THE INFORMATION COMPANY

Matthias Müller
Principal Sales Consultant
STCC Düsseldorf

Oracle Deutschland GmbH

ORACLE

Matthias Müller
matthias.mueller@oracle.com
☎ 0211 / 74839 - 293

Oracle Deutschland GmbH

ORACLE

Regeln ohne Ausnahme

Rules Manager in ORACLE DATABASE 10g R2

DECUS Symposium 2006

Vortrag 2K03 / 17.05.2006 / 11:30 - 12:15 Uhr / Raum Konstantin

ORACLE

Was ist der Rules Manager?

- Neues Feature der Oracle Database 10*g* Release 2
  - Rules Manager verfügbar in allen Datenbank-Editionen
  - Basiert auf **`EXPRESSION FILTER`** der Oracle Database 10*g*R1
- Event-Condition-Action (ECA) Regeln und Event Policies
  - Management durch SQL – Syntax mit XML-Tag-Erweiterungen
  - ECA-Regeln deklarativ – Multi-Terabyte Datenmengen möglich
  - Deklarative Beziehungen – AND, ANY, NOT, Sets, Order, Time
  - Viele Quellen – Anwendungen, Tabellen, DML
  - Verschiedene Datentypen – Skalar, XML, Spatial
  - Große Anzahl von Regeln – quasi unbeschränkt
  - Regeln mit einfacher und komplexer Semantik
  - Auslösung von Aktionen in Datenbank und Anwendungen

ORACLE

Enge Datenbank-Integration

- Skalierbar / performant / sicher / gut administrierbar
- Objekt-Relationale Ereignis-Datenstrukturen
  - Verschiedene Ereignistypen – Dynamisch, Batch, DML
  - Regelauswertung auf Multi Terabyte-Sets in DB
    - Spart Netzwerk-Verkehr und -Ressourcen
    - Reduziert Application Server-Speicherbedarf
- Komplexe Regeln aus einfachen Regeln komponiert
  - Indexierung von Regeln
  - Parallele Ausführung von Regeln
  - Sessions für inkrementelle Evaluierung
  - Datenbank Scheduling
  - Security-Mechanismen

Behandlung von Ereignissen

- Erzeugung von Ereignissen durch Anwendungen
  - PL/SQL
  - APIs
  - SQL
  - JDBC
  - ODBC
  - ADO.NET
  - SQL*Net
- Verarbeitung von Ereignissen durch den Rules Manager
  - ECA-Regeln mit Indexierung
  - Persistenter Ereignis-Status
  - Ereignis-Speicher
  - Ereignis-Policies
  - Action Callback-Prozedur
  - Resultat-View für Regeln

Anwendungsbereiche

- Workflow zur Informationsverteilung
- Workflow zur Aufgabenverwaltung
- Ereignis-basiertes Computing (EBC)
- Radio Frequenz ID-Verarbeitung (RFID)
- Supply Chain Management (SCM)
- Enterprise Application Integration (EAI)
- Business Asset Management (BAM)
- Business Process Management (BPM)

ORACLE

DECUS IT-Symposium 2006 - Vortrag 2K03 – Regeln ohne Ausnahme – Rules Manager-Konzepte

Was ist eine ECA-Regel?

- Regel
  - Direktive zur Beeinflussung von Prozessen
  - Event-Condition-Action (ECA) Regel
    - Event – Status-Information für Prozess
    - Condition – Bool'sche Bedingung, für Ereignis nach TRUE oder FALSE ausgewertet
    - Action – auszuführende Aktivität, wenn Condition nach TRUE ausgewertet

```
ON <event>
IF <condition>
THEN <action>
```

ORACLE

Beispiel für ECA-Regel

- Zwei mögliche Szenarien
  1. Einfache ECA-Regel – *"Wenn eine Bareinzahlung auf ein Girokonto von mehr als 15000 erfolgt, soll das Bundeskriminalamt wegen möglicher Geldwäsche informiert werden."*
  2. Komplexe ECA-Regel – *"Wenn eine Bareinzahlung auf ein Girokonto von mehr als 15000 erfolgt und anschließend von diesem Girokonto eine Auslandüberweisung über mehr als 15000 realisiert wird, soll das Bundeskriminalamt wegen möglicher Geldwäsche informiert werden."*

