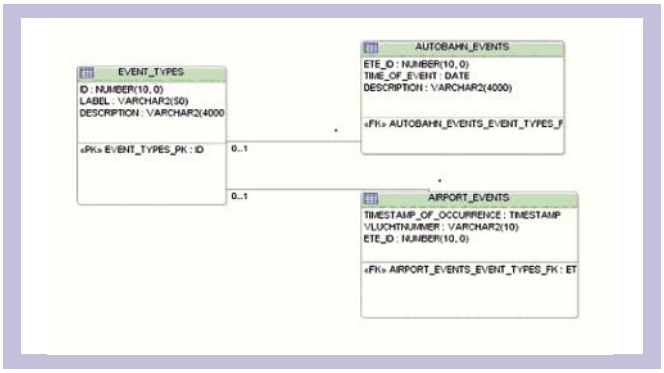
Puzzelen met SQL

Crash SQL Investigation

Deze puzzel is gebaseerd op het verhaal van Carel-Jan Engel tijdens een AMIS Query over High Availability. Hij vertelde hoe hij actief was bij het vliegveld van Dresden in een periode dat daar een belangwekkende zaak speelde: er waren in een jaar tijd meer dan tachtig ongelukken gebeurd op de snelweg naast het vliegveld. Dit was in de periode nadat er een viaduct over de snelweg was aangelegd waarover vliegtuigen van de gate naar de startbaan konden taxiën – bij vertrek – en van de landingsbaan naar de gate – bij aankomst.

Het verhaal dat al spoedig rondging, was dat de ongelukken dus wel het gevolg zouden zijn van het viaduct: automobilisten zouden door vliegtuigen op het viaduct worden afgeleid en vervolgens een ongeluk veroorzaken. Het klonk best aannemelijk – maar klopte het ook?

Het vliegveld registreert alle gebeurtenissen rondom ieder vliegtuig: het moment van landing wordt geregistreerd, het moment waarop het vliegtuig bij de gate staat, het moment waarop de brandstoftank vol is, het precieze tijdstip waarop het vliegtuig los komt van de gate en het exacte moment van opstijgen. Met al deze gegevens in de ene hand en de lijst van verdachte auto-ongelukken in de andere, moet er een serieus onderzoek kunnen worden gedaan naar het gerucht: is een substantieel percentage – meer dan toevallig genoemd kan worden – van de auto-ongelukken gebeurd in de periode dat een vliegtuig over het viaduct reed?



De Uitdagingen

In onze tabellen is een steekproef van een periode van twee weken opgenomen. Deze periode is representatief – zowel voor wat betreft het vliegveld als met betrekking tot de snelweg en de auto-ongelukken. Net als altijd beginnen we deze puzzelen met een eenvoudige vraag:

1) Hoeveel vluchten worden er per dag – voor de periode waarover we gegevens hebben – door het vliegveld van Dresden afgehandeld?

```
select trunc (aet.timestamp_of_occurrence )  day
     , ete.label
, count(*)
  from airport_events aet
  join event_types    ete
    on (ete.id = aet.ete_id)
 where ete.label in ('Touch Down', 'Take Off')
group by trunc (aet.timestamp_of_occurrence )
       , ete.label
;
DAY       LABEL
COUNT(*)
--------- -------------------------------------------------- ----------
02-OCT-06 Take Off
90
02-OCT-06 Touch Down
96
03-OCT-06 Touch Down
179
03-OCT-06 Take Off
183
04-OCT-06 Touch Down
155
04-OCT-06 Take Off
150
05-OCT-06 Touch Down
184
05-OCT-06 Take Off
187
06-OCT-06 Take Off
189
```

De airport_events tabel bevat een kolom waarin het tijdstip van een bepaalde gebeurtenis is opgenomen. Het datatype van deze kolom is een TIMESTAMP. Dit datatype is geintroduceerd in Oracle 8i en kan beschouwd worden als een uitbereiding binnen de DATE familie. Waar DATE een precisie heeft van een seconde, kan TIMESTAMP een granulariteit hebben tot nano-seconden. Er zijn nog wat andere bijzonderheden van dit datatype waar we dadelijk nog op terug zullen komen.
Om deze vraag correct te kunnen beantwoorden, moet je uiteraard rekening houden met het tijdselement wat standaard opgenomen is. Aangezien we nu het aantal vluchten per dag willen hebben, is het tijdselement niet van belang en moeten we gebruiken maken van TRUNC. In bovenstaande query wordt het aantal uitgaande en inkomende vluchten getoond per dag.

