DB2TextExtender

version 2.2

Inledning

DB2 TextExtender är ett tillägg till DB2 Universal Database v7.2. Det kan beskrivas som ett "full-text retrieval program" som möjliggör sökning av textfiler som antingen är lagrade i databasen eller lagrade som externa filer utanför databashanterarens kontroll. TextExtender realiserar sökningen av textdokument genom att söka i ett på förhand definierat index. Sökning sker alltså inte i själva textdokumentet.

Text Extender består huvudsakligen av tre delar. Dessa är:

Command line Interpreter. Detta är en kommandoprompt för att utföra kommandon som är specifika för TextExtender (t.ex. för att indexera dokument). Många av de kommandon man använder när man söker i dokument och skapar tabeller m.m. utförs huvudsakligen i DB2s vanliga miljö (Command Center, Control Center).

User-Defined functions (UDFs). Funktioner som inkluderas i vanliga SQL frågor för att på så sätt möjliggöra sökning i textdokument. I och med att UDF:erna är som ett tillägg till SQL sker sökning som vanligt från Command Center och man kan även integrera frågor på vanliga kolumner (t ex namn, datum etc) med sökningen i textdokument (se bilaga om sökning).

Application programming interface (API). Dessa är funktioner som man kan anropa från C-program för att söka i textdokument och för att visa sökresultat.

