ABIS NIEUWSBRIEF VOOR DB2 PROFESSIONALS

JAARGANG 2 - NUMMER 2, OKTOBER 2003

OPEN CURSOR

Dit nummer van Exploring DB2 is volledig gewijd aan applicatieontwikkeling.
In eerste instantie staan we opnieuw stil bij de problematiek van query-tuning. Daarnaast hebben we het over het ontwikkelen van DB2-applicaties in Java en .NET.
ABIS is overtuigd dat beide ontwikkelplatformen - J2EE en .NET - een belangrijke rol zullen spelen bij applicatieontwikkeling in een DB2 omgeving. Het is hierbij niet onze bedoeling in Exploring DB2 aandacht te besteden aan Java en .NET als ontwikkelplatform op zich. Waar onze aandacht echter wel naar uitgaat is de interfacing met DB2.
Ook in volgende nummers blijven we deze problematiek opvolgen.

Het ABIS DB2-team.

IN DIT NUMMER:

CLOSE CURSOR

In een volgend nummer bieden we u een derde case aan. We staan even stil bij temporary tables en gaan het belang na van triggers en gecontroleerde redundantie.

Tot dan!

Tunen van Java/DB2 applicaties

Pieter Bedert (ABIS)

Het DB2- en Java-team van ABIS is reeds enkele maanden bezig met alle kantjes van een Java/DB2-applicatie te belichten. Ook de performance van die applicaties is niet onbelangrijk. Voor een eerste kennismaking met tuning maakten we gebruik van een relatief eenvoudig programma. We schreven een Java-programma dat met behulp van JDBC een SELECT-statement uitvoert op een redelijk grote database. Van het resultaat lezen we een aantal rijen. We gebruikten voor de JDBC-ondersteuning een IBM-DB2 type 2 en type 4 JDBC-driver.

Dilemma: statische vs. dynamische SQL

Wanneer we in Java programmeren beschikken we over verschillende API's (zie ook Exploring DB2 Jaargang 1, Nummer 7). Enerzijds beschikken we over JDBC, een Java-manier om SQL-statements samen te stellen en uit te voeren. JDBC zal altijd dynamische SQL-vragen naar de database sturen. Anderzijds beschikken we ook over een mogelijkheid om embedded SQL te programmeren: SQLJ. Deze SQL-statements worden statisch uitgevoerd indien ze na compilatie ook gebind worden aan de database, anders worden ze alsnog dynamisch uitgevoerd.

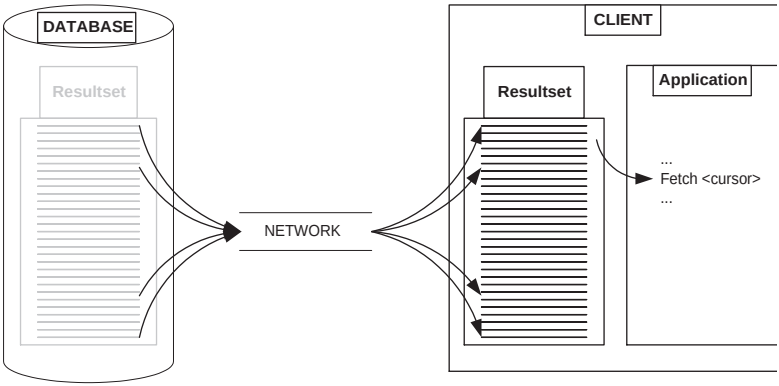
Toch zal de performance-discussie zich niet in de eerste plaats afspelen op het vlak van statische vs. dynamische SQL. Want deze discussie kunnen we voeren voor elke programmeertaal. Statische SQL zal hier ook alle mogelijkheden bieden zoals bijvoorbeeld in COBOL: EXPLAIN, HINTS,... Ook het gebruik van dynamische SQL zal je de klassieke voor- en nadelen bieden: minder controleerbare SQL-statements, geen onbekende variabelen tijdens optimalisatie (kan resulteren in beter toegangspad),...

Gedistribueerde omgevingen

We gaan ons eerder concentreren op de specifieke problemen van een gedistribueerde omgeving. De bottle-neck in omgevingen waar applicatie en database zich op fysiek verschillende machines bevinden, is het datatransport. Onnodig veel data uit de database die via een netwerk tot bij de applicatie moet komen, zorgen voor grote vertragingen bij de antwoordtijden (zie ook figuur 1). Vooral bij transactionele internetapplicaties, waar eindgebruikers niet langer dan een aantal seconden willen wachten, is dit een drama. Maar een grote belasting van je netwerk gaat niet alleen ten koste van die ene applicatie, maar van iedereen die van het netwerk wil gebruik maken.

