Verwendung des ORM – Dusan Resavac if20b207

Der ORM wurde als „code-first“-Variante entwickelt.

Einrichtung

1. Mit Maven Pom.xml benutzen, um die Dependencies herunterzuladen
2. MariaDB (Dialekt von MySQL) installieren und starten – Eine mögliche Vorgehensweise wäre die Software Xampp herunterzuladen und den MySQL Server zu starten
3. Eventuell Schema erstellen, falls noch nicht vorhanden (school)
4. Datebank-Passwort und Benutzer gegebenenfalls ändern (in Main und in den Testklassen)
5. Unit-Tests ausführen (src > test > java)
6. Main() in src > main > java > Main ausführen

Klassen mit Datenbank mappen

Damit die Datenbanktabellen und – spalten mit den dazugehörigen Klassen verbunden werden können, werden folgende Annotationen bereitgestellt:

- `@EntityAnnotation(tableName = String.class)`
  Diese Annotation wird über einer Klasse platziert und mapped die Tabelle mit der Klasse. Sollte sich der Tabellenname vom Klassennamen unterscheiden, muss tableName spezifiziert werden.
- `@PrimaryKeyAnnotation`
  Markiert das Feld als Primary Key.
- `@FieldAnnotation`
  Markiert das Feld als Spalte in der Datenbank.
- `@OneToOne(remoteColumnName = String.class, isInTable = boolean.class)`
  Markiert das Feld als Teil einer 1:1 Beziehung. Dabei muss der Datentyp die Klasse der Gegenseite sein. Sollte die Klasse des Feldes den Wert nicht in der Tabelle speichern (Wenn die Gegenseite den Wert speichert), muss remoteColumnName auf den Namen der Datenbankspalte der Gegenseite gesetzt werden. In dem Fall muss auch das Attribut „isInTable“ auf „false“ gesetzt werden. Die Gegenseite muss nur das Attribut „isInTable“ auf „true“ setzen. Beispiel: Eine Maus hat ein Mausrad. Die Maus speichert den Schlüssel des Mausrads in der Tabelle (in der Spalte „mausrad“), während das Mausrad nur interne Felder in der Tabelle speichert. Nun müsste das Mausrad den remoteColumnName auf „mausrad“ und isInTable auf „false“ setzen.
- `@OneToMany(remoteColumnName = String.class)`
  Markiert ein Feld als 1-Seite in einer 1:n Beziehung. Das „remoteColumnName“ Attribut soll auf den Spaltennamen in der n-Tabelle gesetzt werden. Beispiel: Eine Klasse hat einen Lehrer und ein Lehrer unterrichtet mehrere Klassen. In der Klasse-Tabelle wird ein Feld „teacher“ stehen und muss als „remoteColumnName“ beim Lehrer angegeben werden. Der Datentyp des Feldes wird bei Lehrer eine List<Klasse> und bei der Klasse Lehrer sein.

- `@ManyToOne(columnName = String.class)`
  Markiert ein Feld als n-Seite in einer 1:n Beziehung. Das Attribut „columnName“ muss angegeben werden, wenn der Spaltenname nicht dem Feldnamen entspricht.
- `@ManyToMany(assignment_table = String.class, remoteColumnName = String.class)`
  Markiert ein Feld als n oder m in einer n:m Beziehung. Der Datentyp muss eine Liste von der Klasse des Gegenstücks (der n oder m Seite) sein. Das Attribut „assignment_table“ enthält den Namen der Zwischentabelle und „remoteColumnName“ enthält den Spaltennamen, den die Klasse in der Zwischentabelle hat, wo diese Annotation steht. Beispiel: Schüler besuchen Kurse und Kurse haben mehrere Schüler. In Kurs würde dann als Zwischentabelle z.B. „schueler_kurse“ und als „remoteColumnName“ „kurs“ angegeben werden.
- `@IgnoreAnnotation`
  Gibt an, dass das Feld vom ORM ignoriert werden soll.

Bei den Annotationen `@FieldAnnotation, @OneToOne, @OneToMany` und `@ManyToOne` gibt es die optionalen Attribute „nullable“ und „columnType“. „columnType“ muss fast nie gesetzt werden. `@FieldAnnotation` hat auch das Attribut „columnName“, mit dem der Spaltenname festgelegt werden kann.

