ABIS NIEUWSBRIEF VOOR DB2 PROFESSIONALS

JAARGANG 7 - NUMMER 2, APRIL 2011

OPEN CURSOR

Het is ondertussen traditie van uw aller Exploring DB2 om bij het verschijnen van een nieuwe DB2-versie (DB2 10 for z/OS dus) wat duiding te leveren bij wat de nieuwe mogelijkheden zijn. Zoals de trouwe lezer al weet, zal het niet bij deze ene bijdrage blijven maar zullen we in de volgende nummers uiteraard verder ingaan op wat de nieuwste ontwikkelingen zijn op het DB2-front. En vooral: in hoeverre we die nu ook echt nodig hebben.

Diezelfde bekommernis is ook één van de motivaties voor de reeks over "extended explain tables": wat is de meerwaarde van de nieuwe grafische tools (OSC, OE, Data Studio, Optim, ...) van IBM, en op welke manier kunnen we (voor een subset van die functionaliteit) terugvallen op wat voorhanden is zonder extra inspanningen.

Veel leesplezier!

Het ABIS DB2-team.

IN DIT NUMMER:

- In een eerste artikel, *Temporele data en DB2 10 for z/OS* (I), proberen we de belangrijkste nieuwigheid van versie 10 te duiden door een vergelijking met wat Oracle in dit verband te bieden heeft; wordt vervolgd ...;
- Daarna, in *Queries on extended Explain tables in DB2* vindt u een tweede bijdrage over de nieuwe Explain-tabellen en hoe hieruit performance-informatie te extraheren;
- Tenslotte bespreekt *Scalaire functies: het MySQL-aanbod* de verschillen (en gelijkenissen) tussen DB2 en MySQL voor wat betreft benaming en gebruik van de belangrijkste scalaire functies.

2

CLOSE CURSOR

In het volgende nummer: deel twee van de bijdrage over temporele data in DB2 10; ook wordt de reeks DB2 versus ... voortgezet met een vergelijking van scalaire functies tussen DB2 en SQL Server. Tot dan!

Temporele data en DB2 10 for z/OS (I)

Kris Van Thillo (ABIS)

Mijn favoriete hoofdstuk in een cursus Oracle database administratie is na al die jaren nog steeds het hoofdstuk dat in detail de betekenis en de werking uiteenzet van 'Oracle Undo Segments' - rollback segmenten voor die-hards. Favoriet om een aantal redenen.

Uiteraard, omdat deze undo segmenten, alsook de hiermee geassocieerde behandeling van before-images en data versioning, het hart vormt van elk Oracle database systeem. Inderdaad, diverse mogelijkheden die door de Oracle engine geboden worden, stoelen op het gebruik van deze segmenten. Het Oracle credo "Writers do not block Readers" is geïmplementeerd op basis van undo segmenten; en verklaart ten dele het commerciële succes van Oracle. Favoriet ook, omdat het toelaat aan te stippen dat IBM tot de conclusie is gekomen dat dit alles eigenlijk niet zo dom is want gemiddeld efficiënt, resources bespaart, en de deur opent naar nieuwe en andere types applicaties. En dan kan ik niet anders dan even stil te staan bij wat IBM allemaal moet opzetten en uitdokteren om die functionaliteit te bieden die men in Oracle 'gewoon' is. Maar vooral ook favoriet om iets wat weinigen beseffen: dat Oracle en IBM gewoon toepassen wat door een Vlaming werd uitgevonden, nl. een techniek om te reizen door de tijd (zie *De tuf-tuf-club*, Suske en Wiske (1952)).

Temporele Data

DB2 10 for z/OS biedt de mogelijkheid om verschillende 'versies' van data beschikbaar te hebben voor applicaties. In tegenstelling met vorige releases van DB2 hoeft een applicatie-ontwikkelaar of een applicatie-DBA hiertoe dus niet langer zelf een oplossing te bouwen. Opzetten van ondersteuning voor temporele data, en het gebruik ervan, gebeurt nu gewoon aan de hand van SQL statements.

Alles draait dus om tijd - wanneer is met de data iets gebeurd? En wens ik in mijn applicaties met data te werken van voor of na die gebeurtenis? DB2 10 maakt een onderscheid tussen 2 referentiekaders die hiertoe kunnen worden gebruikt:

- referentiekader *system time*: het 'reële' tijdstip waarop iets is gebeurd met de data, vormt het referentiekader (het moment van de update, delete, insert) voor DB2;
- referentiekader *application time* of *business time*: het temporeel referentiekader wordt bepaald door de applicatie of business context, en moet door hen worden opgelegd. Ben je bijvoorbeeld ingeschreven voor een opleiding voor of na de start van een early bird kortingsperiode?;

'*Start*'-tijden en '*Eind*'-tijden worden dan gebruikt om de validiteit van de data binnen een tijdskader te kunnen bepalen. of anders gezegd: om te bepalen welke *versie* van de data moet worden beschikbaar gemaakt.

Procedure

De procedure voor het werken met temporele data is relatief eenvoudig. Kort samengevat:

- maak de tabel aan waarbij wordt aangegeven dat temporele ondersteuning vereist is. Hiertoe werden de CREATE en ALTER table instructies uitgebreid;
- voorzie de nodige extra kolommen die het DB2 moeten toelaten 'tijd' op te volgen. Dit zullen waarschijnlijk steeds kolommen van het type timestamp zijn, op een specifieke manier gedefinieerd;
- voor temporele ondersteuning op basis van de 'system time' dient een schaduwtabel (history tabel) te worden aangemaakt (CREATE LIKE), die moet worden geassocieerd met de oorspronkelijke tabel (ALTER TABLE).

