ABIS NIEUWSBRIEF VOOR DB2 PROFESSIONALS

JAARGANG 10 - NUMMER 1, FEBRUARI 2016

OPEN CURSOR

DB2 11 for z/OS raakt stilaan overal geïnstalleerd. DBA's zoeken hun weg in de nieuwe automatisatie-features; ontwikkelaars en eindgebruikers ontdekken de nieuwe SQL-mogelijkheden zoals het ARRAY-type, "drop column", en grouping sets; en performance-specialisten verdiepen zich in de gewijzigde aanpak van de optimizer: drastischer query rewrites, en transparantere communicatie met de gebruiker, o.a. wat betreft feedback over een RUNSTATS die betere statistieken had kunnen aanleveren.
Over twee van deze onderwerpen vindt u in dit nummer alvast een bijdrage, en u mag in de volgende nummers meer details verwachten.

Veel leesgenot,
Het ABIS DB2-team.

IN DIT NUMMER:

- *Als statistieken ontbreken...* dan kan de optimizer niet optimaal werken! Lees meer over de nieuwe feedback die de DB2 11 optimizer geeft in de eerste bijdrage van dit nummer.
- Analytische SQL-queries worden steeds complexer; gelukkig heeft DB2 11 daarvoor *Nieuwe SQL: Super-Groups*. Lees meer hierover in het tweede hoofdartikel.
- In Dossier 11 vindt u nog twee nieuwe mogelijkheden van DB2 11 for z/OS: een variant van de LIKE syntax, en SQL-toegang tot de directory.

1

CLOSE CURSOR

In een volgend nummer van Exploring DB2 leest u meer over nieuwe OLAP-syntax in DB2 for z/OS. Verder plannen we ook een themanummer "Big Data" want als we IBM mogen geloven is DB2 11 het geknipte platform voor uw Data Analytics.

Als statistieken ontbreken ...

Kris Van Thillo (ABIS)

Relationele databases – of het nu gaat om Oracle, DB2 voor z/OS of LUW, ... – hebben nood aan gegevens voor het bepalen van optimale toegangspaden naar de data. Deze gegevens zijn vaak statistieken, gebruikt door een optimizer om concreet uit te zoeken hoe dit toegangspad er dient uit te zien.

Wat dit betreft was het gedrag van de optimizer zeer voorspelbaar: ofwel bestaan statistieken, en worden ze gebruikt; ofwel bestaan ze niet, en dan maakt de optimizer een aantal willekeurige veronderstellingen omtrent de te behandelen data. De ene vendor heeft het in deze context over het gebruik van 'default' statistieken, de andere noemt (noemde) dit *rule based optimisatie*.

Vandaag bieden de meeste vendors van relationele databases – al dan niet *out-of-the-box* – bijkomende faciliteiten in deze context.

Context en achtergrond.

Eigenlijk is de redenering relatief eenvoudig: wie beter dan een optimizer is in staat om de kwaliteit van de aangeboden statistieken na te gaan? Concreet: een optimizer moet bij de bepaling van een toegangspad kunnen aangeven dat bepaalde statistieken ontbreken, of dat de kwaliteit van deze statistieken ondermaats is.

Aangeven dat bepaalde *statistieken ontbreken* is natuurlijk niet echt complex: als tijdens de evaluatie van een bepaald access-plan (en de berekening van z'n kostprijs) bepaalde statistieken niet worden gevonden, kan dit 'feit' worden weggeschreven naar een door een beheerder te consulteren systeemtabel.

Evaluatie van de *kwaliteit van de aanwezige statistieken* is minder evident. Men zou zich bijvoorbeeld op de datum van de laatste statistiekgeneratie kunnen baseren; en dus: te oud is minder kwalitatief. Toch wel iets te kort door de bocht misschien... Beter is een systeem dat bij het daadwerkelijke manipuleren van de data '*inline*', '*online*', '*live*' statistieken genereert en vergelijkt met de opgeslagen en gebruikte statistieken bij het bepalen van het initiële toegangspad.

DB2 v11 for z/OS – Feedback tables.