De conclusie is duidelijk: We zullen het data-transport van database naar applicatie moeten beperken tot het hoogst nodige!

Figuur 1: Gevaren van gedistribueerd werken



Algemene oplossing

Het beperken van het data-transport kunnen we op twee manieren. We kunnen het aantal teruggegeven rijen beperken tot een zeker maximum. Dit moet voorkomen dat door 'ongelukjes' na minutenlang wachten duizenden rijen te voorschijn komen waar de eindgebruiker toch niets in terugvindt. Tweede manier is dat we de nodige data pas dan naar de applicatie laten versturen wanneer ze echt nodig is. Wanneer een SQL-statement dan 2000 rijen selecteert, dan vragen we bijvoorbeeld om slechts de eerste 20 als antwoord te geven. Alleen als de applicatie de 21ste rij vraagt wordt het volgende pakketje van 20 rijen verstuurd.

In wat volgt gaan we na in hoeverre dit alles makkelijk kan worden geïmplementeerd in Java-applicaties. Indien niet, gaan we na of er DB2-technisch een alternatief kan worden geboden. Bedoeling is een applicatiestructuur als geschetst in figuur 2 te implementeren.

Java-oplossingen in theorie

Voor beide voorgestelde oplossingen vind je in de Java-documentatie een methode die deze taken vervult.

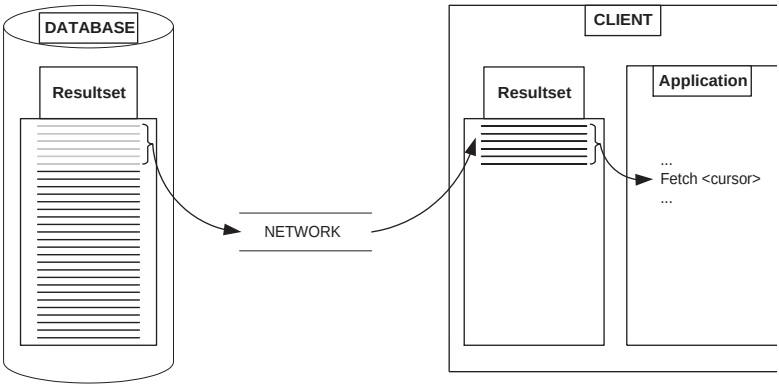
- De methode *setMaxRows(n)* zorgt ervoor dat maximaal 'n' rijen voorkomen in het uiteindelijke resultaat.
- De methode *setFetchSize(n)* zou ervoor moeten zorgen dat het resultaat in groepjes van 'n' rijen beschikbaar komt op het client-platform.

Java-oplossingen in de praktijk (IBM DB2 Type 2/4 JDBC-driver)

Na enkele testen met een IBM DB2-driver in de praktijk kwamen we tot opmerkelijke resultaten. Als je de methode setMaxRows gaat gebruiken dan doet ze functioneel wat ze hoort te doen, met name, de

resultset beperken tot maximaal 'n' rijen. Maar niemand zal hier een performance-voordeel uithalen. Na het uitvoeren van verschillende traces hebben we echter gemerkt dat het volledige geselecteerde resultaat toch verstuurd wordt naar de applicatie (en dus over het netwerk). Pas daar wordt de resultset verminderd tot het maximaal aantal rijen. Dit kan ons geen performance-voordeel opleveren omdat toch alle data over het netwerk naar de applicatie is gegaan.

Figuur 2: Het beperken van de resultset



Na enkele testen is ook gebleken dat de methode setFetchSize ook niet doet wat ze beweert te doen. De methode setFetchSize heeft absoluut geen effect op het transport van data tussen database en applicatie.

De twee schijnbaar nuttige methodes zijn dus onbruikbaar op performance-gebied en we moeten eerlijk zijn, de oorzaak ligt bij de implementatie van IBM. DB2-specifieke drivers van IBM zouden deze opties moeten vertalen in de nodige database-opties (e.g. SQL-statements) - misschien iets voor een volgende versie?

Deze resultaten dwingen ons om kritisch om te springen met deze Java-methodes. Deze methodes staan beschreven in 'interfaces', abstracte klasses met methodes staan die Java in combinatie met een database 'zou moeten kunnen'. Deze interfaces moeten dan concreet ingevuld worden door drivers van IBM voor DB2, door Oracle-drivers of SQL Server-drivers. Of een driver doet wat hij hoort te doen is dus afhankelijk van de driver-specifieke implementatie en vooral op performance-gebied moeten we aandachtig zijn, te meer omdat die functionaliteit enkel achter de schermen merkbaar is.

DB2-oplossingen

We worden verplicht om de Java-filosofie, die zegt dat applicaties 'portable' moeten zijn over operating systemen en databases heen, naast ons neer te leggen en te kiezen voor een DB2-specifieke oplossing om het performance-probleem op te lossen.

