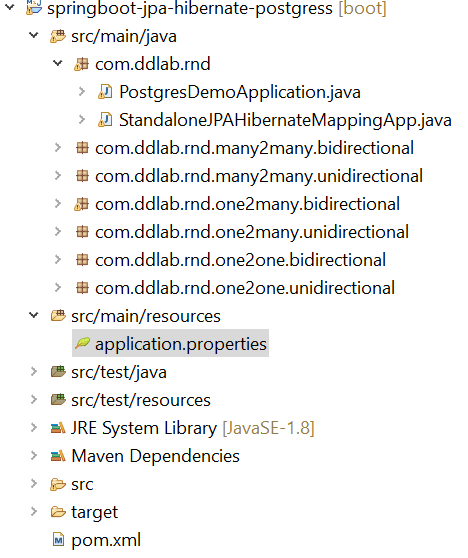
Spring Boot JPA Hibernate Mappings – 2021-2022

**Project Structure**

  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
**application.properties**

**Relevant Portion of Pom.xml**

<parent>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-parent</artifactId>

<version>2.2.1.RELEASE</version>

<relativePath /> <!-- lookup parent from repository -->

</parent>

<properties>

<java.version>1.8</java.version>

</properties>

<dependencies>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-data-jpa</artifactId>

</dependency>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-web</artifactId>

</dependency>

<dependency>

<groupId>org.postgresql</groupId>

<artifactId>postgresql</artifactId>

<scope>runtime</scope>

</dependency>

<dependency>

<groupId>org.projectlombok</groupId>

<artifactId>lombok</artifactId>

<scope>provided</scope>

</dependency>

</dependencies>

<build>

plugins>

<plugin>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-maven-plugin</artifactId>

</plugin>

</plugins>

</build>

## Spring DATASOURCE (DataSourceAutoConfiguration & DataSourceProperties)

**spring.datasource.url=jdbc:postgresql://localhost:5432/test**

spring.datasource.username=postgres

spring.datasource.password=postgres

# The SQL dialect makes Hibernate generate better SQL for the chosen database

**spring.jpa.properties.hibernate.dialect = org.hibernate.dialect.PostgreSQLDialect**

# Hibernate ddl auto (create, create-drop, validate, update)

**spring.jpa.hibernate.ddl-auto = update**

**One to One Unidirectional Mapping**

@Data

@Entity

@Table(name = "ADDRESS")

**public** **class** Address {

@Id

@GeneratedValue

@Column(name = "ADDRESS\_ID")

**private** **long** id;

@Column(name = "CITY")

**private** String city;

**public** Address() {

}

**public** Address(String city) {

**this**.city = city;

}

}

@Data

@Entity

@Table(name = "STUDENT")

**public** **class** Student {

@Id

@GeneratedValue

@Column(name = "STUDENT\_ID")

**private** **long** id;

@Column(name = "FIRST\_NAME")

**private** String firstName;

**@OneToOne(fetch = FetchType.*LAZY*,**

**cascade = CascadeType.*ALL*)**

**@JoinColumn(name="HOME\_ADDRESS\_ID")**

**private Address address;**

**public** Student() {

}

**public** Student(String firstName) {

**this**.firstName = firstName;

}

}

**Repository**

@Repository

**public** **interface** One2OneUnidirectionalRepo **extends** JpaRepository<Student, Long> {

}

**Service**

@Service

**public** **class** One2OneUnidirectionalService {

@Autowired

**private** One2OneUnidirectionalRepo repo;

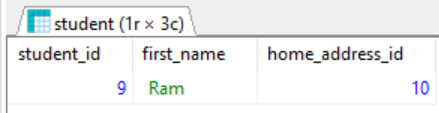
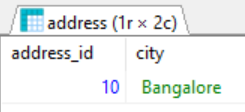
**public** **void** saveInfo(Student s) {

repo.save(s);

