ABIS NIEUWSBRIEF VOOR DB2 PROFESSIONALS

JAARGANG 7 - NUMMER 1, AUGUSTUS 2010

OPEN CURSOR

Voor u ligt het eerste nummer van "Exploring DB2" dat uitsluitend in electronisch formaat wordt uitgebracht. Maar dat er verder uitziet zoals u van ons gewoon bent: met korte, heldere uiteenzettingen over onderwerpen die raken aan de kern van ons aller DB2, of belangrijke nieuwe trends in de wereld van het "data management" proberen te plaatsen in hun relatie tot DB2.
Deze maal met o.a. twee externe bijdragen, waaruit blijkt dat "Exploring DB2" toch wel grondig gelezen (en gesmaakt) wordt.

Veel leesplezier!

Het ABIS DB2-team.

IN DIT NUMMER:

- *Persistentiemogelijkheden tussen Java en DB2*, waarin u leest hoe DB2 kan dienen als persistentielaag voor Java-objecten;
- In *Exploratie van de nieuwe explain-mogelijkheden* vindt u een respons door Abe Kornelis (Bixoft) op een artikel in het vorige nummer van Exploring DB2;
- Verder bespreekt *Scalaire functies: Oracle-tegenhangers* de verschillen (en gelijkenissen) tussen DB2 en Oracle voor wat betreft benaming en gebruik van scalaire functies;
- En tenslotte, in *DDL genereren zonder externe tools* leest u hoe Robert Vial (ICU IT) de standaard-applicatie DB2PLI8 gebruikt om de DB2-catalog te lezen.

1

CLOSE CURSOR

In het volgende nummer wordt de reeks DB2 versus ... voortgezet met een vergelijking van scalaire functies tussen DB2 en SQL Server. Verder lichten we een tipje van de sluier over de nieuwe mogelijkheden in DB2 versie 10 voor z/OS.

Persistentiemogelijkheden tussen Java en DB2

Gie Indesteege (ABIS)

Bij het inzetten van Java voor bedrijfskritische toepassingen moeten er een aantal belangrijke aspecten overwogen worden. Naast functionaliteit, performance en beveiliging van de toepassingen, is zeker de consistentie en betrouwbaarheid van de bedrijfsgegevens van cruciaal belang. Java-toepassingen kunnen op verschillende manieren omgaan met data in de database, in de applicatieserver, in de gebruikersinterface. Java-ontwikkelaars moeten dan ook op de meest efficiënte manier gebruik maken van de vele mogelijkheden om data uit de toepassing te synchroniseren met de databases.

In een Java-toepassing worden gegevens in objecten bijgehouden in tijdelijk geheugen. Om deze gegevens permanent te bewaren, worden ze *gepersisteerd*: dat wil zeggen weggeschreven naar een permanente geheugenstructuur, bvb. een database. In dit artikel worden enkele technieken ingeleid om data op een gecontroleerde manier te gebruiken binnen de context van databasetoegang. Een belangrijk aspect hierbij is het overbruggen van het verschil tussen objectstructuur en databasestructuur.

Gebruikte technieken

Binnen Java zijn er verschillende technieken mogelijk om een connectie te maken met de database:

- native JDBC (Java Data Base Connectivity)

  Dit is de basistechniek waarvan de hieronderstaande technieken ook gebruik maken. Het biedt de mogelijkheid aan de Java-ontwikkelaar om eigenhandig SQL-statements uit te voeren op een database en de resultaten op te vragen. JDBC is onderdeel van de JavaSE-standaard (packages java.sql en javax.sql).

- framework JPA (Java Persistence Architecture)

  Oorspronkelijk ontwikkeld buiten de Java-standaard om, is JPA vandaag (sinds Java EE5) de standaard voor persistentie en object/relationele mapping binnen de Java SE en EE platformen. Het framework levert de nodige APIs om gewone Java-objecten (Plain Old Java Objects) op basis van een object/relationele mapping-specificatie (in Java-annotaties en/of XML-descriptoren) te synchroniseren met een relationele database d.m.v. een persistence manager. Bovendien is er ondersteuning voor statische en dynamische SQL via een eigen JPA-querytaal. (JPA is beschikbaar in de Java-package javax.persistence)

  N.B.: JPA-entiteiten vervangen de vroegere Entity Enterprise Java Beans.

- open source frameworks

Verschillende leveranciers en ontwikkelaars uit de open source-wereld leveren persistence frameworks die op dezelfde basisprincipes als JPA gestoeld zijn. De bekendste frameworks zijn Hibernate, Toplink en JDO. Deze zijn geen onderdeel van de Java EE standaard. Een uitgebreide lijst is te vinden op *http://java-source.net/open-source/persistence*

- SQLJ

  Biedt de mogelijkheid om statische SQL te gebruiken in Java-code. SQLJ-programma's moeten eerst door een preprocessor (de SQLJ-translator) worden voorbereid om JDBC-code te genereren die de statische packages binnen DB2 op te roepen, alvorens zij als Java-toepassingen kunnen gebruikt worden.

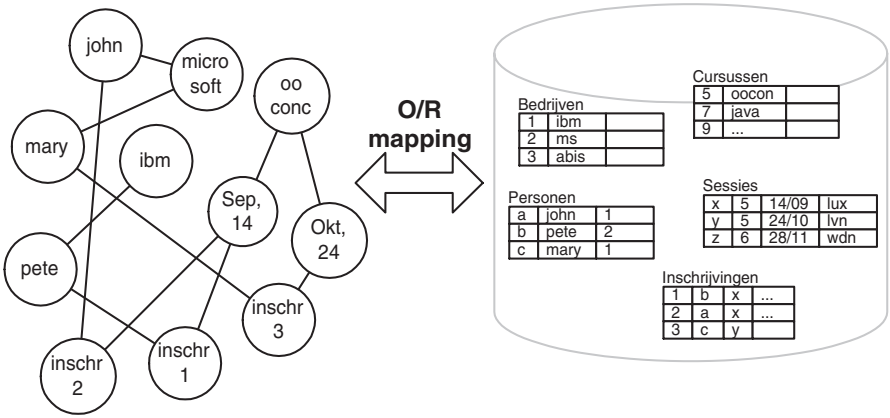
  SQLJ is geen onderdeel van de Java EE standaard.

Wij zullen in dit artikel enkel de native JDBC en het JPA-framework bespreken.

Objecten versus databases: "O/R mapping"

Bij het maken van database-gerichte toepassingen in Java botsen we op een probleem: het logische domeinmodel van een object-georiënteerde toepassing komt gewoon niet overeen met de structuur van relationele databases! Of meer specifiek: Java structureert data in klassen die met elkaar verbonden zijn door overerving, compositie, associatie e.d.; relationele databases daarentegen structureren data in tabellen die met elkaar verbonden worden via foreign key-relaties. Bovendien komen de attributen van klassen niet steeds overeen met de kolommen uit een tabel. Het overbruggen van dit structuurprobleem is één van de uitdagingen bij de ontwikkeling van bedrijfskritische data-gerichte Java-toepassingen.

Het "Object-Relationele mapping" probleem wordt geïllustreerd in de volgende figuur.



Native JDBC

JDBC (Java Data Base Connectivity) is de standaard API om in Java-programma's SQL-instructies te gebruiken. JDBC werkt op basis van dynamische SQL-vragen.

- JDBC-connectie naar de database

Vooraleer met behulp van JDBC SQL-vragen gesteld kunnen worden aan de database, moet eerst duidelijk aangegeven worden van welke database gebruik gemaakt moet worden. Daarom moet eerst een connectie gelegd worden met een bepaalde server. Het creëren van de connectie gebeurt met behulp van een "driver". Sinds DB2 v8 wordt gewerkt met een zogenaamde type 4 driver als meest efficiënte toegang tot de database. Deze driver connecteert rechtstreeks (via de DRDA Application Requester) naar de server (of host). Voor specifieke informatie i.v.m. JDBC-drivers in DB2 verwijzen we naar *http://www.ibm.com/software/data/db2/ad/java.html*

De driver activeren en de connectie ("**Connection**") creëren gebeurt door het gebruik van ofwel de "**DriverManager**" klasse (JDBC 1.0)

```
ourConnection = DriverManager.getConnection( serverURL, userID, password);
```

ofwel een "**DataSource**" object (vanaf JDBC 2.0):

```
ourConnection = dataSource.getConnection();
```

Bij gebruik van zo'n DataSource-object wordt de database dynamisch opgezocht en tegelijk wordt ook de nodige driver geladen.

N.B. Bij toepassingen die draaien onder controle van een Java EE applicatie server wordt de DataSource gedefinieerd en beheerd door de applicatieserver. Om er gebruik van te kunnen maken wordt de DataSource dynamisch opgezocht via een zogenaamde "lookup" in een centrale repository of directory **JNDI** (Java Naming en Directory Interface). Het gebruik van datasources biedt naast zijn flexibiliteit nog extra voordelen zoals connection pooling, transactie management, beveiliging, ...

