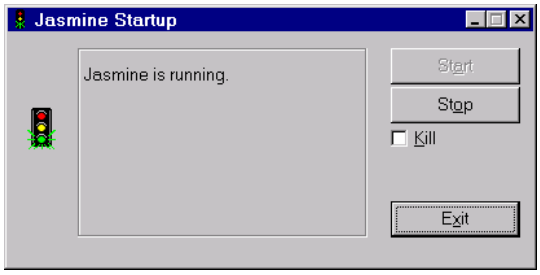
Introduktion till Jasmine 1.2 ODQL

I detta avsnitt beskrivs ett antal praktiska handgrepp som behövs för att köra Jasmine ODQL.

1 ODQL miljön

Man kan enklast köra ODQL mot Jasmine från ett vanligt Command Prompt fönster. ODQL miljön startar man med kommandot **codqlie** och avslutar med kommandot **end;**. För att ODQL miljön ska fungera måste Jasmine vara igång. Man kan starta Jasmine (eller kontrollera att den är igång) från Jasmine Startup fönstret (*Start Menu > Programs > DatabaseManagementSystems, Jasmine > Jasmine Startup*):



Kommandot **codqlie** kan också ta ett antal parametrar:

```
Usage: codqlie [-h|-help|-?] [-dbName string] [-userName string]
          [-passwd string] [-envFile string] [-execFile string]
          [-displayProcessedODQL] [-useExplicitTransactions]
          [-silent]

Option(s):
 -h or -help or -?        this list
 -dbName      string      database name
 -userName    string      login Id
 -passwd      string      password
 -envFile     string      environment file for this session
 -execFile    string      text file containing commands
 -displayProcessedODQL    display ODQL text just prior to execution
 -useExplicitTransactions  turn off implicit transaction handling
 -silent                   do not display startup info
```

Det mest intressanta här är parametern **–execFile**. Man kan använda en fil där man sparar alla ODQL kommandon, och sedan köra den automatiskt när man kommer till ODQL miljön.

Alla kommandon i ODQL miljön måste avslutas med semikolon (**;**). Kommentarer i ODQL börjar med **/*** och avslutas med ***/.** Det är viktigt att komma ihåg att Jasmine ODQL är *case sensitive*! Alla kommandon måste skrivas på rätt sätt. Alla "SQL" ord stavas alltid med gemener, resten bör man kontrollera i Jasmine Online Documentation om man inte vet.

ODQL miljön har en egen buffert, där alla deklarerade variabler finns. Enklast kan man tömma bufferten genom att lämna ODQL miljön med kommandot **end;**. Det rekommenderade sättet att arbeta är att man bygger in alla kommandon (inklusive kommandot **end;**) i en fil, som man sedan använder som parameter i kommandot **codqlie**.

När man kommer in i ODQL miljön, använder systemet den default klassfamiljen **systemCF**. För att ändra till en annan klassfamilj använder man kommandot **defaultCF** och namnet på den klassfamilj man vill använda. Här är ett exempel:

defaultCF CAStore;

Klassfamiljen **CAStore** innehåller den exempeltillämpning som följer med Jasmine. (Observera att CAStore är anpassad för kursen genom ändringar i modellen och inmatning av flera instanser.)

Här följer ett enkelt exempel på hur man kan göra för att exekvera en **select** sats och visa resultatet:

Skapa en fil (t ex **D:\ODQLtemp\fil1.txt**) med följande innehåll:

```
defaultCF CAStore;
String str;
str = select Customer.shoesize from Customer where Customer.name == ”Donald Duck”;
str.print();
end;
```

Först sätter man **CAStore** som default klassfamilj. Den andra raden deklarerar stängvariabeln **str**. Den tredje raden tilldelar variabeln **str** med **select** satsens resultat (Donald Ducks skonummer). Den fjärde raden skriver ut **str** på skärmen. Man kan exekvera filen direkt från **kommando prompt** med följande kommando:

```
codqlie -execFile D:\ODQLtemp\fil1.txt
```

Detta skulle resultera i att siffran 42 (Donald Ducks skonummer) kommer på skärmen, och ODQL miljön avslutas.

