COBOL: preferred supplier voor DB2?

GSE NL COBOL werkgroep

Nieuwegein - 14 juni 2012

Peter Vanroose - ABIS Training & Consulting

COBOL: preferred supplier voor DB2?

Welkom

ABIS Training & Consulting

http://www.abis.be/

Zaagmolenlaan 4, Woerden
Diestsevest 32, Leuven (BE)

Peter Vanroose

- **docent & consultant**
- **DBA voor DB2 z/OS**

Agenda - motivatie

Ingrediënten:

- **COBOL**
- **DB2**
- **preferred supplier**
- **?**

COBOL als (een) applicatieve component van DB2-database-toegang

Zijn er dan alternatieven?

- **PL/1, C, REXX, Assembler**
- **Java, C#, C++, Perl, PHP, ...**
- ***SQL PL***

Uitwisselbaar? Voor- en nadelen? Caveats?

Inhoud - overzicht

1. Intro: applicaties en databases
2. Case study (als rode draad)
3. Stored procedures
4. Waarom COBOL / DB2 / SQL PL ?
   - COBOL & DB2: een ideaal span?
   - COBOL & MVS: een ideaal span?
   - DB2 & SQL PL: een ideaal span?
   - SQL PL & DB2: een ideaal span?
5. Caveats & conclusies

Inhoud

Applicatieve component van (DB2) database-toegang

Taken:

- **Lees-acties:**
  - **logica voor het combineren van tabellen en het selecteren (“filteren”) van gegevens**
    - SQL “SELECT”
    - cursor(s)
    - soms variabelen nodig (b.v. rijen nummeren)
  - **formattering (presentatie)**
- **Schrijf-acties:**

  INSERT / UPDATE / DELETE
  - **converteren van (externe) gegevens naar tabel-formaat (DB2-datatypes: CHAR(n) VARCHAR(n) INT DATE ...)**
  - **fout-afhandeling (SQL-codes)**
  - **transactioneel werken (gegroepeerde “commit”)**

Waarom data op DB2 voor z/OS?

Centrale data-opslag

- want replicatie is moeilijk

Bottleneck

- massale gelijktijdige toegang naar zelfde data

z/OS kan goed om met “workload management”

- z-architectuur
- LPARs
- WLM
- minimale down-time

Relationeel datamodel: universeel gebruikt

DB2:

- zeer schaalbaar (zowel data-volume als concurrency)
- zeer performant
- minimale down-time (voor maintenance)

Waarom decentrale applicaties?

Ontlasten van de centrale server

Verschillende noden & mogelijkheden

- grafische interfaces
- batch-verwerking
- web-applicaties
- koppelen aan e-mail, telefoon, stockbeheer, ...

3rd party software

...

COBOL-applicaties & DB2: situatieschets

COBOL - “batch”

COBOL - online (b.v. CICS)

hoofdprogramma / subprogramma’s

herbruikbaarheid van subprogramma’s

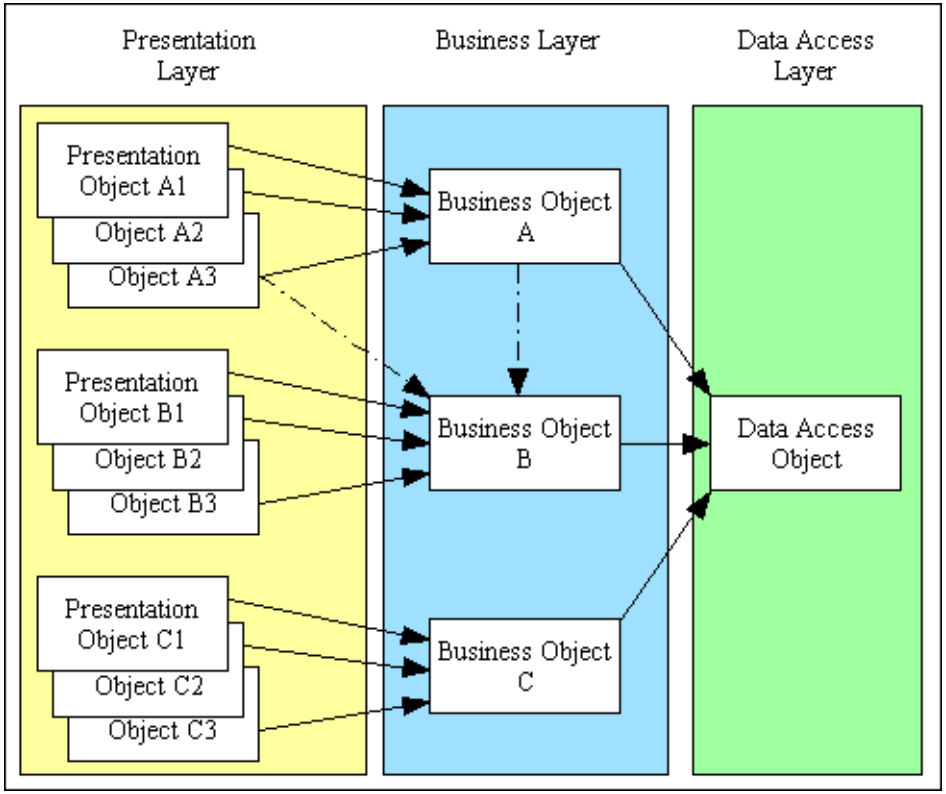
- **vermijden van code duplication**
- **eenvoudig testen van deelfunctionaliteit (*unit tests*)**
- **ook voor “remote” (distributed) applicaties?**

overzichtelijker programma-structuur: scheiden van

- **“business logica” (i.h.b. database-manipulatie)**
- **presentatie-logica (b.v. GUI)**