DB2 v11 for z/OS zet een eerste stap in deze richting. Mits de nodige systeemconfiguratie is uitgevoerd, kan DB2 verzocht worden informatie omtrent ontbrekende en/of conflicterende statistieken in het geheugen bij te houden; en te externaliseren naar systeemtabellen na het verstrijken van bijvoorbeeld een bepaald tijdsinterval.

Dit is het enige wat moet gebeuren:

- aangeven dat DB2 deze informatie inderdaad moet bijhouden voor statische en/of dynamische SQL;
- aangeven wat het publicatie-interval is waarmee deze data moet worden geëxternaliseerd naar hier bedoelde systeemtabellen.

Ook het '*start database*' commando kan worden gebruikt om in het geheugen opgeslagen data op een eerder ad-hoc wijze te externaliseren.

De systeemtabellen die worden bijgehouden zijn de volgende:

- *sysibm.sysstatfeedback*, voor typisch interval-based externalisatie van in het geheugen opgeslagen data
- *userid.dsn_stat_feedback*, voor informatie opgehaald/bekomen op basis van een explain statement.

Elke rij in deze tabellen identificeert een statistiek die DB2 nodig heeft om een correcte accesspath-selectie te doen, maar dus ontbrak en/of conflicteerde met een andere statistiek.

De rijen in bovenstaande tabellen identificeren:

- de betrokken *objecten en/of SQL statements*, met name:
  - bij `sysibm.sysstatfeedback` : voor SQL onafhankelijke object-specifieke feedback (bijvoorbeeld op het niveau van tabel, kolom, of kolomgroep);
  - bij `userid.dsn_stat_feedback` : voor correlatie tussen de ontbrekende statistiek en het betrokken (geoptimaliseerde) SQL statement;
- de reden *(REASON)* waarom het object en/of het SQL statement werd opgenomen in de tabel, met name:
  - *basic* – basis statistieken zijn niet beschikbaar op het object/SQL;
  - *conflict* – conflicterende statistieken bestaan (generatie op verschillende tijdstippen; conflicten dus tussen onderling gerelateerde objecten);
  - *lowcard/nullable/default* – betrokken object bevat 'skewed' – dus niet normaal verdeelde – data en er is geen frequentie/histogram statistische data beschikbaar;
- welke type actie dient te worden ondernomen *(TYPE)* om de betreffende statistiek te kunnen genereren, met name:
  - of er tabel (type '*T*') dan wel index (type '*I*') statistieken moeten worden aangemaakt;
  - of er cardinaliteit (type '*C*'), frequentie (type '*F*') of histogram (type '*H*') statistieken moeten worden aangemaakt;

Belangrijk om weten...

De gegevens opgenomen in bovenstaande tabellen moeten door een beheerder natuurlijk in hun juiste context worden geïnterpreteerd.

De twee genoemde tabellen bevatten info die het DB2 moet toelaten een correctere, meer gefundeerde keuze te maken tussen alternatieve toegangspaden; maar de DB2 optimizer garandeert niets, maakt geen voorspellingen en heeft dus niet getest of deze statistieken toegevoegde waarde zullen bieden (m.b.t. de selectie van een concreet toegangspad).

De opgenomen gegevens spreken zich niet uit over de kwaliteit van de statistieken; enkel ontbrekende en/of conflicterende statistieken worden gerapporteerd. DB2 beschikt niet over een mogelijkheid om de kwaliteit van de bestaande statistieken te evalueren, en zal zich b.v. nooit baseren op de aanmaakdatum van bepaalde statistieken.

Het is overigens de taak van de beheerder met deze gegevens iets te doen: DB2 zelf doet er immers niets mee. De beheerder zal dus eerst zelf moeten nagaan of de ontbrekende statistieken relevant zijn, en dus nuttig voor DB2 toegangspad-selectie. De beheerder moet dus ook zelf zorg dragen voor het gebruiken van de correcte DB2 tools/utilities (RUNSTATS) om de ontbrekende statistieken te genereren; en moet zelf de nodige interne procedures aanpassen om deze ontbrekende statistieken in de toekomst automatisch te laten genereren. Uiteraard kunnen hier ook externe tools voor gebruikt worden.