DML

Het werken met de data op basis van select, update, insert en delete statements wordt mogelijk wat complexer. Details en voorbeelden komen in een volgend nummer van Exploring DB2 aan bod, maar het komt hierop neer:

- voor temporele data op basis van 'system time' doet DB2 automatisch wat nodig is om data binnen het correcte tijdskader te plaatsen. Update, insert en delete statements moeten niet worden aangepast; "delete" wordt b.v. automatisch vertaald naar "verhuis naar de history-tabel". Het ophalen van data 'uit het verleden' vergt natuurlijk wel enige aanpassingen vermits nu dient te worden aangegeven welke versie van de data nodig is: SELECT ... FOR SYSTEM_TIME AS OF ...; FOR SYSTEM_TIME FROM ... TO ...; FOR SYSTEM_TIME BETWEEN ... AND ....
- voor temporele data op basis van 'business time' wordt e.e.a. iets complexer. Vermits het tijdskader door de applicatie of de business wordt bepaald, moet die natuurlijk steeds het tijdskader meegeven bij het manipuleren van de data. Deze instructies werden daarnaast ook uitgebreid met de mogelijkheid uitdrukkelijk te verwijzen naar de business time waarvoor de wijziging moet worden toegepast. Ophalen kan via SELECT ... FOR BUSINESS_TIME AS OF ...; FOR BUSINESS_TIME FROM ... TO ...; FOR BUSINESS_TIME BETWEEN ... AND ...

Het beschikbaar worden van temporele ondersteuning in DB2 biedt enorme mogelijkheden; en zal ongetwijfeld zijn impact hebben op traditionele applicatie-ontwikkeling. Voorts opent het belangrijke mogelijkheden in het domein van de auditing, error detectie en correctie, en statistische analyses. In een volgend nummer van Exploring DB2 komen de details aan bod!

Queries on extended Explain tables in DB2

Abe Kornelis (Bixoft)

This article is a follow-up on a preceding article on the exploration of the extended explain tables, see *http://www.abis.be/resources/exploredb2/j7/984exploredb2j7n1.pdf* where you will also find some pointers to related information sources. For details about the mentioned explain tables, refer to the Performance Monitoring and Tuning Guide.

In the first article the possibility of improving the lay-out of the query results was mentioned. Some readers have asked for more details, so here we will start with doing just that. We will conclude with an explain of the explain query itself.

Improving the lay-out of the query results

The first few columns of the query (as developed in the prior article) contain numbers, but we want to display only blanks when the number is zero. Because a query result column can contain only one type of data, the numbers need to be represented as string values. This is achieved by coding `STRIP(CHAR(colname))` with the disadvantage that numbers now appear left-aligned under the column heading. To make text right-aligned, we need to concatenate with a number of leading spaces, such that the total number of characters tallies up to the width of the column:

```
REPEAT(width-LEN(STRIP(column)), ' ') CONCAT STRIP(column)
```

Where **width** is the number of characters to be used for the result. Essentially this formula removes all spaces, then prefixes as many spaces as needed to make the result field **width** characters wide. It can be used for columns, but a formula can be inserted as well. Thus `STRIP(CHAR(colname))` can be substituted for `column`, resulting in:

```
REPEAT(width-LEN(STRIP(CHAR(colname))), ' ')
CONCAT STRIP(CHAR(colname))
```

This is the formula we've used to right-align various columns that contain mainly numbers, and sometimes blanks (or even other text strings). It works nicely, but... SPUFI and DB2 do not understand that the result is a fixed-width column. They make the result columns very wide because the return value of a REPEAT function has a width which is only available at runtime.

Luckily, there's an easy solution. To make DB2 understand the fixed width, we simply apply the formula `SUBSTR(expression, 1, width)` to the expression above and DB2 will happily take the first **width** characters from the prior result - which changes nothing, except that SPUFI now knows the exact column width before the data output comes in.

The remaining problem is that the formula now gets to be quite complex. Especially because we also have to use a CASE construction to distinguish between zero values and non-zero values. To maintain readability, I've created yet another Common Table Expression (remember we already had three CTEs) named EXPLAIN_DATA which holds the results of what thus far was our final query.

With the use of this new CTE as an intermediate step, the final query now gets to be relatively simple: it just displays relevant columns and applies some formatting when needed. Creating the new CTE is quite straightforward: its `AS` clause takes the complete pre-existing query, consisting of two `SELECT` statements combined using a **`UNION ALL`**.

The new catch is: DB2 does not care if you have multiple result columns with the same name, but for a CTE column names really have to be unique. Our query happens to have two `TYPE` columns: one for `QBLOCK_TYPE` and one for `TABLE_TYPE`, both selected from `PLAN_TABLE`. To resolve the issue I relabelled the second one as `TBLTYP`.

The resulting query did what it had to do, but there was an issue with the `ORDER BY`. The final `SELECT` takes a column `OPSQ` as input; this column is formatted and displayed as result column `OPSQ`. The same name is applied to the input and the result, but the content is not the same. Turns out DB2 will use the result column, while I want to use the input value for sorting. Thus, no error was issued, yet the output rows were in the wrong order. To force DB2 to use the input value, I might have explicitly qualified the column name with the base table name. However, I have chosen to relabel the result column as `OPSEQ`.

Finally, just to show a viable alternative for aligning character data, I've used a feature of the `SUBSTRING` function to align the `MC` (MatchCols) column. By specifying on the `SUBSTR` function a length that may be greater than the length of the input string value, we effectively extend the string on the right hand side with blanks up to the number of characters specified. Before DB2 V8 this would only work for variable-length input strings; as of DB2 V8 it also works for fixed-length input string values. The side effect of this solution is that it will produce a left-aligned column, rather than a right-aligned one. You can see this on the output because the 1-characters align under the heading's 'M' rather than its 'C' character.

After all was said and done (and tested, etc.) I ended up with the query in Appendix A. Please note that column `PAR` has been re-introduced (but removed from the result set) to show applicable parallellism. Also we've applied colours to the various query blocks. This will ease the analysis in the next part of this article.

Appendix B contains the output of that query.

Explaining the Explain query

After executing

```
EXPLAIN ALL SET QUERYNO=115 FOR
```

with the above explain query the various explain tables will be supplied with appropriate data.