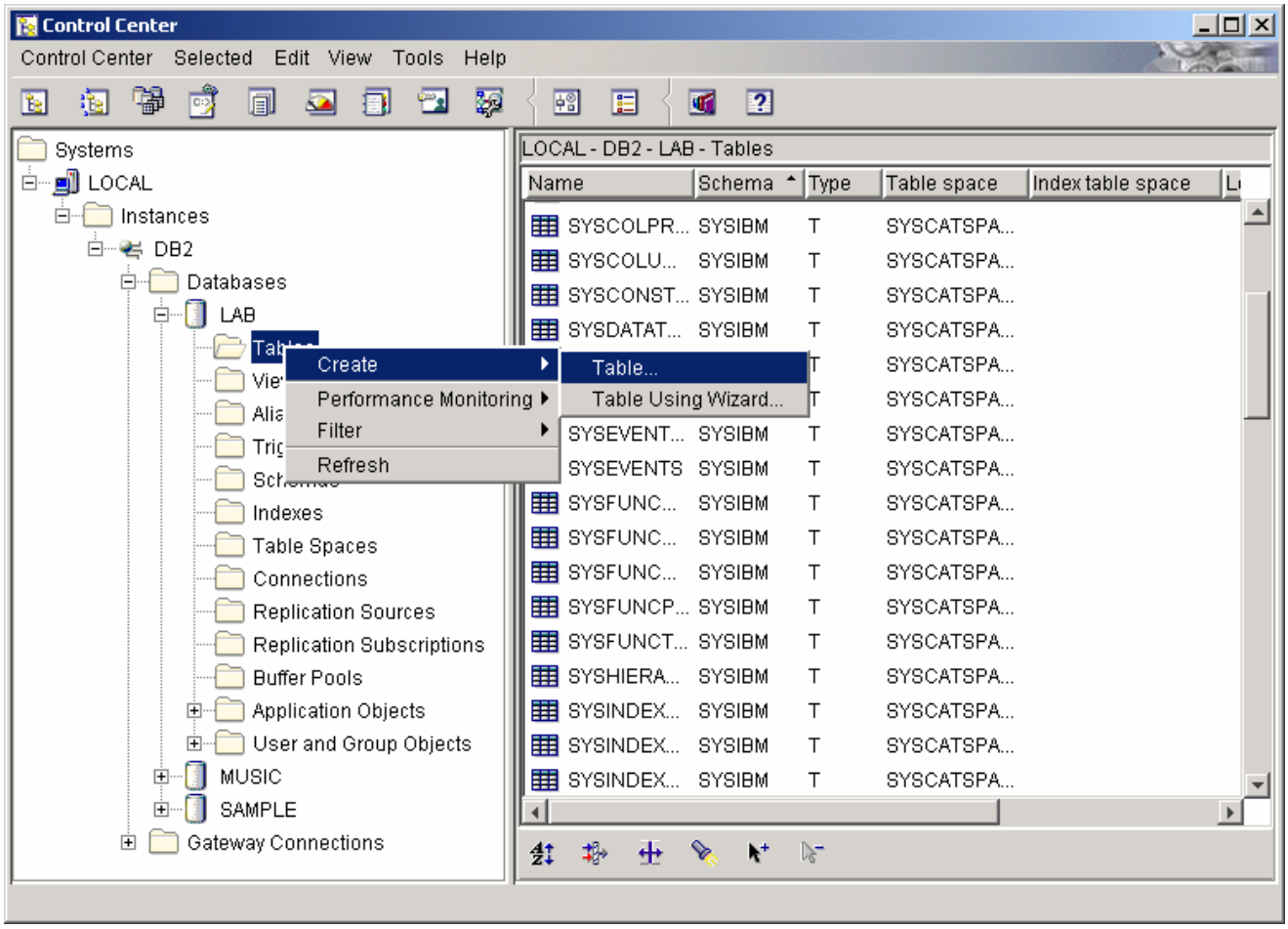
Att arbeta med DB2 TextExtender

Detta kapitel beskriver hur man skapar en fulltextdatabas bestående av magisteruppsatser samt uppgifter om dess författare, titel, år och språk. Denna fulltextdatabas kommer sedan att användas för att göra utsökningar med avseende på uppsatsernas innehåll.