WICHTIG: Die Annotationen werden auf die Felder der Klassen gelegt (nicht Methoden). Alle Felder, die vom ORM erfasst werden sollen, benötigen getter und setter Methoden, die automatisch anhand der Namenskonvention gefunden werden. Sollte eine Methode nicht gefunden werden, kann der Methodenname auch händisch als Attribut der Annotation angegeben werden.

ORM-Methoden

- `Orm.connect(String url)`
  Stellt die Verbindung zur Datenbank und dem spezifischen Schema her. Voraussetzung: Schema existiert bereits.
- `Orm.setLevel(Level level)`
  Stellt das Logging Level ein. Beim erstmaligen Ausführen wird Level.INFO oder Level.OFF empfohlen. Zum Debuggen sollte Level.FINE oder Level.FINER verwendet werden.
- `Orm.set_cache(Cache cache)`
  Setzt den zu verwendenden Cache. Als Option gibt es nur den TrackingCache. Bei den Testklassen sind keine Fehler aufgetreten, jedoch kann dies nicht für andere Klassen garantiert werden. Es wurden zwar einige Edge-Cases abgedeckt, jedoch läuft es darauf hinaus, dass nur bei Lesen Daten im Cache bleiben und beim Speichern verworfen werden (Auch die Objekte, die in einer Beziehung mit dem gespeicherten Objekt stehen). Bei merkwürdigem Verhalten wird empfohlen keinen Cache zu verwenden.
- `Orm.createTables(Class<?> … classes)`
  Erzeugt die Tabellen innerhalb des Schemas zu dem die Verbindung hergestellt wurde. Die angegebenen Klassen müssen in der korrekten Reihenfolge sein, da ansonsten Foreign Key Constraint Fehler auftauchen können.
- `Orm.clearTables(String schema)`
  Löscht die Daten/Zeilen aus allen Tabellen, die sich im angegebenen Schema befinden.

- `Orm.deleteTables(String schema)`
  Löscht (drop) alle Tabellen im angegebenen Schema.
- `Orm.delete(Object obj)`
  Entfernt das angegebene Objekt aus der korrespondierenden Tabelle.
- `Orm.save(Object obj, boolean changeReferencedObject)`
  Speichert ein Objekt in der Datenbank. Siehe Kommentar in der ORM-Klasse für eine genaue Beschreibung (Auch die Testbeispiele enthalten oft eine Beschreibung).
- `Orm.from(Class c)`
  Der Startpunkt für eine Query.
- `Orm.get(Class<T> t, Object pk)`
  Lädt das Objekt der Klasse t mit dem Primary Key pk aus der Datenbank und liefert das Objekt zurück.

Query / Abfragen

Abfragen werden mit `Orm.from(Class c)` gestartet. Diese Methode liefert ein SQLQuery-Objekt zurück. Mit diesem kann über ein Fluent-Interface die Abfrage gebaut werden. Sollen zum Beispiel alle Zeilen ausgegeben werden, kann `Orm.from(Class c).get()` verwendet werden. Dazwischen können die Methoden

- `or()`
- `and()`
- `group()`
- `groupEnd()`
- `not()`
- `is(String columnName, QueryOperations operation, Object… objects)`
- `isNot(String columnName, QueryOperations operation, Object… objects)`

verwendet werden. Die Reihenfolge muss mit der MariaDB Syntax konform sein.

Es stehen die folgenden QueryOperations zur Verfügung:

```
EQUALS, GREATER_THAN, GREATER_THAN_OR_EQUAL, LESS_THAN,
LESS_THAN_OR_EQUAL, LIKE, IN
```

Der 3. Parameter bei is(…) und isNot(…) enthält nur bei der QueryOperation IN mehrere Objekte (Sonst ist es nur ein(e) Objekt/Variable/Konstante).

Beispiele für korrekte Abfragen sind in der Klasse Testing.TestClasses.WithQuery zu finden (src > main > java > Testing > TestClasses > WithQuery).

Sonstiges

Unit-Tests sind im Ordner src > test > java zu finden. Leider konnte nicht herausgefunden werden, wie die Reihenfolge der Testklassen geändert werden kann. Daher werden die Methoden der OrmSaveTest-Klasse in den beiden anderen Testklassen erneut aufgerufen.

Für den Punkt „Framework weist weitere Features auf (zB. Transaktionen)“ wurde das automatische Erstellen und Löschen (drop table und Inhalte der Tabellen Löschen – Unterscheide zwischen Orm.clearTables und Orm.deleteTables) der Tabellen eingebaut.