Running the explain query above for queryno 115 yields the explain information for the explain query itself. For details please see the first query output listing in Appendix B.

Some points of note regarding these results:

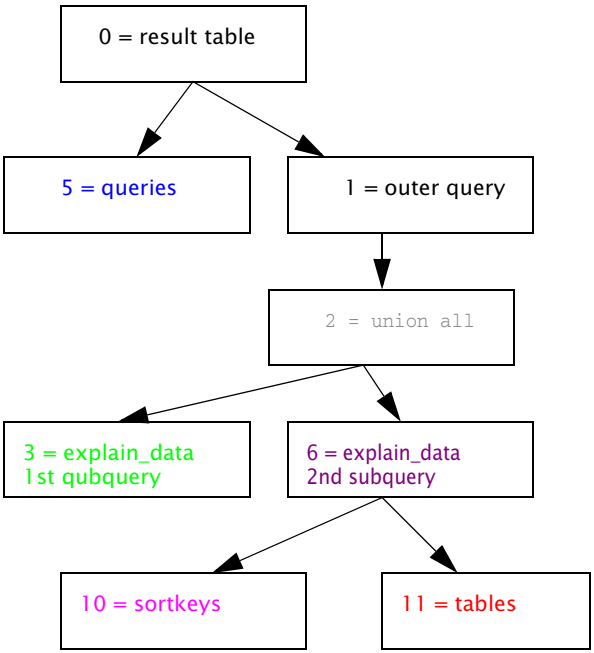
- There are lots of Tablespace Scans with Sequential Prefetch -- not so strange since our explain tables were created without any indexes. We'll get to change that in a future article. For now we'll just concentrate on the interpretation of the available explain data.
- The query is subdivided into seven query blocks, identified by numbers ranging from one to eleven. But it is not readily clear which number corresponds to which part of the query.
- There are two rows that have NULL values for table and creator. We'll get to figure out how and why in a moment.
- The Merge Scan executed for Qblockno=6 and Planno=4 requires no less than four sorts, where two should suffice: one for each input table. And sure enough: the two rows with NULL table references are amongst these four.
- The ordering looks a bit funny. Maybe we need to improve on that, too?

As a first step in interpreting these explain data, we'll need to establish the relationships between the query blocks. Which Qblockno translates to which subquery and how are these blocks related to each other? To correlate the data to pieces of query text, we can either look at the table names or at the correlation names. Since the latter have been omitted from the query to conserve space, we'll use table names. Going from top to bottom we see:

- Three lines for QBLK=10. This block is a Nested Loop Join between CTE Queries and table DSN_Sortkey_Table. This is the implementation of CTE Sortkeys, marked magenta.
- Three lines for QBLK=11. This block is a Nested Loop Join between CTE Queries and table Plan_Table. This is the implementation of CTE Tables, marked red.
- Three lines for QBLK=3. This block is a Nested Loop Join between CTE Queries and table Plan_Table, followed by an Outer Join with table Dsn_Pgrange_Table. This must be the first subquery for the CTE Explain_Data, marked green.
- Eleven lines for QBLK=6. This block contains various joins and sorts. It starts with a Nested Loop Join between CTE Queries and table DSN_Sort_Table. So this must be the second subquery of the CTE Explain_Data, marked purple.
- A single line for QBLK=2, representing the UNION ALL. This completes the CTE Explain_Data, marked light grey (three locations in the query text).

- Two lines for QBLK=1. It uses only data from CTE Explain_Data, and it sports a sort for the Order By clause. Must be our outer query, marked black.
- A single line for QBLK=5. This Qblock uses only Sysibm.Sysdummy1. No doubt about it: this is the CTE Queries, marked light blue.

Having correlated the Qblock Numbers to the various parts and pieces in our query text, we can proceed to draw the relationships between these query blocks. To do so, we look at the columns PBLK (Parent Block) and QBLK (Query Block). From these data we can draw the following diagram (using the same colours):



So, what does this tell us? First, we see that the CTE Queries has been removed from the tree and has been placed on a separate branch. This indicates DB2 will evaluate the subquery before it starts doing all 'other' query blocks. This is a true, materialised CTE in the sense that it is evaluated only once, but its results are used multiple times.

Second, we see that all other CTEs have been placed in a single big branch. This indicates that these CTEs have been substituted into the outer query. The optimizer has converted the references to these CTEs into old-fashioned subqueries. So these CTEs only serve to improve readability, nothing else. I call this type of CTE an optical CTE. Functionally, optical CTEs are exactly the same as sub-queries, but technically, they enhance understandability, readability and maintainability of your queries.

Please mind, that the diagram shows hierarchical relationships, not necessarily temporal ones. It says nothing about the order of execution. Some of the Query blocks may even be executed in parallel -- for example: (3, 10, 11) or (10, 11) followed by (3, 6). If we had more details on how the optimizer is planning to manage all these blocks of work we could draw a network planning diagram.

Anyway, now that we know what is what, we can see that the Sort rows in the result table specify erroneous tables. Take for example Query Block 6. It is reported as doing a sort for Uniqueness on Table Dsn_Sortkey_Table. But it doesn't: it sorts the result-set for CTE Sortkeys. Unfortunately, the correct name of the table being sorted is not available anywhere at all, so we'll have to set these columns to spaces for sort-rows.

But where did we go wrong? It happened because the table DSN_SORTKEY_TABLE holds a table reference which we've used to retrieve the name of the table or CTE being sorted. However, it does not tell what is being sorted; rather it relates to the source of the data being sorted. It will point back to the table (and column) that any specific sort column data item was selected from. When sorting on data that has been mangled by some function or expression, then it contains a zero value. Hence the NULL values in some of our sort-rows.

Now for the Merge Scan executed for Qblockno=6 and Planno=4 which requires no less than four sorts, as we have already noticed. We now know that the NULL values in the CREATOR and TABLE columns are not part of the problem. They have a different cause.

