ABIS NIEUWSBRIEF VOOR DB2 PROFESSIONALS

JAARGANG 2 - NUMMER 6, MAART 2004

OPEN CURSOR

Op 26 maart 2004 is het zover: DB2 for z/OS versie 8 zal formeel boven de doopvont worden gehouden. Zo kunt u zeer binnenkort de nieuwe features, waar u zo naar uitkijkt, toepassen!
Features waar we u reeds vanaf het begin van jaargang 2 van Exploring DB2 via 'Dossier 8' van op de hoogte brengen.
Natuurlijk blijven we in dit en volgende nummers ook aandacht besteden aan features die relevant zijn in v6 en/of v7 - voor de meesten van ons immers nog de 'productieversie'.
In dit nummer komen onderwerpen aan bod die zowel applicatieontwikkelaars als beheerders zullen intereseren. Ook al worden onderwerpen aangehaald die nog niet door elke DB2-site actief worden gebruikt.
Veel leesgenot!

Het ABIS DB2 team.

IN DIT NUMMER:

- Na een maand in spanning te hebben gezeten - hier is het antwoord op de vraag 'Waar zijn mijn cursors heen?' - in *DB2 Data Provider verenigt .NET-client.*
- Gebruikt u reeds LOBs? Lees dan zeker dit artikel: *Een buitenbeetje - LOBs en space management!*
- Dossier 8 - bent u op zoek naar *Extra veiligheid*?
- *Cursusplanning maart 2004 - juni 2004.*

CLOSE CURSOR

Het volgende nummer staat volledig in het teken van Web services - na het Java-nummer verleden jaar, opnieuw een special!

Tot dan!

DB2 Data Provider verenigt .NET-clients - 3

Katrien Platteborze (ABIS)

Inleiding

In vorige artikels hebben we de algemene ADO.NET-architectuur belicht, de IBM DB2 Data Provider die specifieke DB2-klasses toevoegt aan ADO.NET en de IBM add-ins voor Visual Studio 2002. In dit artikel gaan we specifieker in op de data access naar DB2 toe. We gaan het hebben over cursors, transacties en optimistic concurrency.

Waar zijn mijn cursors?

We vallen meteen met de deur in huis: er zijn geen klassieke updateable server-side cursors. Wat is er dan wel?

Zie *DB2 Data Provider - c#-code voorbeeld 1* op p. 13.

De DB2DataReader-klasse

De DB2DataReader-klasse voorziet in een forward-only, read-only, server side cursor. Met behulp van een object van deze klasse beschikt men in de applicatie telkens over 1 rij van het resultaat van een query en kan men een volgende rij aanvragen. De query die uitgevoerd wordt, is gedefinieerd in een property van een db2Command-object. Een DB2DataReader-object wordt verkregen door de ExecuteReader()-methode van het db2Command-object uit te voeren. Met behulp van de read()-methode positioneert men zich op de volgende rij. Met een aantal get()-methodes kan men de waarden van de verschillende kolommen in het resultaat ophalen. Zolang de DB2DataReader records ophaalt, blijft de connectie open - deze is niet herbruikbaar totdat men expliciet de db2DataReader sluit, en daarna de connectie sluit.

De DataSet en de DB2DataAdapter-klasse

De programmeur kan ook gebruik maken van een DataSet en de DB2DataAdapter-klasse. De DataSet is een in-memory structuur op de client die records van één of meerder datasources bevat. Belangrijk is dat we niet record per record werken. Alle records zijn tegelijkertijd beschikbaar op de client. De DataSet is volledig onafhankelijk van de datasource. Eens de records opgehaald, is elke herinnering aan de datasource verdwenen. Met de gegevens in de DataSet werkt men zonder connectie naar de datasource. De gebruiker kan de waarden in de DataSet aanpassen en terug uitvoeren op de database. Hiervoor wordt de connectie terug geopend.

De DataSet-gegevens kan men op twee manieren benaderen: ofwel als tabellen in een relationele database waarbij men ook gebruik kan maken van constraints die door de DataSet gecontroleerd worden, ofwel als XML: de data wordt getransporteerd en gepersisteerd als XML.

De DB2DataAdapter fungeert als brug tussen de datasource en de DataSet, zowel voor het ophalen van de gegevens als voor het uitvoeren van de aanpassingen in de database. Dankzij de Fill()-methode van een db2DataAdapter-object voert deze db2DataAdapter het select statement van een geassocieerd db2Command uit. Hiervoor wordt intern een db2DataReader geïnitialiseerd en gebruikt. Met de Update()-methode voert de db2DataAdapter de gemaakte aanpassingen aan de DataSet uit in de database (Figuur 1). Voor elke aanpassing aan de DataSet - dit kunnen updates, deletes en inserts zijn - voert de db2DataAdapter een SQL statement uit op basis van db2Command-objecten waar naar verwezen wordt op de DeleteCommand, UpdateCommand, en InsertCommand properties van de db2DataAdapter zelf. In deze db2Command-objecten staat het uit te voeren statement hard gecodeerd als CommandText property - weliswaar niet met waarden maar met achteraf in te vullen parameters.