==> klassiek 3-laags model (*3-tier model*)

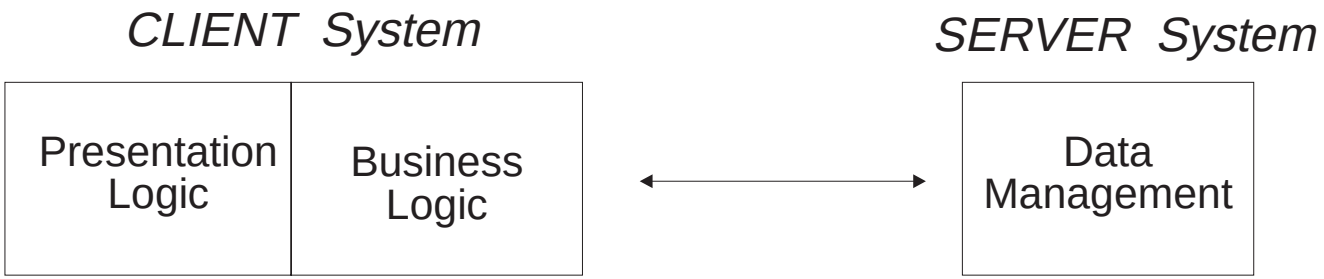
3-tier model

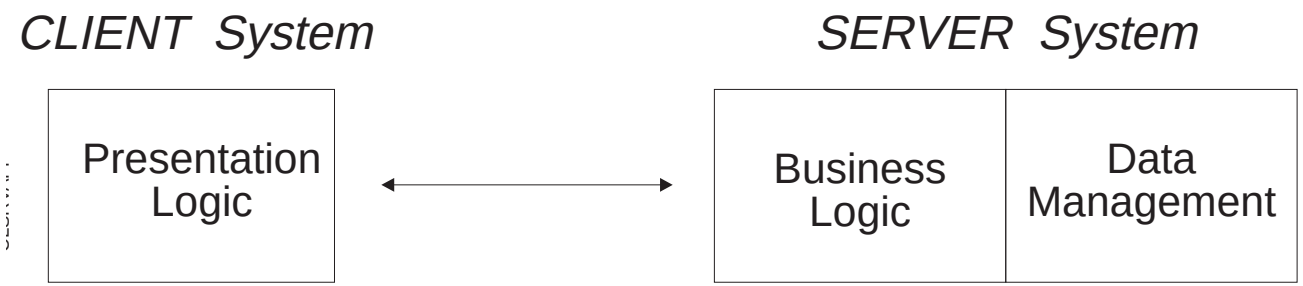


(model ook bruikbaar in een niet-OO omgeving!)

COBOL-hoofdprogr. **COBOL-subprogr.** **DB2-access-path**

3-tier model & client-server model





(In de praktijk dikwijls een mix van beide)

Tweede model: COBOL als “preferred supplier” !

Praktisch probleem: Java / C# main <--------> COBOL subprogr. ??

COBOL: preferred supplier voor DB2?

Inhoud

1. Intro: applicaties en databases

2. Case study (als rode draad)

3. Stored procedures

4. Waarom COBOL / DB2 / SQL PL ?
   - COBOL & DB2: een ideaal span?
   - COBOL & MVS: een ideaal span?
   - DB2 & SQL PL: een ideaal span?
   - SQL PL & DB2: een ideaal span?

5. Caveats & conclusies

Case Study - business case

ABIS Client & Course Administration (ACCA)

- Beheer van klantgegevens
- Beheer van cursussen, sessies, inschrijvingen
- in COBOL, GUI-appl. (ISPF Dialog Manager)
- data-gebaseerde structuur

  Persons - **CO**mpanies - **SE**ssions - **E**nrolments - **C**ourse**F**ees - ...

  zie lezing “SQL PL als programmeertaal. Onze ervaringen”,
  GSE Nationale Conferentie, 13 oktober 2011, Almere,
  http://www.abis.be/resources/presentations/gsenl20111013sqlpl.pdf

Submodule “inschrijven van een cursist in op een cursus”:

- Oorspronkelijk enkel mogelijk via ISPF-appl.
- Moet kunnen herbruikt worden in verschillende contexten
  - web-interface (PHP)
  - intranet (Java)
  - batch-verwerking (b.v. prijsaanpassing)
  - facturatie
  - ...

