Werken met tijdreeksen, een overzicht van de mogelijkheden

RapidBase nogmaals onder de loep


Rick van der Lans

Twee nummers terug schreef Rick van der Lans over RapidBase, een in-memory databaseserver met ingebouwde ondersteuning voor tijdreeksen. Van de werking hiervan gaf hij in dat artikel (DB/M 7/2001) enkele voorbeelden. Voor sommige lezers bleek dit niet voldoende. Hun interesse was gewekt en zij wilden er meer over weten. Wat kon er nu precies wel en wat niet met deze implementatie van tijdreeksen? In dit extra artikel over RapidBase is een aantal voorbeelden uitgewerkt om deze nieuwsgierige DB/M-lezers tegemoet te komen.

Simpel gesteld, is een tijdreeks een reeks van historische kolomwaarden die een record ooit gehad heeft. Zo vormen alle waarden die ooit voor werknemer 83 in de SALARIS-kolom van de WERKNEMER-tabel gestaan hebben een tijdreeks. Een ander voorbeeld van een tijdreeks wordt gevormd door alle waarden die het Shell-aandeel het laatste jaar heeft gehad.

Stel dat de historie van de adresgegevens (van alle werknemers) bijgehouden moet worden, omdat gebruikers willen kunnen opvragen waar een werknemer op een bepaalde dag in het verleden woonde. In RapidBase kunnen we hiervoor de volgende, simpele constructie gebruiken:

```
CREATE TABLE WERKNEMERS
   ( WERKNEMER_ID    INTEGER PRIMARY KEY,
     NAAM            CHAR(40),
     GEBOORTE_DATUM  DATE,
     ADRES           HISTORY (
        STRAATNAAM       CHAR(30),
        STRAATNUMMER     CHAR(5),
        PLAATSNAAM       CHAR(30),
        PROVINCIE        CHAR(2),
        POSTCODE         CHAR(6) )
   )
```

In deze tabel is een nieuwe kolom genaamd ADRES met het datatype HISTORY gecreëerd. Daarbinnen zijn de individuele adreskolommen opgenomen. Door middel van het HISTORY-datatype geven we aan dat als een adrescomponent wijzigt (om precies te zijn één of meer van de adreswaarden), de oude waarde niet door de nieuwe overschreven moet worden, maar bewaard moet blijven.

DE DATABASESTRUCTUUR

Om voorbeelden van instructies te laten zien, is een voorbeeld databasestructuur nodig. De gekozen structuur, bestaande uit drie tabellen, wordt getoond in figuur 1. De feitentabel, genaamd TRANSACTIES, bevat voor elke aankooptransactie een uniek nummer, een transactiedatum, het nummer van de klant die voor de aankoop verantwoordelijk was, het nummer van het gekochte product, het aantal verkochte producten en de per product betaalde prijs.

```
CREATE TABLE TRANSACTIES
  ( TRANSACTIE_NR    INTEGER NOT NULL PRIMARY KEY,
    TRANSACTIE_DATUM DATE    NOT NULL,
    KLANT_NR         INTEGER NOT NULL,
    PRODUCT_NR       INTEGER NOT NULL,
    AANTAL           INTEGER NOT NULL,
    PRIJS            DECIMAL(7,2) NOT NULL
    )
```

De database kent twee dimensietabellen: de KLANTEN- en de PRODUCTEN-tabel. De eerste heeft KLANT_NR als primaire sleutel en bevat onder andere een NAAM-kolom en twee kolommen met tijdreeksen, genaamd STAD en STATUS. Bij de kolom STAD gaan we er voor het gemak vanuit dat er geen twee steden met

Fins Researchproject

Ter herinnering: RapidBase is het resultaat van een research-project dat uitgevoerd wordt door het Finse VTT IT (Valtion Teknillinen Tutkimuskeskus Information Technology). De initiële doelstelling van dit project was de ontwikkeling van een relationele databaseserver die als platform kon dienen voor onderzoek naar nieuwe mogelijkheden, zoals tijdreeksen.

dezelfde naam bestaan. Status is een getal tussen de 1 en de 10. Van beide kolommen houdt RapidBase dus automatisch de historie vast.

```
CREATE TABLE KLANTEN
   ( KLANT_NR     INTEGER   NOT NULL PRIMARY KEY,
     NAAM         CHAR(30)  NOT NULL,
     STAD         HISTORY   (NAAM CHAR(30)),
     STATUS       HISTORY   (STATUS_NR INTEGER),
     ...
   )
```

