Oracle als Baukasten aus Standardsoftware für Individuallösungen

Gerhild Aselmeyer
Analysen, Konzepte & Anwendungsentwicklung für EDV
Essen

Schlüsselworte
Oracle Text, UTL- und andere Packages, VPD, XMLDB, APEX.

Einleitung
Folgende Anforderungen in Ausschreibungen oder von Kunden/ Fachabteilungen vorgetragen, sehen auf den ersten Blick nicht nach schnellen Lösungen aus, sondern nach aufwändigen Großprojekten:

- Eine Wissensdatenbank aufbauen inklusive Indizierung von Texten und Dateien zur komfortablen Recherche aber mit spezifischen Zugriffsrechten

oder

- Daten aus dem Internet über eine XML-Schnittstelle einbinden und dem Anwender fachgerecht präsentieren mit der Möglichkeit Teile der Daten in die eigene Datenhaltung zu übernehmen und die Anwendung mandantenfähig,

alles mit moderner Technik möglichst ohne Client-Installation, offen für Erweiterungen, da die geforderte Funktionsliste noch unvollständig ist, ...
Das muss aber nicht sein, wenn man sich den Funktionsumfang einer Oracle Datenbank zunächst einmal richtig ansieht und die Möglichkeiten wirklich ausschöpft. Mit APEX, Text, UTL_* (z.B. HTTP, FILE, ...), XMLDB und VPD, einem sauberen Datenmodell, dass die wesentlichen Basisattribute beinhaltet und offen für Erweiterungen gestaltet ist, lassen sich die Anforderungen mit geringem Programmieraufwand in wenigen Tagen umsetzen. Der Lieferumfang einer Oracle Datenbank ist inzwischen ein Baukasten aus Standardsoftware, um Individuallösungen zusammenzusetzen.
Einwände gegen solche Lösungen wegen fehlender Trennung von Persistenz-, Logik- und Präsentationsschicht lässt sich entgegenhalten, dass zwar physikalisch alles in der Oracle Datenbank gespeichert ist, aber logisch die 3-Schicht-Architektur eingehalten wird. Diese sieht wie folgt aus:

- Die Datenhaltung erfolgt in Tabellen des Anwendungsschemas (wie bei jeder Datenbankanwendung).
- Die Logik, programmiert in PL/ SQL, ist gekapselt in Prozeduren, Funktionen und Packages.
- Die Präsentation über APEX lässt sich auch durch eine selbst programmierte Oberfläche ersetzen – aber mit höherem Aufwand.

Im folgenden finden Sie die wesentlichen Schritte zum Aufbau einer Wissensdatenbank mit Schlagwortrecherche in Beiträgen und Dokumenten mit selbst definierbarer Zugriffsbeschränkung auf Gruppen- und Nutzerebene sowie einer externen Recherche bei der Deutschen Nationalbibliothek. Der lauffähige Prototyp ist in weniger als 15 PT aus den Bauteilen Oracle Text, VPD, XMLDB, APEX und des Paketes UTL_HTTP zusammengesetzt.

Aufbau der Datenbank
Um lediglich Texte und Dokumente zu speichern und mit einem Schlagwortindex zu versehen, in welchem Anwender hinterher recherchieren können, benötigen wir vier Tabellen und eine View:

- Eine für Dokumententypen: hier wird festgelegt, ob für Dokumente dieses Typs eine Verschlagwortung vorgenommen wird.
- Eine für Dokumente: diese enthält Angaben zu sowie einen Verweis auf Dokumente (BFILE)

oder auch die Dokumente selbst (BLOB).

- Eine für Anwenderbeiträge: hier werden 'Wissensbeiträge' von Anwendern oder auch Texte aus anderen Systemen (z.B. einer Kundendienstanwendung) über entsprechende Schnittstellen zu weiteren Auswertung gespeichert.
- Eine zur Zuordnung von Dokumenten zu Beiträgen.

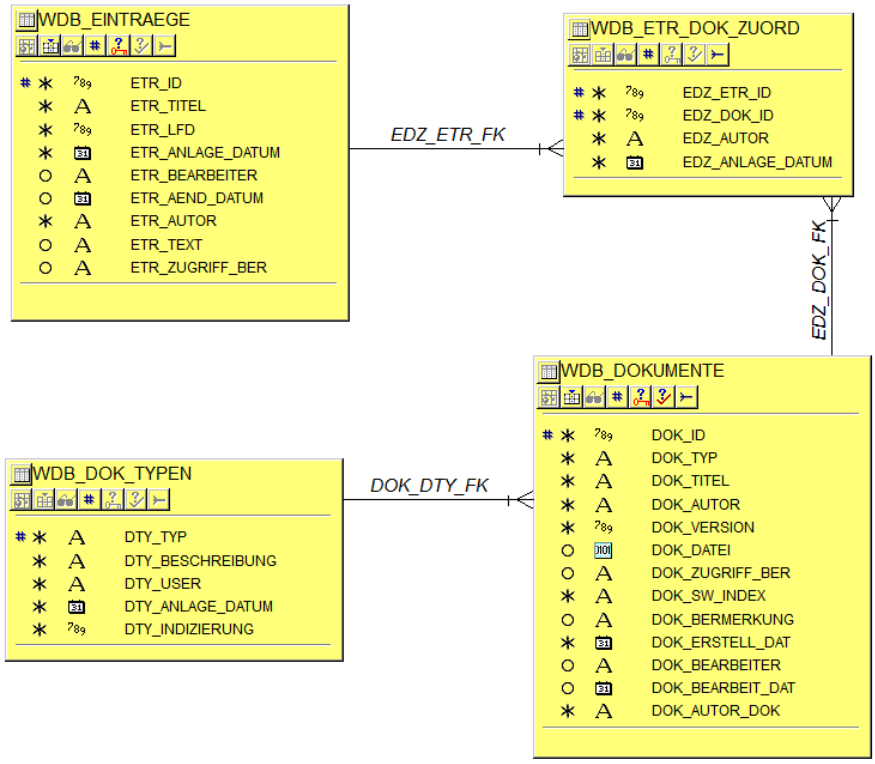

Abb. 1: Tabellen zum Speichern und Recherchieren von 'Wissen'

Außerdem erstellen wir eine View, um bereits gespeicherte Stoppworte nach Anfangsbuchstaben geordnet anzuzeigen. Die Basistabelle liefert Oracle Text – leider nicht gefüllt (zumindest in einer aus älteren Versionen migrierten Datenbank), wie die Dokumentation verspricht.

