PL/SQL, jQuery, CSS:
Sourcecode in APEX clever organisieren

Andreas Wismann
WHEN OTHERS
D-41564 Kaarst

Schlüsselworte

APEX, Oracle Application Express, Anwendungsentwicklung, Quellcode, SQL, PL/SQL, JavaScript, jQuery, HTML, CSS

Einleitung
Bei der APEX-Entwicklung, insbesondere im Team, entstehen oft sehr komplexe Anwendungen. Selten beschränkt man sich auf die selbsterklärenden Standardkomponenten, die APEX "out of the box" mitbringt. Je mehr individuellen Quellcode man an diversen Stellen spontan hinzufügt, desto wichtiger ist es, dass alle Änderungen nachvollziehbar organisiert und kommentiert werden. Schließlich möchte man auch Wochen (Monate, Jahre) später die Abläufe noch verstehen, und nicht vor einer Wüste scheinbar unzusammenhängender Codeschnipsel kapitulieren. Diese Herausforderung besteht selbstverständlich in allen Entwicklungsumgebungen, doch APEX mit seinem "Rapid Application Development"-Ansatz und den zahlreichen Quellcode-Eingabemöglichkeiten macht es besonders attraktiv, nachlässig zu werden ...

Anhand eines typischen APEX-Projekts zeige ich neue Best Practices und effiziente Tricks, wie man diese Problematik von vornherein elegant entschärft. Und zwar so, dass es von den Entwicklern nicht als Belastung empfunden wird. Sie behalten nicht nur die funktionale Übersicht über Ihre Anwendung, sondern können sogar jederzeit einen aktuellen Fortschrittsbericht generieren und alle offenen Punkte auflisten. Es bedarf nur eines leichtgewichtigen, schnell erlernbaren und dennoch maschinenlesbaren Konzepts!

Der „einfache“ Weg: Man schreibe einfach drauflos …
Ohne Umschweife: Um innerhalb weniger Minuten mit Oracle Application Express eine funktionierende Demo-Anwendung zu erstellen, mit der Sie die Endanwender oder Entscheider nachhaltig beeindrucken, dabei würde Sie ein tiefergehendes Konzept nur langsamer machen. Zu diesem Zweck müssen Sie lediglich die bereitgestellten APEX-Komponenten verstehen und abrufen. Sie benötigen ein Datenbankschema und einen Browser, und wenige Mausklicks später krönt ein interaktiver Report Ihre kleine, leistungsstarke Applikation.

In Organisationen, die APEX als strategische Plattform einsetzen, entstehen binnen kurzer Zeit aber zahlreiche miteinander verwobene Anwendungen mit umfangreicher Navigation und Funktionalität, sprich: Die Menge an „Assets“ (Seiten, Regionen, Reports, Formulare, Items, …) steigt erfahrungsgemäß – und rasant. Dazu gehört oft auch eine große Menge an Applikations-Programmcode. Damit steht man früher oder später vor zwei Überlegungen, um dem Quellcode eine praktikable Organisationsstruktur zu geben:

- Wo genau platziere ich den Quellcode in meiner Entwicklungsumgebung?
- Wie formuliere und kommentiere ich den Quellcode, um bestimmte Stellen später sicher wiederzufinden?

Der Vortrag zeigt zahlreiche praxisnahe Tipps aus vielen Jahren Programmierarbeit in großen Anwendungen. Als Appetithäppchen für die Veranstaltung möchte ich in diesem Manuskript eine kleine Auswahl an Themen ausführlicher vorstellen.

APEX-Objekte an Ort und Stelle kommentieren
99 Prozent der Kommentarfelder in APEX-Anwendungen bleiben leer. Warum? Es liegt wohl nicht allein an mangelnder Fantasie der Entwickler, sondern eher daran, dass diese Kommentarfelder ihr Dasein weit abgelegen und wenig prominent am Fuße eine langen scrollbaren Liste fristen. Mit einem relativ einfachen Trick manipulieren Sie die CSS-Darstellung im Page Designer, und fortan fixiert sich das Kommentarfeld weit oben und schreit praktisch danach, beschrieben zu werden. Wie das funktioniert, sowie Vorschläge, welche Informationen Sie in den Kommentar schreiben sollten, folgen in der Präsentation. Details sind nach der Veranstaltung als Download verfügbar.

