ORM (Object-Relational Mapping) in SCL

Carsten Zaddach
BDE Business Datawarehouse
Engineering GmbH
Landsberger Str. 218
12623 Berlin
cz@bde-gmbh.de

Zusammenfassung

Die Verwendung von Objekten für den Zugriff auf relationale Daten bietet mehrere Vorteile. So wird die Komplexität von Tabellenzugriffen aus dem Programm in das entsprechende ORM-Framework verlagert. Ein weiterer Vorteil ist die Typüberprüfung, die bereits während der Kompilierung erfolgt und nicht erst während der Laufzeit des Programms. Ebenso können Typkonvertierungen, Referenzen auf andere Tabellen und Gültigkeitsbereiche von Werten definiert werden.

Schlüsselwörter: ORM, Objektrelationale Abbildung, SAS/AF

1 Einleitung

Die objektrelationale Abbildung bietet in objektorientierten Sprachen die Möglichkeit, die Objekte in relationalen Strukturen (Tabellen) anzulegen. Implementiert wird diese Technik durch entsprechende Frameworks, welche für viele objektorientierte Sprachen bereits zur Verfügung stehen.
Da das Konzept der Klassen und Objekte auch in der Screen-Control-Language (SCL) vorhanden ist stellte sich die Frage, ob eine objektrelationale Abbildung nicht auch dort möglich ist.

2 Konzept der objektrelationalen Abbildung

Das Konzept der objektrelationalen Abbildung beruht auf folgenden Punkten:

- Jede Tabelle hat eine eigene Klasse
- Jeder Datensatz einer Tabelle entspricht einem Objekt
- Zugriff auf die Felder bzw. Tabellenspalten nur über Getter und Setter
- Abfrage auf Tabellen über entsprechende Query-Klassen

Im Folgenden wird auf die einzelnen Punkte detailliert eingegangen.

Jede Tabelle hat eine eigene Klasse: Für jede Tabelle, auf die mittels ORM zugegriffen werden soll, muss eine eigene Klasse existieren. Diese Klasse, in den meisten Fällen

Base-oder Basis-Klasse genannt, stellt die direkte Abbildung der zugrundeliegenden Tabelle dar. Da diese Klasse jedoch bei jedem Generierungslauf durch das Framework neu erstellt wird, wird eine weitere, davon abgeleitete Klasse benötigt. In dieser abgeleiteten Klasse können individuelle Methoden hinzugefügt werden, ohne dass sie beim nächsten Mal überschrieben werden. Der Zugriff auf die Daten erfolgt nur über die abgeleitete Klasse. Durch die Basisklasse werden einige elementare Methoden zur Verfügung gestellt. Diese Methode sind u.a. `save` und `delete`.

Jeder Datensatz einer Tabelle entspricht einem Objekt: Jede Zeile wird durch ein eigenes Objekt repräsentiert. Dies betrifft sowohl das Ergebnis einer Abfrage als auch die Erstellung eines neuen Datensatzes. Enthält das Ergebnis einer Abfrage mehr als einen Datensatz, so werden alle entsprechenden Datensätze als separate Objekte in einer Liste oder einem Array zurückgeliefert.

Zugriff auf die Felder bzw. Tabellenspalten nur über Getter und Setter: Ein großer Vorteil bei der Verwendung der objektrelationalen Abbildung ist die Sicherstellung der Datenintegrität. So wird u.a. gewährleistet, dass numerische Spalten nur mit numerischen Werten gefüllt werden können. Zusätzlich zu den Standardmethoden können Methoden integriert werden, die eine Typkonvertierung z.B. bei Datumswerten vornehmen. So kann das beispielsweise das Datum entweder als numerischer Wert 20881 oder als String „03.03.2017“ übergeben werden.
Die automatische Generierung einer Id zur eindeutigen Identifizierung des Datensatzes ist ebenfalls ein Vorteil von ORM. Dabei erfolgt die Generierung völlig transparent für das aufrufende Programm.

3 Verwendung des Frameworks

Für jede verwendete Tabelle werden durch das Framework automatisch zwei Klassen erzeugt. Eine Basisklasse `B_<Tabellenname>.CLASS` und `<Tabellenname>.CLASS` wobei bei jeder neuen Generierung nur die Klasse `B_<Tabellenname>.CLASS` überschrieben wird. Damit können individuelle Erweiterungen an der Klasse `<Tabellenname>.CLASS` vorgenommen werden. Die Klasse `B_<Tabellenname>.CLASS` wiederum ist abgeleitet von der Klasse `BASETABLE.CLASS`.
Zum Ausführen der Abfragen werden ebenfalls zwei Klassen durch das Framework erstellt. Eine Basisklasse `B_<Tabellenname>_query.CLASS` und `<Tabellenname>_query.CLASS`. Auch hier wird die Klasse mit dem führenden B_ jedes Mal von neuem erstellt. Ähnlich wie bei der Basisklasse für die Tabelle ist die Basisklasse für die Abfrage von der Klasse `BASEQUERY.CLASS` abgeleitet. Die erstellten Klassen sind in Tabelle 2 noch einmal dargestellt.