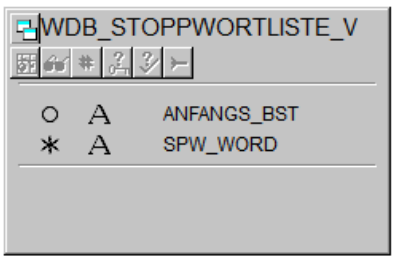

Abb. 2: View gespeicherter Stoppworte

Bei Stoppworten handelt es sich um Artikel, Präpositionen, Konjunktionen etc. - also Worte die in Texten oft vorkommen, aber für den Dokumenteninhalt irrelevant sind und daher nicht in einen Schlagwortindex aufgenommen werden sollten.

Möchten wir zusätzlich noch eine externe Recherche z.B. von der Deutschen Nationalbibliothek mit

optionaler Speicherung des Ergebnisses anbieten, benötigen wir eine weitere Tabelle:

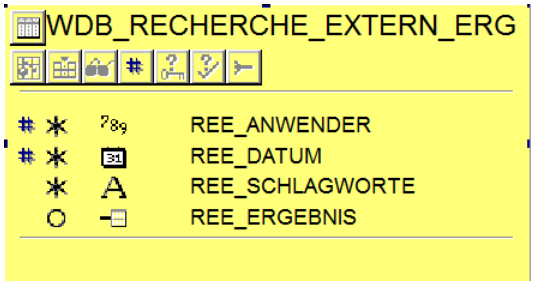

Abb. 3: Tabelle für Ergebnis einer externen Recherche im XML-Format

Darin ist die Spalte REE_ERGEBNIS ein XMLTYPE, um die Antwort eines Web-Service im XML-Format aufzunehmen.

Etwas aufwändiger gestaltet sich die Möglichkeit Dokumente und Anwenderbeiträge mit individuellen Zugriffsrechten auszustatten. Zwar benötigt man lediglich zwei Tabellen (s. Abb. 4), aber um die Rechte auf Gruppen- und Individualebene anzulegen und zu steuern, legen wir zusätzlich vier Views (s. Abb. 5) an.

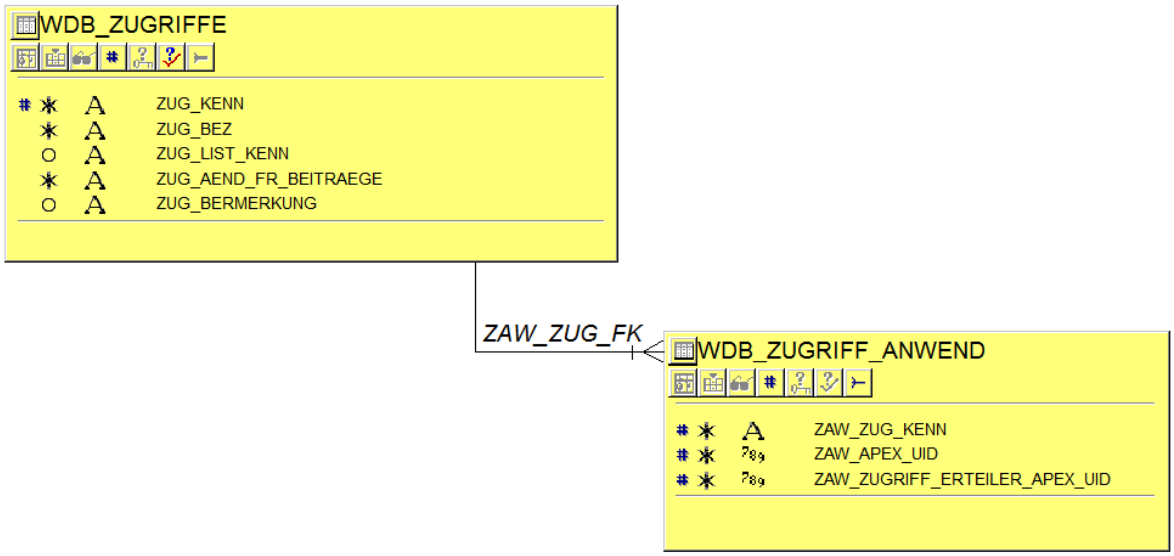

Abb. 4: Tabellen für Zugriffssteuerung auf Dokumente und Anwenderbeiträge

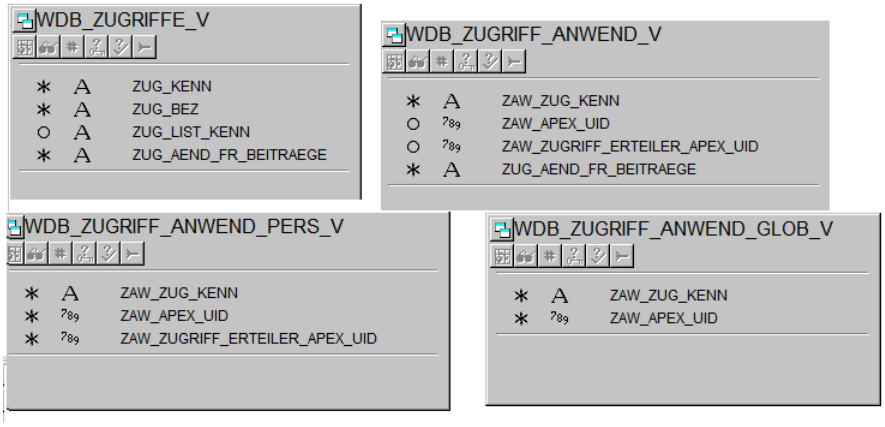

Abb. 5: Views für Zugriffskontrolle

Damit ist die Persistenzschicht für unsere Wissensdatenbank bereits abgeschlossen – zumindest für einen Prototypen. Für den Schlagwortindex stellt uns Oracle die notwendigen Speicherobjekte zur Verfügung. Dieses Schema lässt sich selbstverständlich mühelos erweitern, um weitere Bausteine hinzuzufügen: z.B. eine automatische E-Mail-Benachrichtigung, wenn neue Einträge erfasst werden oder eine Bereinigung veralteter Ergebnisse externer Recherchen, was sich die Nutzer individuell einstellen können.