2) Wat is het drukste uur van de dag qua landingen? Hoeveel vluchten komen er dan per uur binnen?

```
select *
from ( select to_char( timestamp_of_occurrence, 'hh24' )
             , count(*)
       from airport_events aet
       join event_types    ete
         on (ete.id = aet.ete_id)
       where ete.label = 'Touch Down'
       group by to_char( aet.timestamp_of_occurrence, 'hh24' )
       order by count(*) desc
     )
where rownum = 1
TO   COUNT(*)
-- ----------
22         173
```

Om deze vraag te kunnen beantwoorden, moeten je het tijdselement van de TIMESTAMP_OF_OCCURENCE gaan bekijken. Omdat we nu alleen in het aantal landingen geïnteresseerd zijn, moet het Airport Event van het type Touch Down zijn.

In bovenstaande query hebben gebruik gemaakt van de traditonele manier om de tijd van een DATE te bekijken door het te converteren naar een CHAR middels de TO_CHAR functie. Een andere manier om een deel van het tijdselement uit een DATE of TIMESTAMP te halen is met behulp van de EXTRACT functie. Om het uur uit de TIMESTAMP_OF_OCCURRENCE kolom te halen met de EXTRACT functie, kun je de volgende syntax gebruiken:

```
extract (hour from aet.timestamp_of_occurrence)
```

Omdat Oracle altijd meerdere mogelijkheden biedt om hetzelfde vraagstuk op te lossen, zou je er ook voor kunnen kiezen om TRUNC te gebruiken. In dit geval zou je dan voor het tweede argument wat TRUNC aankan, een formaat masker, opgeven.

```
TRUNC (aet.timestamp_of_occurrence, 'hh24')
```

3) Hoe lang staan de vliegtuigen gemiddeld aan de gate? (dure tijd voor de luchtvaartmaatschappijen!)

Om te kunnen bepalen hoe veel tijd er aan de gate wordt doorgebracht, zullen we de tijd tussen het Event 'Off Gate' en 'At Gate' moeten vergelijken.
Dit kunnen we eenvoudig doen door gebruik te maken van de analytische functie LEAD. Laten we eens kijken naar een voorbeeld voor de vlucht met vluchtnummer RB461.

```
select aet.vluchtnummer
     , ete.label
     , aet.timestamp_of_occurrence
  from airport_events aet
  join event_types ete
    on (ete.id = aet.ete_id)
 where aet.vluchtnummer = 'RB461'
   and ete.label in ('At Gate', 'Off Gate')
/
VLUCHTNUMM LABEL                     TIMESTAMP_OF_OCCURRENCE
---------- ------------------------- ----------------------------
RB461      At Gate                    07-OCT-06 01.40.16.000000 PM
RB461      Off Gate                   07-OCT-06 02.58.30.000000 PM
```

Met de LAG-functie is het mogelijk om waarden uit andere rijen te lezen, zonder dat hiervoor een self-join nodig is.

```
lag (aet.timestamp_of_occurrence) over (partition by aet.vluchtnummer

order by aet.timestamp_of_occurrence
                                          )
```

Bij dit statement wordt ervan uitgegaan dat de gebeurtenissen chronologisch opvolgend zijn (de ORDER BY).
Het verschil tussen het aan de Gate komen en het vertrek is nu eenvoudig te bepalen door de LEAD kolom van de TIMESTAMP_OF_OCCURRENCE kolom af te trekken.

```
select vluchtnummer
     , time_at_gate
  from (
select aet.vluchtnummer
     , aet.timestamp_of_occurrence -
```

```
       lag (aet.timestamp_of_occurrence) over (partition by aet.vlucht-
nummer

order by aet.timestamp_of_occurrence
                                                    ) time_at_gate
  from airport_events aet
  join event_types ete
    on (ete.id = aet.ete_id)
 where aet.vluchtnummer = 'RB461'
   and ete.label in ('At Gate', 'Off Gate')
)
 where time_at_gate is not null;

VLUCHTNUMM TIME_AT_GATE
---------- --------------------------
RB461      +000000000 01:18:14.000000
```

