UNIVERSITEIT ANTWERPEN
FACULTEIT WETENSCHAPPEN
DEPARTEMENT WISKUNDE-INFORMATICA

DECEMBER 2002

ELEKTRONISCHE INVOER VAN GEGEVENS IN DE SCOB-DATABANK
EEN ONTWERP

NELE DEXTERS EN FREDERIC HANCKE

Contents

1 Inleiding

We beschrijven de werking van het package **scob.EI**. Dit is geschreven voor wat wij **Electronic Import** noemen, oftewel **de elektronische transfer van op cd-rom staande data in de SCOB-databank** [Dex02].

In het verleden is de SCOB-databank manueel opgevuld met gegevens die overgenomen werden uit de koerslijsten. Er kon op dat moment immers nog geen sprake zijn van een vorm van invoer op elektronische wijze, vermits deze manier van gegevensopslag toen nog niet bestond. We spreken dan vooral over de 19de en het grootste deel van de 20ste eeuw. De laatste jaren van vorige eeuw worden echter gekenmerkt door een toenemende automatisatie en sinds eind jaren '80 zijn er dan ook reeds een aantal koersen in digitale vorm beschikbaar. Een eerste deeltje hiervan wordt aangepakt in deze paper, namelijk gegevens die in bestanden op een voor SCOB beschikbaar opslagmedium staan. In de toekomst is de werkwijze die er hier is toegepast zeker uit te breiden naar een meer veralgemeende vorm van elektronische transfer van data, bijvoorbeeld via het internet. Op die manier kan de SCOB-databank up-to-date blijven door een rechtstreekse invoer van koersgegevens (die onmiddellijk afkomstig zijn van de bron, de Financieel Economische Tijd), zonder vertraging. Om dit mogelijk te maken wordt de databank 24 uur op 24, 7 dagen op 7 via het internet in verbinding gebracht met de bestanden van de FET. Als veiligheidsvoorziening zal de databank echter achter een firewall geplaatst worden. Er zullen wekelijks backups genomen worden van de ingevoerde data en er zal ook op regelmatige basis worden nagekeken of er bij de overdracht geen problemen optreden.

Dit technisch rapport is geschreven door Nele Dexters en Frederic Hancke. Nele Dexters is de vaste informaticamedewerkster van SCOB [sco] die het project gecoördineerd heeft en zal zorgen voor de uiteindelijke uitvoering van de elektronische transfer. De concrete implementatie komt van Frederic Hancke, die een vrijwillig medewerker is van SCOB. Beiden doctoreren sinds 2001 aan de Universiteit Antwerpen.[1]

SCOB, het **S**tudie**C**entrum voor **O**nderneming en **B**eurs, is het resultaat van een grootscheeps onderzoeksproject (GOA) "Catalogisering en digitalisering van het archief van de Beurs van Brussel (1998-2002)", met als bedoeling het archief van de Beurs van Brussel toegankelijk te maken door de bedrijfshistorische informatie te inventariseren en de koersinformatie te digitaliseren. Het project is opgezet onder leiding van een interdisciplinair team van de toenmalige drie Antwerpse universitaire instellingen, RUCA, UFSIA en UIA (faculteiten "Toegepaste Economische Wetenschappen", "Geschiedenis" en "Wiskunde-Informatica") en heeft een eigen website http://www.scob.be, waar er meer informatie te vinden is.

[1] Nele Dexters aan UIA, Frederic Hancke aan RUCA

2 Bestandsformaat

De originele bestanden met de koersen staan op een cd-rom. Deze komen in feite van een 50-tal diskettes die samengebracht zijn op één enkel schijfje. Voor de elektronische invoer worden al deze bestanden naar de harde schijf van één van de computers van SCOB gekopieerd. Hiervoor is een aparte partitie voorzien. Het is ook hier dat de logbestanden (zie **3.2 Log**) weggeschreven zullen worden. Naar schatting zal er na de transfer (die ongeveer anderhalve dag zal duren) zo'n 475 MB ruimte gebruikt worden door de logbestanden, die dan ook onmiddellijk op cd gebrand zullen worden. Het schrijven van deze logbestanden vertraagt het proces aanzienlijk, maar ze zijn een onschatbare bron aan informatie voor de medewerkers van SCOB.

Elk bestand met koersen is een gewoon **tekstbestand**. Er zijn elektronische data ter beschikking van **1989** t.e.m. **1998**. Elk bestand draagt een gelijkaardige naam die telkens begint met de letters **hcrs**, gevolgd door het desbetreffende **jaar**, een **punt** en een volgnummer bestaande uit **drie cijfers**. Deze beginnen te tellen vanaf 1 en de overige plaatsen aan de linkerkant worden telkens opgevuld met nullen. Als voorbeeld halen we hier **hcrs1989.001** aan, het eerste bestand dat er ingelezen zal moeten worden. Bij de overgang naar een nieuw jaar, wordt er terug gestart bij 001.

Elk in te lezen koersbestand bevat **platte tekst**. Dit vergemakkelijkt het automatisch opslaan van de koersbestanden, maar bemoeilijkt het op zicht interpreteren van de gegevens aangezien er nergens wordt aangegeven waar een stuk informatie over een zeker aandeel start en stopt. Voor het automatisch inlezen is dit echter zeer gemakkelijk omdat de nodige informatie steeds op dezelfde (relatieve) positie terug te vinden is (zie verder). Na grondige studie van de structuur van de koersbestanden is gebleken dat er in deze bestanden **twee soorten blokken** bestaan. **De eerste soort** bevat informatie die telkens start met de letter **F** of **S**, en bestaat uit **168** opeenvolgende karakters. Deze blokken worden nergens gescheiden, dus bij het inlezen van de gegevens zal er telkens 168 plaatsen verdergelezen moeten worden om alle bij elkaar horende gegevens te bereiken. Eventuele spaties en returns worden genegeerd omdat zij immers geen extra informatie opleveren. **De andere soort** blok begint telkens met één van de letters **C**, **B**, **R** of **G**, en bestaat uit **84** opeenvolgende karakters. In de bestanden die blokken bevatten van de eerste soort, staat er telkens één blok per lijn. In de bestanden die blokken van de tweede soort bevatten, kunnen we op één lijn telkens twee blokken terugvinden.

