One Filter to Rule them All: Dynamische Regeln für das Business

Robert Marz
ist-people

Schlüsselworte
Rulesmanager, Expression Filter, Metadatengetriebene Regelwerke, PL/SQL, ECA, Active Database.

Einleitung

Rules Manager und Expression Filter sind zwei sehr mächtige aber weitgehend unbekannte Bestandteile der Datenbank, die mit Oracle 10g eingeführt wurden.
Metadatengesteuert – und damit ohne Programmierung – lassen sich komplexe Regelwerke und Prüfungen erstellen.
Rulesmanager setzt auf Expression Filter auf. Durch den Einsatz dessen eigener Indextypen und SQL-Operatoren lassen sich auch sehr große Datenbestände performant bearbeiten.

Moderne Anforderungen an die Datenbewertung

Datenqualität bestimmen ist einfach: Einfach ein paar Regeln in Code gießen, die checken, ob alle wichtigen Felder mit Werten belegt sind und schon kann man seine Datenbestände sinnvoll bewerten…

So einfach war es noch nie. Will man sinnvolle Aussagen treffen, müssen Sätze oftmals auch im Zusammenhang mit Daten aus anderen Quellen bewertet werden.

Hinzu kommt, dass sich die Geschäftsregeln rund um Datenqualität und Bewertung häufig ändern, was normalerweise eine langwierige Anpassung der Programme durch die IT-Abteilung nach sich zieht. Bis die Änderungen implementiert, getestet und abgenommen sind, haben sich die Anforderungen oftmals schon wieder verändert.

Diesen Teufelskreis kann man mit dem Einsatz von Rulesmanager und Expression Filter durchbrechen. Rulesmanager setzt dabei auf Expression Filter auf, das – ähnliche wie zum Beispiel Oracle Text – eigene Schlüsselwörter für die SQL-Sprache definiert und Indextypen auf die Definierten Ausdrücke und Regeln anlegt.

Event Condition Action (ECA)

Rulesmanager folgt dem Prinzip Event Condition Action (ECA), das die Struktur von aktiven Regeln in Ereignisgetriebenen Architekturen und "Active Database Systems" beschreibt.

Solche Regeln bestehen aus drei Teilen:

- Das Ereignis (Event) definiert den Auslöser für das Ausführen der Regel
- Die Bedingung (Condition) ist die logische Prüfung, die, wenn erfolgreich den Aktions-Teil ausführt
- Die Aktion führt schließlich die Änderungen an den lokalen Daten durch.

Im Rulesmanager werden Ereignisse, Bedingungen und Aktionen in Zeilen und Spalten von normalen Tabellen erfasst und lassen sich so als Metadaten auch durch Endanwender edieren.

Einfache Ereignisse

Konstruieren wir uns ein einfaches Beispiel, losgelöst von Datenqualität:
Wir haben eine Tabelle Kunden, die mit den Daten neuer Kunden gefüllt wird:

```
create table kunden
  ( Kundenname    varchar2(50)
  , Firma         varchar2(50)
  , Adresse       varchar2(50)
  --
  , BetreuerAbt   varchar2(50)
  , Betreuer      number
)
/
```

Die Kunden sollen direkt beim ersten Anlegen eine betreuenden Abteilung und der Nummer eines Sachbearbeiters versehen werden. Ist das Feld Firma leer, sind es Privatkunden, sonst Firmenkunden. Die Betreuernummer ist bei Firmenkunden 1000, bei Privatkunden 5000.

Auf herkömmlichen Weg würde man die Tabelle über eine Prozedur wie diese hier füllen:

```
create or replace procedure neuer_kunde
  ( p_kundenname    kunden.kundenname%type
  , p_firma         kunden.firma%type
  , p_adresse       kunden.adresse%type
)
as
  l_BetreuerAbt   kunden.betreuerabt%type;
  l_Betreuer      kunden.betreuer%type;
begin
  /* Geschäftsregel start */
  if p_firma is not null
    then
         l_betreuer := 1000;
         l_betreuerabt := 'Firmen';
    else
         l_betreuer := 5000;
         l_betreuerabt := 'Privat';
   end if;
  /* Geschäftsregel ende */
  insert into kunden (kundenname, firma, adresse, betreuer, betreuerabt)
  values (p_kundenname, p_firma, p_adresse, l_betreuer, l_betreuerabt);
end;
/
```

Ein Aufruf wie dieser liefert dann die erwarten Ergebnisse:

```
begin
  neuer_kunde('Hans Mustermann', NULL, NULL);
  neuer_kunde('Elke Sommer', 'Bundesdruckerrei', null);
end;
/

select kundenname, firma, betreuerabt, betreuer from kunden;
```

```
KUNDENNAME         FIRMA                BETREUERABT     BETREUER
------------------ -------------------- --------------- ----------
Hans Mustermann                         Privat          5000
Elke Sommer        Bundesdruckerrei     Firmen          1000
```

Wenn sich die Anforderungen jetzt ändern kann es aber sehr schnell komplex werden, etwa wenn die Betreuer nach Buchstabenbereichen oder nach der Anzahl der Bereits betreuten Kunden ausgewählt werden sollen. In jedem Fall ist aber Programmierarbeit erforderlich.