DB2 schoont echter wel zelf bovenstaande tabellen op als de gevraagde statistieken later uiteindelijk voorhanden zijn.

Toepasbaarheid.

Bovenstaande aanpak heeft een duidelijk doel: aangeven dat op een bepaald moment statistieken ontbreken. DB2 kiest er voor met deze informatie verder niets te doen: het is de taak van de beheerder om hier verder actie op te ondernemen. E.e.a. lijkt dus vooral nuttig binnen de context van applicatie/query tuning, waarbij de DB2 optimizer nu de mogelijkheid krijgt aan te geven dat o.a. op basis van de structuur van het SQL statement bijkomende statistieken nodig zijn voor een – *mogelijk* – 'correctere' keuze van het te hanteren toegangspad.

Een volgende stap?

We kunnen ons nu afvragen of bovenstaande in volgende versies van DB2 nog verder kan en zal evolueren. Alhoewel enkel IBM in deze weet hoe en wat, is het mogelijk nuttig even stil te staan bij wat andere vendors in deze context te bieden hebben. Specifiek willen we toch even conceptueel ons licht opsteken bij Oracle.

Meer weten? zoek in Oracle docs naar '*Adaptive Query Optimization*'

Beide vendors beschikken over een technologie om '*inline*' runtime statistieken te genereren betreffende eigenschappen van de te manipuleren data, b.v. gedurende index-creatie, of bij specifieke manipulaties van tabelpartities. Oracle kiest er uitdrukkelijk voor om

deze technologie ook in te zetten bij (initiële) SQL statement optimisatie, om na te gaan of de 'opgeslagen' statistieken aan de 'huidige realiteit' voldoen. De vraag naar kwaliteit van de statistieken wordt hierbij dus beantwoord.

Een andere stap die mogelijk kan overwogen worden is de stap richting automatische herberekening van statistieken die al dan niet permanent moeten worden opgeslagen. Concreet kunnen we ons inderdaad afvragen of de vaststelling dat statistieken niet langer correct zijn niet moet leiden tot één of andere automatische generatie van een nieuwe set. Oracle doet dit voor alle duidelijkheid niet, maar slaat zijn statistische bevindingen wel op in 'afzonderlijke' tabellen die, zoals de traditionele cataloogtabellen, bij elke optimalisatie worden geconsulteerd.

Mooi is dat deze gegevens worden opgeslagen zodanig dat ze bruikbaar zijn voor andere statements (object-gebaseerde opslag) en uiteraard voor quasi identieke SQL-statements (SQL-gebaseerde opslag).

Tot slot.

De in dit artikel beschreven 'feedback'-technologie zoals nu geïmplementeerd door DB2 biedt ons de mogelijkheid na te gaan of DB2 over alle mogelijke (statistische) informatie beschikt om een optimale toegangspad-selectie te kunnen uitvoeren. Daarnaast kan het eveneens de basis vormen voor een verdere, geautomatiseerde benadering van statistiekgeneratie en query-optimalisatie naar verdere versies van DB2 toe.

Of dit laatste een streven moet zijn is niet ondubbelzinnig duidelijk; duidelijk is echter wel dat ook IBM ervoor kiest om DB2 for z/OS steeds SMARTer te maken.

‘Nieuwe’ SQL: Super-Groups

Arnout Veugelen (ABIS)

RDBMS-producenten zijn steeds op zoek naar uitbreidingen van de SQL-syntax. Dat kunnen bijvoorbeeld nieuwe functies zijn, of varianten op de bekende DML. Als zo’n uitbreiding interessant blijkt, wordt ze opgenomen in de ANSI-standaard en gaandeweg overgenomen door de andere fabrikanten. Er kan heel wat tijd over gaan vooraleer zo’n vernieuwingen geïmplementeerd en gepopulariseerd worden over de platformen heen. Onbekend is onbemind, dus is er reden genoeg om stil te staan bij enkele van deze ‘nieuwe’ SQL-mogelijkheden. In dit eerste deel hebben we het over de Group By-extensies Grouping Sets, Cube en Rollup, ook wel Super-Groups genoemd.