PL/SQL-Blöcke: Ab damit in die Datenbank!
Man kann es wirklich nicht oft genug wiederholen … Die Editorfelder für PL/SQL-Programmcode, die in APEX quasi „an jeder Ecke“ auf unsere Eingaben warten, sollten tunlichst nicht mit Quellcode überfrachtet werden. Schon einfachste Ablauf- und Entscheidungsstrukturen (`IF … THEN … ELSE … END IF`) sind in der Datenbank wesentlich besser aufgehoben, denn vor der Ausführung muss der hinterlegte Programmcode erst geparst und kompiliert werden, inklusive unserer Tippfehler. Es wäre ideal, dort nur einzelne Befehle zu hinterlegen, die einfach nur Prozeduren oder Funktionen in der Datenbank aufrufen und dabei Werte von APEX-Sessionvariablen übergeben beziehungsweise einlesen.

Zudem besteht in APEX leider keine permanente Statuskontrolle: Wird beispielsweise ein Datenbank-View ungültig, so erhalten abhängige PL/SQL-Packages unmittelbar danach den Status INVALID und lassen sich nicht mehr ausführen, so lange, bis der zugrunde liegende Fehler in der View beseitigt ist. In APEX jedoch wird der Quellcode, der sich auf diese View abstützt und „irgendwo“ in einem Editorfenster lebt, eben nicht INVALID. Hier gibt es zwei Möglichkeiten: Entweder schlägt der (möglicherweise unbemerkte) Fehler erst zur Laufzeit beim User auf, wobei sich entweder die gesamte Anwendung nicht mehr öffnen lässt oder zumindest in Teilen nicht mehr funktioniert. Dies ist übrigens ein wesentliches Unterscheidungsmerkmal zu Oracle Forms! Die andere Möglichkeit ist, regelmäßig den APEX Advisor zu benutzen, denn der findet solche Unstimmigkeiten, indem er jeden einzelnen PL/SQL-Block in der APEX-Anwendung „probeweise“ kompiliert und anschließend die aufgetretenen Fehler in einem Report ausgibt.

Hashtag: @hashtag
Twitter hat es erfolgreich vorgemacht: In einer großen Menge von scheinbar unstrukturierten Informationen (und damit lässt sich schnell wachsender Programmcode ja durchaus vergleichen) fügt man wichtige Suchhinweise in Form von (frei wählbaren) Hashtags hinzu. Warum sollten man sich dieses einfache Prinzip nicht auch für die Programmierung zunutze machen? Aus inzwischen fünfjähriger Erfahrung, in denen ich konsequent mein eigenes Tagging-System einsetze, kann ich nur Positives berichten. Sei es das Auffinden von Baustellen, offenen Fragen, Problemsituationen, mutmaßlichen Fehlern oder das spätere Erstellen von Berichten zum Stand meines Quellcodes: Mit diesen Hashtags steuere ich einen erheblichen Teil meiner täglichen Entwicklungsarbeit.

Oft komme ich in Kontakt mit bereits existierenden, umfangreichen Anwendungen und erhalte die Aufgabe, den vorhandenen Stand weiterzuentwickeln. Typischerweise besteht eine meiner ersten Aufgaben darin, den bestehenden Quellcode zu inspizieren, um einen Eindruck von der Funktionsweise und dem Programmierstil zu bekommen. Bereits zu diesem Zeitpunkt nutze ich die

erste Gelegenheit, den bestehenen Quellcode mit Anmerkungen zu versehen (im Vortrag werde ich erläutern, warum ich hierfür nicht durchgängig englische Ausdrücke benutze). Beispiele:

```
-- @unklar
-- @veraltet
-- @performance
-- @recherche
-- @testen
-- @mueller
```

Es kann sich durchaus empfehlen, diese Editierungen in einer Kopie der Anwendung vorzunehmen, um den Originalzustand zunächst nicht zu verändern. Übrigens: Aufgrund der Tatsache, dass in SQL, PL/SQL, JavaScript, jQuery, CSS, HTML das Zeichen @ üblicherweise nicht als syntaktisches Element verwendet wird (im Gegensatz zum bekannten Hashtag-Prefix #), erhält es den Vorzug.