}

}

**Database**

**Test Class for One-One UniDirectional**

**public** **class** TestOne2OneUnidirectional {

**public** **static** **void** storeInfo(ApplicationContext applicationContext) {

Student s = **new** Student("Ram");

Address a = **new** Address("Bangalore");

s.setAddress(a);

One2OneUnidirectionalService service = applicationContext.getBean(One2OneUnidirectionalService.**class**);

service.saveInfo(s);

}

}

**One to One Bidirectional Mapping**

@Data

@Entity(name = "PostDetails")

@Table(name = "post\_details")

**public** **class** PostDetails {

@Id

@GeneratedValue

**private** Long id;

**@OneToOne(fetch = FetchType.*LAZY*)**

**@JoinColumn(name = "post\_id")**

**private** Post1 post;

**private** String name;

**public** PostDetails() {}

**public** PostDetails(String name) {

**this**.name = name;

}

}

@Data

@Entity(name = "Post1")

@Table(name = "post1")

**public** **class** Post1 {

@Id

@GeneratedValue

**private** Long id;

**private** String title;

**@OneToOne(mappedBy = "post",**

**cascade = CascadeType.*ALL*,**

**fetch = FetchType.*LAZY*)**

**private** PostDetails details;

**public** Post1() {

}

**public** Post1(String title) {

**this**.title = title;

}

**public** **void** setDetails(PostDetails details) {

**if** (details == **null**) {

**if** (**this**.details != **null**) {

**this**.details.setPost(**null**);

}

} **else** {

details.setPost(**this**);

}

**this**.details = details;

}

}

@Repository

**public** **interface** One2ManyBidirectionalRepo **extends** JpaRepository<Post1, Long> {

}

@Service

**public** **class** One2ManyBidirectionalService {

@Autowired

**private** One2ManyBidirectionalRepo repo;

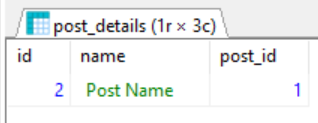
**public** **void** saveCustInfo(Post1 post1) {

repo.save(post1);

}

}

**Database Tables**

As you can see there are 3 columns in post\_details table. But you can also do it in such a manner that there will be two columns in post\_details table.

@Data

@Entity(name = "PostDetails")

@Table(name = "post\_details")

**public** **class** PostDetails {

// Better way to do it

// https://vladmihalcea.com/the-best-way-to-map-a-onetoone-relationship-with-jpa-and-hibernate/

@Id

**private** Long id;

**@OneToOne(fetch = FetchType.*LAZY*)**

**@MapsId**

**private Post1 post;**

**private** String name;

**public** PostDetails() {}

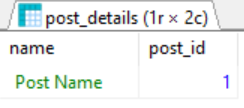
**public** PostDetails(String name) {

**this**.name = name;

}

}

**Database Design**

**Test Class for One-One Bidirectional**

**public** **class** TestOne2ManyBidirectional {

**public** **static** **void** storeInfo(ApplicationContext applicationContext) {

Post1 post1 = **new** Post1("Post Title");

PostDetails pd = **new** PostDetails("Post Name");

post1.setDetails(pd);

One2ManyBidirectionalService service = applicationContext.getBean(One2ManyBidirectionalService.**class**);

service.saveCustInfo(post1);

}

}

**One to Many Unidirectional Mapping**

@Data

@Entity(name = "Post")

@Table(name = "post")

**public** **class** Post {

@Id

@GeneratedValue

**private** Long id;

**private** String title;

@OneToMany(cascade = CascadeType.***ALL***,

orphanRemoval = **true**)

@JoinColumn(name = "post\_id")

**private** List<PostComment> comments =

**new** ArrayList<>();

**public** Post() { }

**public** Post(String title) {

**this**.title = title;

}

}

@Data

@Entity(name = "PostComment")

@Table(name = "post\_comment")