- JDBC - SQL

JDBC SQL komt in grote lijnen overeen met de dynamische SQL uit andere programmeertalen (bvb. COBOL, C,...). Het SQL "statement" wordt in een tekstvariabele (String) weggeschreven en eventueel in de loop van het programma nog gewijzigd of aangevuld. In een eerste fase wordt een "**Statement**" aangemaakt voor een bepaalde "**Connection**". De beschikbare methodes hiervoor zijn: **`createStatement`** voor niet-geparametriseerde SQL, **`prepareStatement`** voor geparametriseerde SQL, en **`prepareCall`** voor het oproepen van een stored procedure. De connection bepaalt op welke server de SQL zal uitgevoerd worden.

```
String sqlStmt = "UPDATE rekeningen SET saldo = ? WHERE rekening_nr = ? " ;
PreparedStatement ourStatement = ourConnection.prepareStatement( sqlStmt ) ;
```

Prepared statements hebben bovendien het extra voordeel dat ze onmiddellijk, eenmalig geoptimaliseerd worden.

De evidente tweede stap is - indien nodig - eerst de vraagtekens (de JDBC-parameters) invullen en daarna het SQL-statement uitvoeren. Dit laatste kan m.b.v. **executeQuery**, **executeBatch**, **executeUpdate** of kortweg **execute**.

```
ourStatement.setInt(1, 10000) ; // Het eerste vraagteken wordt vervangen door
                                // een nieuw saldo van 10000
ourStatement.setString(2, "BE78 1231 2345 6782") ; // Het tweede vraagteken
                               // wordt vervangen door het bankrekeningnummer.
ourStatement.executeUpdate() ;
```

Wanneer een SQL SELECT wordt gebruikt als statement, worden de resultaten van deze SQL-vraag weggeschreven in een zogenaamde "**ResultSet**". Dit is een verzameling van records die worden opgehaald uit de database. Deze Resultset wordt dan in de applicatie verder gebruikt om de gegevens door te sturen naar de gebruikersinterface, om controles te doen, of om eventuele wijzigingen voor te bereiden.

```
String sqlStmt = "SELECT pno, plname from TUTPERSONS WHERE plname LIKE = ? ";
PreparedStatement ourStatement = ourConnection.prepareStatement( sqlStmt );
ourStatement.setString(1, "'A%'"); // Het vraagteken wordt vervangen door de
                                   // zoekstring "A%", om zo alle personen
                                  // te vinden waarvan de naam begint met "A".
ResultSet rs = ourStatement.executeQuery();

while (rs.next()) {
   persoonsNr = rs.getShort(1);
   achterNaam = rs.getString("plname");
   // verwerken van gegevens
}
```

- JDBC-foutenafhandeling e.a.

Naast het puur functioneel uitvoeren van SQL denkt de ontwikkelaar ook onmiddellijk aan "exception handling": het opvangen van mogelijke fouten. Hiervoor biedt de JDBC-API de nodige methodes en klassen (zoals b.v. de klasse SQLException). Een voorbeeld:

```
try { ... }
catch(SQLException se) {
   System.out.println("Probleem met SQL: SQL-code is " + se.getErrorCode());
}
```

Bovendien moet er ook aandacht besteed worden aan transactiebeheer binnen de toepassingen: commit-processing, locking-opties en synchronisatie. Binnen JDBC zijn hiervoor de nodige methodes en klassen voorzien om de connection te manipuleren.

Bijvoorbeeld:

```
ourConnection.setTransactionIsolation(TRANSACTION_SERIALIZABLE);
// Zet het transactie-isolatieniveau op "repeatable read" voor de connectie.
ourConnection.commit() ; // Geeft commit-instructie aan de connectie.
```

- Persistence framework JPA

Voor kleinere toepassingen die niet zwaar belast worden en voornamelijk enkel gegevens lezen, is een native JDBC-oplossing aangewezen. Wanneer het aantal gebruikers en/of de hoeveelheid gegevens

groeit, is het echter aangewezen om gebruik te maken van een persistence framework. Dit laat de ontwikkelaars toe om zich te concentreren op de business-aspecten van de toepassing. De synchronisatie met de onderliggende database wordt overgenomen door de persistence of **entity manager** en door het framework. Dit levert een hoop voordelen: automatisch gebruik van resource pooling, transactie management, beveiliging en flexibele koppeling aan database structuur, ... om er maar enkele op te noemen.

JPA (Java Persistence Architecture) laat toe om de database te gebruiken zonder zelf SQL te schrijven. Hiervoor gaat men de attributen van een Javaklasse associëren/mappen met één of meerdere database-tabellen. Deze mapping kan gebeuren d.m.v. Java-annotaties in een gewone Javaklasse (POJO).
Resultaat: een zogenaamde **JPA-Entiteit**:

```
@Entity
@Table(name="TUTPERSONS") // verwijzing naar de tabelnaam in de server-database
public class Persoon implements Serializable {
  @Id // aanduiding van PK kolom
  @Column(Name="PNO") // verwijzing naar kolomnaam in tabel
  private short persoonsNr;
  @Column(name="PLNAME") // idem
  private String achterNaam;
  ....
}
```

en aan de hand van configuratiebestanden in XML-formaat (`persistence.xml` en eventueel `orm.xml`)

```
<persistence-unit name="KlantAdmin">
  <jta-data-source>jdbc/KlantDB</jta-data-source>
  <class>be.abis.entities.Persoon</class>
  <properties>
    <property name="openjpa.jdbc.Schema" value="ABIS"/>
  </properties>
</persistence-unit>
```

die o.a. de koppeling met de juiste database/datasource definiëren.

Het controleren van de persistentie gebeurt door de **entity manager** van de toepassing, die instaat voor de communicatie met de database.

```
import javax.persistence.*;
EntityManagerFactory emf =
                      Persistence.createEntityManagerFactory("KlantAdmin");
EntityManager em = emf.createEntityManager();
```

N.B. Bij toepassingen die draaien onder controle van een Java EE applicatie-server wordt de entity manager beheerd door de applicatieserver. Om er gebruik van te kunnen maken wordt de entity manager dynamisch opgezocht en geïnjecteerd op basis van de annotaties voor de **persistence context**.

```
@PersistenceContext(unitName = "KlantAdmin")
private EntityManager em;
```

Als de entity manager bekend is, wordt deze gebruikt om de database te benaderen via de beschikbare methodes. Enkele voorbeelden:

```
Persoon p = em.find(Persoon.class,id); // opzoeken van bestaande persoon
                                       // op basis van zijn nummer
Persoon px = new Persoon(....);
em.persist(px) ; // toevoegen van nieuwe persoon in de personen-tabel
```

De nodige SQL (in dit geval: SELECT en INSERT) wordt dan gegenereerd op basis van de annotaties en configuratie, maar de ontwikkelaar kan ook zelf nog SQL-instructies schrijven en die aanleveren aan de entity manager. Merk op dat in deze SQL geen enkele verwijzing staat naar namen uit de DB-structuur, maar gewerkt wordt met logische namen uit de Java-klasse Persoon.

```
@NamedQuery(name="Persoon.zoekOpNaam",
         query="SELECT p FROM Persoon p WHERE p.achterNaam LIKE :persoonsNaam")
            // Annotatie bij de klasse Persoon

String naam = "Bond";
Query q = em.createNamedQuery("Persoon.zoekOpNaam"); // bepalen van de SQL
q.setParameter("persoonsNaam", naam); // invullen van de waarde voor achternaam
Collection<Persoon> lijstPersonen = q.getResultList(); // uitvoeren, met als
                                     // resultaat een verzameling van personen
```

De logische foutafhandeling moet natuurlijk nog steeds binnen de toepassing gebeuren. Als de object/relationele mapping ingewikkelder wordt omwille van complexere klassemodellen, komen de voordelen van het persistence framework pas echt tot uiting. Het JPA framework laat toe om enkelvoudige en meervoudige relaties, uni- of bidirectioneel, met of zonder overerving, te definiëren. Als de verschillende associaties en relaties goed gedefinieerd zijn via de annotaties in de verschillende Javaklassen, dan zal de entity manager in alle omstandigheden de meest efficiënte manier uitzoeken om, transparant voor de ontwikkelaar, de object-data te synchroniseren met de onderliggende databasetabel(len). Verder worden complexe aspecten zoals transactie-isolatie, locking, resource pooling en dergelijke ook volledig beheerd door het framework. Hiervoor moeten natuurlijk ook de juiste annotaties en configuraties worden aangemaakt.

Conclusie

Java biedt verschillende mogelijkheden om te werken met gegevens uit een database. Afhankelijk van de grootte van de toepassing, de belasting van het systeem en het gemak waarmee toekomstige wijzigingen doorgevoerd moeten worden, kan men beslissen om gebruik te maken van een JPA framework ofwel te werken met native JDBC. Maar eenmaal men de voordelen van een framework zoals JPA ontdekt heeft zal de keuze snel gemaakt zijn. Indien de ontwikkelaars zelf de SQL wensen te schrijven waarmee met de database gecommuniceerd wordt, is dit in alle gevallen nog steeds mogelijk.