Figuur 1: Statements gedefinieerd op de db2Command-objecten

```
INSERT INTO  "TB"."COMP" ("NO", "NAME", "STRT", ...)
VALUES (?, ?, ?, ...)

UPDATE "TB"."COMP"
SET "NO" = ?, "NAME" = ?, "STRT" = ?, ...
WHERE ("NO" =  ?) AND ("NAME" =  ?) AND ("STRT" =  ?) AND ("STRNO" =  ?
OR CAST(? as VARCHAR(10)) IS NULL AND "STRNO" IS NULL)  AND ...

DELETE FROM "TB"."COMP"
WHERE ("NO" =  ?) AND ("NAME" =  ?) AND ("STRT" =  ?) AND ("STRNO" =  ?
OR CAST(? as VARCHAR(10)) IS NULL AND "STRNO" IS NULL) AND ...
```

Het is echter ook mogelijk om met een db2CommandBuilder-object te werken die zelf de update, insert en delete statements genereert op basis van het select statement. Hiertoe moet wel metadata opgehaald worden - de gegenereerde update statements zijn lichtjes verschillend.

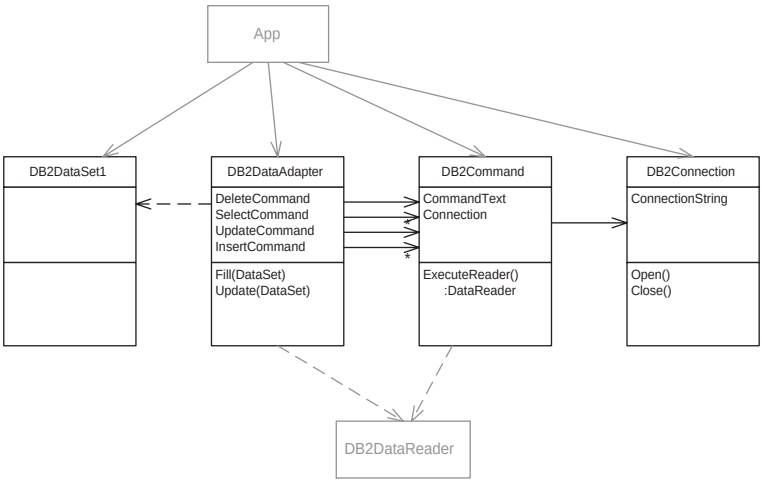
In een applicatie kan de DataSet niet rechtstreeks getoond worden aan de gebruiker. Dit gebeurt via een datagrid-object afgeleid van de System.Windows.Forms.DataGrid-klasse. Het is ook in deze datagrid dat de gebruiker zijn aanpassingen uitvoert.

In het klassediagram (figuur 2) ziet u het verband tussen de verschillende klasses. Wanneer we dit toepassen creëren we een applicatie die met een druk op de knop de gegevens ophaalt en met een druk op een andere knop gegevens terugbrengt naar de database. De interface voor de gebruiker ziet u in figuur 3; de code die uitgevoerd wordt bij een druk op de knop in figuur 4 - button1_Click.

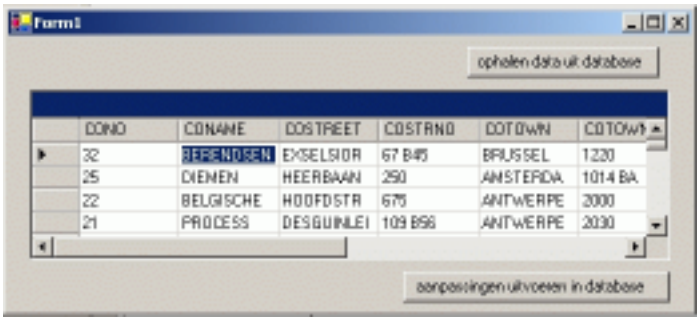
Zie *DB2 Data Provider - c#-code voorbeeld 2* op p. 15.

Uiteraard kan de programmeur de interface uitbreiden met textboxes waar de gebruiker waarden voor where-condities invult zodanig dat de resultset beperkt blijft.

Figuur 2: Klassediagram



Figuur 3: Voorbeeld interface



Figuur 4: Code uitgevoerd bij het optreden van button_Click-events (1)

```
private void button1_Click(object sender, System.EventArgs e)
{
 db2DataSet11.Clear();
 db2DataAdapter1.Fill(db2DataSet11);
}

private void button2_Click(object sender, System.EventArgs e)
{
 db2DataAdapter1.Update(db2DataSet11);
}
```

Aandachtspunten

Bij deze manier van werken horen een aantal bedenkingen.

De volgorde waarin de db2DataAdapter de gegenereerde SQL statements uitvoert in de database staat niet vast. Dit leidt tot problemen wanneer de volgorde belangrijk is; bijvoorbeeld bij updates en inserts van de PK, updates van child-parent tabellen. Er is programmatorische controle mogelijk.

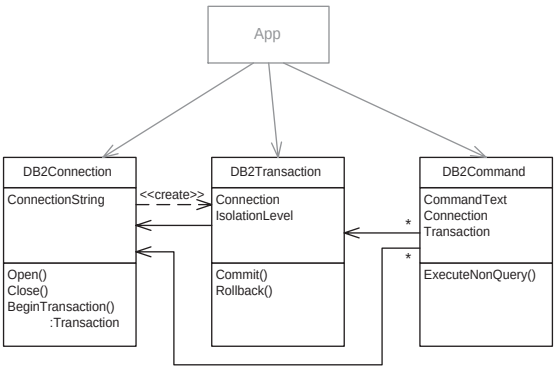
Eén uitgevoerd SQL statement is een impliciete transactie op zichzelf; de uitvoer stopt wanneer een fout optreedt. Dit betekent dat men niet zeker weet wat uitgevoerd is en wat niet, ook en vooral omdat de volgorde van uitvoering niet gekend is. Expliciete transacties bieden hier een oplossing. Men kan ook de code aanpassen zodanig dat toch alle statements uitgevoerd worden - behalve datgene dat de fout veroorzaakte - en waarbij de gebruiker een verwittiging krijgt.

