ABIS NIEUWSBRIEF VOOR DB2 PROFESSIONALS

JAARGANG 6 - NUMMER 3, JANUARI 2010

OPEN CURSOR

Voor u ligt het laatste nummer van "Exploring DB2" dat in papieren versie wordt uitgebracht. Vanaf de volgende jaargang stappen we over op een milieuvriendelijker alternatief: een PDF-versie die u van de ABIS-site kunt downloaden.

Trouwens, voor wie het nog niet had gemerkt: die elektronische versie bestaat al enige tijd, en onze website laat u bovendien toe, alle vorige nummers te raadplegen.

Voor wie dit nummer op de trein aan het lezen is: geniet nog een laatste maal van de papieren versie, en kopieer volgende keer "Exploring DB2" naar uw laptop vooraleer op de trein te stappen!

Het ABIS DB2-team.

IN DIT NUMMER:

- *Standaard SQL: scalaire functies*, een eerste artikel in een reeks over de verschillen tussen relationele databases voor wat betreft benaming en gebruik van scalaire functies. In deze bijdrage worden de verschillen met de SQL-standaard besproken.
- *Explain: oude en nieuwe mogelijkheden*, geeft een kort overzicht van het gebruik van EXPLAIN in DB2 for z/OS. Als nieuwe mogelijkheid wordt in het bijzonder toegelicht hoe u optimisatie-info verkrijgt over het gebruik van nog niet bestaande indexen.
- En tenslotte, zoals ondertussen gebruikelijk, een bijdrage i.v.m. "nieuw in DB2 9 voor z/OS" met als titel "*Waarom u nieuwe DB2 9 SQL echt nodig hebt!*"

3

CLOSE CURSOR

In het volgende (elektronische) nummer zullen we het hebben over persistentiemogelijkheden tussen Java en DB2, en zal ook een vergelijking gemaakt worden tussen de mogelijkheden van DB2 en Oracle voor wat betreft scalaire functies.

Standaard SQL: scalaire functies

Steven Scheldeman (ABIS)

SQL is de standaardtaal om een RDBMS te benaderen. We weten allemaal dat de verschillende platformen (DB2, Oracle, SQL Server, MySQL, ...) hun eigen variant van de standaard SQL hanteren. Ook al zegt men wel eens dat de SQL zoals die binnen DB2 gehanteerd wordt, in feite overeenkomt met standaard SQL, in werkelijkheid blijken er toch afwijkingen te bestaan tussen beide, vooral dan wat betreft het aanbod aan SQL-**functies**. Vandaar deze bijdrage, een eerste in een reeks, die een poging doet om orde te scheppen in de "chaos" van (scalaire) functies met hun toch-niet-zo-standaard varianten.

Al bestaan er grote verschillen tussen het aanbod aan functies in b.v. DB2 en Oracle, we merken wel dat de verschillende RDBMS-platformen – onder druk van de gebruiker– steeds meer naar elkaar toe zijn geëvolueerd. Dit is vooral zichtbaar bij de functies, zowel de "scalar functions", als de "aggregate functions". Op zich is dit niet verwonderlijk: wanneer de ene RDBMS een nieuwe functionaliteit inbouwt, is het te verwachten dat de andere deze vroeg of laat ook zal opnemen. En niet alleen worden functionaliteiten overgenomen, ook de syntax wordt meer en meer op elkaar afgestemd.

Deze observatie was de aanleiding voor deze artikelreeks, waarvan deze eerste bijdrage de scalaire functies van standaard SQL vergelijkt met hun overeenkomstige varianten binnen DB2, Oracle en SQL Server. In tweede instantie zullen we een vergelijking maken tussen de platformen voor wat betreft de niet-standaard SQL-functies.

Als uitgangspunt zijn volgende versies genomen: DB2 for z/OS v9.1, DB2 for LUW v9.7, Oracle v10.2, SQL Server 2008 en MySQL v5.0.

Online documentatie kan via de volgende links gevonden worden:

- Standaard SQL:
  - *http://www.wiscorp.com/sql_2003_standard.zip*
  - *http://www.wiscorp.com/sql200n.zip*
- DB2 (z/OS en LUW resp.):
  - *http://publib.boulder.ibm.com/infocenter/dbzichelp/*
  - *http://publib.boulder.ibm.com/infocenter/db2luw/v9*
- Oracle:
  - *http://otn.oracle.com/documentation*
  - *http://download.oracle.com/docs/cd/B19306_01/server.102/b14200.pdf*
- SQL Server:
  - *http://msdn.microsoft.com/en-us/library/*
  - *http://msdn.microsoft.com/en-us/library/bb510741.aspx*

- MySQL:
  - http://dev.mysql.com/doc/
  - http://dev.mysql.com/doc/refman/5.0/en/functions.html

In wat volgt overlopen we alle scalaire functies waarvoor een ANSI- en ISO-standaard is vastgelegd, met telkens de manier waarop DB2, Oracle en SQL Server diezelfde functionaliteit implementeert. We hebben een poging gedaan de functies logisch te groeperen.

