Ereignissen auf der Spur: Big-Data-Analysen in der Datenbank

Alfred Schlaucher, ORACLE Deutschland B.V. & Co. KG

Viele Analysen im Big-Data-Umfeld suchen in den Unternehmensdaten nach Zusammenhängen, die durch einfaches Aggregieren nicht zu finden sind. Weit verbreitet sind sogenannte „Pattern-Analysen". Die einfachsten Analysen dieser Art verwenden das Suchen von Strings.

Viele Map-Reduce-Programme der ersten Stunde suchen und zählen Strings in Unternehmensdaten oder zugekauften Daten. Man entwickelte eine Liste von bewerteten Begriffen und zählte die Häufigkeit der Verwendung solcher Begriffe in Blogs, Mails oder Presseartikeln beispielsweise in Verbindung mit den Namen der eigenen Produkte. Diese „Häufigkeiten"-Analyse-Art berücksichtigt jedoch nicht die Beziehungen oder Abhängigkeiten, die zwischen den einzelnen Treffern (Vorkommnissen, Ereignissen oder nennen wir sie hier schon mal „Events") stattfinden:

- Gibt es Abhängigkeiten zwischen verschiedenen Entwicklungen?
- Welche Rolle spielt die Zeit bei der Abfolge einzelner Events?
- Treten unterschiedliche Events zeitlich zusammenhängend auf?

Zum besseren Verständnis sind solche Fragen auf folgende konkrete Beispiele übertragen:

- Ist am Verlauf von Aktienkurs-Änderungen ein wiederkehrendes Muster zu erkennen, mit der Möglichkeit, Prognosen zu treffen?
- Kann man anhand der Abfolge von Seitenaufrufen eines Web-Shop-Benutzers erkennen, warum bestimmte Abfolgen zum Kaufabschluss führen und andere nicht?
- Kann man anhand von Sensordaten einer Fließband-Maschine zu Stromverbrauch, Drehzahl, Temperatur etc. ein typisches Muster im zeitlichen Verlauf erkennen, um Ausfälle vorhersagen zu können?

Alle diese Analysen haben folgende Gemeinsamkeiten:

- Es werden mehrere Merkmale erhoben und in Bezug zueinander gesetzt
- Es wird eine Serie der Merkmale gemessen
- Meist orientiert sich diese Serie an einem zeitlichen Verlauf

Ein erstes Beispiel

Zum allgemeinen Verständnis stellt das Beispiel den Verlauf eines Aktienkurses vor. Der Wert einer Aktie wird täglich festgestellt. Tabelle 1 zeigt die einzelnen Datensätze. Beim Betrachten des zeitlichen Verlaufs kann man eine Entwicklung des Aktienkurses feststellen (siehe Abbildung 1).

Die Entwicklung ist von längeren Aufwärtsphasen und mehr oder weniger abrupten oder steilen Abwärtsbewegungen geprägt. In der unsortierten Ansammlung der massenhaften Datensätze einer Tabelle kann man diese Entwicklung nur schwer erkennen.

Das Beispiel ist überschaubar. Man stelle sich aber nicht 30 Sätze pro Monat und Aktie vor, sondern Sätze bezo-

SYMBOL	ZEIT	PREIS
ORCL	10.12.13	32
ORCL	11.12.13	32
ORCL	12.12.13	32
ORCL	13.12.13	33
ORCL	19.12.13	34
ORCL	09.12.13	31
ORCL	08.12.13	30
ORCL	30.12.13	35