Terwijl de gebruiker aanpassingen uitvoert in een DataSet zijn er geen locks in de database. Ondertussen zou dus eenzelfde record door iemand anders kunnen aangepast zijn. Optimistic concurrency control biedt een oplossing en is standaard ingewerkt in DB2 - ADO.NET applicaties.

Over transacties

Om een expliciete transactie te gebruiken moet men een db2Transaction-object koppelen aan de statements die behoren tot de transactie. Dit doet men via het transaction property van een db2Command-object. De BeginTransaction()-methode van een db2Connection-object maakt een nieuwe transactie aan. Men kan tegelijkertijd een IsolationLevel meegeven. Men moet wel een .NET isolation level gebruiken. Figuur 5 geeft een klassediagram weer.

Figuur 5: Klassediagram



Wanneer alle aanpassingen aan een DataSet uitgevoerd moeten worden als één transactie, dan ziet het button2_click event er als volgt uit - zie figuur 6. Als het misloopt in de try block - db2 gerelateerd -

wordt er een exception gegooid die in de catch blok opgevangen wordt, waarin een rollback voorkomt. Let ook op het expliciet openen en sluiten van de connectie.

Figuur 6: Code uitgevoerd bij het optreden van button_Click events (2)

```
private void button2_Click(object sender, System.EventArgs e)
{
 try
 {
   db2Connection1.Open();
   db2Transaction1 = db2Connection1.BeginTransaction();
   db2UpdateCommand1.Transaction = db2Transaction1;
   db2InsertCommand1.Transaction = db2Transaction1;
   db2DeleteCommand1.Transaction = db2Transaction1;
   db2DataAdapter1.Update(db2DataSet11);
   db2Transaction1.Commit();
   db2Connection1.Close();
 }
 catch (IBM.Data.DB2.DB2Exception exep)
 {
   MessageBox.Show(exep.Message, exep.GetType().ToString());
   db2Transaction1.Rollback();
   db2DataAdapter1.Fill(db2DataSet11);
   db2Connection1.Close();
 }
}
```

De mapping tussen de .NET en DB2 isolation levels wordt in onderstaande tabel weergegeven.

.NET	DB2
ReadCommitted	Cursor Stability
ReadUncommitted	Uncommitted Read
RepeatableRead	Read Stability
Serializable	Repeatable Read

Zonder expliciete transacties

Standaard stopt een applicatie als er een fout optreedt. Opdat dit niet het geval zou zijn, kan men het ContinueUpdateOnError property van het db2DataAdapter-object op true zetten. Het nadeel van deze techniek is dat er ook helemaal niet opgemerkt wordt dat er iets misgelopen is; er kan dan ook niet gepast gereageerd worden.

Indien men dit toch wenst, biedt het RowUpdated event een oplossing: men kan code voorzien die wordt uitgevoerd na elk SQL statement (update, insert of delete) in de database. Men kan dan per rij nagaan of het SQL statement goed is uitgevoerd, en indien niet, eventueel gepast reageren (Figuur 7).

Andere mogelijkheid: de status van de rij aanpassen. Na een fout staat deze op 'ErrorsOccurred'. De volgende rijen worden afgewerkt als de status van de fout veroorzakende rij op 'SkipCurrenRow' gezet wordt (Figuur 8).

Figuur 7: Versie 1 - ContinueUpdateOnError property van het db2DataAdapter-object

```
this.db2DataAdapter1.ContinueUpdateOnError = true;
this.db2DataAdapter1.RowUpdated += new
IBM.Data.DB2.DB2RowUpdatedEventHandler(this.db2DataAdapter_RowUpd);

private void db2DataAdapter_RowUpd(object sender,
IBM.Data.DB2.DB2RowUpdatedEventArgs e)
{
   //if (e.Status == UpdateStatus.ErrorsOccurred)   !!alternatief!!
   if (e.RecordsAffected == 0)
   {
      e.Row.RowError = "Fout";
      MessageBox.Show("in RowUpdatedevent : " + e.Errors.ToString());
   }
}
```

Figuur 8: Versie 2 - UpdateStatus

```
this.db2DataAdapter1.RowUpdated += new
IBM.Data.DB2.DB2RowUpdatedEventHandler(this.db2DataAdapter_RowUpd);

private void db2DataAdapter_RowUpd(object sender,
IBM.Data.DB2.DB2RowUpdatedEventArgs e)
{
    if (e.RecordsAffected == 0)
    {
     e.Row.RowError = "Fout";
     e.Status = UpdateStatus.SkipCurrentRow
     MessageBox.Show("in RowUpdatedevent : " + e.Errors.ToString());
    }
}
```

Optimistic concurrency

De gegenereerde statements zijn zodanig dat de update niet kan plaatsvinden wanneer iemand anders een waarde van dezelfde rij aangepast heeft. De waarden voor de where-condities komen uit de DataSet zelf. In de DataSet wordt van elke waarde 2 versies bijgehouden: de originele en de aangepaste. Wanneer de rij niet gevonden wordt, wordt er een concurrency exception gegooid die specifiek opgevangen kan worden in een catch block (zie figuur 9).

Figuur 9: Optimistic concurrency

```
private void button2_Click(object sender, System.EventArgs e)
{
try
   {
   db2DataAdapter1.Update(db2DataSet11);
   }
catch (DBConcurrencyException exep)
   {
   MessageBox.Show("foutDBconcurrencyException");
   }
}
```

Een buitenbeentje - LOBs en space management

Eric Venmans (ABIS)

Inleiding

Het gebruik van LOBs (Large OBjects) verhoogt zeker de functionaliteit van onze DB2 databases. We kunnen nu ook documenten, beelden, geluidsfragmenten en aanverwante multimedia gegevens gebruiken. Op mainframe (OS/390 of z/OS) gaat het in de eerste plaats over het bewaren van deze gegevens. Manipuleren is vooral weggelegd voor applicaties die op andere platformen (Linux, Unix, Windows) actief zijn.