We beschrijven nu in detail hoe de blokken van elke soort zijn opgebouwd. "WNVB" staat voor "wordt niet verder beschouwd" en duidt op gegevens die niet relevant zijn voor de SCOB-databank en bijgevolg ook niet gebruikt zullen worden.

- Blokken startende met **F** of **S**, bestaande uit **168** karakters

Positie	Inhoud
0	F of S
1 - 10	svm-code
11 - 15	jaartal
16 - 17	maand
18 - 19	dag
20 - 31	openingskoers (grondtal)
32 - 34	openingskoers (decimalen)
35	gereserveerd (WNVB)
36 - 47	slotkoers (grondtal)
48 - 50	slotkoers (decimalen)
51	gereserveerd (WNVB)
52 - 63	totale volume (grondtal)
64 - 66	totale volume (decimalen)
67 - 78	maximum koers (grondtal)
79 - 81	maximumkoers (decimalen)
82 - 93	minimumkoers (grondtal)
94 - 96	minimumkoers (decimalen)
97 - 108	volume bij opening (grondtal) (WNVB)
109 - 111	volume bij opening (decimalen) (WNVB)
112 - 123	volume bij afsluiting (grondtal) (WNVB)
124 - 126	volume bij afsluiting (decimalen) (WNVB)
127 - 167	gereserveerd (WNVB)

- Blokken startende met **C**, **B**, **R** of **G**, bestaande uit **84** karakters

Positie	Inhoud
0	C, B, R of G
1 - 10	svm-code
11 - 15	jaartal
16 - 17	maand
18 - 19	dag
20 - 31	openingskoers (grondtal)
32 - 34	openingskoers (decimalen)
35 - 36	code van de openingskoers
37 - 48	slotkoers (grondtal)
49 - 51	slotkoers (decimalen)
52 - 53	code van de slotkoers
54 - 65	totale volume verhandelde aandelen per dag (WNVB)
66 - 68	totale volume verhandelde aandelen per dag (WNVB)
69 - 80	totaal volume bij sluiting (WNVB)
81 - 83	totaal volume bij sluiting (WNVB)

Uit deze verschillende velden (telkens aangegeven door de begin- en eindpositie van de te gebruiken karakters) binnen een blok dat alle gegevens voor één aandeel per datum groepeert, halen we de individuele waarden die er in de databank op de juiste plaatsen ingevoerd moeten worden. De svm-code[2] wordt gebruikt ter identificatie van het desbetreffende aandeel in de SCOB-databank. We geven nu in het kort de betekenis van de afkortingen F, S, C, B, R en G die er in bovenstaande opsomming gebruikt zijn. Deze lettercodes dienen om de bijbehorende aandelen in de juiste sector te kunnen plaatsen.

Letter	Sector
F	Forward, Marche à Terme
S	States Bonds (Emprunts du Secteur Public)
C	Actions, Droits et Warrants Belges et Etrangères
B	Obligations Belges en Etrangères du Secteur Privé
R	Obligations Non Regularisées par le Fonds des Rentes
G	Obligations Cotées au Fixing

Wat betreft de inhoud van de codes voor de openings- en slotkoersen, kunnen we de volgende 2-lettercombinaties tegenkomen. Voor de SCOB-databank worden er hier slechts enkelen van gebruikt. De overigen worden niet verder beschouwd (WNVB).

Letters	Betekenis
..	2 spaties, niks
AR	Cours Acheteurs Réduit (WNVB)
VR	Cours Vendeur Réduit (WNVB)
AA	Attestation Acheteur (WNVB)
AC	Attestation Acheteur au Cours Côté (WNVB)
AV	Attestation Vendeur (WNVB)
VC	Attestation Vendeur au Cours Côté (WNVB)
CA	Cours Modifié Argent
CP	Cours Modifié Papier
CI	Cours Indicatif (WNVB)
NC	Non Côté (WNVB)

[2] **S**tock **V**aleurs **M**obilières, in het Nederlands **srw**-code, **S**tock **R**oerende **W**aarde, één van de verschillende codes die er in de koerslijsten gebruikt worden ter identificatie van de verschillende aandelen

3 Package scob.EI

Het package **scob.EI** is geschreven in Java[3] [jav] en bestaat uit vier modules: **Brussels89**, **Log**, **ImportBrussels89** en **Main**, die elk afzonderlijk in detail worden besproken. Ook de wisselwerking tussen de verschillende modules zal in detail bekeken worden.

Het package zal in een DOS-venster command-line gestart worden. Dit venster zal tijdens het transferproces ook gebruikt worden om aan de gebruiker duidelijk te maken dat het proces nog bezig is, en niet is vastgelopen. Zo kan men zien wanneer er logbestanden *gecreëerd* worden, en welk bestand er *behandeld* wordt. Er wordt geschat dat het volledige transferproces ongeveer 37 uur (of anderhalve dag) bezig zal zijn.

3.1 Brussels89

In `Brussels89` wordt er **één koersbestand met blokken** ingelezen. Elk blok groepeert alle bij elkaar horende informatie zoals aandeelcode, datum, de verschillende koersen of hoeveelheden, ... Per blok wordt al deze informatie in een lijst (klasse `Vector` in Java) gezet, om er nadien alles gemakkelijk uit te kunnen halen voor het transferproces. `Brussels89` voorziet een functie (`getNextData`) waarmee de lijst velden van het volgende blok uit het koersbestand gehaald kunnen worden. Het zorgt hierbij dus voor het inlezen van 84 of 168 karakters in het huidige bestand en zet dit om in een lijst.

Er gaat hierna nooit meer gebruik gemaakt worden van de platte tekst, doch enkel nog van de gevonden informatieblokken. Bovendien wordt er nuttige informatie, zoals of het te verwerken koersbestand bestaat of niet, en of er onbekende symbolen zijn tegengekomen (bijvoorbeeld een A i.p.v. F, S, C, B, R of G als eerste karakter), in het zoeken naar velden binnen een blok weggeschreven naar het logbestand dat overeenkomt met het huidige koersbestand.