PRODUCT_NR is de primaire sleutel van de PRODUCTEN-tabel. Deze tabel bevat tevens de tijdreeks ADVIES_PRIJS.

```
CREATE TABLE PRODUCTEN
 ( PRODUCT_NR     INTEGER NOT NULL PRIMARY KEY,
 ADVIES_PRIJS     HISTORY (BEDRAG DECIMAL(7,2)),
 ...
 )
```

DE VOORBEELDINSTRUCTIES

Om een beeld te geven van de mogelijkheden van RapidBase geven we enkele voorbeelden van vragen en bijbehorende SELECT-instructies.

Vraag 1: Geef de lijst met steden waar momenteel klanten wonen.

```
SELECT    STAD.NAAM
FROM      KLANTEN
GROUP BY  STAD.NAAM
```

Toelichting: wordt in de instructie niets opgegeven aangaande de tijd, dan wordt de instructie uitgevoerd op de huidige gegevens. Historische gegevens zullen niet in het resultaat voorkomen.

Omdat RapidBase geen DISTINCT toestaat, wordt de GROUP BY-component gebruikt om dubbele waarden uit het resultaat te verwijderen. Anders had de instructie er nog eenvoudiger uitgezien:

```
SELECT    DISTINCT STAD.NAAM
FROM      KLANTEN
```

Vraag 2: Geef het aantal huidige klanten per stad.

```
SELECT    STAD.NAAM, COUNT(*)
FROM      KLANTEN
GROUP BY  STAD.NAAM
```

Toelichting: aan de bovenstaande instructies mogen we ook de specificatie VALID NOW toevoegen om expliciet aan te geven dat het systeem naar de huidige situatie moet kijken. De bovenstaande instructie zou er dan als volgt uitzien:

```
SELECT    STAD.NAAM, COUNT(*)
FROM      KLANTEN
WHERE     VALID NOW
GROUP BY  STAD.NAAM
```

Vraag 3: Geef het aantal steden dat in het systeem bekend was op 23 november 1998.

```
SELECT    COUNT(STAD.NAAM)
FROM      KLANTEN
WHERE     VALID '1998-11-23'
```

Toelichting: door in de VALID-specificatie een specifieke datum op te nemen, geven we aan dat RapidBase niet naar de huidige waarden moet kijken, maar naar de situatie zoals die was op de gespecificeerde datum.

Vraag 4: Geef de nummers en de namen van de klanten die vóór 1999 in Rotterdam woonden en na 1999 een status hadden hoger dan 8.

```
SELECT    KLANT_NR, NAAM
FROM      KLANTEN
WHERE     STAD.NAAM = 'Rotterdam'
AND       STAD.OTS_END  '1999-01-01'
AND       STATUS.STATUS_NR  8
AND       STATUS.OTS_END  '1999-12-31'
```

Toelichting: OTS geeft de datum waarop een waarde actueel is geworden en OTS_END geeft de datum waarop een waarde ver-

vallen is. Deze twee specificaties zijn alleen bij een tijdreeks te gebruiken. De waarde van OTS wordt door RapidBase werkelijk in de database opgeslagen, terwijl de OTS_END-waarde virtueel is, dus afgeleid wordt. Is de waarde nog actueel, dan is OTS_END gelijk aan CURRENT TIMESTAMP.

Vraag 5: Geef het aantal huidige klanten -waar zij momenteel ook wonen- die op 23 november 1998 in Rotterdam woonden.

```
SELECT    STAD.NAAM, COUNT(STAD.NAAM)
FROM      KLANTEN
WHERE     VALID '1998-11-23'
AND       STAD.NAAM = 'Rotterdam'
GROUP BY  STAD.NAAM
```

Toelichting: uit deze instructie blijkt dat situaties uit verschillende momenten in de tijd zijn te combineren. In deze instructie worden huidige gegevens gecombineerd met de gegevens die golden op 23 november 1998.