First of all, let me show you the most relevant columns from DSN_SORTKEY_TABLE for these four sort steps:

SORTNO	ORDERNO	EXPTYPE	TEXT
1	1	COL	SKEY.QUERYNO
1	2	COL	SKEY.QBLOCKNO
1	3	COL	SKEY.PLANNO
1	4	COL	SKEY.SORTNO
1	5	COL	SKEY.TABNO
2	1	COL	PLAN.QUERYNO
2	2	COL	PLAN.TABNO
2	3	COL	PLAN.TABLE_TYPE
2	4	COL	PLAN.CREATOR
2	5	COL	PLAN.TNAME
3	1	EXP	CAST(SKEY.QUERYNO AS INTEGER)
3	2	EXP	CAST(SKEY.TABNO AS SMALLINT)
4	1	EXP	CAST(TBLS.QUERYNO AS INTEGER)
4	2	EXP	CAST(TBLS.TABNO AS SMALLINT)

Looking back at the output of the query, we can see that this planno will join userid.TABLES to a WORK table. So userid.TABLES must be the result set from the CTE TABLES, which is being joined to the result set of the preceding joins between QUERIES, DSN_SORT_TABLE and CTE SORTKEYS. This join is using the ON-clause:

```
ON  TBLS.QUERYNO = SKEY.QUERYNO
AND TBLS.TABNO   = SKEY.TABNO
```

So one might expect a sort on columns QUERYNO and TABNO for both tables. This is what we see in sorts 3 and 4. Even though they sort on a CAST function for these columns, they sort these two tables on the very same columns. The CAST function does not change anything. From the GJOUGJOU column, which gives the purpose of the sort, we can see that sort 3 is for the 'combined table' (result of prior joins) and sort 4 is for the 'new' table (TABLES).

Understanding the place and function of sorts 3 and 4 leaves us with sorts 1 and 2, marked A and B in the query output. They both sort a 'combined' table for a join (the GJOUGJOU column shows the first J for JOIN). Furthermore, sorts 1 and 2 do not even sort on compatible columns. A quick look at DSN_SORTKEY_TABLE reveals that sort 1 will sort on data retrieved from DSN_SORTKEY_TABLE rows and sort 2 will sort on data retrieved from PLAN_TABLE rows. The second sort column is QBLOCKNO for sort 1 and TABNO for sort 2. There seems to be no direct relation between the two sorts.

At the same time, planno 5 will do a join to PLAN_TABLE but has no sorts associated with it, even though one would expect two sorts there. Could it be sorts 1 and 2 relate to planno 5, rather than planno 4? At least sort 2 sorts on data derived from PLAN_TABLE rows. That is what I'd expect for the sort for 'new' table. Hmmm, could it be the explain data is in error? Maybe because there is a CASE construct in the ON clause? I have asked IBM. Thus far, no answers...

The last issue I raised was about ordering. The order of rows looks a bit funny. In the first place the sort rows for Matching Scan Joins appear after the row for the actual Join. When we wrote the ordering inversion (alphabetic sort, but blanks at end) for column SORT in the ORDER BY clause, we selected NLjoin only but should have included MSjoin in the exception rule as well.

In the second place, we have concluded that CTE QUERIES should be performed first, but it appears at the very end of the output. I'd rather have it at the top. For the moment, we'll leave it where it is. In theory, we should be able to move it to its proper place using data from DSN_STRUCT_TABLE, but again, there seem to be issues with the data in this table. Too bad.

About the author: Abe F. Kornelis (1962) has been an assembler application programmer and system programmer since the early 1980's. In the 2000's he has been a mainframe and Cobol instructor, a DB2 infrastructure specialist, and a DB2 DBA, focusing on query quality and all aspects of DB2-related performance issues. Abe has been self-employed for most of these years.

If you have any questions or remarks you can send a mail to abe@bixoft.nl.

Scalaire Functies: het MySQL-aanbod

Peter Vanroose (ABIS)

Dit is het derde artikel in de vergelijkende reeks over de SQL scalaire functies van DB2 en andere relationele database-systemen. Deze maal bekijken we de tegenhangers op MySQL. Net zoals in de vorige bijdragen bekijken we hier MySQL vanuit de DB2-invalshoek, en vermelden vooral die functies waarvoor een ISO/ANSI-standaardfunctie gedefinieerd is. In de handleiding van MySQL 5.0 of recenter vindt u uiteraard een volledig overzicht van beschikbare functies: zie b.v. hoofdstuk 11 van de MySQL Reference Manual.

Tekst-manipulatie-functies

Het eerste wat opvalt bij het overlopen van de beschikbare functies in MySQL, is de expliciete verwijzing naar compatibiliteit met standaard (ANSI/ISO) SQL. Wat de tekst-manipulatie-functies betreft, voorziet MySQL echter geen mogelijkheid om een argument van de vorm `USING <char length units>` mee te geven. Verder wordt in veel gevallen (zoals in DB2) gelijkaardige functionaliteit geboden door functies met andere namen of met 'klassiekere' argument-syntax.

- Lengte van een string:

```
CHAR_LENGTH(<character valued expression>)
CHARACTER_LENGTH(<character valued expression>)                    [cf DB2]
OCTET_LENGTH(<string>)
LENGTH(<string>)                                                   [= DB2]
BIT_LENGTH(<character valued expression>)
```

Bij de eerste twee functies is het resultaat het aantal characters, bij OCTET_LENGTH en LENGTH drukt het resultaat het aantal bytes van de string uit. De eerste drie functies zijn compatibel met de ISO-standaard-functie, weliswaar dus zonder de 'USING'-mogelijkheid.
De laatste functie geeft het aantal bits van het argument (wat meestal 8 keer zo groot is als het aantal bytes).