3.2 Log

Deze klasse is geschreven voor **het maken van logbestanden**, met als hoofdbedoeling zo veel mogelijk nuttige informatie tijdens de transfer bij te houden. Op deze manier beschikken de medewerkers van SCOB over een schat aan informatie over de door de transfer al dan niet effectief in de databank ingevoerde koersen. Aan de hand van het logbestand kan er ook op de correcte plaats ingegrepen worden indien er iets fout zou zijn gelopen bij de transfer. Bovendien wordt er ook alle extra informatie over niet in de databank bestaande aandelen (niet gekende aandeelcodes) bijgehouden. Indien er hierover meer informatie gewenst is, hoeft men dus enkel het desbetreffende logbestand te raadplegen. Als laatste voordeel van het hebben

[3] De gebruikte implementatie is J2SE 1.4.1, ontwikkeld door Sun Microsystems, Inc.

van logbestanden halen we hier aan dat een logbestand een zeer nuttig werkinstrument is voor het manueel controleren van een steekproef van ingevoerde data. Het logbestand is immers een rechtstreekse weerspiegeling van de handelingen die er plaatsvonden tijdens de transfer en de gegevens die er getransfereerd werden, op een overzichtelijke en gebruiksvriendelijke manier weergegeven.

Per ingelezen koersbestand wordt er een afzonderlijk logbestand gecreëerd. Dit krijgt als naam $<$ `bestandsnaam koersbestand` $>$ $-$ `i.log`, waarbij de `i` telkens verhoogd wordt. Er wordt van `1` gestart. Deze `i` dient om te voorkomen dat bij het testen en meermaals uitvoeren van hetzelfde proces eenzelfde logbestand niet overschreven wordt. Bij de eerste uitvoer van het proces is deze `i` gewoon `1`. Als het hele proces nu nog een tweede keer wordt uitgevoerd **en** de logbestanden van de eerste keer staan er nog, dan wordt `i` $=$ `2`. De `i` dient dus als volgnummer van de bestaande logbestanden om overschrijving te voorkomen. Door de speciale naamgeving kan het corresponderende logbestand van een koersbestand eenvoudig teruggevonden worden. In `Main` (zie **3.4**) kan er meegegeven worden waar deze logbestanden weggeschreven moet worden.

De transfer wordt in een DOS-venster command-line gestart en in dit venster zullen regelmatig boodschappen verschijnen die aangeven dat de transfer wel degelijk bezig is en niet is vastgelopen. In dit venster wordt er ook getoond wanneer een bepaald logbestand gecreëerd is.

3.3 ImportBrussels89

`ImportBrussels89` dient voor **het invoeren van de data** (gebruik makende van het handelbare vectorformaat, bekomen door `Brussels89`) **in de juiste tabellen in de databank**. Hierin gebeurt bijgevolg **het eigenlijke proces van het schrijven van de data in de databank**.

We beschrijven nu de werking van `ImportBrussels89`. Bij aanvang van de module wordt er een logbestand `start` gecreëerd waarin de exacte starttijd van het globale proces wordt bijgehouden, zodat er na afloop precies gevonden kan worden hoe lang dit geduurd heeft. Daarna wordt de connectie gemaakt met de SCOB-databank en het logbestand `start` wordt afgesloten aangezien de start van het proces hiermee is afgelopen.

Nu worden de in te lezen bestanden één voor één overlopen. Dit kan op een redelijk eenvoudige manier gebeuren door de uniformiteit in de naamgeving van de koersbestanden en het feit dat het aantal in te lezen bestanden op voorhand gekend is. Per bestand wordt er een afzonderlijk logbestand gemaakt waarin de start van het afzonderlijke proces voor dat koersbestand wordt bijgehouden. Per koersbestand wordt er dan `Brussels89` aangeroepen en vervolgens worden hieruit

blokken data opgeroepen (m.b.v. `getNextData`, waarmee de reeds besproken lijst velden van het volgende blok wordt bekomen, zie **3.1**) tot er geen blokken meer beschikbaar zijn in het behandelde koersbestand. Dan wordt `Brussels89` met het volgende koersbestand aangeroepen, enz., tot alle koersbestanden behandeld zijn en alle blokken ter beschikking zijn. Elke keer als er een nieuw blok gevonden wordt, wordt er per blok `handleData` opgeroepen, wat er op neer komt dat de gewenste gegevens uit de blokken worden gehaald en eventueel in de databank worden weggeschreven.

Telkens na de volledige verwerking van één enkel koersbestand (inlezen, blokken opsporen, gegevens al dan niet in de databank invoeren, . . .) wordt het aantal gevonden en verwerkte blokken in het overeenkomende logbestand weggeschreven. Gedurende de verwerking wordt er eveneens allerhande informatie over de behandelde blokken bijgehouden, zoals niet in de databank teruggevonden svm-codes en de effectief in de databank weggeschreven gegevens. Begin- en eindtijd van het verwerken van dit koersbestand worden eveneens in het logbestand bijgehouden om tijdens en na het verwerkingsproces een idee te hebben van de vorderingen en een juist beeld te krijgen van de tijdsduur van de verwerking van één enkel koersbestand. Na de volledige afwerking van één koersbestand wordt het corresponderende logbestand afgesloten.

Als alle koersbestanden doorlopen en verwerkt zijn, wordt er een overkoepelend logbestand `end` gecreëerd waarin wordt meegedeeld hoeveel koersbestanden er in totaal gevonden en behandeld zijn. De connectie met de databank wordt verbroken en het tijdstip waarop het proces eindigde, wordt opgeslagen. Hierna wordt `end` ook afgesloten, omdat het globale proces van het afsluiten beëindigd is. Indien er onderweg fouten zouden zijn opgetreden, dan staat dit natuurlijk ook in het desbetreffende logbestand!