Tabelle 1: Beispiel Aktienkurse

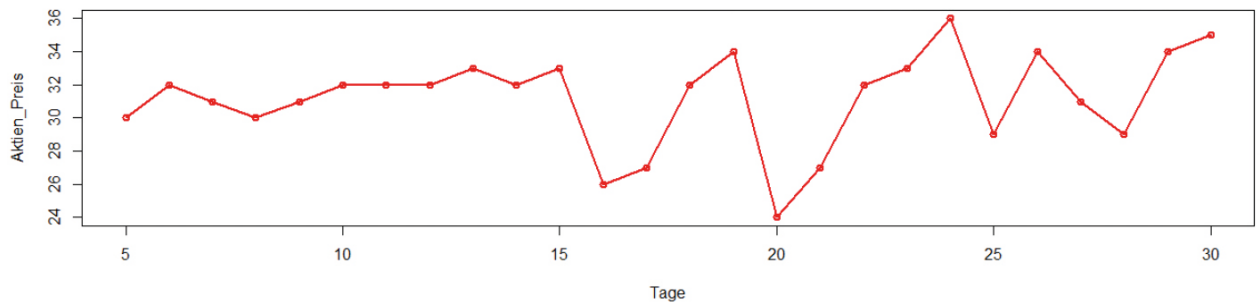

Abbildung 1: Der Aktienkurs aus der Beispiel-Tabelle über die Zeit visualisiert

```
SELECT * FROM Aktien
MATCH_RECOGNIZE (
  PARTITION BY SYMBOL ORDER BY Zeit
  MEASURES                                                  -- Ausgabe-Felder und Berechnungen
     match_number()             AS Pattern_Nr,            -- Durchzaehlen aller Pattern
     STRT.zeit                  AS Aufsetz_zeit,          -- Beginnzeit eines neuen Pattern
     first(DOWN.zeit)           AS Start_Down_Zeit,       -- Beginnzeit Down-Phase
     LAST(DOWN.zeit)            AS Ende_Down_Zeit,        -- Endezeit Down-Phase
     LAST(UP.zeit)              AS Ende_Up_zeit,          -- Endezeit Up-Phase
     count(Down.zeit)           AS anzahl_downs,          -- Anzahl Downs pro Pattern
     count(Up.Zeit)             AS anzahl_up              -- Anzahl Ups pro Pattern
  one ROW PER MATCH                                         -- nur eine Zeile pro Pattern
  AFTER MATCH SKIP TO LAST UP
  PATTERN  (STRT DOWN+ UP+)                                 -- Kombination aller Ereignisse
  DEFINE
     DOWN AS DOWN.preis < PREV(DOWN.preis),                 -- Down - Ereignis
     UP   AS UP.preis   > PREV(UP.preis)                    -- Up   - Ereignis
) MR
ORDER BY MR.symbol, MR.Aufsetz_zeit
/
```

Listing 1

gen auf 24 Stunden mal 365 Tage mal 10.000 Kurse. Wer will in diesen knapp 100 Millionen Sätzen Muster einer Abfolge finden? Dazu ein Hinweis: Es geht hier nicht um klassisches Aggregieren – also die Betrachtung einzelner Sätze und das Aufsummieren von Werten. Hier dreht es sich um Satz-übergreifende Analysen, das Identifizieren von unterschiedlich umfangreichen Gruppen in einer Menge verstreuter Sätze und das Analysieren der Sätze bezogen auf einen gruppierenden Zusammenhang. Das tun zum Beispiel Finanzunternehmen, bezogen auf Aktienkurse. Sie haben dafür recht komplexe Routinen entwickelt, die solche Analysen meist außerhalb der Datenhaltung sehr Zeit- und Ressourcen-aufwändig erledigen.

SQL-Pattern-Matching

Das SQL-Pattern-Matching in der Oracle-Datenbank 12*c* macht diese Art der Analyse jetzt wesentlich einfacher und vor allem ohne aufwändigen Datentransport innerhalb der Datenbank möglich. Die Analyse ist in SQL als mächtige Funktion „MATCH_RECOGNIZE“ eingebaut, man muss also kein zusätzliches Tool mit zusätzlichen Kosten und zusätzlichem Know-how einsetzen. Die Abfrage in Listing 1 analysiert die bereits vorgestellten Aktien-Datensätze. Tabelle 2 zeigt das Ergebnis.

Die letzten beiden Spalten der Ausgabe verraten bereits ein erstes Analyse-Ergebnis: Über alle erkannten Muster hinweg gibt es mehr Aufwärts- als Abwärtsbewegungen – über die Größe, also die Qualität der Schritte, wird allerdings noch nichts gesagt.

Wie das Ganze funktioniert

Der erste Schritt ist das Sortieren der Daten: Weil das tägliche Wechselspiel der Aktienkurse im Fokus steht, müssen die Tabellensätze für die Verarbeitung in einen zeitlichen Verlauf gebracht werden, und weil der Verlauf bezogen auf eine Aktiensorte interessiert, sind die Sätze nach Aktiensorten zu sortieren. Dies erreicht man über die Klausel: „PARTITION BY Symbol ORDER BY Zeit”.

Im zweiten Schritt muss definiert werden, was ein Muster (Pattern) ist: Fallende und steigende Kurse haben das Muster eines „V“. Die Definition dieses Musters lautet also:

- *Phase des Fallens (down)*
  Finde den Zeitpunkt, an dem ein Kurs fällt
- *Phase des Steigens (up)*
  Finde den Zeitpunkt, an dem der Kurs steigt

Die grafische Darstellung in Abbildung 1 zeigt dieses „V“ genau sechs Mal.

SYMBOL	PATTERN_NR	AUFSETZ_ZEIT	START_DOWN_ZEIT	ENDE_DOWN_ZEIT	ENDE_UP_ZEIT	ANZAHL_DOWNS	ANZAHL_UP
ORCL	1	06.12.13	07.12.13	08.12.13	10.12.13	2	2
ORCL	2	13.12.13	14.12.13	14.12.13	15.12.13	1	1
ORCL	3	15.12.13	16.12.13	16.12.13	19.12.13	1	3
ORCL	4	19.12.13	20.12.13	20.12.13	24.12.13	1	4
ORCL	5	24.12.13	25.12.13	25.12.13	26.12.13	1	1
ORCL	6	26.12.13	27.12.13	27.12.13	30.12.13	1	2

Tabelle 2

```
    PATTERN    (STRT DOWN+ UP+)
    DEFINE
       DOWN AS DOWN.preis < PREV(DOWN.preis),
       UP   AS UP.preis   > PREV(UP.preis)
```

Listing 2

```
    PATTERN    (STRT DOWN+ UP+ UP+ DOWN+)
    DEFINE
       DOWN AS DOWN.preis < PREV(DOWN.preis),
       UP   AS UP.preis   > PREV(UP.preis)
) MR
ORDER BY MR.symbol, MR.Aufsetz_zeit
/
```

Listing 3

In der Syntax der Funktion „MATCH_RECOGNIZE“ wird ein Muster über die „PATTERN“-Klausel und die zugehörige „DEFINE“-Unterklausel festgelegt (siehe Listing 2).