Nu komt een van de eigenaardigheden van het TIMESTAMP datatype om de hoek kijken. Als je twee TIMESTAMPs van elkaar aftrekt is het resultaat een INTERVAL. Het resultaat van de bovenstaande query laat zien dat vlucht RB461 één uur, 18 minuten en 14 seconden aan de gate heeft doorgebracht. Een INTERVAL heeft als groot voordeel dat het een stuk eenvoudiger te lezen is dan het verschil in aantal dagen, wat het resultaat is als je twee DATEs van elkaar aftrekt. Er zijn twee smaken van Intervals, te weten het Year Month Interval en het Hour Second Interval. Het valt buiten de scope van dit artikel om hier uitgebreid op in te gaan. Hoewel het wel mogelijk is om intervals bij elkaar op te tellen en af te trekken, is het helaas niet mogelijk om met Intervals te aggregeren. Ook in Oracle 11g is dit nog niet mogelijk. Maar we zijn niet voor een gat te vangen. Door gebruik te maken van de Oracle Data Cartridge kunnen we zelf een Interval Aggregator bouwen. Omdat we waarschijnlijk niet de enigen in de wereld zijn die tegen dit probleem aanlopen, is het de moeite waard om allereerst eens te gaan Googlen. William Robertson heeft een AVG_DSINTERVAL en een SUM_DSINTERVAL gemaakt die wel kunnen aggregeren met een INTERVAL DAY TO SECOND datatype. Een link naar zijn website en de code vindt u onderaan dit artikel. Om met de Oracle Data Cartrigde aan de slag te gaan behoeft niets extra's te worden geïnstalleerd. Het is een kwestie van het implementeren van een door Oracle gestelde API, maar een volledige beschrijving hiervan valt ook buiten de scope van dit artikel.

```
select avg_dsinterval (time_at_gate) avg_time_at_gate
  from (select vluchtnummer
             , too - lag (too) over (partition by vluchtnummer
                                           order by too
                                    ) time_at_gate
          from (select aet.vluchtnummer
                     , ete.label
                     , aet.timestamp_of_occurrence too
                  from airport_events aet
                  join event_types ete
                    on (ete.id = aet.ete_id)
                 where aet.vluchtnummer in ('RB461'
                                           ,'CD160'
                                           ,'PK278'
                                           ,'SK633'
                                           ,'UY331'
                                           ,'BN417'
                                           ,'EB58'
                                           ,'CS468'
                                           ,'FN684'
                                           )
                   and ete.label in ('At Gate', 'Off Gate')
                )
         );


AVG_TIME_AT_GATE
-------------------------------
+000000000 01:14:09.888889000
```

Naast de Oracle Data Cartridge mogelijkheid, is er ook een andere methode om deze vraag te beantwoorden. Als bij een TIMESTAMP een INTERVAL wordt opgeteld is het resultaat nog steeds een TIMESTAMP. Als bij een TIMESTAMP een NUMBER wordt opgeteld is het resultaat een DATE. Deze truc gebruiken we in de volgende query:

```
select numtodsinterval( avg( ( 0 + too ) - ( 0 + lag_too ) ), 'day' )
from ( select ete.label
              , aet.timestamp_of_occurrence too
              , lag( timestamp_of_occurrence ) over ( partition by aet.
vluchtnummer

order by aet.timestamp_of_occurrence
                                                        ) lag_too
        from airport_events aet
        join event_types    ete
          on (ete.id = aet.ete_id)
      where ete.label in ('At Gate', 'Off Gate')
      )
where label = 'At Gate';
NUMTODSINTERVAL(AVG((0+TOO)-(0+LAG_TOO)),'DAY')
---------------------------------------------
+000000000 03:31:24.999999999
```

Het verschil tussen twee DATE is het aantal dagen ertussen, zoals eerder gezegd. Om dit te converteren naar een INTERVAL maken we gebruik van de NUMTODSINTERVAL functie. Deze functie zet een NUMBER om naar een INTERVAL DAY TO SECOND. Het tweede argument voor de NUMTODSINTERVAL functie is de interval unit. In ons geval geeft het getal het aantal dagen aan, dus is het tweede argument "DAY".