Bewaren van data heeft uiteraard te maken met space management. In het geval van LOBs gaat het meestal over veel tot zeer veel space. Bovendien zal DB2 om met deze 'speciale' gegevens te kunnen werken, een afwijkend space management toepassen. Dit brengt eigen, specifieke problemen met zich mee.

Probleemstelling

In de context van LOB space management, werden we laatst met volgende vraag geconfronteerd: 'Hoe kunnen we space recupereren, nadat we een groot aantal LOB-gegevens verwijderd hebben?'. Het ging hier over een 'archivering'. Men wou een productietabel met actuele gegevens compacter maken met het oog op onderhoud (backup en recovery, reorganisaties). Daarvoor werden een groot aantal LOBs met nauwelijks gebruikte informatie gekopieerd naar een 'archive' tabel. Nadien werd deze informatie uit de productietabel verwijderd. Na een reorganisatie bleek dit echter niet de verwachte space besparing op te leveren. We zochten naar een verklaring en een oplossing.

Space gebruik bij LOB manipulaties

Het gebruik van LOBs werd reeds uitvoerig besproken in vorige artikels (zie Exploring DB2, jaargang 1, nr. 9). Ter opfrissing herhalen we hier wat er gebeurt op niveau van space gebruik, wanneer een LOB gemanipuleerd wordt. We bekijken enkel de LOB tablespace. Hierin komen de LOB-gegevens terecht samen telkens met een referentie (RowID) naar de corresponderende 'parent' rij in de basistabel. Deze bevat de 'gewone' gegevens die in een 'gewone' tablespace worden bewaard.

- INSERT - Een LOB-record wordt toegevoegd aan een LOB tablespace:
  - wanneer een nieuwe rij met LOB-gegevens wordt toegevoegd aan de basistabel;
  - of wanneer voor een bestaande rij ontbrekende LOB-gegevens (NULL-indicatie) ingevuld worden met 'echte' gegevens.

In beide gevallen zal DB2 in de LOB tablespace een 'chunk' zoeken met minimaal één lege page. Een 'chunk' is een set van 16 aaneensluitende pages met eigen space-informatie (in een 'low level space map'). De nieuwe LOB-informatie wordt in de vrije page(s) in de chunk geplaatst. De pages worden 'gealloceerd' met als identificatie de RowID van de 'parent'-rij. De page is hierbij de 'unit-of-space', d.w.z. dat een page nooit informatie bevat van LOBs uit verschillende rijen.

Als in een 'chunk' na het vullen van de beschikbare pages, meer space nodig is voor het plaatsen van de LOB-gegevens, zoekt en gebruikt het systeem een volgende chunk met vrije pages.

- UPDATE - Meestal werkt men op LOB-tablespace-niveau met 'LOG NO'. Log datasets zouden wel eens zeer snel kunnen vollopen met gemanipuleerde LOB-gegevens. Daarom gebeurt de aanpassing niet rechtstreeks. Een 'rollback' moet immers mogelijk blijven. DB2 zal bij een 'update' nieuwe pages 'alloceren' om de aangepaste versie van de LOB te noteren. De pages met de oude versie worden 'gedealloceerd'. Na een 'commit' worden deze gedealloceerde pages vrij gegeven voor eventueel herbruik. In geval van een 'rollback' worden de 'oude' pages opnieuw gealloceerd.

- DELETE - Als informatie moet verwijderd worden, zal het systeem de betrokken pages 'dealloceren'. Ze blijven echter 'gelocked' tot aan de 'commit' van de delete-actie. Het systeem zal ze echter 'realloceren', wanneer een 'rollback' wordt gevraagd.

Werking van de Reorg utility

Bij het manipuleren van LOB-informatie raakt deze mogelijk gefragmenteerd.

Bij toevoegingen achteraan de dataset kan een aaneensluitend stuk storage gebruikt worden. Als dit de enige manipulaties zijn (naast lezen) blijft de LOB tablespace goed georganiseerd. Alleen zal deze allicht groeien en moet af en toe extra ruimte gezocht worden. De gebruikte dataset kan via extents meer space krijgen. Bereikt de dataset zijn maximale lengte (4 of 64 GB) dan wordt een secondary dataset gealloceerd. Dit kan herhaald worden tot het maximum van in totaal 254 datasets (1 of 16 TB) bereikt wordt. Bij partitioned tablespaces loopt dit theoretisch maximum op tot 4064 TB.

Indien men de LOB-gegevens in de tablespace ook wijzigt en/of verwijdert, treedt fragmentatie op door hergebruik van vrijgekomen (gedealloceerde) pages. De LOB-gegevens raken verspreid over niet-aaneensluitende pages. Na een zekere periode dringt de nood aan reorganisatie zich op.

De 'gewone' REORG utility met zijn UNLOAD, (SORT), RELOAD, (BUILD INDEXES) houdt echter geen rekening met LOB-gegevens die in een aparte tablespace zitten. We voeren deze utility apart uit voor LOB tablespaces.

De werking verschilt merkelijk van een klassieke reorganisatie. Voor een LOB tablespace gaat de utility:

- de gebruikte dataset(s) niet verwijderen en niet opnieuw alloceren;
- wel proberen LOBs in aaneensluitende pages te krijgen door verplaatsing van gegevens naar vrije pages; dit is de eigenlijke reorganisatie;
- proberen hierbij secondary datasets (als die er zijn) leeg te krijgen;
- eventuele lege secondary datasets vernietigen;
- trachten overblijvende vrij ruimte te groeperen aan het einde van de dataset.