Group By-extensies behoren al sinds SQL-99 tot de ANSI-standaard, en zijn dus zeker niet nieuw. DB2 z/OS ondersteunt de bijhorende syntax echter pas sinds de nieuwste versie 11 en vele database-professionals gebruiken ze zelden of nooit. Nochtans vormen Grouping Sets, Cube en Rollup een erg nuttige uitbreiding op de vertrouwde GROUP BY-syntax.

Voor de voorbeelden in dit artikel maken we gebruik van volgende tabel ‘Orders’:

```
CREATE TABLE Orders ( Ordernr     SMALLINT NOT NULL PRIMARY KEY,
                      Orderdate   DATE,
                      Salesperson VARCHAR(20),
                      Prodid      CHAR(4),
                      Qty         SMALLINT,
                      Amount      DECIMAL(10,2)
                    );
```

Ordernr	Orderdate	Salesperson	Prodid	Qty	Amount
1	2014-05-06	Ann	1234	16	150
2	2014-07-08	Ann	5678	5	25
3	2015-08-16	Bob	1234	4	40
4	2015-10-20	Bob	1234	1	10
5	2015-10-22	Ann	9000	10	900
6	2015-10-24	Ann	9000	5	500
7	2016-01-10	Bob	1234	2	20

GROUPING SETS

Grouping sets maken het mogelijk om meerdere groeperingen te combineren in een query, en bijgevolg om allerlei subtotalen te berekenen. Stel bijvoorbeeld dat we volgende verkoopstotalen willen zien:

- Per product (grouping set 1)
- Per product, per verkoper (grouping set 2)
- Globaal (grouping set 3)

Dit kan heel eenvoudig met de volgende SQL:

```
SELECT   Prodid,
         Salesperson AS "Verkoper",
         SUM(Amount) AS "Omzet"
FROM     Orders
GROUP BY GROUPING SETS ( (Prodid),
                         (Prodid, Salesperson),
                         ()
                       );
```

Dit geeft onderstaand resultaat. De eerste drie rijen zijn afkomstig van de eerste grouping set (Prodid), de volgende vier van de tweede grouping set (Prodid, Salesperson) en de laatste rij van de derde grouping set ( ), met name de som van de hele tabel.

Prodid	Verkoper	Omzet
1234	(null)	220
5678	(null)	25
9000	(null)	1400
1234	Ann	150
1234	Bob	70
5678	Ann	25
9000	Ann	1400
(null)	(null)	1645

Weinig verrassend ontstaan er een aantal NULLs (in de kolom Verkoper bij de eerste groepering en in de kolommen Verkoper en Prodid voor de derde groepering). Verderop in dit artikel bekijken we hoe we dat desgewenst kunnen aanpassen.

Zoals u ziet, is de syntax van grouping sets niet erg moeilijk:

```
GROUP BY GROUPING SETS ( (Set1_Kolom1, Set1_Kolom2, ... ),
                         (Set2_Kolom1, Set2_Kolom2, ...), ...
                       )
```

We creëren dus verschillende groeperingen, die overlappend mogen zijn. Klassiek zouden we hetzelfde resultaat kunnen bereiken via een UNION ALL van verschillende queries die enkel verschillen in hun GROUP BY, maar via Grouping Sets gaat het dus veel compacter.

CUBE

Bovendien bestaan er nog twee praktische shortcuts voor specifieke gevallen: ROLLUP en CUBE. Cube neemt het cartesisch product van alle opgesomde grouping sets.

Bijvoorbeeld:

```
SELECT   Salesperson AS "Verkoper",
         Prodid AS "Product",
         COUNT(*) AS "# Transacties",
         SUM(Amount) AS "Omzet"
FROM     Orders
GROUP BY CUBE (Salesperson, Prodid)
```

Verkoper	Prodid	# Transacties	Omzet
(null)	(null)	7	1645
(null)	1234	4	220
(null)	5678	1	25
(null)	9000	2	1400
Ann	(null)	4	1575
Ann	1234	1	150
Ann	5678	1	25
Ann	9000	2	1400
Bob	(null)	3	70
Bob	1234	3	70

Gezien Cube hier met twee argumenten uitgevoerd is, wordt er op vier manieren gegroepeerd: per product, per verkoper, per product & verkoper, en globaal. Op dezelfde manier zou een Cube met drie argumenten overeenkomen met negen verschillende manieren van groeperen. Zonder deze 'nieuwe' syntax zou je dan voor hetzelfde resultaat een UNION van maar liefst negen queries moeten maken.