- Gedeelte van een string:

```
SUBSTRING(<char valued expr> FROM <start_position> [FOR <length>])
SUBSTRING(<char valued expr>,<start_position>[,<length>])          [cf DB2]
MID(<char valued expr>,<start_position>,<length>)
SUBSTR(<char valued expr>,<start_position>[,<length>])             [cf DB2]
```

Bij alle functies kan <position> negatief zijn; in dat geval wordt van achteraan geteld, dus b.v. SUBSTRING('ABC', -1) geeft als resultaat 'C'. DB2 voorziet deze mogelijkheid niet!

Alle functies doen precies hetzelfde. De eerste vorm van de functie SUBSTRING is compatibel met ISO, maar mist opnieuw het optionele argument 'USING' voor het aangeven van de tel-eenheid: alle functies tellen in character-eenheden, dus nooit in byte-eenheden.

- Een substring vinden in een string:

```
POSITION(<search> IN <source>)
INSTR(<source>,<search>)
LOCATE(<search>,<source>[,<startpos>])                    [= DB2]
```

Alle functies geven een "character position expression" terug, dus de (begin)positie van de gevonden substring, of 0 indien niet gevonden.

Alleen de eerste functie is ISO-standaard; opnieuw ontbreekt dus de "USING"-mogelijkheid: interpretatie is altijd in character-eenheden.

- Argument omzetten naar hoofdletters/kleine letters:

```
UPPER(<string>)      LOWER(<string>)                      [= DB2]
UCASE(<string>)      LCASE(<string>)                      [= DB2]
```

UPPER en LOWER zijn compatibel met standaard SQL.

- haal spaties (of andere padding) achteraan of vooraan weg:

```
TRIM(<string>)
LTRIM(<string>)      RTRIM(<string>)                      [= DB2]
```

De ISO/ANSI standaard voorziet bovenop TRIM(<string>) voor blanktrimming enerzijds ook de mogelijkheid om andere padding-characters (dan spaties) weg te halen: `TRIM(<char> FROM <string>)`.

Anderzijds zijn LTRIM en RTRIM geen standaard-functies, maar heeft TRIM volgens de SQL-standaard de optionele argumenten LEADING en TRAILING om dit te doen: `TRIM(LEADING <char> FROM <string>)`.

MySQL heeft geen van beide mogelijkheden, en volgt hier dus DB2!

- Overlay: In het resultaat zal het tweede argument te zien zijn als vervanging van een gedeelte van het eerste argument, vanaf een bepaalde positie.

```
INSERT(<source_string>,<offset>,<length>,<replacement_string>)      [= DB2]
REPLACE(<source_string>,<search_string>,<replacement_string>)       [= DB2]
```

Dit dus i.p.v. de ISO/ANSI standaard

`OVERLAY(<source> PLACING <repl> FROM <offset> [FOR <length>])`.

of de gelijkaardige DB2-variant (met komma's i.p.v. keywords).

REPLACE is eigenlijk gelijk aan POSITION gevolgd door INSERT.

Datum-manipulatie-functies

- Net zoals recentelijk in DB2 en Oracle toegevoegd, ondersteunt ook MySQL de standaard EXTRACT-functie, die een onderdeel toont van een datum of een tijd:

```
EXTRACT(<extract field> FROM <extract source>)
```

Naast de DB2-mogelijkheden kan `<extract_field>` ook `quarter`, `week`, `year_month`, `day_second`, `hour_minute`, `microsecond` , ... zijn.

Numerieke functies

Alle genoemde functies zijn standaard SQL-functies die ook bestaan in MySQL. De meerderheid van deze functies spreken voor zich, en zijn ook in DB2 beschikbaar, zodat verdere uitleg overbodig is:

```
ABS(<numeric value expression>)
FLOOR(<numeric value expression>)
CEILING(<numeric value expression>)
SQRT(<numeric value expression>)
LN((<numeric value expression>)
EXP((<numeric value expression>)
MOD(<numeric value expression dividend>, <numeric value expression divisor>)
POWER(<numeric value expression base>, <numeric value expression exponent>)
```

Daarnaast heeft MySQL ook de functie LOG als synoniem voor LN en CEIL als synoniem voor CEILING. MySQL voorziet verder de volgende alternatieve operator-notatie voor de functie MOD:

```
<numeric value expr dividend> % <numeric value expr divisor>
```

Dit is duidelijk geen standaard SQL, maar wel compatibel met wat de meeste programmeertalen voorzien voor de modulo-operatie.

Algemene functies

De CAST-functie, die als resultaat de representatie geeft van het 1ste argument met als datatype de specificatie van het 2de argument:

```
CAST(<value expression> AS <data type>))                    [= DB2]
```

Niet-standard functies

Behalve de hierboven reeds genoemde niet-helemaal-standaard-functies heeft MySQL nog een zeer groot aanbod aan niet-standaard scalaire functies. Dit uitzonderlijk grote aantal functies is vooral een gevolg van het feit dat in MySQL (als open-source software) zeer veel individuen hebben bijgedragen aan de software, en dus i.h.b. implementaties hebben geschreven voor zeer veel scalaire functies die ondertussen standaard meegeleverd worden met MySQL.

Om alleen de meest opvallende te noemen:

- Tekstmanipulatie-functies:

`CONCAT_WS(<sep>,<string1>,<string2>,...)`: concat met separator

`REVERSE(<string>)`: geef characters in omgekeerde volgorde

`ORD(<char>)`: character code voor het eerste character (mogelijk multi-byte) van de gegeven tekst

`HEX(<string>)`: geeft de hexadecimale representatie van (de character code van) alle characters in `<string>`

`QUOTE(<string>)`: plaatst single quotes omheen het argument, en escape eventuele embedded quotes en backslashes;
geeft het woord "NULL" (zonder quotes) indien argument NULL is.

`CHARSET(<string>)` en `COLLATION(<string>)`:
geeft de character set en "collation" van de gegeven tekst-string.

- Numerieke functies:

`PI()`: het getal pi = 3.141592...

`RAND()`: geeft een random (float) getal tussen 0 en 1

`UUID()`: genereer een (random) "UUID", d.w.z., een Universal Unique Identifier (een 128-bit getal).

