Puzzelen met SQL

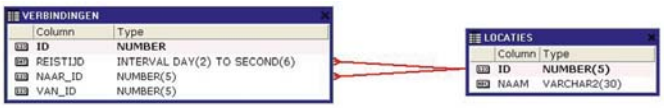
Vrije Tijd?

Hebben de kinderen van tegenwoordig het veel drukker dan wij vroeger? Is dit een teken dat wij 'op leeftijd' raken? Of is het daadwerkelijk zo? Als ik naar de buitenschoolse activiteiten van mijn eigen kinderen kijk, is het altijd een hele puzzel om ervoor te zorgen dat de kinderen op tijd bij de voetbal, atletiek, turnen en bij de dansles zijn. Ogenschijnlijk heeft mijn vrouw geen probleem om de kinderen bij de juiste activiteit en op tijd te krijgen. Voor mezelf, daarentegen, ik heb een uitgeschreven schema nodig. En dan nog blijft het altijd haasten.

In deze puzzel gaan we eens kijken of zo'n schema met behulp van SQL te maken is.

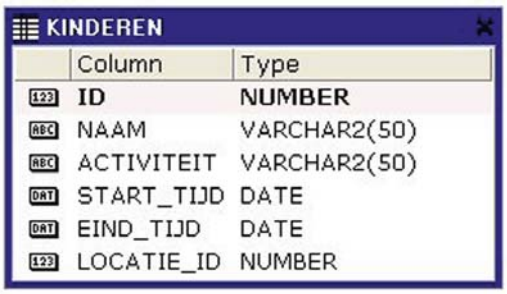
Het Datamodel

Om goed in kaart te kunnen brengen waar we allemaal naar toe moeten kunnen gaan en hoe lang het gaat duren om van A naar B te komen, maken we gebruik van een tweetal tabellen. De LOCATIES tabel waarin de verschillende locaties komen te staan, zoals de voetbal. De andere tabel betreft de VERBINDINGEN waarin de onderlinge relaties tussen de verschillende locaties staan.



Een bijzonder datatype dat we in de VERBINDINGEN tabel gebruiken, is die van de kolom REISTIJD. In deze kolom staat de tijdsduur, het datatype dat we hiervoor gebruiken is de INTERVAL. Van het INTERVAL datatype zijn twee smaken beschikbaar sinds Oracle 8.1.7., de INTERVAL DAY TO SECOND en de INTERVAL YEAR TO MONTH. Omdat we niet van plan zijn om bij te houden hoeveel jaren en maanden het gaat duren om van A naar B te reizen, maken we gebruik van de INTERVAL DAY TO SECOND.

Een andere tabel die we nodig hebben toont welk kind waar moet zijn op welk tijdstip en uiteraard wanneer je als ouder wordt geacht het kind weer op te halen.



Per kind kan er een activiteit inclusief begin- en eindtijd worden opgenomen in de tabel. Uiteraard ook waar het kind dan moet zijn.

Het vullen van de tabellen.

Na wat willekeurige straten in de LOCATIES tabel te hebben gezet, met wat recht-toe-recht-aan INSERT statements die we uit de tekst van deze puzzel hebben weggelaten, is het natuurlijk ook noodzakelijk om de verbindingen tussen de verschillende straten in de LOCATIES tabel te gaan zetten. Dan kunnen we er voor kiezen om ook hiervoor een heleboel INSERT statements te schrijven, maar het is natuurlijk handiger om een matrix te laten genereren van de verschillende straten. Naderhand kunnen we dan de reistijd tussen de verschillende straten invullen.
Door de LOCATIES tabel te joinen met zichzelf krijgen we een cartetisch product met teveel gegevens. Omdat het niet noodzakelijk is om van straat A naar straat A te reizen of van straat A naar straat B èn straat B naar straat A, wordt de join een non-equijoin.

```
SQL> select l1.id van_id
  2       , l2.id naar_id
  3    from locaties l1
  4    join locaties l2
  5      on (l1.id < l2.id)
  6  /

    VAN_ID    NAAR_ID
```

```
---------- ----------
         1          2
         1          3
         1          4
         2          3
         2          4
         3          4
```

Met behulp van deze query kunnen we nu de CONNECTIONS tabel gaan vullen.

