Oracle Forms: How to create a Framework

Gerd Volberg
OPITZ CONSULTING Deutschland GmbH
Gummersbach

Keywords:

Oracle Forms, PL/SQL, Datenbank-API, Framework, Standards, GUI, Style Guide, Versionierung, Namenskonventionen, Referenz-Templates

Einleitung

Oracle Forms Applikationen sind ab einer bestimmten Größe nur durch den Einsatz von Frameworks effizient herstell- und wartbar.

Im einfachsten Fall sind es ein paar selbst geschriebene PL/SQL-Libraries die man in seiner Applikation nutzt. Im besten Fall setzt man PL/SQL- und Objekt-Libraries, Referenztemplates und Datenbank-API's ein.

Neben diesen technischen Hilfsmitteln braucht es auch eine gute Sammlung von Handbüchern, damit jedem Entwickler im Team neben seiner Expertise als Forms-Entwickler auch ein Leitfaden an die Hand gegeben wird, wie die zu entwickelnde Applikation aussehen soll.

Nicht fehlen dürfen ein genereller Style Guide, Namenskonventionen für Sourcecode und Objekte sowie ein Versionierungs-Handbuch.

Dieses Manuskript beschreibt die wichtigsten Elemente und Handbücher.

Style Guide

Innerhalb eines Style Guides sollten allgemeine Fragen beantwortet werden wie z.B.

- Surrogate Keys oder sprechende Schlüssel
- Programmier-Vorschriften
- Layout-Vorschriften

Surrogate Keys

In der relationalen Welt benötigt jede Tabelle einen Primärschlüssel. Vor dem Erzeugen des Datenmodells sollte man sich im Team darauf verständigen, ob man mit oder ohne sprechenden Schlüsseln arbeiten möchte.

Surrogate Keys benötigen pro Tabelle eine eigene Spalte für den Primary Key. Diese Spalte wird immer automatisch über eine Sequence befüllt und in den Detailtabellen über den Foreign Key verlinkt. Die Vorteile liegen auf der Hand:

- Einfachste Art der Nutzung und Befüllung

- Der Wert ist nicht änderbar
- Optimale Indizierung
- Wartungsfrei
- Jeder Foreign Key besteht immer nur aus einer Spalte

Sprechende oder auch zusammengesetzte Schlüssel nutzen eindeutige Spalten als ihren Primärschlüssel. Die Vorteile und Nachteile dieser Technik sind

- Foreign Keys, die auf sprechende Schlüssel verweisen, sind somit ebenfalls sprechend und aussagekräftig

Meines Erachtens überwiegen jedoch die Nachteile. Ein einfacher Update auf einem Primary Key wird hier schon zur Herausforderung, da sämtliche abhängigen Tabellen ebenfalls ein Daten-Update brauchen.

Programmiervorschriften

Hier sollte man Reglementierungen definieren, die in jedem Stück Sourcecode eingehalten werden müssen. Zum Beispiel

- GOTO verbieten. Wer mit GOTO arbeiten möchte, soll in BASIC weiterarbeiten
- LABEL verbieten.
- RETURN in einer Prozedur verbieten
- EXIT restriktiv einsetzen. Jede Schleife kann auch ohne Exit abgebrochen werden, indem ein einfacher Loop in eine WHILE- oder FOR-Schleife umgeschrieben wird.
- TAB's zur Einrückung verbieten. Stattdessen mit x Leerzeichen arbeiten (z.B. x=2)

Layout-Vorschriften

In diesem Bereich definiert man die Standardgrössen der Formsmasken. Zum Beispiel 1280*1024, damit man eine bildschirmfüllende Maske anbieten kann.

Um ein einheitliches Layout zu gewährleisten, sollte jeder Canvas in einer Maske in den Ruler Settings folgende Eigenschaften eingestellt haben:

