

[<- Back to NucleoCore](#)

Spring Data Repository Library

- [Getting Started](#)
- [Connection Repository](#)
- [Data Repository](#)
- [EventListener](#)

Getting Started

NucleoDB Spring Repository Library makes it easier to use NucleoDB.

Installation

Dependencies

- Kafka Cluster
 - `/docker/kafka/docker-compose.yml`

Import library

```
repositories {  
    mavenCentral()  
    maven { url "https://nexus.synload.com/repository/maven-repo-releases/" }  
}  
dependencies {  
    implementation 'com.nucleodb:spring:3.3.49'  
}
```

Initializing DB

```
@SpringBootApplication  
@EnableNDBRepositories(  
    dbType = NucleoDB.DBType.NO_LOCAL, // does not create a disk of current up-to-date version of DB  
    // Feature: Read To Time, will read only changes equal to or before the date set.  
    //readToTime = "2023-12-17T00:42:32.906539Z",  
    scanPackages = {  
        "com.package.string" // scan for @Table classes  
    },  
    basePackages = "com.package.string.repos"  
)  
class Application{  
    public static void main(String[] args) {  
        SpringApplication.run(Application.class);  
    }  
}
```

```
}  
}
```

DataEntry model files

```
import java.io.Serializable;  
  
@Table(tableName = "author", dataEntryClass = AuthorDE.class)  
public class Author implements Serializable {  
    @Index  
    String name;  
    public Author(String name) {  
        this.name = name;  
    }  
    public String getName() {  
        return name;  
    }  
  
    public void setName(String name) {  
        this.name = name;  
    }  
}  
  
public class AuthorDE extends DataEntry<Author> {  
  
}
```

Repository for DataEntry

```
@Repository  
public interface AuthorDataRepository extends NDBDataRepository<AuthorDE, String> {  
    Set<AuthorDE> findByNameAndKey(String name, String key);  
    Author findByName(String name);  
    void deleteByName(String name);  
}
```

Connection model files

```
@Conn("CONNECTION_BETWEEN_DE")  
public class ConnectionBetweenDataEntryClasses extends Connection<ConnectingToDataEntryDE,  
ConnectingFromDataEntryDE> {  
    public ConnectionBetweenDataEntryClasses() {
```

```

    }

    public ConnectionBetweenDataEntryClasses(ConnectingFromDataEntryDE from, ConnectingToDataEntryDE to)
    {
        super(from, to);
    }

    public ConnectionBetweenDataEntryClasses(ConnectingFromDataEntryDE from, ConnectingToDataEntryDE to,
    Map<String, String> metadata) {
        super(from, to, metadata);
    }
}

```

Repository for Connections

```

@Repository
public interface ConnectionRepository extends NDBConnRepository<ConnectionBetweenDataEntryClasses,
String, ConnectingFromDataEntryDE, ConnectingFromDataEntryDE>{

}

```

Connection Repository

Connection repository integrates with the spring data repository. This enables your application to lookup based on either the destiny or the originating DataEntry.

Definition

```
@Repository
public interface AuthoredConnectionRepository
    extends NDBConnRepository<AuthoredConnection, String, BookDE, AuthorDE>{

}
```

This Repository definition contains not only the Connection (AuthoredConnection) but also the originating DataEntry (AuthorDE) and the destination DataEntry (BookDE).

Usage

```
@RestController
public class AnimeController{

    @Autowired
    AuthorDataRepository authorRepository;

    @Autowired
    AuthoredConnectionRepository authoredConnectionRepository;

    @GetMapping("/authored/{name}")
    public Set<AuthoredConnection> getByName(@PathVariable String name){
        Optional<AuthorDE> byName = authorRepository.findByName(id);
        if(byName.isPresent()){
            return authoredConnectionRepository.getByFrom(byName.get());
        }
        return null;
    }
}
```

```

@DeleteMapping("/authored/{name}")
public void deleteByName(@PathVariable String name){
    Optional<AuthorDE> byName = authorRepository.findByName(id);
    if(byName.isPresent()){
        authoredConnectionRepository.getByFrom(byName.get()).foreach(authored-
>authoredConnectionRepository.delete(authored));
    }
    return null;
}

@UpdateMapping("/authored/{name}/{country}")
public void updateCountryByName(@PathVariable String name, @PathVariable String country){
    Optional<AuthorDE> byName = authorRepository.findByName(id);
    if(byName.isPresent()){
        authoredConnectionRepository.getByFrom(byName.get()).foreach(authored->{
            authored.setCountry(country);
            authoredConnectionRepository.save(authored);
        });
    }
    return null;
}
}

```

Data Repository

Simplifies the retrieval and saving of data entries into the NucleoDB database.

Definition

```
@Repository
public interface AuthorDataRepository extends NDBDataRepository<AuthorDE, String>{
    Optional<AuthorDE> findByNameAndKey(String name, String key);
    Optional<AuthorDE> findByName(String name);
    void deleteByName(String name);
}
```

Usage

```
@RestController
public class AnimeController{

    @Autowired
    AuthorDataRepository authorRepository;

    @GetMapping("/get/{name}")
    public Optional<AuthorDE> getByName(@PathVariable String name){
        return authorRepository.findByName(id);
    }

    @DeleteMapping("/get/{name}")
    public void deleteByName(@PathVariable String name){
        Optional<AuthorDE> byName = authorRepository.findByName(id);
        if(byName.isPresent()){
            authorRepository.delete(byName.get());
        }
        return null;
    }

    @UpdateMapping("/get/{name}/{genre}")
```

```
public void updateCountryByName(@PathVariable String name, @PathVariable String genre){  
    Optional<AuthorDE> byName = authorRepository.findByName(id);  
    if(byName.isPresent()){  
        AuthorDE author = byName.get();  
        author.getData().setGenre(genre);  
        authorRepository.save(author);  
    }  
}  
}
```


EventListener

Data Entry

Code Sample

```
@RestController
public class BookController{
    @EventListener
    public void bookCreatedEvent(DataEntryCreatedEvent<BookDE> bookCreated){
        Serializer.log("Book created "+bookCreated.getDataEntry().getData().getName());
    }
}
```

Events

- DataEntryCreatedEvent
- DataEntryUpdatedEvent
- DataEntryDeletedEvent

Connection

Code Sample

```
@RestController
public class BookController{
    @EventListener
    public void bookConnectionCreatedEvent(ConnectionCreatedEvent<AuthoredConnection> authored){
        Serializer.log("Authored connection created "+authored.getConnection().getMetadata().get("rating"));
    }
}
```

```
}  
}
```

Events

- ConnectionCreatedEvent
- ConnectionUpdatedEvent
- ConnectionDeletedEvent