ABIS NIEUWSBRIEF VOOR DB2 PROFESSIONALS

JAARGANG 6 - NUMMER 2, AUGUSTUS 2009

OPEN CURSOR

Bij ABIS verkennen we graag de grenzen van de (IT-)mogelijkheden, in het bijzonder van DB2.
In het kader van z'n stage bij ABIS dit voorjaar, heeft Timót Méchler één van onze interne applicaties geoptimaliseerd, met name het efficiënt opzoeken van personen en bedrijven in onze database. Z'n verslag over de "grenzen van DB2" die we hierbij verkend hebben, leest u in dit nummer. U vindt z'n bijdrage in de oorspronkelijke versie, nl. in net Frans; de vertaling kunt u op onze website nalezen.
Andere grenzen worden verlegd met een nieuwe IEEE-standaard voor "floating point"-getalrepresentatie, waar DB2 als de kippen bij is met z'n nieuwe datatype DECIMAL FLOAT.
En tenslotte brengen we verslag van recente ontwikkelingen voor Data Warehouses en Business Intelligence (BI), met name de Cognos-producten van IBM.

Veel leesgenot!
Het ABIS DB2-team.

IN DIT NUMMER:

- In *Optimisation: chaînes de caractères* vindt u een gedetailleerd verslag van een geslaagde implementatie van efficiënt zoeken op basis van een "ongeveer"-key in DB2 v8 (iets wat in DB2 9 nog een stuk eenvoudiger wordt dankzij "index on expression").
- Dan duiken we volop in versie 9 van DB2 (z/OS en LUW versies) met *DECFLOAT en ander nieuws in DB2 9* waarin u o.a. meer te weten komt over een zeer recente standaardisatie-inspanning van IBM.
- Tenslotte vervolgen we de reeks bijdragen over het samengaan van IBM en Cognos, en de gevolgen hiervan voor het Data Warehouse landschap, met *Cognos, an IBM Company (deel 2).*

CLOSE CURSOR

Dit najaar vindt zoals gebruikelijk de Europese IDUG-conferentie plaats, deze maal in Rome. De ideale plaats om de vinger aan de pols te houden wat betreft "hot topics" en ontwikkelingen voor DB2-gebruikers. In ons volgend nummer zullen we dus ongetwijfeld verslag uitbrengen van belangrijke nieuwe ontwikkelingen en tendenzen binnen de DB2-wereld!

Optimisation: chaînes de caractères

Timót Mechler (ABIS / Haute École de Bruxelles)

De Nederlandstalige vertaling van dit artikel vindt u in *Bijlage* op p. 17

La recherche pertinente d'informations dans la base de données telles que le nom d'un client ou d'une entreprise, n'est pas aussi simple qu'il n'y paraît. Cet article vous montrera quelques-unes des difficultés a surmonter; et le support d'implémentation offerte par DB2.

Contexte de base

Une personne effectue une recherche d'une compagnie avec pour filtre le nom de l'entreprise. Mais pour que la recherche soit fructueuse, il faut que la personne ait fourni exactement le même nom en entrée que le nom se trouvant dans la base de données.

En tenant compte du set de valeurs ci-dessous, on peut constater qu' une recherche ayant pour valeur 'Dehertog' renverra un résultat NULL, non seulement à cause de l'absence de traitement mixte des lettres majuscules/minuscules, mais aussi parce que l'espace situé entre les deux mots faussera le résultat.

Dans le même ordre d'idées, la recherche 'menestra&co (finance)', 'KRGC' et 'sanseau' n'afficheront également aucune compagnie dans les résultats.

Vous l'aurez compris, la recherche est donc très contraignante car basique: une simple et stricte comparaison entre la valeur de recherche encodée à l'écran d'une part, et les noms de compagnies existants dans la base de données d'autre part.

Exemple: la table companies

```
CREATE TABLE companies
(cono    SMALLINT NOT NULL,
 coname  VARCHAR(45));

SELECT * FROM companies;

 CONO  CONAME
  1    De Hertog
  2    menestra&co -finance-
  3    K.R.G.C.
  4    sanséau
```

Application (COBOL avec SQL):

```
SELECT cono,  coname
INTO   :var1, :var2
FROM   companies
WHERE  coname LIKE :varInput
```

Optimisation

Pour optimiser la recherche, il faut donc pouvoir faire une comparaison entre les valeurs en entrée et ceux de la base de données, sans tenir compte de la 'CASE' ainsi que des caractères accentués, caractères spéciaux et espaces.

On pourrait arriver à ce résultat en utilisant correctement dans la requête des fonctions comme UPPER et TRANSLATE, permettant d'épurer les chaînes de caractères en vue d'une comparaison dans la clause WHERE. Malheureusement, ceci est beaucoup trop coûteux en termes de performances pour des tables excédant le millier de lignes, car ces fonctions seront invoquées pour chacune des lignes de la table.

La solution que nous vous proposons est simple, performante et valable pour les bases de données relationnelles supportant l'usage de triggers.

On commence par rajouter dans la table *companies* une nouvelle colonne contenant le nom de l'entreprise 'normalisé': épuré des caractères indésirés. La valeur de recherche en entrée ('varInput' dans l'exemple) sera quant à elle traitée par l'application afin d'être également épurée des caractères non désirés. Ce sera donc sur base de cette dernière ainsi que de la nouvelle colonne que se fera la comparaison dans la clause WHERE de la requête.

Technique

Du coté base de données (table *companies* dans l'exemple):

- On rajoute une nouvelle colonne contenant la valeur épurée et prête pour une comparaison, c.à d. sans espaces (on aura donc une forme concaténée), caractères spéciaux et accentués (ces derniers sont remplacés par leurs équivalents sans accents) et le tout en majuscule ou minuscule, au choix.

Exemple: création d'une nouvelle colonne

```
ALTER TABLE companies
  ADD COLUMN  conamex  VARCHAR(45);

CREATE INDEX compidx1 ON companies (conamex ASC) CLUSTER  ;

UPDATE companies SET conamex =
      REPLACE(UPPER(TRANSLATE(coname,
      'AAAAAAAAEEEEIIIIOOOOOOUUUUYYCNAAAAAAAAEEEEIIIIOOOOOOUUUUYCN ',
      'áàâäãåăæéèêëíìîïóòôöõøúùûüÿýçñÁÀÂÄÃÅĂÆÉÈÊËÍÌÎÏÓÒÔÖÕØÚÙÛÜÝÇÑ'||
      '-_%()''.,:;[]/\+<>{}!?|º*$£µ`~=^',' ')),' ','') ;

SELECT * FROM companies

 CONO  CONAME                 CONAMEX
 1     De Hertog              DEHERTOG
 2     menestra&co -finance-  MENESTRACOFINANCE
 3     K.R.G.C.               KRGC
 4     sanséau                SANSEAU
```

- On créera également un index sur cette nouvelle colonne afin d' augmenter les performances lors des recherches.
- Finalement, on doit aussi mettre la colonne à jour en mettant les valeurs de la colonne 'coname' sous une forme épurée dans la colonne rajoutée *conamex*.