CURSUSPLANNING JUNI-AUGUSTUS 2011

DB2 concepten	450 EUR	op aanvraag
DB2 for z/OS, een totaaloverzicht	2025 EUR	06.06(L)
DB2 UDB for LUW, totaaloverzicht	2025 EUR	najaar
RDBMS-concepten	375 EUR	06.06(L)
Basiskennis SQL	375 EUR	07.06(L)
DB2 for z/OS basiscursus	1275 EUR	najaar
DB2 UDB for LUW basiscursus	1275 EUR	najaar
SQL-QMF voor eindgebruikers	1275 EUR	najaar)
SQL workshop	800 EUR	20.06(W)
SQL voor gevorderden	450 EUR	24.06(L)
DB2 SQL PL, triggers, stored procedures	450 EUR	20.06(W)
DB2 for z/OS programmeren voor gevorderden	900 EUR	najaar
DB2 for z/OS: SQL performance	1350 EUR	21.06(W)
XML in DB2	450 EUR	najaar
DB2 for z/OS database administratie	1900 EUR	14.06(W)
DB2 for z/OS data recovery	1425 EUR	28.06(UK), 11.08 (UK)
DB2 for z/OS systems performance and tuning	1000 EUR	15.08(UK)
DB2 LUW DBA - Kernvaardigheden	1800 EUR	20.06(W)
DB2 v8 upgrade, DB2 9 upgrade	450 EUR	op aanvraag
Data warehouse concepten	450 EUR	najaar
SQL voor BI	450 EUR	najaar
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