Wie bei jedem Design sollte auch hier bereits zu Beginn einige Regeln berücksichtigt werden, damit auch bei wachsendem Datenvolumen die Anwendung akzeptable Antwortzeiten aufweist:

- Natürlicher Schlüssel in jeder Tabelle,
  denn der Anwender will diesen auf jeder Ebene sehen und Verknüpfungen von mehreren Tabellen, um diesen zusammen zu suchen, trägt nicht zu kurzen Antwortzeiten bei.
- Tabellen und Indices in verschiedenen Tablespaces speichern,
  da sich hierdurch die physikalischen Zugriffe i.a. verringern lassen – und die Physik ist immer noch das langsamste Glied in der Prozesskette.
- Tabellen mit sehr unterschiedlichen Satzlängen in verschiedenen Tablespaces speichern,
  da sich hierdurch unnötige Lücken vermeiden lassen und damit wiederum i.a. physikalische Zugriffe.
- Indexorganisierte Tabellen zu Indices,
  da der Optimizer diese wie Indices behandelt.
- Storage-Parameter gezielt einsetzen (s. Oracle® Database Performance Tuning Guide 11g Release 2 (11.2)):
  nutzen Sie automatische Segment-Verwaltung für Tablespaces und die voreingestellten Werte für die Storage-Parameter, damit habe ich bisher i.a. gute Ergebnisse erzielt.

Verarbeitungslogik – inklusive der Schlagwortindizierung

Für die Schlagwortrecherche fehlt uns jetzt noch das Wichtigste: die Verschlagwortung. Dafür treffen wir einige Vorbereitungen, um Oracle Text einzubinden. Zunächst erhält das Datenbankschema die Rolle CTXAPP und die Ausführungsrechte für die bereitgestellte Verarbeitung:

```
GRANT CTXAPP to WDB_ADM;
GRANT EXECUTE ON CTXSYS.CTX_ADM TO WDB_ADM;
GRANT EXECUTE ON CTXSYS.CTX_CLS TO WDB_ADM;
GRANT EXECUTE ON CTXSYS.CTX_DDL TO WDB_ADM;
GRANT EXECUTE ON CTXSYS.CTX_DOC TO WDB_ADM;
GRANT EXECUTE ON CTXSYS.CTX_OUTPUT TO WDB_ADM;
GRANT EXECUTE ON CTXSYS.CTX_QUERY TO WDB_ADM;
GRANT EXECUTE ON CTXSYS.CTX_REPORT TO WDB_ADM;
GRANT EXECUTE ON CTXSYS.CTX_THES TO WDB_ADM;
GRANT EXECUTE ON CTXSYS.CTX_ULEXER TO WDB_ADM;
```

Außerdem leiten wir aus den mitgelieferten Vorlagen eigene für Sprachbasis, Wortliste und die Tablespace-Zuordnung ab, um diese ggf. bearbeiten zu können und vor allem um die fehlenden Stoppworte nachzuladen:

```
BEGIN
   CTX_DDL.CREATE_PREFERENCE('Grundeinstellung_LEX', 'BASIC_LEXER');
   CTX_DDL.CREATE_PREFERENCE('Grundeinstellung_SWL', 'BASIC_WORDLIST');
   CTX_DDL.CREATE_STOPLIST('Grundeinstellung_SWS');
   CTX_DDL.CREATE_PREFERENCE('Grundeinstellung_SPE', 'BASIC_STORAGE');

   CTX_DDL.SET_ATTRIBUTE('Grundeinstellung_SPE', 'I_INDEX_CLAUSE',
                         'tablespace WDB_INDEX_01 compress 2');
```

```
   CTX_DDL.SET_ATTRIBUTE('Grundeinstellung_SPE', 'K_TABLE_CLAUSE',
                         'tablespace WDB_INDEX_01');
   CTX_DDL.SET_ATTRIBUTE('Grundeinstellung_SPE', 'N_TABLE_CLAUSE',
                         'tablespace WDB_INDEX_01');
END;
```

Um auch für die Tabellen und Indices des Schlagwortindex die oben aufgeführten Regeln für Performanz zu berücksichtigen, fügen wir TABLESPACE-Zuordnungen für Indices und als Index organisierte Tabellen der Präferenz für die Speicherung als Attribute hinzu.

Für die Dokumente legen wir dann einen `CONTEXT` Index an, dem außer den vorher angelegten Vorgaben noch die Spalte zur Kennzeichnung des Dokumentenformates als Parameter mitgegeben wird:

```
CREATE INDEX DOK_SW_IDX1
ON WDB_DOKUMENTE(DOK_DATEI)
INDEXTYPE IS CTXSYS.CONTEXT ONLINE
PARAMETERS('lexer Grundeinstellung_LEX wordlist Grundeinstellung_SWL
storage Grundeinstellung_SPE stoplist Grundeinstellung_SWS FORMAT COLUMN
DOK_SW_INDEX');
```

Das Dokumentenformat ist

- TEXT für HTML oder einfachen Text,
- BINARY für alle anderen für Indizierung unterstützten Dokumente (PDF, MS DOC, RTF u.v.m),
- IGNORE für nicht unterstütze Formate wie Bilder.

Für die Bemerkung zu Dokumenten und die Anwenderbeiträge kann man auch einen `CTXCAT` Index einsetzen, vor allem wenn zusätzlich weitere Spalten wie z.B. der Titel oder das Erstellungs-/ Anlagedatum in die Abfrage einbezogen werden soll. Aber um einfach und schnell erst einmal zum Ziel einer reinen Schlagwort-Recherche zu gelangen, habe ich auch für diese Spalten CONTEXT Indices erstellt.
Auf eine automatische Synchronisation der Indices habe ich verzichtet, da mir dies bei einer größeren Anzahl an Dokumenten nicht praktikabel erscheint – vor allem nicht bei jedem COMMIT. Daher muss später beim Speichern von Einträgen und Dokumenten der jeweils zuständige Index manuell durch einen Aufruf von

CTX_DDL.SYNC_INDEX('[Indexname]', ...);

synchronisiert werden.
Als letztes benötigen wir noch Prozeduren, um Stoppworte zur Liste hinzuzufügen, zu löschen und nach der Bearbeitung die Indices neu auf zu bauen.

```
procedure zufuegenStoppworte(sListenname varchar2, sStoppwortliste
varchar2)
is
   aStoppwortliste APEX_APPLICATION_GLOBAL.VC_ARR2
   := APEX_UTIL.STRING_TO_TABLE(sStoppwortliste, ',');
begin
   for i in 1..aStoppwortliste.count loop
      CTX_DDL.ADD_STOPWORD(sListenname, aStoppwortliste(i));
   end loop;
   commit;
end;
```

Analog sieht die Prozedur *loeschenStoppworte* aus; anstelle von ADD_STOPWORD nutzt man REMOVE_STOPWORD aus dem selben CTX-Paket.

```
procedure NeuaufbauIndices(sListenname varchar2, sIndexliste varchar2)
is
   aIndexliste APEX_APPLICATION_GLOBAL.VC_ARR2
   := APEX_UTIL.STRING_TO_TABLE(sIndexliste, ',');
   sAlterStatement varchar2(255);
begin
   for i in 1..aIndexliste.count loop
      sAlterStatement
      := 'ALTER INDEX '|| aIndexliste(i) ||' REBUILD ONLINE
          PARAMETERS (''||'REPLACE STOPLIST '|| upper(sListenname) ||'')
          PARALLEL 3';
      execute immediate sAlterStatement;
   end loop;
end;
```

Ohne einen Neuaufbau des jeweiligen Index werden die Änderungen nur für nachträglich hinzugefügte Dokumente berücksichtigt.