4) Worden vluchten met een oneven nummer sneller afgehandeld (tijd tussen landing en take-off) dan vluchten met even nummers?

Het bepalen van het verschil tussen landing en take-off kun je op dezelfde manier bepalen als in de derde opgave. Wij hebben in dit geval gekozen voor de laatst beschreven methodiek, een nummer bij de TIMESTAMP optellen zodat dit een impliciete data conversie oplevert. In deze query willen we graag de kracht van het CASE-statement laten zien. Om te bepalen of het een even of oneven vluchtnummer betreft hebben we het laatste karakter van het vluchtnummer nodig.
Je zou ervoor kunnen kiezen om een DECODE te gebruiken:

```
DECODE (substr (vluchtnummer, -1), '0', 'even'
                                  , '2', 'even'
                                  , '4', 'even'
                                  , '6', 'even'
                                  , '8', 'even'
                                  , 'oneven'
                                  )
```

Maar zoals je ziet levert dit nogal "omslachtige" code op.
Een vergelijkbaar CASE statement is veel eleganter en gemakkelijker te lezen:

```
case
  when substr( vluchtnummer, -1 ) in ( '0', '2', '4', '6', '8' )
  then 'even'
  else 'oneven'
end
```

De total query wordt dan:

```
select even_oneven
     , sum( ( 0 + too ) - ( 0 + lag_too ) )
from ( select ete_id
            , timestamp_of_occurrence too
            , lag( timestamp_of_occurrence ) over ( partition by
vluchtnummer

order by timestamp_of_occurrence ) lag_too

, case when substr( vluchtnummer, -1 ) in ( '0', '2', '4', '6', '8' )
                then 'even'
                else 'oneven'
             end even_oneven
       from airport_events
       where  ete_id in ( 1, 6 )
     )
where ete_id = 6
group by even_oneven
;
EVEN_O SUM((0+TOO)-(0+LAG_TOO))
------ ------------------------
oneven                79.5320949
even                  85.2151968
```

Het verschil tussen even- en oneven vluchten is dan ook minimaal

5) Selecteer alle vluchten op zondagen. Toon het vluchtnummer, het tijdstip van landen, van aan de gate arriveren, van vertrek bij de gate en van opstijgen. Sorteer de vluchten op de totale tijd die ze aan de grond zijn in Dresden.

Om een zondag voor alle database instellingen te vinden gebruiken we NLS_DATE_LANGUAGE in de query. Uiteraard gebruiken we dan ook het Duits als uitgangspunt, het gaat tenslotte om het vliegveld in Dresden.
Nu gebruiken we de lead-functie met een extra parameter die aangeeft hoeveel records we naar voren willen kijken, de default waarde 1 voor de aankomst bij de gate, 2 records voor het vertrek bij de gate of 3 records voor het opstijgen.

```
select vluchtnummer
     , landen
     , gate_aankomst
     , gate_vertrek
     , opstijgen
from ( select vluchtnummer
            , timestamp_of_occurrence landen
            , lead( timestamp_of_occurrence ) over ( partition by
vluchtnummer

order by timestamp_of_occurrence
                                                          ) gate_aankomst
            , lead( timestamp_of_occurrence, 2 ) over ( partition by
vluchtnummer

order by timestamp_of_occurrence
                                                          ) gate_vertrek
            , lead( timestamp_of_occurrence, 3 ) over ( partition by
vluchtnummer

order by timestamp_of_occurrence
                                                          ) opstijgen
            , ete_id
       from airport_events
      where ete_id in ( 1, 2, 5, 6 )
        and vluchtnummer = 'KY146'
     )
where ete_id = 1
  and to_char( landen, 'dy', 'NLS_DATE_LANGUAGE=GERMAN' ) = 'so'
order by opstijgen - landen

VLUCHTNUMMER :   KY146
LANDEN       :   15-OCT-06 04.03.20.000000 PM
GATE_AANKOMST:   15-OCT-06 04.13.46.000000 PM
GATE_VERTREK :   15-OCT-06 05.32.12.000000 PM
OPSTIJGEN    :   15-OCT-06 05.47.53.000000 PM
```

Note: Wegens het formaat van de pagina's is het resultaat van deze query onder elkaar weergegeven.