Die „DEFINE“-Subklausel definiert sogenannte „Pattern-Variablen“. In diesem Beispiel sind das „DOWN“ und „UP“. Diesen Variablen stehen für einen oder mehrere Datensätze, die bestimmte Bedingungen erfüllen („DOWN“-Set und „UP“-Set). In diesem Beispiel lautet die Bedingung für „DOWN“, dass der Aktienpreis von zwei aufeinanderfolgenden Sätzen sinkt. Über die Funktion „PREV(...)“ kann man auf den Vorläufersatz innerhalb des „DOWN-Sets“ zugreifen, da man Zugriff auf alle Felder aller zum Set gehörenden Sätze hat.

Die „PATTERN“-Klausel enthält die Kombination dieser Variablen: „UP“ folgt nach „DOWN“. Da es mehrere „DOWN“- oder „UP“-Phasen nacheinander geben könnte, steht hinter den Variablen-Namen das „+“-Zeichen. Es signalisiert, dass die „PATTERN“-Klausel auch „Regular Expressions“ versteht. Dies verdeutlicht, wie flexibel diese Art der Musterbeschreibung sein kann, denn „Regular Expressions“ können beliebige Kombinationen von Ereignissen beschreiben.

In dem Beispiel gibt es noch die dritte Variable „STRT“, die nicht über „DEFINE“ beschrieben ist. Als undefinierte Variable referenziert sie jeden Satz. Das bedeutet, dass das darauf folgende Muster „DOWN+“/„UP+“ an beliebigen Stellen in der Tabelle vorkommen kann. Ein Satz des „STRT“-Sets markiert den Beginnsatz eines neuen Musters.

In Listing 3 sind die Beispieldaten und die Abfrage leicht verändert. Über die veränderte „DEFINE“-Klausel findet das System auch andere Muster. Abbildung 2 stellt den grafischen Verlauf der geänderten Beispieldaten dar. Die Ausgabe zeigt Muster mit zwei zeitlich eingrenzbaren Aufwärtsphasen, die durch eine DOWN-Phase eingeschlossen sind (siehe Tabelle 3).

Im dritten Schritt gibt die „MEASURE“-Klausel das sortierte Datensatz-Material aus. Zur Verfügung steht jeder Spaltenwert eines jeden Satzes in den einzelnen Pattern-Sets, beispielsweise der letzte Wert der „DOWN“-Phase mit „LAST(DOWN.ZEIT)“. Möglich sind hier „LAST“, „FIRST“, „MAXIMUM“ und „MINIMUM“, aber auch „AVERAGE“ und andere statistische Funktionen, die auf ein Set von Werten anwendbar sind.

Die „MEASURE“-Ausgabe liefert nur einen Satz pro gefundenem „V“-Muster, also sechs für das erste Beispiel und zwei für das zweite. Das wird durch die Angaben „ONE ROW PER MATCH” erreicht. Diese Art der Ausgabe ist sinnvoll, wenn die Muster-Eigenschaften als Ganzes angesprochen sind, also etwa die Anzahl der Sätze in der „DOWN-Phase“ durch „COUNT(DOWN.ZEIT)“. Hier ergibt auch die Funktion „MATCH_NUMBER()“ einen Sinn. Sie zählt die einzelnen Pattern-Treffer mit einer laufende Nummer durch.

Das Gegenstück dazu ist „ALL ROWS PER MATCH“ – hier wird jeder Einzelwert geliefert. Weil man die Sätze optisch in der Ausgabe nicht mehr so

SYMBOL	PATTERN_NR	AUFSETZ_ZEIT	ANZAHL_DOWNS	ANZAHL_UP
ORCL	1	06.12.13	2	3
ORCL	2	16.12.13	2	4

Tabelle 3

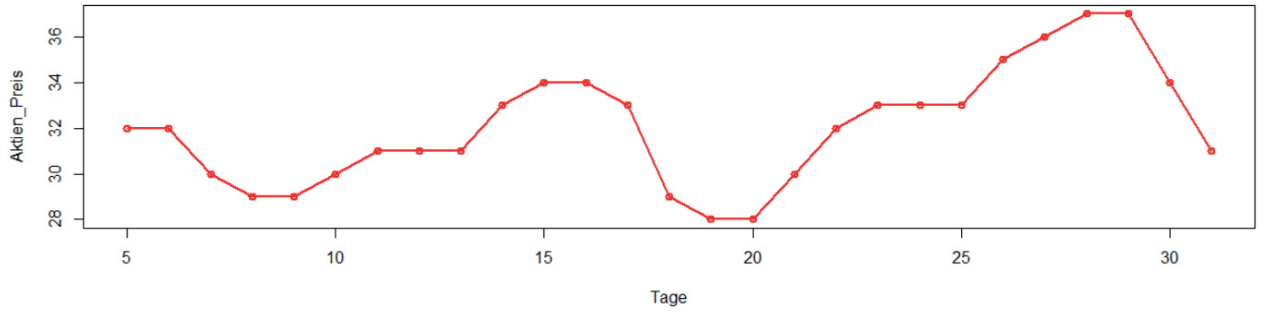

Abbildung 2: Der grafische Verlauf der geänderten Beispieldaten

leicht den einzelnen Phasen zuordnen kann, sollte man die Ausgabesätze mit der Funktion „CLASSIFIER()“ markieren. Sie liefert pro Ausgabesatz den zugehörigen Variablen-Namen, das ist der String „DOWN“ oder „UP“. Zur Ergebnismenge zählen alle Sätze, die einer „DOWN“- beziehungsweise „UP“-Phase zuzuordnen sind.