In het SQL-statement voegen we een DB2-optie toe, FETCH FIRST n ROWS ONLY om het resultaat te beperken, en OPTIMIZE FOR n ROWS om het resultaat in pakketjes van n rijen te laten transporteren naar de applicatie.

Andere cursors

De hierboven beschreven resultaten gelden voor de meest eenvoudige cursor. We kunnen natuurlijk nog cursors gaan definiëren 'FOR UPDATE OF'; wanneer we naar andere drivertypes overstappen kunnen we zelfs scrollable cursors gaan ondersteunen. Uitvoerig tracen van applicaties heeft ons geleerd dat elke andere opzet van cursors of elk gebruik van andere drivers kan invloed hebben op de manier waarop data naar de applicatie verstuurd wordt. Om maar één voorbeeld te geven: de cursor 'FOR UPDATE OF' zal in elke situatie, onafhankelijk van gegeven opties, rij per rij communiceren van de database naar de applicatie.

Conclusie

De invloed van statische vs. dynamische SQL zal geen extra bekommernis zijn.

De vraag of we 'portable' SQL (enkel JDBC-opties) dan wel 'proprietary' SQL (met expliciete DB2-opties) gaan gebruiken is in de Java-context belangrijker. Vooral wanneer we in uitgebreide ontwikkelomgevingen terecht komen met meerdere databases (ook niet DB2-databases), is deze beslissing niet makkelijk.

Bovendien, wanneer performance-problemen opduiken moet men kritisch staan tegenover de gebruikte JDBC-drivers. Uitgebreide traces van communicatie tussen uw database en uw applicaties zullen daarbij een dankbaar hulpmiddel zijn. Dus zelfs bij nieuwe mogelijkheden kritisch blijven ten opzichte van data transport!

DB2 performance - case 2

Eric Venmans (ABIS)

In een vorig artikel over DB2 performance werd aangetoond dat optimaliseren te ver kan gaan, in die zin dat we na het optimaliseren een toegangspad beschreven krijgen dat enkel beter lijkt, maar zeker niet beter is.

Een andere weinig interessante situatie is deze waarbij we wel degelijk een beter toegangspad krijgen, maar waarbij dit ten koste gaat van het correct uitvoeren van de query. Het is geen frequent voorkomende situatie, maar als ze voorkomt, wordt het vertrouwen in DB2 wel even stevig op de proef gesteld. DB2 is echter een robuust, betrouwbaar systeem en het ongelijk ligt zoals meestal bij de gebruiker.

CASE 2: beter toegangspad, foutief resultaat!

We nemen een vereenvoudigd voorbeeld om de essentie van het probleem duidelijk zichtbaar te houden.

Een tabel 'PROJECTEN' bevat zoals de naam laat uitschijnen, informatie over projecten. Als essentiële informatie in ons voorbeeld vermelden we een unieke 'project identificatie' en een 'project startdatum'. De tabel bevat projecten die reeds afgewerkt zijn, projecten die aan de gang zijn en projecten die enkel nog maar gepland zijn. Nieuwe projecten worden doorgaans gepland na alle reeds in de tabel genoteerde projecten. Als het echter om een dringend project gaat, wordt het op een geschikte datum gepland en worden alle projecten die na de gekozen datum gepland staan, verschoven. De verschuiving gebeurt door de startdatum te verhogen van de projecten die na de startdatum van het nieuwe, dringende project gepland zijn. Met hoeveel dagen dit gebeurt, wordt opgegeven door een verantwoordelijke.

Voor het uitvoeren van deze verschuivingsactie moet een programma geschreven worden. Dit moet niet alleen het tussenvoegen en verschuiven van projecten ondersteunen. Het moet bovendien elke verschuiving individueel rapporteren. Kwestie van de betrokkenen te kunnen verwittigen zodat individuele correcties kunnen aangebracht worden.

Eerste oplossing

We werken met een "DECLARE ... CURSOR FOR SELECT ... FOR UPDATE OF ...". Op deze manier krijgen we elk project dat aangepast wordt, individueel te zien en is de rij die we op elk ogenblik willen aanpassen gemakkelijk aan te duiden (zie voorbeeld 1).

Voorbeeld 1: oplossing versie 1

```
DECLARE uitstel_projecten CURSOR FOR
 SELECT project_id_kolom, ...
   FROM projecten_tabel
  WHERE startdatum_kolom > :nieuw_project.startdatum
    FOR UPDATE OF startdatum_kolom

FETCH uitstel_projecten
 INTO :project_id, ...

UPDATE projecten_tabel
   SET startdatum_kolom = startdatum_kolom + :opgegeven_duurtijd
 WHERE CURRENT OF uitstel_projecten
```

Deze constructie werkt foutloos. Alleen, na onderzoek van het toegangspad (via EXPLAIN), blijkt de index op de kolom startdatum niet gebruikt te worden door DB2. Dit is nochtans wenselijk omdat de tabel erg veel projecten bevat (vooral afgewerkte) en in verhouding weinig projecten die voor verschuiving in aanmerking komen (de geplande).