Rollup

Rollup biedt de mogelijkheid om subtotalen in een bepaalde volgorde op te bouwen tot een algemeen totaal. Zo zouden we bijvoorbeeld op zoek kunnen gaan naar verkoopstatistieken, respectievelijk per maand, per jaar en globaal.

Met ROLLUP kan dat zo:

```
SELECT   YEAR(Orderdate)  AS "Jaar",
         MONTH(Orderdate) AS "Maand",
         COUNT(*) AS "# Transacties",
         SUM(Amount) AS "Omzet"
FROM     Orders
GROUP BY ROLLUP ( YEAR(Orderdate), MONTH(Orderdate) );
```

Opmerking: gebruik bij voorkeur eerder de standaard datumfuncties `EXTRACT(YEAR FROM Orderdate)` en `EXTRACT(MONTH FROM Orderdate)`.

Jaar	Maand	# Transacties	Omzet
2014	5	1	150
2014	7	1	25
2014	(null)	2	175
2015	8	1	40
2015	10	3	1410
2015	(null)	4	1450
2016	1	1	20
2016	(null)	1	20
(null)	(null)	7	1645

Deze rollup is een shortcut voor volgende drie grouping sets:

- (year, month)
- (year)
- ( )

Het algemeen totaal wordt steeds meegenomen bij rollup, net zoals bij cube dus.

NULL-problemen

We kunnen de NULLs in bovenstaand resultaat wegwerken met behulp van COALESCE. Om het woord 'TOTAL' te laten verschijnen in de kolommen maand en jaar, converteren we bovendien naar CHAR.

```
SELECT   COALESCE(CAST(YEAR(Orderdate)  AS CHAR(4)), 'TOTAL') AS "Jaar",
         COALESCE(CAST(MONTH(Orderdate) AS CHAR(4)), 'TOTAL') AS "Maand",
         COUNT(*) AS "# Transacties",
         SUM(Amount) AS "Omzet"
FROM     Orders
GROUP BY ROLLUP ( YEAR(Orderdate), MONTH(Orderdate) );
```

Jaar	Maand	# Transacties	Omzet
2014	5	1	150
2014	7	1	25
2014	TOTAL	2	175
2015	8	1	40
2015	10	3	1410
2015	TOTAL	4	1450
2016	1	1	20
2016	TOTAL	1	20
TOTAL	TOTAL	7	1645

Op deze manier wordt het resultaat een stuk leesbaarder, maar helemaal ideaal is deze oplossing niet. Zoals zo vaak het geval is, kunnen NULL-waarden in de te onderzoeken tabel roet in het eten gooien. Stel dat er in de datumkolom NULLs voorkomen:

Ordernr	Orderdate	Salesperson	Prodid	Qty	Amount
8	(null)	Bob	1234	2	20

Bij het groeperen op (delen van) de datum, zullen in de overeenkomstige kolommen ook NULLs getoond worden, al betreft het geen subtotalen.

De GROUPING-functie kan hier een uitweg bieden: deze functie toont een 1 wanneer de rij een door een super-group gegenereerde NULL bevat in de aangegeven kolom. Bijvoorbeeld:

```
SELECT   YEAR(Orderdate) AS "Jaar",
         MONTH(Orderdate) AS "Maand",
         COUNT(*) AS "# Transacties",
         GROUPING (YEAR(Orderdate)) AS "Vlag_jaartotaal",
         GROUPING (MONTH(Orderdate)) AS "Vlag_maandtotaal"
FROM     Orders
GROUP BY ROLLUP ( YEAR(Orderdate), MONTH(Orderdate) );
```