Nicht enthalten sind die Sätze mit gleichbleibendem Kursverlauf. Eine solche Ausgabesatzmenge kann selbst wieder Input für weitere Analysen sein. So könnte man beispielsweise den Output über eine View-Definition an eine R-Analyse oder eine grafische Aufbereitung weiterreichen.

Ein zweites Beispiel mit aufwändiger Ausgabe

Das nächste Beispiel erweitert den „MEASURE“-Bereich noch etwas. Es werden zusätzliche Ausgaben für den Aktienwert zu Beginn und am Ende der jeweiligen Fall- und Steig-Phasen und die durchschnittliche Fall- und Steig-Stärke pro Tag ausgegeben (siehe Listing 4). Die Möglichkeiten sind hier unbegrenzt, selbst Funktionen oder „CASE“-Ausdrücke sind möglich. Die Tabellen 4 und 5 zeigen die entsprechende Ausgabe.

```
SELECT *
FROM Aktien
MATCH_RECOGNIZE (
  PARTITION BY SYMBOL
  ORDER BY Zeit
  MEASURES
     match_number()        AS Pattern_Nr,
     STRT.zeit             AS Aufsetz_zeit,
     STRT.PREIS            AS Start_Preis,
     LAST(DOWN.PREIS)      AS Tief_Preis,
     first(DOWN.zeit)      AS Start_Down_Zeit,
  LAST(DOWN.zeit)          AS Ende_Down_Zeit,
     LAST(UP.zeit)         AS Ende_Up_zeit,
     LAST(up.preis)        AS Ende_Up_Preis,
     count(Down.zeit)      AS anzahl_downs,
     count(Up.Zeit)        AS anzahl_up,
   case
     when count(DOWN.Preis) = 1 then
         STRT.Preis - LAST(Down.preis)
     when count(DOWN.Preis) > 1 then
         (STRT.Preis-LAST(Down.preis))/count(DOWN.PREIS)
   end durchschn_Fallwerte ,
   case
     when count(UP.Preis) = 1 then
         LAST(up.preis) - LAST(DOWN.PREIS)
     when count(UP.Preis) > 1 then
         round((LAST(up.preis) - LAST(DOWN.PREIS))/count(UP.PREIS),1)
  end durchschn_Steigwerte
  one ROW PER MATCH
  AFTER MATCH SKIP TO LAST UP
  PATTERN (STRT DOWN+ UP+)
  DEFINE
     DOWN AS DOWN.preis < PREV(DOWN.preis),
     UP   AS UP.preis   > PREV(UP.preis)
) MR
ORDER BY MR.symbol, MR.Aufsetz_zeit
/
```

Listing 4

SYMBOL	PATTERN_NR	AUFSETZ_ZEIT	START_PREIS	TIEF_PREIS	START_DOWN_ZEIT	ENDE_DOWN_ZEIT	~
ORCL	1	06.12.13	32	30	07.12.13	08.12.13	~
ORCL	2	13.12.13	33	32	14.12.13	14.12.13	~
ORCL	3	15.12.13	33	26	16.12.13	16.12.13	~
ORCL	4	19.12.13	34	24	20.12.13	20.12.13	~
ORCL	5	24.12.13	36	29	25 .12.13	25.12.13	~
ORCL	6	26.12.13	34	31	27.12.13	27.12.13	~

Tabelle 4

~	ENDE_UP_ZEIT	ENDE_UP_PREIS	ANZAHL_DOWNS	ANZAHL_UP	DURCHSCHN_FALLWERTE	DURCHSCHN_STEIGWERTE
~	10.12.13	32	2	2	1	1
~	15.12.13	33	1	1	1	1
~	19.12.13	34	1	3	7	2,7
~	24.12.13	36	1	4	10	3
~	26.12.13	34	1	1	7	5
~	30.12.13	35	1	2	3	2

Tabelle 5

Ein drittes Beispiel mit komplexer Mustersuche

Das nächste Beispiel sucht in dem Datenstrom der sich ständig ändernden Aktienwerte nach einem plötzlichen Fall von mehr als 8 Prozent (definierter Schwellenwert, siehe Listing 5 mit der Tabelle 6 als Ausgabe). Die Beispieldaten-Menge ist die gleiche wie im ersten Beispiel.

```
SELECT *
FROM Aktien
MATCH_RECOGNIZE (
  PARTITION BY SYMBOL
  ORDER BY Zeit
  MEASURES
      match_number()          as Pattern_Nr,
      A.Zeit                  as Ausgangs_Zeit,
      A.Preis                 as Ausgangs_Preis,
      B.Preis                 as Gefallener_Preis,
      count(C.*)+1            as Anzahl_Tage
   one ROW PER MATCH
   AFTER MATCH SKIP PAST LAST ROW
   PATTERN    (A B C*)
   DEFINE
      B as (B.preis - A.preis) / A.preis < -0.08,
      C as C.preis < A.preis)
```

Listing 5

SYMBOL	PATTERN_NR	AUSGANGS_ZEIT	AUSGANGS_PREIS	GEFALLENER_PREIS	ANZAHL_TAGE
ORCL	1	15.12.13	33	26	3
ORCL	2	19.12.13	34	24	4
ORCL	3	24.12.13	36	29	5