Performancegericht alternatief

Na wat experimenteren blijkt volgende oplossing wel het gebruik van de betrokken index op te leveren. Bovendien kan men nu een ORDER BY toevoegen waardoor de hanteerbaarheid van de gerapporteerde aanpassingen verhoogt (voorbeeld 2).

Voorbeeld 2: oplossing versie 2

```
DECLARE uitstel_projecten CURSOR FOR
 SELECT project_id_kolom, ...
   FROM projecten_tabel
  WHERE startdatum_kolom > :nieuw_project.startdatum
  ORDER BY startdatum_kolom

FETCH uitstel_projecten
 INTO :project_id, ...

UPDATE projecten
   SET startdatum_kolom = startdatum_kolom + :opgegeven_duurtijd
 WHERE project_id_kolom = :project_id
```

Helaas, deze constructie stuurt ons programma in een 'loop'. DB2 voert namelijk alle updates (ook de indexaanpassingen) uit 'in flight', d.w.z. onmiddellijk. We zetten even op een rijtje wat er gebeurt in het systeem terwijl onze applicatie in uitvoering is:

- FETCH: via de index zoekt DB2 de eerste rij die met de cursor-select wordt aangeduid.
- UPDATE: de eerste rij wordt aangepast en ook de bijhorende index; de oorspronkelijke index info wordt verwijderd en nieuwe wordt toegevoegd verder in de index; de datum waarop de index informatie geordend is wordt namelijk groter
- volgende FETCH: via de (gewijzigde) index zoekt DB2 de volgende rij die met de cursor-select wordt aangeduid

- en dan opnieuw UPDATE en FETCH en ...

Vroeg of laat komt DB2 elke gewijzigde datum opnieuw tegen in de index. Dit geeft aanleiding tot het opnieuw aanpassen van reeds gewijzigde rijen. Hierdoor wordt ook de betrokken index informatie opnieuw aangepast en verplaatst naar waar DB2 straks nog gaat zoeken. Zo blijven we bezig tot één of ander mechanisme (time-out, te veel locks, ...) de zaak blokkeert. Dit heeft normaal een 'rollback' tot gevolg. DB2 heeft veel energie verbruikt om uiteindelijk terug bij 'AF' aan te komen, tenminste wat onze applicatie betreft.

Het beschreven mechanisme wordt door DB2 enkel toegepast als het systeem GEEN verband ziet tussen het lezen (cursor-select) en het aanpassen. Als we de ORDER BY weglaten en opnieuw werken met FOR UPDATE OF (expliciet aanduiden dat er WEL een verband is tussen lezen en aanpassen) is het 'loopen' van de baan. DB2 gebruikt een toegangspad waaarbij de aanpassingen niet interferen met de leesacties. Deze leesacties gebeuren in de tijd vóór de eerste update: ofwel worden de aangeduide rijen 'gematerialiseerd' in een tijdelijk werkbestand en van daaruit sequentieel gelezen, ofwel wordt de betrokken indexinformatie eerst opzij gezet en bevroren in de RID-pool (als onderdeel van de list-prefetch techniek) en van daaruit gebruikt voor het opzoeken van de aangeduide rijen.

Correcte oplossingen

Het voorbeeld hier geeft geen aanleiding tot drama's. We merken onmiddellijk dat er iets mis is en we kunnen ingrijpen. Een greep uit mogelijke oplossingen:

- toch versie 1 met FOR UPDATE OF nemen en performance inleveren
- ofwel zelf lezen en aanpassen scheiden in de tijd (zie voorbeeld 3)
- ofwel werken met een timestamp (zie voorbeeld 4)

Voorbeeld 3: oplossing versie 3

```
INSERT INTO temp_projecten_tabel
     SELECT project_id_kolom, ...
       FROM projecten_tabel
      WHERE startdatum_kolom > :nieuw_project.startdatum

DECLARE uitstel_projecten CURSOR FOR
 SELECT project_id_kolom, ...
   FROM temp_projecten_tabel
  ORDER BY startdatum_kolom

FETCH uitstel_projecten
 INTO :project_id, ...

UPDATE projecten
   SET startdatum_kolom = startdatum_kolom + :opgegeven_duurtijd
 WHERE project_id_kolom = :project_id
   AND startdatum_kolom > :nieuw_project.startdatum (*)
```