De opdrachten

1) Maak een overzicht van alle mogelijke combinaties tussen de verschillende locaties, inclusief de reistijd.

Om deze vraag te kunnen beantwoorden maken we gebruik van de VERBINDINGEN en de LOCATIES tabellen. Omdat er een tweetal relaties tussen deze tabellen liggen, het VAN_ID en het NAAR_ID, moet de LOCATIES tabel twee maal gebruikt worden in de query.

```
SQL> select loc_van.naam "Van Naam"
  2       , loc_naar.naam "Naar Naam"
  3       , vbg.reistijd
  4    from verbindingen vbg
  5    join locaties loc_van
  6      on (vbg.van_id = loc_van.id)
  7    join locaties loc_naar
  8      on (vbg.naar_id = loc_naar.id)
  9  /

Van Naam        Naar Naam         REISTIJD
--------------- --------------- --------------------
Kruidenlaan     Vijfeikenweg    +00 00:10:00.000000
Vijfeikenweg    De Heuvel       +00 00:10:00.000000
Kruidenlaan     De Heuvel       +00 00:10:00.000000
De Heuvel       Statendamweg    +00 00:10:00.000000
Vijfeikenweg    Statendamweg    +00 00:15:00.000000
Kruidenlaan     Statendamweg    +00 00:05:00.000000
```

In bovenstaande query worden de tabellen met elkaar verbonden via de ANSI join syntax. Deze syntax wordt sinds Oracle 9i ondersteund. Het fraaie van deze ANSI join syntax is dat de join criteria (hoe de tabellen aan elkaar worden geknoopt) en de filter criteria (welk deel van de tabellen heb je nodig) gescheiden worden van elkaar. Bij de traditionele manier van tabellen joinen staan beide criteria door elkaar heen. Er is geen onderscheid meer tussen de join- en de filter criteria.
Iets anders wat misschien opvalt, is het gebruik van de dubbele aanhalingstekens in de kolom alias. Met behulp van deze syntax kun je de kolom naam (of althans de alias) hoofdletter gevoelig maken. Of bijvoorbeeld spaties opnemen in de kolom header zoals we hier hebben gedaan.

2) Wat is de route om de kinderen naar de verschillende activiteiten te brengen?

Om deze vraag te kunnen beantwoorden, moeten we eerst weten op welke locatie we moeten zijn en hoe laat. De gegevens over de activiteiten staan in de KINDEREN tabel. De begin- en de eindtijd zijn daar als aparte kolommen opgenomen. Deze begin- en eindtijden bepalen ook de route die afgelegd moet worden. Om de route overzichtelijk onder elkaar te krijgen willen we de begin- en eindtijd eigenlijk in één kolom hebben. Hiervoor kunnen we bijvoorbeeld de KINDEREN tabel met zichzelf joinen, maar we hebben niet voor niets een Oracle 11g database.

```
SQL> select id
  2       , naam
  3       , activiteit
  4       , locatie_id
  5       , actie
  6       , to_char (tijden, 'hh24:mi:ss') tijd
  7    from kinderen unpivot
  8    (tijden for actie in (start_tijd as 'brengen'
  9                          ,eind_tijd as 'halen'
 10                          )
 11    )
 12   order by tijden
 13  /

        ID NAAM            ACTIVITEIT      LOCATIE_ID ACTIE   TIJD
---------- --------------- --------------- ---------- ------- --------
         1 Tim             Voetbal                  2 brengen
14:45:00
         2 Lara            Dans                     3 brengen
14:55:00
         2 Lara            Dans                     3 halen
15:45:00
         1 Tim             Voetbal                  2 halen
16:00:00
```

In de bovenstaande query maken we gebruik van de UNPIVOT operator. Met de UNPIVOT operator kun je van kolommen rijen maken. Na de tabel in de FROM clause geef je aan dat je een UNPIVOT wilt uitvoeren. Vervolgens moet je aangeven voor welke kolommen je de UNPIVOT wilt uitvoeren. Het gevolg is dan dat het aantal rijen in de resultaat set vergroot wordt. Iedere begin- en eindtijd staat nu in dezelfde kolom, maar onder elkaar.