In `ImportBrussels89` is `handleData`, zoals reeds eerder vermeld, de hoofdprocedure, die de informatie van een bepaald blok in lijstvorm meekrijgt (zie **3.1**). Er wordt hier gekeken naar de eerste letter van het blok en alzo wordt er de juiste functie voor verdere behandeling aangeroepen. Als de eerste letter **S**, **F**, **C** of **B** is, wordt er verder gegaan, indien de eerste letter **R** of **G** is, wordt er gestopt omdat deze twee gevallen niet behandeld worden door SCOB. Indien er toevallig nog een andere eerste letter zou optreden (wat in theorie niet mogelijk is), is er een veiligheid ingebouwd die de mededeling geeft dat de gevonden gegevens niet geldig zijn. De verwerking van het huidige blok stopt, en het proces gaat door met het volgende blok.

Vooraleer er effectief data in de databank ingevoerd kunnen worden, moet er geweten zijn over welk aandeel het gaat. Hiertoe wordt uit elk blok *de svm-code* van het aandeel gehaald en *herschreven* naar het formaat waarin deze code terug te vinden is in de SCOB-databank. We werken enkel met stocks waarvan de

code reeds in de databank aanwezig is, dus aandelen die door SCOB gekend zijn. Moesten er bij de elektronische data ongekende stocks zitten, dan worden deze in het logbestand genoteerd, zodat ze later op eenvoudige wijze terug opgespoord kunnen worden en indien nodig alsnog in de databank ingevoerd kunnen worden. Omdat er bij aanvang van het schrijven van dit package niet zo duidelijk was wat er precies moest gebeuren met deze ongekende aandelen, is er hiervoor ook code geschreven. Deze komt neer op het aanmaken van een volledig nieuw aandeel in de databank met alle bijbehorendheden. Aangezien dit nu echter niet meer van toepassing is, staat deze code in het uiteindelijke resultaat weggecommentarieerd. We hebben er bewust voor gekozen ze niet te verwijderen, omdat ze in de toekomst misschien nog van pas kan komen, bijvoorbeeld als er beslist wordt toch de niet gekende aandelen in te voeren.

We beschikken nu over de *svm-code* van het desbetreffende aandeel *zoals* deze terug te vinden is in de databank. Hiermee wordt er op zoek gegaan naar *de corresponderende interne stockcode*. SCOB heeft dus voor elke officiële stockcode (svm) zijn eigen interne code, waarmee er verderop telkens gewerkt zal worden. Deze zal in het vervolg van de tekst gewoon stockcode genoemd worden.

Met deze gevonden stockcode gaat er nu op zoek gegaan worden naar *de corresponderende notationcode* (aangezien dit verder ingaat op het design en de werking van de SCOB-databank, verwijzen we naar [Dex02]). Deze notationcode bestaat altijd. Het enige waar er hier op gelet moet worden, is dat er per stock meerdere notaties kunnen zijn, afhankelijk van de tijd. Natuurlijk moet hieruit de juiste gekozen worden, op basis van de ingelezen datum. Als er geen tijdsspanne in de databank gevonden kan worden die overeenkomt of overeen kan komen met de ingelezen datum, wordt er een nieuwe notatie aangemaakt.

Nu gaat er gekeken worden of er voor de ingelezen stock met de corresponderende gevonden notatie op de desbetreffende datum reeds *koersen* in de databank zijn ingegeven. Zo ja, dan betekent dit dat het desbetreffende aandeel reeds manueel behandeld is, en dat de gegevens niet nog eens ingevoerd moeten worden. Indien er echter nog geen koersen voor handen zijn voor de gegeven combinatie van stock, notatie en datum, dan wordt er een nieuwe notatie ingevoerd met als inhoud de ingelezen gegevens. Dit is natuurlijk de **kern** van de elektronische transfer: het in de databank invoeren van nieuwe gegevens over reeds gekende aandelen. Tegelijkertijd gaat er gekeken worden of er voor de ingelezen stock en de corresponderende notatie en bijbehorende datum reeds (verhandelde) *hoeveelheden* zijn ingevoerd. Als dit het geval is, dan mag er gewoon verder gegaan worden, want dan zitten de gegevens al in de databank. Is dit echter nog niet zo, dan wordt er een nieuwe notatie bijgemaakt waarin de nieuwe gegevens verwerkt worden. Ook dit maakt deel uit van de **kern** van de data transfer: het opvullen van de databank met nieuwe gegevens.

3.4 Main

`Main` is **de hoofdklasse van het project**. Hierin wordt er meegegeven waar alle koersbestanden terug te vinden zijn en waar de logbestanden weggeschreven moeten worden. Bovendien wordt hierin het hele proces gestart door `ImportBrussels89` aan te roepen.

Vóór de transfer worden alle koersbestanden van de cd-rom overgebracht naar een map `koersen` op een partitie op de harde schijf van één van de computers van SCOB. Op deze partitie wordt ook een afzonderlijke map `log` voorzien waarin de logbestanden weggeschreven worden. Na afloop van het proces worden deze bestanden onmiddellijk op cd gebrand.