Exploratie van de nieuwe Explain-mogelijkheden

Abe Kornelis (Bixoft)

Het artikel over EXPLAIN in het vorige nummer van Exploring DB2 (januari 2010) eindigde met een vraag aan de lezers om een query in te sturen voor het presenteren van informatie uit de extended explain tabellen. Deze uitdaging heb ik natuurlijk graag aangenomen.

De query

Voor de broodnodige variatie wilde ik explain-informatie bekomen over een query die gebruik maakt van een Common Table Expression en ten minste één gepartitioneerde tabel. Een catalog query was daardoor minder geschikt. Dus heb ik een query gebouwd op de IVP database (zie *http://publib.boulder.ibm.com/infocenter/dzichelp/v2r2/topic/com.ibm.db29.doc.inst/db2z_ensurev8ivpsavail.htm*) want deze bevat een gepartitioneerde tabel. Bovendien hebben de meeste installaties de IVP database geïnstalleerd zodat iedereen uit de voeten kan met deze query:

```
with managers (mgrno, mgrname)
 as (select distinct
            mgr.empno
          , strip(coalesce(firstnme, '')) CONCAT ' ' CONCAT
            strip(coalesce(lastname, '???')) as mgrname
     from  DSN8810.DEPT dept  join  DSN8810.EMP  mgr  on  mgr.empno = dept.mgrno
    )

select mgr.mgrname, dept.deptname, emp.lastname, emp.firstnme

from   DSN8810.EMP emp   left outer join   DSN8810.DEPT dept
                                           on  dept.deptno = emp.workdept
                         left outer join   managers mgr
                                           on  mgr.mgrno = dept.mgrno

where  emp.empno between '000000' and '150000'
  and  emp.workdept between 'A' and 'EEE'

order by case when dept.deptno IS NULL     then '*No dept'
              when mgr.mgrno IS NULL       then '*No mgr'
              else                              mgr.mgrname
         end,
         emp.empno
```

De resultaat-set van deze query vindt u terug in *Appendix A: output van de query op de IVP-database.* op p. 20.

Dat dit geen bijzonder efficiënte query is, behoeft geen betoog. We willen namelijk, via Explain, te weten komen hoe de optimizer een dergelijke relatief complexe query implementeert. Optimisatie van deze query is dan ook niet het doel van dit artikel.

Zorg er allereerst voor dat de extended explain tabellen aangemaakt zijn. (Zie de Performance Monitoring and Tuning Guide voor de details; deze tabellen worden trouwens automatisch aangemaakt door OSC. U kunt ook member DSNTIJOS uit de SDSNSAMP gebruiken.)

Explain met range-informatie

Na uitvoering van

```
EXPLAIN ALL SET QUERYNO=114 FOR
```

met bovenstaande query hebben we de diverse explain-tabellen voorzien van data.

Een basis-query op alleen de PLAN_TABLE levert het inzicht op, dat er 7 stappen (tabelregels) zijn voor het uitvoeren van deze query. Uit de DSN_SORT_TABLE leren we dat er 4 sort-stappen zijn: twee voor de order-by clause, 1 voor de distinct clause in de Common Table Expression (CTE), en 1 voor het uitvoeren van de join met de resultaatset van de CTE.

Als eerste stap wilde ik mijn standaard-query voor PLAN_TABLE uitbreiden met gegevens voor een eventuele page-range scan. Dat maakt immers een tablespace scan (accesstype=R) iets minder bezwaarlijk... Hiertoe dienen we de betreffende info uit de tabel DSN_PGRANGE_TABLE te gebruiken. Ik heb er voor gekozen om het aantal deelnemende partities toe te voegen aan de access kolom. De query vindt u terug in *Appendix B: Query on PLAN_TABLE with Page-Range info* op p. 21.

Dit levert de volgende resultaat-set op:

```
PBLK PLNO TYPE  JOIN METHOD TYPE  CREATOR TABLE   NDXNAME ACCESS PREF XO MC GJOUN
  QBLK OPSQ
1 3 1  TABLEX       First  Table DSN8810 DEPT  XDEPT2  I          XO
1 3 2  TABLEX Innr  NLjoin Table DSN8810 EMP   XEMP1   I             1
1 3 3  TABLEX       Sort                                              U
  1 1  SELECT       First  Table DSN8810 EMP   XEMP2   I(2)  LST     1
  1 2  SELECT L/R   NLjoin Table DSN8810 DEPT  XDEPT1  I             1
  1 3  SELECT L/R   NLjoin Work  TU00001 MANAGERS       T/R                N
  1 4  SELECT       Sort                                            O
```

Hierbij vallen de volgende zaken op:

- De formattering is niet ideaal -- vooral de eerste 5 kolommen kunnen wel een paar voorloopspaties gebruiken. Echter, de presentatie was niet het doel van deze exercitie.
- De sortering is - binnen de query - aflopend op Parent Qblockno. Ik had hem eerst oplopend gedefinieerd, maar dat gaf een (naar mijn gevoel) onlogische volgorde. Met aflopende sort sluit de presentatie beter aan bij wat DB2 doet in volgordelijke zin tijdens de uitvoering van de query.
- In de kolom Access zien wij een I(2) - overigens met LST in de PREF kolom, oftewel met List Prefetch - hetgeen inhoudt dat DB2 een Index scan over 2 partities zal uitvoeren. Aanzienlijk beter dan een volledige Index scan, want die zou alle 5 partities hebben moeten doorlezen.

- In diezelfde kolom Access zien wij ook een T/R. Dit houdt in dat de resultaten van de CTE zijn gematerialiseerd en nu benaderd worden via een sparse index. Dat is waar de T voor staat. Als dit niet lukt, dan zal DB2 terug vallen op een scan van de resultaat-set van de CTE; dit alternatief wordt weergegeven door de R.
- Informatie over locking wordt hier niet weergegeven. Wie hierin is geïnteresseerd, kan de betreffende informatie makkelijk zelf toevoegen, want die is beschikbaar via de kolom TSLOCKMODE van de PLAN_TABLE.

Explain met range- en sort-informatie

Vervolgens wilde ik zien, of ik ook de informatie uit DSN_SORT_TABLE in deze query kon opnemen om zo een beter totaal-beeld van de query te verkrijgen. Complicatie hierbij is, dat dit aanvullende rijen uit een andere tabel betreft, en dat er dus een UNION ALL moet worden toegepast. Op zich geen punt, maar het heeft wel enkele consequenties. Allereerst dient de ORDER BY clausule aangepast te worden. Deze kan niet langer met kolomnamen werken, maar moet nu worden geformuleerd met gebruikmaking van kolomnummers. Dat maakt de query minder leesbaar en begrijpelijk, maar het is niet anders.

Tevens ontstaat nu de situatie dat beide sub-queries een WHERE-clausule bevatten waarin de juiste query moet worden geselecteerd voor het ophalen van de explain data. Ik houd daar niet van, omdat het te vaak gebeurt dat de WHERE clausules uit de pas gaan lopen, waardoor het eindresultaat niet meer bruikbaar is. Daarom CTE QUERIES toegevoegd om het nummer (of de nummers) van de te explainen query/queries te definiëren.

Verder bleek ik nog twee extra CTEs nodig te hebben om de te sorteren tabelnummers en de bijbehorende tabelnamen te vinden. Om de resultaten van deze CTEs in omvang te beperken, heb ik gebruik gemaakt van de CTE QUERIES die de relevante query-nummers bevat. Zodoende vervangt de CTE QUERIES nu in totaal 4 WHERE clausules.

Aldus had ik drie CTEs om mijn query verder mee uit te bouwen. Dit hield echter een risico in: bij gebruik van drie of meer CTEs kan een CONCAT functie in de resultaat-set resulteren in kolomwaarden die een extra voorloop-byte krijgen met waarde X'00', wat erg storend kan zijn. In deze query komt dat gelukkig niet voor, dat komt omdat niet voldaan is aan de 'eis' voor het optreden van dit probleem dat er tussen de CTEs een specifieke relatie moet bestaan.

Nadat ik dat had geverifieerd kon ik de query daadwerkelijk verder uitwerken. Het eerste deel van de query bleef vrijwel ongewijzigd. Eigenlijk hoefde alleen de WHERE-clausule aangepast te worden. De selectie op query-nummer moest worden vervangen door een inner join met de relevante query-nummers (CTE queries). Maar omdat de SORT-gegevens nu uit een andere tabel gehaald gaan worden, kunnen de rijen met METHOD=3 worden overgeslagen.