On veillera également à ce que la nouvelle colonne soit tenue à jour: création de nouveaux triggers maintenant à jour la colonne *conamex* lors d'une modification du nom de la compagnie ou lors d'une nouvelle entrée dans la table.

Exemple: création du trigger

```
CREATE TRIGGER compnsrt
   AFTER INSERT ON companies
   REFERENCING NEW AS n
   FOR EACH ROW MODE DB2SQL
   UPDATE companies SET conamex =
            REPLACE(UPPER(TRANSLATE(coname,
          'AAAAAAAAEEEEIIIIOOOOOOUUUUYYCNAAAAAAAAEEEEIIIIOOOOOOUUUUYCN ',
          'áàâäãååæéèêëíìîïóòôöõøúùûüÿýçñÁÀÂÄÃÅÅÆÉÈÊËÍÌÎÏÓÒÔÖÕØÚÙÛÜÝÇÑ'||
            '-_%()''.,:;[]/\+<>{}!?|°*$£µ`~=^',' ')),' ','')
   WHERE cono = n.cono;

CREATE TRIGGER compudpt
   AFTER UPDATE OF coname ON companies
   REFERENCING OLD AS o NEW AS n
   FOR EACH ROW MODE DB2SQL
   UPDATE companies SET conamex =
            REPLACE(UPPER(TRANSLATE(coname,      <etc>;
```

Du côté de l'application, il faut épurer la chaîne de caractère en entrée ('varInput' dans l'exemple), pour y arriver on fera les même traitements que du côté de la base de données, à 2 exceptions près:

- il ne faut non pas enlever les espaces mais les remplacer par des '%' (afin que les espaces soient interprétés comme wild card) ;
- on rajoute le caractère '%' à la fin de la chaîne pour s'offrir une plus grande souplesse lors des recherches; il suffira alors de ne spécifier que le début de la valeur recherchée.

Exemple: l'application (COBOL avec SQL)

```
SELECT cono,  coname
INTO   :var1, :var2
FROM   companies
WHERE  conamex LIKE :varInput
```

Le code dépend fortement de la langue de programmation utilisée. Cependant il est conseillé de favoriser la réutilisabilité du code (méthode, fonction, sous-programme,...) car il ne se limite pas à la recherche d'un nom, mais bien à toutes sortes de recherche par chaîne de caractères.

Au final nous pouvons rechercher dans la base de données l'entreprise 'De Hertog' avec les valeurs comme: DeHertog, d'Ehertog, de hertog, deHértög, deher, de tog. Et ce même si le nom de l'entreprise 'De Hertog' dans la base de données comprend des caractères spéciaux ou accentués.

Il faut tenir compte du fait que la solution proposée peut néanmoins avoir des conséquences non désirables (au niveau des valeurs retournées par l'application).

DB2 for z/OS v9

Il est intéressant de noter que l'on peut arriver au même résultat de façon plus simple si l'on possède une version récente de la base de données (à partir de la version 9 de DB2 for z/OS).

Connu sous divers noms ('index on expression' sur DB2 et Oracle, 'functional index' sur PostgreSQL), un index de ce type nous permet d'éviter la création d'une nouvelle colonne et des triggers réalisés. En effet, une fois créé sur la colonne originelle, l'index porte sur les valeurs traitées de la colonne référencée.

Exemple: création d'un index en DB2 9

```
CREATE INDEX compindx
  ON companies
  ( REPLACE(UPPER(TRANSLATE(coname,
      'AAAAAAAEEEEIIIIOOOOOOUUUUYYCNAAAAAAAEEEEIIIIOOOOOOUUUUYCN ',
      'áàâäãåæéèêëíìîïóòôöõøúùûüÿýçñÁÀÂÄÃÅÆÉÈÊËÍÌÎÏÓÒÔÖÕØÚÙÛÜÝÇÑ'||
      '-_%()''.,:;[]/\+<>{}!?|º*$£µ`~=^',' ')),' ','') ) ;

SELECT  cono,  coname
INTO    :var1, :var2
FROM    companies
WHERE   coname LIKE :varInput
```

L'index tenu à jour par le système de gestion de la base de données rend la déclaration de nouveaux triggers superflue.

Au niveau de la requête, c'est donc sur base de l'index sur la colonne originale que se fera la comparaison avec la valeur en entrée.

DECFLOAT en ander nieuws in DB2 9

Peter Vanroose (ABIS)

Nieuw in DB2 9

Bij het bekijken van de "nieuwe" SQL-features in versie 9 van DB2 for z/OS én DB2 for Linux, UNIX en Windows (LUW), vallen een aantal zaken op. Eerst en vooral dat DB2 for z/OS enkele nieuwe datatypes introduceert: BIGINT, dat reeds bestond in DB2 v8 voor LUW; BINARY en VARBINARY, als alternatieven voor CHAR/VARCHAR FOR BIT DATA (maar vreemd genoeg nog niet in DB2 9 voor LUW), en vooral DECFLOAT, waarover we het in deze bijdrage in detail willen hebben.

Bij het introduceren van nieuwe SQL-syntax, bij elke nieuwe release van DB2, spelen duidelijk altijd de volgende argumenten:

- Compatibiliteit met de (laatste) SQL-standaard
- Wegwerken van syntactische verschillen tussen DB2 for z/OS en DB2 for LUW
- Aanbieden van mogelijkheden die door Oracle-gebruikers worden geapprecieerd

Vooral punt twee wordt expliciet genoemd door IBM bij elke nieuwe release; over punt drie wordt gezwegen, maar vooral in versie 9, en vooral op LUW, valt het toch op dat een aantal scalaire functies zijn ingevoerd (zoals b.v. `TO_CHAR` en `TO_DATE`!) die geen SQL-standaard zijn maar door Oracle-gebruikers zeer geliefd.