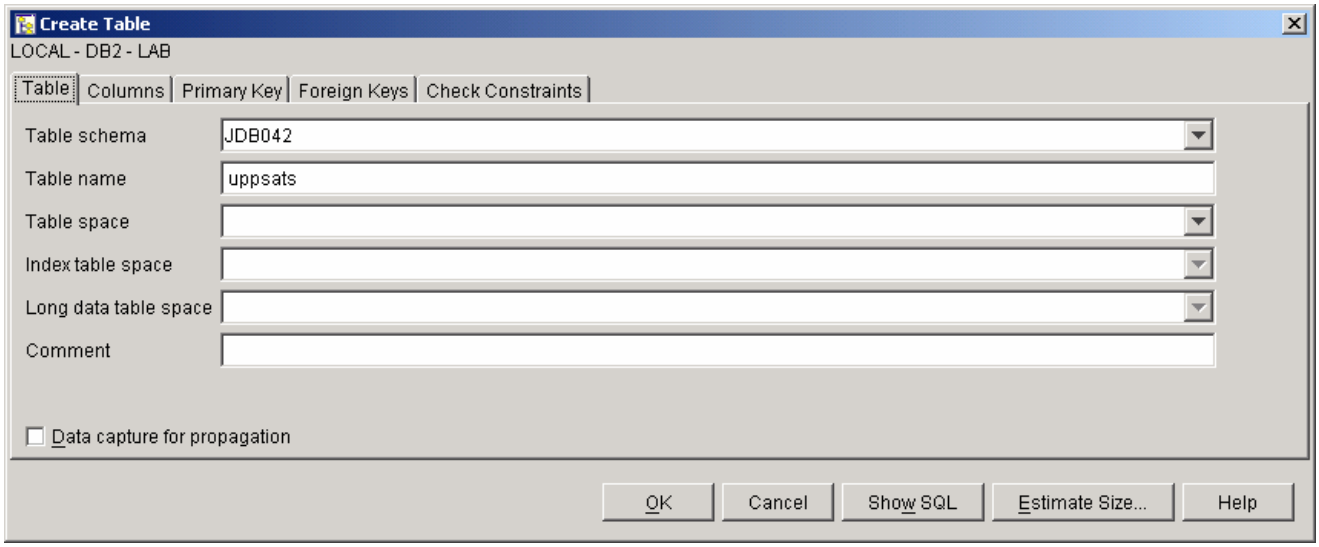
Skapa databasen

1. Skapa en databas via **Control Center** och namnge den *Lab*

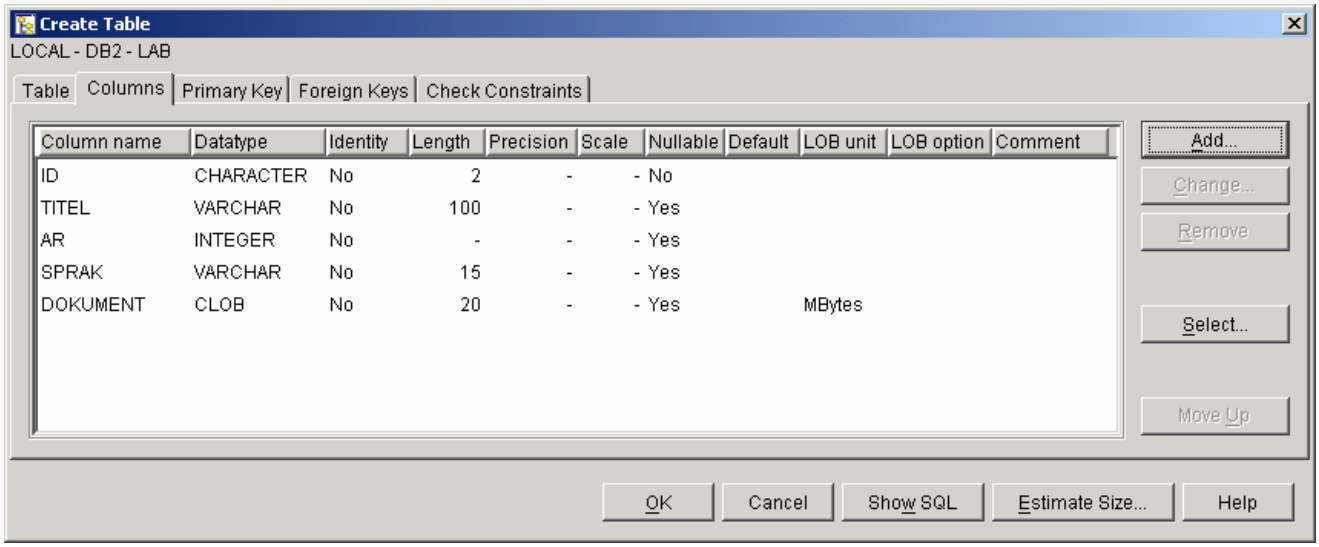
2. Skapa en tabell som heter *Uppsats* från DB2 **Control Center** enligt följande



- Ange ditt schema som Table schema (i det här fallet använder vi JDB042). Se nedan.