Jaar	Maand	# Transacties	Vlag_jaartotaal	Vlag_maandtotaal
(null)	(null)	1	0	0
(null)	(null)	1	0	1
2014	5	1	0	0
2014	7	1	0	0
2014	(null)	2	0	1
2015	8	1	0	0
2015	10	3	0	0
2015	(null)	4	0	1
2016	1	1	0	0
2016	(null)	1	0	1
(null)	(null)	8	1	1

Via GROUPING kunnen we dus de verschillende subtotalen gemakkelijk identificeren. In combinatie met CASE valt zo alsnog het vooropgestelde resultaat te genereren:

```
SELECT    CASE
            WHEN GROUPING( YEAR(Orderdate) ) = 0
             THEN CAST( YEAR(Orderdate) AS CHAR(4))
            ELSE  'TOTAAL'
          END             AS "Jaar",
          CASE
            WHEN GROUPING( MONTH(Orderdate) ) = 0
             THEN CAST( MONTH(Orderdate) AS CHAR(4))
            ELSE  'TOTAAL'
          END             AS "Maand",
           COUNT(*)       AS "# Transacties",
           GROUPING (YEAR(Orderdate))  AS "Vlag_jaartotaal",
           GROUPING (MONTH(Orderdate)) AS "Vlag_maandtotaal"
FROM      Orders
GROUP BY ROLLUP( YEAR(Orderdate), MONTH(Orderdate) );
```

Bovendien kan GROUPING ook een interessante insteek bieden voor een HAVING-voorwaarde of een ORDER BY.

Conclusie

Grouping sets, Cube en Rollup vormen duidelijk een interessante uitbreiding op de standaard GROUP BY. Je vermijdt een overvloed aan UNIONs en bovendien kunnen er, afhankelijk van de gebruikte omgeving, performance-winsten gemaakt worden. Ook hybride constructies zijn uiteraard mogelijk, zodat de mogelijkheden haast onbeperkt zijn.

In een volgend nummer gaan we verder met 'nieuwe' SQL: we zullen het hebben over Windowing Functions. Die maken het mogelijk om kolomfuncties, die typisch op een groep of een hele tabel uitgevoerd worden, toch te combineren met informatie uit individuele rijen.

DOSSIER 11

Twee opvallende nieuwe mogelijkheden

Blank insignificant LIKE.

Stel dat uw tabel t een kolom k heeft van type CHAR(10). De volgende query geeft dan alle rijen terug met "Voorbeeld " (inclusief spatie) in die kolom:

```
SELECT * FROM t WHERE k = 'Voorbeeld'
```

Anderzijds geeft de volgende query geen enkele van die rijen terug:

```
SELECT * FROM t WHERE k LIKE '%beeld'
```

want k eindigt helemaal niet op "beeld" maar op "beeld " (inclusief de spatie).

DB2 11 laat nu toe om de manier waarop LIKE omgaat met "trailing blanks" bij te sturen, zodat die laatste SELECT dezelfde rijen (en nog wat meer) teruggeeft als de eerste SELECT. Uiteraard is dit gewijzigd gedrag van LIKE allerminst backward-compatible; daarom is deze functionaliteit niet actief "by default".

Activeren van dit "trailing blanks insignificant" gedrag van LIKE (indien gewenst) gebeurt door de waarde van de nieuwe zPARM `LIKE_BLANK_INSIGNIFICANT` op `YES` te zetten; de default is uiteraard `NO`.

Extra caveat: omdat deze "nieuwe" LIKE eerst een `rtrim` uitvoert op z'n linkerlid, zal omgekeerd "`k LIKE '%beeld_'`" nu plots geen rijen meer teruggeven (voordien wel)!

SQL-toegang tot de directory.

Wat was ook weer het belangrijkste verschil tussen Catalog en Directory? De catalog bestaat uit "gewone" tabellen, weliswaar grotendeels read-only, maar we kunnen ze lezen met gewone SELECT-statements, b.v.

```
SELECT name, creator FROM sysibm.systables WHERE dbname = 'MYDB';
```

De directory daarentegen bestaat uit een 5-tal tablespaces maar die gegevens kunnen hoogstens geraadpleegd worden via commands zoals "`-DISPLAY UTIL(*)`".