**public** **class** PostComment {

@Id

@GeneratedValue

**private** Long id;

**private** String review;

**public** PostComment() { }

**public** PostComment(String review) {

**this**.review = review;

}

}

@Repository

**public** **interface** PostRepo **extends** JpaRepository< Post, Long> {

}

@Service

**public** **class** PostService {

@Autowired

**private** PostRepo repo;

**public** **void** savePost(Post post) {

repo.save(post);

}

}

**Test Class for One To Many UniDirectional**

**public** **class** TestOne2ManyUniDirectional {

**public** **static** **void** storeInfo(ApplicationContext applicationContext) {

Post post = **new** Post("First post");

post.getComments().add(**new** PostComment("My first review"));

post.getComments().add(**new** PostComment("My second review"));

post.getComments().add(**new** PostComment("My third review"));

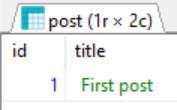
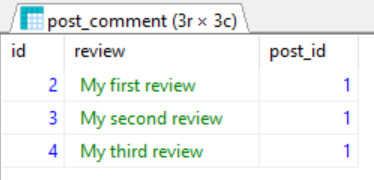
PostService service = applicationContext.getBean(PostService.**class**);

service.savePost(post);

}

}

**Database Design**

**One to Many Bidirectional Mapping**

@Data

@Entity(name = "BiDirPost")

@Table(name = "bidirpost")

**public** **class** BiDirPost {

@Id

@GeneratedValue

**private** Long id;

**private** String title;

@OneToMany(

// attribute from other class BiDirPostComment, ie. private BiDirPost post;

mappedBy = "post",

cascade = CascadeType.***ALL***,

orphanRemoval = **true**

)

**private** List<BiDirPostComment> comments = **new** ArrayList<>();

**public** BiDirPost() {

}

**public** BiDirPost(String title) {

**this**.title = title;

}

**public** **void** addComment(BiDirPostComment comment) {

comments.add(comment);

comment.setPost(**this**);

}

**public** **void** removeComment(BiDirPostComment comment) {

comments.remove(comment);

comment.setPost(**null**);

}

}

@Service

**public** **class** BiDirPostService {

@Autowired

**private** BiDirPostRepo repo;

**public** **void** savePost(BiDirPost post) {

repo.save(post);

}

}

@Data

@Entity(name = "BiDirPostComment")

*@Table(name = "bidirpost\_comment")*

***public******class*** *BiDirPostComment {*

*@Id*

*@GeneratedValue*

***private*** *Long id;*

**private** String review;

@ManyToOne(fetch = FetchType.***LAZY***)

**private** BiDirPost post;

**public** BiDirPostComment() { }

**public** BiDirPostComment(String review) {

**this**.review = review;

}

}

@Repository

**public** **interface** BiDirPostRepo **extends** JpaRepository<BiDirPost, Long> {

}

**Test Class**

**public** **class** TestOne2ManyBidirectional {

**public** **static** **void** storeInfo(ApplicationContext applicationContext) {

BiDirPost post = **new** BiDirPost("First post");

post.addComment(**new** BiDirPostComment("My first review"));

post.addComment(**new** BiDirPostComment("My second review"));

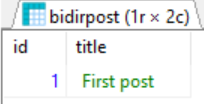
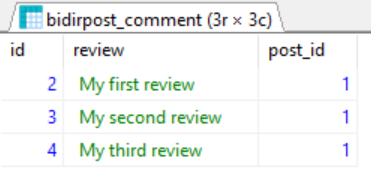
post.addComment(**new** BiDirPostComment("My third review"));

BiDirPostService service = applicationContext.getBean(BiDirPostService.**class**);

service.savePost(post);

}

} **Database Design**

**Many to Many Unidirectional Mapping**

@Data

@Table(name = "author")

@Entity

**public** **class** Author {

@Id

@GeneratedValue

**private** Long authorId;

@Column(name = "author\_name")

**private** String authorName;

}

@Data

@Entity

@Table(name = "book")

**public** **class** Book {

@Id

@GeneratedValue

**private** Long bookId;

@Column(name = "book\_name")

**private** String bookName;

@ManyToMany(cascade = CascadeType.***ALL***)