Vraag 6: Geef van klant 4 de lijst met steden waar hij ooit gewoond heeft.

```
SELECT    STAD.NAAM
FROM      KLANTEN
WHERE     NOT VALID NOW
AND       KLANT_NR = 4
```

Toelichting: de specificatie NOT VALID NOW zorgt ervoor dat RapidBase alleen naar verouderde gegevens kijkt.

Vraag 7: Geef van elk product het productnummer en de huidige en twee daarvoor geldende adviesprijzen.

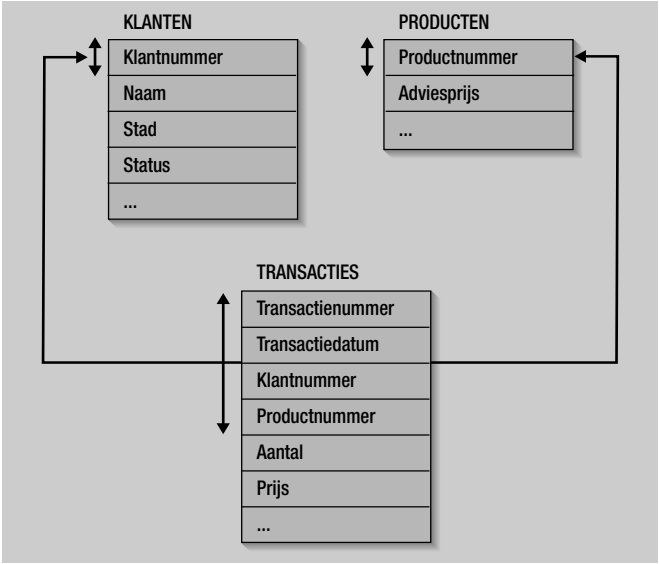
```
SELECT    PRODUCT_NR, ADVIES_PRIJS.BEDRAG
FROM      PRODUCTEN
WHERE     VALID ALWAYS ROWS 3
```

Toelichting: de specificatie VALID ALWAYS ROWS 3 geeft van elk product de laatste drie prijswaarden. Als de adviesprijs van een product nooit of slechts eenmaal is gewijzigd, verschijnt dat product niet in het resultaat.

Vraag 8: Geef voor elk product het productnummer en de voorlaatste prijs.

```
SELECT    PRODUCT_NR, ADVIES_PRIJS.BEDRAG
FROM      PRODUCTEN
WHERE     VALID ALWAYS ROWS 2
AND       NOT VALID NOW
```

Toelichting: door twee VALID-specificaties te combineren, kan deze vraag beantwoord worden.



Figuur 1: Structuur van de voorbeelddatabase.

Vraag 9: Geef van elk product het productnummer en de oudste prijs, plus de begin- en einddatum van die prijs.

```
SELECT   PRODUCT_NR, ADVIES_PRIJS.BEDRAG,
         ADVIES_PRIJS.OTS, ADVIES_PRIJS.OTS_END
FROM     PRODUCTEN AS P1
WHERE    ADVIES_PRIJS.OTS =
        (SELECT    MIN(P2.ADVIES_PRIJS.OTS)
         FROM      PRODUCTEN AS P2
         WHERE     P2.PRODUCT_NR = 1.PRODUCT_NR)
```

Toelichting: het opvragen van de oudste waarde gaat dus niet zo elegant, zoals blijkt uit de bovenstaande instructie. Hiervoor moeten we teruggrijpen naar bekende, maar wel lastige SQL-constructies.

Vraag 10: geef van elke transactie het unieke transactienummer, het nummer en de naam van de bijbehorende klant en de waarde van de STAD en SOORT-kolom op het moment van de transactie.

```
SELECT    T.TRANSACTIE_NR, T.TRANSACTIE_DATUM,
          T.KLANT_NR, K.NAAM, K.STAD.NAAM,
          K.KLANT_TYPE.TYPE_NR
FROM      TRANSACTIES AS T,
          KLANTEN AS K
WHERE     T.KLANT_NR = K.KLANT_NR
AND       VALID T.TRANSACTIE_DATUM
```

Toelichting: met de specificatie VALID T.TRANSACTIE_DATUM geven we aan dat naar de situatie van de KLANTEN-tabel gekeken moet worden die gold op het moment van de transactiedatum.