Tabelle 6

```
SELECT * FROM Aktien_handel
MATCH_RECOGNIZE (
  PARTITION BY SYMBOL
  ORDER BY Zeit
  MEASURES
      match_number()          as Pattern_Nr,
      FIRST (A.Zeit)          as in_hour_of_trade,
      SUM (A.VOLUMEN)         as sum_of_large_volumes
   one ROW PER MATCH
   AFTER MATCH SKIP PAST LAST ROW
   PATTERN    (A B* A B* )
   DEFINE
      A as ((A.volumen > 30000 ) and
            ((A.Zeit - FIRST (A.Zeit)) < 0.042 )),
      B as ((B.volumen <= 30000 ) and
            ((B.Zeit - FIRST (B.Zeit)) < 0.042))
 )
```

Listing 6

SYMBOL	PATTERN_NR	IN_HOUR	SUM_OF_LARGE_VOLUMES
ORCL	1	23.12.13	64300

Tabelle 7

Ein viertes Beispiel

Das nächste Beispiel beobachtet die Menge der Aktien, die innerhalb einer bestimmten Zeit verkauft werden. Die Regel lautet: „Suche alle Aktienverkaufs-Events innerhalb einer Stunde, die einen zeitlichen Abstand von weniger als 60 Minuten haben und ein Volumen von mehr als 30.000 Stück ausmachen.“ Wenn solche Muster erkannt werden, kann ein Täuschungsverdacht bestehen (siehe Listing 6).

Der Wert „0.042“ ist der 24ste Teil von 1. Die Subtraktion der beiden Datumswerte liefert das Ergebnis bezogen auf ganze Tage zurück, wobei die Einheit ein Tag ist. Damit entspricht „0.042“ einem Wert von 60 Minuten. Die Alternative wäre „((A.Zeit - FIRST (A.Zeit))*24 < 1)“. Tabelle 7 zeigt die Ausgabe und Tabelle 8 die Daten für dieses Beispiel.

Zeit	PREIS	SYMBOL	VOLUMEN
20.12.13	24	ORCL	1233
21.12.13	27	ORCL	453
23.12.13	32	ORCL	33300
23.12.13	36	ORCL	453
23.12.13	29	ORCL	31000
23.12.13	33	ORCL	4432
26.12.13	34	ORCL	422
27.12.13	31	ORCL	44
30.12.13	35	ORCL	453

Tabelle 8

Ein letztes Beispiel aus dem Bereich der Sensordaten

Einsatzfelder des „Pattern Matching“ gibt es in allen Branchen. Hier noch ein Sensordaten-Beispiel aus der Fertigungsindustrie. Die Maschinen einer Fertigungsanlage werden über eine Vielzahl von Sensoren überwacht. Etwa alle 20 Sekunden entsteht ein neuer Datensatz. Tabelle 9 zeigt einen Ausschnitt.

Für die gemessenen Maschinen gelten Maximalwerte für Temperatur (200°), Drehzahl (1000 Upm) und Stromleistungsaufnahme (1300 Watt). Während des Betriebs überschreiten die Maschinen diese Werte immer wieder, was zu

SATZNR	MESSZEITPUNKT	STUECKNR	DREHZAHL	WATT	TEMPERATUR
499994	30.04.2008 11:04:55	5000	1031	1094	139
499995	30.04.2008 11:05:16	5000	991	1315	159
499996	30.04.2008 11:05:37	5000	1030	1284	205
499997	30.04.2008 11:05:58	5000	882	1182	186
499998	30.04.2008 11:06:18	5000	818	1406	204

Tabelle 9

frühzeitigem Verschleiß und häufiger Wartung führt. In einer Abfrage sollen Zeiten für Konstellationen der drei Kennwerte gefunden werden, in denen alle Werte nacheinander die Höchstmarke überschreiten (siehe Listing 7).

Zunächst sind alle Sätze nach „MaschinenNr“ und „StueckNr“ gruppiert, also nach dem Werkstück, das eine bestimmte Maschine gerade produziert. Pro Werkstück und Maschine fallen etwa 100 Mess-Sätze an. Sie sind nach der Mess-Zeit und einer künstlichen Satznummer sortiert.

Die „DEFINE“-Klausel sucht für die jeweiligen Kennzahlen („DREHZAHL“, „WATT“ und „TEMPERATUR“) durch einen Vergleich des aktuellen Werts mit seinem Vorgänger nach einem Anstieg der Werte. Zudem betrachtet man nur die Sätze, in denen eine Überschreitung der Maximalwerte vorliegt (etwa „DREHZAHL > 1000“).

Die „PATTERN“-Klausel bringt die Vorkommnisse „A“, „B“ und „C“ in eine zeitliche Reihenfolge.

Man betrachte also zunächst die Drehzahl, in der zeitlichen Folge die Leistungsaufnahme und dann die Temperatur. Dieses Beispiel deckt nicht ganz die Realität ab, da Drehzahl und Leistungsaufnahme zeitlich wahrscheinlich synchron stattfinden, während der Temperaturanstieg nachhinkt, da eine Maschine nur dann schneller läuft, wenn auch mehr Strom fließt. Die Folge ist ein zeitlich versetzter Temperaturanstieg, der sich über mehrere Sekunden hinziehen kann. An diesem Beispiel ist gut sichtbar, wie mit wenigen Handgriffen die Logik der Analyse schnell und leicht änderbar ist. Auch das Einstellen der Überschreitung der Grenzwerte ist leicht machbar (siehe Listing 8 und Tabelle 10 mit einem Ausschnitt aus der Liste der gefundenen „PATTERN“).