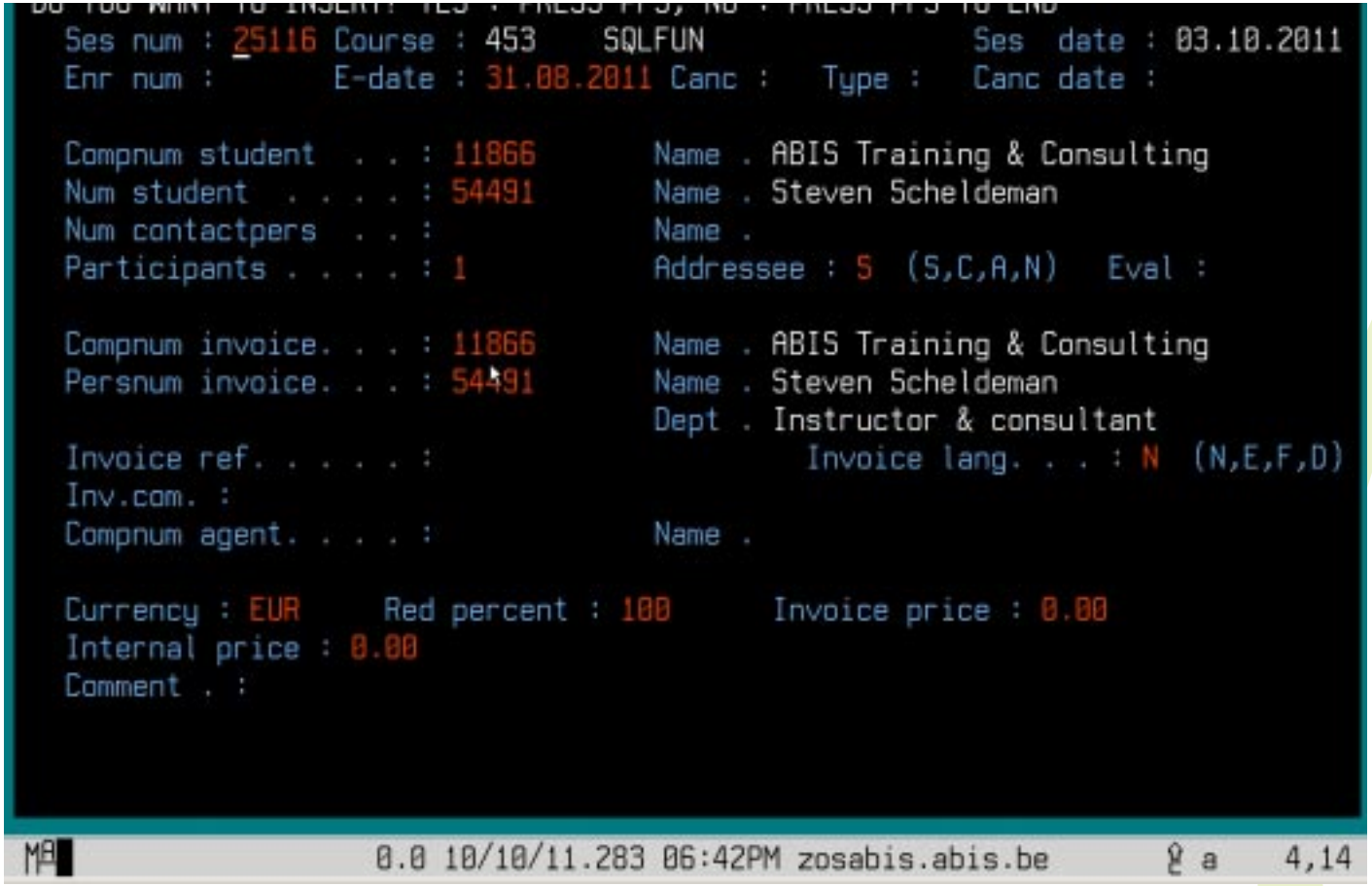
Case Study - business case

ABIS Client & Course Administration (ACCA)

Submodule “inschrijven van een cursist in op een cursus”:

- **Op te geven:**
  - identificatie van de cursist (persoonsnummer: pno)
  - identificatie van z’n bedrijf (facturatie-adres: cono)
  - identificatie van de cursus-sessie (datum/locatie: seno)
- **Validatie door de module:**
  - reeds ingeschreven; volzet; ongeldige gegevens ...
- **Business-logica door de module:**
  - basisprijs cursus aflezen uit DB2-tabellen
  - prijs omrekenen naar Euro
  - korting berekenen op basis van bedrijfsgegevens, voucher, ...
  - inschrijving opslaan in DB2-tabel
  - bevestigingsmail sturen aan inschrijver / cursist / docent / ...
- **Resultaat retour aan oproeper (b.v. om te tonen in GUI):**
  - gegenereerd inschrijvingsnummer (eno) als identificator
  - afgeleide gegevens (naam; cursusdatum; ...) voor visuele verificatie

Case Study - de GUI



Rode gegevens: invulbaar;

Witte gegevens: door de module teruggegeven

Onderste lijnen: door de module teruggegeven, maar wijzigbaar

COBOL: preferred supplier voor DB2?