Voor het toevoegen van de sorteergegevens is een tweede query aan de eerste toegevoegd. Deze levert natuurlijk een resultaat-set met dezelfde kolommen, maar haalt haar informatie uit DSN_SORT_TABLE in plaats van PLAN_TABLE. Dit levert de query op van *Appendix C: Query on PLAN_TABLE with Page-Range info and sorting info* op p. 23, met als resultaat-set:

```
PBLK PLNO TYPE   JOIN METHOD TYPE  CREATOR TABLE   NDXNAME ACCESS PREF XO MC GJOUN
  QBLK OPSQ
---- ---- ------ ---- ------ ----- ------- ------- ------- ------ ---- -- -- -----
1    3    1      TABLEX       First  Table DSN8810 DEPT    XDEPT2  I            XO
1    3    2      TABLEX Innr NLjoin Table DSN8810 EMP     XEMP1   I               1
1    3    3 A    TABLEX      Sort   Table DSN8810 EMP                             U
  1  1    1      SELECT       First  Table DSN8810 EMP     XEMP2   I(2)   LST      1
  1  1    2      SELECT L/R  NLjoin Table DSN8810 DEPT    XDEPT1  I               1
  1  1    3     SELECT L/R  NLjoin Work  TU00001 MANAGERS        T/R                N
  1  1    3 A   SELECT      Sort   Work  TU00001 MANAGERS        T/R                N
  1  1    4 A   SELECT      Sort   Table DSN8810 EMP                               O
  1  1    4 B   SELECT      Sort   Table DSN8810 EMP                               O
```

Ook hierbij zijn enkele kanttekeningen te plaatsen:

- De kolommen PAR en CTEREF zijn nu weggelaten: in dit voorbeeld waren ze leeg.
- Ten opzichte van de vorige query zijn twee rijen verdwenen. Dit zijn de rijen die een sort aangaven (method = 3). Ik had ze ook kunnen laten staan, maar dan zouden ze redundant aanwezig zijn geweest. Kwestie van voorkeur. Bij join-met-sort verschijnt een deel van de informatie wel dubbel. Dit zou te onderdrukken zijn, maar het stoorde mij niet. Opnieuw: kwestie van voorkeur.
- 4 nieuwe rijen zijn toegevoegd op basis van de inhoud van DSN_SORT_TABLE. Netto dus twee rijen meer dan in de vorige query. Aldus hebben wij nu meer informatie tot onze beschikking dan uit de vorige query.
- De sortering is ietwat onhandig. Bij een JOIN die een sort vereist wordt de JOIN getoond vóór de bijbehorende SORT, terwijl vanzelfsprekend de SORT altijd zal worden uitgevoerd vóór de JOIN; niet erna. Dit is wel oplosbaar, maar valt buiten de scope van dit artikel.
- Kolom OPSQ oftewel MIXOPSEQ bevat nu nummers voor de rijen uit PLAN_TABLE (niet zichtbaar in ons voorbeeld) en letters voor de verschillende sort-steps (SORTNO) uit de sort-tabel. Dit is in de eerste plaats gedaan om te voorkomen dat er eventueel dubbele nummers zouden verschijnen, maar heeft als bijkomend voordeel dat het onderscheid tussen de beide rij-typen zichtbaar blijft.
- We kunnen nu ook beter zien hoe dat zit met die access T/R: de resultaat-set van de CTE wordt eerst gesorteerd, dat levert een tussenresultaat op die middels sparse index access efficiënt te benaderen valt. Mocht DB2 om wat voor reden ook de sort niet kunnen uitvoeren, dan valt DB2 terug op de 'gewone' tablespace scan.
- Het lijkt net of de Order by clausule wordt gerealiseerd door een dubbele sort. Als we goed kijken, dan zien we echter dat de eerste sort exact identiek is aan de sort voor DISTINCT in de CTE. Nader onderzoek door Bart Steegmans van IBM België heeft uitgewezen dat de eerste sort niet wordt uitgevoerd. De betreffen-

de rijen in DSN_SORT_TABLE en DSN_SORTKEY_TABLE zijn onterecht aangemaakt door explain. IBM heeft hiervoor APAR PM16586 geopend.

Tenslotte:

Ik had slechts een zeer beperkte set explain gegevens tot mijn beschikking. Ik kan dus niet garanderen dat deze queries de explain data ook bruikbaar weergeven voor 'echte' SQL uit uw ontwikkel- of productie-omgevingen. Mocht u een defect vinden, dan hoor ik dat natuurlijk graag. Ook andere opmerkingen of vragen zijn van harte welkom.

Dit artikel, inclusief de queries, is ook te downloaden van *http://www.bixoft.nl/hollands/explain1.htm.*

OVER DE AUTEUR

Abe Cornelis (°1962), van origine assembler-programmeur, houdt zich sinds 2000 bezig met DB2 infrastructuur, performance, tuning en kwaliteitsbewaking. Abe is tegenwoordig werkzaam als onafhankelijk consultant. Hij helpt u graag om complexe queries te temmen.

Contactgegevens: *abe@bixoft.nl*, tel. (+31)-(0)6227.554.01

Scalaire functies: Oracle-tegenhangers

Steven Scheldeman (ABIS)

Dit is het tweede artikel in de vergelijkende reeks over de SQL scalaire functies van DB2 en andere relationele database-systemen.

Het onderwerp is deze maal de vergelijking tussen enerzijds de scalaire functies zoals ze op het DB2-platform gebruikt worden en anderzijds hun tegenhangers op Oracle. In geval van ontbreken van éénduidige tegenhangers worden alternatieven aangedragen.

Vooraf even twee opmerkingen. Ten eerste zal deze lijst niet exhaustief zijn. Enkel de meest gebruikte scalaire functies komen aan bod. Voor een volledige lijst verwijzen we graag naar de manuals.

Ten tweede is de lijst opgesteld met DB2 als invalshoek. Dit houdt in dat er geen Oracle-functies zijn opgenomen waarvoor geen DB2-tegenhanger bestaat. Opnieuw verwijzen we naar de manuals voor een volledig overzicht.

We proberen in wat volgt de functies zodanig te groeperen, dat onmiddellijk duidelijk wordt waar verschillen bestaan; en hoe hiermee dient te worden omgegaan.

Graag toch even aandacht voor het volgende: zelfs indien DB2 en Oracle een identieke scalaire functie hanteren voor een specifieke operatie, wil dit niet steeds zeggen dat beide functies volledig identiek zijn. Een typisch voorbeeld: de functie die in DB2 het aantal seconden sinds middernacht bepaalt (MIDNIGHT_SECONDS), levert een numerieke waarde op; in Oracle (via TO_CHAR) is dit standaard een char. Zelfs als beiden een 'tekenset' als output leveren kan dit in b.v. DB2 een char zijn, en in Oracle een varchar.
Met dit soort onderscheid houden we in wat volgt geen rekening.

Numerieke functies

Volgende lijst geeft een overzicht van DB2-functies die in Oracle standaard op een identieke wijze kunnen worden toegepast.

```
ABS(n), ACOS(n), ASIN(n), ATAN(n), ATAN2(n, m), CEILING(n), COS(n), COSH(n),
EXP(n), FLOOR(n), LN(n), MOD(n,m), POWER(m, n), ROUND(n[,m]), SIGN(n), SIN(n),
SINH(n), SQRT(n), TAN(n), TANH(n)
```

De volgende DB2-functies bestaan echter niet in Oracle. We geven rechts een Oracle-alternatief aan met dezelfde functionaliteit.

`ATANH(n)`	`LN((1+n)/(1-n))/2`
`DEGREES(n)`	`(45*n/ATAN(1))`
`RADIANS(n)`	`n*ATAN(1)/45`
`LOG10(n)`	`LOG(n, 10)`
`RAND([n])`	gebruik Oracle PL/SQL functies en procedures in DBMS_RANDOM
`TRUNCATE(n, m)`	`TRUNC(n, m)`

Tekst-manipulatie-functies

In wat volgt herkennen we verschillende argument-'types', als volgt:

- *(c1,c2)* vertegenwoordigen *"character strings"*
- *(n,m)* vertegenwoordigen numerieke waarden
- *(a,b)* vertegenwoordigen individuele letters
- *(set, to-set, from-set)* vertegenwoordigen reeksen van letters.

Volgende lijst geeft een overzicht van DB2-tekstfuncties die in Oracle standaard op een identieke wijze kunnen worden toegepast:

```
LENGTH(c1), SUBSTR(c1, m[,n]), CONCAT(c1, c2), c1 || c2, LTRIM(c1), RTRIM(c1),
LOWER(c1), UPPER(c1), CHR(n), REPLACE(c, a, b), TRANSLATE(c1, to-set, from-set)
NULLIF(c1, c2), COALESCE (c1, c2, c3, ...)
```

Volgende DB2-functies vergen soms wat extra codeerwerk in Oracle.

```
LEFT(c1, n)          SUBSTR(c1,1,n)
RIGHT(c1, n)         SUBSTR(c1,-n, n)
STRIP(c1[,{B,L,T}[,x]TRIM(leading|trailing|both x from c1)
                     LTRIM, RTRIM
INSERT(c1, n, m, c2)SUBSTR(c1,1,(n-1))||c2||substr(c1,(n+m))
LCASE(c1)            LOWER(c1)
UCASE(c1)            UPPER(c1)
LOCATE(c2, c1[,n])   INSTR(c1, c2[,m[,n]])
```

Voor volgende functie is een alternatief niet eenvoudig generiek neer te schrijven.

```
REPEAT(c1,n): maakt een string aan die bestaat uit n keer c1
```

Datum- en tijdmanipulatie-functies

De belangrijkste verschillen tussen DB2 en Oracle wat scalaire functies betreft, bevinden zich ongetwijfeld in deze categorie.