2 Mängdhantering

ODQL liknar i mycket SQL. En viktig skillnad är att man i ODQL kan arbeta explicit med mängder. De vanligaste mängderna är ”bags”, alltså egentligen inte mängder eftersom de kan innehålla dubbletter. En bag ser ut så här:

```
Bag{1, 2, 4, 1, 8}
```

ODQL tillhandahåller ett antal mängdoperationer, de viktigaste är:

union	***differ***	***scan***
intersect	***hasElement***	***directAdd***

Detaljerad information om och exempel på dessa operationer kan man hitta i Jasmine Online Documentation (*Start Menu > Programs > DatabaseManagementSystems > Jasmine > Jasmine Online Documentation*). Där hittar man också andra användbara operationer som inte nämns i den här introduktionen.

Här är ett litet exempel om mängdhantering:

```
Integer set is1;
Integer set is2;
Integer set is3;
Integer set is4;
Integer set is5;

is1 = Bag{1,2,3,4};
is2 = Bag{1,3,5,7};
is3 = Bag{};
is4 = Bag{};

is3 = is1.union(is2);
is3.print();

is3 = is1.intersect(is2);
is3.print();

is4.directAdd(5);
is4.directAdd(81);
is4.print();

is4 = is4.add(73);
```

```
is4.directRemove(81);
is4.print();

is5.print();
is5.directAdd(5);
is5.print();
```

Exekvering av dessa kommandon resulterar i följande:

```
Bag{ 4, 2, 1, 3, 5, 7 }
Bag{ 1, 3 }
Bag{ 5, 81 }
Bag{ 5, 73 }
NIL
NIL
```

Det är viktigt att observera här att en mängd-variabel som inte har instansierats har värdet **NIL**. En del mängdfunktioner (som **add()** och **directAdd()**) fungerar då inte på variabeln. I exemplet var variabeln **is5 NIL** innan och efter **directAdd()**. Variabeln **is4** å andra sidan tilldelades först en tom mängd. Därefter kunde man utföra **add()** och **directAdd()** operationer på den.

I exemplet ovan användes mängder (**set**) av heltal (**Integer**). I Jasmine ODQL kan man använda följande basklasser (motsvarar datatyper):

```
Integer        Real        String        Date
```

Utöver dem kan man använda sig av alla klasser som man har i den aktuella klassfamiljen. Om man till exempel har en klass **Customer**, kan man definiera variabler och tilldela dem enligt följande exempel:

```
Customer c1;
Customer set cs1;
cs1 = select Customer from Customer;
```

Här är ett annat exempel som använder operationen **hasElement()**:

```
/* Vilka accessoarer innehåller inte blått? */
String set strset;
```

```
strset = select Accessory.name
from Accessory
where not(Accessory.colors.hasElement("blue"));
strset.print();
```

Först ges en kommentar inom /*…*/. Därefter deklareras en variabel **strset** som är mängdvärd. Därefter tilldelas **strset** namnen på de accessoarer som inte är blåa. **Accessory.colors** är en mängd, därför kan man använda en mängdoperation. Till slut skrivs värdet på **strset** ut. Resultatet blir så här:

```
Bag{ Red Umbrella, Beige Bag, Pigskin Bag, Red Bag, Large bag, Red Sun Hat, Black Satin Sandal, Loafer, Sandal }
```

Man kan iterera över en mängd med hjälp av **scan()**-satsen. I följande exempel skrivs namnen på alla kunder ut:

```
Customer c1;
Customer set cs1;

cs1 = select Customer from Customer;

scan(cs1, c1) {
        c1.name.print();
};
/* c1 är en variabel som itererar över elementen i cs1 */
```

Resultatet blir såhär:

```
Scrooge McDuck
Donald Duck
Huey Duck
Dewey Duck
Louie Duck
Etc.
```

3 Imperativa konstruktioner

ODQL innehåller vanliga imperativa programspråkskonstruktioner som if-then-else och while. I exemplet nedan skrivs en mängd om den innehåller elementet 2.

```
Integer set is1;
Integer set is2;

is1 = Bag{1,2,3,4};
is2 = Bag{1,3,5,7};

if (is1.hasElement(2)) {
   is1.print();
};

if (is2.hasElement(2)) {
   is2.print();
};
```

4 ODQL och Jasmine studio

Mycket av det som går att göra i ODQL kan också göras i Jasmine Studio, men Studio har åtskilliga begränsningar.

5 Resurser

För det mesta kan man hitta information i Jasmine Online Documentation.
Start Menu > Programs > DatabaseManagementSystems > Jasmine > Jasmine Online Documentation