Inhoud

1. Intro: applicaties en databases

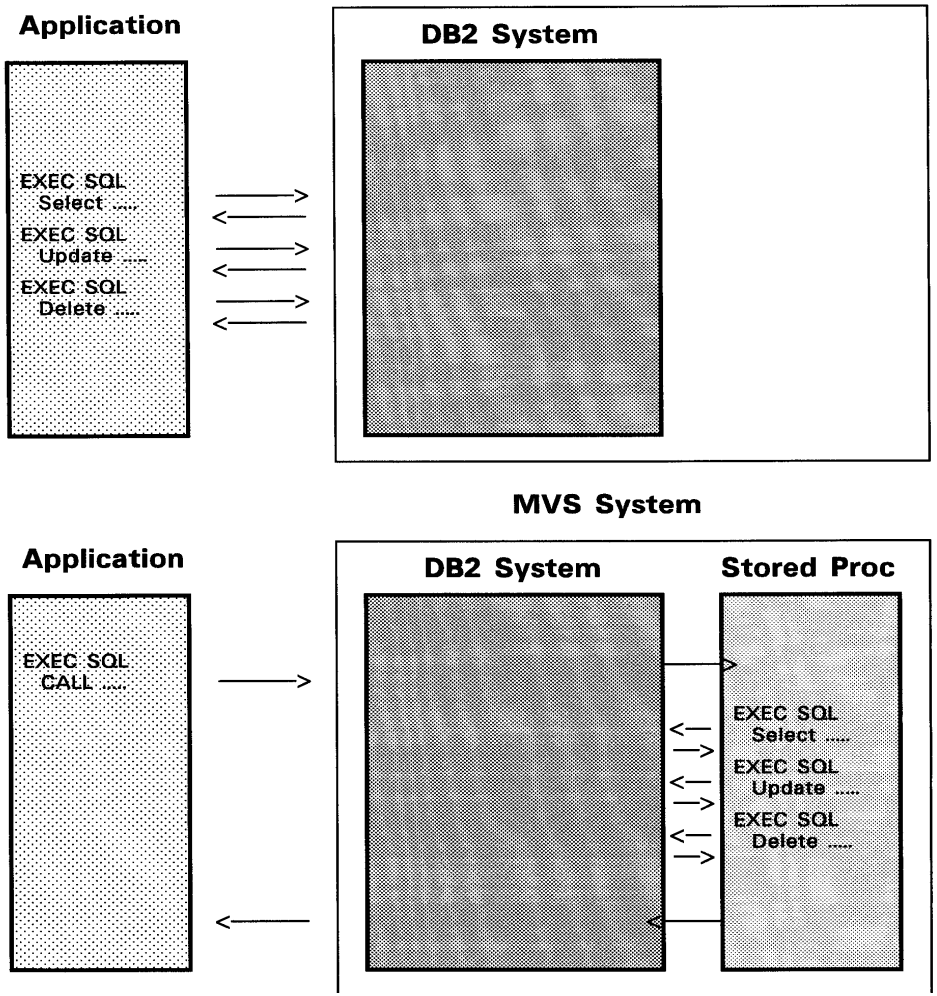
2. Case study (als rode draad)

3. Stored procedures

4. Waarom COBOL / DB2 / SQL PL ?
   - COBOL & DB2: een ideaal span?
   - COBOL & MVS: een ideaal span?
   - DB2 & SQL PL: een ideaal span?
   - SQL PL & DB2: een ideaal span?

5. Caveats & conclusies

Stored procedures



“business logica” op de data server ==> centraal, herbruikbaar

Stored procedures

COBOL: preferred supplier voor DB2?

Wat?

- soort subprogramma:

  CALL proc(arg1, arg2, ...)
- onder beheer van DB2
- toegankelijk via DB2
  - vanuit een COBOL-programma
  - vanuit een distributed programma
  - vanuit een REXX script

Hoe?

- Declaratie (in DB2): naam & datatypes van de argumenten
- Implementatie:
  - external: COBOL, PL/1, C, REXX, Assembler, ...
  - native : SQL PL
- Runtime-omgeving: WLM

DB2-security van toepassing bij oproep

Stored procedures: voor- en nadelen

PRO ++

- modulair, herbruikbaar.
- black box: één object, vastgelegde functionaliteit & interface
  alleen openen om implementatie aan te passen
  kwaliteitscontrole (testen); tuning; taakverdeling impl.
- auditing: afscherming van toegang tot gevoelige data
  ==> applicatie heeft geen autorisatie op tabel-niveau nodig
- volumineus data-trafiek blijft binnen DB2
  ==> minder communicatie tussen applicatie en DB2
- universele toegang mogelijk (Java, PHP, REXX, ...)

CONTRA --

- extra layer ==> overhead ==> minder performant?
- central point of failure (nl. DB2)
- complexer om non-DB2 data te benaderen (o.a. 2-phase commit)

Stored procedure: een uitgewerkt voorbeeld

gewenste functionaliteit van de Stored Procedure:

- **schrijf een cursist in op een cursus:**
  - pno van de cursist
  - cono van z'n bedrijf
  - seno van de sessie
- **validatie:**
  - reeds ingeschreven; volzet; ongeldige gegevens ...
- **business-logica**
  - basisprijs afleiden uit DB2-tabellen
  - evtl. omrekenen naar Euro
  - korting(en) berekenen
  - inschrijving opslaan in DB2-tabel
- **resultaat retour aan oproeper:**
  - inschrijvingsnummer (eno)
  - afgeleide gegevens (naam; cursusdatum; ...)
  - foutenboodschap / foutencode (logisch of technisch)

Stored procedure: een uitgewerkt voorbeeld (II)

Kies een NAAM voor de nieuwe St.Proc

- InsEnrol

Definieer de INTERFACE voor de oproeper (CALL) van de St.Proc

- **INPUT voor de St.Proc:**
  - p-seno sessie-nummer waarop ingescheven wordt
  - p-studpno persoons-nummer van de in te schrijven student
  - p-invcono bedrijfs-nummer van het te factureren bedrijf
- **OUTPUT uit de St.Proc:**
  - p-eno enrolment-nummer (uniek voor deze seno)
  - p-studname de naam van student p-studpno
  - p-compname de naam van bedrijf p-invcono
  - p-msg tekstuele vorm van de (fouten)boodschap
  - p-sqlcode numerieke vorm (DB2) van de evtl. fout
- **OUTPUT die t.z.t. ook (optionele) INPUT kan zijn:**
  - p-redpercent reductie-percentage; indien NULL: zie tabel
  - p-invprice berekende prijs (indien NULL on input)