(*) is nodig indien de mogelijkheid bestaat dat de startdatum van een project gewijzigd wordt (kleiner wordt) terwijl ons programma loopt.

```
DECLARE uitstel_projecten CURSOR FOR
 SELECT project_id_kolom, ...
   FROM projecten_tabel
  WHERE startdatum_kolom > :nieuw_project.startdatum
    AND laatste_aanpassing_TS_kolom < :huidigeTS (*)
  ORDER BY startdatum_kolom

FETCH uitstel_projecten
 INTO :project_id, ...

UPDATE projecten
   SET startdatum_kolom = startdatum_kolom + :opgegeven_duurtijd,
       laatste_aanpassing_TS_kolom = :huidigeTS
 WHERE project_id_kolom = :project_id
```

(*) deze kolom moet toegevoegd worden aan de tabel en allicht mee opgenomen worden in de index om een optimaal toegangspad te behouden.

Gevaarlijke situaties

Het hier gegeven voorbeeld is vrij onschuldig, omdat, zoals reeds gezegd, het 'fout lopen' onmiddellijk duidelijk is doordat het programma in een 'loop ' terecht komt.

Aan de hand van dit basisschema kunnen we echter een aantal situaties schetsen die een stuk gevaarlijker zijn omdat ze niet onmiddellijk opvallen of zich niet onmiddellijk voordoen. Gevaarlijk zijn ook situaties waarbij maar af en toe iets misloopt.

Stel dat we te maken hebben met een probleem dat vergelijkbaar is met het probleem van ons basisvoorbeeld, maar waarbij we een BETWEEN-conditie hebben op de waarde die we willen groter maken. Nu gaat ons programma niet 'loopen', maar rijen waarvan de waarde na aanpassing nog steeds tussen de BETWEEN-limieten valt, wordt telkens opnieuw aangepast tot hij buiten die limieten valt. Zoveel te kleiner de kans is dat zulk een meervoudige aanpassing gebeurt (d.w.z. indien meestal de nieuwe waarde na één aanpassing reeds buiten de limieten valt) zoveel te groter de kans dat de foutieve aanpassingen te laat worden vastgesteld.

Vergelijkbaar is een situatie waarbij de nieuwe waarde meestal kleiner blijft dan de eerst daaropvolgende. De waarde wordt aangepast in de index, maar blijft fysiek op dezelfde plaats. Bij een volgende leesactie wordt correct een volgende rij opgehaald. We zeggen wel 'meestal', dus af en toe kan het weer behoorlijk mis gaan.

Gevaarlijk zijn ook de situaties waarbij in het begin niets misloopt. Neem als voorbeeld de 'versie 2-oplossing', maar dan met een index die niet geclustered is. Als DB2 de index al gebruikt voor het opzoeken van de rijen die men wil aanpassen, zal hierbij allicht list-prefetch worden toegepast. D.w.z. dat de index reeds gelezen is nog voor de eerste aanpassing gebeurt. Dus is er niets aan de hand, tot de DBA besluit om de omgeving te 'optimaliseren'. Hij of zij maakt van de betrokken index een cluster-index. List-prefetch wordt overbodig. DB2 gaat de index nu lezen tijdens de aanpassingen en we

zitten opnieuw met een 'loop'. Op een gelijkaardige manier kan het omschakelen van een ascending index naar een 'descending' of omgekeerd, de zaak in de war sturen.

Besluit

De kern van het probleem is het samengaan van twee factoren:

- er wordt gelezen met een voorwaarde op een geïndexeerde kolom die men gaat aanpassen;
- DB2 is niet op de hoogte van het verband tussen lezen en aanpassen.

De eenvoudigste en theoretisch correcte oplossing blijft het uitschakelen van deze tweede factor door de cursor-select te definiëren met FOR UPDATE OF. We laten dit element niet zomaar weg om via een aantrekkelijker toegangspad een betere performance te krijgen. Doen we dit wel dan moet het alleszins omzichtig gebeuren. Een grondige analyse is nodig. We mogen immers niet vergeten dat dataconsistentie nog altijd belangrijker is dan applicatieperformance.

ABIS PRESENTATIES

ABIS presenteert op volgende conferences!

BMC Summit 2003, Eindhoven, 20-21 november 2003, '*DB2 en WebServices*', Kris Van Thillo (ABIS).

Javapolis 2003, Antwerpen, 3-4 december 2003, '*Transactions and J2EE*', Gie Indesteege (ABIS).

DB2 Data Provider verenigt .NET-clients en DB2

Katrien Platteborze (ABIS)

.NET

Het .NET-framework stoelt op twee basisonderdelen: enerzijds de CLR (Common Language Runtime) en anderzijds de class-library. Bedoeling is 'managed code' uit te voeren.

