Dieser Artikel zeigt, wie man mehrsprachige Apex-Anwendungen erstellt, ohne dass für jede Sprache ein separates Exemplar der Anwendung aufgebaut sein muss. Durch Pflege der Texte in der Datenbank und eine dynamische Übersetzung zur Laufzeit lässt sich erheblicher Aufwand bei den Deployment-Prozessen und bei der Wartung von Apex-Applikationen sparen.

Mehrsprachige Apex-Anwendungen mithilfe von dynamischen Übersetzungen aus der Datenbank heraus

Christian Piasecki, TEAM Partner für Technologie und angewandte Methoden der Informationsverarbeitung GmbH

Die Funktion der Mehrsprachigkeit, die Apex von Hause aus mitbringt, ist im Prinzip gut gelöst. Das Verfahren, die Texte in eine Übersetzungsdatei zu exportieren und nach dem Übersetzen wieder zu importieren, ist an sich komfortabel gelöst. Diese Standard-Vorgehensweise besitzt aber einen großen Nachteil. Dadurch, dass für jede Sprache ein separates Exemplar der Anwendung erstellt wird, steigt der Aufwand bei Deployment und Wartung aller „Parallel"-Anwendungen enorm. Sobald es Änderungen an der Anwendung gibt – und mögen diese auch nur den PL/SQL-Code betreffen – sind die Anwendungen für alle unterstützten Sprachen neu zu generieren und zu deployen. Geht man hier von drei technischen Systemen pro Anwendung aus (Entwicklung, Test und Produktion) führt das zu erheblichen Mehraufwänden. Um diese zu verringern, wird hier ein alternativer Ansatz vorgestellt, der sich schon in mehreren Kundenprojekten bewährt hat.

Ziel des hier vorgestellten Ansatzes ist es, den Aufwand für die Bereitstellung einer mehrsprachigen Anwendung möglichst gering zu halten. Gelöst wird diese Aufgabenstellung, indem man von der Standard-Vorgehensweise abweicht und die Abwicklung der Mehrsprachigkeit in die Datenbank-Schicht verschiebt.

Kernfunktionalität in der Datenbank

Die Kernfunktionalität der Mehrsprachigkeit wird in die Datenbank ausgelagert und besteht aus einer Übersetzungstabelle und einem PL/SQL-Package. Die Übersetzungstabelle hält die Information darüber, welches in der Oberfläche darzustellende Textelement in welcher Sprache mit welchem tatsächlichen Text angezeigt wird. Das Package übernimmt die Aufgabe, diese sprachbezogenen Daten in die Übersetzungstabelle einzutragen und auszulesen.

Es bestehen nun unterschiedliche Möglichkeiten, diese Tabelle aufzubauen. Eine möglichst schlanke ist nachfolgend vorgestellt. Es reicht im Grunde aus, wenn die Übersetzungstabelle aus drei Spalten besteht: ID, LANG, LSTRING (siehe Listing 1).

```
create table T_LANG_STRING
( ID                  VARCHAR2(40) not null,
  LANG              VARCHAR2(20) not null,
  LSTRING             VARCHAR2(4000) not null
);

alter table T_LANG_STRING
  add constraint CO_UK_LANG_STRING unique (ID, LANG);
```

Listing 1

Die erste Spalte beinhaltet eine entsprechende ID, die das verwendete Textelement in der Anwendung referenziert. Die zweite Spalte wird mit einer eindeutigen Sprach-Kennung gefüllt und die dritte Spalte enthält den tatsächlichen Text in der gewählten Sprache, der dann oberflächenseitig angezeigt wird. Eine gefüllte Tabelle kann dann folgendermaßen aussehen (siehe Tabelle 1).

In dieser Tabelle werden für alle Objekte, die in der Apex-Anwendung verwendet werden, zentral die Übersetzungstexte gepflegt. Tragen unterschiedliche darzustellende Objekte die gleiche Bezeichnung, müssen die Sprach-Informationen nur einmal zentral in der zugrundeliegenden Tabelle gepflegt werden und stehen an allen zugehörigen Stellen zur Verfügung. Das Eintragen und Auslesen der Übersetzungen wird durch ein PL/SQL-Package realisiert, das im Wesentlichen aus zwei Funktionen besteht. Blendet man Tracing und Parameterüberprüfung einmal aus, reichen wenige Zeilen Code (siehe Listing 2). Sobald dieses Grundgerüst steht, kann mit der Einbindung in die Apex-Applikation begonnen werden.