Merk op dat afhankelijk van de functie het resultaat een numerieke waarde, een datum of een *character string* kan zijn. Input kan ook variëren: een datum *(d)*, een tijd *(t)*, een timestamp *(ts)*, een *character string* die een geldige representatie van een datum is *(c)* of een numerieke waarde tussen 1 en 3652059 *(n)* die zal geïnterpreteerd worden als het aantal dagen sinds 1 januari van het jaar 1.

In wat volgt geven we voor elke individuele DB2-functie een specifiek Oracle-alternatief. We doen een poging e.e.a. logisch te groeperen.

```
Systeemvariabelen

CURRENT DATE        CURRENT_DATE
CURRENT TIMESTAMP   CURRENT_TIMESTAMP
CURRENT TIME        TO_CHAR(CURRENT_TIMESTAMP, 'HH:MM:SS')
```

Werken met onderdelen van datums/timestamps:

```
enkel in DB2:
DAY(d/ts), HOUR(d/ts) MINUTE(d/ts), MONTH(d/ts), SECOND(d/ts), YEAR(d/ts)
```

```
in zowel DB2 als Oracle (SQL-standaard):
EXTRACT (DAY| HOUR| MINUTE | MONTH | SECOND | YEAR FROM d of ts)

Andere DB2-functies links, met hun Oracle-equivalent rechts:
DAYOFWEEK(d of ts)  TO_CHAR(d of ts, 'D')
DAYOFYEAR(d of ts)  TO_CHAR(d of ts, 'DDD')
QUARTER(d of ts)    TO_CHAR(d of ts, 'Q')
WEEK(d of ts)       TO_CHAR(d of ts, 'W')
TIME(d of ts)       TO_CHAR(d of ts, 'hh:mi:ss')
MICROSECOND(ts)           TO_CHAR(ts,'ff')
MIDNIGHT_SECONDS(d of ts) TO_CHAR(d of ts, 'SSSSS')
JULIAN_DAY(d of ts)       TO_CHAR(d of ts, 'J')
DAYS(d of ts)             SYSDATE - TO_DATE('01.01.0001', 'DD.MM.YYYY')

Rekenen met datums en tijd
in DB2 - labelled durations:
n YEARS, n MONTHS, n DAYS, n HOURS, n MINUTES, n SECONDS
in Oracle - geen labelled durations:
ADD_MONTHS(d of ts, n)
berekeningen op data zijn toegelaten, met de dag als eenheid, bijvoorbeeld:
+1 voeg één dag toe
+3/24 voeg drie uur toe
```

Conversiefuncties

Deze functies nemen het argument en vormen dit om tot een ander datatype. De argumenten kunnen numeriek *(n)* of alfanumeriek *(c)* of willekeurig (*x*) zijn. Het kan een datum *(d)*, een tijd *(t)* of een timestamp *(ts)* zijn. Soms wordt het gewenste datatype *(type)* expliciet meegegeven, soms het gewenste formaat *(fmt)* en soms ook een lengte *(len)*. Het resultaat van de functie hangt af van de functie zelf. De verschillen tussen DB2 en Oracle zijn aanzienlijk; slechts een beperkt aantal mogelijkheden wordt vermeld.

Volgende standaard-SQL conversiefunctie wordt zowel door DB2 als door Oracle ter beschikking gesteld:

```
CAST (x AS type)
```

Andere DB2-functies links, met hun Oracle-equivalent rechts:

```
DATE (ts, n, c)     TO_DATE (argumenten*)
TIMESTAMP (d, c, t) TO_TIMESTAMP (argumenten*)
CHAR(d, c, n)       TO_CHAR (argumenten*)
BLOB (x)            TO_BLOB (argumenten*)
CLOB (x)            TO_CLOB (argumenten*)
DECIMAL(x)          TO_NUMBER (argumenten*)
DIGITS(n)           TO_CHAR (argumenten*)
FLOAT(x)            TO_NUMBER, TO_BINARY_FLOAT (argumenten*)
DOUBLE(x)           TO_NUMBER, TO_BINARY_FLOAT (argumenten*)
REAL(x)             TO_NUMBER (argumenten*)
INTEGER(x)          TO_NUMBER (argumenten*)
SMALLINT(x)         TO_NUMBER (argumenten*)
VARCHAR(x)          TO_CHAR (argumenten*)
VARCHAR_FORMAT(x)   TO_CHAR (argumenten*)
(*) een veelheid aan conversieformaten is beschikbaar in Oracle.
```

DDL genereren zonder externe tools

Robert Vial (ICU IT Services)

De meeste bedrijven zullen een tool in huis hebben, waarmee ze DDL kunnen genereren. Dit kan dan een vendor tool zijn, maar ook een eigen geschreven REXX/CLIST. Bij een nieuwe release van DB2 zal het DDL generatie-tool aangepast moeten worden. Met DB2 V9 betekent dit een aanzienlijke inspanning.

Wat nu voor die bedrijven, die over geen enkel tool beschikken? Nou, die kunnen terugvallen op een IBM-programma, dat gratis te downloaden is voor de gewenste DB2 versie. Dit programma heet ***DB2PLI8*** en wordt door IBM gebruikt voor het genereren van rapporten, indien een performance probleem wordt aangemeld. Het programma zal dus altijd up-to-date moeten zijn en dient beschikbaar te zijn vanaf dag 1 van een nieuwe DB2 release.

In dit artikel worden achtereenvolgens de volgende onderwerpen uitgewerkt:

- wat is DB2PLI8
- welke functionaliteit biedt het programma
- hoe is er meer uit te halen.

DB2PLI8

DB2PLI8 is een gratis te downloaden programma van IBM en er is een versie voor V7, V8 en V9. Met het programma is het mogelijk om DDL en statistics te genereren. Voor de statistics worden update en insert-statements gegenereerd, waarmee de DB2 catalog geüpdated kan worden. Om DDL te genereren zijn er twee modes: ofwel door de naam van een tabel of view op te gegeven, ofwel door de namen van tabellen in een plan_table te laten opzoeken.

Het programma zal alle componenten genereren, die bij de opgegeven tabel of view horen. Dus database, tablespace, table, views, indexen en foreign keys. Comments worden niet gegenereerd.

Meer info is te vinden op
http://www.ibm.com/support/docview.wss?uid=swg21206998

Functionaliteit van het programma

DB2PLI8 draait onder IKJEFT01 en wordt als volgt aangeroepen:

```
//DB2PLI8 EXEC PGM=IKJEFT01,DYNAMNBR=20
//STEPLIB  DD DISP=SHR,DSN=<DB2.library>
//SYSTSPRT DD SYSOUT=*
//SYSTSIN  DD *
 DSN SYSTEM(DB2x)
 RUN PROGRAM(DB2PLI8) PLAN(DB2PLI8) PARMS('/...,..')
//SYSTSPRT DD SYSOUT=*
```

waarbij op de puntjes de naam van een tabel of view wordt gegeven, gevolgd door bijkomende opties:

Vb. Creator = SYSIBM, name table = PLAN_TABLE:

```
PARMS('/SYSIBM.PLAN_TABLE,DDL')
```

Als bijkomende parm kan gekozen worden voor de optie `DDL' of `DDLONLY'. Met DDL wordt naast het genereren van de ddl ook insert/update-statements gegenereerd voor het bijwerken van de DB2-catalog. Met de optie `DDLONLY' krijgt men alleen de ddl:

```
PARMS('/SYSIBM.PLAN_TABLE,DDLONLY')
```

Indien een view wordt opgegeven, dan worden ook alle objecten behorende bij de view gegenereerd (dus db, ts, tb, ix, fk).

En wordt in de view bijv. naar 4 tabellen gerefereerd, dan wordt alles meegenomen!!! Het generatieproces houdt rekening met dubbels. Dus stel alle 4 tabellen zitten in dezelfde database, dan wordt maar éénmaal een 'create database'-statement gegenereerd.

Het genereren van de volgorde van de foreign keys is helaas niet te beïnvloeden. Dit zal in de meeste gevallen handmatig ingrijpen vereisen.

Wat als de input een plan_table is?