Geschreven code (C# bijvoorbeeld) wordt eerst gecompileerd naar Microsoft Intermediate Language (MSIL of IL). Vervolgens wordt deze opnieuw gecompileerd door de just-in-time (JIT) compiler van de CLR. Het resultaat is managed code. Dit is de code én de metadata die nodig zijn bij het uitvoeren van het programma. De sourcecode kan in zeer veel talen geschreven worden zelfs in COBOL (multi-language feature); de taal bij uitstek voor een .NET-omgeving is evenwel C#. .NET is niet multi-platform.

De CLR is de omgeving waarin managed code wordt uitgevoerd en beheerd. De metadata-component van de managed code bepaalt welke support services van de CLR worden aangeroepen: denk aan het beheer van geheugen, het aanmaken van threads, etc.

Onafhankelijk van de gekozen programmeertaal kan men steeds gebruik maken van dezelfde class library, die een aantal standaard, generieke diensten omvat. Denk hierbij bijvoorbeeld aan GUI-bouw aan de hand van de 'System.Windows.Forms'-klasse.

ADO.NET

Data access in een .NET-omgeving wordt mogelijk gemaakt met ADO.NET-klasses. ADO.NET voorziet klasses die connecties leggen met een database, SQL-statements uitvoeren, voorzien in transactiemanagement, etc. Alhoewel de naam het anders laat vermoeden heeft ADO.NET niets te maken met ADO (ActiveX Data Objects).

In elke ADO.NET-applicatie kan men typisch twee soorten klasses onderkennen: klasses die afhankelijk zijn van de datasource en die geleverd worden door de dataproviders enerzijds, en klasses die onafhankelijk zijn van de datasource anderzijds. Een typisch voorbeeld van de eerste groep is het 'Connection'-object. Men kiest expliciet voor SQLConnection, OLEDBConnection of DB2Connection. Een voorbeeld van de tweede groep is het 'Dataset'-object dat verder besproken wordt.

Drie grote families van ADO.NET-klasses - providers - kunnen worden onderkend. Een en ander wordt in figuur 1 schematisch weergegeven.

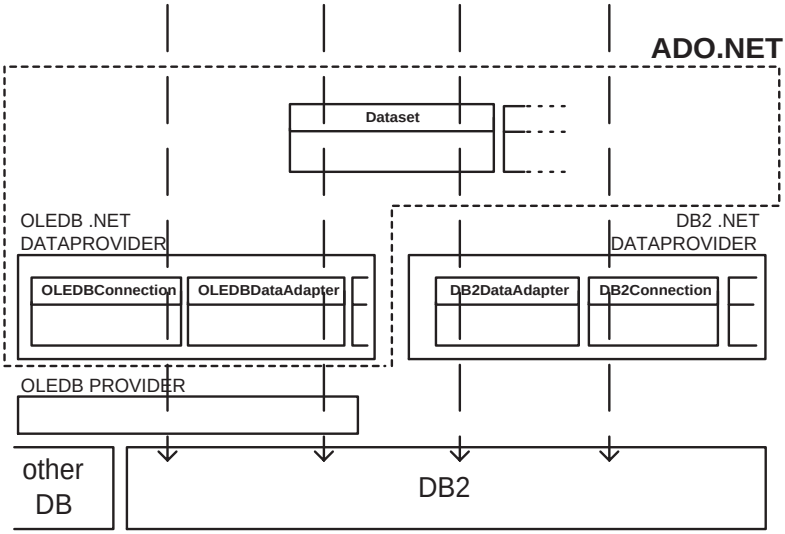
- Generieke providers aangeboden door Microsoft: de OLE DB .NET Data Provider, de ODBC .NET Data provider. Gelijk welke data-

base kan via deze provider benaderd worden, op voorwaarde dat ze OLE DB en/of ODBC ondersteunen;

- Vendor-specifieke providers, aangeboden door specifieke database vendors - denk hierbij aan de IBM DB2 Data Provider, de Oracle Data Provider, etc;

- De SQL Server .NET Data Provider communiceert rechtstreeks met SQL Server via het native datatransfer-protocol van SQL Server.

Figuur 1 : van .NET managed client naar DB2-database via OLE DB .NET Data Provider en via DB2 .NET Data Provider



Visual Studio

Visual studio is de Microsoft-specifieke IDE voor applicatieontwikkeling in een .NET-omgeving. Voor het ontwikkelen van DB2-specifieke .NET-applicaties heeft IBM een aantal add-ins voor Visual Studio beschikbaar gemaakt. Bedoeling is op een snelle en efficiënte wijze DB2-databronnen in .NET te kunnen aanspreken.