In de specificaties wordt herhaaldelijk `<char length units>` gebruikt. Volgens de ISO-standaard kan dat gelijk zijn aan hetzij CHARACTERS hetzij OCTETS (d.w.z.: bytes). Zo is de lengte van het woord "cliché" gelijk aan 6 characters, maar (bij encodering in UTF-8) is de octet-lengtegelijk aan 7 bytes omdat UTF-8 voor accentletters 2 bytes nodig heeft. De default is "CHARACTERS".
DB2 gebruikt niet-standaard waarden voor `<char length units>`, nl. `OCTETS`, `CODEUNITS32` voor "characters", en `CODEUNITS16` voor 2-byte characters, b.v. bij encodering met UTF-16.

Tekst-manipulatie-functies: lengte van een string

```
(I)    CHAR_LENGTH(<string expression> [USING <char length units>])
(I,D)  CHARACTER_LENGTH(<string expression> [USING <char length units>])
(M)    CHAR_LENGTH(<string expression>)
(M)    CHARACTER_LENGTH(<string expressions>)
(I,M)  OCTET_LENGTH(<string expression>)
(O,M)  LENGTH(<string expression>)
(O)    LENGTHB(<string expression>)
(S)    LEN(<string expression>)
(S)    DATALENGTH(<string expression>)
(M)    BIT_LENGTH(<string expression>)
(I: ISO/ANSI, D: DB2, O: Oracle, S: SQL Server, M: MySQL)
```

Element `<string expression>` stelt de te doorzoeken tekst voor. De functies OCTET_LENGTH, `LENGTHB` en `DATALENGTH` laten binaire argumenten toe. Bovendien is `DATALENGTH` de enige functie uit de lijst waarbij het argument ook van een ander datatype kan zijn (zonder dat het eerst naar tekst wordt omgezet): `DATALENGTH` geeft steeds de byte-lengte van de interne voorstelling van z'n argument.

Tekst-manipulatie-functies: gedeelte van een string

```
(I)    SUBSTRING(<string expression> FROM <start position>
               [FOR <string length>] [USING <char length units>])
(M)    SUBSTRING(<string expression> FROM <start position>
               [FOR <string length>])
(D)    SUBSTRING(<string expression>, <start position>
               [, <string length>] [, <char length units>])
(O,M)  SUBSTR(<string expression>, <start position> [, <string length>])
(S,M)  SUBSTRING(<char expression>, <start position>, <string length>)
```

Oracle heeft hiernaast functies als SUBSTRB, SUBSTRC, ... om de afwezigheid van een argument `<char length units>` te compenseren.

Bij SQL Server is `<string length>` niet optioneel, maar dit mag wel de maximale lengte van `<char expression>` overschrijden.

Tekst-manipulatie-functies: beginpositie in een string

```
(I)    POSITION(<search string> IN <source string> [USING <char length units>])
(M)    POSITION(<search string> IN <source string>)
(D)    POSITION(<search string>,<source string>,<char length units>)
(O,M)  INSTR(<source string>,<search string>[, <start pos> [, <occurence>]])
(S)    CHARINDEX(<search string>,<source string>[, <start pos>])
```

Oracle's INSTR laat toe om zowel voorwaarts als achterwaarts (indien `<start pos>` negatief is) in een string te zoeken, pas te beginnen zoeken vanaf een bepaalde positie, en zelfs te bepalen in het hoeveelste voorkomen van de gezochte string je geïnteresseerd bent (door het geheel getal `<occurence>` op te geven). MySQL kent dezelfde functie INSTR, maar met slechts twee argumenten.

Merk ook op dat we bij Oracle, in tegenstelling tot de standaard, eerst de string vermelden waarin we zoeken, en dan pas de string waar we naar zoeken.

Ook bij de SQL Server-functie `CHARINDEX` kan een startpositie opgegeven worden; indien echter `<start pos>` negatief is of gelijk aan 0, dan wordt 1 genomen.

Tekst-manipulatie: vertaling naar hoofd-/kleine letters

```
(I,O,S,M) UPPER(<char expression>)
(I,O,S,M) LOWER(<char expression>)
(D)        UPPER(<char expression>[,<locale name>][,<integer expression>])
(D)        LOWER(<char expression>[,<locale name>][,<integer expression>])
```

Hierbij is `<locale name>` voornamelijk bedoeld om de UNICODE-characters te ondersteunen en `<integer expression>` om het resultaatveld groot genoeg te maken om bij omzetting de uitbreiding van de UNICODE-vertaling op te vangen

Tekst-manipulatie: weghalen van leading/trailing (blanks)

```
(I,O,M)   TRIM([[<trim specification>] [<what>] FROM] <char expression>)
(D)        STRIP(<char expression> [, <trim specification> [, <what>]])
(S,D,O,M) LTRIM(<char expression>)
(S,D,O,M) RTRIM(<char expression>)
```

Hierbij kan `<trim specification>` één van de waarden `LEADING`, `TRAILING` of `BOTH` aannemen. Bij DB2 kunnen deze nog afgekort worden tot L, T of B. De expressie `<what>`, spatie "by default", geeft aan wat moet weggehaald worden.
De functie `LTRIM` komt overeen met "`LEADING`", `RTRIM` met "`TRAILING`".

SQL Server beschikt enkel over de RTRIM en LTRIM functies, waarbij respectievelijk langs links of langs rechts de blanco's verwijderd worden. Deze twee functies vinden we ook terug bij DB2, Oracle en MySQL maar niet in de standaard.