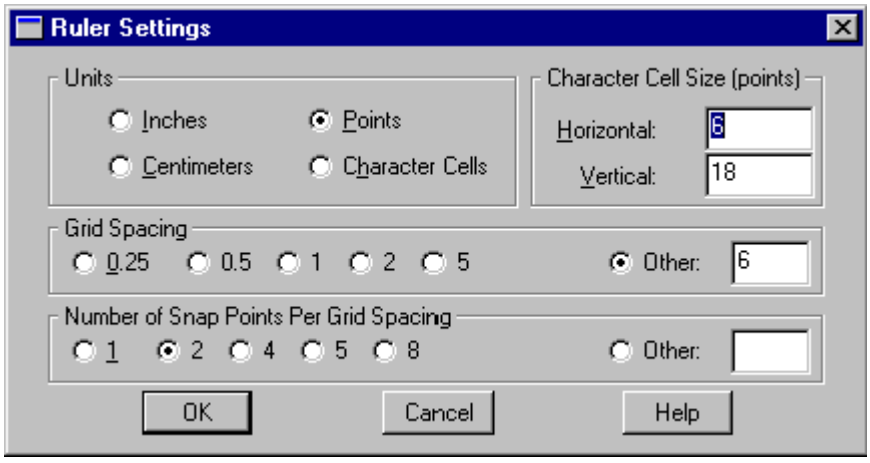

Abb. 1: Ruler Settings

Da alle Applikationen im Real-Koordinaten-Modus entwickelt werden, sollte desweiteren das Grid eingeschaltet sein und der Canvas nicht angezeigt werden.

Abb. 2: Forms-Menü „View“

Dies ermöglicht im Zusammenspiel mit den Ruler Settings eine Layout-Editor-Darstellung, mit der sehr effektiv gearbeitet werden kann:

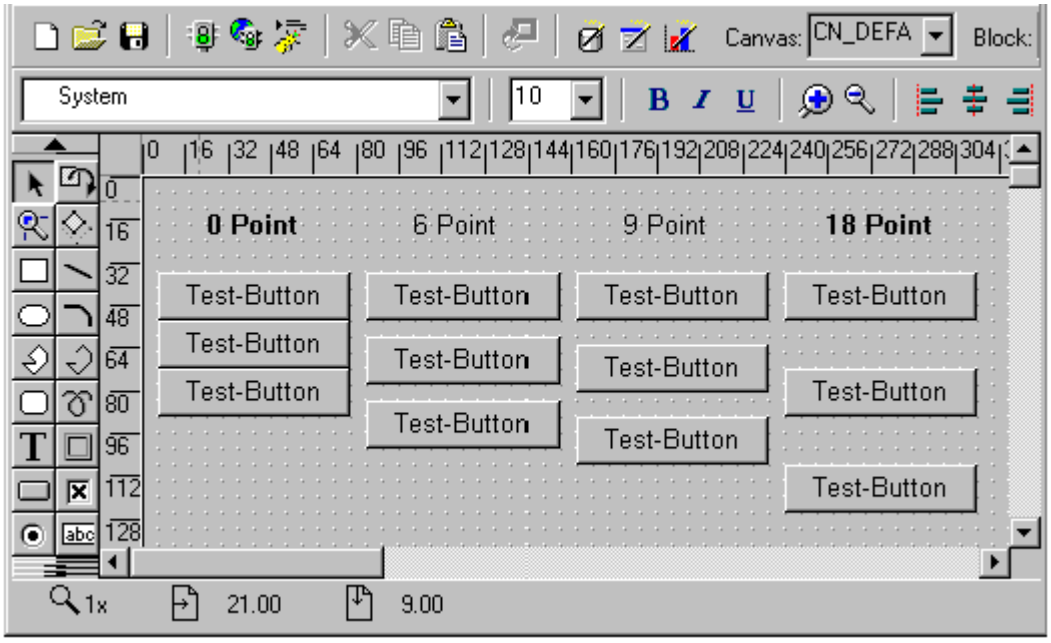

Abb. 3: Layout-Editor

Regeln für gutes Coding

In diesem Bereich sind alle Dinge aufgeführt, die man unter keinen Umständen tun sollte, bzw. Richtlinien, wie man arbeiten sollte.

Verbote

- Kein ActiveX, OCX, OLE, VBX einsetzen. Stattdessen JavaBeans
- Keine Combo-Boxen
- Keine WHEN-MOUSE-ENTER/LEAVE/MOVE - Trigger

Richtlinien

- Der Einfluss des Netzwerks auf die Ausführungsgeschwindigkeit der Applikation muss berücksichtigt werden.
- Mouse Trigger sollten nicht verwendet werden.
- Timer sollten nur sparsam verwendet werden.
- Begrenze die Anzahl der Image Items und Background Images.
- Die Netzwerk-Verbindungen sollten optimiert werden.
- Datenbankabfragen sollten so effizient wie möglich formuliert werden und nach Möglichkeit nur über Stored Objekte realisiert sein.