DB2PLI8 is gemaakt om IBM te kunnen helpen bij het oplossen van performance-problemen. Een performance-probleem speelt zich af op programmaniveau en daarom is het - met een plan_table als input - mogelijk om voor meerdere objecten DDL en/of statistics te laten genereren, die door het betreffende programma gebruikt worden. Als input moet dan het volgende opgegeven worden: naast creator van de plan_table ook de range van *queryno*'s, waarmee aangegeven kan worden voor welk stuk in de plan_table DB2PLI8 aan het werk moet en voor welk programma (de kolom *progname* in de plan_table)

De parameter ziet er dan als volgt uit:

```
PARMS('/<creator>.PLAN_TABLE, <progname>, <qno-start>,
<qno-end>, DDLONLY')
```

Verder uitnutten van DB2PLI8

Omdat er als input een plan_table opgegeven kan worden, is het mogelijk om met een simpel truukje DDL te kunnen laten genereren van alles wat gewenst is, dus bijv. ook van alle definities in een DB2-subsysteem! En de voorbereiding om dit te bereiken is relatief klein:

Maak voor de DB2-versie waarmee gewerkt wordt een plan_table aan en neem daar de volgende wijzigingen in op (zie *Appendix D: Aangepaste PLAN_TABLE voor gebruik met DB2PLI8* op p. 27):

- alle kolommen krijgen een 'NOT NULL WITH DEFAULT' clause
- de kolom progname krijgt een default waarde, bijv. 'ABIS'
- kolom queryno wordt een integer met 'GENERATED ALWAYS AS IDENTITY'.

Door nu de plan_table te vullen met die objecten waarvan men DDL en/of statistics gegenereerd wil hebben, is het met onderstaande parameter altijd mogelijk om DB2PLI8 op de gehele plan_table te laten werken. Hoe groot deze dan ook mag zijn.....

```
PARMS('/<creator>.PLAN_TABLE, ABIS, 1, 9999999, DDLONLY')
```

De betreffende plan_table kan heel gemakkelijk gevuld worden met behulp van onderstaande insert:

```
INSERT INTO <creator>.PLAN_TABLE (CREATOR,TNAME)
 SELECT CREATOR,NAME
 FROM SYSIBM.SYSTABLES
 WHERE TYPE='T'
```

Een eenvoudige en snelle manier dus om DDL te kunnen genereren voor alle tabellen op het hele systeem!

Bijkomend voordeel is, dat het programma ook snel is.

Echter, met dit programma heb ik gewerkt in DB2 9 en daar zijn toch wel wat schoonheidsfoutjes uitgekomen:

- Volgorde van foreign keys is helaas willekeurig.
- owners / qualifiers worden niet altijd goed gegenereerd.
- in de values-clause worden geen quotes gegenereerd
- 'audit change' moet zijn 'audit changes'
- 'not padded' wordt bij alle indexen gegenereerd, ook indien ze geen VARCHAR-kolom bevatten; dit kan een warning, sqlcode +20002 opleveren
- piecesize heeft niet altijd de goede waarde.
- het genereren van meer dan 1 unique constraint gaat niet goed
- 'enforce' moet zijn 'enforced'.

Echter, deze kleine foutjes zijn makkelijk te corrigeren en dan blijkt DB2PLI8 toch wel ineens een heel krachtig programma, waar ongetwijfeld menigeen zijn voordeel mee kan doen.

OVER DE AUTEUR

Robert Vial zette ruim 20 jaar geleden z'n eerste stappen in de database-wereld als programmeur. Recent is hij, in z'n rol van DBA bij een grote luchtvaartorganisatie, vooral met Data Warehousing bezig. Hij is één van de infrastructuur-specialisten bij ICU IT Services.

Contactgegevens: http://www.linkedin.com/profile?key=4412474&viewProfile=

CURSUSPLANNING SEPT - DEC 2010

DB2 concepten	450 EUR	op aanvraag
DB2 for z/OS, een totaaloverzicht	2025 EUR	30/08(L), 27/09(W), 15/11(L)
DB2 UDB for LUW, totaaloverzicht	2025 EUR	11/10(L), 22/11(W)
RDBMS-concepten	375 EUR	30/08(L), 27/09(W), 11/10,(L), 15/11(L), 22/11(W)
Basiskennis SQL	375 EUR	31/08(L), 28/09(W), 12/10,(L), 16/11(L), 23/11(W)
DB2 for z/OS basiscursus	1275 EUR	01/09(L), 29/09(W), 17/11(L)
DB2 UDB for LUW basiscursus	1275 EUR	13/10(L), 24/11(W)
SQL-QMF voor eindgebruikers	1275 EUR	24/11(W)
SQL workshop	800 EUR	23/09(L), 25/10(W), 29/11(L), 13/12(W)
SQL voor gevorderden	450 EUR	22/10(L), 12/11(W)
DB2 SQL PL, triggers, stored procedures	450 EUR	31/08(L), 09/12(W)
DB2 for z/OS programmeren voor gevorderden	900 EUR	29/11(L)
DB2 for z/OS: SQL performance	1350 EUR	08/11(L), 06/12(W)
XML in DB2	450 EUR	30/11(W), 17/12(L)
DB2 for z/OS database administratie	1900 EUR	02/11(W), 20/12(L)
DB2 for z/OS operations & recovery	1425 EUR	06/10(UK), 01/12(UK)
DB2 for z/OS systems performance and tuning	1000 EUR	16/09(UK), 09.12(UK)
DB2 LUW DBA - Kernvaardigheden	1800 EUR	29.03(L), 22.06(W)
DB2 v8 upgrade, DB2 9 upgrade	450 EUR	op aanvraag
Data warehouse concepten	450 EUR	17/09(L), 19/11(W)
SQL voor BI	450 EUR	28/09(L)
Cognos cursussen		zie *http://www.abis.be/html/nlDWCalendar.html*

Plaats: L = Leuven, W = Woerden, UK = High Wycombe (bij Londen);
voor details en andere cursussen, zie http://www.abis.be/html/nlTraining.html

Postbus 220
Diestsevest 32
BE-3000 Leuven
Tel. 016/245610
Fax 016/245639
http://www.abis.be/
training@abis.be

abis
TRAINING & CONSULTING

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

Appendix A: output van de query op de IVP-database.

MGRNAME	DEPTNAME	LASTNAME	FIRSTNME
CHRISTINE HAAS	SPIFFY COMPUTER SERVICE DIV.	HAAS	CHRISTINE
CHRISTINE HAAS	SPIFFY COMPUTER SERVICE DIV.	LUCCHESI	VINCENZO
CHRISTINE HAAS	SPIFFY COMPUTER SERVICE DIV.	O'CONNELL	SEAN
EVA PULASKI	ADMINISTRATION SYSTEMS	PULASKI	EVA
EVA PULASKI	ADMINISTRATION SYSTEMS	JEFFERSON	JAMES
EVA PULASKI	ADMINISTRATION SYSTEMS	MARINO	SALVATORE
EVA PULASKI	ADMINISTRATION SYSTEMS	SMITH	DANIEL
EVA PULASKI	ADMINISTRATION SYSTEMS	JOHNSON	SYBIL
EVA PULASKI	ADMINISTRATION SYSTEMS	PEREZ	MARIA
IRVING STERN	MANUFACTURING SYSTEMS	STERN	IRVING
IRVING STERN	MANUFACTURING SYSTEMS	ADAMSON	BRUCE
IRVING STERN	MANUFACTURING SYSTEMS	PIANKA	ELIZABETH
IRVING STERN	MANUFACTURING SYSTEMS	YOSHIMURA	MASATOSHI
IRVING STERN	MANUFACTURING SYSTEMS	SCOUTTEN	MARILYN
IRVING STERN	MANUFACTURING SYSTEMS	WALKER	JAMES
IRVING STERN	MANUFACTURING SYSTEMS	BROWN	DAVID
IRVING STERN	MANUFACTURING SYSTEMS	JONES	WILLIAM
IRVING STERN	MANUFACTURING SYSTEMS	LUTZ	JENNIFER
MICHAEL THOMPSON	PLANNING	THOMPSON	MICHAEL
SALLY KWAN	INFORMATION CENTER	KWAN	SALLY
SALLY KWAN	INFORMATION CENTER	QUINTANA	DOLORES
SALLY KWAN	INFORMATION CENTER	NICHOLLS	HEATHER