Tekst-manipulatie: "vertalen" van characters

De characters van de gegeven tekst worden omgezet naar de overeenkomstige tekens van een gegeven set.

```
(I)    TRANSLATE(<char expression> USING <transliteration name>)
(D)    TRANSLATE(<char expression>[,<resulting chars>[,<source chars>[,<pad char>]]])
(O)    TRANSLATE(<char expression>, <source chars>, <resulting chars>)
(O)    TRANSLATE(<char expression> USING CHAR_CS)
(O)    TRANSLATE(<char expression> USING CHAR_NCS)
(S,M)  REPLACE(<char expression>,<source char expr>,<replacement char expr>)
```

DB2 heeft een `TRANSLATE`-functie die iets anders werkt dan de standaard: alle characters uit `<source chars>` worden omgezet in de characters van de `<resulting chars>` op een één per één basis, in gelijke volgorde. Ontbrekende characters in `<resulting chars>` worden vervangen door het `<pad char>`.

Oracle heeft beide versies van de `TRANSLATE`-functie.

SQL Server heeft geen `TRANSLATE`-functie. Er bestaat wel een `REPLACE`-functie. Deze kan gebruikt worden als de `TRANSLATE`-functie van DB2, op voorwaarde dat men het doet voor één letter per keer.

De REPLACE-functie van MySQL laat toe om alle voorkomens van een gegeven tekst (tweede argument, 1 of meer chars) te vervangen door een andere tekst (derde argument).

Tekst-manipulatie: overschrijven van (deel van) string.

```
(I)    OVERLAY(<original expr> PLACING <new expr> FROM <start position>
               [FOR <string length>] [USING <char length units>])
(D)    OVERLAY(<original expr>, <new expr>, <start position>
               [, <string length>], <char length units>)
(M)    INSERT(<original expr>, <start position>, <string length>, <new expr>)
```

Noch Oracle, noch SQL Server kent een OVERLAY functie. Dit kan gesimuleerd worden door een combinatie van de SUBSTR (voor Oracle) of SUBSTRING (voor SQL Server) functie met string concatenatie. Bij MySQL heet deze functie INSERT.

Tekst-manipulatie: omzetten van characters naar andere

```
(I)    CONVERT(<character expression> USING <transcoding name>)
(O)    CONVERT(<character expression>,<transcoding name>[,<source code name>])
```

DB2 heeft geen CONVERT-functie.

SQL Server heeft wel een CONVERT, maar deze is eerder een CAST functie, en is dus niet equivalent met de CONVERT van ISO/ANSI

Tekst-manipulatie: maak genormaliseerde UNICODE

```
(I)    NORMALIZE(<character expression [,<normal form> [,<result length>]])
(D)    NORMALIZE_STRING(<char expression>[,<normal form>[,<result length>]])
```

Hierbij kan <normal form> één van de waarden NFC, NFD, NFKC of NFKD zijn.

Oracle kent de NORMALIZE-functie niet, maar vervangt deze door COMPOSE. Deze laatste werkt echter op basis van een concatenatie van characters met UNICODE-symbolen als argument.

SQL Server en MySQL kennen de NORMALIZE-functie evenmin.

Datum-manipulatie-functie: EXTRACT

Deze functie toont een deel van een datum, een tijd of een interval.

```
(I,D,O,M)    EXTRACT(<extract field> FROM <extract source>)
(S)          DATEPART(<extract field>, <extract source>)
```

Het veld <extract source> moet een geldige expressie van datatype <time>, <date>, <timestamp> of <interval> zijn,
dus b.v. CURRENT_TIME, CURRENT_TIMESTAMP, ...

<extract field>-s kunnen zijn:

- (ISO/ANSI) SECOND, MINUTE, HOUR, DAY, MONTH, YEAR, TIMEZONE_HOUR, TIMEZONE_MINUTE
- (DB2) SECOND, MINUTE, HOUR, DAY, MONTH en YEAR
- (Oracle) zoals ISO, en bovendien TIMEZONE_REGION en TIMEZONE_ABBR
- (MySQL) zoals DB2, en bovendien MICROSECOND, QUARTER, en combinaties zoals YEAR_MONTH, DAY_MINUTE, enz.
- (SQL Server) year, yy, yyyy, quarter, qq, q, month, mm, m, dayofyear, dy, y, day, dd, d, week, wk, ww, weekday, dw, hour, hh, minute, mi, n, second, ss, s, millisecond, ms, microsecond, mcs, nanosecond, ns, TZoffset, tz, ISO_WEEK, isowk, isoww

Numerieke functies

De meeste platformen hebben een zeer groot aanbod aan numerieke functies. De ANSI/ISO-standaard heeft hiervan slechts enkele gestandaardiseerd: zie hieronder.