Das in den Prozeduren APEX-Funktionen benutzt werden, ist unabhängig davon, womit die Benutzerschnittstelle programmiert wird – APEX muss lediglich in der Datenbank zur Verfügung stehen. Will man sich unabhängig von einer APEX-Oberfläche dessen Packages bedienen (z.B. im Batch), ist es notwendig eine Initialisierung vorzunehmen; dafür muss dann zumindest ein WORKSPACE angelegt worden sein.

```
procedure ini_APEX
is
   sWorkspace     APEX_APPLICATIONS.WORKSPACE%type := 'DUMMY';
begin
   /* Parameter für APEX Session setzen, wenn notwendig */
   if ( WWV_FLOW_API.GET_SECURITY_GROUP_ID <>
        APEX_UTIL.FIND_SECURITY_GROUP_ID(sWorkspace) ) then
      WWV_FLOW_API.SET_SECURITY_GROUP_ID
      (APEX_UTIL.FIND_SECURITY_GROUP_ID(sWorkspace));
   end if;
end;
```

Damit haben wir für die Schlagwortrecherche bereits alle Programmierarbeiten erledigt, wenn die Dokumente in einem BLOB gespeichert werden. Sollen diese als BFILE in der Datenbank nur referenziert werden, fehlt noch der Transfer aus einem temporären BLOB ins Dateisystem (vorausgesetzt die Oberfläche wird mit APEX erstellt), sonst ist der gesamte Prozess zum Hochladen einer Datei manuell zu programmieren.

```
create or replace function DateiTransfer
(sVerzeichnis varchar2, sDateiname varchar2, nVersion number, nFlowFilesID
number)
return bfile
is
   sDateiname_Vers varchar2(255)
   := replace(sDateiname, '.', '_'|| to_char(nVersion) ||'.');
   nDateiOffset    number := 1;
   nMaxInhaltsteil number := 32767;
   nSQLFehler      number;
```

```
   tDatei          UTL_FILE.file_type;
   bDateiinhalt    blob;
begin
   /* Dokumentdaten aus dem temporären APEX-Speicher abholen */
   select BLOB_CONTENT into bDateiinhalt
     from wwv_flow_files
     where ID    = nFlowFilesID;
   /* Dokument in VARCHAR2-Teilen im vorgegebenen Verzeichnis ablegen */
   while ( DBMS_LOB.SUBSTR(bDateiinhalt, nMaxInhaltsteil, nDateiOffset)
           is not null )
   loop
      if ( nDateiOffset = 1 ) then
         tDatei := UTL_FILE.fopen(sVerzeichnis, sDateiname_Vers, 'WB',
                                  nMaxInhaltsteil);
      end if;
      utl_file.put_raw(tDatei, DBMS_LOB.SUBSTR(bDateiinhalt,
                       nMaxInhaltsteil, nDateiOffset));
      utl_file.fflush(tDatei);
      nDateiOffset := nDateiOffset + nMaxInhaltsteil;
   end loop;
   if UTL_FILE.is_open (tDatei) then utl_file.fclose(tDatei); end if;

   delete from wwv_flow_files
    where ID    = nFlowFilesID;

   return BFILENAME (sVerzeichnis, sDateiname_Vers);
EXCEPTION
…
     return NULL;
end;
```

Leider steht UTL_FILE auch in Oracle 11g immer noch nur für das Speichern von VARCHAR2 zur Verfügung und nicht für LOB – also bleibt nichts weiter übrig, als das Dokument in maximale Teile von PL/SQL-VARCHAR2 zu zerlegen und in 'Häppchen' im vorher definierten Oracle Directory abzulegen.

Wenden wir uns jetzt der externen Recherche zu. Da wir hierfür in die weite Welt des Internet vorstoßen wollen, müssen wir ab Oracle 11g diesen Zugriff explizit erlauben:

```
BEGIN
/* ACL für Wissensdatenbank (WDB) */
 DBMS_NETWORK_ACL_ADMIN.CREATE_ACL (
  acl          => 'Netzzugriffe4WDB_ADM.xml',   -- Listenname
  description  => 'Netzzugriffe für WDB/ HTTP',
  principal    => 'WDB_ADM',                    -- zugelassen für Schema
  is_grant     => TRUE,
  privilege    => 'connect');

/* HTTP-Zugriff über alle Server über Port 80*/
  DBMS_NETWORK_ACL_ADMIN.ASSIGN_ACL('Netzzugriffe4WDB_ADM.xml',
                                    '*'  -- Server oder Domänenangabe/
                                         Adresse od. Name
                                    /* mögliche Angaben:
                                       - www.us.mycompany.com
                                       - 192.168.0.*
                                       - *.us.mycompany.com
```

```
                                                - *.com
                                                - *   */,
                                              80, 80);
  commit;
END;
```

Danach steht dem Einsatz von UTL_HTTP nichts mehr im Weg. Die folgende Funktion erledigt den HTTP-Dialog und liefert ein XML-Dokument als einfachen Text zurück.

```
function getXMLSpec(sSchlagworte varchar2, sZugangscode varchar2,
                    sProxyFull varchar2)
return CLOB
is
   /* externe Abfrage zusammenstellen */
   sRequest          varchar(255)
   := 'http://services.dnb.de/sru/accessToken~'|| sZugangscode ||
      '/dnb?version=1.1'||chr(38)||'operation=searchRetrieve'||
      chr(38)||'query='|| replace(sSchlagworte, ' ', '+');
   tXMLFilePiceTable UTL_HTTP.HTML_PIECES;
   cXMLOut           CLOB := sRequest;
begin
   /* Ergebnis der externen Abfrage in 2000-Byte Stücken abholen und
    * speichern
    */
   tXMLFilePiceTable := UTL_HTTP.REQUEST_PIECES(sRequest, 100, sProxyFull);
   if ( tXMLFilePiceTable.count > 0 ) then
      for i in 1..tXMLFilePiceTable.count loop
         cXMLOut := cXMLOut || tXMLFilePiceTable(i);
      end loop;
   end if;

   return replace(replace(replace(cXMLOut, sRequest, ''), '<!DOCTYPE',
                          '<!--DOCTYPE'), '.dtd">', '.dtd" -->');

end;
```

Enthält das XML-Dokument eine DOCTYPE-Anweisung muss diese auskommentiert oder entfernt werden. Da Oracle XML kein DTD unterstützt, ist die lokale Speicherung der DTD-Datei nicht möglich und aus Sicherheitsgründen wegen des direkten Internetzugriffs lässt sich das nicht manipulierte Dokument nicht als XMLTYPE ablegen. Wird die Dokumententransformation durch ein XMLSchema (xsl(t)-Datei) beschrieben, können Sie diese lokal in die Datenbank laden und für die Datenaufbereitung nutzen.