Erweiterung der ECA-Regel

- Zwei mögliche Szenarien
  1. Einfache ECA-Regel
     - Event – Datenstruktur zur Aufzeichnung des Geldeingangs
     - Condition – Bareinzahlung auf ein Girokonto von mehr als 15000
     - Action – Information des Bundeskriminalamtes wegen möglicher Geldwäsche
  2. Komplexe ECA-Regel
     - Event – Datenstruktur zur Aufzeichnung des Geldausgangs (zusätzlich zu 1.)
     - Condition – Auslandüberweisung vom Girokonto von mehr als 15000 (zusätzlich zu 1.)

Ereignis Terminologie

- Ereignis (event)
  - Ereignisstruktur ist abstrakter Datentyp mit Attributen
  - Abstrakter Datentyp realisiert als Objektdatentyp
  - Attributmenge beschreibt Ereignis-Eigenschaften
  - **DBMS_RLMGR**-Package zur Administration
    - Erstellung
    - Verwaltung
    - Nutzung
  - Definition als Objektstruktur
    - Einfach = Attribute der Objektstruktur rein skalar
    - Komplex = Objektstruktur bettet einfache Ereignisse ein

Auswertbarkeit von Ereignissen

- Einfach (primitiv)
  - Sofort verfügbar und atomar
  - Gebunden an einen speziellen Zeitpunkt
  - Existiert komplett oder überhaupt nicht
  - Regeln mit dem Ereignis voll auswertbar
- Komplex (zusammengesetzt)
  - Zwei oder mehr einfache Ereignisse kombiniert
  - Regeln ohne Auftreten aller korrespondierenden Ereignisse nicht auswertbar

Beispiel für einfaches Ereignis

- Geldeingang auf Girokonto

```
CREATE OR REPLACE
TYPE GeldEingang AS OBJECT (
      Konto          NUMBER,
      Einzahler      VARCHAR(30),
      Eingangsart    VARCHAR(30),
      Betrag         NUMBER,
      Datum          DATE);
```


Einfache Regelgruppe

- Regelgruppe (rule class)
  - Tabelle zur Speicherung einer Regelmenge
  - Verbindet Ereignis mit einer Regelmenge
  - Container für Regeln bezogen auf ein Ereignis

```
DBMS_RLMGR.CREATE_RULE_CLASS (
 rule_class   => 'Bankenaufsicht',
 event_struct => 'GeldEingang',
 action_cbk   => 'Informieren',
 actprf_spec  => 'Behoerde VARCHAR2(250)',
 rlcls_prop   => '<simple autocommit="yes"
                  storage="tablespace users"/>');
```

Regelgruppe als Tabelle

- Implementiert als Tabelle in der Datenbank
- Automatisch mit CREATE_RULE_CLASS angelegt
- Erhält ebenfalls automatisch den Namen der Regelgruppe, hier BANKENAUFSICHT

```
SQL> DESC Bankenaufsicht
Name                  Null?      Type
----------------------------------------
RLM$RULEID            NOT NULL   VARCHAR2(100)
BEHOERDE                         VARCHAR2(250)
RLM$RULECOND                     VARCHAR2(4000)
RLM$RULEDESC                     VARCHAR2(1000)
```


DECUS IT-Symposium 2006 - Vortrag 2K03 – Regeln ohne Ausnahme – Rules Manager-Konzepte

Regelgruppe erzeugt Prozedur

```
SQL> DESC Informieren
PROCEDURE informieren
Argument Name     Type           In/Out   Default?
---------------- -------------- -------- ---------------
RLM$EVENT         GELDEINGANG    IN
RLM$RULE          RECORD         IN
 RLM$RULEID       VARCHAR2(100)  IN
 BEHOERDE         VARCHAR2(250)  IN
 RLM$RULECOND     VARCHAR2(4000) IN
 RLM$RULEDESC     VARCHAR2(1000) IN

SQL> SELECT line, text FROM user_source WHERE
name='INFORMIEREN';
LINE TEXT
---- --------------------------------------------------
   1 procedure "INFORMIEREN" (rlm$event "GELDEINGANG",
     rlm$rule "BANKENAUFSICHT"%ROWTYPE) is
   2  begin
   3   null;
   4  end;
```