Numerieke functies: absolute waarde

```
(I,D,O,S,M)  ABS(<numeric expression>)
```

Numerieke functies: grootste geheel getal niet groter dan

```
(I,D,O,S,M)  FLOOR(<numeric expression>)
```

Numerieke functies: kleinste geheel getal niet kleiner dan

```
(I,D,S,M)    CEILING(<numeric expression>)
(I,O,M)      CEIL(<numeric expression>)
```

Numerieke functies: vierkantswortel

```
(I,D,O,S,M)   SQRT(<numeric value expression>)
```

Numerieke functies: rest bij gehele deling

```
(I,D,O,M)     MOD(<numeric expression dividend>, <numeric expression divisor>)
(S,M)         <numeric expression dividend> % <numeric expression divisor>
```

Numerieke functies: natuurlijk logaritme

```
(I,D,O,M)     LN(<numeric expression>)
(S,M)         LOG(<numeric expression>)
```

Numerieke functies: e tot de vermelde macht

```
(I,D,O,S,M)   EXP(<numeric expression>)
```

Numerieke functies: machtsverheffing

```
(I,D,O,S,M)   POWER(<numeric expression base>, <numeric expression exponent>)
```

Algemene functie: omzetting naar een specifiek datatype

```
(I,D,O,S,M)   CAST(<value expression> AS <data type>)
```

De CAST-functie was de eerste ISO/ANSI-functie met meer dan één argument (dus met een woordelijke argumentscheiding i.p.v. een komma) die door alle relationele databases ondersteund werd (en wordt). De meeste platformen hebben nog steeds, als alternatief voor CAST, hun “oude” manier om een expressie expliciet naar een ander datatype te converteren. Zo heeft DB2 bij voorbeeld voor elk datatype een functie met de naam van het datatype, met één (of soms meer) argument(en), zoals bij voorbeeld:

```
(D)           CHAR(<date value expression>, EUR)
(D)           CHAR(<text value expression>, 20)
```

Besluit

De ANSI/ISO SQL-standaarden SQL:2003, SQL:2006 en SQL:2008 doen een schuchtere poging om scalaire functies te standaardiseren. Het lijstje van gestandaardiseerde functies is relatief kort, vergeleken met de lijsten van beschikbare scalaire functies op elk relationeel platform apart.

Ook de platformen doen schuchtere pogingen om de standaardvorm van deze functies te ondersteunen, meestal naast een alternatieve, eigen vorm die niet consistent is, noch met de standaard, noch met andere platformen. Hierover meer in een volgende bijdrage.

EXPLAIN: oude en nieuwe mogelijkheden

Peter Vanroose (ABIS)

Explain: de “goeie ouwe” manier

Ook DB2 for z/OS ontsnapt niet aan de tendens naar meer visuele, grafische gebruikers-interfaces. Zeker voor het analyseren van access paths is dat inderdaad een nuttige evolutie. DB2 levert hiertoe de (gratis te downloaden) MS-Windows-applicaties Visual Explain en z'n opvolger, OSC (“Optimisation Service Center”).

Toch is het niet altijd mogelijk of wenselijk om een MS-Windows-applicatie te laten connecteren naar DB2 op z/OS. Gelukkig blijft de informatie die de optimizer ons via EXPLAIN meedeelt (voorlopig toch) nog altijd toegankelijk op de klassieke manier, nl. als rijen van de tabel PLAN_TABLE.

Geef het SQL-statement EXPLAIN als argument de query waarover men access-path-informatie wil; de optimizer plaatst dan in uw PLAN_TABLE één rij per tabel uit de query, met een rijke (zij het onvolledige) hoeveelheid aan details i.v.m. met het optimale “stappenplan” dat door de data manager zal gevolgd worden om de query te implementeren.

Hier volgt een kort overzichtje van de belangrijkste kolommen van PLAN_TABLE, en hoe de waarden in deze kolommen te interpreteren:

TNAME en CREATOR: de naam (en schemanaam) van de tabel waarover op deze lijn van PLAN_TABLE informatie gegeven wordt.

ACCESSTYPE: toegang tot deze tabel via hetzij een tablescan (R), via een index (I, N of I1), of m.b.v. meerdere indexen tegelijk (M).

ACCESSNAME: in geval van index-toegang: naam van die index.

INDEXONLY: Y indien na het gebruik van een index geen tablespace-toegang meer gebeurt.

MATCHCOLS: in geval van index-toegang: is dit een “matching index access”, en zo ja: op hoeveel kolommen van de index?

METHOD: 1, 2 of 4 indien deze tabel betrokken is bij een join als tweede of volgende tabel. De drie waarden komen overeen met de verschillende join-methodes (nested loop, merge scan, hybrid). METHOD is 3 voor de “pseudo-tabel” (naamloos) die nodig is bij materialisatie (voor een ORDER BY, een GROUP BY, of een DISTINCT). In alle andere gevallen (b.v. de eerste tabel van een join) is METHOD=0.

QBLOCK_TYPE, QBLOCKNO en PARENT_QBLOCKNO: in geval van een nested query die dus uit meerdere query blocks bestaat: type en volgnummer van het query block dat hier gedocumenteerd wordt. QBLOCKTYPE kan o.a. zijn: SELECT (de “hoofd”query), UNION, NCOSUB (niet-gecorreleerde subquery), CORSUB of TABLEX (table expr.)

Is bijkomende sortering nodig, en zo ja om welke reden? Er zijn 5 kolommen die telkens Y of N kunnen zijn: SORTC_ORDERBY, SORTC_GROUPBY, SORTC_UNIQ (omwille van een expliciete of impliciete “SELECT DISTINCT”), SORTC_JOIN, en SORTN_JOIN.

