<- Back to NucleoCore

Models

Table Class

```
@Table(tableName = "author", dataEntryClass = AuthorDE.class)
public class Author implements Serializable {
  private static final long serialVersionUID = 1;

  @Index()
  String name;

  public Author() {}

  public Author(String name) {
    this.name = name;
  }

  public String getName() {
    return name;
  }

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

@Table

```
@Table(tableName = "author", dataEntryClass = AuthorDE.class)
```

tableName indicates the name of the table but also the topic that will be used by the MQS.

dataEntryClass points to the data entry class that encloses this table data class.

serialVersionUID

For the local cache this is needed for serializing and deserializing the data entries in the database. This will speed up startup for subsequent startups and will not grab all entries from the MQS.

@Index

This member variable will be indexed with the name of the variable. Nested objects in the data table class can also be indexed. You can specify the indexed key name by giving the Index annotation a value.

```
@Index("keyName")
```

When using a custom index key name you will need to use this key name for lookups.

DataEntry Class

Data Entry class encloses the table class. This is to differentiate the meta data from the actual data of the table. *Only the table class can have indexed member variables.*

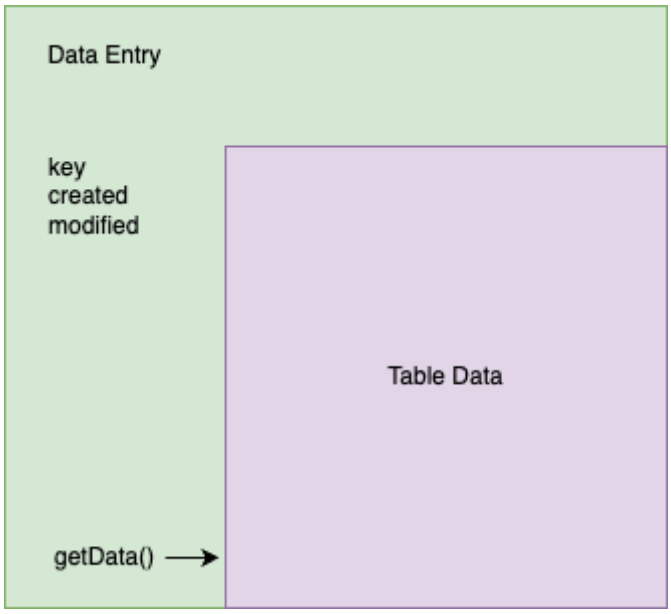
Author class

```
public class AuthorDE extends DataEntry<Author>{
  public AuthorDE(Author obj) {
    super(obj);
  }

  public AuthorDE(Create create) throws ClassNotFoundException, JsonProcessingException {
    super(create);
  }

  public AuthorDE() {
  }

  public AuthorDE(String key) {
    super(key);
  }
}
```

DataEntry

```
public class AuthorDE extends DataEntry<Author>
```

DataEntry contains metadata definitions and in the case above the *Author* class contains table data definitions. Metadata is maintained by the NucleoDB database. Modified/Created dates are based on the ledger in the MQS.



Constructors

Constructors are required for internal operations and instantiating a new DataEntry into the database.

Below is used when inserting a new Author table class into the database. This will also generate a new key using UUID.randomUUID().

```
public AuthorDE(Author obj) {
  super(obj);
}
```

Below is used when a Create is read in from the MQS

```
public AuthorDE(Create create) throws ClassNotFoundException, JsonProcessingException {
  super(create);
}
```

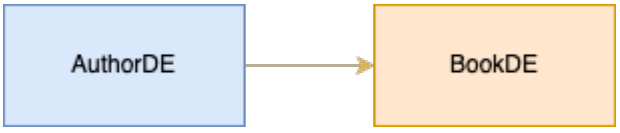
Connection

```
@Conn("AUTHORED")
public class AuthoredConnection extends Connection<BookDE, AuthorDE>{
  public AuthoredConnection() {
  }

  public AuthoredConnection(AuthorDE from, BookDE to) {
    super(from, to);
  }

  public AuthoredConnection(AuthorDE from, BookDE to, Map<String, String> metadata) {
    super(from, to, metadata);
  }
}
```

Connection