6) Hoeveel tijd per dag is er maximaal een vliegtuig boven de snelweg? Dat is de som van alle intervallen tussen de landing van iedere vlucht en het moment waarop het vliegtuig aan de gate staat plus de som van alle intervallen tussen het verlaten van de gate en het moment van opstijgen.

Gebruik makend van reeds genoemde technieken in voorgaande opgaven, komen we tot deze query:

```
select min( sum( 0 + gate_aankomst - ( 0 + landen ) + ( 0 + opstijgen
) - ( 0 + gate_vertrek ) ) ) minimale_tijd

, max( sum( 0 + gate_aankomst - ( 0 + landen ) + ( 0 + opstijgen ) - (
0 + gate_vertrek ) ) ) maximale_tijd
from ( select vluchtnummer
             , timestamp_of_occurrence landen
             , lead( timestamp_of_occurrence ) over ( partition by
vluchtnummer

order by timestamp_of_occurrence
                                                    ) gate_aankomst
             , lead( timestamp_of_occurrence, 2 ) over ( partition by
vluchtnummer

order by timestamp_of_occurrence
                                                    ) gate_vertrek
             , lead( timestamp_of_occurrence, 3 ) over ( partition by
vluchtnummer

order by timestamp_of_occurrence
                                                        ) opstijgen
             , ete_id
       from airport_events
       where ete_id in ( 1, 2, 5, 6 )
     )
where ete_id = 1
group by trunc( landen )
;
MINIMALE_TIJD       MAXIMALE_TIJD
------------------- ----------------
1,66829861111111    3,78208333333333
```

Dus per dag is er maximaal "3,78 dag" een vliegtuig in de buurt van het viaduct. Dit rare getal komt natuurlijk doordat er meerder vliegtuigen tegelijk in bezig zijn met landen, taxiën en opstijgen, maar het geeft wel aan dat er eigenlijk altijd wel een vliegtuig boven de snelweg is te vinden. Het zou dus goed mogelijk zijn dat de taxiënde vliegtuigen oorzaak zijn van de ongevallen die rondom het vliegveld van Dresden plaatsvinden.

7) Het viaduct kan alleen de aanleiding zijn van autoongelukken als de ongelukken gebeuren tijdens of kort na het overrijden van een vliegtuig. We gaan dus op zoek naar het percentage van het totaal aantal auto-ongelukken dat plaatsvond binnen vijf minuten na het over het viaduct taxiën van een vliegtuig.

```
select count( ( select max( ape.timestamp_of_occurrence )
                from airport_events ape
                where ape.ete_id in ( 1, 2, 5, 6 )

and   ape.timestamp_of_occurrence between abe.time_of_event - numtods-
interval( 5, 'minute' )

and abe.time_of_event
                )
            ) / count(*) * 100 pct
from autobahn_events abe ;
         PCT
----------
67.7419355
```

Uit de steekproef die we hebben genomen, de twee weken data, mag blijken dat de taxiënde vliegtuigen over het viaduct bij Dresden niet direct als oorzaak kunnen worden aangemerkt. Ook in het echte onderzoek wat plaatsvondt, was de conclusie dat het vliegtuig viaduct niet de oorzaak was van de ongevallen die rondom het vliegveld van Dresden plaatsvonden.
De oorzaak werd uiteindelijk wel gevonden. Het had te maken met een te korte invoegstrook. De sourcecode is vanaf de Technology Blog van AMIS te downloaden. Het web adres is: http://technology.amis.nl/blog/?p=2830
De titel is afgeleid van het National Geographic Channel programma Crash Scene Investigation waarin onderzoek naar vliegtuigongelukken centraal staan.

Links:

- Aggregate for Interval Day to Second: www.williamrobertson.net
- Oracle Data Cartridge: http://download.oracle.com/docs/cd/B19306_01/appdev.102/b14289/dciwhatis.htm#sthref14

Anton Scheffer en Alex Nuijten, AMIS.