ORACLE

Beispiel für Prozedur

```
SQL> SELECT line, text FROM user_source WHERE
name='INFORMIEREN';
LINE TEXT
---- ------------------------------------------------------
   1 procedure "INFORMIEREN" (rlm$event "GELDEINGANG",
     rlm$rule "BANKENAUFSICHT"%ROWTYPE) is
   2  begin
   3   null;
   4  end;

CREATE OR REPLACE PROCEDURE INFORMIEREN (rlm$event
"GELDEINGANG", rlm$rule "BANKENAUFSICHT"%ROWTYPE) IS
BEGIN
 INSERT INTO aktion VALUES
 (rlm$rule.rlm$ruleid || '(' || rlm$rule.behoerde ||
 ') = TRUE (' || rlm$event.einzahler || ').');
END;
```


Regel Terminologie

- Regel (rule)
  - Definiert durch eine Ereignisstruktur
  - Ausgewertet für die Instanz eines Ereignisses
  - Besitzt
    - Bedingung (rule condition) – beschrieben durch Ereignisattribute mit SQL **WHERE**-Klausel Syntax
    - Voreinstellungen der Aktionen (action preferences) – spezifizieren Aktionen und deren Details
    - Aktion (rule action) – benutzt Ereignisattribute und Voreinstellungen

Beispiel für einfache Regel

- Einfache Regel
  - Ereignis **GELDEINGANG**
  - Bedingung **"EINGANGSART = 'BAREINZAHLUNG' AND BETRAG > 15000"**

```
INSERT INTO bankenaufsicht
(RLM$RULEID, BEHOERDE, RLM$RULECON,
 RLM$RULEDESC)
 VALUES
('Regel1','BKA','EINGANGSART="BAREINZAHLUNG"
 AND BETRAG > 15000','Bundeskriminalamt wird
 wegen möglicher Geldwäsche informiert');
```


Auswertung einer Regel

- Auswertung erfolgt bei definiertem Ereignis
- Anwendung veranlasst Prozeduraufruf **DBMS_RLMGR.PROCESS_RULES**
- Automatisch angelegter **INSERT**-Trigger
- Trigger für **UPDATE** / **DELETE** "von Hand"

```
DBMS_RLMGR.PROCESS_RULES (
 rule_class => 'BANKENAUFSICHT',
 event_inst => GeldEingang.getVarchar(
 123456789,'Muster','BAREINZAHLUNG',20000,
 to_date('17-MAY-2006 11:30',
         'dd-MON-yyyy hh24:mi')));
```

Zusammengesetztes Ereignis

- Ereignis-Definition komponiert aus Objekttyen

```
CREATE OR REPLACE
TYPE GeldAusgang AS OBJECT (
      Konto          NUMBER,
      Inhaber        VARCHAR(30),
      Ausgangsart    VARCHAR(30),
      Betrag         NUMBER,
      Datum          DATE);
```

```
CREATE OR REPLACE
TYPE KontoBewegung AS OBJECT (
 Eingang    GeldEingang,
 Ausgang    GeldAusgang);
```

ORACLE

Komplexe Regelgruppe

- Regelgruppe für zusammengesetzte Ereignisse
- Automatisch Spalte **RLM$CRTTIME** mit Datentyp **TIMESTAMP** ergänzt
- Weitere Objekte und Prozeduren intern erzeugt

```
DBMS_RLMGR.CREATE_RULE_CLASS (
 rule_class   => 'Bankenaufsicht',
 event_struct => 'KontoBewegung',
 action_cbk   => 'Informieren',
 actprf_spec  => 'Behoerde VARCHAR2(250)',
 rlcls_prop   => '<composite autocommit="yes"
                  storage="tablespace users"/>');
```