Jegliche „Baustellen“ und offene Punkte erhalten drei Schrägstriche `///`, da diese Zeichenfolge in keiner von APEX genutzten Programmiersprache eine syntaktische Bedeutung hat (dieses Vorgehen stammt übrigens aus den Best Practices beim Programmieren mit JavaScript; dort beginnen Zeilenkommentare sowieso mit einem doppelten Slash `//`):

```
-- /// bis zum @2018-03! klären
```

Werde ich unterbrochen und kann erst am Folgetag meine Arbeit fortsetzen, so kennzeichne ich das an der betreffenden Stelle noch schnell mit dem Tag

```
-- /// @weiter
```

Sie mögen nun argumentieren, dass man dabei schnell die Übersicht verlieren kann. Da würde ich Ihnen sogar recht geben, gäbe es nicht die vielen hilfreichen Datenbank- und APEX-Views, mit denen Sie Zugriff auf sämtliche von Ihnen ersonnene Kommentare erhalten, und besäßen wir nicht Reguläre Ausdrücke als Suchwerkzeug, um einen kompletten Katalog aller hinzugefügten Hashtags mit deren Inhalten erstellen zu können.

Die Abfragen und Ergebnisse dieser mächtigen Volltextsuchen zeige ich im Vortrag.

Konstanten verwenden

Für jedes größere Projekt erstelle ich ein Package namens CONST ohne Body, welches nur Konstanten-Deklarationen enthält. Dort stehen etwa folgende (beispielhafte) Zeilen:

```
dokumenttyp_eingangsrechnung   CONSTANT dokumenttypen.id%type := 3;
dokumenttyp_ausgangsrechnung   CONSTANT dokumenttypen.id%type := 7;
dokumenttyp_gutschrift         CONSTANT dokumenttypen.id%type := 4;

yes  CONSTANT NATURALN := 1;
no   CONSTANT NATURALN := 0;

one  CONSTANT NATURALN := 1;

SUBTYPE t_max_varchar2     IS VARCHAR2(32767);
max_varchar2_length        CONSTANT INTEGER := 32767;
```

Für die ersten drei Zeilen erhalte ich vermutlich noch Ihre Zustimmung, doch welchen Vorteil sollen die `NATURALN`-Typdefinitionen bringen?

Wer jemals in einem Meer von Quellcode die verschiedenen Variationen von TRUE und FALSE ergründen musste (und zwar nicht als `BOOLEAN`-Datentyp, da dies mit SQL ja nicht funktioniert), der wird nachempfinden können, wie wichtig eine einheitliche und nachvollziehbare Regelung ist. Man erinnere sich nur an `"TRUE"` und `"FALSE"` (VARCHAR2-Ausdrücke aus Oracle Forms, und ich habe mehr als ein Mal geprüft, ob `l_return = "FLASE"` ist …), oder etwa `"Y"` und `"N"`, `"Yes"` und `"No"`, oder war es doch `"YES"` und `"NO"` ? Und wo, zum Kuckuck, war das nochmal nachzulesen?

Das Schmankerl zum Schluss: USER_IDENTIFIERS
Falls Ihnen der Sinn des folgenden Statements nicht ins Auge springt, freuen Sie sich auf den Vortrag.

```
CREATE OR REPLACE PACKAGE const AS
  SUBTYPE t_max_varchar2 IS VARCHAR2(32767);
END const;
/
CREATE OR REPLACE PACKAGE test1 AS
  v_max_varchar2 const.t_max_varchar2;
END test1;
/

SELECT *
  FROM user_identifiers;
```

Ausblick auf den Vortrag
Die Präsentation am Mittwoch, 22. November um 9:00 Uhr im Raum Sydney zeigt anhand praxisnaher Live-Demos viele dieser „Rezepte“ (nicht zur solche zum Thema PL/SQL!), die Sie beim Umgang mit großen Mengen an Quellcode und Kommentaren in die Lage versetzen, Ihre Individualprogrammierung in APEX effektiver und effizienter in den Griff zu bekommen.

Freuen Sie sich unter anderem auf:

- Praxisgerechte Namenskonventionen
- Ein Report über bevorstehende Deadlines, erzeugt aus Quellcode-Kommentaren
- Bedingtes Platzieren von jQuery-Funktionen
- Generieren von dynamischen CSS Stylesheets zur Laufzeit

Kontaktadresse:
Andreas Wismann
WHEN OTHERS Inh. Andreas Wismann
Hirschstr. 10
D-41564 Kaarst

Telefon: +49 (0) 2131 - 314 9966
mobil: +49 (0) 176 - 7800 3109
E-Mail: wismann@when-others.com
Internet: http://when-others.com