정의한다
- **Read UnCommitted, Read Committed, Repeatable Read, Serializable의 상태가 있다**
- 모든 트랜잭션은 이 중 하나의 격리 수준을 갖는다.
- 데이터베이스마다 조금씩 다를 순 있다. **(mysql, h2, postgresql)**

Spring Data JPA - Transaction

Read UnCommitted

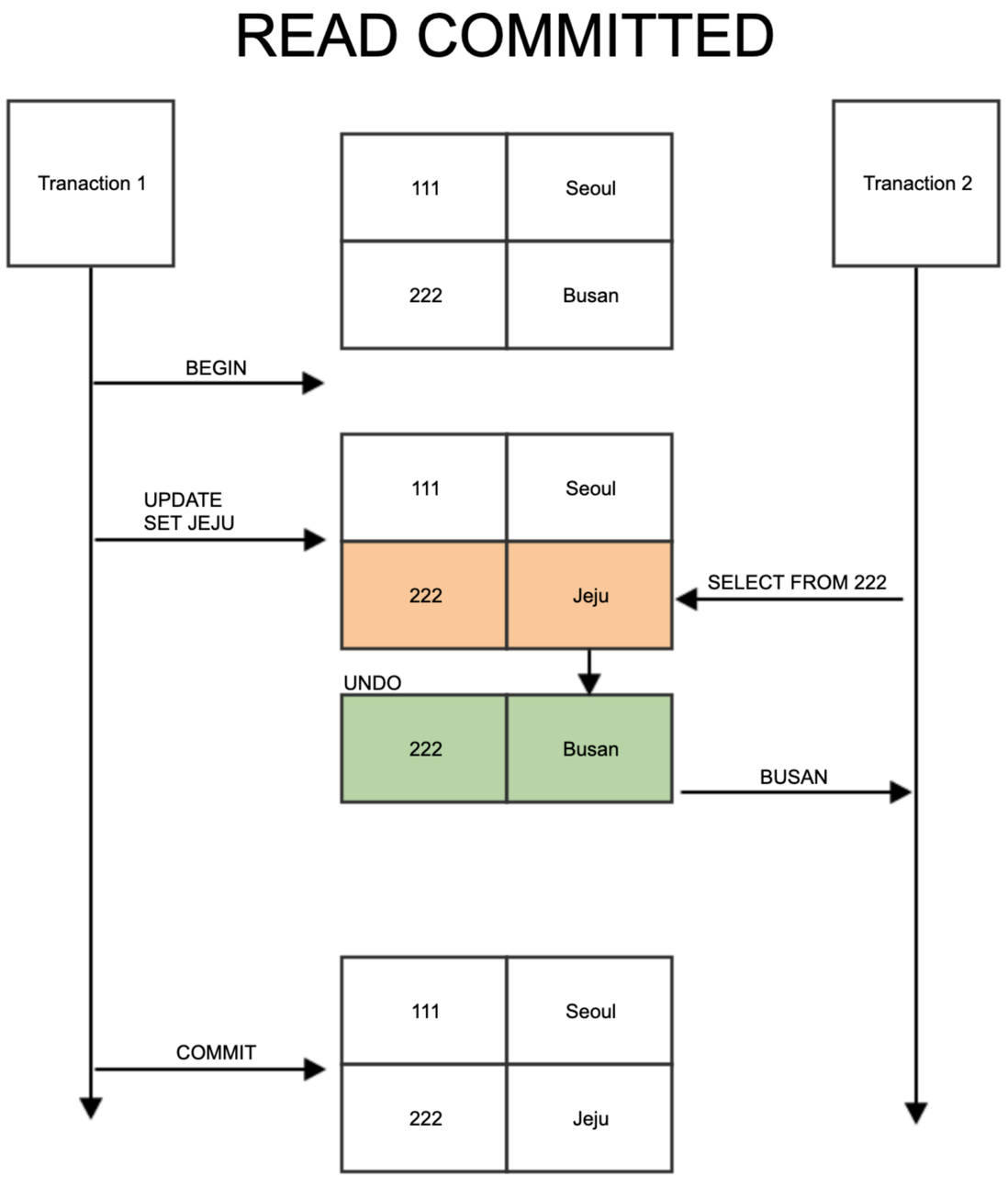
- 각 트랜잭션에서의 변경 내용이 **Commit**이나 **Rollback** 여부에
- 상관없이 다른 트랜잭션에서 값을 읽을 수 있다.
- 그러므로 **DIRTY READ** 현상이 발생할 수 있다.



Spring Data JPA - Transaction

Read Committed

- **commit 된 데이터만 읽는 격리수준이다.**
- **RDB에서 대부분 기본적으로 사용하고 있는 격리 수준이다.**
- **Dirty Read와 같은 현상은 발생하지 않는다**
- **항상 같은 결과를 보장해 주지 않을 수가 있다는 문제점이 있다.**



Spring Data JPA - Transaction

Repeatable Read
- 트랜잭션마다 트랜잭션 ID를 부여해서 트랜잭션 ID보다 작은 트랜잭션번호에서 변경한 것만 읽게된다.

Serializable
- 모든 작업을 하나의 트랜잭션에서 처리하는 것과 같은 높은 고립수준을 제공한다고 합니다
- 고립 수준이 높은 만큼 동시에 처리하는 효율은 가장 떨어진다.
- 하나씩 처리하는 느낌인듯

Spring Data JPA - Transaction

Transaction Propagation Level을 수정할 수 있다

- **REQUIRED : 부모 트랜잭션 내에서 실행하며 부모 트랜잭션이 없을 경우 새로운 트랜잭션을 생성**
- **REQUIRES_NEW : 부모 트랜잭션을 무시하고 무조건 새로운 트랜잭션이 생성**
- **SUPPORT : 부모 트랜잭션 내에서 실행하며 부모 트랜잭션이 없을 경우 nontransactionally로 실행**
- **MANDATORY : 부모 트랜잭션 내에서 실행되며 부모 트랜잭션이 없을 경우 예외가 발생**
- **NOT_SUPPORT : nontransactionally로 실행하며 부모 트랜잭션 내에서 실행될 경우 일시 정지**
- **NEVER : nontransactionally로 실행되며 부모 트랜잭션이 존재한다면 예외가 발생**
- **NESTED : 해당 메서드가 부모 트랜잭션에서 진행될 경우 별개로 커밋되거나 롤백될 수 있음. 둘러싼 트랜잭션이 없을 경우 REQUIRED와 동일하게 작동**