PREFETCH: S indien sequentiële prefetch gepland is, L indien list prefetch zal gebeuren (met sorteren van de RID list), D bij dynamische prefetch, en blanco indien er geen prefetch zal gebeuren.

TABLE_TYPE: indien deze rij informatie bevat over een “tussenresultaat”-tabel, vertelt deze kolom over welk soort tussentabel het gaat; b.v. S voor subquery, C voor common table expression (CTE), W voor work file, M voor MQT, ... Voor een gewone tabel staat hier “T”.

Daarnaast bevat PLAN_TABLE nog informatie over locking, over CPU, sysplex of I/O parallellisme, en kolommen die het mogelijk maken om optimisation hints door te geven aan de optimizer.

Een voorbeeldje:

```
    EXPLAIN PLAN SET QUERYNO=3000 FOR
      WITH C (name) AS (SELECT DISTINCT first_name FROM persons)
      SELECT name, COUNT(*)  FROM C    GROUP BY name
```

In de tabel PLAN_TABLE van de persoon die deze EXPAIN uitvoert, worden 3 rijen teruggevonden. De volgende query laat toe om de meest relevant informatie op een compacte manier te tonen:

```
SELECT  substr(tname,1,10) AS tname, accesstype AS ac, substr(accessname,1,10),
        matchcols AS mc, indexonly AS io, method, qblock_type, prefetch AS pf,
     sortc_uniq||sortc_orderby||sortc_groupby||sortc_join||sortn_join AS UOGCN
  FROM  PLAN_TABLE
 WHERE  queryno = 3000
ORDER BY timestamp, qblockno, planno
```

Deze query geeft b.v. volgende rijen terug (indien er een index gedefinieerd is op de kolom first_name van tabel persons):

```
TNAME     AC              MC  IO METHOD QBLOCK_TYPE  PF  UOGNC  QBLOCKNO
C         R                0  N       0  SELECT       S   NNNNN         1
                           0  N       3  SELECT           NNYNN         1
PERSONS   I   IX_FIRSTNA   0  Y       0  TABLEX       S   NNNNN         2
```

Hieruit lezen we o.a. af dat de CTE herschreven is naar een “gewone” nested table expression, die de informatie uit PERSONS-tabel via een index-only non-matching index scan ophaalt. Bijkomende sortering is nodig omwille van GROUP BY. Sequentiële prefetch gebeurt voor zowel de index als daarna voor de table scan van de tussentabel C.

Visual Explain en OSC

Deze twee “front ends” voor explain geven een grafische voorstelling van de informatie die we ook al in PLAN_TABLE konden aflezen. Het zijn Windows-applicaties die achter de schermen gewoon de “goeie ouwe” EXPLAIN en de PLAN_TABLE gebruiken. Toch zien we op het scherm nuttige informatie verschijnen die we niet in de PLAN_TABLE waren tegengekomen. Zo is b.v. statistische info zicht-

baar (grootte van de tabellen, cluster ratio van de index, ...) en ook een geschatte kostberekening (CPU). In OSC wordt o.a. ook nog verteld als er nog stage-2-filtering gebeurt of niet.

Hoe komt OSC aan deze informatie? Statistische gegevens worden uiteraard uit de catalog gehaald. Kostprijs-schattingen werden door EXPLAIN weggeschreven naar de tabel DSN_STATEMNT_TABLE. OSC zorgt er bij eerste uitvoering voor dat deze tabel gecreëerd wordt indien nog niet aanwezig.

Bijkomende informatie voor OSC wordt eveneens door EXPLAIN gegenereerd en naar nog een aantal andere, recenter toegevoegde tabellen weggeschreven. De tabel DSN_STATEMNT_TABLE kent u wellicht al: hierin vindt u de CPU-kost-schatting van uw query. DB2 9 voor z/OS heeft een aantal bijkomende explain-tabellen, die OSC voor u aanmaakt indien ze nog niet bestaan.

Nieuwe explain-tabellen

Deze nieuwe tabellen zijn (bewust) niet uitgebreid gedocumenteerd: zo bevat de SQL Reference Guide (SC18-9854) enkel informatie over de twee reeds genoemde tabellen en de tabel DSN_FUNCTION_TABLE (die het gebruik van user-defined functions in uw query documenteert).

De "Performance Monitoring en Tuning Guide" (SC18-9851) bevat gedetailleerdere informatie over deze drie tabellen, en eveneens (in hoofdstuk 24) de nieuwe tabellen, met details over hun kolomnamen en betekenis.

Kort samengevat:

`DSN_STRUCT_TABLE`: explain-info per query block, zoals het verwacht aantal rijen dat door dit blok gegenereerd wordt, en een schatting van het aantal keren dat dit query block uitgevoerd zal worden.

`DSN_VIEWREF_TABLE`: overzicht van alle views en MQTs in de query.

`DSN_FILTER_TABLE` en i.h.b. z'n kolom `STAGE` bevat informatie, per "WHERE"-predikaat, over de manier waarop de filtering hiervoor gebeurt: stage-1 (matching in index, screening in index, of in tablespace), dan wel stage-2. Deze informatie moet gecombineerd worden met die in de tabel `DSN_PREDICAT_TABLE`, waarin de predikaten zelf beschreven worden (met hun filter factor, query block number, en hun inhoud, mooi uiteengerafeld in samenstellende delen).