4 Code

4.1 Code van Brussels89

```
package scob.EI;

import java.io.File;
import java.io.FileReader;
import java.util.Vector;

public class Brussels89 {
  private static final String CLASSNAME = "Brussels89";

  private static final int BLOCKWIDTH_FS = 168;
  private static final int BLOCKWIDTH_CBRG = 84;
  private static final int TYPE_FS = 0;
  private static final int TYPE_CBRG = 1;

  private Log log;
  private int type;
  private File file;
  private FileReader filereader;
  private boolean exists = false;

  public Brussels89(Log log, String filename) {
    try {
      this.log = log;
      file = new File(filename);
      if (file.exists()) {
        printMsg(filename + " found.");
        filereader = new FileReader(file);
        exists = true;
      } else {
        printMsg(filename + " not found.");
        exists = false;
      }
    } catch(Exception e) {
      e.printStackTrace();
    }
  }
```

```
public Vector getNextData() {
  try {
    String block = readBlock();
    if (block == null) {
      return null;
    } else {
      return parseBlock(block);
    }
  } catch(Exception e) {
    e.printStackTrace();
  }
  return null;
}

private String readBlock() throws Exception {
  String block = "";
  int blockwidth = 0;
  int ch;
  while (true) {
    ch = filereader.read();
    if (ch == -1) {
      return null;
    } else if ((ch != 10) && (ch != 13)
           && (ch != 32) && (ch != 9)) {
      switch((char)ch) {
        case 'F':
        case 'S':
          blockwidth = BLOCKWIDTH_FS;
          type = TYPE_FS;
          break;
        case 'C':
        case 'B':
        case 'R':
        case 'G':
          blockwidth = BLOCKWIDTH_CBRG;
          type = TYPE_CBRG;
          break;
        default:
          printMsg("Type " +
              (new Character((char)ch)).toString() +
              " unknown.");
          return null;
      }
```

```
      block += (char)ch;
      break;
    }
  }
  for (int i = 0; i < blockwidth - 1; i++) {
    ch = filereader.read();
    if (ch == -1) {
      return null;
    } else {
      block += (char)ch;
    }
  }
  return block;
}

private Vector parseBlock(String block) throws Exception {
  switch(type) {
    case TYPE_FS:
      return parseBlockFS(block);
    case TYPE_CBRG:
      return parseBlockCBRG(block);
    default:
      printMsg("Type " + Integer.toString(type)
              + " unknown.");
      return null;
  }
}

private Vector parseBlockFS(String block) throws Exception {
  Vector data = new Vector();

  data.add(block.substring(0, 1));     // F of S
  data.add(block.substring(1, 11));    // SVM-CODE
  data.add(block.substring(11, 16));   // JAAR
  data.add(block.substring(16, 18));   // MAAND
  data.add(block.substring(18, 20));   // DAG
  data.add(block.substring(20, 32));   // OPENINGSKOERS
  data.add(block.substring(32, 35));   // OPENINGSKOERS
  data.add(block.substring(35, 36));   // GERESERVEERD
  data.add(block.substring(36, 48));   // SLOTKOERS
  data.add(block.substring(48, 51));   // SLOTKOERS
  data.add(block.substring(51, 52));   // GERESERVEERD
  data.add(block.substring(52, 64));   // TOTALE VOLUME
  data.add(block.substring(64, 67));   // TOTALE VOLUME
```

```
    data.add(block.substring(67, 79));   // MAXIMUM KOERS
    data.add(block.substring(79, 82));   // MAXIMUM KOERS
    data.add(block.substring(82, 94));   // MINIMUM KOERS
    data.add(block.substring(94, 97));   // MINIMUM KOERS
    data.add(block.substring(97, 109));  // VOLUME BIJ OPENING
    data.add(block.substring(109, 112)); // VOLUME BIJ OPENING
    data.add(block.substring(112, 124)); // VOLUME BIJ AFSLUITING
    data.add(block.substring(124, 127)); // VOLUME BIJ AFSLUITING
    data.add(block.substring(127, 168)); // GERESERVEERD

    return data;
  }

  private Vector parseBlockCBRG(String block) throws Exception {
    Vector data = new Vector();

    data.add(block.substring(0, 1));    // C, B, R of G
    data.add(block.substring(1, 11));   // SVM-CODE
    data.add(block.substring(11, 16));  // JAARTAL
    data.add(block.substring(16, 18));  // MAAND
    data.add(block.substring(18, 20));  // DAG
    data.add(block.substring(20, 32));  // OPENINGSKOERS
    data.add(block.substring(32, 35));  // OPENINGSKOERS
    data.add(block.substring(35, 37));  // CODE OPENINGSKOERS
    data.add(block.substring(37, 49));  // SLOTKOERS
    data.add(block.substring(49, 52));  // SLOTKOERS
    data.add(block.substring(52, 54));  // CODE SLOTKOERS
    data.add(block.substring(54, 66));  // TOTAAL VOLUME
    data.add(block.substring(66, 69));  // TOTAAL VOLUME
    data.add(block.substring(69, 81));  // TOTAAL KAPITAAL PER DAG
    data.add(block.substring(81, 84));  // TOTAAL KAPITAAL PER DAG

    return data;
  }

  public boolean exists() {
    return exists;
  }

  private void printMsg(String msg) {
    log.writeMsg("[" + CLASSNAME + "]: " + msg);
  }
}
```

4.2 Code van Log

```
package scob.EI;

import java.io.File;
import java.io.FileWriter;

public class Log {
  private static final String CLASSNAME = "Log";

  private File file;
  private FileWriter filewriter;

  public Log(String prefix) {
    try {
      int i = 1;
      do {
        file = new File(Main.PATH_LOGFILE + prefix
               + "-" +
               Integer.toString(i) + ".log");
        i++;
      } while(file.exists());
      filewriter = new FileWriter(file);
      printMsg("Logfile " + file.getName()
               + " created.");
    } catch(Exception e) {
      e.printStackTrace();
    }
  }

  public void writeMsg(String msg) {
    try {
      filewriter.write(msg + "\n");
    } catch(Exception e) {
      e.printStackTrace();
    }
  }

  public void end() {
    try {
      filewriter.flush();
      filewriter.close();
      printMsg("Logfile " + file.getName() + " closed.");
    } catch(Exception e) {
```

```
      e.printStackTrace();
    }
  }

  private void printMsg(String msg) {
    System.out.println("[" + CLASSNAME + "]: " + msg);
  }
}
```