Verwendung in der Apex-Oberfläche

Um eine mehrsprachige Maske, wie in den Abbildungen 1 und 2 gezeigt, zu erhalten, kann die Apex-Applikation ganz normal entwickelt werden. Die

ID	LANG	LSTRING
USER	DE	Benutzer
USER	EN	User
LANGUAGE	DE	Sprache
LANGUAGE	EN	Language

Tabelle 1: Übersetzungstabelle für den einfachen Ansatz der Mehrsprachigkeit

Abbildung 1: Benutzermaske auf Deutsch

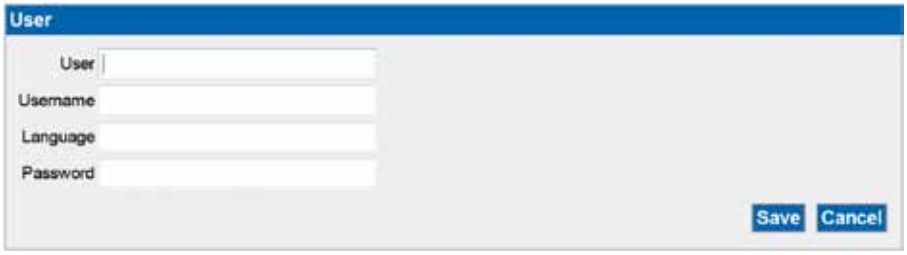

Abbildung 2: Benutzermaske auf Englisch

Mehrsprachigkeit kommt erst bei der Beschriftung der Labels, Buttons und Tabellen-Überschriften zum Tragen.

Die einzelnen Oberflächen-Objekte werden wie bewährt erstellt. Neu hinzu kommt, dass zu jedem Element der Oberfläche ein Element für den Beschriftungstext generiert wird. Die Elemente für die Beschriftungen werden in einer separaten Region erstellt und sind vom Typ „Ausgeblendet“. Als Quelltyp wird für die „Beschriftungselemente“ die Option „PL/SQL-Funktions-Body“ ausgewählt und mit dem folgenden Funktionsaufruf (exemplarisch für das Element „P3_USER_ LANG“) gefüllt:

```
begin
return pk_ui.get_label(p_id =>
,USER‘, p_lang => :F1_LANG);
end;
```

Zwei Punkte der Implementierung sind jetzt noch offen:

1. Was steht in dem globalen Element „F1_LANG“ und wie und wo wird das Element gefüllt?
2. Angenommen „F1_LANG“ hat den Wert „DE“, dann steht in „P3_ USER_ LANG“ jetzt der Inhalt „Benutzer“. Doch wie wird dieser Inhalt als Beschriftung verwendet?

Füllen der Beschriftung

In dem Element „P3_USER“ muss als Inhalt für das Label Folgendes eingetragen werden: „&P3_USER_LANG.“. So wird sichergestellt, dass der Wert des Elements „P3_USER_LANG“ als Beschriftung verwendet wird (siehe Abbildung 3). Dieses Vorgehen kann natürlich auch auf alle anderen Seiten-Objekte wie Region, Buttons, Seitenüberschriften etc. und auch auf Applikations-Elemente wie Navigationen oder Registerkarten angewendet werden. Zu unterscheiden ist nur, dass bei einigen das Feld „Label“ und bei anderen das Feld „Titel“ zu füllen ist. Auch eine mehrsprachige Darstellung von Tabellenüberschriften, wie in den Abbildungen 4 und 5 gezeigt, lässt sich so auf einfache Weise realisieren.

```
create or replace package body
pk_lang_string is

   function get_label
   (
      p_id   in varchar2,
      p_lang in varchar2
   ) return varchar2 is

      v_output varchar2(4000);

   begin
      select lstring
        into v_output
        from t_lang_string
       where lang = p_lang
         and id = p_id;
      return v_output;
   end get_label;

   function add_lstring
   (
      p_id      in varchar2,
      p_lang    in varchar2,
      p_lstring in varchar2
   ) return boolean is

      v_exists number;

   begin
      begin
         select 1
           into v_exists
           from t_lang_string ls
          where ls.id = p_id
            and ls.lang = p_lang;
      exception
         when no_data_found then
            v_exists := 0;
      end;

      if v_exists > 0
      then
         return false;
      else
         insert into t_lang_string
            (id,
             lang,
             lstring)
         values
            (p_id,
             p_lang,
             p_lstring);

         return true;
      end if;
   end add_lstring;

end pk_lang_string;
```

Listing 2

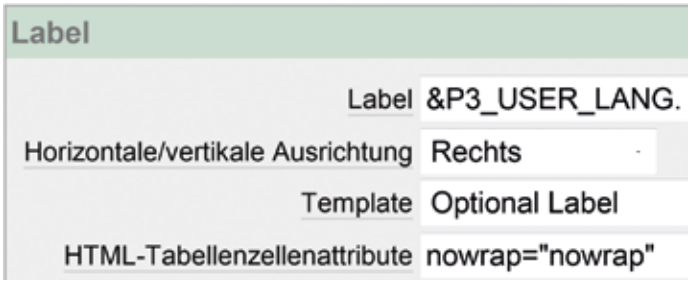

Abbildung 3: Füllen des Labels mit dem Wert eines Elements

Abbildung 4: Tabellenüberschriften auf Deutsch

Abbildung 5: Tabellenüberschriften auf Englisch

Abbildung 6: Login

Abbildung 7: Datumsformat

In den Berichtsattributen wird als Überschriftstyp die Option „PL/SQL“ gewählt und folgender Code hinterlegt (siehe Listing 3). Jetzt bleibt noch zu klären, wie das Anwendungselement, das die gewählte Sprache beinhaltet, gefüllt wird.