Versie 9 introduceert dus inderdaad ook een groot aantal nieuwe SQL-functies. Veel daarvan zijn gerelateerd aan de drie nieuwe datatypes. Maar deze keer ook opvallend veel functies die met tekst-encoding te maken hebben, dus i.h.b. met alles wat met EBCDIC, ASCII en (vooral) UNICODE te maken heeft. Blijkbaar heeft de uitdrukkelijke intrede van UNICODE (en in het bijzonder van UTF-8) in versie 8 voor een aantal verwarrende (en onverwachte?) toestanden gezorgd, die dus nu (in versie 9) rechtgetrokken worden, of in elk geval beter ondersteund, o.a. door het aanbieden van nieuwe scalaire functies.

Zo is er nu b.v. `NORMALIZE_STRING(s, NFC)` die toelaat om een tekstexpressie s in UTF-8 te converteren naar een genormaliseerde versie (dus één waarbij b.v. een letter "e" gecombineerd met een accent ´ het codepoint 233 oplevert, en niet b.v. codepoint 714 (accent) gevolgd door codepoint 101 (de letter e)). Meer details over UNICODE vindt u in Exploring DB2 Jaargang 4 Nr 4 van juni 2006.

Verder zijn er de functies `UNICODE(c)`, die het codepoint van character c teruggeeft; `ASCII(c)`, die iets gelijkaardigs doet, ten minste voor zoverre c in de systeem-ASCII-tabel voorkomt; de functies `ASCII_CHR(n)` (synoniem `CHR(n)`) en `EBCDIC_CHR(n)`, die het character teruggeven op positie n in de ASCII- resp. EBCDIC-codetabel; en de

functies `UNICODE_STR(s)`, `ASCII_STR(s)` en `EBCDIC_STR(s)`, die de tekst s expliciet "vertalen" naar UTF-8, ASCII, of EBCDIC.

Tenslotte zijn er enkele functies met functionaliteit die goed lijkt op die van reeds bestaande functies, maar waarbij nu de "code units" kunnen opgegeven worden waarmee de functie hoort te werken. Zonder te zeer in detail te gaan, hier toch alvast een smaakmaker.

Neem als voorbeeld de functie `LENGTH(s)`: zoals we weten geeft deze functie de lengte terug van tekst s. Is dat het aantal letters van s? Nee, ten minste niet indien s een UTF-8-tekst is: `LENGTH(s)` geeft het aantal *bytes* van s terug, en dat kan meer zijn dan het aantal letters vermits in UTF-8 een letter door één, twee, drie of vier bytes kan voorgesteld worden.

De nieuwe functie `CHARACTER_LENGTH(s, units)` geeft hetzij het aantal bytes (wanneer we als units de uitdrukking "OCTETS" specificeren), hetzij het aantal letter (indien we voor units "CODEUNITS32" opgeven) van s. Op dezelfde manier vervangt de functie `LOCATE_IN_STRING(s,patt [,pos[,n], units])` de "oude" `POSSTR(s, patt)`, en wordt `SUBSTR(s, startpos [,len])` vervangen door `SUBSTRING(s, startpos [, len], units)`.

Datatypes, domeinen en relationele systemen

Wanneer we de bijdragen nalezen van één van de belangrijkste grondleggers van het relationele model, Edgar Codd, zien we twee belangrijke uitgangspunten: *ten eerste*, dat een relationeel datamodel, zoals dat door DB2 zichtbaar gemaakt wordt (via SQL) niets hoeft te zeggen over de fysische implementatie die er achter zit; en *ten tweede* dat een domein meer is dan een datatype: volgens Edgar Codd dekt een domein de volledige semantiek van een datakolom. Door van een bepaald attribuut (b.v. "bedrag factuur") te zeggen dat het tot het domein "bedragen in Euro" behoort, worden zinloze operaties (zoals optellen bij bedragen in dollars, of vermenigvuldigen met andere bedragen) verboden. DB2 ondersteunt dit extreme domein-model niet automatisch; we moeten de hulp inroepen van check constraints, user-defined functions, triggers, ... om iets dergelijks tot stand te brengen. Het domein-concept blijft niettemin een belangrijk abstractie-niveau in een relationele database, zoals nog maar eens duidelijk zal worden i.v.m. het nieuwe datatype DECFLOAT; maar daarover verderop meer.

"DATE" is een mooi voorbeeld van een domein, voorgedefinieerd door DB2. Een datum is geen elementair datatype (zoals `INT` of `CHAR(n)`) maar kan eerder opgevat worden als een samengesteld type, gebouwd op drie gehele getallen (jaar, maand en dag). Deze observatie doet trouwens geen uitspraak over de fysische implementatie van het datatype DATE in DB2 (of in een andere database), helemaal in de lijn van het eerste uitgangspunt. Het enige wat vastligt is de *interface*: in DB2 kan dat op drie manieren, met name via een tekstveld (CHAR(10)) van hetzij de vorm '2009-06-26', hetzij '26.06.2009', hetzij '06/26/2009'. Iets gelijkaardigs geldt uiteraard ook voor de datatypes `TIME` en `TIMESTAMP`.

DB2 versie 9 heeft enkele nieuwe datum/tijd-gerelateerde functies toegevoegd, zoals `TIMESTAMPADD`, `VARCHAR_FORMAT`, en

TIMESTAMP_FORMAT. Deze laatste twee hebben (maar dan wel enkel op LUW) de alias-namen resp. TO_CHAR en TO_DATE meegekregen, en inderdaad: ze implementeren de in Oracle zeer populaire tekst/timestamp-conversies. Voorbeeld: de expressie
TIMESTAMP_FORMAT('26/06/2009','DD/MM/YYYY')
geeft timestamp '2009-06-26-00.00.00.000000' terug.

Vermeldenswaard is ook de nieuwe functie EXTRACT (die weliswaar op LUW al bestond in versie 8), de standaard-SQL-manier om uit een datum, tijd of timestamp de componenten (zoals maandnummer, jaar, uur, ...) te extraheren. Daarvoor gebruikten we in het verleden uiteraard de functies YEAR, MONTH, etc., maar dit zijn geen standaardfuncties. De "nieuwe" standaard-manier om b.v. het maandnummer van vandaag op te vragen is EXTRACT(month FROM current date).

SQL-standaardisatie en datatypes

Dit brengt ons terug bij één van de belangrijke drijfveren voor nieuwe functionaliteit in DB2, en in het bijzonder in versie 9: compatibiliteit met standaarden, en i.h.b. met de SQL-standaard (ANSI en ISO).

De SQL:1999 standaard specificeert de volgende datatypes:
CHAR(n), VARCHAR(n), CLOB(n), BLOB(n),
INT, SMALLINT, BIGINT, DECIMAL(n,p), REAL, FLOAT(n), DOUBLE,
DATE, TIME, TIMESTAMP, en INTERVAL.
SQL:2003 heeft daar enkel BOOLEAN aan toegevoegd, maar dat datatype werd in SQL:2008 terug afgevoerd. Tenslotte voegt SQL:2008 nog BINARY en VARBINARY toe. DB2 is dus bijna volledig standardscompliant: enkel INTERVAL (het datatype voor tijdsverschillen) bestaat niet. Anderzijds heeft DB2 geen "overbodige" datatypes (gezien vanuit de standaard), behalve dan GRAPHIC, VARGRAPHIC en ROWID.