Bij dit alles blijft de primary dataset intact.

Wil men deze kleiner maken omdat bijvoorbeeld veel LOB informatie verdwenen is (zie o.a. ons archiveringsprobleem), dan moet men een oplossing zoeken buiten de REORG utility.

Space recuperatie

▪ Een combinatie van de UNLOAD en de LOAD utility komt hiervoor in aanmerking. Er is echter een belangrijke voorwaarde. De UNLOAD mag geen records genereren die langer zijn dan 32 KB. Een record bevat alle gegevens van een rij (LOB-gegevens inbegrepen!) en wat extra systeeminformatie.De UNLOAD plaatst alle rijen als records in een unload dataset. Na dit proces kan men de oorspronkelijke LOB dataset vernietigen en terug alloceren met aangepaste space kenmerken. De LOAD REPLACE zal de rijen terug in de basis tablespace plaatsen. De LOB-gegevens gaan naar de LOB tablespace. Ze worden daar sequentieel en aaneensluitend neergezet. Als de oorspronkelijke LOB tablespace veel lege pages bevatte, zal men op deze manier veel space besparen.Voorwaarde is echter dat de totale rijlengte, inclusief LOB-gegevens, kleiner blijft dan 32K. Het gaat dus niet over echte 'Large' OBjects.

▪ De RECOVERY utility wordt aangeraden voor space-recuperatie wanneer de beperking van 32 KB speelt. Alleen is het effect meestal gering. De COPY utility, die gebruikt wordt om een backup te maken van een LOB tablespace, kopieert alleen gealloceerde pages. Vrije, lege pages worden 'geskipped' bij dit proces. Dit wil niet zeggen dat na een RECOVERY automatisch minder space nodig is. De RECOVERY utility gaat namelijk lege pages terug genereren. Dit moet wel om interne pointers tussen tablespace pages correct te houden (de RECOVERY utility gaat zeker geen pointers aanpassen!). Alleen lege pages aan het einde van de dataset worden niet opnieuw gegenereerd. Het gevolg kan zijn dat er minder extents nodig zijn voor de dataset(s).

▪ Wil je meer space recupereren (en je hebt problemen met de 32 KB limiet) is er ook volgende procedure:

- CREATE table NEWTable like LOBProductionTable ....
- INSERT into NEWTable (...) SELECT ... from LOBProductionTable
- DROP table LOBProductionTable

- RENAME NEWTable to LOBProductionTable
- of: CREATE view LOBProductionTable as SELECT ... from NEW-Table ...

Besluit

Omdat het space management in LOB tablespaces merkelijk verschilt van het 'klassieke' space management, vraagt het ook een enigzins andere aanpak. Het probleem dat we hier hebben besproken is eerder een éénmalig probleem: space recupereren van een tablespace die een te grote primary dataset heeft. Dit is eerder een luxeprobleem: meestal worden 'dingen' groter i.p.v. kleiner. Toch leert het ons één en ander over de werking van LOB tablespaces. Als we deze werking kennen, is het vinden van oplossingen voor specifieke problemen, een kwestie van gezond verstand vermengd met een kleine dosis creativiteit. Geef er een LOB op.

DOSSIER 8

Op zoek naar extra veiligheid?

Uw applicaties behandelen gevoelige informatie? Extra maatregelen zijn gewenst wat betreft confidentialiteit? Privacy?

DB2 for z/OS versie 8 biedt u extra mogelijkheden om data in uw tabellen te versleutelen - encryptie en decriptie behoren tot de mogelijkheden!

De nieuwe ENCRYPT-functie zorgt ervoor dat data in bepaalde kolommen worden versleuteld, op basis van een encryptiepaswoord. Dit paswoord kan tijdens de datamanipulatie worden opgegeven, dan wel via een traditioneel special register ENCRYPTION PASSWORD worden ingesteld. Eens versleuteld, kunnen de data niet meer worden gelezen zonder specificatie van het correcte paswoord. Merk op dat indien dit special register niet wordt gebruikt, het dus mogelijk is een verschillend paswoord op te geven per functieaanroep! En voor de vergeetachtigen onder ons is het zelfs mogelijk per aanroep ook een hint-tekst te voorzien! De GET_HINT-functie kan gebruikt worden om deze hint op te halen.

Functies om te ontcijferen? DECRYPT_BIN en DECRYPT_CHAR.

En nu maar stoeien - denk toch maar even aan de extra kolomruimte die moet worden voorzien om al deze informatie - paswoord en hint - op te slaan!

Kris Van Thillo (ABIS)