ORACLE

Beispiel für komplexe Regel

- Zusammenhänge mit XML-Tags formuliert

```
INSERT INTO bankenaufsicht
 (RLM$RULEID, BEHOERDE, RLM$RULECOND, RLM$RULEDESC) VALUES
 ('Regel1','BKA',
  '<condition>
    <and join="eingang.konto=ausgang.konto">
     <object name="Eingang">
      EINGANGSART="BAREINZAHLUNG" AND BETRAG > 15000
     </object>
     <object name="Ausgang">
      AUSGANGSART="AUSLANDUEBERWEISUNG" AND BETRAG > 15000
     </object>
    </and>
   </condition>',
  'Bundeskriminalamt wird wegen möglicher Geldwäsche
   informiert');
```


Definition komplexer Regeln

- Verfügbare XML-Tags
  - **CONDITION**, **AND**, **JOIN**, **OBJECT**, **NAME**
  - **ANY** – Auftreten bestimmer Zahl von Ereignissen
  - **NOT** – dient der Negation
  - **NOTANY** – reagiert auf Nicht-Auftreten
  - **SEQUENCE** – definiert notwendige Reihenfolge
  - **EQUAL** – zur Verbindung von Prädikaten
  - **COUNT** – zum Umsetzung einer ANY n-Semantik
  - **BY** – Festlegung von Zeitgrenzen

Modifikation der Prozedur

- Rudimentär angelegte Action-Prozedur **INFORMIEREN** wegen veränderter Eingabeparameter modifiziert
- Ausgeführt, wenn komplexe Bedingung erfüllt

```
CREATE OR REPLACE PROCEDURE INFORMIEREN (eingang GELDEINGANG,
ausgang GELDAUSGANG,rlm$rule BANKENAUFSICHT%ROWTYPE) IS
BEGIN
 INSERT INTO aktion VALUES
 (rlm$rule.rlm$ruleid || '(' || rlm$rule.behoerde ||
 ') = TRUE (' || eingang.einzahler || ').');
END;
```


Menge von Ereignissen

- Zwei aufeinander folgende Ereignisse

```
DBMS_RLMGR.PROCESS_RULES (
 rule_class => 'BANKENAUFSICHT',
 event_type => 'GeldEingang',
 event_inst => GeldEingang.getVarchar(
 123456789,'Muster','BAREINZAHLUNG',20000,
 to_date('17-MAY-2006 11:30',
         'dd-MON-yyyy hh24:mi')));
...
DBMS_RLMGR.PROCESS_RULES (
 rule_class => 'BANKENAUFSICHT',
 event_type => 'GeldAusgang',
 event_inst => GeldAusgang.getVarchar(
 123456789,'Muster','AUSLANDSUEBERWEISUNG',20000,
 to_date('17-MAY-2006 11:45',
         'dd-MON-yyyy hh24:mi')));
```

Abarbeitung von Ereignissen

1. Signalisierung einer **BAREINZAHLUNG** über 20000 durch **DBMS_RLMGR.PROCESS_RULES** -Aufruf
2. Angelegte Regel **Regel1** wird geprüft
3. Prozedur **INFORMIEREN** wird nicht ausgeführt
4. Signalisierung einer **AUSLANDSUEBERWEISUNG** über 20000 durch **DBMS_RLMGR.PROCESS_RULES** -Aufruf
5. Angelegte Regel **Regel1** wird überprüft
6. Da jetzt beide Regelbedingungen erfüllt sind, erfolgt Aufruf von Prozedur **INFORMIEREN** mit Parametern
7. An **INFORMIEREN** übergebene Parameter sind **RULEID**, **BEHOERDE**, **EINZAHLER**

Alternative zur Prozedur

- Resulat-View für Regelgruppe
  - Ausführung von Aktionen für passende Regeln durch Anwendung außerhalb der Datenbank
  - Nicht durch einfaches SQL **ORDER BY** auflösbare Konfliktsituationen
  - Vermeidung des Action-Prozeduraufrufs durch **DBMS_RLMGR.ADD_EVENT** -Aufruf statt **DBMS_RLMGR.PROCESS_RULES** API-Call

```
DBMS_RLMGR.CREATE_RULE_CLASS (
 ...
 rslt_viewnm  => 'PassendeRegeln',
 ...
);
```