DECFLOAT: een gloednieuw datatype

Verrassend genoeg dus nu -in versie 9- een nieuw, maar niet-standaard datatype: DECFLOAT. Of loopt DB2 hier al vooruit op een nieuwe SQL-standaard?

Ja en neen: om het nut van "DECFLOAT" en de reden van de invoering in DB2 9 te begrijpen, moeten we even stilstaan bij de interne representatie van (numerieke) data, en de standaardisatie daarvan (buiten de SQL-standaard dus).

Het uitgangspunt van Edgar Codd dat de data*betekenis* niets hoeft te zeggen over de data*representatie* is ook hier van toepassing, zij het dat zelfde betekenis met verschillende representatie gegarandeerd problemen geeft bij het kopiëren van data tussen DB2 en de applicatie (dus de programmeertaal: COBOL of C of Java of ...) Niet alleen omdat er voortdurend moet geconverteerd worden tussen representaties, maar vooral ook omdat er bij zo'n conversie vervorming kan optreden. Bij tekst-representaties kan die vervorming tegenwoordig vermeden worden door UNICODE te gebruiken; bij numerieke data van types INT en DECIMAL is er evenmin vervorming, omdat de precisie en dus de mogelijk voor te stellen getallen duidelijk afgesproken is. Ook bij DATE, TIME en TIMESTAMP geen verwarring, uiteraard.

Alleen bij “floating point”-getallen (vlottende komma) kan de discrepantie tussen betekenis (“getallen met willekeurige orde-grootte en precisie”) en representatie (die vooral beperkt wordt door de lengte van de voorstelling: meestal 8 of 16 bytes) een probleem worden.

Vlottende komma

Een concreet voorbeeld: het decimale getal 1.2 wordt door de SQL-datatypes REAL, FLOAT en DOUBLE intern voorgesteld als het binaire getal `1.001100110011`.... want inderdaad: `1/5 = 1/8 + 1/16 + 1/128 +` ... Binair kan het decimale 1.2 dus blijkbaar niet exact voorgesteld worden! Diezelfde binaire datarepresentatie wordt overigens gebruikt door ongeveer alle programmeertalen bij hun representatie van vlottende-komma-getallen.

Omdat de representatie 1.001100110011... onvermijdelijk ergens moet afbreken, blijkt dus dat het niet-gehele decimale getal 1.2 exact dezelfde representatie heeft als het getal 1.1999998 (of zoiets). Dit kan tot ongewenste afrondingseffecten leiden, iets wat zeker in een database-context ongewenst is.

Een decimale representatie zou dit “probleem” grotendeels oplossen: als we zeggen dat we het getal 1.2000000 willen opslaan, dus met expliciet 8 beduidende cijfers, dan zeggen we hierbij ook uitdrukkelijk dat dit getal verschillend is van 1.1999998, eveneens met 8 beduidende cijfers. Maar mogelijk wel gelijk aan 1.199999998 (10 beduidende cijfers), als we maar in 8 decimalen zijn geïnteresseerd.

Uit het voorbeeld blijkt alvast dat een vlottende-komma-representatie van een getal, naast de numerieke waarde, twee andere gegevens moet specificeren: *precisie* (= aantal beduidende cijfers in een gegeven grondtal, meestal hetzij 10 hetzij 2) en plaats van de decimale punt, dus orde-grootte (meestal *exponent* genoemde, zie verderop).

Floating point: representatie en IEEE-754

De meest algemene voorstelling van een floating-point-getal ziet er als volgt uit:

$(-1)^s \times c \times b^{1-p} \times b^q$

Hierbij is b de basis (meestal 10 of 2), die op voorhand wordt afgesproken. Ook p, de precisie, ligt op voorhand vast. De drie andere gegevens (s, c en q) zijn de eigenlijke variabele gegevens van het floating-point-getal:

s is de teken-bit (0 voor positieve getallen, 1 voor negatieve), c is de “coëfficiënt”, een positief geheel getal bestaande uit p cijfers (dit zijn dus de eigenlijke beduidende cijfers), en q is de (gehele) exponent (positief of negatief), die eveneens beperkt is in z’n representatielengte, laat ons zeggen w bits (w vast), d.w.z: gelegen tussen -2^{w-1} en $2^{w-1}-1$.

Voor het opslaan van zo’n getal hebben we dus 1 bit nodig voor s, w bits voor q, en p digits (grondtal b) voor s. Blijft de vraag: wat zijn goede keuzes voor b, p en w? Als iedereen dezelfde *(b,p,w)* gebruikt, lig de data-representatie en dus interpretatie ervan eenduidig vast.

In 1985 werd door IEEE de 754-standaard vastgelegd, die ondertussen door zowat alle hardware floating-point-units (FPUs), compilers

en databases is overgenomen; eigenlijk gaat het over twee standaard-keuzes voor *(b,p,w)*:

datatype REAL: *b*=2, *p*=23, *w*=8 (dus exponent *q* tussen 2^{-128} & 2^{127})

datatype DOUBLE: *b*=2, *p*=52 en *w*=11.

REAL heeft dus 1+23+8=32 bits (4 bytes) nodig, terwijl DOUBLE er 1+52+11=64 bits (8 bytes) nodig heeft. Het decimale getal 1.2 wordt door REAL voorgesteld met *s*=0, *c*=10011001100110011001100 (binair) en *q*=0.

Het belangrijkste voordeel van *b*=2 is de efficiëntere implementatie van de vermenigvuldiging, en een iets compactere representatie (minder verlies) dan bij *b*=10, maar daar staat tegenover dat een decimale representatie (*b*=10) niet voortdurend moet converteren tussen decimale representatie (gebruikersinvoer en -uitvoer) en binaire representatie (intern), en dat was nu net het probleem met de representatie van het decimale getal 1.2 uit het voorbeeld: een binaire precisie van 23 bits is (theoretisch gesproken) een decimale precisie van 6.92, dus *bijna* 7 decimalen, maar conversie van de binaire *c* met 23 bits hierboven geeft de decimale 1.1999998: weliswaar gelijk aan 1.2 na afronding op 7 decimalen, maar na vermenigvuldiging met 4 krijgen we niet 4.800000 maar wel 4.799999, en dit is onaanvaardbaar.