Traditionele applicatieontwikkeling

Met .NET en C# kunnen uiteraard traditionele DB2-applicaties worden gecodeerd. Voorbeeld 1 gaat hier wat dieper op in. Vooreerst wordt een connectie met DB2 gedefinieerd en geopend; vervolgens wordt een statement uitgevoerd; tenslotte wordt via een traditionele loop-structuur (cursor aanpak) het resultaat binnengehaald en verwerkt.

Niets nieuws onder de zon, toch?

Voorbeeld 1: traditionele applicatieontwikkeling

```
...
private void button1_Click(object sender, System.EventArgs e)
{
  DB2Connection Con = new DB2Connection("database=ADB;uid=db;pwd=db");
  DB2Command catCMD= testkpConn.CreateCommand();
  catCMD.CommandText = "SELECT plname,pfname FROM persons";
  Con.Open();
  DB2DataReader myReader = catCMD.ExecuteReader();
  while (myReader.Read())
  {
   ...
  }
  myReader.Close();
  testkpConn.Close();
 }
...
```

Een nieuwe architectuur voor applicaties

Microsoft heeft echter ook een volledig 'nieuwe' applicatiearchitectuur uitgewerkt en gerealiseerd onder de naam ADO.NET. Een applicatiearchitectuur die minder 'client/server' gericht is, maar eerder georiënteerd is naar web-based applicaties. De basisprincipes van deze architectuur? Disconnected werken, en integratie met XML.

Aan de basis van deze architectuur ligt het DATASET-object.

Voorbeeld 2: een nieuwe aanpak

```
...
DB2Connection con = new DB2Connection("database=AD3DB;uid=db;pwd=db");
DB2DataAdapter cmdCustomers =
    new DB2DataAdapter("Select substr(pfname,1,5),pname, p_cono, ptel
                        from vmtpersons order by pname", con);
statusBar1.Text ="Loading Courses...";
cmdCustomers.Fill(customersDataSet1, "vmtpersons");
statusBarl.Text ="Updating Grid...";
...
```

Een DATASET is een snapshot, bijgehouden in het geheugen, van een stukje van de database. De DATASET kan één of meerdere tabellen bevatten inclusief de tabeldefinitie met constraints. De gebruiker kan met de data in de DATASET volledig gedisconnecteerd werken - er wordt geen sessie met de database opengehouden. Er zijn dus ook geen locks in de database. Het principe van optimistic concurrency wordt hier gebruikt. Indien updates uitgevoerd op de DATASET doorgevoerd worden naar de database, zal eerst moeten geverifieerd worden of de overeenkomstige gegevens in de database ondertussen niet aangepast werden.

Het verband tussen een DATASET en DB2-tabellen wordt gelegd met het object DB2DataAdapter (zie ook voorbeeld 2). De DB2DataAdapter heeft een Fill methode waarmee de DATASET opgevuld wordt. De connectie moet niet expliciet geopend en gesloten worden. De DB2DataAdapter opent en sluit deze - na het vullen - au-

tomatisch. Dit uiteraard om te garanderen dat gedisconnecteerd wordt gewerkt. Ook updates worden doorgevoerd met DB2DataAdapter methodes.

De DATASET is op verschillende vlakken sterk verwoven met XML. Naast het opvullen van de DATASET met relationele data kan deze even goed opgevuld worden met XML-data. De DATASET kan zowel aan de hand van Xpath dan aan de hand van SQL-statements worden ondervraagd. En zonder enige moeilijkheid kan men de inhoud van de DATASET laten wegschrijven als een XML-document. Dit alles onafhankelijkheid van de oorspronkelijke bron van de data.

Het formaat waarin dat de DATASET intern bewaard wordt is het diffgram-formaat - een XML-formaat dat zowel originele als huidige waarden bijhoudt. Omwille van dit diffgram-formaat is de DATASET een geschikte keuze wanneer men bijvoorbeeld wenst gebruik te maken van:

- optimistic concurrency
- datatransport over het netwerk
- webservices

DB2 en ADO.NET

Zoals reeds eerder vermeld kiest men voor het ontwikkelen van een .NET-applicatie in een DB2-omgeving ofwel voor een OLEDB-tussenlaag, ofwel voor een native DB2-dataprovider. Naast de dataprovider die de klasses bevat heeft IBM ook een aantal add-ins voorzien die ontwikkeling in een Visual Studio omgeving naar DB2 (zowel DB2 voor OS/390 en z/OS als DB2 voor LUW) toe vergemakkelijkt. Het gaat hier zowel over het schrijven van .NET-applicaties met DB2 als datasource als over het gebruik van Visual Studio voor het ontwikkelen van server-side objecten zoals stored procedures, user-defined functions of het creëren van tabellen, views en indexen.