Stored procedure: de COBOL-implementatie

De declaratie (SQL DDL-statement):

```
CREATE PROCEDURE InsEnrol
( IN        p_seno          INT
,   OUT p_eno           SMALLINT
, IN        p_studpno       INT
,   OUT p_studname      VARCHAR(80)
, IN        p_invcono       INT
,   OUT p_compname  VARCHAR(80)
, INOUT p_redpercent  INT
, INOUT p_invprice      DECIMAL(10,2)
,   OUT p_msg           VARCHAR(80)
,   OUT p_sqlcode       INT
)
    dynamic result sets 0
    external name INSENROL      -- naam v/h gecompileerde COBOL-progr.
    language COBOL
    parameter style   GENERAL WITH NULLS
    modifies SQL data
    commit on return NO
    [ WLM environment <naam> ]          ;
```

De implementatie: (deel 1)

```
DATA DIVISION.
WORKING-STORAGE SECTION.
77  w-crrate                PIC S9(4)V9(6) COMP-3.
LINKAGE SECTION.
01 params.
    02  p-seno              PIC S9(9) COMP.
    02  p-eno               PIC S9(4) COMP.
    02  p-studpno           PIC S9(9) COMP.
    02  p-stuname.
        49 p-stuname-len     PIC S9(4) COMP.
        49 p-stuname-txt     PIC X(80).
    02  p-invcono           PIC S9(9) COMP.
    02  p-coname.
        49 p-coname-len      PIC S9(4) COMP.
        49 p-coname-txt      PIC X(80).
    02  p-redpercent        PIC S9(9) COMP.
    02  p-invprice          PIC S9(8)V99 COMP-3.
    02  p-msg.
        49 p-msg-len         PIC S9(4) COMP.
        49 p-msg-txt         PIC X(80).
    02  p-sqlcode           PIC S9(9) COMP.
    02  indicators          PIC S9(4) COMP   OCCURS 10 TIMES.
```

De implementatie: (deel 2)

```
 PROCEDURE DIVISION USING params.
*    Validatie van de invoer:
     EXEC SQL    SELECT 0 INTO :p-sqlcode
                       FROM sessions  WHERE seno = :p-seno            END-EXEC
     IF SQLCODE NOT = ZERO THEN
        MOVE  SQLCODE TO p-sqlcode
        MOVE  'Session does not exist'  TO  p-msg-txt     MOVE 22 TO p-msg-len
        GOBACK
     END-IF
*  [ analoog voor p-studpno & p-invcono; & ophalen p-studname & p-compname ]
*    Opzoeken van reductie-percentage en berekenen van de prijs:
     PERFORM compute-price-with-reduction
*    Resultaat wegschrijven naar DB2 (tabel "enrolments"):
     EXEC SQL   SELECT coalesce(MAX(eno), 0) + 1  INTO :p-eno
                      FROM enrolments   WHERE e_seno= :p-seno        END-EXEC
     EXEC SQL  INSERT INTO enrolments
                     VALUES (:p-seno, :p-eno, :p-studpno, ...)       END-EXEC
     IF SQLCODE NOT = ZERO THEN  [ foutboodschap ]  END-IF
     MOVE  SQLCODE  TO  p-sqlcode
     MOVE  'Enrolment inserted'  TO  p-msg-txt       MOVE 18 TO p-msg-len
     GOBACK
     .
```

De implementatie: (business-logica voor p-redpercent & p-invprice)

```
 compute-price-with-reduction.
     IF indicators(8) < 0 THEN
*     D.w.z.: prijs was niet opgegeven door de oproeper
        IF indicators(7) < 9 THEN
*        D.w.z.: ook reductie-percentage was niet opgegeven door de oproeper
           EXEC SQL   SELECT repercent    INTO :p-redpercent
                                FROM reductions  WHERE re_cono = :p-invcono
           END-EXEC
           IF SQLCODE NOT = ZERO THEN   MOVE 0 TO p-redpercent   END-IF
        END-IF
        EXEC SQL  SELECT cfprice, crrate  INTO :p-invprice, :w-crrate
                            FROM coursefees JOIN currency ON cf_crcode = crcode
                           WHERE cf_seno = :p-seno
        END-EXEC
*    Korting toepassen:
        COMPUTE p-invprice ROUNDED = p-invprice * (1.00 - (p-redpercent * 0.01))
*    Conversie naar EURO (indien prijs b.v. in £; meestal is w-crrate = 1.00) :
        DIVIDE  w-crrate INTO p-invprice
     END-IF
     .
```

Stored procedure: de oproep

De oproep (vanuit een ander COBOL-programma: GUI of batch of ...):

```
        MOVE 22643 TO seno
*       (etc.)
        MOVE -1 TO ind1 ind2
        EXEC SQL
            CALL InsEnrol
            ( :seno, :eno, :pno, :naam:naam-ind, :cono, :coname:ind
            , :redpercent:ind1, :invprice:ind2, :msg, :code)
        END-EXEC
        DISPLAY naam ' enrolled for session 22643; message: ' msg
```

- **Oproep van een St.Proc dus zeer gelijkaardig aan oproep van een COBOL-subprogramma**
  - conversie van subprogramma naar St.Proc is “eenvoudig”
- **Parameters:**
  - als “host-variabelen” ==> moeten DB2-datatypes zijn
  - NULL-indicatoren waar nodig!
    (kan vermeden worden indien gewenst ==> minder flexibel)

Stored procedure: voordelen van COBOL-implementatie

als “back-end” in 3-tier model:

- *porten* van een bestaand COBOL-subprogramma:
  - bijna geen “rewrite” nodig
  - declareer parameters als “GENERAL” (without NULLs)
  - gebruik enkel X(n) & S9(9) COMP in linkage section
- nieuwe / herwerkte software:
  - gebruik evtl. “GENERAL WITH NULLS”
  - VARCHAR, DATE waar mogelijk / gewenst
  - efficiënte communicatie met DB2 (data-transfer) (zie verderop)
  - efficiënte toegang naar MVS (incl. XML-data; MQ; USS(?); ...)