Aus dem XML-Dokument lassen sich dann gewünschte eindeutige Einträge z.B. mit der folgenden Funktion herauslesen – zumindest wenn der Oracle XML-Parser den gewünschten Knoten als Elementknoten identifiziert hat und nicht als Text- oder sonstigen unbenannten Knoten:

```
function getXMLKnotenWert(nAnwenderID number, dDatum date,
                          sKnotenName varchar2, cXMLDoc clob)
return varchar2
is
  tXMLNode xmltype;
  sValue   varchar2(4000) := '';
begin

   if ( cXMLDoc is null ) then
```

```
…
   else
      tXMLNode := xmltype.createXML(cXMLDoc);
   end if;
   /* wenn ein XPath-Ausdruck in sKnotenname übergeben wurde
    * und der entsprechende Knoten existiert, dann …
    */
   if ( sKnotenName is not null and
        tXMLNode.existsNode('//'|| sKnotenName)>0 ) then
      /* Knoten extrahieren und den Wert als Text speichern */
      tXMLNode := tXMLNode.extract('//'|| sKnotenName);
      sValue   := tXMLNode.getStringVal();
   end if;

   return sValue;
end;
```

Die folgende Funktion liefert eine Liste von Textknoteninhalten, wenn *sKnotenName* einen XPath-Ausdruck enthält, der auf Elementknoten mit reinem Textinhalt verweist (z.B. *<dc:date>2013</dc:date>*), sonst jeweils den Wert des ersten Kind-Knoten der angegebenen Elemente. Bei Listen aus XML-Dokumenten – ob Knotennamen oder Werte - sollte allerdings nicht das (APEX-Standard-)Trennzeichen Doppelpunkt verwendet werden, da dieses Zeichen auch innerhalb der Listeneinträge häufig vorkommt.

```
function getXMLKnWerteliste(nAnwenderID number, dDatum date,
                            sKnotenName varchar2, cXMLDoc clob)
return varchar2
is
  tXML       xmltype;
  tXMLNode   DBMS_XMLDOM.DOMNODE;
  tXMLListe  DBMS_XMLDOM.DOMNODELIST;
  nRecordAnz number;
  sRecords   varchar2(4000) := '';
begin

   if ( cXMLDoc is null ) then
…
   else
      tXML := xmltype.createXML(cXMLDoc);
   end if;

   if ( sKnotenName is not null ) then
      tXML       := tXML.extract(sKnotenName);
      tXMLNode   := DBMS_XMLDOM.MAKENODE(DBMS_XMLDOM.NEWDOMDOCUMENT(tXML));
      tXMLListe  := DBMS_XMLDOM.GETCHILDNODES( tXMLNode );
      nRecordAnz := DBMS_XMLDOM.GETLENGTH( tXMLListe );

      for i in 1..nRecordAnz loop
         tXMLNode := DBMS_XMLDOM.ITEM(tXMLListe, i-1);
         sRecords := sRecords ||'#'|| DBMS_XMLDOM.GETNODEVALUE(
                                      DBMS_XMLDOM.GETFIRSTCHILD(tXMLNode));
      end loop;
   end if;

   return rtrim(ltrim(sRecords, '#'), '#');
end;
```

Um weitere Möglichkeiten aus XMLDB (XMLTYPE und DBMS_XMLDOM-Package) abhängig von der erhaltenen Service-Antwort und den Anwenderwünschen zu nutzen, sollte man sich vorher in XML, DOM und XPath einlesen.

Zuletzt kümmern wir uns um die Zugriffsbeschränkung. Mit Hilfe von VPD benötigen wir dafür minimal ein bis zwei Funktionen und ein Skript zum Definieren der Richtlinien. Die Anzahl der Funktionen richtet sich danach, ob Ansehen (SELECT) und Änderungen (DML) dieselben oder unterschiedliche Bedingungen erhalten sollen. Wenn nicht alle zu schützenden Tabellen über dieselbe Bedingung eingeschränkt werden können, erhöht sich die Anzahl zu erstellender Funktionen. Allerdings sind die Möglichkeiten Zugriffsbedingungen zu definieren über diesen Weg sehr viel umfangreicher als beim Einsatz von Views, da die Bedingungen zur Laufzeit als reiner Text mit Hilfe von PL/SQL generiert werden.
Des weiteren lässt sich eine Zugriffskontrolle über VPD sehr viel einfacher nachträglich ergänzen, da keine oder kaum Änderungen in der graphischen Oberfläche vorgenommen werden müssen; höchstens fehlende Spalten für die beschränkenden Bedingungen sind eventuell hinzu zu fügen. Außerdem stellt sich die Vergabe von Zugriffsrechten an andere Schemata – z. B. für Schnittstellen – unproblematischer dar. GRANTs können aus datenschutzrechtlicher Sicht ohne Probleme auf Tabellenebene vergeben werden.

Funktionen für den Einsatz mit VPD, die in Richtlinien eingetragen werden, müssen genau zwei Übergabeparameter haben: die erste erhält beim Aufruf das Schema, die zweite den Namen des Objektes (Tabelle oder View). Der Rückgabewert der Funktion ist eine WHERE-Bedingung als Text wie im folgenden Beispiel:

```
function WDBInfo_Policy(obj_schema varchar2, obj_name varchar2)
return varchar2
is
   sPolicyClause varchar2(2000) := '';
   sAPPUser      APEX_WORKSPACE_APEX_USERS.USER_NAME%type
   := nvl(v('APP_USER'), user);
   sColName_Pref user_tab_columns.column_name%type;
   nAPEXUserId   number := APEX_UTIL.GET_USER_ID(sAPPUser);
begin
   if ( upper(sAPPUser) not in (upper(obj_schema), 'SYS', 'ADMIN') ) then

      select substr(column_name, 1, 3) into sColName_Pref
        from all_tab_columns
       where owner      = upper(obj_schema)
         and table_name = upper(obj_name)
         and rownum     = 1;
      sPolicyClause
      := '('|| sColName_Pref ||'_AUTOR = '''|| sAPPUser ||''' or '||
         sColName_Pref ||'_ZUGRIFF_BER is null or '||
         sColName_Pref ||'_ZUGRIFF_BER in '||
         '(select ZAW_ZUG_KENN from WDB_ZUGRIFF_ANWEND '||
            '  where ZAW_APEX_UID = '|| nAPEXUserId ||
            '    and ( ZAW_ZUGRIFF_ERTEILER_APEX_UID
                  = APEX_UTIL.GET_USER_ID('||
                              sColName_Pref ||'_AUTOR)'||
            '          or ZAW_ZUGRIFF_ERTEILER_APEX_UID = -1 ))' ||
            ')';
   end if;

   return sPolicyClause;
```

```
EXCEPTION WHEN OTHERS THEN
   if ( sqlcode >0 ) then return sPolicyClause;
   else              raise;
   end if;
end;
```