In een volgend nummer staan we stil bij specifieke aandachtspunten die bij het ontwikkelen van DB2 .NET-applicaties moeten worden in ogenschouw genomen. Concreet denken we hierbij aan de impact van .NET-opties en karakteristieken op het hele DB2 databasegebeuren.

DOSSIER 8

Over schema-evolutie ...

'Schema-evolutie' is een belangrijk onderwerp geworden. Het gaat over de mogelijkheid door een RDBMS geboden om met minimale impact op de applicaties, de logische en fysieke structuur van een database-schema aan te passen. Het belang van schema-evolutie is toegenomen onder druk van nieuwe objectgeoriënteerde applicatieontwikkelparadigma's: extreme programming, RUP, etc. Vaak ver vooruit geplande schema-upgrade-acties, die voor een bepaalde tijd data onbeschikbaar zullen maken, zijn niet langer een haalbaar alternatief.

In DB2 voor OS/390 versie 8 zorgen sommige schema-wijzigingen niet langer voor onbeschikbaarheid van de desbetreffende data. Men kan wel in een AREO (Advisory Reorg Pending) toestand terecht komen, wat duidt op mogelijke performance degradatie, typisch het gevolg van deze wijzigingen. Een greep uit de nieuwe mogelijkheden:

a) Data type en lengte van kolommen kan onmiddellijk en online worden gewijzigd. Nieuwe data wordt in het juiste formaat weggeschreven; oude data wordt geherformatteerd wanneer ze gelezen wordt.

b) Het toevoegen van partities aan een tablespace, het roteren van partities na het archiveren van oude partitiedata, en het herbalanceren van partities kan online worden uitgevoerd.

c) Kolommen die in eenzelfde unit-of-work aan een tabel en aan een index worden toegevoegd, zorgen niet langer voor het tijdelijk onbeschikbaar worden van de index.

Daarnaast biedt DB2 voor OS/390 versie 8 ook een aantal aanverwante nieuwe features, die binnen de context van schema-evolutie bijkomende voordelen bieden. Merk echter op dat deze features niet steeds online kunnen worden uitgevoerd. In deze context denken we bijvoorbeeld aan:

- de mogelijkheid om partities aan de maken zonder cluster index
- de mogelijkheid om data clustering onafhankelijk te definiëren van partitionering
- de mogelijkheid om een cluster index te wijzigen.

Maar schema-evolutie behelst nog één ander cruciaal idee: de mogelijkheid om verschillende versies van één schema, gestoeld op éénzelfde fysieke implementatie, naast mekaar aan te bieden. Applicaties kunnen dan beroep doen op de voor hen geschikte versie. Dit alles kan bijvoorbeeld worden gerealiseerd aan de hand van views, triggers en stored procedures. Dit element van schema-evolutie wordt door IBM DB2 nog niet automatisch ondersteund.

Kris Van Thillo (ABIS)

CURSUSPLANNING OKT - DEC 2003

DB2 concepten	375 EUR	24/10 (L), 09/12 (W)
DB2 for OS/390, een totaaloverzicht	1625 EUR	03-07/11(L), 08-12/12 (W)
DB2 UDB, een totaaloverzicht	1625 EUR	03-04&12-14/11 (L)
RDBMS concepten	325 EUR	03/11 (L), 08/12 (W)
Basiskennis SQL	325 EUR	04/11(L), 09/12 (W)
DB2 for OS/390 basiscursus	975 EUR	05-07/11(L), 10-12/12 (W)
DB2 UDB basiscursus	975 EUR	20-22/10 (W), 12-14/11(L)
SQL workshop	700 EUR	27-28/10 (W), 17-18/11 (L), 18-19/12 (W)
DB2 for OS/390 programmering voor gevorderden	1050 EUR	27-29/10 (W), 03-05/12 (L)
DB2 for OS/390: SQL performance	1200 EUR	03-05/11(W), 15-17/12 (L)
DB2 UDB applicatieperformance	400 EUR	10/11 (W)
Database applicatieprogrammering met Java	800 EUR	20-21/10 (L), 13-14/11(W)
Fysiek ontwerp van relationele databases.	700 EUR	06-07/11(L)
DB2 for OS/390 database administratie	1600 EUR	20-23/10 (W), 02-05/12 (L)
DB2 for OS/390 system programming	1650 EUR	05-07/11(L)
DB2 UDB systeembeheer en performance	400 EUR	11/11 (W), 15/12 (L)
DB2 for OS/390 V7 upgrade voor ontwikkelaars	375 EUR	12/11 (W)
DB2 UDB en zijn extenders: XML en text search	200 EUR	23/10 (W), 16/12 (L)
DB2 UDB integratie met MQSeries	200 EUR	23/10 (W), 16/12 (L)

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