Memory

28/12/2023

Fernando José

2 DAM / Data Access

Índice

1 Transformer

The transformer code is divided into three main parts:

1. Create Objects

In this section, the functions extract_data and extractNestedEntity are defined, responsible for extracting data from the selected tables in the relational database and for handling the extraction of nested entities. Here, reflection is used to create instances of the entity classes and map them to the results of the SQL query.

2. Persist (Save)

The persist and persistEntity functions are responsible for persisting the extracted entities in the object-relational database (ObjectDB). The persist function handles the persistence transaction, while persistEntity is responsible for recursively persisting entities, including those that are nested objects.

3. Start executeTransformation

The executeTransformation function represents the main part of the transformer. Here, data extraction is performed from the selected tables using the functions defined in the previous sections. The extracted data is then persisted to the object-relational database (ObjectDB) at the specified location.

1 Transformer

1 Extract data and extractNestedEntity

The extract_data function extracts data from a specific table in a relational database and maps it to corresponding entity objects. It uses reflection to instantiate the entity class and maps the entity's fields to columns in the result set of an SQL query. In the process, it handles nested objects by calling the extractNestedEntity function recursively for those fields that represent relationships.

On the other hand, the extractNestedEntity function is responsible for extracting a nested entity by performing a recursive query in the relational database. Creates instances of the nested entity class and maps the fields to columns in the result set of an SQL query based on the primary key of the nested entity. It also handles nested objects recursively, ensuring that the nested entity is completely filled before being returned.

2 Persist and persistEntity

The persist function takes a list of entities and persists (saves) them into an ObjectDB database at the specified path. It uses an EntityManager to interact with the ObjectDB database. First, it starts a transaction, then iterates over each entity in the list and calls the persistEntity function to handle persistence, including recursive handling of nested objects. Finally, commit the transaction, close the EntityManager and the EntityManagerFactory to release resources.

The persistEntity function is responsible for persisting an individual entity in the ObjectDB database. It uses reflection to get the entity fields, and for each field, it checks if it is an object (not primitive) and if it is not a collection of entities. If so, it gets the nested object and persists it recursively by calling persistEntity again. This ensures that nested objects are properly persisted before the parent entity is persisted. In the end, the main entity is persisted if it is not a collection of entities. Any exceptions during the process are printed to the console.

3 ExecuteTransformation

This part is intended for the initialization of transformation objects and the execution of the data conversion process. The class has attributes to store the name of the relational database and the list of selected tables. Its constructor assigns these provided values as parameters. The executeTransformation method performs the transformation, obtaining a connection to the relational database and, for each selected table, extracting data using the extract_data function. This data is accumulated in a list called extractedData. Finally, the persist function is invoked to store the extracted data in an ObjectDB database, the path of which is provided as a parameter. If successful, a message is printed indicating that the conversion was successful. Together, this code encapsulates the essential logic for transforming data from a relational database to an ObjectDB database.

2 DAO

This code will be able to perform CRUD (Create, Read, Update, Delete) operations on entities. In this case, the DAO is parameterized by the type of entity (T) it will handle, which means that it can work with different types of entities in the database.

The DAO class has methods to perform basic CRUD operations, providing a consistent interface for interacting with the ObjectDB database. Each part is explained below:

1.Constructor (public DAO(Class<T> entityClass))

- The constructor takes the entity class (entityClass) as a parameter and stores it in a private field (this.entityClass).
- This constructor allows you to create instances of the DAO to operate with a specific entity class.
-

2.create(T entity) method

- This method creates a new entity in the database.
- Gets an EntityManager through the ObjectDBUtil.getEntityManager() utility class.
- Start a transaction using EntityTransaction.
- Attempts to persist the entity in the database.
- If the operation is successful, the transaction is confirmed; otherwise it is rolled back.
- Finally, the EntityManager is closed.

3.read(Object primaryKey) method

- This method retrieves an entity from the database based on its primary key.
- Gets an EntityManager.
- Use the find method to search for the entity by its primary key.
- Closes the EntityManager before returning the retrieved entity.

4. update(T entity) method

- Updates an existing entity in the database.
- Gets an EntityManager.
- Start a transaction.
- Use merge to synchronize the entity with the persistence context.
- Confirm or reverse the transaction depending on the result.
- Close the EntityManager.
-

5. delete(Object primaryKey) method

- Removes an entity from the database based on its primary key.
- Gets an EntityManager.
- Start a transaction.
- Use find to get the entity by its primary key.
- Delete the entity from the database.
- Confirm or reverse the transaction depending on the result.
- Close the EntityManager.

3 Map

Now I am going to explain how I have mapped the classes. For this I will use the Team class as an example:

The Team class is annotated with Java Persistence API (JPA) annotations to map to a specific table (team) in the ObjectDB database. Here is the breakdown of how the classes and associated functions have been mapped:

Class and Field Annotations:

@Entity: Indicates that the class is a JPA entity and must be mapped to a table in the database.

@Table(name = "team", catalog = "nasa_db"): Specifies the name of the table in the database (team) and the catalog to which it belongs (nasa_db).

@Id: Marks the TeamID field as the primary key of the entity.

@ManyToOne and @JoinColumn: Maps the many-to-one relationship between Team and Mission using the foreign key Mission_ID.

@OneToMany(fetch = FetchType.LAZY, mappedBy = "team"): Maps the one-to-many relationship to the Astronaut entity, indicating that the team property on the Astronaut entity is the inverse of this relationship.

@ManyToMany(fetch = FetchType.LAZY, mappedBy = "teams"): Maps a many-to-many relationship to the Ship entity, indicating that the teams property on the Ship entity is the inverse of this relationship.

Builders:

The no-arg constructor (public Team()) is required for JPA.

The constructor that takes an integer (public Team(int teamid)) is used to create an instance of Team with the specified identifier.

The main constructor (public Team(int teamId, Mission mission, String teamname, Set<Astronaut> astronauts, Set<Ship> ships)) is used to completely construct a Team object with all its attributes.

The constructor that takes a list of objects (public Team(List<Object> values)) is designed to construct a Team object from a list of values. This builder uses additional methods to set up the team's mission, astronauts, and ships.

Functions to Obtain Data:

setMision(int mission_id): Retrieves a mission from the database based on the mission ID associated with the team.

setAstronauts(): Retrieves the astronauts associated with the team from the database.

setNaves(): Retrieves the ships associated with the team from the database.

toString() method:

4 GUI

The graphical interface (GUI) called TransformerGUI is created in Java using the Swing library. This interface is the main interface of the "Database Transformer" application, which allows users to explore databases, select tables, and transform them using the Transformer program. The interface includes a JComboBox to select databases, a JList to display tables, and a JButton to initiate table transformation.

The TransformerGUI class inherits from JFrame, making it a Swing window. In the constructor, interface components such as the JComboBox, JList, and JButton are initialized and configured. Additionally, the database names are loaded into the JComboBox and the selected database change and transform button click events are configured.

The layout of the components in the interface is done in the initLayout() method, which uses a BorderLayout to arrange the components in the north, center, and south part of the window.

The application uses the ConnectionDB class to interact with the database, loading database and table names. The loadTableNames() method loads the table names into the JList based on the selected database. The transformSelectedTables() method starts the transformation process for the selected tables, calling the Transformer program with the tables as parameters.