Vraag 11: Geef de omzet per stad ooit.

```
SELECT    K.STAD.NAAM, SUM(T.AANTAL * T.PRIJS)
FROM      KLANTEN AS K,
          TRANSACTIES AS T
```

```
WHERE     K.KLANT_NR = T.KLANT_NR
AND       T.TRANSACTIE_DATUM BETWEEN
             K.STAD.OTS AND K.STAD.OTS_END
GROUP BY  K.STAD.NAAM
```

Toelichting: met de conditie op de TRANSACTIE_DATUM-kolom zoeken we naar de stad waar de betreffende klant op die datum woonde.

Vraag 12: Geef de omzet per stad van dit jaar.

```
SELECT    K.STAD.NAAM, SUM(T.AANTAL * T.PRIJS)
FROM      KLANTEN AS K,
          TRANSACTIES AS T
WHERE     K.KLANT_NR = T.KLANT_NR
AND       TS_GET_YEAR(T.TRANSACTIE_DATUM) =
             TS_GET_YEAR(CURRENT_DATE)
AND       T.TRANSACTIE_DATUM BETWEEN
             K.STAD.OTS AND K.STAD.OTS_END
GROUP BY  K.STAD.NAAM
```

Vraag 13: Geef de som van het aantal verkochte producten (van alle transacties) vermenigvuldigd met de adviesprijs die gold op het moment van de transactie.

```
SELECT    SUM(T.AANTAL * P.ADVIES_PRIJS.BEDRAG)
FROM      TRANSACTIES AS T,
          PRODUCTEN AS P
WHERE     T.PRODUCT_NR = P.PRODUCT_NR
AND       T.TRANSACTIE_DATUM BETWEEN
            P.ADVIES_PRIJS.OTS AND
            P.ADVIES_PRIJS.OTS_END
```

Vraag 14: Geef voor elke stad de som van het aantal verkochte producten vermenigvuldigd met de op dat moment geldende adviesprijs van alle transacties.

```
SELECT    K.STAD.NAAM,
          SUM(T.AANTAL * P.ADVIES_PRIJS.BEDRAG)
FROM      KLANTEN AS K,
          TRANSACTIES AS T,
          PRODUCTEN AS P
WHERE     K.KLANT_NR = T.KLANT_NR
AND       T.PRODUCT_NR = P.PRODUCT_NR
AND       T.TRANSACTIE_DATUM BETWEEN
            P.ADVIES_PRIJS.OTS AND
            P.ADVIES_PRIJS.OTS_END
GROUP BY  K.STAD.NAAM
```

Let wel, niet elke vraag is eenvoudig af te handelen met deze tijdreeksen. Vragen waarbij waarden van een zelfde kolom uit verschillende perioden met elkaar vergeleken moeten worden, zijn lastig of helemaal niet te formuleren. Willen we bijvoorbeeld het verschil berekenen tussen de omzet van dit jaar en die van het vorige jaar, dan lukt dat bij RadidBase niet met één SELECT-instructie. Maar misschien moeten we ons afvragen of dit wel functionaliteit is die een SQL-product moet bieden? Is dit niet de taak van een tool voor business intelligence?

Oefening voor de lezer: ontwikkel de vergelijkbare databasestructuur voor een databaseserver die geen tijdreeksen ondersteunt en probeer vervolgens de bijbehorende SELECT-instructies te bedenken voor de bovenstaande vragen. Vergelijk daarna de complexiteit.

STIEFKIND

Uit al deze voorbeelden blijkt duidelijk dat een databaseserver die tijdreeksen ondersteunt het manipuleren van die gegevens sterk vereenvoudigt. Omdat datawarehouses vaak en veel historische gegevens bevatten, zouden we vooral in die omgevingen baat hebben bij deze eigenschap.

De huidige situatie is echter dat de meeste bekende databaseleveranciers weinig tot geen functionaliteit op dit gebied geïmplementeerd hebben. Ook in de nieuwe SQL3-standaard is geen functionaliteit voor tijdreeksen gedefinieerd. Het onderwerp tijdreeksen blijft een stiefkind van de databasewereld.

Ten slotte wil ik Antti Pesonen (VTT) en Frank Habers (Inergy Analytical Solutions) bedanken voor hun hulp bij het schrijven van dit artikel en het opbouwen van de voorbeelden. ●

Rick F. van der Lans (rick@r20.nl) is onafhankelijk consultant en redactieadviseur van Database Magazine.