De IEEE-754-standaard definieert ook entiteiten “oneindig” en “ongeldig getal” (met name: `+Inf`, `-Inf`, `NaN`, `sNaN`) en de manieren om deze getallen binnen de genoemde 32 resp. 64 bits op te slaan. Zo is `+Inf` het resultaat van de deling `+1/0`, terwijl `0/0` als resultaat `NaN` geeft.

De IEEE-754-standaard werd in augustus 2008 uitgebreid met de volgende drie gevallen, zie b.v. *http://speleotrove.com/decimal/*:

b=2, *p*=112, *w*=15, dus expmax=16383 (“long double”, 128 bits),

b=10, *p*=16, expmax=383 (“decimal float”, 64 bits)

b=10, *p*=34, expmax=6143 (“long decimal float”, 128 bits)

IBM was één van de drijvende krachten achter deze recente standaardisatie, en is op dit ogenblik ook de enige die reeds hardware-acceleratie heeft voor de twee nieuwe “decimal float”-datatypes, met name met hun z9, z10 en Power6. Ondertussen is er ook al softwareondersteuning door de gcc-compiler 4.2, door alle IBM-compilers, en door DB2 9 (alle platformen, ook indien geen HW-ondersteuning).

DECFLOAT(16) en DECFLOAT(34)

Inderdaad: de twee nieuwe datatypes in DB2 9 zijn de twee nieuwe IEEE-754-datatypes met *b*=10. “DECFLOAT” is equivalent met “DECFLOAT(34)”. De parameter (16 of 34) verwijst uiteraard naar het aantal beduidende (decimale!) cijfers van het datatype. Alhoewel dus ook de “range” van de exponent (dus de mogelijke plaatsen van de decimale punt) belangrijk is: hoogstens 383 voor DECFLOAT(16), en hoogstens 6143 voor DECFLOAT(34). Anders gezegd: het grootste getal dat kan voorgesteld worden in DECFLOAT is net geen 10^{6144}, terwijl het kleinste strikt positieve getal 10^{-6144} is.

In DB2 hebben we nu dus twee nieuwe datatypes DECFLOAT(34) en DECFLOAT(16), waarvan we ondertussen de interne representatie

kennen, maar hoe communiceert DB2 met de buitenwereld voor deze datatypes? SQL-numerieke data schiet hier tekort, want hoe specificeren we b.v. het getal -1.23×10^{99} ?

Net zoals voor het communiceren van waarden voor de datatypes DATE, TIME en TIMESTAMP gebruikt DB2 hiervoor tekst-constanten (dus single-quote-delimited), met daarin de gebruikelijke "scientific notation", in de "constructor" van DECFLOAT; dus b.v.:

```
CREATE TABLE t ( p INT NOT NULL, d DECFLOAT ) ;
INSERT INTO t(p,d) VALUES (1, 123.45) ;
INSERT INTO t(p,d) VALUES (2, -1000) ;
INSERT INTO t(p,d) VALUES (3, decfloat('123e99')) ;
INSERT INTO t(p,d) VALUES (4, CAST('-123e-99' AS decfloat)) ;
INSERT INTO t(p,d) VALUES (5, decfloat('-Inf')) ;
```

Alhoewel we in het voorbeeld 123e99 slechts 3 decimalen opgeven, worden er toch (intern) 34 decimalen opgeslagen. Toch houdt DB2 expliciet bij dat we slechts 3 decimalen precisie wensen: de getallen 123e99 en 123.0e99 zijn dus bewust "verschillend", alhoewel uiteraard de operator "=" deze getallen als "numeriek identiek" zal bestempelen. Het kunnen specificeren van het aantal beduidende cijfers heeft o.a. als gevolg dat ook de manier van afronden volledig voorspelbaar is: met DECFLOAT is het dus onmogelijk dat 4*1.2 = 4.7999998. Wanneer we b.v. vragen naar het resultaat van 123e99 x 9 in drie beduidende cijfers, krijgen we als antwoord 1.11e102 (voor afronding was dat 1.107e102). De manier van afronden kan trouwens expliciet gespecificeerd worden, maar daar gaan we hier niet verder op in.

DECFLOAT-gerelateerde functies

`NORMALIZE_DECFLOAT(f)`

deze functie "normaliseert" een decfloat-getal door de precisie zo klein mogelijk te maken, en exact 1 cijfer voor de decimale punt te plaatsen; b.v.:
`normalize_decfloat('56.7800e+8') = '5.678e+9'`

`QUANTIZE(f, q)`

rondt het decfloat-getal f af op de precisie en met de exponent van het decfloat-getal q. Dus b.v.
`quantize('56.7820e+8','100e7') = '568e+7'`, en
`quantize('56.7820e+8','100.00000e7') = '567.82000e+7'`

`COMPARE_DECFLOAT(f1, f2)`

geeft als antwoord 0 indien f1 en f2 intern identiek worden voorgesteld, 1 indien f1 < f2, 2 indien f1 > f2, en 3 indien f1 en f2 niet vergelijkbaar zijn, met name omdat één van beide `NaN` is. B.v.:
`compare_decfloat('Inf',3.141592)=2`.

`TOTALORDER(f1, f2)`

Dit is *de facto* de functie die impliciet door ORDER BY (en ook door indexen) gebruikt wordt: alle getallen zijn vergelijkbaar, met '`Inf`' < '`NaN`' en b.v. ook '1.0' < '1.00'. De functie geeft 0 als antwoord indien de interne representaties van f1 en f2 identiek zijn, -1 indien f1 < f2, en 1 indien f1 > f2.

Cognos, an IBM Company (deel 2)

Kris Van Thillo (ABIS)

zie: Jaargang 5 Nr 3, juni 2008.

In een vorig artikel stonden we reeds even stil bij het waarom van de overname van Cognos door IBM. In dit artikel willen we een typische Cognos 8 architectuur voorstellen. Bedoeling is zeker niet volledig te zijn: we wensen juist in tegendeel een inzicht te bieden in het globale plaatje. Details, optionele componenten en configuratiemogelijkheden en -alternatieven vallen buiten de scope van dit artikel.

Uitgangspunt is Cognos 8 voor web-based BI.

Architectuurcomponenten

Volgende componenten maken globaal deel uit van een traditionele Cognos 8 architectuur - we doen een poging deze componenten logisch geordend aan te kaarten.

Web-based Cognos communicatie wordt opgezet via web-server gateways; de *Cognos 8 gateway* is verantwoordelijk voor o.a. de communicatie tussen Cognos 8 componenten en services (o.a. de Cognos 8 dispatcher) en een standaard web browser. Deze wordt typisch geïnstalleerd op één of meer webservers. Ondersteuning voor o.a. CGI (default), ISAPI (Microsoft IIS server), Apache_mod (Apache webserver), servlet (Java EE application server) standaarden is beschikbaar.