@JoinTable(name = "author\_book",

joinColumns = @JoinColumn(name = "book\_id"),

inverseJoinColumns = @JoinColumn(name = "author\_id"))

**private** Set<Author> authors;

}

@Repository

**public** **interface** UniDirectionalRepo **extends** JpaRepository< Book, Long> {

}

@Service

**public** **class** UniDirectionalService {

@Autowired

**private** UniDirectionalRepo repo;

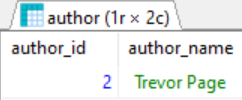
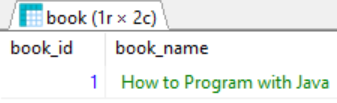
**public** **void** saveInfo(Book book) {

repo.save(book);

}

}

**DataBase Design**

   
  
**Test Class**  
**public** **class** TestMany2ManyUnidirectional {

**public** **static** **void** storeInfo(ApplicationContext applicationContext) {

Set<Author> howToProgramWithJavaAuthor = **new** HashSet<Author>();

Set<Author> howToProgramWithJava2ndAuthors = **new** HashSet<Author>();

Author author = **new** Author();

author.setAuthorName("Trevor Page");

howToProgramWithJavaAuthor.add(author);

Author author2 = **new** Author();

author2.setAuthorName("John Doe");

howToProgramWithJava2ndAuthors.add(author);

howToProgramWithJava2ndAuthors.add(author2);

Book book = **new** Book();

book.setBookName("How to Program with Java");

book.setAuthors(howToProgramWithJavaAuthor);

UniDirectionalService service = applicationContext.getBean(UniDirectionalService.**class**);

service.saveInfo(book);

}

}

**Many to Many Bidirectional Mapping**

@Entity

@Table(name = "albums")

**public** **class** Albums {

@Id

@GeneratedValue

@Column(name = "album\_id")

**private** Long albumId;

@Column(name = "album\_name")

**private** String albumName;

**@ManyToMany(cascade = CascadeType.*ALL*)**

**@JoinTable(name = "artist\_album",**

**joinColumns = @JoinColumn(name = "album\_id"),**

**inverseJoinColumns = @JoinColumn(name = "artist\_id"))**

**private Set<Artist> artists;**

get()/set() method, don’t use Lombok @Data

}

@Entity

@Table(name = "artist")

**public** **class** Artist {

@Id

@GeneratedValue

@Column(name = "artist\_id")

**private** Long artistId;

@Column(name = "artist\_name")

**private** String artistName;

@ManyToMany(cascade = CascadeType.***ALL***,

mappedBy = "artists")

**private** Set<Albums> albums;

get()/set() method, don’t use Lombok @Data

}

@Repository

**public** **interface** BidirectionalRepo **extends** JpaRepository<Artist, Long> {

}

@Service

**public** **class** BidirectionalService {

@Autowired

**private** BidirectionalRepo repo;

**public** **void** saveInfo(Artist artist) {

repo.save(artist);

}

}

Test Class

**public** **class** TestMany2ManyBidirectional {

**public** **static** **void** storeInfo(ApplicationContext applicationContext) {

Artist artist1 = **new** Artist();

artist1.setArtistName("Trevor Page");

Set<Artist> popArtists = **new** HashSet<Artist>();

popArtists.add(artist1);

Artist artist2 = **new** Artist();

artist2.setArtistName("John Doe");

Set<Artist> classicalArtists = **new** HashSet<Artist>();

classicalArtists.add(artist1);

classicalArtists.add(artist2);

Albums album1 = **new** Albums();

album1.setAlbumName("How to Program with Java");

Albums album2 = **new** Albums();

album2.setAlbumName("How to Program with Java 2nd Edition");

Set<Albums> popAlbums = **new** HashSet<Albums>();

popAlbums.add(album1);

popAlbums.add(album2);

Albums album3 = **new** Albums();

album3.setAlbumName("How to Play Guitar");