`DSN_DETCOST_TABLE`:: gedetailleerd kostenschatting, inclusief filter factoren (matching en screening), I/O- en CPU-kosten voor verschillende stappen van het access path, en aantal pages, rijen en kolommen dat verwerkt wordt: index leaf pages, rijen na stage-1, rijen na stage-2, aantal te sorteren rijen, ...

`DSN_PGRANGE_TABLE`: vooral nuttig voor een gepartitioneerde tabel: vermeldt de partities waarop, in geval van een table scan, een "page range scan" gebeurt.

DSN_SORTKEY_TABLE: info over op welke kolom of expressie gesorteerd wordt, van welk datatype deze kolom is, en hoe het expliciet sorteren verloopt (indien één van de SORT_-kolommen van de PLAN_TABLE of de nieuwe DSN_SORT_TABLE de waarde 'Y' bevat).

Daarnaast zijn er nog de voor dit betoog minder belangrijke tabellen DSN_PTASK_TABLE, DSN_PGROUP_TABLE, DSN_STATEMENT_CACHE_TABLE, DSN_STATEMENT_RUNTIME_INFO, DSN_QUERY_TABLE. Verder bestaat ook de mogelijkheid, althans voor sommige van de genoemde tabellen, om die met prefix DB2OSC aan te maken i.p.v. met de eigen user-id.

Een nieuwe explain-mogelijkheid

Tenslotte is er ook een tabel, SYSIBM.DSN_VIRTUAL_INDEXES die globaal zichtbaar is (prefix SYSIBM), en waarin lijnen komen met de structuur van SYSIBM.INDEXES. Deze tabel laat voor het eerst "what if"-optimisatie toe: "wat zou het access path worden wanneer bepaalde index(en) zouden gecreëerd of verwijderd worden?" (zonder daartoe effectief deze indexen te moeten creëren of verwijderen).

Merken we eerst en vooral op dat deze nieuwe tabel (van DB2 9) ondertussen geretrofitted is naar DB2 v8, zodat ook OSC voor DB2 v8 gebruik kan maken van de "what if"-mogelijkheden. Goed nieuws voor wie niet dadelijk naar versie 9 gaat!

De tabel DSN_VIRTUAL_INDEXES bevat, naast de kolommen die ook in SYSIBM.SYSINDEXES bestaan, additioneel nog een kolom MODE (mogelijke waarden 'C' of 'D': creëer dan wel delete deze index), en een kolom ENABLE (mogelijke waarden 'Y' of 'N'): deze info door EXPLAIN laten gebruiken of niet. Anders gezegd: voor EXPLAIN is het net alsof de rijen met ENABLE='Y' effectief in SYSINDEXES zitten!

In de praktijk

IBM suggereert zelf om alle genoemde tabellen enkel te gebruiken via de "optimisation tools" zoals OSC of 3rd party-producten. Nochtans laat de (weliswaar summiere) informatie in de manuals toe om zelf aan de slag te gaan met deze tabellen. Zeker indien het gebruik van OSC bij u geen optie is, zou dit een nuttig alternatief kunnen zijn.

Eerst en vooral moeten de bewuste tabellen gecreëerd worden. De nodige SQL-statements kunnen zo gekopieerd worden uit de Performance Monitoring en Tuning Guide.

Dan gewoon EXPLAIN draaien, vanuit SPUFI of met de BIND-optie EXPLAIN(YES). En tenslotte de genoemde tabellen bekijken. Queries schijven voor het ondervragen van die tabellen laten we als oefening voor de lezer. Hebt u één of meer dergelijke queries in elkaar kunnen knutselen? We horen het graag, en publiceren ze (met uw naam erbij, indien u dat wenst) in één van onze volgende nummers van Exploring DB2!

Waarom u nieuwe DB2 9 SQL echt nodig hebt!

Kris Van Thillo (ABIS)

Elke nieuwe versie van DB2 for z/OS introduceert bijkomende mogelijkheden op het gebied van SQL statements. In wat volgt pogen we aan de hand van concrete situaties hun nut en effectiviteit aan te geven.

Het ETT/ETL proces: MERGE.

In een data warehouse omgeving komt volgend scenario vaak voor. Bij een 'refresh' van data uit één of andere (OLTP, batch, ...) omgeving geldt het volgende: is een gegeven nog niet geregistreerd in het warehouse, dan moet dit gegeven worden toegevoegd; is dit gegeven reeds aanwezig, dan moet het worden aangepast/geüpdatet. Of nog: als een gegeven niet bestaat, voeg het toe; als het bestaat, wijzig het indien nodig of relevant. Bijhouden van een volledige 'historiek' is immers niet steeds noodzakelijk!

Het UPSERT (“update” dan wel “insert”) of MERGE statement doet zijn intrede in DB2 for z/OS versie 9. Merk op dat IBM ervoor gekozen heeft zich niet te houden aan de reeds lang door andere leveranciers van relationele database gehanteerde implementatiesyntax! De winst aan efficiëntie bij gebruik is in elk geval aanzienlijk: één statement vervangt programmalogica waarin individuele UPDATE en INSERT statements dienen te worden gecodeerd.