Ermitteln und Setzen der vom Benutzer gewünschten Sprache

Das praktikabelste Vorgehen ist hier, den Login-Dialog der Apex-Anwendung zu überarbeiten und ein Feld für die Sprachauswahl zu erstellen (siehe Abbildung 6). Hier kann der Benutzer zwischen den angebotenen Sprachen auswählen und diese werden bei Weiterleitung der Seite in das globale Element für die Spracheinstellung geschrieben (siehe Listing 4).

Im nächsten Schritt kann man sogar weitergehen und sicherstellen, dass der Benutzer auch die Login-Seite in der Sprache angezeigt bekommt, in der er sich beim letzten Mal ein-/ausgeloggt hat. Hierzu wird ein Prozess vor dem Laden der Seite erzeugt, der sich die Daten aus einem Cookie heraus ermittelt (siehe Listing 5).

Auf dieses Weise wird erreicht, dass eine Apex-Anwendung ohne Programmieraufwand schnell und flexibel auf eine andere Sprachversion umgestellt werden kann. Unter „Mehrsprachigkeit“ fällt aber nicht nur die Übersetzung von Texten.

Um dem Systembenutzer ein komfortables Arbeiten zu ermöglichen, sollten auch die Datums- und Zahlenfelder in dem für die Sprache korrekten Format dargestellt werden. Diese erweiterte Fragestellung der Mehrsprachigkeit wird in dem folgenden Absatz vorgestellt.

Länderkonforme Datumsformate

Um auch Datumsformate länderkonform darzustellen, gibt es die Möglichkeit, die Formate pro Sprache in einer Registry-Tabelle abzulegen und beim Darstellen von Datums- oder Zahlenformaten zu verwenden. Listing 6 zeigt ein mögliches Beispiel, um ein Datumsformat auszulesen.

Ist so eine Funktion vorhanden, kann, analog zu dem Beispiel oben, ein globales Element der Anwendung damit gefüllt werden.

```
begin
return pk_ui.get_date_format
(p_lang => :F1_LANG);
end;
```

Für die Formatierung von Datumsfeldern kann nun auf den Wert des Elements zurückgegriffen werden (siehe Abbildung 7).

Weitere Möglichkeiten und Fazit

Die hier vorgestellte Implementierung der Mehrsprachigkeit in Apex soll als alternativer Ansatz zum Standard-Vorgehen betrachtet werden. Insbesondere lässt er sich weiter ausbauen. Auch das Handling bei den Funktionsaufrufen kann weiter vereinfacht werden, indem man zum Beispiel auf der Seite 0 eine Package-Konstante mit der Sprach-Einstellung füllt und so die Parameter-Übergabe komfortabler wird. Weitere Ausbaustufen sind denkbar und wünschenswert.

Warum empfehlen wir dieses Vorgehen zur Umsetzung von Mehrsprachigkeit?

Der reine Implementierungsaufwand für die benannten (und auch weiter ausgebauten) Formen der Mehrsprachigkeit ist zunächst erkennbar höher als bei „normalen“ Vorgehensweisen. Der Spareffekt findet sich aber im Nachhinein im Lebenszyklus der Anwendungen, etwa bei Wartung und weiterer Konfiguration. Es muss immer nur eine Version der Anwendung gepflegt und „deployed“ werden und die Erweiterung um zusätzliche Sprachen findet im Rahmen einer Datenveränderung in einer Tabelle statt und nicht im Rahmen einer Strukturänderung der Anwendung oder von Tabellen.

Positiv ist zudem, dass der Benutzer seine Sprache frei wählen kann und nicht auf die Einstellungen seines Web-Browsers festgelegt ist.

```
begin
 return
 pk_ui.get_label(p_id => ‚USER‘, p_lang => :F1_LANG)
 ||‘:‘|| pk_ui.get_label(p_id => ‚USERNAME‘, p_lang =>   :F1_
 LANG)
 ||‘:‘|| pk_ui.get_label(p_id => ‚LANGUAGE‘, p_lang => :F1_
 LANG);
 end;
```

Listing 3

```
declare
begin
:F1_LANG := upper(:P101_LANGUAGE);
:F1_USER := upper(:P101_USER);
end;
```

Listing 4

```
declare
    v varchar2(255) := null;
    c owa_cookie.cookie;
begin
   c := owa_cookie.get(‚LOGIN_USERNAME_COOKIE‘);
   :P101_USER := c.vals(1);
   c := owa_cookie.get(‚LANG_LOCALE_COOKIE‘);
   :P101_LANGUAGE := c.vals(1);
exception when others then
null;
end;
```

Listing 5

```
function get_date_format(p_lang varchar2) return varchar2 is
   v_ret varchar2(100);
begin
   v_ret := pk_mds.get_string_value(p_path => ‚dateformat.supported.‘
||p_lang);
   return v_ret;
exception
   when others then
      v_ret := pk_mds.get_string_value(p_path => ‚dateformat.de-
fault‘);
      return v_ret;
end get_date_format;
```

Listing 6

Christian Piasecki
TEAM GmbH
cpi@team-pb.de