Dieses Beispiel kann immer noch verbessert werden. Es kann sein, dass eine Steigerung der Leistungsaufnahme nicht unbedingt zu einer Erhöhung der Drehzahl führen muss. Im Gegenteil: Wird die Maschine durch einen unvorhersehbaren Vorgang an der Fertigungsstraße stark gebremst oder kommt sie sogar zum Stillstand, steigt der Stromverbrauch enorm, was in kurzer Zeit zur Schädigung der Maschine führt. Listing 9 zeigt das Muster, das eine solche Situation finden müsste.

```
SELECT * FROM Sensor_daten
MATCH_RECOGNIZE (
  PARTITION BY MaschinenNr, StueckNr
  ORDER BY MessZeit, satznr
  MEASURES
      match_number()          as Pattern_Nr,
      FIRST (A.satznr)        as first_Satznr,
--    satznr                  as satznr,
              LAST (A.satznr)         as Last_satznr,
              count(A.satznr)         as count_satznr,
              FIRST (A.drehzahl)      as first_drehzahl,
              LAST (A.drehzahl)       as LAST_drehzahl,
--    LAST (A.Watt)           as last_A_Watt,
--            FIRST (b.Watt)          as First_watt,
      LAST (b.Watt)           as LAST_watt,
              LAST (c.temperatur)     as last_temperatur,
--    c.messzeit              as c_messzeit,
      FIRST (A.messzeit)      as Messzeit
   one ROW PER MATCH
   AFTER MATCH SKIP PAST LAST ROW
   PATTERN    (A+ B C)
   DEFINE
      A as (A.drehzahl > PREV(A.DREHZAHL) and DREHZAHL > 1000) ,
      B as (B.watt > PREV(B.WATT) and B.watt > 1280),
      C as (C.Temperatur > PREV(C.Temperatur) and C.temperatur > 245)
 )
```

Listing 7

```
   PATTERN    (A+ C)
   DEFINE
      A as (A.drehzahl > PREV(A.DREHZAHL)    and
       DREHZAHL > 1020)              and
           (A.watt > PREV(A.WATT))           and
           (A.Watt > 1500)                   ,
      C as (C.Temperatur > PREV(C.Temperatur) and C.temperatur > 245)
```

Listing 8

STUECKNR	PATTERN_NR	LAST_SATZNR	LAST_DREH-ZAHL	LAST_WATT	LAST_TEMPE-RATUR	MESSZEIT
8	1	720	1040	1547	252	01.01.2008 04:09:52
46	1	4531	1021	1530	250	02.01.2008 02:12:26
46	2	4534	1048	1548	247	02.01.2008 02:13:29
63	1	6220	1033	1524	248	02.01.2008 11:58:35
69	1	6882	1031	1514	252	02.01.2008 15:48:20
105	1	10426	1040	1527	247	03.01.2008 12:18:14
118	1	11728	1035	1528	249	03.01.2008 19:50:05
145	1	14414	1047	1543	248	04.01.2008 11:22:14
156	1	15504	1037	1535	248	04.01.2008 17:40:30
160	1	15910	1032	1527	255	04.01.2008 20:01:24

Tabelle 10

```
PATTERN       ( C A)
   DEFINE
       C  as (C.Temperatur > PREV(C.Temperatur)    ,          -- einfacher Temperaturanstieg
      (A.watt > PREV(A.WATT))        and (A.Watt > 1300) ,          -- unterer Grenzwert / Anstieg Strom
       A as (A.drehzahl <= PREV(A.DREHZAHL)                  -- gleiche oder sinkende Drehzahl
```

Listing 9

Vorstufe für zusätzliche Statistiken und Data Mining

Die Ergebnismenge des Beispiels zuvor kann je nach Festlegung der Grenzwerte mehrere Tausend Zeilen umfassen. Es liegt deshalb nahe, diese Menge einer weiteren Analyse zu unterziehen. So könnte man zum Beispiel herausfinden, ob Drehzahl und Temperatur miteinander korrelieren. Dafür eignet sich besonders gut die Sprache „R“, weil damit Analysen in der Datenbank möglich sind, ohne die Pattern-Ergebnisdaten zu bewegen.

Eine Korrelation zwischen allen drei Kennwerten der gefundenen Muster, also „DREHZAHL“, „LEISTUNGSAUFNAHME“ und „TEMPERATUR“, könnte mit einer Routine leicht durchgeführt werden (siehe Abbildung 3). Als Vorarbeit in der Datenbank fasst eine View-Definition die Ergebniszeilen zusammen (siehe Listing 10). Listing 11 zeigt das Oracle-R-Enterprise-Skript. Es ist hier abgedruckt, um aufzuzeigen, wie schnell und einfach man zu Analyse-Ergebnissen gelangt.

Skalierung

Sensordaten werden oft als Beispiel für Big-Data-Anwendungen angeführt. Dieses Beispiel zeigt jedoch, wie gut solche Daten in einer klassischen Datenbank aufgehoben sind. Obwohl in einem Abstand von 20 Sekunden immer ein neuer Satz pro Maschine entsteht, kommt man bei einer praxisrelevanten Testreihe kaum über eine Million Sätze. Um eine für Testzwecke höhere Datenmenge von 5 Millionen Sätzen zu erzeugen, wurden für das oben ausgeführte Beispiel über fünf Jahre Daten gesammelt.