4.3 Code van ImportBrussels89

```
package scob.EI;

import scob.DB.*;
import java.sql.*;
import java.util.GregorianCalendar;
import java.util.Vector;

public class ImportBrussels89 {
  private static final String CLASSNAME = "ImportBrussels89";

  private static final java.sql.Date ENDDATE = new
  java.sql.Date(3999 - 1900, 12 - 1, 31);

  private Log log;
  private Connection con;

/*  private PreparedStatement pstmt_insertStock;
  private PreparedStatement pstmt_insertStockName;
  private PreparedStatement pstmt_insertStockCodeType;
  private PreparedStatement pstmt_insertNotation;
  private PreparedStatement pstmt_getHighestStock;
  private PreparedStatement pstmt_getHighestNotation;
*/
  private PreparedStatement pstmt_insertNotationPrice;
  private PreparedStatement pstmt_insertNotationTradedQuantity;
  private PreparedStatement pstmt_getStock;
  private PreparedStatement pstmt_getNotation;
  private PreparedStatement pstmt_getNotationPrice;
  private PreparedStatement pstmt_getNotationTradedQuantity;

  public ImportBrussels89() {
    Brussels89 b;
```

```
Vector data = null;
int files = 0;
int blocks = 0;
String filename = null;

try {
  log = new Log("start");

  printMsg("Process started (" +
  (new GregorianCalendar()).getTime().toString()
          + ").");

  con = (new JDBCLayer()).getConnection();
  printMsg("Connection created.");

  makePreparedStatements();

  log.end();

  int year;
  int n;

  for (year = 1989; year < 1999; ++year) {
    for (n = 1; n < 51; ++n) {
      filename = "hcrs" + Integer.toString(year) +
            "." + getN(n);
      log = new Log(filename);
      printMsg("Processing of " + filename +
            " started ("
            + (new GregorianCalendar()).getTime().toString()
            + ").");
      b = new Brussels89(log, Main.PATH_FILE + filename);
      if (b.exists()) {
        while ((data = b.getNextData()) != null) {
          handleData(data);
          blocks++;
        }
        printMsg(Integer.toString(blocks) +
              " block(s) found in " + filename);
        files++;
        blocks = 0;
      }
      printMsg("Processing of " + filename +
             " ended ("
```

```
                + (new GregorianCalendar()).getTime().toString()
                + ").");
            log.end();
          }
        }

        log = new Log("end");

        printMsg(Integer.toString(files) +
              " file(s) found.");

        con.close();
        printMsg("Connection closed.");

        printMsg("Process terminated (" +
              (new GregorianCalendar()).getTime().toString()
              + ").");

        log.end();
      } catch (Exception e) {
        printMsg("Error occurred in file " + filename + ",
        block number " + Integer.toString(blocks) + "!");
        printData(data);
        printMsg(e.toString());
      }
    }

    private String getN(int n) {
      String sn;
      if (n < 10) {
        sn = "00" + Integer.toString(n);
      } else if (n < 100) {
        sn = "0" + Integer.toString(n);
      } else if (n < 1000) {
        sn = Integer.toString(n);
      } else {
        sn = "---";
      }
      return sn;
    }

    private void makePreparedStatements() throws Exception {

  /*  pstmt_insertStock = con.prepareStatement(
```

```
      "INSERT INTO STOCK(ID, SHARETYPE, STOCKEXCHANGE) "
      + "VALUES(?, ?, ?)");
    pstmt_insertStockName = con.prepareStatement(
      "INSERT INTO STOCKNAME(STOCK, NAME, STARTDATE, ENDDATE) "
      + "VALUES(?, ?, ?, ?)");
    pstmt_insertStockCodeType = con.prepareStatement(
      "INSERT INTO STOCK_CODETYPE(STOCK, CODETYPE,
      CODE, STARTDATE, ENDDATE) "
      + "VALUES(?, ?, ?, ?, ?)");
    pstmt_insertNotation = con.prepareStatement(
      "INSERT INTO NOTATION(ID, STOCK, SECTOR,
      STARTDATE, ENDDATE) "
      + "VALUES(?, ?, ?, ?, ?)");
    pstmt_getHighestStock = con.prepareStatement(
      "SELECT MAX(ID) " +
      "FROM STOCK");
    pstmt_getHighestNotation = con.prepareStatement(
      "SELECT MAX(ID) " +
      "FROM NOTATION");
*/
    pstmt_insertNotationPrice = con.prepareStatement(
      "INSERT INTO NOTATION_PRICE(NOTATION, DAY,
      ARGENT, PAPIER, OPEN, CLOSE, MIN, MAX, PREVIOUS,
      EXTERN, PROCENT) "
      + "VALUES(?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?)");
    pstmt_insertNotationTradedQuantity = con.prepareStatement(
      "INSERT INTO NOTATION_TRADEDQUANTITY(NOTATION, DAY, VOLUME) "
      + "VALUES(?, ?, ?)");
    pstmt_getStock = con.prepareStatement(
      "SELECT STOCK, STARTDATE, ENDDATE " +
      "FROM STOCK_CODETYPE " +
      "WHERE CODE=? " +
      "ORDER BY STARTDATE DESC");
    pstmt_getNotation = con.prepareStatement(
      "SELECT ID, STARTDATE, ENDDATE " +
      "FROM NOTATION " +
      "WHERE STOCK=? " +
      "ORDER BY STARTDATE DESC");
    pstmt_getNotationPrice = con.prepareStatement(
      "SELECT NOTATION " +
      "FROM NOTATION_PRICE " +
      "WHERE NOTATION=? AND DAY=?");
    pstmt_getNotationTradedQuantity = con.prepareStatement(
      "SELECT NOTATION " +
```

```
      "FROM NOTATION_TRADEDQUANTITY " +
      "WHERE NOTATION=? AND DAY=?");
  }

/*  private void execInsertStock(double f1, double f2, double f3)
throws Exception {
    pstmt_insertStock.setDouble(1, f1);
    pstmt_insertStock.setDouble(2, f2);
    pstmt_insertStock.setDouble(3, f3);
//    int i = pstmt_insertStock.executeUpdate();
    int i = -1;
    printMsg("Tuple (" + Double.toString(f1)
        + ", " + Double.toString(f2) + ", "
        + Double.toString(f3) + ") inserted in
        STOCK [" + Integer.toString(i) + " tuple(s)].");
     }

  private void execInsertStockName(double f1, String f2,
        java.sql.Date f3,