Set<Albums> classicalAlbums = **new** HashSet<Albums>();

classicalAlbums.add(album2);

classicalAlbums.add(album3);

artist1.setAlbums(popAlbums);

artist2.setAlbums(classicalAlbums);

album1.setArtists(popArtists);

album2.setArtists(classicalArtists);

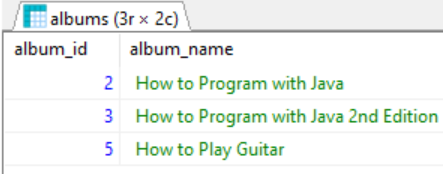
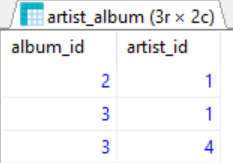
BidirectionalService service = applicationContext.getBean(BidirectionalService.**class**);

service.saveInfo(artist1);

}

}

**DataBase Design**

**Complete Spring Boot Class is given below**

@SpringBootApplication

**public** **class** StandaloneJPAHibernateMappingApp {

**public** **static** **void** main(String[] args) {

ApplicationContext applicationContext = SpringApplication.*run*(StandaloneJPAHibernateMappingApp.**class**, args);

// One-One UniDirectional

// TestOne2OneUnidirectional.storeInfo(applicationContext);

// One-One BiDirectional

// TestOne2ManyBidirectional.storeInfo(applicationContext);

// // One-Many Unidirectional

// TestOne2ManyBidirectional.storeInfo(applicationContext);

//

// // One-Many Bidirectional

// TestOne2OneBidirectional.storeInfo(applicationContext);

//

// // Many-Many UniDirectional

// TestMany2ManyUnidirectional.storeInfo(applicationContext);

//

// // Many-Many Bidirectional

TestMany2ManyBidirectional.*storeInfo*(applicationContext);

}

}

**Usage of JPA Useful Annotation**

@Repository

**public** **interface** EmployeeRepository **extends** JpaRepository<Employee, Long> {

@Query("FROM Employee p WHERE p.name = :name")

Employee findByName(@Param("name") String name);

@Query("SELECT COUNT(\*) FROM Employee p")

**long** getTotalNoEmployees();

@Transactional

@Modifying

// Make sure to add @Transactional annotation while adding @Modifying annotation otherwise it will throw

// javax.persistence.TransactionRequiredException: Executing an update/delete query

@Query("UPDATE Employee p SET p.name = :name WHERE p.id = :id")

**void** changeName(@Param("id") **long** id, @Param("name") String name);

@Transactional

@Modifying

@Query(

value =

"insert into Employee (id, name, address, project\_name) values (:id, :name, :address, :project\_name)",

nativeQuery = **true**)

**void** insertEmployee(@Param("id") **long** id, @Param("name") String name,

@Param("address") String address, @Param("project\_name") String project\_name);

@Transactional

@Modifying

@Query(value = "update Employee e set address = ? where e.name = ?", nativeQuery = **true**)

**int** updateEmployeeSetAddressForNative(String address, String name);

}

**Service Class**

@Service

**public** **class** EmployeeService {

@Autowired

**private** EmployeeRepository empRepo;

**public** **void** createEmployee(Employee emp) {

empRepo.save(emp);

}

**public** Employee getById(Long id) {

**return** empRepo.findById(id).get();

}

**public** Employee getByName(String name) {

**return** empRepo.findByName(name);

}

**public** **long** getTotalEmployees() {

**return** empRepo.getTotalNoEmployees();

}

**public** **void** changeEmployeeName(Long id, String name) {

empRepo.changeName(id, name);

}

**public** **void** insertEmployee(**long** id, String name, String address, String project\_name) {

empRepo.insertEmployee(id, name, address, project\_name);

}

**public** **void** changeAddress(String address, String name) {

empRepo.updateEmployeeSetAddressForNative(address, name);

}

}

Hibernate Spring JPA Batch Processing

**Entity**

@Data

@Entity

@Table(name = "customer")