CURSUSPLANNING MRT - JUN 2004

DB2 concepten	375 EUR	07/06 (W)
DB2 for OS/390, een totaaloverzicht	1625 EUR	29/03-02/04 (W), 24-28/05 (W), 07-11/06 (L)
DB2 UDB, een totaaloverzicht	1625 EUR	24-25/05&01-03/06 (W)
RDBMS concepten	325 EUR	29/03 (W), 24/05 (W), 07/06 (L)
Basiskennis SQL	325 EUR	30/03 (W), 25/05(W), 08/06 (L)
DB2 for OS/390 basiscursus	975 EUR	31/03-02/04 (W), 09-11/06 (L)
DB2 UDB basiscursus	975 EUR	01-03/06 (W)
SQL workshop	700 EUR	15-16/04 (W), 21-22/06 (L)
DB2 for OS/390 programmering voor gevorderden	700 EUR	17-18/05 (W)
Gebruik van DB2 procedural extensions	350 EUR	19/05(W)
DB2 for OS/390: SQL performance	1200 EUR	23-25/06 (W)
DB2 UDB applicatieperformance	400 EUR	08/06 (W)
Database applicatieprogrammering met Java	800 EUR	29-30/04 (L), 01-02/06 (W)
Fysiek ontwerp van relationele databases.	700 EUR	03-04/05 (L)
DB2 for OS/390 database administratie	1600 EUR	24-27/05(L)
DB2 for OS/390 operations and recovery	1500 EUR	21-23/04(L)
DB2 UDB systeembeheer en performance	400 EUR	30/04 (L), 22/06 (W)
DB2 UDB en zijn extenders: XML en text search	200 EUR	04/06 (W)
DB2 UDB integratie met MQSeries	200 EUR	04/06 (W)

Plaats: L = Leuven; W = Woerden; details en extra cursussen: www.abis.be

Postbus 220
Diestsevest 32
BE-3000 Leuven
Tel. 016/245610
Fax 016/245691
training@abis.be

abis
TRAINING & CONSULTING

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

Bijlagen

DB2 Data Provider - c#-code voorbeeld 1

```
using System;
using System.Drawing;
using System.Collections;
using System.ComponentModel;
using System.Windows.Forms;
using System.Data;
namespace WindowsApplication7
{
   /// <summary>
   /// Summary description for Form1.
   /// </summary>
   public class Form1 : System.Windows.Forms.Form
   {
      private System.Windows.Forms.Button button1;
      private IBM.Data.DB2.DB2Connection dB2Connection1;
      private IBM.Data.DB2.DB2Command dB2Command1;
      private IBM.Data.DB2.DB2DataReader dB2DataReader1;
      /// <summary>
      /// Required designer variable.
      /// </summary>
      private System.ComponentModel.Container components = null;

      public Form1()
      {
         //
         // Required for Windows Form Designer support
         //
         InitializeComponent();

         //
         // TODO: Add constructor code after InitializeComponent call
         //
      }

      /// <summary>
      /// Clean up any resources being used.
      /// </summary>
      protected override void Dispose( bool disposing )
      {
         if( disposing )
         {
            if (components != null)
            {
               components.Dispose();
            }
         }
         base.Dispose( disposing );
      }

      #region Windows Form Designer generated code
      /// <summary>
      /// Required method for Designer support - do not modify
      /// </summary>
      private void InitializeComponent()
      {
         this.button1 = new System.Windows.Forms.Button();
```

```
        this.dB2Connection1 = new IBM.Data.DB2.DB2Connection();
        this.dB2Command1 = new IBM.Data.DB2.DB2Command();
        this.SuspendLayout();
        //
        // button1
        //
        this.button1.Location = new System.Drawing.Point(368, 16);
        this.button1.Name = "button1";
        this.button1.TabIndex = 0;
        this.button1.Text = "button1";
        this.button1.Click += new
System.EventHandler(this.button1_Click);
        //
        // dB2Connection1
        //
        this.dB2Connection1.ConnectionString = "database=SOC8080;user
Id=tb00986;password=nearend3;persist security info=true";
        //
        // dB2Command1
        //
        this.dB2Command1.CommandText = "select coname from
tb00986.tutcompanies";
        //
        // Form1
        //
        this.AutoScaleBaseSize = new System.Drawing.Size(5, 13);
        this.ClientSize = new System.Drawing.Size(480, 189);
        this.Controls.AddRange(new System.Windows.Forms.Control[] {
                this.button1});
        this.Name = "Form1";
        this.Text = "Form1";
        this.ResumeLayout(false);

    }
    #endregion

    /// <summary>
    /// The main entry point for the application.
    /// </summary>
    [STAThread]
    static void Main()
    {
       Application.Run(new Form1());
    }

    private void button1_Click(object sender, System.EventArgs e)
    {
       dB2Command1.Connection = dB2Connection1;
       dB2Connection1.Open();
       IBM.Data.DB2.DB2DataReader dB2DataReader1 =
dB2Command1.ExecuteReader();
        while (dB2DataReader1.Read())
        {
           Console.WriteLine(dB2DataReader1.GetString(0));
        }
        dB2DataReader1.Close();
        dB2Connection1.Close();
    }


}
```