3.1 Definition der Tabellen

Als Basis für den Generator dient eine XML-Datei. Diese enthält alle benötigten Tabellen mit deren Spalten und ggf. vorhandenen Referenzen. Nachfolgend ist eine solche XML-Struktur am Beispiel der Tabelle SASHELP.CLASS aufgeführt.

```
<?xml version="1.0" encoding="UTF-8"?>
<tablist ormcat="mycat.orm" >
  <table libname="sashelp" member="class" sasname="class">
    <column name="name" type="char" required="true"/>
    <column name="sex" type="char" required="true"/>
    <column name="age" type="num" required="true"/>
    <column name="height" type="num" required="true"/>
    <column name="weight" type="num" required="true"/>
  </table>
</tablist>
```

In der folgenden Tabelle sind die Attribute der einzelnen XML-Elemente im Detail beschrieben:

Tabelle 1: XML-Elemente und ihre Attribute

XML-Element	Attribut	optional	Beschreibung
tablist	ormcat		Name des Kataloges, in dem die generierten Klassen abgelegt werden sollen
table	libname		Name der Library, in der die Tabelle liegt
	member		Name der Tabelle
	sasname	x	ggf. aussagekräftigerer Name, der anstatt von member für den Namen der Klasse verwendet werden soll
column	name		Name der Spalte in der Tabelle
	type		Typ der Spalte char oder num
	required	x	true = Wert ist erforderlich
	primarykey	x	true = Spalte ist der Primärschlüssel (andernfalls werden ALLE Spalten zur Identifizierung des Datensatzes in der Tabelle verwendet)
	autoinc	x	true = Numerische Id soll automatisch hochgezählt werden
	sasname	x	ggf. aussagekräftigerer Name, der anstatt von name für den Namen der Methoden verwendet werden soll
	foreignkey	x	Name verweist auf die angegebene Tabelle zu der eine Referenz besteht
	default	x	Soll die Variable mit einem bestimmten Wert vorbelegt sein, so kann dieser hier angegeben werden

3.2 Generierung der Klassen

Die Generierung der Klassen erfolgt durch Aufruf des SCL-Programms `ORMGEN.SCL`. Dieses liest die XML-Datei ein und ermittelt mittels der Verwendung von regulären Ausdrücken

```
  colregex = prxparse ('/name=\"(\w+)\"/');
  typeregex = prxparse ('/type=\"(\w+)\"/');
  reqregex = prxparse ('/required=\"(\w+)\"/');
  incregex = prxparse ('/autoinc=\"(\w+)\"/');
  keyregex = prxparse ('/primarykey=\"(\w+)\"/');
  fkeyregex = prxparse ('/foreignkey=\"(\w+)\"/');
  defaultregex = prxparse ('/default=\"(\w+)\"/');
```

die Werte für die entsprechenden Attribute und speichert diese in einer Liste. Nach dem erfolgreichen Einlesen wird in einer temporären Datei für jede Variable eine entsprechende get- und set-Methode definiert. Zusätzlich zu den get- und set-Methoden wird in jeder Basisklasse eine Methode `save` und `delete` implementiert. Die Logik, ob es sich beim Speichern eines Datensatzes um einen neuen Datensatz handelt oder ein existierender aktualisiert werden soll, ist in der save-Methode abgebildet.

Da der generierte Code nicht direkt in eine SCL-Datei in einem Katalog geschrieben werden kann, ist der Umweg über eine temporäre Datei notwendig. Im Folgenden ist der Code für die Befüllung der SCL-Datei angegeben:

```
ormtab = "";
rc = filename(ormtab, "temp");

/*
  Einfügen des generierten Codes in die Temporäre Datei.
*/

* Den generierten Code einfügen *;
rc = preview('INCLUDE', ormtab);

* Save contents of preview buffer to SCL entry *;
rc = preview('SAVE', ormcat !! ".b_" !! table.sasname !! ".scl");

rc = preview('CLEAR');
rc = preview('CLOSE');

entry = "b_" !! table.sasname !! ".scl";

* Die erstellte Klasse kompilieren *;
SUBMIT CONTINUE;
   PROC BUILD CATALOG=&ormcat BATCH;
        COMPILE select=&entry;
   RUN;
ENDSUBMIT;
```

Durch den Generator werden für jede Tabelle insgesamt 4 Klassen erzeugt. Diese 4 Klasse sind folgende:

Tabelle 2: Übersicht über die erstellten Klassen

`B_<Member bzw. sasname>`	Basisklasse für die Tabelle. Wird bei jeder Generierung überschrieben.
`B_<Member bzw. sasname>_query`	Basisklasse für die Erstellung von Abfragen auf die Tabelle. Wird bei jeder Generierung überschrieben.
`<Member bzw. sasname>`	Klasse für die Tabelle und erbt alle Methoden von `B_<Member bzw. sasname>`. Diese Klasse wird bei der Initialen Erstellung erzeugt aber danach nicht mehr überschrieben. Individuelle Änderungen und Erweiterungen dürfen nur in dieser Klasse erstellt werden.
`<Member bzw. sasname>_query`	Klasse für die Erstellung von Abfragen auf die Tabelle. Diese Klasse wird bei der Initialen Erstellung erzeugt aber danach nicht mehr überschrieben. Individuelle Änderungen und Erweiterungen dürfen nur in dieser Klasse erstellt werden.

Bei neuen Mappings müssen die Klassen manuell erstellt werden. Dies erfolgt über den Menüeintrag `Datei -> Als Klasse speichern`. Bei der Erstellung ist die folgende Reihenfolge wichtig, da die Klassen aufeinander aufbauen bzw. einander verwenden:

1. B_<Member bzw. sasname>
2. <Member bzw. sasname>
3. B_<Member bzw. sasname>_query
4. <Member bzw. sasname>_query

3.3 Verwendung des Frameworks

Für den Zugriff auf Datensätze von Tabellen stehen jeweils zwei Klassen zur Verfügung. Die `<Member bzw. sasname>_query` -Klasse dient zur Erstellung und Ausführung der Abfragen und die `<Member bzw. sasname>`−Klasse beinhaltet die Ergebnisse der Abfrage.

Folgende Methoden stehen in jeder `<Member bzw. sasname>`−Klasse zur Verfügung:

Tabelle 3: Methoden der <Member bzw. sasname>-Klasse

Methode	Beschreibung
get<Variablenname>	Hat als Rückgabewert den Wert der Variablen des entsprechenden Datensatzes
set<Variablenname>	Setzt den Wert der Variablen auf den übergebenen.
save	Speichert den entsprechenden Datensatz ab
getNextID	Liefert die nächste freie Nummer zurück
delete	Löscht den entsprechenden Datensatz

Auch in der <Member bzw. sasname>_query-Klasse werden durch das Framework einige Standardmethoden implementiert. Folgende Tabelle beschreibt die Methoden, die für die Erstellung und Ausführung der Abfragen zur Verfügung gestellt werden:

Tabelle 4: Methoden der <Member bzw. sasname>_query-Klasse

Methode	Beschreibung
filterby<Variablenname>	Definiert einen Filter auf Basis der Variablen und des übergebenen Wertes. Soll der Wert als Ausschlusskriterium verwendet werden, so muss als zweites Argument das Wort „Not“ mit übergeben werden.
orderby<Variablenname>	Definiert die Sortierreihenfolge. Standardmäßig wird aufsteigend sortiert. Soll absteigend sortiert werden, so muss als Argument das Wort „desc“ mit übergeben werden.
find	Führt die Abfrage mit den gesetzten Filtern und Sortierungskriterien aus und gibt als Ergebnis eine Liste mit den gefundenen Datensätzen zurück. Wurde kein Datensatz gefunden, ist die Liste leer.
findbyId	Findet genau den Datensatz, bei dem die Schlüsselvariable den übergebenen Wert hat.

Für die Verwendung der query-Klasse muss zuerst ein entsprechendes Objekt der Klasse erstellt werden.

```
dcl SASHELPCLASS_QUERY query = _NEW_ SASHELPCLASS_QUERY();
```

Anschließend kann die Abfrage zusammengebaut und ausgeführt werden. An dem folgenden Beispiel sollen die Funktionen des Frameworks dargestellt werden.
Folgende Aufgabe soll gelöst werden: Es sollen alle Personen aus der Tabelle SASHELP.CLASS ermittelt werden, die männlichen Geschlechtes sind. Das Ergebnis soll außerdem aufsteigend nach dem Namen sortiert sein.

Tabelle 5 zeigt auf der linken Seite die Realisierung der Abfrage auf herkömmlichen Weg und auf der rechten Seite mittels Verwendung des Frameworks.

Tabelle 5: Gegenüberstellung herkömmlicher Weg und ORM

Herkömmlicher Weg	Mittels Framework

SUBMIT CONTINUE;
proc sort data =
sashelp.class(where=(sex="M")
)
out = result;
by name;
run;
ENDSUBSMIT;
 	
dsid = open("result");
 	
do while(fetch(dsid) = 0);
end;
 	
dsid = close(dsid);
	dcl SASHELPCLASS_QUERY query = _NEW_
	SASHELPCLASS_QUERY();
	
	ergebnisse = query.filterBySex("M")
	.orderByName().find();
	
	do i = 1 to listlen(ergebnisse);
	classobj = getitemo(ergebnisse, i);
	end;