        java.sql.Date f4) throws Exception {
    pstmt_insertStockName.setDouble(1, f1);
    pstmt_insertStockName.setString(2, f2);
    pstmt_insertStockName.setDate(3, f3);
    pstmt_insertStockName.setDate(4, f4); /
//    int i = pstmt_insertStockName.executeUpdate();
    int i = -1;
    printMsg("Tuple (" + Double.toString(f1) + ", "
          + f2 + ", "
          + f3.toString() + ", " + f4.toString() + ")
          inserted in STOCK_NAME ["
          + Integer.toString(i) + " tuple(s)].");
  }

  private void execInsertStockCodeType(double f1, double f2,
  String f3, java.sql.Date f4, java.sql.Date f5) throws Exception {
    pstmt_insertStockCodeType.setDouble(1, f1);
    pstmt_insertStockCodeType.setDouble(2, f2);
    pstmt_insertStockCodeType.setString(3, f3);
    pstmt_insertStockCodeType.setDate(4, f4);
    pstmt_insertStockCodeType.setDate(5, f5);
//    int i = pstmt_insertStockCodeType.executeUpdate();
    int i = -1;
    printMsg("Tuple (" + Double.toString(f1) + ", "
        + Double.toString(f2) + ", " + f3 + ", "
```

```
        + f4.toString() + ", " + f5.toString() + ")
        inserted in
        STOCK_CODETYPE [" + Integer.toString(i) + " tuple(s)].");
  }

  private void execInsertNotation(double f1, double f2,
  double f3, java.sql.Date f4, java.sql.Date f5) throws Exception {
    pstmt_insertNotation.setDouble(1, f1);
    pstmt_insertNotation.setDouble(2, f2);
    pstmt_insertNotation.setDouble(3, f3);
    pstmt_insertNotation.setDate(4, f4);
    pstmt_insertNotation.setDate(5, f5);
//    int i = pstmt_insertNotation.executeUpdate();
    int i = -1;
    printMsg("Tuple (" + Double.toString(f1) + ", "
        + Double.toString(f2) + ", "
        + Double.toString(f3) + ", "
        + f4.toString() + ", " + f5.toString() + ")
        inserted in
        NOTATION [" + Integer.toString(i) + " tuple(s)].");
  }

  private ResultSet execGetHighestStock() throws Exception {
    return pstmt_getHighestStock.executeQuery();
  }

  private ResultSet execGetHighestNotation() throws Exception {
    return pstmt_getHighestNotation.executeQuery();
  }
*/
  private void execInsertNotationPrice(double f1,
  java.sql.Date f2, double f3, double f4, double f5, double f6,
  double f7, double f8, double f9, double f10, String f11)
        throws Exception {
    pstmt_insertNotationPrice.setDouble(1, f1);
    pstmt_insertNotationPrice.setDate(2, f2);
    pstmt_insertNotationPrice.setDouble(3, f3);
    pstmt_insertNotationPrice.setDouble(4, f4);
    pstmt_insertNotationPrice.setDouble(5, f5);
    pstmt_insertNotationPrice.setDouble(6, f6);
    pstmt_insertNotationPrice.setDouble(7, f7);
    pstmt_insertNotationPrice.setDouble(8, f8);
    pstmt_insertNotationPrice.setDouble(9, f9);
    pstmt_insertNotationPrice.setDouble(10, f10);
```

```
    pstmt_insertNotationPrice.setString(11, f11);
    int i = pstmt_insertNotationPrice.executeUpdate();
//    int i = -1;
    printMsg("Tuple (" + Double.toString(f1) + ", " +
         f2.toString() + ", "
         + Double.toString(f3) + ", "
         + Double.toString(f4) + ", "
         + Double.toString(f5) + ", "
         + Double.toString(f6) + ", "
         + Double.toString(f7) + ", "
         + Double.toString(f8) + ", "
         + Double.toString(f9) + ", "
         + Double.toString(f10) + ", "
         + f11 + ") inserted in
         NOTATION_PRICE [" + Integer.toString(i) + " tuple(s)].");
  }

  private void execInsertNotationTradedQuantity(double f1,
  java.sql.Date f2, double f3) throws Exception {
    pstmt_insertNotationTradedQuantity.setDouble(1, f1);
    pstmt_insertNotationTradedQuantity.setDate(2, f2);
    pstmt_insertNotationTradedQuantity.setDouble(3, f3);
    int i = pstmt_insertNotationTradedQuantity.executeUpdate();
//    int i = -1;
    printMsg("Tuple (" + Double.toString(f1) + ", "
        + f2.toString() + ", "
        + Double.toString(f3) + ")
        inserted in
        NOTATION_TRADEDQUANTITY ["
        + Integer.toString(i) + " tuple(s)].");
  }

  private ResultSet execGetStock(String f1) throws Exception {
    pstmt_getStock.setString(1, f1);
    return pstmt_getStock.executeQuery();
  }

  private ResultSet execGetNotation(String f1) throws Exception {
    pstmt_getNotation.setString(1, f1);
    return pstmt_getNotation.executeQuery();
  }

  private ResultSet execGetNotationPrice(double f1, java.sql.Date f2)
  throws Exception {
```

```
    pstmt_getNotationPrice.setDouble(1, f1);
    pstmt_getNotationPrice.setDate(2, f2);
    return pstmt_getNotationPrice.executeQuery();
  }

  private ResultSet execGetNotationTradedQuantity(double f1,
           java.sql.Date f2)
  throws Exception {
    pstmt_getNotationTradedQuantity.setDouble(1, f1);
    pstmt_getNotationTradedQuantity.setDate(2, f2);
    return pstmt_getNotationTradedQuantity.executeQuery();
  }

  private void handleData(Vector data) throws Exception {
    if (data.elementAt(0).equals("F") ||
    data.elementAt(0).equals("S")) {
      handleDataFSCBRG(data, true);
    } else if (data.elementAt(0).equals("C") ||
    data.elementAt(0).equals("B")) {
      handleDataFSCBRG(data, false);
    } else if (data.elementAt(0).equals("R") ||
    data.elementAt(0).equals("G")) {
//      handleDataCBRG(data);
    } else {
      printMsg("Invalid ID " + data.elementAt(0));
    }
  }

  private void handleDataFSCBRG(Vector data, boolean fs)
     throws Exception {
    // svmcode: leading zero's eliminated, always ending at .--
    String svmcode = Double.toString(Double.parseDouble(
      ((String)data.elementAt(1)).substring(0, 8) +
      "." +
      ((String)data.elementAt(1)).substring(8, 10)));

    java.sql.Date date = new java.sql.Date(
      Integer.parseInt((String)data.elementAt(2)) - 1900,