Namenskonventionen

Benamung von Objekten in der Datenbank ist sehr wichtig. Hier sollten strenge Konventionen gesetzt werden. Zum Beispiel:

Objekt	Notation	
Tabelle	<28 Zeichen>	Sprechender Name. Tabellennamen immer Plural.
Primary Key	<TabellenAlias> _ID	Vierstelliger eindeutiger Alias des Tabellennamens.
Foreign Key (1)	<ReferenzAlias> _ID	<ReferenzAlias> ist der Alias der Referenz-Tabelle.
Foreign Key (2)	<ReferenzAlias> "_“ <22 Zeichen> _ID	Mehrfachreferenz auf die gleiche Tabelle.
Foreign Key (3)	<TabellenAlias> "_“ <Primary Key>	Selbstreferenz auf die eigene Tabelle.
Tabellen-Spalte	<25 Zeichen>	Sprechender Name, 25 Zeichen max. wg View-Spalten innerhalb einer Tabelle.
View	<TabellenName> _V	View, die auf einer Tabelle aufbaut (Standard-View).
View	<TabellenName>[x] _V	Mehrere Views auf der gleichen Tabelle.
View	<28 Zeichen> _V	Der Name der View beinhaltet mindestens den Namen der wichtigsten Tabellen, die benutzt werden.
View	<27 Zeichen>[x] _V	Bei mehreren Views, die auf die gleichen Tabellen schauen.
View-Spalte	<TabellenAlias> "_" <Tabellen-Spalte>	Der Name einer View-Spalte setzt sich aus dem Alias der zugehörigen Tabelle und dem Originalnamen der Spalte in dieser Tabelle zusammen. Damit sind die Spalten der View immer eindeutig.
Sequence	<TabellenAlias> _SEQ	Die Standard-Sequence jeder Tabelle setzt sich aus dem Tabellen-Alias und "_SEQ" zusammen
Sequence	<TabellenAlias> "_“ <Spalte> _SEQ	Sequences, die nicht auf dem Primary Key arbeiten, heißen wie der Alias der Tabelle, zusammen mit dem Spaltenname, für den sie bestimmt sind und dem Suffix "_SEQ".

Interne Benamungen in der DB

Objekt	Notation	
Tabellen-Alias	<4 Zeichen>	Abkürzung der Tabelle (Unique über alle Projekte)
Primary Key	<Alias> _PK	
Foreign Key	<Alias_von>_<Alias_nach> _FK	
Foreign Key	<Alias_von>_<Alias_nach> _ [x] _FK	Falls mehrere benötigt werden => durchnummerieren
Unique Key	<Alias> _UK	Bei einem Unique-Key
Unique Key	<Alias> _ [x] _UK	Falls mehrere benötigt werden => durchnummerieren
Check Constraint	<Alias> _CHK	Bei einem Check Constraint
Check Constraint	<Alias> _ [x] _CHK	Bei mehreren
Index	<Alias> _I	Bei einem Index
Index	<Alias> _ [x] _I	Falls mehrere benötigt werden => durchnummerieren
DB-Trigger	<Alias> _[A|B] [R|S] [<I> <U> <D>]	A – After; B – Before R – Row; S – Statement I – Insert; U – Update; D – Delete

Oracle Forms Objekte

Objekt	Präfix
Blöcke	Kein Präfix, sprechender Name
Alert	AL_
Canvas	CN_
Editor	ED_
Frame	FR_
LOV	LV_
Object Group	OG_
Parameter	PA_
Popup Menu	PM_
Property Class	PC_
Record Group	RG_
Visual Attribute (Common)	VA_
Visual Attribute (Prompt)	VP_
Visual Attribute (Title)	VT_
Window	WN_
User Named Trigger	UN_

Item Typen

Objekt	Präfix
Bean Area	BA_
Button	BT_
Button-LOV	BT_LV_
Checkbox	CB_
Chart Item	CI_

Display Item	DI_
Hierarchical Tree	HT_
Image	IM_
List Item	LI_
OLE-Container	OC_
Radio Group	RA_
Radio Button	RB_
Sound	SO_
Text Item	TI_
User Area	UA_