Voorbeeld

In een BI/DW tabel “`klanten`” willen we per klant het aantal kinderen (kolom “`kind`”) bijhouden. Historiek m.b.t. dit aantal is niet vereist.

```
MERGE INTO klanten K
USING VALUES ((:klno, :kind, :kupdate) FOR :n ROWS)                          (a)
AS N (klno, kind, kupdate)                                                   (b)
ON (K.klno = N.klno)                                                         (c)
WHEN MATCHED THEN     UPDATE  SET K.kind = N.kind, K.kupdate = N.kupdate
WHEN NOT MATCHED THEN INSERT (klno, kind, kupdate)
                      VALUES (N.klno, N.KIND, N.kupdate)
NOT ATOMIC CONTINUE ON SQL EXCEPTION;                                        (d)
```

Input van de operatie is een array (a) (of één enkele rij); alle rijen van de array worden individueel verwerkt (b). Afhankelijk van het al dan niet voldoen aan de MERGE voorwaarde (c) wordt een insert of een update actie uitgevoerd. De optionele instructie (d) geeft aan dat niet het MERGE statement, maar de individuele rijen die door MERGE worden behandeld als atomaire (en dus annuleerbare) acties moeten worden behandeld.

Het merge-statement is bruikbaar overal waar SELECT, INSERT, DELETE en UPDATE statements toegelaten zijn (interactief gebruik, in applicaties, dynamisch PREPAREd, ...). In combinatie met triggers biedt het MERGE statement extra uitdagingen waar nauwkeurig moet op worden gelet. Zo is bijvoorbeeld een MERGE statement niet toegelaten op een view waarop een INSTEAD OF trigger werd gedefinieerd.

Eenvoudiger SQL door meer verzamelingenleer!

DB2 for z/OS ondersteunt reeds lang de UNION operator; sinds DB2 9 zijn ook de statements INTERSECT en EXCEPT beschikbaar.

- INTERSECT levert rijen op die in twee of meer verschillende result-sets (d.w.z. resultaten van queries) terug te vinden zijn;
- EXCEPT levert rijen op die in slechts één van de twee result sets (d.w.z., resultaten van queries) terug te vinden zijn.

INTERSECT ALL en EXCEPT ALL vertonen het verwachte gedrag: dubbels blijven behouden in het resultaat. Eisen wat betreft de kolomcompatibiliteit van de individuele SELECT statements blijven natuurlijk gelden zoals bij UNION!

Belangrijkste voordeel van deze statements: de mogelijkheid eenvoudigere en intuïtievere SQL te schrijven. Stel bijvoorbeeld dat u alle primaire keys wil terugvinden waarvoor geen overeenkomende waarde in een foreign key kolom voorkomt...

Voorbeeld

```
SELECT pk_col  FROM pk_table
  EXCEPT
SELECT fk_col  FROM fk_table
```

De alternatieve formulering, met NOT EXISTS, is in dit geval een stuk omslachtiger en vooral ook niet symmetrisch in de twee tabellen.

Opschonen van interface tabellen.

Met het oog op het efficiënt uitvoeren van bepaalde applicaties wordt vaak met 'staging' tabellen gewerkt - tabellen waarin data voor latere verwerking wordt voorbereid. Deze data dient dan, na verwerking, te worden opgeschoond. Deze tabellen zijn meestal ofwel 'echte', standaard tabellen, ofwel bijvoorbeeld “declared global temporary tables”. Hoe nu de inhoud van deze tabellen efficiënt opschonen? TRUNCATE biedt ons nieuwe mogelijkheden.

Voorbeeld

```
TRUNCATE print_orders
DROP STORAGE                                (a)
IGNORE DELETE TRIGGERS                      (b)
IMMEDIATE                                   (c)
```

Dit statement is eigenlijk niet meer of niet minder dan een efficiënte massieve delete operatie. Standaard worden bestaande delete triggers niet uitgevoerd (b); en kan tevens worden aangegeven dat de vrijgekomen ruimte na delete moet worden behouden, dan wel echt

vrijgegeven aan de 'tablespace' (a). REUSE STORAGE kan echter niet worden opgenomen voor tabellen in een simple tablespace. (Maar die gebruikt u niet meer - toch?) Het is ook mogelijk aan te geven of de TRUNCATE moet kunnen worden geannuleerd - via een ROLLBACK - of niet (c)! Het TRUNCATE-statement kan deel uitmaken van een omvangrijke transactie, en zal dus niet automatisch aanleiding geven tot het uitvoeren van een COMMIT-operatie. Merk op dat een PK-tabel in een referentiële structuur niet kan worden ge-TRUNCATE-d!

Hebt u een Oracle achtergrond? Denk dan even hieraan: de DB2 truncate logt! Misschien de gekende LOAD REPLACE met de LOG NO optie? Misschien in voorliggende context toch maar een 'mass delete' gebruiken (bij tablespaces van het segmented of 'universal' type)?

Juist is juist - right?