**public** **class** Customer {

@Id

@GeneratedValue

**private** Long id;

**private** String firstName;

**private** String lastName;

**public** Customer() {}

**public** Customer(String firstName, String lastName) {

**this**.firstName = firstName;

**this**.lastName = lastName;

}

}

**application.properties**

spring.jpa.properties.hibernate.generate\_statistics=true

**spring.jpa.properties.hibernate.jdbc.batch\_size=5 🡨 Important**

**spring.jpa.properties.hibernate.order\_inserts=true 🡨 Important**

**Repository**

@Repository

**public** **interface** CustomerRepository **extends** JpaRepository<Customer, Long> {

}

**Service Layer**

@Service

**public** **class** CustomerService {

@Autowired

**private** CustomerRepository repo;

@PersistenceContext

**private** EntityManager em;

@Value("${spring.jpa.properties.hibernate.jdbc.batch\_size}")

**private** **int** batchSize;

**public** **void** saveAllCustomers(List<Customer> custList) {

repo.saveAll(custList);

}

@Transactional

**public** **void** saveAllCustomers\_usingEntityManager(List<Customer> custList) {

System.***out***.println("Now batch Size : "+batchSize);

**for**(**int** i = 0; i < custList.size(); i++) {

Customer cust = custList.get(i);

**if** (i > 0 && i % batchSize == 0) {

em.flush();

em.clear();

}

em.persist(cust);

}

}

}

**Test Class**

**public** **class** TestBatchInsert {

**private final static int *TOTAL* = 4;**

**public** **static** **void** batchInsert(ApplicationContext applicationContext) {

CustomerService service = applicationContext.getBean(CustomerService.**class**);

List<Customer> custList = **new** ArrayList<Customer>();

**for** (**int** i = 0; i < ***TOTAL***; i++) {

Customer cust = **new** Customer("First Name - " + i, "Last Name - " + i);

custList.add(cust);

}

service.saveAllCustomers(custList);

}

**public** **static** **void** batchInsertUsingEntityManager(ApplicationContext applicationContext) {

CustomerService service = applicationContext.getBean(CustomerService.**class**);

List<Customer> custList = **new** ArrayList<Customer>();

**for** (**int** i = 0; i < ***TOTAL***; i++) {

Customer cust = **new** Customer("First Name - " + i, "Last Name - " + i);

custList.add(cust);

}

service.saveAllCustomers\_usingEntityManager(custList);

}

}

Spring Boot Main Application

@SpringBootApplication

**public** **class** StandaloneJPAHibernateMappingApp {

**public** **static** **void** main(String[] args) {

ApplicationContext applicationContext = SpringApplication.*run*(StandaloneJPAHibernateMappingApp.**class**, args);

TestBatchInsert.*batchInsert*(applicationContext);

}

}

Output will be like this

27315 nanoseconds spent acquiring 1 JDBC connections;

0 nanoseconds spent releasing 0 JDBC connections;

603684 nanoseconds spent preparing 4 JDBC statements;

3268688 nanoseconds spent executing 3 JDBC statements;

**4028317 nanoseconds spent executing 2 JDBC batches;**

0 nanoseconds spent performing 0 L2C puts;

0 nanoseconds spent performing 0 L2C hits;

0 nanoseconds spent performing 0 L2C misses;

….

Table and Field Auditing using Hibernate Envers

**POM Dependency**

<dependency>

<groupId>org.hibernate</groupId>

<artifactId>hibernate-envers</artifactId>

</dependency>

@Data

@Entity

@Table(name = "blog")

@Audited

**public** **class** Blog {

@Service

**public** **class** AuditService {

@Autowired

**private** AuditRepo repo;

**public** **void** saveInfo(Blog blog) {

repo.save(blog);

}

}

@Id

@GeneratedValue

**private** Long id;

@Audited

**private** String name;

@Audited

**private** String title;

@Audited

**private** **int** version;

@NotAudited

**private** **int** size;

}

@Repository

**public** **interface** AuditRepo **extends** JpaRepository<Blog, Long> {

}

**Test Class**

**public** **static** **void** storeInfo(ApplicationContext applicationContext) {

AuditService service = applicationContext.getBean(AuditService.**class**);