Die so erzeugte zusätzliche Bedingung wird zur Laufzeit an jedes Statement (SELECT, DML) für die Objekte angehängt, für die wir eine entsprechende Richtlinie erzeugen:

```
begin
   dbms_rls.add_policy( object_schema => '&&WDB_Schema',
                        object_name => 'WDB_DOKUMENTE',
                        policy_name => 'WDB_InfoPolicy',
                        function_schema => '&&WDB_Schema',
                        policy_function =>
                               'WDB_Administration_pkg.WDBInfo_Policy',
                        statement_types => 'select',
                        update_check => FALSE,
                        enable => TRUE );
   dbms_rls.add_policy( object_schema => '&&WDB_Schema',
                        object_name => 'WDB_EINTRAEGE',
                        policy_name => 'WDB_InfoPolicy',
                        function_schema => '&&WDB_Schema',
                        policy_function =>
                               'WDB_Administration_pkg.WDBInfo_Policy',
                        statement_types => 'select',
                        update_check => FALSE,
                        enable => TRUE );
   commit;
end;
```

Leider stellte sich heraus, dass mit dem Anlegen der Richtlinien für die Zugriffsbeschränkung die Neuerstellung der Textindices als APEX-Anwender nicht mehr funktionierte bzw. nur noch mit abgeschalteten Richtlinien während der Erstellung. Aber letzteres widerspricht einem uneingeschränkten Mehrbenutzersystem.
Außerdem gibt es für die Nutzung von VPD in einer '*Wolke*' ein erhebliches Hindernis: Um Richtlinien einzurichten benötigt der Datenbanknutzer (DBSchema) SYSDBA-Rechte – und diese erhält man in gemanagten Umgebungen i.a. nicht. Daher lässt sich ein solches Vorhaben nur mit höchstens IaaS (Infrastruktur als Service) umsetzen.

Damit sind die Programmierarbeiten für die Wissensdatenbank – zumindest für einen Prototypen – abgeschlossen.

Die Anwenderschnittstelle – APEX
Auf APEX selber soll in diesem Vortrag nicht detailliert eingegangen werden, da dieses Thema andere Beiträge umfangreich behandeln. Einiges über die in der Anwendung zur Verfügung stehenden Seiten und deren Funktionen erfahren Sie in einer kurzen Demo. Dieser Beitrag geht im übrigen nur auf die Spezifika für Oracle Text, HTTP-Requests und XMLDB ein; die Besonderheiten beschränken sich auf manuell erstellte Prozesse, um die vorher programmierten Funktionen und Prozeduren aufzurufen, der Nutzung der View WDB_STOPPWORTLISTE_V basierend auf CTX_USER_STOPWORDS und dem SELECT-Statement für die Schlagwortrecherche.

Zum administrativen Bereich der Anwendung gehört eine Seite zum Pflegen der Stoppworte. Diese gestaltet sich recht einfach: spezifisch für die Einbindung von Oracle Text ist eine Region

Stoppwortliste (s. Abb. 5 und 6), drei Prozesse (s. Abb. 5) und ein Radio Group Item P42_ALPHABET (s. Abb. 7). Die Prozesse rufen die oben bereits vorgestellten Funktionen '*zufuegenStoppworte*', '*loeschenStoppworte*' und '*NeuaufbauIndices*' auf.

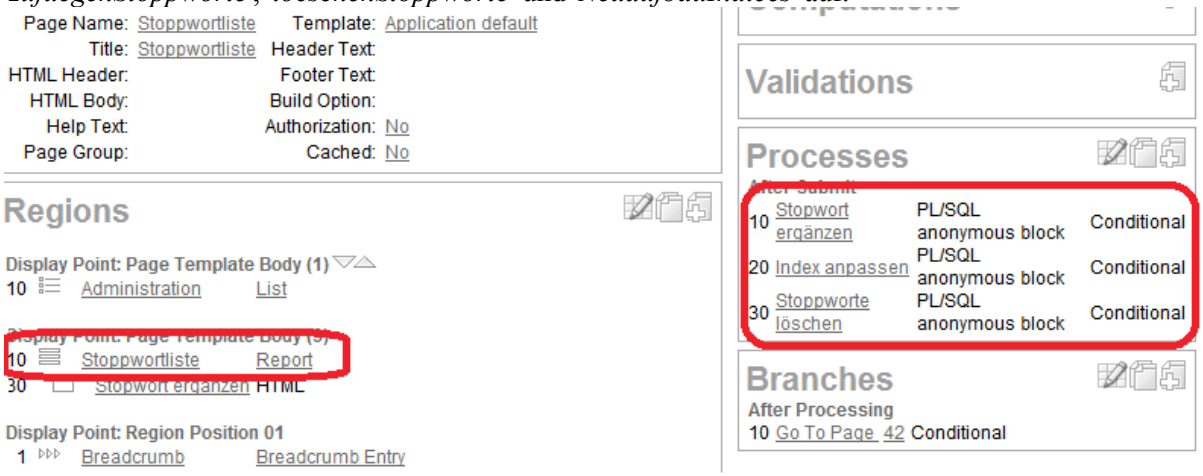

Abb. 5: Seite Stoppwortliste

Die Report-Region basiert auf der View WDB_STOPPWORTLISTE_V und bietet dem Anwender die Möglichkeit nicht mehr benötigte Einträge zu markieren, um sie zu löschen.