Voordeel van *St.Proc* voor impl. subprogramma:

- datatype-validatie is automatisch (door DB2)
- flexibel autorisatie-beheer van oproeper (door DB2)
- ook oproepbaar van buiten z/OS (distributed apps)

Stored procedure: de SQL PL-implementatie

Declaratie en implementatie in één keer (SQL DDL)

- declaratie: gelijkaardig aan die voor de COBOL-implementatie:

```
CREATE PROCEDURE InsEnrol
( IN    p_seno          INT
,   OUT p_eno           SMALLINT
, IN    p_studpno       INT
,   OUT p_studname      VARCHAR(80)
, IN    p_invcono       INT
,   OUT p_compname      VARCHAR(80)
, INOUT p_redpercent    INT
, INOUT p_invprice      DECIMAL(10,2)
,   OUT p_msg           VARCHAR(80)
,   OUT p_sqlcode       INT
)
    dynamic result sets 0
    language SQL
    modifies SQL data
    [ disable debug mode ]
    [ WLM environment <naam> ]      -- naar keuze het één of het ander
```

- **dadelijk gevolgd door het implementatie-deel (dat bij deze dus “gecompileerd” wordt):**

```
BEGIN
    DECLARE SQLCODE INT;
    DECLARE crrate DECIMAL(10,6);
    SELECT 0 INTO p_sqlcode FROM sessions WHERE seno = p_seno;
    IF SQLCODE <> 0 THEN
        [ exception handling & return ]
    END IF;
    [ analoog voor p-studpno & p-invcono; & ophalen p-studname & p-compname ]
    IF p_invprice IS NULL THEN
    /* D.w.z.: prijs was niet opgegeven door de oproeper */
        [ business-logica om p_redpercent op te zoeken, en p_invprice te berekenen ]
    END IF;
    SELECT coalesce(MAX(eno), 0) + 1  INTO p_eno
       FROM enrolments   WHERE e_seno= p_seno;
    INSERT INTO enrolments VALUES (p_seno, p_eno, p_studpno, ....);
    /* Geen COMMIT hier: dat is een beslissing van de oproeper */
    IF SQLCODE <> 0 THEN  [ foutboodschap ]  END IF;
    SET p_sqlcode = SQLCODE;
    SET p_msg = 'Enrolment inserted';
END
```

- **business-logica voor p_redpercent en p_invprice:**

```
IF p_redpercent IS NULL THEN
    SELECT repercent    INTO p_redpercent
      FROM  reductions  WHERE re_cono = p_invcono ;
    IF SQLCODE <> 0 THEN   SET p_redpercent = 0   END IF;
END IF;
SELECT cfprice, crrate  INTO p_invprice, crrate
  FROM coursefees INNER JOIN currency ON cf_crcode = crcode
WHERE cf_seno = p_seno ;
/* Korting: */
SET p_invprice = p_invprice * ( 1.00 - p_redpercent * 0.01 );
/* Conversie naar EURO (indien prijs b.v. in Britse ponden): */
SET p_invprice = p_invprice / crrate;
```

- **jammer genoeg niet als “paragraaf” & perform**
  ==> moet in-line ingevoegd worden in hoofdprogramma
- **syntax niet zo verschillend van COBOL**
- **geen EXEC SQL**
  ==> SQL <---> programma-logica “loopt meer dooreen”
- **free-format; instream comments; end-of-statement is “;”**
- **logische condities: SQL-syntax (*a IS NULL; a IN (..,..,..); a <> b; ...*)**

Inhoud

1. Intro: applicaties en databases

2. Case study (als rode draad)

3. Stored procedures

4. Waarom COBOL / DB2 / SQL PL ?
   - COBOL & DB2: een ideaal span?
   - COBOL & MVS: een ideaal span?
   - DB2 & SQL PL: een ideaal span?
   - SQL PL & DB2: een ideaal span?

5. Caveats & conclusies

COBOL & DB2: een ideaal span!

Data-uitwisseling is performant:

- STATIC SQL (<---> b.v. Java of C#) (maar = PL/1, C, C++, SQLJ)
- gelijkaardige(r) datarepresentatie

COBOL	DB2	Java
PIC X(n)	CHAR(n)	-
PIC S9(9) COMP	INTEGER	int
PIC S9(4) COMP & PIC X(n)	VARCHAR(n)	string
PIC 9(n)P(m)	-	-
PIC S9(n)V9(m) COMP-3	DECIMAL(n+m,m)	-
-	FLOAT / DOUBLE / DECFLOAT	float / double / -
-	XML / CLOB	-

Geen dataconversie nodig voor gros v/d DB2-data

==> efficiënte inter-process I/O

COBOL-programma's zelf ook performant(er)

- geen call stack opbouw; geen dynamische allocaties; ...

  (argumentatie gelijkaardig aan die voor Assembler)

St.Proc: waarom COBOL?