Die hier vorgestellten „MATCH_RECOGNIZE“-Funktionen benötigten auf einem Standard-Laptop gerade mal 35 Sekunden Bearbeitungszeit. Durch den Parallelisierungsgrad von zwei (mehr ist auf einem solchen Laptop nicht spürbar), richtig aktualisierte Statistiken und Caching sinkt die Ausführzeit auf unter fünf Sekunden. Am Ausführungsplan der Datenbank erkennt man, dass die Bearbeitung kom-

```
CREATE VIEW SENSOR_PATTERN as
     SELECT Drehzahl, Watt, Temperatur  FROM ( Select des vorigen
     Beispiels)
```

Listing 10

```
ore.connect (connect-Daten zum DB-Schema)]
ore.doEval(
   function(param) {
              library(ORE)
       ore.connect(-> DB-Schema)
ore.sync(DB-Schema)
ore.attach(SENSOR_PATTERN)
cor.matrix <- cbind(Drehzahl,Watt,Temperatur)
rcorr(cor.matrix)
});
```

Listing 11

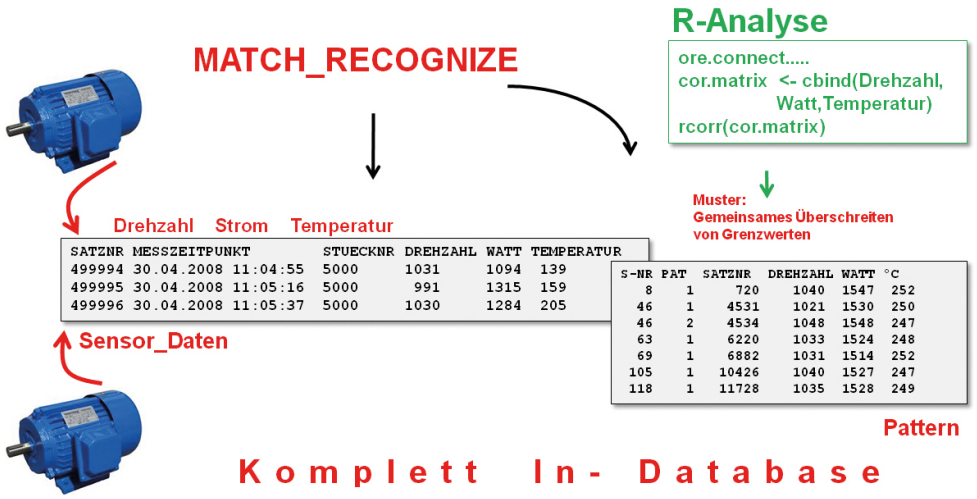

Abbildung 3: Einsatz-Szenario „Pattern-Suche“ – als Input die Korrelations-Analyse mit R

plett im Kern der Datenbank (letztlich im Hauptspeicher) abläuft (siehe Abbildung 4). Natürlich wird die Komplexität der Anwendung in der Praxis steigen. Dann verwendet man allerdings auch keinen Laptop, sondern einen größeren Datenbank-Server mit einer echten Parallelisierungs-Option. Damit würde die Pattern-Suche auf unter eine Sekunde Laufzeit sinken und wäre auch für Realtime-Szenarien einsatztauglich.

Streaming und Event-Measurement

Klassische Kennzahlen-Analysen betrachten in der Regel abgelaufene Berichtszeiträume. Sie liefern Tages-, Monats-, Quartals- oder Jahres-Ergebnisse und beantworten Fragen über das Wie, also wie etwas gelaufen ist. Um künftige Geschäftsentwicklungen vorhersagen oder sogar durch frühzeitiges Erkennen neuer Entwicklungen planen zu können, reichen solche Kennzahlen meist nicht aus. Hierzu ist nach dem Warum zu fragen, also warum etwas geschehen ist.

Über das vorgestellte Pattern-Matching lassen sich Ereignisse und Zusammenhänge zwischen den Ereignissen aufdecken. Hierzu ist regelmäßiges Beobachten der relevanten Geschäftstransaktionen nötig (Event-Monitoring, siehe Abbildung 5). Die relevanten Transaktionsdaten werden dazu permanent den Analysen zugeführt (Streaming). Beim Eintreten eines nicht geplanten Ereignisses kann man spontan und rechtzeitig reagieren.

Pattern-Matching ist ideal für diese Verfahren. Allerdings ist „Streaming“ nicht gleich „Streaming“. Es ist nicht damit getan, Daten fließen zu lassen und sich diese Daten beim Vorbeifließen anzuschauen. Drei wichtige Aspekte sind zu berücksichtigen:

1. Es muss Regeln geben. Man kann nur etwas überprüfen, wenn man es mit etwas anderem vergleicht, von dem es dann abweicht oder nicht. Beim vorgestellten Feature sind das die Klauseln „PATTERN“ und „DEFINE“.
2. Es muss einen messbaren Bereich geben. Eine Veränderung (ein Ereignis) muss also auch als solches erkennbar sein. In den genannten Beispielen war das beispielsweise die „PREV“-Funktion innerhalb der „DEFINE“-Klausel. Man könnte sagen, Streaming bedeutet immer: „Vergleiche den vorhergehenden Wert mit einem nachfolgenden”. Wenn man sich jedoch die möglichen Regeln anschaut, dann ist nicht immer klar, was genau ein Folge- beziehungsweise ein Vorgängerwert darstellt. Bei dem Aktien-Beispiel könnte beispielsweise ein Wert nach einer „DOWN“-Phase über mehrere Tage hinweg (mehrere Messungen hinweg) gleich bleiben, ohne in eine „UP“-Phase zu wechseln. Die Ausdehnung des zu messenden Bereichs (die Menge der Sätze) ist also von anderen Faktoren abhängig.
3. Ein historisches Gedächtnis in Form eines Data Warehouse ist erforderlich. Durch das Wissen über das Vergangene lassen sich Verhaltensregeln und Planung modifizieren.

Zumindest der zweite Punkt führt dazu, dass das hier betrachtete Streaming doch keinen kontinuierlichen Datenstrom darstellt, sondern eher eine Kette von gruppierten Sätzen, die zusammenhängend zu analysieren sind. Das „PARTITION BY“ des „PATTERN MATCHING“ gibt bereits eine solche Gruppierung vor. Für ein kontinuierliches Event-Monitoring muss es darüber hinaus eine Ablaufumgebung geben, die das Pattern-Matching im Kern periodisch aufruft. Diese Datenbank-Techniken unterstützen das Streaming:

- Einsatz des „MATCH RECOGNIZE“ im Select einer Materialized View. Man kann das Refresh der Materialized View auf „ON COMMIT“ setzen, sodass bei Änderung der Ausgangs-Tabelle immer wieder aufs Neue Pattern gesucht werden. Diese Variante ist jedoch nur sinnvoll, wenn sich die Änderungen in der Ausgangs-Tabelle in Grenzen hal-

```
-------------------------------------------------------------------------------------------------
| Id  | Operation                                        | Name         | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT                                 |              |  4999K|   166M| 34419   (1)| 00:00:02 |
|   1 |  VIEW                                            |              |  4999K|   166M| 34419   (1)| 00:00:02 |
|   2 |   MATCH RECOGNIZE SORT DETERMINISTIC FINITE AUTOMATON|          |  4999K|    81M| 34419   (1)| 00:00:02 |
|   3 |    TABLE ACCESS FULL                             | SENSOR_DATEN |  4999K|    81M|  6911   (1)| 00:00:01 |
-------------------------------------------------------------------------------------------------
```

Abbildung 4: Ausführungsplan der „MATCH_RECOGNIZE“-Verarbeitung auf fünf Millionen Sätzen

ten. Die Alternative ist ein regelmäßig aufgerufener expliziter Refresh.
- Die Verwendung des „MATCH RECOGNIZE“ in einer „Table Function“. Man erreicht damit eine zusätzliche Parallelität von Pattern-Suche und Weiterverarbeitung bereits gefundener Pattern.

Fazit

Das neue Pattern-Matching der Oracle-12*c*-Datenbank ist ein genialer Griff in die Möglichkeiten des klassischen SQL. Einerseits löst man aktuelle Aufgabenstellungen, wie sie zum Beispiel im Rahmen des Big Data vorkommen. Andererseits minimiert man die Aufwände auf das Nötigste:

- Die Daten bleiben in der Datenbank
- Datenbank-Ressourcen mit Know-how und auch Infrastruktur werden wiederverwendet
- Komplexe Analysen über definierbare Datenbereiche hinweg sind ohne Programmierung leicht machbar

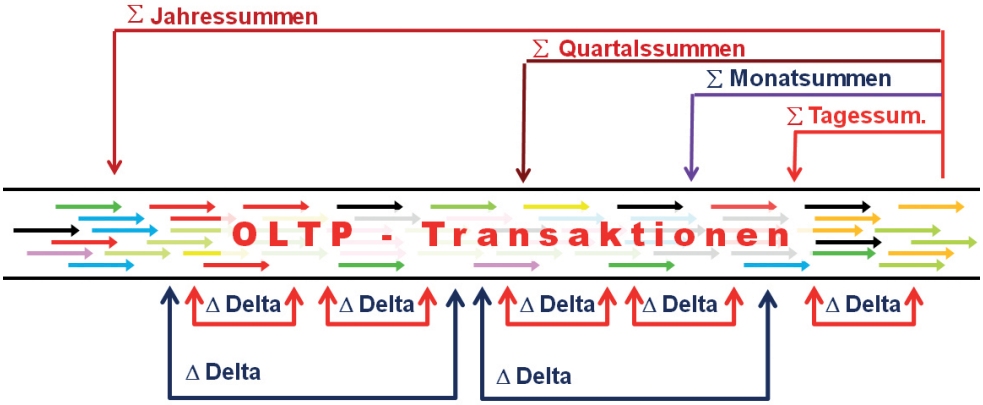

Abbildung 5: Event-Monitoring im Gegensatz zum klassischen Aggregieren vergangener Transaktionen

Hinweis:

Die Beispieldaten zu diesem Artikel können unter „http://www.oracle-dwh.de/downloads/AutoIndex-2.2.4/index.php?dir=Artikel/&file=BSP_Daten_Match_Recognize.zip“ heruntergeladen werden.

Alfred Schlaucher
alfred.schlaucher@oracle.com