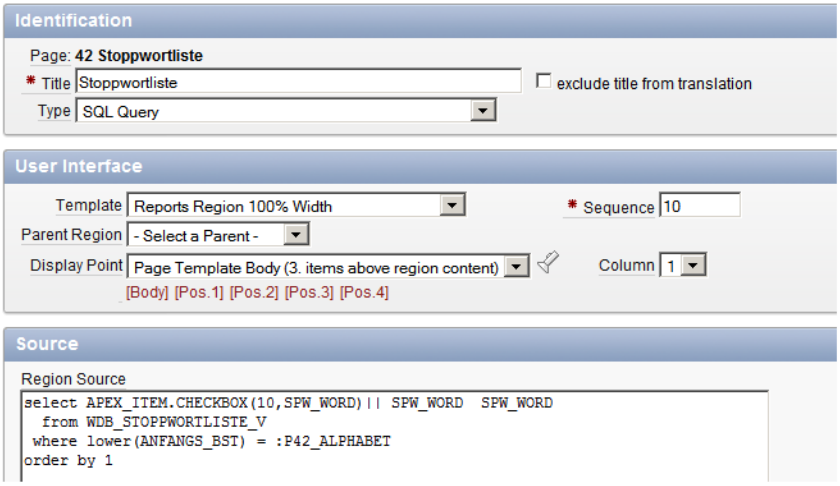

Abb. 6: Region Stoppwortliste

Das Radio Group Item basiert auf derselben View und bietet dem Anwender jeweils eine Sicht gemäß Anfangsbuchstaben auf die Stoppworte. Bei solchen WEB-Seiten werden i.a. Reiter verwendet, aber ein TABLIST Item gibt es im APEX Standard nicht und hier habe ich auch darauf verzichtet nach zu sehen, ob jemand anderer bereits ein entsprechendes PLUG-IN geschrieben hat.

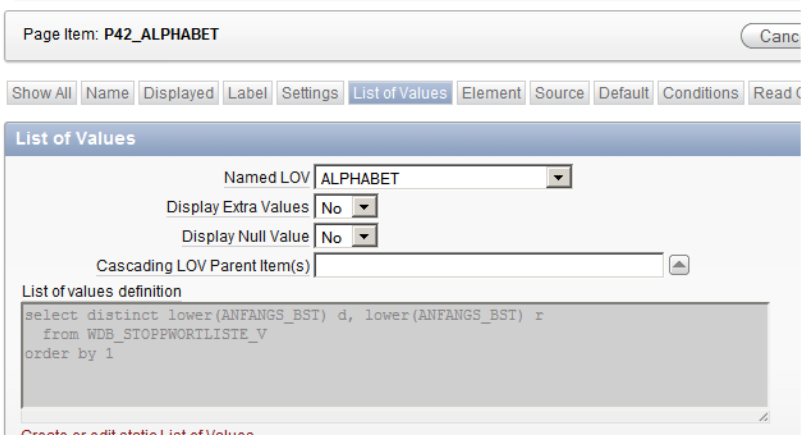

Abb. 7: Radio Group Item zur Auswahl des Anfangsbuchstaben

Um die hoch geladenen Dokumente als BFILE zu speichern, benötigt man in APEX einen zusätzlichen Prozess (*DOK_DATEI ergänzen* s. Abb. 8); der APEX Standardprozess *Process Row of WDB_DOKUMENTE* legt die Datei in einem von APEX zur Verfügung gestellten BLOB in der Datenbank ab. Da in der Tabelle für die Dokumente auch eine Spalte für Bemerkungen existiert, die ebenfalls in den Schlagwortkatalog aufgenommen wurde – vor allem für Bilddateien, die nicht zu indizieren sind – gibt es einen weiteren spezifischen Prozess zum Synchronisieren des entsprechenden Indexes (s. Abb. 8). Dieser ruft die mit Oracle Text bereitgestellte Standardprozedur CTX_DDL.SYNC_INDEX auf.

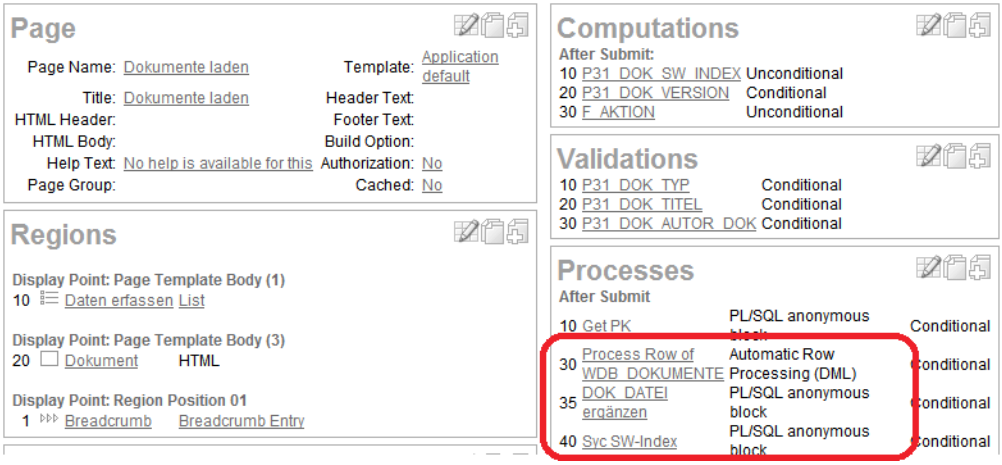

Abb. 8: APEX Seite Dokumente laden

Kommen wir jetzt zur Recherche, dem wesentlichen Ziel einer Wissensdatenbank. Die APEX-Seite hierfür enthält zwei Spezifika: die Report Region für das Ergebnis der Schlagwortsuche (s. Abb. 10) und eine Validierung für Schlagworte (s. Abb. 11). Letztere ist durch eine Kundenpräsentation der Anwendung eingeflossen, da der Anwender eine entsprechende Meldung erwartete, wenn kein Ergebnis erscheint, wenn die Schlagwortvorgabe nur aus Stoppworten besteht.