```
public class AuthoredConnection extends Connection<BookDE, AuthorDE>{
```

In the example above the AuthorDE is pointing to BookDE in a OneToMany relationship



@Conn(name)

Configures the name of the connection and the topic name used in MQS.

Indexing

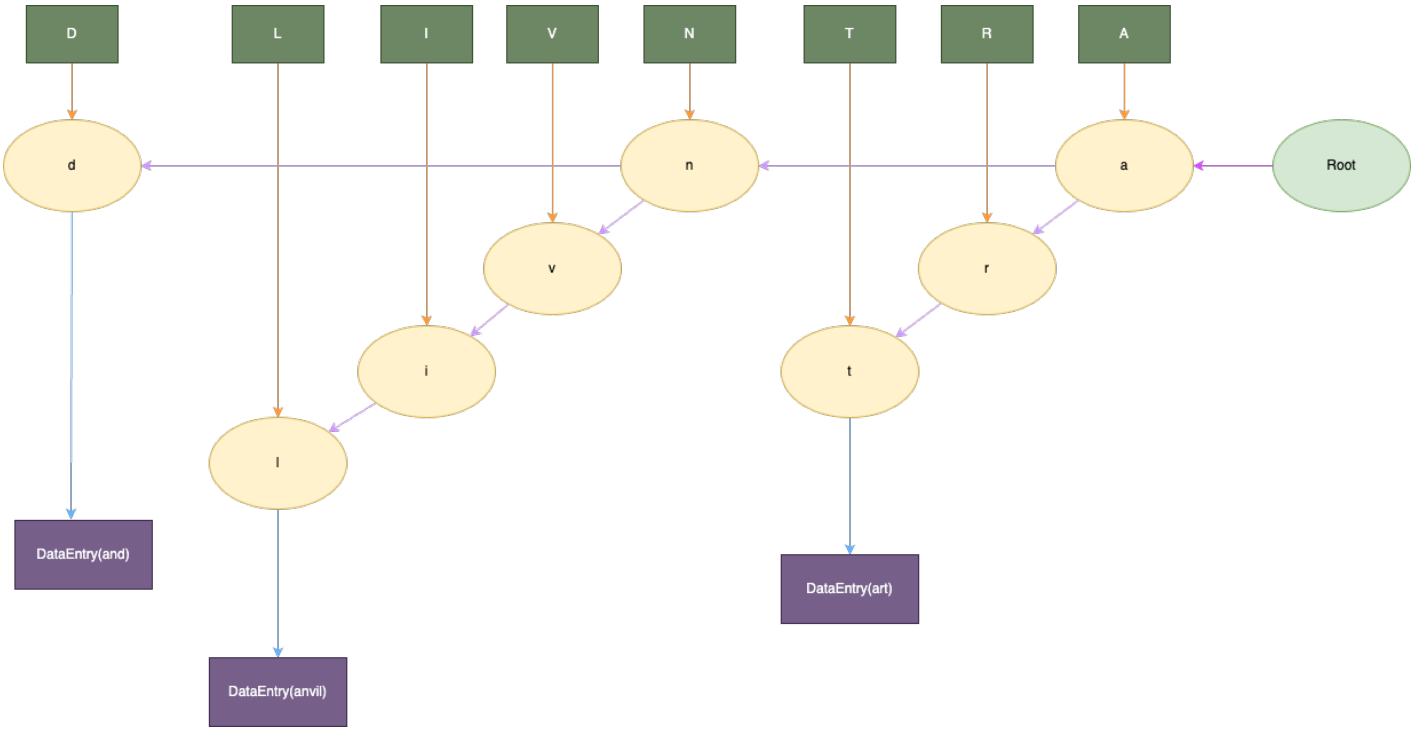
Currently there are 2 index types, **Tree** and **Trie**.

Trie Index

Supported value types: String

Heavy memory usage and allows for partial text search.

```
@Index(type = TrieIndex.class)
```



Tree Index (Default)

Support value types: Numbers, String, Dates

Lower memory usage and allows for lookup by object.

```
@Index(type = TreeIndex.class)
```