De Cognos 8 Cognos *Connection* component is een portal die dienst doet als een 'single-port-of-call' voor het merendeel van de ontwikkel en configuratieacties eigen aan een Cognos 8 omgeving. Deze portal wordt bijvoorbeeld ter beschikking gesteld middels een applicatieserver (JVM container geladen in één van de ondersteunde Java EE applicatiesevers). Een aantal 'services' maken deel uit van deze portlet - de boven aangehaalde dispatcher service is daar één van. Men spreekt in deze context van de Cognos 8 Application Tier Components.

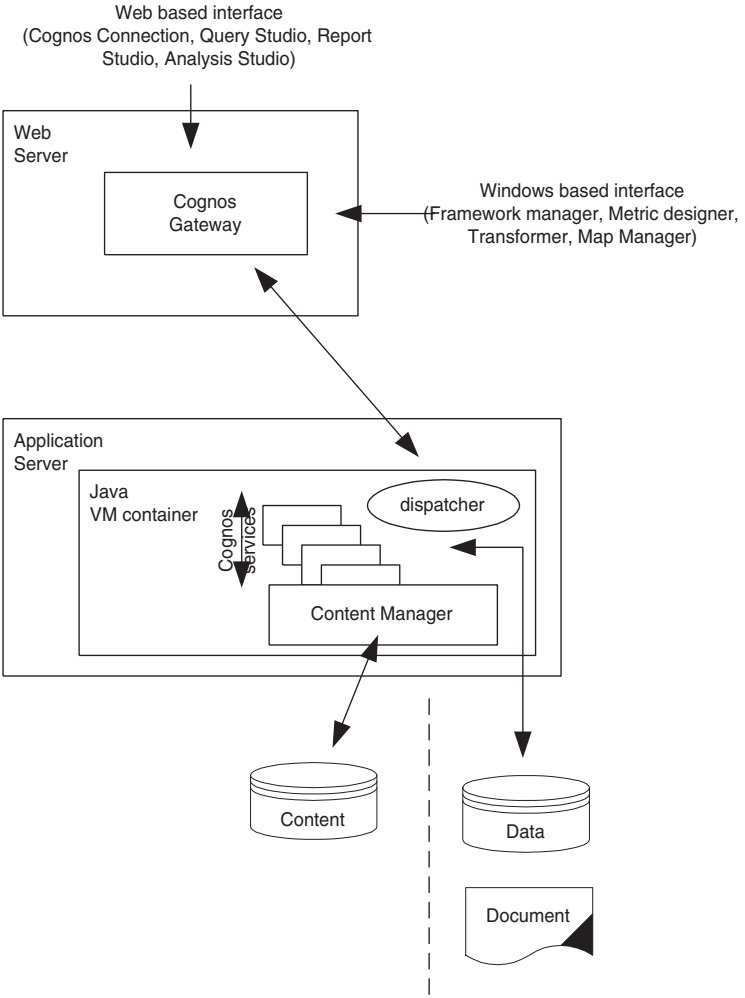
De Cognos 8 *Content Manager* component is verantwoordelijk voor het beheren en opslaan van configuratiedata, rapportdefinities en specificaties, rapport output, modellen, ... Een belangrijk onderdeel van deze component is de Access Manager. Opslag gebeurt standaard in een *Content Store* of een *Content Database*. Alhoewel niet helemaal correct kan je deze database beschouwen als een soort 'meta-data' database: de echte 'te-analyseren-data' bevindt zich inderdaad elders (bijvoorbeeld, in een andere relationele database). Als content Store wordt standaard Apache Derby weerhouden.

De data - bron en doel van de analyse - kan zich in een hele reeks van 'bronbestanden' bevinden: relationele databases, multidimensionele databases/cubes, files, XML, ... Vaak wordt gerefereerd naar de 'Report/Query Database' omgeving.

Een aantal andere componenten verdienen kort onze aandacht:

- Cognos 8 *Viewer* biedt de mogelijkheid alle middels Cognos 8 aangemaakte rapporten, analyses, ... te publiceren via het web;
- Cognos 8 *Analysis Studio*, *Query Studio*, and *Report Studio* behoren tot de typische set analyse en rapportage tools, elk gericht op een specifiek BI 'doelpubliek'. Zoals verwacht richt *Analysis Studio* zich eerder op de experts (OLAP tools, dimensionele analyse); *Reporting Studio* wordt eerder gebruikt door power users met omvangrijke business en reporting kennis; *Query Studio* richt zich eerder tot eindgebruikers.

Cognos 8 componenten - Overzicht.



Achtergrondinformatie

- De Cognos 8 Gateway wordt typisch geïnstalleerd op meerdere webservers (fail-over) - de IBM HTTP Webserver kan uiteraard worden gebruikt (web tier component).
- De volgende services maken o.a. deel uit van de Cognos 8 Application Tier: de Presentation Service, Job Service, Delivery Service, Content Manager Service, Interactive/Batch Report Service, Monitoring service, ... Vanaf de gateway kunnen requests naar verschillende dispatchers worden doorgerouted (dispatcher list) voor redenen van beschikbaarheid, indien de 'preferred' dispatcher niet beschikbaar zou zijn. Dispatchers kunnen requests zowel naar 'lokale' als 'remote' applicatie tier services doorsturen - bijvoorbeeld om redenen van load-balancing.