Er zijn vaak situaties denkbaar waar 'juist' niet steeds 'juist' moet zijn. Denk bijvoorbeeld aan BI/DW applicaties: wat is de gewenste nauwkeurigheid van bepaalde rapporten? En wat is de toegevoegde waarde bekomen door het eisen van absolute nauwkeurigheid? Denk aan queue-achtige tabellen, waarbij de data in een tabel kan worden beschouwd als een set van te verwerken acties, waarbij de volgorde tussen deze acties niet relevant is.

De optie SKIP LOCKED DATA kan gebruikt worden in SQL SELECT, SELECT INTO, PREPARE en UPDATE/INSERT instructies met geïntegreerde select. Bij gebruik van deze optie wordt data die op een of andere manier 'exclusief' geLOCK-ed is door andere transacties indien nodig gewoon 'geSKIPped' - zonder enige waarschuwing. Een “positioned delete” na verwerking zorgt ervoor dat de data geen tweede maal verwerkt wordt.

Opnieuw - interfacetabellen.

Te verwerken data wordt in een interfacetabel voorbereid en aangepast/aangevuld - voor efficiënte “processing” achteraf. Denk hierbij bijvoorbeeld aan een applicatie die voor alle cursusinschrijvingen van een bepaalde periode in een 'email_to_send' tabel alle gegevens voor een automatisch te versturen 'bevestiging van inschrijving' klaarmaakt; zodanig dat tijdens de nachtelijke batchverwerking een job kan lopen die deze info uit de tabel ophaalt, de e-mails genereert en verstuurt.

In vorige releases van DB2 zou de 'e-mail genererende' applicatie minstens 2 SQL statements moeten bevatten: een SELECT om de gegevens uit de interface tabel op te halen; en een DELETE om deze te verwijderen. Dit is niet langer het geval! Versie 9 heeft inderdaad de mogelijkheid geïntroduceerd om het resultaat van een UPDATE, DELETE of MERGE ook onmiddellijk in variabelen te SELECT-eren! Concreet: de ge-DELETE data wordt in één actie geselecteerd voor latere verwerking!

Indien rij-per-rij verwerking nodig is, kan deze instructie zonder veel problemen in een “declare cursor” worden geïntegreerd.

Voorbeeld (pseudo)

```
DECLARE EMAIL_CUR CURSOR FOR
SELECT EMAIL, LASTNAME, COURSE, COURSEDATE, ... FROM OLD TABLE
  (DELETE FROM EMAIL_TO_SENT
   WHERE ENROLDATE = ...)
```

Merk op dat SELECT op INSERT statements reeds sinds versie 8 door DB2 zijn ondersteund - noodzakelijk, vermits steeds vaker data niet door de applicatie, maar door DB2 zelf wordt gegenereerd (via triggers, sequences, ...).

Tot slot...

Tot slot, en off-topic: interface tabellen slaat u toch ook op in NOT LOGGED tablespaces, niet?

CURSUSPLANNING FEB - JUN 2010

DB2 concepten	450 EUR	op aanvraag
DB2 for z/OS, een totaaloverzicht	2025 EUR	01.03(L), 17/05(W), 07.06(L)
DB2 UDB for LUW, totaaloverzicht	2025 EUR	17.05(W)
RDBMS-concepten	375 EUR	01.03(L), 12.04(L),17.05(W), 07.06(L)
Basiskennis SQL	375 EUR	02.03(L), 13.04(L), 18.05(W), 08.06(L)
DB2 for z/OS basiscursus	1275 EUR	03.03(L), 19.05(W), 09.06(L)
DB2 UDB for LUW basiscursus	1275 EUR	26.05(W)
SQL-QMF voor eindgebruikers	1275 EUR	26.05(W)
SQL workshop	800 EUR	11.03(W), 22.03(L), 17.06(W), 21.06(L)
SQL voor gevorderden	450 EUR	25.03(L), 21.06(W)
DB2 SQL PL, triggers, stored procedures	450 EUR	26.03(L), 22.06(W)
DB2 for z/OS programmeren voor gevorderden	900 EUR	22.03(L), 31.05(W)
DB2 for z/OS: SQL performance	1350 EUR	26.04(L), 23.06(W)
XML in DB2	450 EUR	29.04(L), 12.05(W)
DB2 for z/OS database administratie	1900 EUR	03.05(L)
DB2 for z/OS operations & recovery	1425 EUR	22.03(W), 26.05(UK), 28.07(UK)
DB2 for z/OS systems performance and tuning	1000 EUR	25.03(W), 12.04(UK), 19.07(UK)
DB2 LUW DBA - Kernvaardigheden	1800 EUR	29.03(L), 22.06(W)
DB2 v8 upgrade, DB2 9 upgrade	450 EUR	op aanvraag
Data warehouse concepten	450 EUR	16.03(W)
SQL voor BI	450 EUR	22.04(W)
Exploring DB2 - live!	175 EUR	op aanvraag

Plaats: L = Leuven, W = Woerden, UK = High Wycombe (bij Londen);
voor details en andere cursussen, zie `http://www.abis.be/html/nlTraining.html`

Postbus 220
Diestsevest 32
BE-3000 Leuven
Tel. 016/245610
Fax 016/245639
http://www.abis.be/
training@abis.be

abis
TRAINING & CONSULTING

Postbus 122
Pelmolenlaan 1-K
NL-3440 AC Woerden
Tel. 0348-435570
Fax 0348-432493
http://www.abis.be/
training@abis.be