- **Code beschikbaar**
- **Vertrouwde omgeving**
  - **interactie met DB2 goed gedocumenteerd**

Aandachtspunten:

- **COBOL compiler met precompile (DB2)**
  - **één source-bestand**
  - **twee “binaries”**
    - COBOL compiler ==> load module
    - DB2 optimizer ==> package
- **Foutafhandeling**
  - **validatie van input (“contract” met oproeper)**
  - **DB2-message (SQLCODE) ==> hoe naar oproeper communiceren (of “binnenkamers” houden)**
  - **business-logica ==> eigen foutencode/return code genereren**
  - **“CALL St.Proc” zelf kan fout gaan ==> SQLCODE gegenereerd!**
- **NULLs zijn “lastig”**
- **St.Proc moet “generieker”, robuuster zijn dan COBOL-subprogr.**

COBOL & z/OS (MVS)

Ideaal in combinatie met b.v. CICS

==> precompiler, gelijkaardig aan die voor DB2

datasets met fixed-length records:

==> efficiënte I/O van/naar COBOL-programma

(WLM)

(MQ)

Verdere argumentatie: buiten de scope van deze lezing.

DB2 en SQL PL: een ideaal span!

DB2 vóór versie 9:

- SQL PL niet volwaardig beschikbaar
- C-compiler nodig

Zeer volledige ondersteuning van de standaard

- incl. exception handling (zie verder)

Integratie van SQLCODE & SQLSTATE

- maar let op met SQLCODE = 100 (zie verder)

Nog relatief weinig gebruikt (“onbekend is onbemind”)

Compiler-boodschappen & debugging: kan beter!

Keuze tussen WLM (“fenced”) of binnen DB2-engine (DBM1)

- transparante keuze (gemaakt door DBA)
- workload kan goed in de hand gehouden worden (vooral belangrijk voor remote oproepen)

SQL Procedural Language (SQL PL) -- een overzicht

Standaard SQL (ANSI & ISO) beschrijft wat de syntax is van SQL PL in document

SQL/PSM ("Persistent Stored Modules"):

- declaraties van variabelen
- condities (IF ... ELSE ...)
- iteraties (WHILE ...)
- samengestelde statements (blocks): BEGIN ... END
- cursoren declareren, openen, en lezen (FETCH)
- exception handling opzetten

Uiteraard wordt er gezwegen over een variabele genaamd SQLCODE ...

SQL Control Statements

- COMMENTS:
  - between: /* and */
  - can be more than one line
- Statement separator:
  - All SQL statements inside a compound statement block must be ended by a semicolon ";"
- Compound statement block:

```
BEGIN
    <local variable declarations>
    <local cursor declarations>
    <local handler declarations>
    <SQL statements>
END
```

- SQL variable declaration:

```
DECLARE <var-name> [, <var-name>... ]  <data-type>
                [ DEFAULT <value> ]
```

- **Two special “DB2 return code” variables**

```
DECLARE  SQLSTATE CHAR(5) [ DEFAULT <value> ]
DECLARE  SQLCODE INTEGER [ DEFAULT <value> ]
```

  - These variables are automatically assigned to by DB2
  - but must be declared when used

- **SQL cursor declaration:**

```
DECLARE  <name>  CURSOR FOR  <SELECT-statement>
```

- **SQL handler declaration**

```
DECLARE  { CONTINUE | EXIT | UNDO }  HANDLER FOR
    { SQLSTATE <sqlstate-value>
        | <condition-name>
        | SQLEXCEPTION
        | SQLWARNING
        | NOT FOUND       }
    <SQL PL statement>
```

- **ASSIGNMENT-statement:**

```
SET <var-name> = <scalar-expression>
```

- **IF statement:**

```
IF <condition> THEN
    <SQL PL statement(s)>
ELSEIF <condition> THEN
    <SQL PL statement(s)>
ELSE
    <SQL PL statement(s)>
END IF
```

- **CASE statement:**

```
CASE
  WHEN <condition>  THEN <SQL PL statement(s)>
[ WHEN <condition> THEN <SQL PL statement(s)>
   ...                                                    ]
[ ELSE                          <SQL PL statement(s)> ]
END CASE
```

- **WHILE...DO statement:**

```
WHILE <condition> DO
      <SQL PL statement(s)>;
END WHILE
```

- **More iterator statements: REPEAT, ITERATE, LOOP, LEAVE, FOR.**

St.Proc: waarom SQL PL?

- naadloze integratie met “embedded SQL”
- zelfde datatypes; NULLs op natuurlijke manier afgehandeld
- gebruik van “IF naam LIKE 'Jans%en%'”, “IF date BETWEEN ...”, ...
- Geen aparte address space nodig (maar wel mogelijk)

Aandachtspunten:

- Sourcecode-beheer is “anders”
  - actuele source wordt in de DB2-datalog bijgehouden
  - maar lastig om op te halen / aan te passen
  - versioning ...
- Foutafhandeling
  - business-logica ==> eigen foutencode/return code genereren kan m.b.v. SQL-statement “SIGNAL SQLSTATE” (applicatieve foutcode b.v. ‘75002’ + tekst foutenboodschap)
- NULLs
  - binnenkomende NULLs interpreteren in St.Proc
  - buitengaande NULLs (spaarzaam!) ==> interpretatie: oproeper

SQL PL: voor DB2 en ...