Appendix B: Query on PLAN_TABLE with Page-Range info

```
SELECT   SUBSTR(STRIP(CHAR(PLAN.QUERYNO)), 1, 5) AS QUERY
       , CASE PLAN.PARENT_QBLOCKNO
              WHEN 0 THEN ' '
                     ELSE
                    SUBSTR(STRIP(CHAR(PLAN.PARENT_QBLOCKNO)), 1, 4)
         END AS PBLK
       , SUBSTR(STRIP(CHAR(PLAN.QBLOCKNO)), 1, 4) AS QBLK
       , SUBSTR(STRIP(CHAR(PLAN.PLANNO)), 1, 4) AS PLNO
       , CASE PLAN.MIXOPSEQ
              WHEN 0 THEN ' '
                     ELSE SUBSTR(STRIP(CHAR(PLAN.MIXOPSEQ)), 1, 4)
         END AS OPSQ
       , PLAN.QBLOCK_TYPE AS TYPE
       , CASE PLAN.JOIN_TYPE
              WHEN 'F' THEN 'Full'
              WHEN 'P' THEN 'Pair'
              WHEN 'S' THEN 'Star'
              WHEN 'L' THEN CASE PLAN.METHOD
                                 WHEN 1 THEN 'L/R'
                                 WHEN 2 THEN 'L/R'
                                 WHEN 4 THEN 'L/R'
                                        ELSE '?'
                                 END
              WHEN ' ' THEN CASE PLAN.METHOD
                                 WHEN 1 THEN 'Innr'
                                 WHEN 2 THEN 'Innr'
                                 WHEN 4 THEN 'Innr'
                                        ELSE ' '
                                 END
                        ELSE '*ERR*'
         END AS JOIN
       , CASE PLAN.METHOD
              WHEN 0 THEN 'First'
              WHEN 1 THEN 'NLjoin'
              WHEN 2 THEN 'MSjoin'
              WHEN 3 THEN 'Sort'
              WHEN 4 THEN 'Hybrid'
                     ELSE 'UNKNWN'
         END AS METHOD
       , CASE PLAN.TABLE_TYPE
              WHEN 'B' THEN 'Buffer'
              WHEN 'C' THEN 'CTE'
              WHEN 'F' THEN 'TabFun'
              WHEN 'M' THEN 'MQT'
              WHEN 'Q' THEN 'Temp'
              WHEN 'R' THEN 'Recurs'
              WHEN 'T' THEN 'Table'
              WHEN 'W' THEN 'Work'
                       ELSE ' '
         END AS TYPE
       , SUBSTR(PLAN.CREATOR, 1, 8) AS CREATOR
       , SUBSTR(PLAN.TNAME, 1, 18) AS TABLE
       , SUBSTR(PLAN.ACCESSNAME, 1, 8) AS NDXNAME
       , CASE PLAN.PRIMARY_ACCESSTYPE
              WHEN 'D' THEN 'D/'
              WHEN 'T' THEN 'T/'
                       ELSE ''
         END CONCAT
         STRIP(PLAN.ACCESSTYPE) CONCAT
         CASE WHEN PLAN.PAGE_RANGE = 'Y'
```

```
              THEN '(' CONCAT
                  STRIP(SUBSTR(STRIP(CHAR(RANGE.NUMPARTS)), 1, 4))
                   CONCAT ')'
              ELSE ''
         END AS ACCESS
       , CASE PLAN.PREFETCH
              WHEN 'S' THEN 'SEQ'
              WHEN 'L' THEN 'LST'
              WHEN 'D' THEN 'DYN'
                       ELSE ' '
         END AS PREF
       , CASE PLAN.INDEXONLY
              WHEN 'Y' THEN 'XO'
                       ELSE ' '
         END AS XO
       , CASE PLAN.MATCHCOLS
              WHEN 0 THEN ''
                   ELSE SUBSTR(STRIP(CHAR(PLAN.MATCHCOLS)), 1, 2)
         END AS MC
       , CASE PLAN.SORTC_GROUPBY
              WHEN 'Y' THEN 'G'
                       ELSE ' '
         END CONCAT
         CASE PLAN.SORTC_JOIN
              WHEN 'Y' THEN 'J'
                       ELSE ' '
         END CONCAT
         CASE PLAN.SORTC_ORDERBY
              WHEN 'Y' THEN 'O'
                       ELSE ' '
         END CONCAT
         CASE PLAN.SORTC_UNIQ
              WHEN 'Y' THEN 'U'
                       ELSE ' '
         END CONCAT
         CASE PLAN.SORTN_JOIN
              WHEN 'Y' THEN 'N'
                       ELSE ' '
         END AS GJOUN
       , CASE PLAN.PARALLELISM_MODE
              WHEN 'I' THEN 'I/O'
              WHEN 'C' THEN 'CPU'
              WHEN 'X' THEN 'SYS'
                       ELSE ' '
         END AS PAR
       , CASE PLAN.CTEREF
              WHEN 0 THEN ' '
              ELSE STRIP(CHAR(PLAN.CTEREF))
         END AS CTEREF
FROM     PLAN_TABLE PLAN
LEFT OUTER JOIN
         DSN_PGRANGE_TABLE RANGE
     ON  RANGE.QUERYNO  = PLAN.QUERYNO
     AND RANGE.QBLOCKNO = PLAN.QBLOCKNO
     AND RANGE.TABNO    = PLAN.TABNO
WHERE    PLAN.QUERYNO = 114
ORDER BY PLAN.QUERYNO
       , PLAN.PARENT_QBLOCKNO DESC
       , PLAN.QBLOCKNO
       , PLAN.PLANNO
       , PLAN.MIXOPSEQ
;
```

Appendix C: Query on PLAN_TABLE with Page-Range info and sorting info

```
WITH     QUERIES
        (QUERYNO)
AS      (SELECT   114
         FROM     SYSIBM.SYSDUMMY1
         )
       , SORTKEYS
AS      (SELECT DISTINCT
                  SKEY.QUERYNO
                , SKEY.QBLOCKNO
                , SKEY.PLANNO
                , SKEY.SORTNO
                , SKEY.TABNO
         FROM     DSN_SORTKEY_TABLE SKEY
         JOIN     QUERIES QRYS
              ON  QRYS.QUERYNO = SKEY.QUERYNO
         WHERE    SKEY.TABNO <> 0
         )
       , TABLES
AS      (SELECT DISTINCT
                  PLAN.QUERYNO
                , PLAN.TABNO
                , PLAN.TABLE_TYPE
                , PLAN.CREATOR
                , PLAN.TNAME
         FROM     PLAN_TABLE PLAN
         JOIN     QUERIES QRYS
              ON  QRYS.QUERYNO = PLAN.QUERYNO
         WHERE    PLAN.TABNO <> 0
         )
SELECT   SUBSTR(STRIP(CHAR(PLAN.QUERYNO)), 1, 5) AS QUERY
       , CASE PLAN.PARENT_QBLOCKNO
              WHEN 0 THEN ' '
                     ELSE
                    SUBSTR(STRIP(CHAR(PLAN.PARENT_QBLOCKNO)), 1, 4)
         END AS PBLK
       , SUBSTR(STRIP(CHAR(PLAN.QBLOCKNO)), 1, 4) AS QBLK
       , SUBSTR(STRIP(CHAR(PLAN.PLANNO)), 1, 4) AS PLNO
       , CASE PLAN.MIXOPSEQ
              WHEN 0 THEN ' '
                   ELSE SUBSTR(STRIP(CHAR(PLAN.MIXOPSEQ)), 1, 4)
         END AS OPSQ
       , PLAN.QBLOCK_TYPE AS TYPE
       , CASE PLAN.JOIN_TYPE
              WHEN 'F' THEN 'Full'
              WHEN 'P' THEN 'Pair'
              WHEN 'S' THEN 'Star'
              WHEN 'L' THEN CASE PLAN.METHOD
                                 WHEN 1 THEN 'L/R'
                                 WHEN 2 THEN 'L/R'
                                 WHEN 4 THEN 'L/R'
                                        ELSE '?'
                                 END
              WHEN ' ' THEN CASE PLAN.METHOD
                                 WHEN 1 THEN 'Innr'
                                 WHEN 2 THEN 'Innr'
                                 WHEN 4 THEN 'Innr'
                                        ELSE ' '
                                 END
                        ELSE '*ERR*'
         END AS JOIN
```