Blog blog = **new** Blog();

blog.setName("My New Blog");

blog.setSize(20);

blog.setTitle("Java programming");

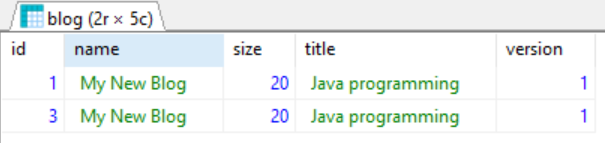
blog.setVersion(1);

service.saveInfo(blog);

}

}

**Database Tables**

Reference: <https://dzone.com/articles/all-jpa-annotations-mapping-annotations>

Eventual Consistency Vs Strong Consistency

**Eventual Consistency**

Eventual consistency guarantees that if an update is made to the data of a node (say node N), then the updated value will eventually be propagated to all the replicas of that node and eventually all the replica will become consistent to the original node(N). Nodes will get eventually consistent means it will take time for updates to reach other replicas. Hence eventual consistency is a consistency model used to achieve high availability and is a weak consistency model. Eventual consistency makes sure that data of each node of the database gets consistent eventually. *Eventual consistency offers low latency at the risk of returning stale data*

**Strong Consistency**

In contrast to eventual consistency, Strong consistency guarantee that if an update is made to a node (say N), then the updated value will be propagated to all the replicas (N1 & N2) of the node(N), immediately in other words after the update completes, any subsequent access (to N, N1, or N2) will always return the updated value. Note during the time these replicas are being updated with new data, any read/write request to any of these replicas will get delayed as all replicas are busy in keeping each other consistent and once they will become consistent with the original node they will start accepting read/write request again. As opposed to eventual consistency in strong consistency, always only one consistent state will be observed.

**What is the difference between CrudRepository and JpaRepository ?**

JpaRepository extends PagingAndSortingRepository which in turn extends CrudRepository.

Their main functions are:

* CrudRepository mainly provides CRUD functions.
* PagingAndSortingRepository provides methods to do pagination and sorting records.
* JpaRepository provides some JPA-related methods such as flushing the persistence context and deleting records in a batch.

Because of the inheritance mentioned above, JpaRepository will have all the functions of CrudRepository and PagingAndSortingRepository. So if you don't need the repository to have the functions provided by JpaRepository and PagingAndSortingRepository , use CrudRepository.

Below are the differences between [CrudRepository](https://docs.spring.io/spring-data/commons/docs/current/api/org/springframework/data/repository/CrudRepository.html) and [JpaRepository](https://docs.spring.io/spring-data/jpa/docs/current/api/org/springframework/data/jpa/repository/JpaRepository.html) as:

**CrudRepository**

1. CrudRepository is a base interface and extends the [Repository](https://docs.spring.io/spring-data/commons/docs/current/api/org/springframework/data/repository/Repository.html) interface.
2. CrudRepository mainly provides CRUD (Create, Read, Update, Delete) operations.
3. Return type of saveAll() method is [Iterable](https://docs.oracle.com/en/java/javase/11/docs/api/java.base/java/lang/Iterable.html).
4. Use Case - To perform CRUD operations, define repository extending CrudRepository.

**JpaRepository**

1. JpaRepository extends [PagingAndSortingRepository](https://docs.spring.io/spring-data/commons/docs/current/api/org/springframework/data/repository/PagingAndSortingRepository.html?is-external=true) that extends CrudRepository.
2. JpaRepository provides CRUD and pagination operations, along with additional methods like flush(), saveAndFlush(), and deleteInBatch(), etc.
3. Return type of saveAll() method is a [List](https://docs.oracle.com/en/java/javase/11/docs/api/java.base/java/util/List.html).
4. Use Case - To perform CRUD as well as batch operations, define repository extends JpaRepository.