  De IBM WebSphere Application Server kan als standaard application server worden ingezet (application tier component).
- Cognos 8 Content Manager.
  - Eén, of meerdere Content Managers kunnen worden geconfigureerd voor een typische Cognos 8 omgeving. Indien meerdere Content Manager worden aangemaakt, is er slecht één actief. De andere krijgt de status van 'standby' Content Manager; en wordt pas actief op het moment van het falen van de eerste. Deze applicatie tier component kan mogelijk in een afzonderlijke applicatie server worden geïnstalleerd.
  - De Content Manager vereist tevens de beschikbaarheid van een relationele database als data store. Naast de standaard Apache Derby database bestaat ondersteuning voor: Oracle, DB2, Microsoft SQL Server, Sybase. Niet voor elke database wordt gebruik gemaakt van een thin JDBC client-connectie; voor DB2 is vaak een lokale DB2 client-installatie nodig op de Content Manager Service node. Vaak wordt trouwens aangeraden op deze node een DB2 UDB database server te configureren. Database beheer (b.v. backup) wordt niet door Cognos 8 service uitgevoerd, en behoort dus tot de taken van een database DBA.
  - De Access Manager die deel uitmaakt van de Content Manager is verantwoordelijk voor user identificatie, validatie, encryptie, etc. Voor een aantal van deze taken wordt standaard beroep gedaan op externe diensten (eg. user identificatie en validatie); andere diensten zoals bijvoorbeeld het beheer en authenticatie van certificaten wordt door Cognos 8 zelf verricht, maar kan door een reeks API's geëxternaliseerd worden.
- In functie van de aard van de 'Report/Query' database dient op de Cognos 8 omgeving de nodige client software geïnstalleerd worden; afhankelijk van de concrete situatie wordt een Java thin, Java OCI, of zelfs een ODBC connectie opgebouwd (e.e.a. kan dus bijkomende configuratie vereisen). Op basis van Enterprise Information Integration (EII) kunnen daarnaast een veelheid aan andere databronnen op een geïntegreerde wijze (Enterprise-wide view) worden geconsulteerd - bijvoorbeeld gebruik makende van LDAP, Open XML, of WSDL.

Tot slot

Over dit alles kan natuurlijk veel meer worden gezegd en geschreven - meer informatie, omtrent een hele reeks andere Cognos 8 componenten, configuratie alternatieven... zijn te vinden op de IBM website.

CURSUSPLANNING SEPT - DEC 2009

DB2 concepten	450 EUR	op aanvraag
DB2 for z/OS, een totaaloverzicht	2025 EUR	05.10(L), 12.10(W), 30.11(L), 14.12(W)
DB2 UDB for LUW, totaaloverzicht	2025 EUR	05.10(L), 14.12(W)
RDBMS-concepten	375 EUR	03.09(L), 05.10(L), 12.10(W), 30.11(L), 14.12(W)
Basiskennis SQL	375 EUR	04.09(L), 06.10(L), 13.10(W), 01.12(L), 15.12(W)
DB2 for z/OS basiscursus	1275 EUR	07.10(L), 19.10(W), 07.12(L), 16.12(W)
DB2 UDB for LUW basiscursus	1275 EUR	07.10(L), 16.12(W)
SQL-QMF voor eindgebruikers	1275 EUR	28.10(W)
SQL workshop	800 EUR	22.10(L), 02.11(W), 21.12(L)
SQL voor gevorderden	450 EUR	29.10(L), 23.11(W)
DB2 SQL PL, triggers, stored procedures	450 EUR	30.10(L), 24.11(L)
DB2 for z/OS programmeren voor gevorderden	900 EUR	09.11(L), 10.12(W)
DB2 for z/OS: SQL performance	1350 EUR	26.10(L), 07.12(W)
XML in DB2	450 EUR	20.11(L), 07.12(W)
DB2 for z/OS database administratie	1900 EUR	14.09(W), 16.11(L)
DB2 for z/OS operations & recovery	1425 EUR	16.09(UK), 05.10(W), 18.11(UK)
DB2 for z/OS systems performance and tuning	1000 EUR	08.09(W), 16.11(UK)
DB2 LUW DBA - Kernvaardigheden	1800 EUR	19.10(L), 05.12(W)
DB2 v8 upgrade, DB2 9 upgrade	450 EUR	op aanvraag
Data warehouse concepten	450 EUR	25.09(L), 02.11(W)
SQL voor BI ***(nieuw)***	450 EUR	19.10(L), 12.11(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
training@abis.be

abis
TRAINING & CONSULTING

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

Bijlage

Optimalisatie: doorzoeken van string

Timót Mechler (ABIS/Haute École de Bruxelles)

Het gestructureerd doorzoeken van een database op namen (strings) - bedrijfsnamen, namen van personen - is niet zo eenvoudig als wat men in eerste instantie zou denken. In dit artikel staan we stil bij een aantal uitdagingen in deze context; en gaan we na hoe DB2 voor z/OS ons hierbij behulpzaam kan zijn.

Voorbeeld: de tabel companies

```
CREATE TABLE companies
(cono    SMALLINT NOT NULL,
 coname  VARCHAR(45));

SELECT * FROM companies;

 CONO  CONAME
  1    De Hertog
  2    menestra&co -finance-
  3    K.R.G.C.
  4    sanséau
```

Applicatie (COBOL met SQL):

```
SELECT cono,  coname
INTO   :var1, :var2
FROM   companies
WHERE  coname LIKE :varInput
```

Context en voorbeeld

Een applicatiegebruiker wenst op zoek te gaan naar specifieke bedrijfsgegevens, op basis van een bedrijfsnaam. Deze bedrijfsnaam dient dus als zoekfilter te worden gehanteerd. Om de gezochte resultaten te vinden, dient het zoekcriterium enerzijds, en de waarde opgeslagen in de database anderzijds, identiek te zijn - en daar wringt meestal het schoentje.

Zo zal de zoekstring ‘Dehertog’ in bovenstaand voorbeeld geen resultaten opleveren (NULL resultaat): niet enkel het niet respecteren van hoofdletters/kleine letters is hier een probleem; ook de ontbrekende spatie tussen beide woorden in de bedrijfsnaam zorgt hier voor.

Ook zoekopdrachten zoals ‘menestra&co (finance)’, ‘KRGC’ en ‘sanseau’ zullen lege resultaatsverzamelingen opleveren!

Het doorzoeken van strings in de database is met andere woorden standaard aan één eenvoudige regel onderworpen: de zoekstring en de waarde in de database moeten gewoon volledig identiek zijn!

Optimalisatie

Een belangrijke optimalisatiestap bestaat er dus in om bij het zoeken naar strings in een database op basis van een filter geen rekening te hoeven houden met hoofdletters/kleine letters, extra spaties/blanco's, en/of speciale lettertekens.

Mogelijk kan men overwegen deze uitdaging aan te pakken middels het gebruik van bestaande scalaire functies als UPPER en TRANSLATE. Deze laten inderdaad toe op het moment van doorzoeken van de database alle 'niet-reguliere' tekens, spaties, ... weg te nemen. Belangrijkste nadeel is natuurlijk dat dit enerzijds dient te gebeuren voor elke rij die wordt doorzocht (met aanzienlijke efficiëntienadelen naarmate het aantal rijen in de tabel toeneemt); maar ook, dat dit elke keer opnieuw moet gebeuren!

De techniek die we in wat volgt graag voorstellen is zeer efficiënt en rechttoe/rechtaan, en zeer makkelijk toe te passen in de context van relationele databases die triggers ondersteunen.