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

```
          END CONCAT
          CASE PLAN.SORTN_JOIN
               WHEN 'Y' THEN 'N'
                        ELSE ' '
          END AS GJOUN
 FROM     PLAN_TABLE PLAN
 JOIN     QUERIES QRYS
      ON  QRYS.QUERYNO = PLAN.QUERYNO
 LEFT OUTER JOIN
          DSN_PGRANGE_TABLE RANGE
      ON  RANGE.QUERYNO  = PLAN.QUERYNO
      AND RANGE.QBLOCKNO = PLAN.QBLOCKNO
      AND RANGE.TABNO    = PLAN.TABNO
 WHERE    PLAN.METHOD <> 3
UNION ALL
 SELECT   SUBSTR(STRIP(CHAR(PLAN.QUERYNO)), 1, 5) AS QUERY
        , CASE PLAN.PARENT_QBLOCKNO
               WHEN 0 THEN ' '
                      ELSE
                     SUBSTR(STRIP(CHAR(PLAN.PARENT_QBLOCKNO)), 1, 4)
          END AS PBLK
        , SUBSTR(STRIP(CHAR(PLAN.QBLOCKNO)), 1, 4) AS QBLK
        , SUBSTR(STRIP(CHAR(PLAN.PLANNO)), 1, 4) AS PLNO
        , CASE WHEN PLAN.MIXOPSEQ <> 0
               THEN SUBSTR(STRIP(CHAR(PLAN.MIXOPSEQ)), 1, 4)
                ELSE ''
          END CONCAT
          CASE WHEN SORT.SORTNO > 0
               THEN SUBSTR('ABCDEFGHIJKLMNOPQRSTUVWXYZ',
                           SORT.SORTNO, 1)
                ELSE ''
          END AS OPSQ
        , PLAN.QBLOCK_TYPE AS TYPE
        , '' AS JOIN
        , 'Sort' AS METHOD
        , CASE TBLS.TABLE_TYPE
               WHEN 'B' THEN 'Buffer'
               WHEN 'C' THEN 'CTE'
               WHEN 'F' THEN 'TabFun'
               WHEN 'M' THEN 'MQT'
               WHEN 'Q' THEN 'Temp'
               WHEN 'R' THEN 'Recurs'
               WHEN 'T' THEN 'Table'
               WHEN 'W' THEN 'Work'
                        ELSE ' '
          END AS TYPE
        , SUBSTR(TBLS.CREATOR, 1, 8) AS CREATOR
        , SUBSTR(TBLS.TNAME, 1, 18) AS TABLE
        , SUBSTR(PLAN.ACCESSNAME, 1, 8) AS NDXNAME
        , CASE PLAN.PRIMARY_ACCESSTYPE
               WHEN 'D' THEN 'D/'
               WHEN 'T' THEN 'T/'
                        ELSE ''
          END CONCAT
          STRIP(PLAN.ACCESSTYPE)
          AS ACCESS
        , CASE PLAN.PREFETCH
               WHEN 'S' THEN 'SEQ'
               WHEN 'L' THEN 'LST'
               WHEN 'D' THEN 'DYN'
                        ELSE ' '
          END AS PREF
        , CASE PLAN.INDEXONLY
```

```
             WHEN 'Y' THEN 'XO'
                          ELSE ' '
          END AS XO
        , CASE PLAN.MATCHCOLS
             WHEN 0 THEN ''
                    ELSE SUBSTR(STRIP(CHAR(PLAN.MATCHCOLS)), 1, 2)
          END AS MC
        , SUBSTR(SORT.SORTC, 1, 4) CONCAT
          SUBSTR(SORT.SORTN, 2, 1) AS GJOUN
FROM     DSN_SORT_TABLE SORT
JOIN     QUERIES QRYS
     ON  QRYS.QUERYNO = SORT.QUERYNO
LEFT OUTER JOIN -- Obtain nr of table being sorted
         SORTKEYS SKEY
     ON  SKEY.QUERYNO  = SORT.QUERYNO
     AND SKEY.QBLOCKNO = SORT.QBLOCKNO
     AND SKEY.PLANNO   = SORT.PLANNO
     AND SKEY.SORTNO   = SORT.SORTNO
LEFT OUTER JOIN -- Obtain name of table being sorted
         TABLES TBLS
     ON  TBLS.QUERYNO = SKEY.QUERYNO
     AND TBLS.TABNO   = SKEY.TABNO
LEFT OUTER JOIN -- Join back to relevant PLAN_TABLE row
         PLAN_TABLE PLAN
     ON  PLAN.QUERYNO  = SORT.QUERYNO
     AND PLAN.QBLOCKNO = SORT.QBLOCKNO
     AND PLAN.PLANNO   = SORT.PLANNO
     AND(   PLAN.METHOD   = 3
         OR(    PLAN.SORTC_JOIN = 'Y'
            AND SUBSTR(SORT.SORTC, 2, 1) = 'J'
            )
         OR(    PLAN.SORTN_JOIN = 'Y'
            AND SUBSTR(SORT.SORTN, 2, 1) = 'J'
         )  )
ORDER BY 1, 2 DESC, 3, 4, 5
;
```

Appendix D: Aangepaste PLAN_TABLE voor gebruik met DB2PLI8

```
CREATE TABLE PLAN_TABLE (
 QUERYNO                    INTEGER  GENERATED ALWAYS AS IDENTITY
,QBLOCKNO                   SMALLINT        NOT NULL WITH DEFAULT
,APPLNAME                   VARCHAR  ( 24)  NOT NULL WITH DEFAULT
,PROGNAME                   VARCHAR  (128)  NOT NULL WITH DEFAULT 'ABIS'
,PLANNO                     SMALLINT        NOT NULL WITH DEFAULT
,METHOD                     SMALLINT        NOT NULL WITH DEFAULT
,CREATOR                    VARCHAR  (128)  NOT NULL WITH DEFAULT
,TNAME                      VARCHAR  (128)  NOT NULL WITH DEFAULT
,TABNO                      SMALLINT        NOT NULL WITH DEFAULT
,ACCESSTYPE                 CHAR     (  2)  NOT NULL WITH DEFAULT
,MATCHCOLS                  SMALLINT        NOT NULL WITH DEFAULT
,ACCESSCREATOR              VARCHAR  (128)  NOT NULL WITH DEFAULT
,ACCESSNAME                 VARCHAR  (128)  NOT NULL WITH DEFAULT
,INDEXONLY                  CHAR     (  1)  NOT NULL WITH DEFAULT
,SORTN_UNIQ                 CHAR     (  1)  NOT NULL WITH DEFAULT
,SORTN_JOIN                 CHAR     (  1)  NOT NULL WITH DEFAULT
,SORTN_ORDERBY              CHAR     (  1)  NOT NULL WITH DEFAULT
,SORTN_GROUPBY              CHAR     (  1)  NOT NULL WITH DEFAULT
,SORTC_UNIQ                 CHAR     (  1)  NOT NULL WITH DEFAULT
,SORTC_JOIN                 CHAR     (  1)  NOT NULL WITH DEFAULT
,SORTC_ORDERBY              CHAR     (  1)  NOT NULL WITH DEFAULT
,SORTC_GROUPBY              CHAR     (  1)  NOT NULL WITH DEFAULT
,TSLOCKMODE                 CHAR     (  3)  NOT NULL WITH DEFAULT
,TIMESTAMP                  CHAR     ( 16)  NOT NULL WITH DEFAULT
,REMARKS                    VARCHAR  (762)  NOT NULL WITH DEFAULT
,PREFETCH                   CHAR     (  1)  NOT NULL WITH DEFAULT
,COLUMN_FN_EVAL             CHAR     (  1)  NOT NULL WITH DEFAULT
,MIXOPSEQ                   SMALLINT        NOT NULL WITH DEFAULT
,VERSION                    VARCHAR  (122)  NOT NULL WITH DEFAULT
,COLLID                     VARCHAR  (128)  NOT NULL WITH DEFAULT
,ACCESS_DEGREE              SMALLINT        NOT NULL WITH DEFAULT
,ACCESS_PGROUP_ID           SMALLINT        NOT NULL WITH DEFAULT
,JOIN_DEGREE                SMALLINT        NOT NULL WITH DEFAULT
,JOIN_PGROUP_ID             SMALLINT        NOT NULL WITH DEFAULT
,SORTC_PGROUP_ID            SMALLINT        NOT NULL WITH DEFAULT
,SORTN_PGROUP_ID            SMALLINT        NOT NULL WITH DEFAULT
,PARALLELISM_MODE           CHAR     (  1)  NOT NULL WITH DEFAULT
,MERGE_JOIN_COLS            SMALLINT        NOT NULL WITH DEFAULT
,CORRELATION_NAME           VARCHAR  (128)  NOT NULL WITH DEFAULT
,PAGE_RANGE                 CHAR     (  1)  NOT NULL WITH DEFAULT
,JOIN_TYPE                  CHAR     (  1)  NOT NULL WITH DEFAULT
,GROUP_MEMBER               VARCHAR  ( 24)  NOT NULL WITH DEFAULT
,IBM_SERVICE_DATA           VARCHAR  (254)  NOT NULL WITH DEFAULT  FOR BIT DATA
,WHEN_OPTIMIZE              CHAR     (  1)  NOT NULL WITH DEFAULT
,QBLOCK_TYPE                CHAR     (  6)  NOT NULL WITH DEFAULT
,BIND_TIME                  TIMESTAMP       NOT NULL WITH DEFAULT
,OPTHINT                    VARCHAR  (128)  NOT NULL WITH DEFAULT
,HINT_USED                  VARCHAR  (128)  NOT NULL WITH DEFAULT
,PRIMARY_ACCESSTYPE         CHAR     (  1)  NOT NULL WITH DEFAULT
,PARENT_QBLOCKNO            SMALLINT        NOT NULL WITH DEFAULT
,TABLE_TYPE                 CHAR     (  1)  NOT NULL WITH DEFAULT
,TABLE_ENCODE               CHAR     (  1)  NOT NULL WITH DEFAULT
,TABLE_SCCSID               SMALLINT        NOT NULL WITH DEFAULT
,TABLE_MCCSID               SMALLINT        NOT NULL WITH DEFAULT
,TABLE_DCCSID               SMALLINT        NOT NULL WITH DEFAULT
,ROUTINE_ID                 INTEGER         NOT NULL WITH DEFAULT
,CTEREF                     SMALLINT        NOT NULL WITH DEFAULT
,STMTTOKEN                  VARCHAR  (240)  NOT NULL WITH DEFAULT
,PARENT_PLANNO              SMALLINT        NOT NULL WITH DEFAULT  );
```