Appendix A: query bij het artikel van Abe Kornelis.

```
--
-- Query on PLAN_TABLE with Page-Range info
--                      and sorting info
--          with improved lay-out
--
   WITH    QUERIES (QUERYNO)
   AS      (SELECT   114                -- replace with your queryno
            FROM     SYSIBM.SYSDUMMY1
           )
         , SORTKEYS
   AS      (SELECT DISTINCT
                    SKEY.QUERYNO
                  , SKEY.QBLOCKNO
                  , SKEY.PLANNO
                  , SKEY.SORTNO
                  , SKEY.TABNO
            FROM     DSN_SORTKEY_TABLE SKEY
            JOIN     QUERIES QRYS
                 ON  QRYS.QUERYNO = SKEY.QUERYNO
            WHERE    SKEY.TABNO <> 0
           )
         , TABLES
   AS      (SELECT DISTINCT
                    PLAN.QUERYNO
                  , PLAN.TABNO
                  , PLAN.TABLE_TYPE
                  , PLAN.CREATOR
                  , PLAN.TNAME
            FROM     PLAN_TABLE PLAN
            JOIN     QUERIES QRYS
                 ON  QRYS.QUERYNO = PLAN.QUERYNO
            WHERE    PLAN.TABNO <> 0
           )
         , EXPLAIN_DATA
   AS      (SELECT   CHAR(PLAN.QUERYNO) AS QUERY
                  , CASE PLAN.PARENT_QBLOCKNO
                         WHEN 0 THEN ' '
                                ELSE CHAR(PLAN.PARENT_QBLOCKNO)
                    END AS PBLK
                  , CHAR(PLAN.QBLOCKNO) AS QBLK
                  , CHAR(PLAN.PLANNO) AS PLNO
                  , CASE PLAN.MIXOPSEQ
                         WHEN 0 THEN ' '
                                ELSE CHAR(PLAN.MIXOPSEQ)
                    END AS OPSQ
                  , ' ' AS SORT
                  , PLAN.QBLOCK_TYPE AS TYPE
                  , CASE PLAN.JOIN_TYPE
                         WHEN 'F' THEN 'Full'
                         WHEN 'P' THEN 'Pair'
                         WHEN 'S' THEN 'Star'
                         WHEN 'L' THEN CASE PLAN.METHOD
                                            WHEN 1 THEN 'L/R'
                                            WHEN 2 THEN 'L/R'
                                            WHEN 4 THEN 'L/R'
                                                   ELSE '?'
                                       END
                         WHEN ' ' THEN CASE PLAN.METHOD
                                            WHEN 1 THEN 'Innr'
                                            WHEN 2 THEN 'Innr'
                                            WHEN 4 THEN 'Innr'
```

```
                                    ELSE ' '
                            END
                ELSE '*ERR*'
  END AS JOIN
, CASE PLAN.METHOD
       WHEN 0 THEN 'First'
       WHEN 1 THEN 'NLjoin'
       WHEN 2 THEN 'MSjoin'
       WHEN 3 THEN 'Sort'
       WHEN 4 THEN 'Hybrid'
              ELSE 'UNKNWN'
  END AS METHOD
, CASE PLAN.TABLE_TYPE
       WHEN 'B' THEN 'Buffer'
       WHEN 'C' THEN 'CTE'
       WHEN 'F' THEN 'TabFun'
       WHEN 'M' THEN 'MQT'
       WHEN 'Q' THEN 'Temp'
       WHEN 'R' THEN 'Recurs'
       WHEN 'T' THEN 'Table'
       WHEN 'W' THEN 'Work'
                ELSE ' '
  END AS TBLTYP
, SUBSTR(PLAN.CREATOR, 1, 8) AS CREATOR
, SUBSTR(PLAN.TNAME, 1, 18) AS TABLE
, SUBSTR(PLAN.ACCESSNAME, 1, 8) AS NDXNAME
, CASE PLAN.PRIMARY_ACCESSTYPE
       WHEN 'D' THEN 'D/'
       WHEN 'T' THEN 'T/'
                ELSE ''
  END CONCAT
  STRIP(PLAN.ACCESSTYPE) AS ACCESS
, CASE WHEN PLAN.PAGE_RANGE = 'Y'
       THEN CHAR(RANGE.NUMPARTS)
       ELSE ''
  END AS NUMPARTS
, CASE PLAN.PREFETCH
       WHEN 'S' THEN 'SEQ'
       WHEN 'L' THEN 'LST'
       WHEN 'D' THEN 'DYN'
                ELSE ' '
  END AS PREF
, CASE PLAN.INDEXONLY
       WHEN 'Y' THEN 'XO'
                ELSE ' '
  END AS XO
, CASE PLAN.MATCHCOLS
       WHEN 0 THEN ''
              ELSE CHAR(PLAN.MATCHCOLS)
  END AS MC
, CASE PLAN.SORTC_GROUPBY
       WHEN 'Y' THEN 'G'
                ELSE ' '
  END CONCAT
  CASE PLAN.SORTC_JOIN
       WHEN 'Y' THEN 'J'
                ELSE ' '
  END CONCAT
  CASE PLAN.SORTC_ORDERBY
       WHEN 'Y' THEN 'O'
                ELSE ' '
  END CONCAT
  CASE PLAN.SORTC_UNIQ
```

```
                    WHEN 'Y' THEN 'U'
                             ELSE ' '
               END CONCAT
               CASE PLAN.SORTN_GROUPBY
                    WHEN 'Y' THEN 'G'
                             ELSE ' '
               END CONCAT
               CASE PLAN.SORTN_JOIN
                    WHEN 'Y' THEN 'J'
                             ELSE ' '
               END CONCAT
               CASE PLAN.SORTN_ORDERBY
                    WHEN 'Y' THEN 'O'
                             ELSE ' '
               END CONCAT
               CASE PLAN.SORTN_UNIQ
                    WHEN 'Y' THEN 'U'
                             ELSE ' '
               END AS GJOUGJOU
             , CASE PLAN.PARALLELISM_MODE
                    WHEN 'I' THEN 'I/O'
                    WHEN 'C' THEN 'CPU'
                    WHEN 'X' THEN 'SYS'
                             ELSE ' '
               END AS PAR
     FROM      PLAN_TABLE PLAN
     JOIN      QUERIES QRYS
          ON   QRYS.QUERYNO = PLAN.QUERYNO
LEFT JOIN      DSN_PGRANGE_TABLE RANGE
          ON   RANGE.QUERYNO  = PLAN.QUERYNO
          AND  RANGE.QBLOCKNO = PLAN.QBLOCKNO
          AND  RANGE.TABNO    = PLAN.TABNO
     WHERE     PLAN.METHOD <> 3
    UNION ALL
     SELECT    CHAR(PLAN.QUERYNO) AS QUERY
             , CASE PLAN.PARENT_QBLOCKNO
                    WHEN 0 THEN ' '
                      ELSE CHAR(PLAN.PARENT_QBLOCKNO)
               END AS PBLK
             , CHAR(PLAN.QBLOCKNO) AS QBLK
             , CHAR(PLAN.PLANNO) PLNO
             , CASE WHEN PLAN.MIXOPSEQ = 0
                    THEN ''
                    ELSE CHAR(PLAN.MIXOPSEQ)
               END AS OPSQ
             , CASE WHEN SORT.SORTNO = 0
                    THEN ' '
                    WHEN SORT.SORTNO < 27
                    THEN SUBSTR('ABCDEFGHIJKLMNOPQRSTUVWXYZ',
                                SORT.SORTNO, 1)
                    ELSE '*'
               END AS SORT
             , PLAN.QBLOCK_TYPE AS TYPE
             , '' AS JOIN
             , 'Sort' AS Method
             , CASE TBLS.TABLE_TYPE
                    WHEN 'B' THEN 'Buffer'
                    WHEN 'C' THEN 'CTE'
                    WHEN 'F' THEN 'TabFun'
                    WHEN 'M' THEN 'MQT'
                    WHEN 'Q' THEN 'Temp'
                    WHEN 'R' THEN 'Recurs'
                    WHEN 'T' THEN 'Table'
```

```
                        WHEN 'W' THEN 'Work'
                                 ELSE ' '
                   END AS TBLTYP
                 , SUBSTR(TBLS.CREATOR, 1, 8) AS CREATOR
                 , SUBSTR(TBLS.TNAME, 1, 18) AS TABLE
                 , SUBSTR(PLAN.ACCESSNAME, 1, 8) AS NDXNAME
                 , CASE PLAN.PRIMARY_ACCESSTYPE
                        WHEN 'D' THEN 'D/'
                        WHEN 'T' THEN 'T/'
                                 ELSE ''
                   END CONCAT
                   STRIP(PLAN.ACCESSTYPE)
                   AS ACCESS
                 , '' AS NUMPARTS
                 , CASE PLAN.PREFETCH
                        WHEN 'S' THEN 'SEQ'
                        WHEN 'L' THEN 'LST'
                        WHEN 'D' THEN 'DYN'
                                 ELSE ' '
                   END AS PREF
                 , CASE PLAN.INDEXONLY
                        WHEN 'Y' THEN 'XO'
                                 ELSE ' '
                   END AS XO
                 , CASE PLAN.MATCHCOLS
                        WHEN 0 THEN ''
                               ELSE CHAR(PLAN.MATCHCOLS)
                   END AS MC
                 , SUBSTR(SORT.SORTC, 1, 4) CONCAT
                   SUBSTR(SORT.SORTN, 1, 4) AS GJOUGJOU
                 , CASE PLAN.PARALLELISM_MODE
                        WHEN 'I' THEN 'I/O'
                        WHEN 'C' THEN 'CPU'
                        WHEN 'X' THEN 'SYS'
                                 ELSE ' '
                   END AS PAR
          FROM     DSN_SORT_TABLE SORT
          JOIN     QUERIES QRYS
               ON  QRYS.QUERYNO = SORT.QUERYNO
     LEFT JOIN     SORTKEYS SKEY -- Obtain nr of table being sorted
               ON  SKEY.QUERYNO  = SORT.QUERYNO
               AND SKEY.QBLOCKNO = SORT.QBLOCKNO
               AND SKEY.PLANNO   = SORT.PLANNO
               AND SKEY.SORTNO   = SORT.SORTNO
     LEFT JOIN     TABLES TBLS -- Obtain name of table being sorted
               ON  TBLS.QUERYNO = SKEY.QUERYNO
               AND TBLS.TABNO   = SKEY.TABNO
     LEFT JOIN     PLAN_TABLE PLAN--Join back to relevant PLAN_TABLE row
               ON  PLAN.QUERYNO  = SORT.QUERYNO
               AND PLAN.QBLOCKNO = SORT.QBLOCKNO
               AND PLAN.PLANNO   = SORT.PLANNO
               AND(   PLAN.METHOD   = 3
                   OR(    PLAN.SORTC_JOIN = 'Y'
                      AND SUBSTR(SORT.SORTC, 2, 1) = 'J'
                     )
                   OR(    PLAN.SORTN_JOIN = 'Y'
                      AND SUBSTR(SORT.SORTN, 2, 1) = 'J'
                    )  )
        )
SELECT   SUBSTR(REPEAT(' ', 5-LENGTH(STRIP(QUERY))) CONCAT
                                    STRIP(QUERY),
                 1, 5
                ) AS QUERY
```

```
        , SUBSTR(REPEAT(' ', 4-LENGTH(STRIP(PBLK))) CONCAT
                                    STRIP(PBLK),
                 1, 4
                ) AS PBLK
        , SUBSTR(REPEAT(' ', 4-LENGTH(STRIP(QBLK))) CONCAT
                                    STRIP(QBLK),
                 1, 4
                ) AS QBLK
        , SUBSTR(REPEAT(' ', 4-LENGTH(STRIP(PLNO))) CONCAT
                                    STRIP(PLNO),
                 1, 4
                ) AS PLNO
        , SUBSTR(REPEAT(' ', 4-LENGTH(STRIP(OPSQ))) CONCAT
                                    STRIP(OPSQ),
                 1, 4
                ) CONCAT SORT AS OPSEQ
        , TYPE
        , JOIN
        , METHOD
        , TBLTYP
        , CREATOR
        , TABLE
        , NDXNAME
        , ACCESS CONCAT
          CASE WHEN NUMPARTS = ''
               THEN ''
               ELSE '(' CONCAT STRIP(NUMPARTS) CONCAT ')'
          END AS ACCESS
        , PREF
        , XO
        , CASE WHEN LENGTH(STRIP(MC)) = 1
               THEN ' '
               ELSE ''
          END CONCAT STRIP(MC) AS MC
        , GJOUGJOU
        , PAR
 FROM     EXPLAIN_DATA
 ORDER BY QUERY
        , PBLK DESC
        , QBLK
        , PLNO
        , OPSQ
        , CASE WHEN METHOD = 'NLjoin'
                AND SORT = ''
               THEN 'Z'
               ELSE SORT
          END
;
```