      Integer.parseInt((String)data.elementAt(3)) - 1,
      Integer.parseInt((String)data.elementAt(4)));

    String stock = null;
    boolean makeNewStock = false;
    ResultSet rs_stock = execGetStock(svmcode);
```

```
    if (rs_stock.next()) {
      // at least one tuple found
      do {
        if (date.equals(rs_stock.getDate(2)) ||
        date.after(rs_stock.getDate(2))) {
          if (date.before(rs_stock.getDate(3))) {
            stock = rs_stock.getString(1);
            break;
          } else {
            makeNewStock = true;
          }
        }
      } while (rs_stock.next());
    } else {
      // no tuples found
      makeNewStock = true;
    }
    if (makeNewStock) {
      printMsg("No stock ID found with SVM-code " + svmcode + ".");

/*      ResultSet rs_highestStock = execGetHighestStock();
      if (rs_highestStock.next()) {
        stock = Double.toString(Double.parseDouble
             (rs_highestStock.getString(1))
             + 1);
        execInsertStock(Double.parseDouble(stock), 1, 1);
        execInsertStockName(Double.parseDouble(stock),
        "Nieuw Aandeel: SVM/SRW: " + svmcode + "
        Zoek van dit aandeel de naam op in de officiele koerslijsten.",
               date, ENDDATE);
        execInsertStockCodeType(Double.parseDouble(stock), 2, svmcode,
               date, ENDDATE);
      } else {
        throw new Exception("No highest stock found.");
      }
*/
    }

    if (stock == null) { // er is geen stock gevonden
      printMsg("No stock ID found.");
      return;
    }

    String notation = null;
```

```
    boolean makeNewNotation = false;
    ResultSet rs_notation = execGetNotation(stock);
    if (rs_notation.next()) {
      // at least one tuple found
      do {
        if (date.equals(rs_notation.getDate(2)) ||
        date.after(rs_notation.getDate(2))) {
          if (date.before(rs_notation.getDate(3))) {
            notation = rs_notation.getString(1);
            break;
          } else {
            makeNewNotation = true;
          }
        }
      } while (rs_notation.next());
    } else {
      // no tuples found
      makeNewNotation = true;
    }
    if (makeNewNotation) {
      printMsg("No notation ID found with stock ID " + stock + ".");

/*      ResultSet rs_highestNotation = execGetHighestNotation();
      if (rs_highestNotation.next()) {
        notation = Double.toString(
        Double.parseDouble(rs_highestNotation.getString(1)) + 1);
        execInsertNotation(Double.parseDouble(notation),
        Double.parseDouble(stock), sector, date, ENDDATE);
      } else {
        throw new Exception("No highest stock found.");
      }
*/
    }

    if (notation == null) {
      printMsg("No notation ID found.");
      return;
    }

    ResultSet rs_notationPrice = execGetNotationPrice(
    Double.parseDouble(notation), date);
    if (rs_notationPrice.next()) {
      // at least one tuple found
    } else {
```

```
// no tuples found
if (fs) {
  execInsertNotationPrice(
    Double.parseDouble(notation),
    date,
    0,
    0,
    Double.parseDouble(data.elementAt(5) + "." +
    data.elementAt(6)),
    Double.parseDouble(data.elementAt(8) + "." +
    data.elementAt(9)),
    Double.parseDouble(data.elementAt(15) + "." +
    data.elementAt(16)),
    Double.parseDouble(data.elementAt(13) + "." +
    data.elementAt(14)),
    0,
    0,
    "N");
} else {
  String argent;
  String papier;
  String open;

  if (data.elementAt(7).equals("CA")) {
    argent = data.elementAt(5) + "." + data.elementAt(6);
    papier = "0";
    open = "0";
  } else if (data.elementAt(7).equals("CP")) {
    argent = "0";
    papier = data.elementAt(5) + "." + data.elementAt(6);
    open = "0";
  } else {
    argent = "0";
    papier = "0";
    open = data.elementAt(5) + "." + data.elementAt(6);
  }

  execInsertNotationPrice(
    Double.parseDouble(notation),
    date,
    Double.parseDouble(argent),
    Double.parseDouble(papier),
    Double.parseDouble(open),
    Double.parseDouble(data.elementAt(8) + "."
```

```
                + data.elementAt(9)),
            0,
            0,
            0,
            0,
            "N");
      }
    }

    ResultSet rs_notationTradedQuantity =
    execGetNotationTradedQuantity(Double.parseDouble(notation), date);
    if (rs_notationTradedQuantity.next()) {
      // at least one tuple found
    } else {
      // no tuples found
      // exactly same positions for FS and CBRG
      execInsertNotationTradedQuantity(
        Double.parseDouble(notation),
        date,
        Double.parseDouble(data.elementAt(11) + "."
            + data.elementAt(12)));
    }
  }

  private void printData(Vector data) {
    for (int i = 0; i < data.size(); i++) {
      printMsg("[" + Integer.toString(i) + "] " + data.elementAt(i));
    }
  }

  private void printMsg(String msg) {
//    System.out.println("[" + CLASSNAME + "]: " + msg);
    log.writeMsg("[" + CLASSNAME + "]: " + msg);
  }
}
```

4.4 Code van Main

```
package scob.EI;

import java.util.Vector;

public class Main {
  public static final String PATH_FILE = "g:\\EI\\koersen\\";
```

```
  public static final String PATH_LOGFILE = "g:\\EI\\log\\";

  public Main() {
  }

  public static void main(String[] args) {
    new ImportBrussels89();
  }
}
```

References

[Dex02] Nele Dexters. Omzetting van complexe beurskoersinformatie in een relationele databankstructuur, de beurs van brussel 1830-2002: van een theoretisch ontwerp naar een geslaagde implementatie. Technical report, University of Antwerp, 2002.

[Eck00] Bruce Eckel. *Thinking in Java*. Prentice Hall PTR, 2 edition, 2000.

[jav] java. http://www.java.sun.com.

[sco] scob. http://www.scob.be.