Potentieel voordeel van SQL PL:

standaard SQL (“PSM”), dus *cross-vendor*; in elk geval DB2 LUW

Praktijk:

- **nogal wat verschillen met**
  - **PL/SQL (Oracle)**
  - **Transact-SQL (SQL Server, Microsoft)**
- **zeer gelijkaardig aan**
  - **PostgreSQL (PL/pgPSM)**
  - **MySQL (sinds v5.5)**

  Maar: geen rechtstreekse toegang tot “SQLCODE” of “SQLSTATE”
- **DB2-implementatie is momenteel de meest volledige**
  - **wegbereider!**
  - **goede voorbereiding op de toekomst (evtl. zonder DB2?)**
- **Beperkingen van DB2:**
  - **SQL PL enkel voor St.Proc, Triggers, User-Defined Functions**

SQL PL: voor DB2 en ...

Stored Procedure body DB2-onafhankelijk(er) maken:
gebruik van standaard “exception handling”:

```
BEGIN
    DECLARE crrate DECIMAL(16,6);
    DECLARE EXIT HANDLER FOR SQLEXCEPTION   SET p_sqlcode = SQLCODE;
    DECLARE EXIT HANDLER FOR SQLWARNING     SET p_sqlcode = SQLCODE;
    DECLARE EXIT HANDLER FOR NOT FOUND      SET p_sqlcode = 100;
    SET p_msg = 'No such session';
    SELECT 0 INTO p_sqlcode FROM sessions WHERE seno = p_seno;
    IF p_redpercent IS NULL THEN  BEGIN
        DECLARE eof INT DEFAULT 0;
        DECLARE CONTINUE HANDLER FOR NOT FOUND SET eof = 1;
        SELECT repercent  INTO p_redpercent  FROM  reductions  WHERE ... ;
        IF eof = 1  THEN   SET p_redpercent = 0   END IF;
    END;  END IF;
    SET p_msg = 'Insert failed';
    INSERT INTO enrolments VALUES (p_seno, p_eno, p_studpno, ....);
    SET p_msg = 'Enrolment inserted';
END
```

Inhoud

1. Intro: applicaties en databases

2. Case study (als rode draad)

3. Stored procedures

4. Waarom COBOL / DB2 / SQL PL ?
   - COBOL & DB2: een ideaal span?
   - COBOL & MVS: een ideaal span?
   - DB2 & SQL PL: een ideaal span?
   - SQL PL & DB2: een ideaal span?

5. Caveats & conclusies

Caveats

Gebruik van SQLCODE

- Wordt *enkel* gewijzigd door DB2 bij DML, DCL, DDL
  (SELECT, UPDATE, INSERT, DELETE, FETCH, OPEN, CLOSE, ...)
  Niet bij b.v. SET x = y
- SQLCODE = 100 : einde van cursor-loop
  of geen gevonden rijen in WHERE-conditie
  Kan éénmalig getest worden (WHILE SQLCODE = 0 ...),
  daarna is waarde blijkbaar terug 0 ?!? Werkt dus niet:

```
FETCH c INTO v;
WHILE SQLCODE = 0 DO
    <process v>;
    FETCH c INTO v;
END WHILE;
/* op dit punt is SQLCODE blijkbaar terug gelijk aan 0 ... */
IF SQLCODE <> 100 THEN
    <error message: onverwachte SQL-code>   /* verrassend: SQLCODE = 0 */
END IF;
```

Caveats

Datatype DECIMAL(n,m) en precisie van tussenresultaten

```
DECLARE  invprice     DECIMAL(10,2) DEFAULT  675.00;
DECLARE  redpercent INT                 DEFAULT 13;
DECLARE crrate         DECIMAL(10,6) DEFAULT 1.000000;
SET invprice = invprice * (1.00 - redpercent * 0.01);   /* waarde is nu   00000587.25 */
SET invprice = invprice / crrate;                                  /* blijft 587.25 ... of toch niet? */
```

Geobserveerde waarde is echter invprice = 587.20

Waarom niet 587.25? Waar is de laatste decimaal naartoe?

Verklaring (zie DB2 SQL Reference Guide, “decimal division”):

- **Tussenresultaat DECIMAL(w,x) / DECIMAL(y,z) is DECIMAL(15,s)**
  waarbij s = 15 - (w - x + z) = 15 - (10 - 2 + 6) = 15 - 14 = 1
  Dus: 587.25 / 1.000000 = 00 000 000 000 587.2
  (vermijdt overflow: 87654321.99 / 0.000001 = 87 654 321 990 000)
- ***Oplossing:* declareer crrate b.v. als DECIMAL(*16*,6)**
  ==> tussenresultaat is dan DEC(**31**,s) i.p.v. DEC(**15**,s)
  waarbij nu s = y - w + x - z = *16* - 10 + 2 - 6 = 2

COBOL: preferred supplier voor DB2?

Besluit

Applicatie- en data-design

- **centrale database ==> DB2 op z/OS**
- **decentrale applicaties (3-tier model)**
- **back-end: als stored procedure**
  - **gelijkvormige interface**
  - **universeel toegankelijk**
- **St.Proc: in COBOL of in SQL PL ?**
  - **pro's / contra's**
    geval per geval beoordelen
  - **"mix" is perfect mogelijk**
  - **argumentatie:**
    - performance
    - onderhoudbaarheid
    - portability
    - integratie met andere (toekomstige) systemen

Vragen ?

Q&A

Peter Vanroose

Docent & Consultant

pvanroose@abis.be

bedankt voor uw aandacht!