Welnu, DB2 11 voor z/OS brengt daar verandering in! De gegevens in de belangrijkste directory-tablespaces zijn nu ook tabellen, en dus op te vragen met SQL SELECT.

Hier zijn ze:

De Database Descriptors (in tablespace `DSNDB01.DBD01`) zijn zichtbaar via de nieuwe directory-tabel `SYSIBM.DBDR` : één rij per DBD-section, met als kolommen: `DBID` en `SECTION` (samen de primary key), en `DBD_DATA` (binair, nl. van datatype `BLOB(2G)`);

`SYSIBM.SCTR` bevat de Skeleton Cursor Table, dus de "plan" metadata; kolommen: `SCTNAME` (14 bytes: plan-naam+section nr.) en `SCTDAT`;

`SYSIBM.SPTR` bevat de Skeleton Package Table, de "package" metadata dus; kolommen: `SPTCOLID`, `SPTNAME`, `SPT_DATA`, `SPT_EXPLAIN`, `SPTCONID` en `SPTVER`;

`SYSIBM.SYSLGRNX` bevat Log range informatie, met o.a. de `DBID` & `OBID` van de getroffen objecten, de change timestamp, en de start/stop RBA & LRSN;

SYSIBM.SYSUTIL bevat alle details over utilities die aan het draaien zijn; deze tabel bevat dus dezelfde informatie als wat -DISPLAY UTIL laat zien.

Peter Vanroose

CURSUSPLANNING, FEBRUARI – JUNI 2016

DB2 for z/OS, een totaaloverzicht	2175 EUR	11.04(W), 18.05(L)
DB2 for LUW, een totaaloverzicht	2175 EUR	11.04(W), 18.05(L)
Basiskennis SQL & relationele databases	810 EUR	25.02(L),23.03(W),09.05(L),30.05(W)
DB2 for z/OS basiscursus	1365 EUR	11.04(W), 18.05(L)
DB2 for LUW basiscursus	1365 EUR	11.04(W), 18.05(L)
SQL-QMF voor eindgebruikers		op aanvraag
SQL workshop	860 EUR	21.03(L), 21.04(W), 23.06(L)
SQL voor gevorderden	480 EUR	21.04(L), 01.07(W)
SQL voor BI rapportering & analyse	910 EUR	19.04(L), 13.06(W)
Software-ontwikkeling met SQL PL	960 EUR	26.05(L)
DB2 triggers, stored procedures, en User-Defined Functions	480 EUR	20.05(L)
DB2 for z/OS: programmeren voor gevorderden	960 EUR	09.05(L)
DB2 for z/OS: SQL performance	1440 EUR	20.06(L)
XML in DB2		op aanvraag
DB2 for z/OS: database administratie	2020 EUR	06.06(L)
DB2 for z/OS: installation & migration	850 GBP	14.03(UK), 12.05(UK)
DB2 for z/OS: data recovery	850 GBP	08.03(UK), 23.06(UK)
DB2 for z/OS: systems performance and tuning	850 GBP	21.04(UK)
DB2 LUW DBA – Kernvaardigheden	1920 EUR	28.06(W)
Database applicatieprogrammering met JDBC	480 EUR	08.06(L)
DB2 10 for z/OS: new features		op aanvraag
DB2 10 for LUW: new features		op aanvraag
DB2 11 for z/OS: changes & new features	510 EUR	09.03(W), 22.04(L)
Actief gecoachte zelfstudie mainframe		op aanvraag
Data warehouse concepten	480 EUR	23.03(L), 13.05(W)
Big Data concepten	480 EUR	18.04(L), 30.05(W)
Big Data in de praktijk	960 EUR	22.02(L),07.03(W),02.05(L),15.06(W)

Plaats: L = Leuven, W = Woerden, UK = High Wycombe (bij Londen);
alle cursussen ook op aanvraag;
Voor details en andere cursussen, zie `http://www.abis.be/html/nlTraining.html`
Pour détails et autres cours, voir `http://www.abis.be/html/frTraining.html`
For details and other courses, see `http://www.abis.be/html/enTraining.html`