Appendix B: query output.

Result-set for Query on PLAN_TABLE with Page-Range info and sorting info with improved lay-out

QUERY	PBLK	QBLK	PLNO	OPSEQ	TYPE	JOIN	METHOD	TBLTYP	CREATOR	TABLE	NDXNAME	ACCESS	PREF	XO	MC	GJOU	GJOU
115	6	10	1		TABLEX		First	CTE	TU00001	QUERIES		R	SEQ				
115	6	10	2		TABLEX	Innr	NLjoin	Table	TU00001	DSN_SORTKEY_TABLE		R	SEQ				
115	6	10	3	A	TABLEX		Sort	Table	TU00001	DSN_SORTKEY_TABLE							U
115	6	11	1		TABLEX		First	CTE	TU00001	QUERIES		R	SEQ				
115	6	11	2		TABLEX	Innr	NLjoin	Table	TU00001	PLAN_TABLE		R	SEQ				
115	6	11	3	A	TABLEX		Sort	Table	TU00001	PLAN_TABLE							U
115	2	3	1		NCOSUB		First	CTE	TU00001	QUERIES		R	SEQ				
115	2	3	2		NCOSUB	Innr	NLjoin	Table	TU00001	PLAN_TABLE		R	SEQ				
115	2	3	3		NCOSUB	L/R	NLjoin	Table	TU00001	DSN_PGRANGE_TABLE		R	SEQ				
115	2	6	1		NCOSUB		First	CTE	TU00001	QUERIES		R	SEQ				
115	2	6	2		NCOSUB	Innr	NLjoin	Table	TU00001	DSN_SORT_TABLE		R	SEQ				
115	2	6	3		NCOSUB	L/R	MSjoin	Work	TU00001	SORTKEYS		R	SEQ			J	J
115	2	6	3	A	NCOSUB		Sort	Table	TU00001	DSN_SORT_TABLE		R	SEQ			J	
115	2	6	3	B	NCOSUB		Sort	Work	TU00001	SORTKEYS		R	SEQ				J
115	2	6	4		NCOSUB	L/R	MSjoin	Work	TU00001	TABLES		R	SEQ			J	J
115	2	6	4	A	NCOSUB		Sort	Table	TU00001	DSN_SORTKEY_TABLE		R	SEQ			J	
115	2	6	4	B	NCOSUB		Sort	Table	TU00001	PLAN_TABLE		R	SEQ			J	
115	2	6	4	C	NCOSUB		Sort		-------	----------		R	SEQ			J	
115	2	6	4	D	NCOSUB		Sort		-------	----------		R	SEQ				J
115	2	6	5		NCOSUB	L/R	NLjoin	Table	TU00001	PLAN_TABLE		R	SEQ				
115	1	2	1		UNIONA		First										
115		1	1		SELECT		First	Work	TU00001	EXPLAIN_DATA		R	SEQ				
115		1	2	A	SELECT		Sort	Work	TU00001	EXPLAIN_DATA						O	
115		5	1		NCOSUB		First	Table	SYSIBM	SYSDUMMY1		R	SEQ				

Colours added to relate explain data lines to query blocks.