Für den Einsatz des Rulesmanager sind zunächst einige Vorarbeiten nötig:

```
-- Das Objekt für das Ereignis anlegen
create or replace type NeuKunde
  AS OBJECT
  ( kundenname    varchar2(50)
  , firma         varchar2(50)
  , adresse       varchar2(50)
  )
/

-- Anlegen der Regelklasse
begin
  dbms_rlmgr.create_rule_class
    ( rule_class =>'KundenZuordnung'  -- Regelname
    , event_struct =>'NeuKunde'       -- Unser Objekt
    , action_cbk =>'KundenZO'         -- Callback Prozedur
    , rlcls_prop => '<simple consumption="exclusive"
                            ordering="rlm$rule.rlm$ruleid"/>'
    , actprf_spec => 'betreuerabt  varchar2(50), Betreuer  number'
    );
end;
/

-- Callback Prozedur
create or replace procedure KundenZO
  ( rlm$event   NeuKunde
  , rlm$rule    KundenZuordnung%ROWTYPE
  )
is
begin
  insert into kunden (kundenname, firma, adresse, betreuer, betreuerabt)
  values ( rlm$event.kundenname, rlm$event.firma, rlm$event.adresse
         , rlm$rule.betreuer, rlm$rule.betreuerabt);
end;
/
```

Die Hauptprozedur von oben ist dann etwas einfacher:

```
create or replace procedure neuer_kunde
  ( p_kundenname    kunden.kundenname%type
  , p_firma         kunden.firma%type
  , p_adresse       kunden.adresse%type
)
as
begin
  dbms_rlmgr.process_rules
    ( rule_class => 'KundenZuordnung'
    , event_inst => NeuKunde.getvarchar (p_kundenname, p_firma, p_adresse)
    );
end;
/
```

Nach dem Anlagen der Regelklasse finden wir in unserem Schema eine neue Tabelle namens Kundenzuourdnung:

```
desc KundenZuordnung -- Neu angelegte Tabelle für die Regeln

Name          Null     Typ
------------ -------- --------------
RLM$RULEID   NOT NULL VARCHAR2(100)
BETREUERABT           VARCHAR2(50)
BETREUER              NUMBER
RLM$RULECOND          VARCHAR2(4000)
RLM$RULEDESC          VARCHAR2(1000)
RLM$ENABLED           CHAR(1)
```

Die Tabelle ist zunächst leer. Was auffällt, sind die beiden Spalten BetreuerAbt und Betreuer, die letztlich unser Regelergebnis widerspiegeln, wenn die Regelbedingung erfüllt ist (Spalte RLM$RULECOND).

Legen wir also die Regeln für unseren ersten einfachen Fall an:

```
insert into KundenZuordnung
   (rlm$ruleid, betreuerabt, betreuer, rlm$rulecond)
  values ('F1-01', 'Privat', 5000,'firma is null');

insert into KundenZuordnung
   (rlm$ruleid, betreuerabt, betreuer, rlm$rulecond)
  values ('F1-999999', 'Firmen', 1000,'1=1');

commit;

begin
  neuer_kunde('Hans Mustermann', NULL, NULL);
  neuer_kunde('Elke Sommer', 'Bundesdruckerrei', null);
end;
/
```

Das Ergebnis ist wie erwartet identisch zu dem der PL/SQL Prozedur von oben.

Da wir die Regelklasse mit den Properties '<simple consumption="exclusive" ordering="rlm$rule.rlm$ruleid"/>' angelegt haben, werden die Regeln in der Reihenfolge der rlm$ruleid abgearbeitet und die Verarbeitung stoppt, wenn es den ersten Treffer gab. Die Zweite Regel ist mit der Bedingung ‚1=1' eine Catchall-Regel, die immer trifft. Ohne eine solche würde die Callback-Prozedur nicht aufgerufen, wenn keine Bedingung zutrifft und es würde dann kein Satz eingefügt.

Ändern der Regeln

Wenn sich jetzt unserer Geschäftsregeln ändern, brauchen wir nur die Metadaten anzupassen.