Als we nu toevoegen wanneer ze uit school komen, en wanneer ze verwacht worden bij het avondeten dan is het rondje compleet.
Voor de puzzel gaan we ervan uit dat we op Locatie met het ID nummer 1 wonen. Van vertrekpunt 1, moeten we naar locatie punt 2 om Tim naar de Voetbal te brengen. Vervolgens Lara bij

de Dans afzetten, even rust -misschien even tijd om boodschappen te doen- en Lara weer ophalen bij de Dans om via de Voetbal (nog steeds op locatie nummer 2) weer huiswaarts te keren.
De route zoals hierboven beschreven is te vangen in de volgende query:

```
SQL> select activiteit van_activiteit
  2       , locatie_id van_locatie
  3       , lead (activiteit) over (order by tijden) naar_activiteit
  4       , lead (locatie_id) over (order by tijden) naar_locatie
  5    from kinderen unpivot
  6         (tijden for actie in (start_tijd as 'brengen'
  7                               ,eind_tijd as 'halen'
  8                               ))
  9  order by tijden
 10  /

VAN_ACTIVITEIT  VAN_LOCATIE NAAR_ACTIVITEIT NAAR_LOCATIE
--------------- ----------- --------------- ------------
Uit School                1 Voetbal                    2
Voetbal                   2 Dans                       3
Dans                      3 Dans                       3
Dans                      3 Voetbal                    2
Voetbal                   2 Eten                       1
Eten                      1
```

De laatste regel kunnen we uit het resultaat wegfilteren, de gehele route is dan afgelegd. Ook de derde regel kunnen we uit de resultaat set weg laten, we blijven daar op dezelfde locatie. Om te bepalen wat de volgende 'stop' is, maken we gebruik van de analytische functie LEAD. Met de lead functie kun je 'vooruit' in de resultaat set kijken en daar waardes vanuit lezen. Als je bijvoorbeeld kijkt naar de eerste regel uit de resultaat set, dan zie je dat er Voetbal staat bij de NAAR_ACTIVITEIT. Deze waarde wordt 'gelezen' uit de tweede regel in de resultaat set uit de kolom VAN_ACTIVITEIT. Hetzelfde geldt voor de NAAR_LOCATIE. Met de LEAD functie is het ook mogelijk om meer dan één regel vooruit te lezen, maar in dit voorbeeld hebben we dat niet nodig.

De route die we af moeten leggen staat in bovenstaand overzicht. Om te kunnen bepalen wat de tijd is die nodig is om van de ene locatie naar de andere te komen, moeten we de VERBINDINGEN tabel gaan raadplegen.

```
SQL> select *
  2    from verbindingen
  3  /

        ID     VAN_ID     NAAR_ID REISTIJD
---------- ---------- ---------- --------------------
         1          1          2 +00 00:10:00.000000
         2          1          3 +00 00:10:00.000000
         3          1          4 +00 00:05:00.000000
         4          2          3 +00 00:10:00.000000
         5          2          4 +00 00:15:00.000000
         6          3          4 +00 00:10:00.000000
```