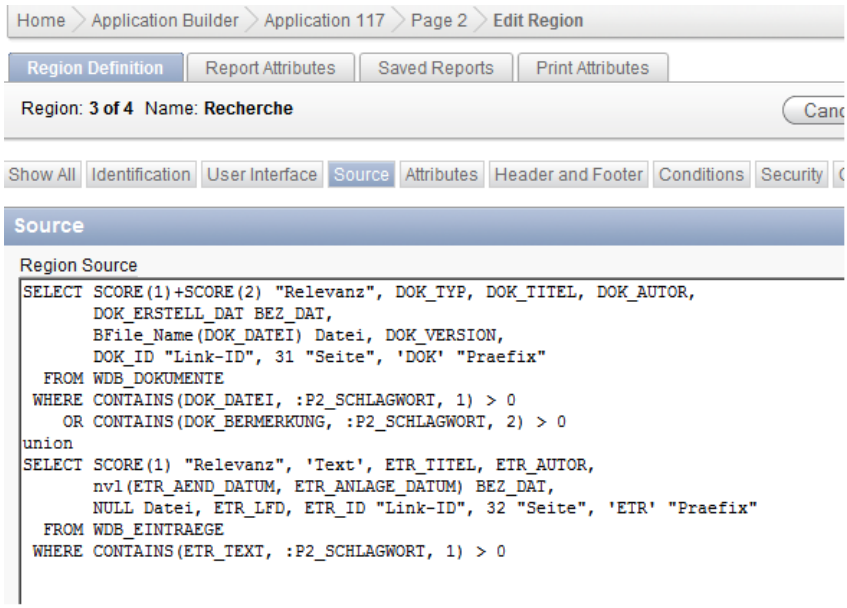

Abb. 10: SELECT-Statement für die Schlagwortrecherche

Sicherlich wäre es sogar besser gewesen, die Validierung als Funktion auszulagern. Ich habe darauf verzichtet, da diese Funktionalität keine Datenverarbeitung beinhaltet, sondern der Anwenderinformation dient.

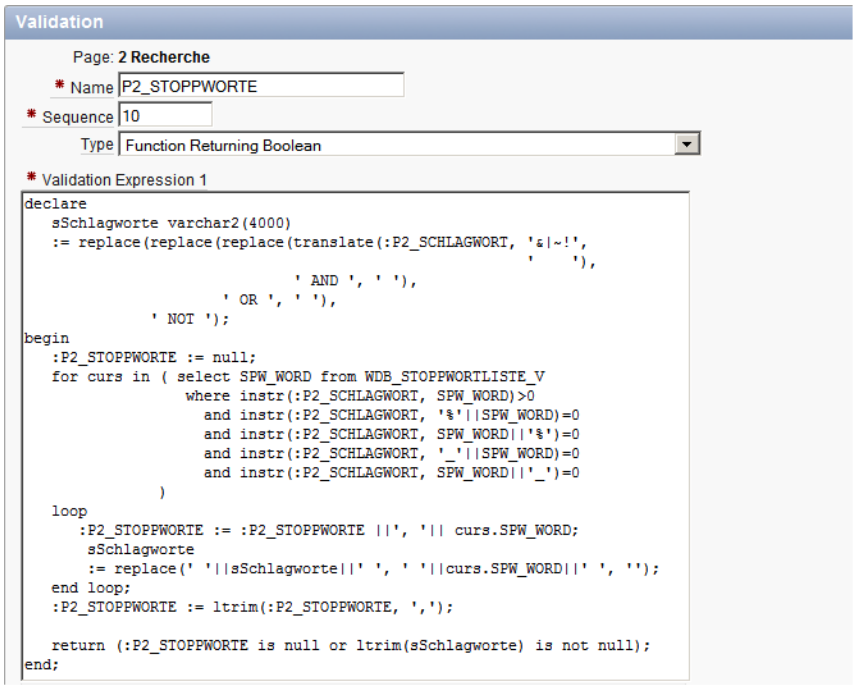

Abb. 11: Validierung für Schlagworte auf Stoppworte

Die restlichen Seiten der Anwendung weisen keine weiteren Spezifika auf.

Aus Erfahrung lernt man am besten

Und aus den Erfahrungen und der Arbeit anderer vielleicht sogar effizienter. Wie sagte mein Mathematiklehrer in der Oberstufe: „Mathematiker sind faul – intelligent faul. Bevor sie mit der

Lösung einer Aufgabe beginnen, verwenden sie zunächst lieber einige Zeit sich damit zu beschäftigen, was andere bereits zur (Teil-)Lösung erarbeitet haben. Kein Mathematiker erfindet das Rad ein zweites Mal.“
Auch bei der Erstellung dieses Prototypen hat sich für mich bewahrheitet, dass es nützlich ist, sich zunächst mit bereits Existierendem zu beschäftigen. Eine Oracle Datenbank stellt einen wahren Baukasten mit Standardsoftware für Individualanwendungen dar.
Aber wo Licht ist, gibt es auch Schatten: Sicherlich wird sich herausstellen, dass im produktiven Betrieb der Anwendung bei steigendem Datenvolumen die Schlagwortindizierung zu optimieren ist; aber hierfür bietet Oracle Text auch noch Möglichkeiten. Außerdem bleibt abzuwarten, wie performant das System bei steigenden Benutzerzahlen ist. Anpassungen im Bereich der Datenverteilung auf verschiedene Tablespaces oder auch Partitionierung bieten hierbei Optimierungsmöglichkeiten.
Allerdings kommen wir hiermit zu einem – bereits oft angesprochenen – Kritikpunkt der die Freude trübt: der Lizenzierung. Zwar stehen die hier verwendeten Bausteine bis auf VPD sogar in einer 11g XE (zum Teil mit Einschränkungen) zur Verfügung, aber dass VPD auch für eine SE nicht einmal kostenpflichtig hinzu lizenziert werden kann, verstehe ich nicht. Auf Anwendungsebene lässt sich der Verzicht auf VPD mit einigem zusätzlichen Aufwand über Views zwar umgehen, aber auf Datenbankebene haben DB-Schemata, denen Rechte auf eine Tabelle erteilt wurden ungehinderten Zugriff auf alle Daten. Außerdem steht Partitionierung auch nur bei einer EE zusätzlich kostenpflichtig zur Verfügung.
Kommen wir jetzt aber zu den wesentlichen Hürden und Grenzen des 'Baukastens'. Auch wenn sich der Aufbau der Anwendung kurz und wenig aufwändig darstellt, gibt es Schwierigkeiten und gewisse Einschränkungen. Ich habe diese im vorausgehenden Text bereits kurz angesprochen: die fehlenden Stoppworte, die Kollision von Text und VPD beim Indexneuaufbau sowie die Nutzung von XPath in der von Oracle unterstützten Version bei Textknoten bzw. was der Oracle XML-Parser für Textknoten hält. Hierauf werde ich im Vortrag detaillierter eingehen sowie zur Lösung oder Strategien zur Umgehung.

Kontaktadresse:
Gerhild Aselmeyer
G. Aselmeyer, Dipl. Math. - Analysen, Konzepte & Anwendungsentwicklung für EDV
Rellinghauser Straße 344
D-45136 Essen

Telefon: +49 (0) 201 - 254271
Fax: +49 (0) 201 - 72239974
E-Mail g.aselmeyer@aka-edv.de
Internet: www.aka-edv.de