Benutzung der Resultat-View

- Resultate verfügbar bis Ende der Session einer Regel
- Regel-Session i.d.R. von **`connect`** bis **`disconnect`**
- View zurücksetzen mit **`DBMS_RLMGR.reset_session`**

```
DBMS_RLMGR.ADD_EVENT (
 rule_class => 'BANKENAUFSICHT',
 event_inst => GeldEingang.getVarchar(
 123456789,'Muster','BAREINZAHLUNG',20000,
 to_date('17-MAY-2006 11:30',
         'dd-MON-yyyy hh24:mi')));
```

```
SELECT pr.rlm$eventid, pr.rlm$event,
     pr.rlm$ruleid, pr.actionType, pr.actionParam
FROM PassendeRegeln pr;
```

ORACLE

Nutzung des Rules Manager

- Schritte zur Nutzung des Rules Manager
  1. Ereignis-Strukturen repräsentierende Objektdatentypen für Ereignisse erzeugen
  2. Ereignis-Strukturen und Regelgruppen-Eigenschaften nutzende Regelgruppe erzeugen
  3. Action-Prozedur als Callback-Prozedur implementieren bzw. Fragment überschreiben
  4. Regeln zur Regelgruppe mit Standard-DML hinzufügen
  5. Regeln für eintreffende Ereignisse auswerten
  6. Action-Prozedur ausführen / Resultat-View nutzen

ORACLE

Management von Ereignissen

- "Verbrauch" von Ereignissen
  - Bestimmt, ob Ereignis(instanz) zur Ausführung mehrerer Regeln benutzt werden kann
- Lösung von Konflikten
  - Spezifiziert Reihenfolge der Ausführung der Regel-Aktionen
    - Basierend auf korrespondierenden Ereignissen
    - Abhängig von Attributen der einzelnen Regeln
- Dauer
  - Bestimmt Lebensdauer nicht "verbrauchter", einfacher Ereignisse bzw. Ereignisinstanzen

Schnittstellen zum Management

- SQL API mit PL/SQL-Paketen und -Prozeduren
  - **DBMS_RLMGR**-Package
- ORACLE Enterprise Manager- Werkzeuge
  - Browsen und Erzeugen neuer Ereignisstrukturen
  - Browsen und Erzeugen neuer Regelgruppen
  - Browsen und Erzeugen neuer Eigenschaften für Regelgruppen
  - Browsen, Erzeugen und Verändern von Regeln
- ORACLE JDeveloper-Erweiterung
  - Generiert Java Stubs und Source-Files
  - Versteckt PL/SQL APIs hinter Java-Methoden
  - Erlaubt Regelverarbeitung mit generiertem Code

Performance

- Rules Manager ausgesprochen performant
- Nutzung bewährter DB-Technologien
- Keine Deadlocks beim Zugriff großer Benutzergruppen auf gleiche Regelklassen
- Große Regelmengen i.d.R. nicht schädlich für Performance
- Automatisch angelegter **`domain index`** auf Regelgruppen-Tabelle gewährleistet schnellen Zugriff (nur Enterprise Edition)

Views für Entwickler

- Wichtige Views zu angelegten Objekten
  - **`USER/ALL_RLMGR_EVENT_STRUCTS`** – mit Informationen zu Ereignissen
  - **`USER/ALL_RLMGR_RULE_CLASSES`** – mit Informationen zu Regelklassen
  - **`USER/ALL_RLMGR_PRIVILEGES`** – mit Informationen zu Privilegien für Regelklassen
  - **`USER/ALL_RLMGR_COMPRCLS_PROPERTIES`** – mit Informationen zu einfachen Ereignissen für eine Regelgruppe

Und was gibt es noch?

- Rule Manager ist Weiterentwicklung der **EXPRESSION FILTER**-Funktionalität aus Oracle Database 10*g* Release 1
- Oracle empfiehlt Anwendungsmigration von **EXPRESSION FILTER**-Funktionalität nutzenden Projekten zu neuem Rules Manager
- Weiterführende Literatur
  - *Oracle PL/SQL Packages and Types Reference (B14258-01)* – Ausführliche Beschreibung des Package **DBMS_RLMGR**
  - *Oracle Application Developer's Guide - Rules Manager and Expression Filter (B14288-01)* – Konzepte und Vorgehen beim Arbeiten mit dem Rules Manager

Herzlichen Dank ...

... für Ihre Aufmerksamkeit!

N☺ch Fragen?

ORACLE

ORACLE®