De tijd die nodig is om van punt A naar punt B te komen is gelijk aan de tijd die nodig is om van punt B naar punt A te komen. Om het zoeken naar de juiste tijd wat te vergemakkelijken kunnen we gebruik maken van oude bekende (en misschien zelfs vergeten) functies 'GREATEST' en 'LEAST'.

```
SQL> with van_naar_activiteit as
  2  (select activiteit van_activiteit
  3        , locatie_id van_locatie
  4        , lead (activiteit) over (order by tijden) naar_activiteit
  5        , lead (locatie_id) over (order by tijden) naar_locatie
  6        , lead (tijden) over (order by tijden) activiteit_start
  7     from kinderen unpivot
  8          (tijden for actie in (start_tijd as 'brengen'
  9                                ,eind_tijd as 'halen'
 10                                )
 11          )
 12    order by tijden)
 13  , x as
 14  (select van_activiteit
 15        , van_locatie
 16        , naar_activiteit
 17        , naar_locatie
 18        , activiteit_start
 19        , least (van_locatie, naar_locatie) van_locatie_tijd
 20        , greatest (van_locatie, naar_locatie) naar_locatie_tijd
 21     from van_naar_activiteit
 22    where naar_activiteit is not null
 23  )
 24  select x.van_activiteit
 25       , x.van_locatie
 26       , x.naar_activiteit
 27       , x.naar_locatie
 28       , to_char (x.activiteit_start, 'hh24:mi:ss') activiteit_start
 29       , vbg.reistijd
 30    from x
 31       , verbindingen vbg
 32   where vbg.van_id = x.van_locatie_tijd
 33     and vbg.naar_id = x.naar_locatie_tijd
 34  /

VAN_ACTIVITEIT  VAN_LOCATIE NAAR_ACTIVITEIT NAAR_LOCATIE ACTIVITE
REISTIJD
--------------- ----------- --------------- ------------ -------- -----
---------------
Uit School                1 Voetbal                    2 14:45:00
+00 00:10:00.000000
Voetbal
2 Dans                          3 14:55:00 +00 00:10:00.000000
Dans                            3 Voetbal                    2 16:00:00
+00 00:10:00.000000
Voetbal
2 Eten                          1 18:00:00 +00 00:10:00.000000
```

De queries die we al eerder hebben opgebouwd komen allemaal tezamen in bovenstaande query. De techniek die we hiervoor gebruiken heeft 'Subquery Factoring', ook wel de WITH-clause genoemd. Met Subquery Factoring kun je een subquery uit de FROM clause van het SELECT statement eruit halen en een naam geven. Op deze manier kun je stukje bij beetje je

SQL statement op bouwen. Deze techniek bestaat al sinds Oracle 9i, met de beperking dat de gedeclareerde Subquery ook daadwerkelijk gebruikt moest worden in het statement. Deze laatste beperking is uit Oracle 11g verdwenen.
Misschien is het je nog niet opgevallen, maar de reistijd tussen Voetbal en de Dans is tien minuten. De reistijd tussen deze twee locaties is ook tien minuten. Dit zou wel eens een probleem kunnen zijn. Om dit soort problemen wat inzichtelijker te maken kunnen we wederom gebruik maken van analytische functies. In de hoofdquery (het laaste deel van het gehele statement) voegen we toe:

```
case
   when activiteit_start - vbg.travel_time  <= lag (activiteit_start)
over (order by activiteit_start)
   then 'Misschien wel erg krap...'
end opmerkingen
```

In deze CASE expressie vergelijken we de starttijd van de huidige rij minus de reistijd die nodig is om op deze locatie te komen met de starttijd van de activiteit uit de voorgaande regel. Als het verschil tussen te klein is, dan laten we een opmerking zien.
En dan is dit het laatste stukje van de totale resultaat set:

```
... ACTIVITE REISTIJD                   OPMERKINGEN
... -------- ------------------------- --------
... 14:45:00 +00 00:10:00.000000
... 14:55:00 +00 00:10:00.000000        Misschien wel erg krap...
... 16:00:00 +00 00:10:00.000000
... 18:00:00 +00 00:10:00.000000
```

Nu hebben we alles wat we weten willen om de kinderen op tijd bij de buitenschoolse activiteiten te krijgen,... als nu de trainingen maar doorgaan want ik ben vergeten op de kalender te kijken.

Referenties

Oracle documentatie over INTERVAL datatype:
http://download-west.oracle.com/docs/cd/A87860_01/doc/server.817/a85397/sql_elem.htm#38072
Oracle documentatie over Subquery Factoring:
http://download.oracle.com/docs/cd/B19306_01/server.102/b14200/statements_10002.htm#SQLRF01702
AMIS Technology Blog over het maken van een route planner:
http://technology.amis.nl/blog/1221/building-a-route-planner-with-oracle-sql-finding-the-quickest-route-in-a-graph

Patrick Barel en **Alex Nuijten,** AMIS Services BV