Die Privatkundenzahl ist stark gewachsen und sollen nach Namen sortiert werden – alle Kunden, deren Namen mit A-M beginnt, bekommen den Betreuer 5100, der Rest die 5200. Außerdem sollen bei den Firmenkunden der Öffentliche Dienst besonders behandelt werden:

```
-- Alte Regeln ausser Kraft setzten
update KundenZuordnung set rlm$enabled='N';

insert into KundenZuordnung (rlm$ruleid, betreuerabt, betreuer,
rlm$rulecond)
  values ('F2-01', 'Privat A-M', 5100,'firma is null and
substr(upper(kundenname),1,1) <''N'' ');

insert into KundenZuordnung (rlm$ruleid, betreuerabt, betreuer,
rlm$rulecond)
  values ('F2-02', 'Privat N-Z', 5200,'firma is null and
substr(upper(kundenname),1,1) >=''N'' ');

insert into KundenZuordnung (rlm$ruleid, betreuerabt, betreuer,
rlm$rulecond)
  values ('F2-03', 'Öffentlicher Dienst', 1900,'firma like ''Bundes%'' ');

insert into KundenZuordnung (rlm$ruleid, betreuerabt, betreuer,
rlm$rulecond)
  values ('F2-999999', 'Firmen', 1000,'1=1');

commit;

begin
  neuer_kunde('Mustermann. Frauke', NULL, NULL);
  neuer_kunde('Zacharias, Erich', NULL, NULL);
  neuer_kunde('Winter, Elfriede ', 'Bundesamt für Sommer', null);
  neuer_kunde('Bonham Carter, Helena', 'Sweeney Todd', null);
end;
/

KUNDENNAME              BETREUERABT          BETREUER
----------------------- -------------------- ----------
Hans Mustermann         Privat               5000
Elke Sommer             Firmen               1000
Mustermann. Frauke      Privat A-M           5100
Zacharias, Erich        Privat N-Z           5200
```

```
Winter, Elfriede        Öffentlicher Dienst  1900
Bonham Carter, Helena   Firmen               1000
```

Die Regeln wurden nur durch das Anpassen der Metadaten geändert. Dies könnte zum Beispiel auch über eine Oberfläche geschehen…

Wie in den Regeln F2-01 und F2-02 zu sehen ist, können die Bedingungen auch Funktionsaufrufe enthalten. Das können natürlich auch selbstgeschriebene Funktionen sein.

Regelklassen können so angelegt werden, dass sie jeweils nur genau eine Regel ausführen – wie in unserem Fall – oder für jede Regel ausgeführt werden, Regeln können aus den Tabellen gelöscht, oder einfach deaktiviert werden. Die Möglichkeiten sind beinahe Endlos.

Komplexe Ereignisse

In den vorangegangen Beispielen haben einfach Regeln, basierend auf einer Tabelle abgebildet.

Regeln können aber auch ungleich komplexer werden, indem sie zum Beispiel Bedingungen haben, die mit Joins mehrere Tabellen verbinden.

Außerdem ist es möglich, komplexe Events aus mehreren einfachen zusammenzusetzten. Diese komplexen Events werden dann nur ausgelöst, wenn alle Vorereignisse ausgelöst wurden. Solange werden sie in Tabellen zwischengespeichert.

Da die Bedingungen beliebig SQL-Abfragen sind, lassen sich auf diese Weise Ereignisketten definieren, die z.B. auch zeitlich voneinander abhängen können.

Demos dazu finden sich in den Skripts zum Vortrag.

Auslösen von Ereignissen

In den Beispielen bisher haben wir die Events immer explizit über den Aufruf von dbms_rlmgr.process_rules ausgelöst.

Regeln lassen sich aber auch DML-Ereignisse wie Insert,Update und Delete knüpfen.

Wenn man mit einem Update Millionen Zeilen bearbeitet, dauert es natürlich seine Zeit, bis die Regeln für jede Zeile abgearbeitet wurden.

Regelklassen können aber so definiert werden, dass sie das „Continous Query Notification“-Feature der Datenbank nutzen und so nach einer Transaktion für alle Sätze in der Tabelle auf einmal gefeuert werden.

Szenarios aus der Datenqualität:

Rulesmanager lässt sich ganz hervorragend im Bereich Datenqualität einsetzen.

Einfache Vollständigkeitsprüfungen kann sich sicherlich jeder vorstellen.

Diese können durch einfache Updates falls nötig kurzfristig deaktiviert werden.

Mit komplexen Events lassen sich Informationen aus mehreren Quellen zusammenfassen, und so ein Qualitätsscoring erzeugen.

Mit geschicktem Einsatz der Tuning-Möglichkeiten von Expression Filter lassen sich auch große Datenmengen performant bearbeiten und Auswerten.

Ausblick / Update

Nachdem dieser Vortrag eingereicht und angenommen wurde, hat Oracle in der Support Note [ID 1244535.1] bekannt gegeben, dass Rulesmanager und Expression Filter veraltet sind und im nächsten Hauptrelease der Datenbank nicht enthalten seien werden.

Nutzern von Expression Filter wird der Umstieg auf Continuous Query Notification empfohlen.

Als Ersatz für den Rulesmanager wird dem Kunden „Oracle Business Rules“ ans Herz gelegt, welches allerdings kein Bestandteil der Datenbank sondern eines von Fusion Middleware ist.
Gemeinsam ist beiden, dass sie ECA-Regeln verwenden.
Während Rulesmanager Datenbankzentriert ist, betreibt Business Rules einen Workflow-zentrischen Ansatz. Anstelle der PL/SQL Events des Rulesmanager stellt Business Rules Facts fest, die dann als Java-Klasse implementiert werden.

Einen einfachen Migrationspfad wird es nicht geben.

Kontaktadresse:

Robert Marz

its-people Hochtaunus GmbH
Lyoner Straße 44-48

60528 Frankfurt am Main

Telefon: +49 (69) 247521-00
Fax: +49 (69) 247521-021
E-Mail: robert.marz@its-people.de
Internet: www.its-people.de