DB2 Data Provider - c#-code voorbeeld 2

```
using System;
using System.Drawing;
using System.Collections;
using System.ComponentModel;
using System.Windows.Forms;
using System.Data;

namespace WindowsApplication8
{
   public class Form1 : System.Windows.Forms.Form
   {
      private System.Windows.Forms.Button button1;
      private System.Windows.Forms.Button button2;
      private System.Windows.Forms.DataGrid dataGrid1;
      private IBM.Data.DB2.DB2Connection db2Connection1;
      private IBM.Data.DB2.DB2DataAdapter db2DataAdapter1;
      private IBM.Data.DB2.DB2Command db2SelectCommand1;
      private IBM.Data.DB2.DB2Command db2InsertCommand1;
      private IBM.Data.DB2.DB2Command db2UpdateCommand1;
      private IBM.Data.DB2.DB2Command db2DeleteCommand1;
      private WindowsApplication8.DB2DataSet1 db2DataSet11;

      private System.ComponentModel.Container components = null;

      public Form1()
      {
         InitializeComponent();
      }

      protected override void Dispose( bool disposing )
      {
         if( disposing )
         {
            if (components != null)
            {
               components.Dispose();
            }
         }
         base.Dispose( disposing );
      }

      private void InitializeComponent()
      {
         this.button1 = new System.Windows.Forms.Button();
         this.button2 = new System.Windows.Forms.Button();
         this.dataGrid1 = new System.Windows.Forms.DataGrid();
         this.db2Connection1 = new IBM.Data.DB2.DB2Connection();
         this.db2DataAdapter1 = new IBM.Data.DB2.DB2DataAdapter();
         this.db2SelectCommand1 = new IBM.Data.DB2.DB2Command();
         this.db2InsertCommand1 = new IBM.Data.DB2.DB2Command();
         this.db2UpdateCommand1 = new IBM.Data.DB2.DB2Command();
         this.db2DeleteCommand1 = new IBM.Data.DB2.DB2Command();
         this.db2DataSet11 = new WindowsApplication8.DB2DataSet1();

((System.ComponentModel.ISupportInitialize)(this.dataGrid1)).BeginIni
t();

((System.ComponentModel.ISupportInitialize)(this.db2DataSet11)).Begin
Init();
         this.SuspendLayout();
         //
```

```
        // button1
        //
        this.button1.Location = new System.Drawing.Point(344, 8);
        this.button1.Name = "button1";
        this.button1.Size = new System.Drawing.Size(144, 23);
        this.button1.TabIndex = 0;
        this.button1.Text = "ophalen data uit database";
        this.button1.Click += new
System.EventHandler(this.button1_Click);
        //
        // button2
        //
        this.button2.Location = new System.Drawing.Point(296, 176);
        this.button2.Name = "button2";
        this.button2.Size = new System.Drawing.Size(200, 23);
        this.button2.TabIndex = 2;
        this.button2.Text = "aanpassingen uitvoeren in database";
        this.button2.Click += new
System.EventHandler(this.button2_Click);
        //
        // dataGrid1
        //
        this.dataGrid1.DataMember = "TUTCOMPANIES";
        this.dataGrid1.DataSource = this.db2DataSet11;
        this.dataGrid1.HeaderForeColor =
System.Drawing.SystemColors.ControlText;
        this.dataGrid1.Location = new System.Drawing.Point(16, 40);
        this.dataGrid1.Name = "dataGrid1";
        this.dataGrid1.Size = new System.Drawing.Size(480, 128);
        this.dataGrid1.TabIndex = 3;
        //
        // db2Connection1
        //
        this.db2Connection1.ConnectionString = "database=SOC8080;user
Id=xxxxxx;password=xxxxxx;persist security info=true";
        //
        // db2DataAdapter1
        //
        this.db2DataAdapter1.DeleteCommand = this.db2DeleteCommand1;
        this.db2DataAdapter1.InsertCommand = this.db2InsertCommand1;
        this.db2DataAdapter1.SelectCommand = this.db2SelectCommand1;
        this.db2DataAdapter1.TableMappings.AddRange(new
System.Data.Common.DataTableMapping[] {



...
Tabel en kolom mappings
...
        })
        this.db2DataAdapter1.UpdateCommand = this.db2UpdateCommand1;
        //
        // db2SelectCommand1
        //
        this.db2SelectCommand1.CommandText = "SELECT \"CONO\",
\"CONAME\", \"COSTREET\", \"COSTRNO\", \"COTOWN\", \"COTOWNNO\",
\"COCOUNTR\", \"COTEL\", \"COVAT\", \"COBANKNO\", \"COC_PNO\" FROM
\"TB00986\".\"TUTCOMPANIES\"";
        this.db2SelectCommand1.Connection = this.db2Connection1;
        //
        // db2InsertCommand1
        //
```

```
        this.db2InsertCommand1.CommandText = "INSERT INTO
\"TB00986\".\"TUTCOMPANIES\" (\"CONO\", \"CONAME\", \"COSTREET\",
\"COSTRNO\", \"COTOWN\", \"COTOWNNO\", \"COCOUNTR\", \"COTEL\",
\"COVAT\", \"COBANKNO\", \"COC_PNO\") VALUES (?, ?, ?, ?, ?, ?, ?, ?,
?, ?, ?)";
        this.db2InsertCommand1.Connection = this.db2Connection1;
        this.db2InsertCommand1.Parameters.Add(new
IBM.Data.DB2.DB2Parameter("CONO", IBM.Data.DB2.DB2Type.SmallInt, 2,
"CONO"));
        this.db2InsertCommand1.Parameters.Add(new
IBM.Data.DB2.DB2Parameter("CONAME", IBM.Data.DB2.DB2Type.Char, 45,