In eerste instantie dient aan de boven beschreven `companies`-tabel een kolom te worden toegevoegd die de 'gecorrigeerde' string zal bevatten: de originele string gezuiverd van spaties, speciale leestekens, hoofdletters/kleine letters, en eventueel ook accentletters; een *genormaliseerde* versie van de bedrijfsnaam dus. Dit dient eveneens te gebeuren op het niveau van de applicatie die de zoekstring naar de database doorstuurt. Het opzoeken van de detailgegevens zal dienen te gebeuren gebruik makende van deze 'gecorrigeerde' data.

Techniek

Volgende wijzigingen dienen aan de database te worden aangebracht (tabel *companies* in ons voorbeeld):

- Een nieuwe kolom dient te worden voorzien voor elke tekstkolom die moet kunnen doorzocjt worden. In deze kolom wordt alle tekst in hoofdletters opgenomen; spaties/blanco's worden weggehaald; en letters met speciale tekens (b.v. accenten, trema, ...) worden vervangen door hun "genormaliseerde" equivalent, zonder speciale tekens dus.

Voorbeeld: aanmaken van een nieuwe kolom

```
ALTER TABLE companies
  ADD COLUMN  conamex  VARCHAR(45);

CREATE INDEX compidx1 ON companies (conamex ASC) CLUSTER  ;

UPDATE companies SET conamex =
      REPLACE(UPPER(TRANSLATE(coname,
      'AAAAAAAAEEEEIIIIOOOOOOUUUUYYCNAAAAAAAAEEEEIIIIOOOOOOUUUUYCN ',
      'áàâäãåæéèêëíìîïóòôöõøúùûüýÿçñÁÀÂÄÃÅÆÉÈÊËÍÌÎÏÓÒÔÖÕØÚÙÛÜÝÇÑ'||
      '-_%()''.,:;[]/\+<>{}!?|°*$£µ`~=^',' ')),' ','') ;

SELECT * FROM companies
```

CONO	CONAME	CONAMEX
1	De Hertog	DEHERTOG
2	menestra&co -finance-	MENESTRACOFINANCE
3	K.R.G.C.	KRGC
4	sanséau	SANSEAU

- Om de efficiëntie van het doorzoeken van deze nieuwe kolom te verhogen, wordt op deze kolom een index geplaatst.
- In de meeste situaties zal de bedoelde tabel reeds data bevatten; de toegevoegde kolom moet dus worden opgevuld met de nodige startwaarden.

Belangrijkste onderdeel van deze strategie is natuurlijk het 'up-to-date' houden van deze nieuwe kolom, telkens waneer bestaande waarden in de originele tabelkolom worden bijgewerkt, dan wel rijen aan deze tabel worden toegevoegd. Hiertoe kunnen natuurlijk triggers worden gebruikt, zowel "after insert" als "after update".

Voorbeeld: aanmaken van trigger

```
CREATE TRIGGER compnsrt
   AFTER INSERT ON companies
   REFERENCING NEW AS n
   FOR EACH ROW MODE DB2SQL
   UPDATE companies SET conamex =
            REPLACE(UPPER(TRANSLATE(coname,
          'AAAAAAAEEEEIIIIOOOOOOUUUUYYCNAAAAAAAEEEEIIIIOOOOOOUUUUYCN ',
          'áàâäãåæéèêëíìîïóòôõøúùûüÿýçñÁÀÂÄÃÅÆÉÈÊËÍÌÎÏÓÒÔÕØÚÙÛÜÝÇÑ'||
            '-_%()''.,:;[]/\+<>{}!?|°*$£µ`~=^',' ')),' ','')
   WHERE cono = n.cono;

CREATE TRIGGER compudpt
   AFTER UPDATE OF coname ON companies
   REFERENCING OLD AS o NEW AS n
   FOR EACH ROW MODE DB2SQL
   UPDATE companies SET conamex =
            REPLACE(UPPER(TRANSLATE(coname,     <etc>;
```

Ook de applicatie zelf dient te worden aangepast - met name, de inhoud van de variabele 'varInput' (in ons voorbeeld) moet globaal gesproken de hierboven besproken wijzigingen ondergaan. Er zijn evenwel 2 uitzonderingen:

- spaties/blanco's dienen niet te worden weggehaald, maar vervangen door het '%' symbool (te gebruiken in de context van de SQL LIKE instructie);
- het symbool '%' dient te worden toegevoegd aan het eind van de (zoek)string om ons toe te laten slechts een beperkt deel van de zoekstring op te geven.

Voorbeeld: de applicatie (COBOL met SQL)

```
SELECT cono,  coname
INTO   :var1, :var2
FROM   companies
WHERE  conamex LIKE :varInput
```

En dus wordt het nu mogelijk de gegevens van bedrijf 'De Hertog' op te zoeken op basis van bijvoorbeeld volgende zoekstrings: DeHertog, d'ehertog, de hertog, deHértög, Hertog, de tog. En dit natuurlijk ook, als in de tabel de naam van het bedrijf anders, met spaties en/of accenten, werd geschreven.

Het spreekt voor zich dat deze oplossing mogelijk eigenaardige bijwerkingen kan hebben (b.v. wat betreft de inhoud van de opgeleverde data, of de sorteervolgorde van de opgeleverde lijsten).

DB2 9 for z/OS

Indien gebruik wordt gemaakt van DB2 versie 9 kan het gebruik van een 'index on expression' in deze situatie een nuttig alternatief zijn. Deze specifieke index ('index on expression' in DB2, 'functie index' in Oracle en PostgreSQL) biedt ons een aantal voordelen: de tabel moet niet worden aangepast (geen kolom toegevoegd), en ook triggers zijn totaal overbodig.

Voorbeeld: aanmaken van een index in DB2 9

```
CREATE INDEX compindx
  ON companies
  ( REPLACE(UPPER(TRANSLATE(coname,
      'AAAAAAAEEEEIIIIOOOOOOUUUUYYCNAAAAAAAEEEEIIIIOOOOOOUUUUYCN ',
      'áàâäãåæéèêëíìîïóòôöõøúùûüÿýçñÁÀÂÄÃÅÆÉÈÊËÍÌÎÏÓÒÔÖÕØÚÙÛÜÝÇÑ'||
      '-_%()''.,:;[]/\+<>{}!?|º*$£µ`~=^',' ')),' ','') ) ;

SELECT  cono,  coname
INTO    :var1, :var2
FROM    companies
WHERE   coname LIKE :varInput
```

Het relationele database-systeem houdt automatisch de index bij; elke wijziging aan de waarden in de bedoelde kolom geeft aanleiding tot het aanpassen van de index. In de applicatie zelf verwijst de query nu opnieuw naar de basiskolom.