PL/SQL

Objekt	Notation
Constant Variable	C_
Cursor	Cur_
Parameter	P_
Recordvariable	R_
Typ	T_
Variable	V_

Sourcecode-Layout

Gut geschriebener Sourcecode ist nicht nur fehlerfrei, sondern auch gut lesbar. Folgende Dinge sind hierbei wichtig und müssen vor Projektstart definiert werden:

- Einrückung
  - z.B. 2 Leerzeichen und niemals Tabs
- Groß / Kleinschreibung
  - Reservierte Worte werden gross geschrieben
  - Alle anderen Worte in Initcap
- Leerzeichen bei Parameterübergabe
  - Vor jeder Klammer ein Leerzeichen
  - Nach jedem Komma ein Leerzeichen
  - Vor und hinter jeder Zuweisung ein Leerzeichen
- Aliasnutzung beim Zugriff auf mehrere Tabellen
  - Wenn in einem Select die FROM-Clause mehr als eine Tabelle benutzt, sollte jede Tabelle mit einem Alias versehen werden. Aliase wie im Kapitel „Interne Benamungen in der DB“

Sourcecode-Formatierung

Jede denkbare und relevante Struktur in PL/SQL sollte von der Formatierung vorgegeben werden. Beispielsweise für SELECT, INSERT, UPDATE und DELETE

```
SELECT Auftrag_Nr,
       Datum
  FROM Auftraege
```

```
 WHERE Auftrag_ID = V_Auftrag_ID
   AND Datum > SYSDATE - 10
 ORDER BY Auftrag_Nr;

INSERT INTO Positionen
  (POSI_ID,
   AUFT_ID,
   Position_Nr,
   Datum,
   Anzahl,
   Wert)
VALUES
  (V_POSI_ID,
   V_AUFT_ID,
   '1',
   SYSDATE,
   '17',
   '12.50');

UPDATE Auftraege
   SET Kunden_Name = 'Müller'
 WHERE AUFT_ID = V_AUFT_ID;
```

Exception Handling

In jeder Routine muss am Ende ein Exception-Handling geschrieben werden. Beispiel:

```
EXCEPTION
  WHEN <Exception> THEN
    Statement1;
  WHEN OTHERS THEN
    Statement2;
END;
```

Kommentare

Ein effektiver Code-Stil hat zum Ziel, verständlichen, produktiven und wartbaren Code zu erzeugen. Die meisten Programme profitieren von den im Programm dokumentierten Kommentaren und Erklärungen.

Diese Kommentare erhöhen die Lesbarkeit des Source-Codes enorm. Entwickler, die den Source-Code in der Wartungsphase das erste Mal zu sehen bekommen, haben meist keine Chance, den Code zu verstehen, ohne eine gute Inline-Dokumentation.

Mit den nun folgenden Richtlinien kann man in hohem Maße selbst-dokumentierenden und auch für den, der den Code nicht geschrieben hat, gut lesbaren Code erzeugen.

Zuerst einmal einige generelle Tipps:

- Schreibe Source-Code geradeaus und ohne Tricks und Kniffe (egal, wie gut diese dokumentiert sind)
- Variablen-, Modul- und andere Namen bekommen sprechende Namen (egal, wie lang sie sind)
- Nutze immer Konstanten statt Literale als Werte (sowohl als Übergabewerte, wie auch als Vergleichswerte oder an anderen Stellen)
- Arbeite immer mit den gleichen, sauberen und durchgängigen Source-Code-Layout-Regeln (wie in diesem Dokument spezifiziert)
- Kommentiere den Source-Code während Du ihn schreibst.

Mit diesen Tips alleine kann schon eine Menge Dokumentation wegfallen, da der Source-Code viel übersichtlicher und lesbarer wird.

Dieses Manuskript kann nur einen kleinen Einblick geben in all die vielen Facetten, die in einem Framework enthalten sein sollten. Mittlere Frameworks benötigen für diese Handbücher meist schon viele hundert Seiten zur Dokumentation. Viel Spaß bei der Präsentation im November.

Kontaktadresse:

Gerd Volberg
OPITZ CONSULTING Deutschland GmbH
Kirchstr. 6
51647 Gummersbach

Phone: +49(0) 2261-6001 0
Fax: +49(0) 2261-6001 4200
Email: gerd.volberg@opitz-consulting.com
Blog: http://talk2gerd.blogspot.com