"CONAME"));

...
```
Vervolg parameters voor het db2InsertCommand1 object
```
...
        //
        // db2UpdateCommand1
        //
        this.db2UpdateCommand1.CommandText = @"UPDATE
""TB00986"".""TUTCOMPANIES"" SET ""CONO"" = ?, ""CONAME"" = ?,
""COSTREET"" = ?, ""COSTRNO"" = ?, ""COTOWN"" = ?, ""COTOWNNO"" = ?,
""COCOUNTR"" = ?, ""COTEL"" = ?, ""COVAT"" = ?, ""COBANKNO"" = ?,
""COC_PNO"" = ? WHERE (""CONO"" =  ?) AND (""CONAME"" =  ?) AND
(""COSTREET"" =  ?) AND (""COSTRNO"" =  ? OR CAST(? as VARCHAR(10)) IS
NULL AND ""COSTRNO"" IS NULL) AND (""COTOWN"" =  ?) AND (""COTOWNNO""
=  ? OR CAST(? as CHARACTER(10)) IS NULL AND ""COTOWNNO"" IS NULL) AND
(""COCOUNTR"" =  ? OR CAST(? as CHARACTER(4)) IS NULL AND ""COCOUNTR""
IS NULL) AND (""COTEL"" =  ? OR CAST(? as CHARACTER(16)) IS NULL AND
""COTEL"" IS NULL) AND (""COVAT"" =  ? OR CAST(? as CHARACTER(11)) IS
NULL AND ""COVAT"" IS NULL) AND (""COBANKNO"" =  ? OR CAST(? as
CHARACTER(14)) IS NULL AND ""COBANKNO"" IS NULL) AND (""COC_PNO"" =  ?
OR CAST(? as SMALLINT) IS NULL AND ""COC_PNO"" IS NULL) ";
        this.db2UpdateCommand1.Connection = this.db2Connection1;
        this.db2UpdateCommand1.Parameters.Add(new
IBM.Data.DB2.DB2Parameter("CONO", IBM.Data.DB2.DB2Type.SmallInt, 2,
"CONO"));
        this.db2UpdateCommand1.Parameters.Add(new
IBM.Data.DB2.DB2Parameter("CONAME", IBM.Data.DB2.DB2Type.Char, 45,
"CONAME"));

...
```
Vervolg parameters voor het db2UpdateCommand1 object (current values)
```
...

this.db2UpdateCommand1.Parameters.Add(new
IBM.Data.DB2.DB2Parameter("Original_CONO",
IBM.Data.DB2.DB2Type.SmallInt, 2,
System.Data.ParameterDirection.Input, false, ((System.Byte)(0)),
((System.Byte)(0)), "CONO", System.Data.DataRowVersion.Original,
null));
        this.db2UpdateCommand1.Parameters.Add(new
IBM.Data.DB2.DB2Parameter("Original_CONAME",
IBM.Data.DB2.DB2Type.Char, 45, System.Data.ParameterDirection.Input,
false, ((System.Byte)(0)), ((System.Byte)(0)), "CONAME",
System.Data.DataRowVersion.Original, null));

...
```
Vervolg parameters voor het db2UpdateCommand1 object (original values)
```
...

        //
        // db2DeleteCommand1
```

```
        //
        this.db2DeleteCommand1.CommandText = @"DELETE FROM
""TB00986"".""TUTCOMPANIES"" WHERE (""CONO"" =  ?) AND (""CONAME"" =
?) AND (""COSTREET"" =  ?) AND (""COSTRNO"" =  ? OR CAST(? as
VARCHAR(10)) IS NULL AND ""COSTRNO"" IS NULL) AND (""COTOWN"" =  ?) AND
(""COTOWNNO"" =  ? OR CAST(? as CHARACTER(10)) IS NULL AND ""COTOWNNO""
IS NULL) AND (""COCOUNTR"" =  ? OR CAST(? as CHARACTER(4)) IS NULL AND
""COCOUNTR"" IS NULL) AND (""COTEL"" =  ? OR CAST(? as CHARACTER(16))
IS NULL AND ""COTEL"" IS NULL) AND (""COVAT"" =  ? OR CAST(? as
CHARACTER(11)) IS NULL AND ""COVAT"" IS NULL) AND (""COBANKNO"" =  ? OR
CAST(? as CHARACTER(14)) IS NULL AND ""COBANKNO"" IS NULL) AND
(""COC_PNO"" =  ? OR CAST(? as SMALLINT) IS NULL AND ""COC_PNO"" IS
NULL) ";
        this.db2DeleteCommand1.Connection = this.db2Connection1;
        this.db2DeleteCommand1.Parameters.Add(new
IBM.Data.DB2.DB2Parameter("Original_CONO",
IBM.Data.DB2.DB2Type.SmallInt, 2,
System.Data.ParameterDirection.Input, false, ((System.Byte)(0)),
((System.Byte)(0)), "CONO", System.Data.DataRowVersion.Original,
null));
        this.db2DeleteCommand1.Parameters.Add(new
IBM.Data.DB2.DB2Parameter("Original_CONAME",
IBM.Data.DB2.DB2Type.Char, 45, System.Data.ParameterDirection.Input,
false, ((System.Byte)(0)), ((System.Byte)(0)), "CONAME",
System.Data.DataRowVersion.Original, null));

...
Vervolg parameters voor het db2DeleteCommand1 object
...

        //
        // db2DataSet11
        //
        this.db2DataSet11.DataSetName = "DB2DataSet1";
        this.db2DataSet11.Locale = new
System.Globalization.CultureInfo("en-GB");
        this.db2DataSet11.Namespace = "http://www.tempuri.org/
DB2DataSet1.xsd";
        //
        // Form1
        //
        this.AutoScaleBaseSize = new System.Drawing.Size(5, 13);
        this.ClientSize = new System.Drawing.Size(512, 205);
        this.Controls.AddRange(new System.Windows.Forms.Control[] {
                this.dataGrid1,
                this.button2,
                this.button1});
        this.Name = "Form1";
        this.Text = "Form1";

((System.ComponentModel.ISupportInitialize)(this.dataGrid1)).EndInit(
);

((System.ComponentModel.ISupportInitialize)(this.db2DataSet11)).EndIn
it();
        this.ResumeLayout(false);

    }
    #endregion

    /// <summary>
    /// The main entry point for the application.
    /// </summary>
```

```
    [STAThread]
    static void Main()
    {
       Application.Run(new Form1());
    }

    private void button1_Click(object sender, System.EventArgs e)
    {
       db2DataSet11.Clear();
       db2DataAdapter1.Fill(db2DataSet11);
    }

    private void button2_Click(object sender, System.EventArgs e)
    {
       db2DataAdapter1.Update(db2DataSet11);
    }
}
```