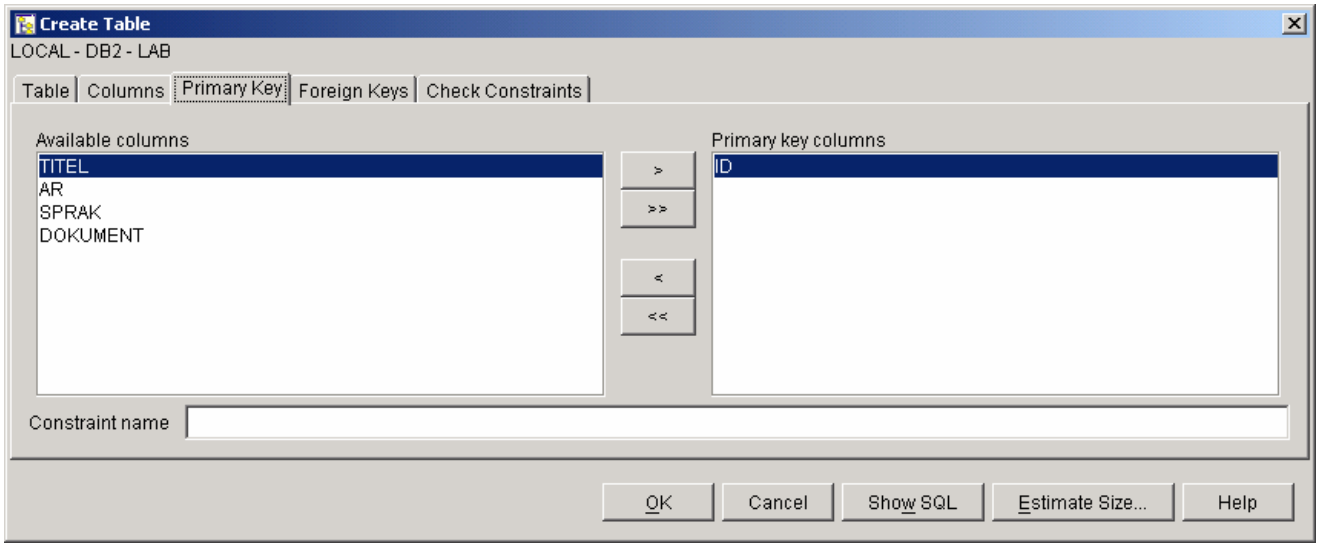


- Definiera samma kolumner som bilden nedan (i Columns tabben).



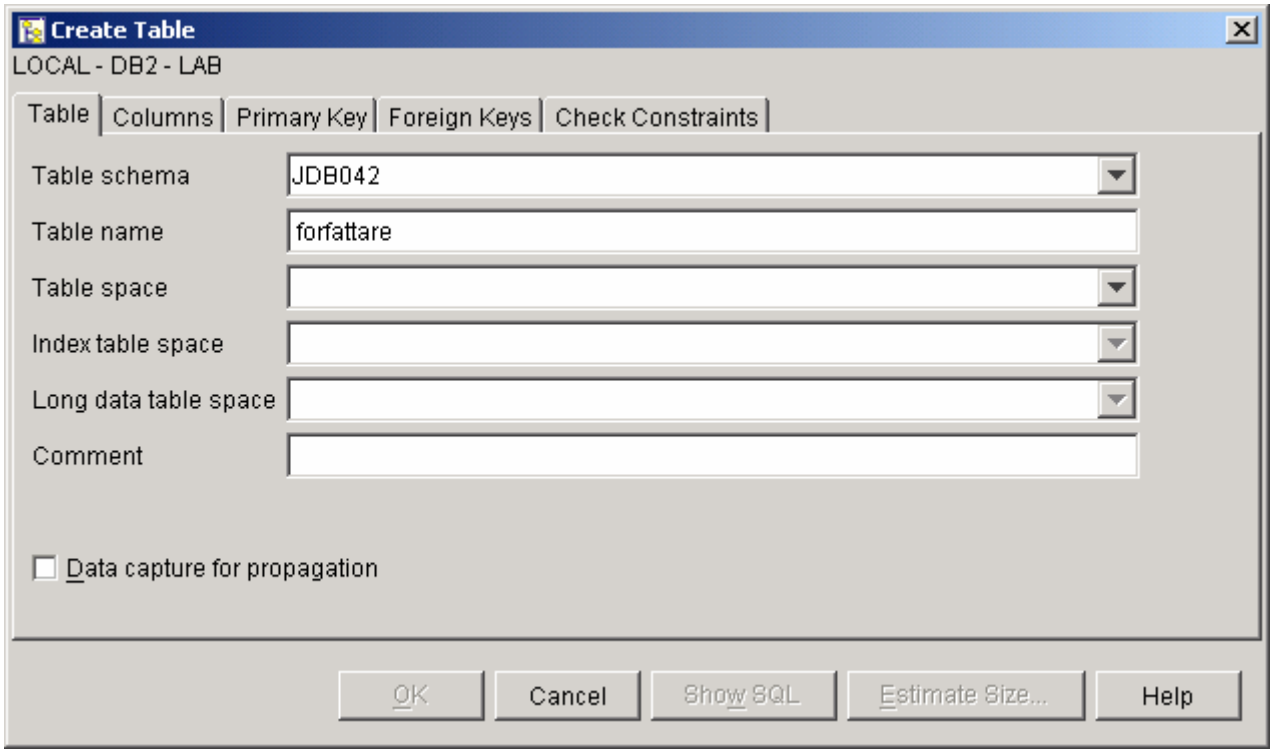
DOKUMENT kolumnen ska vara av datatypen CLOB (Character Large OBject). Det är denna kolumn som ska innehålla filerna med uppsatser som senare skall importeras. Ändra storleken på CLOB:en (till ca 20Mb) för att försäkra att dokumentena får plats.

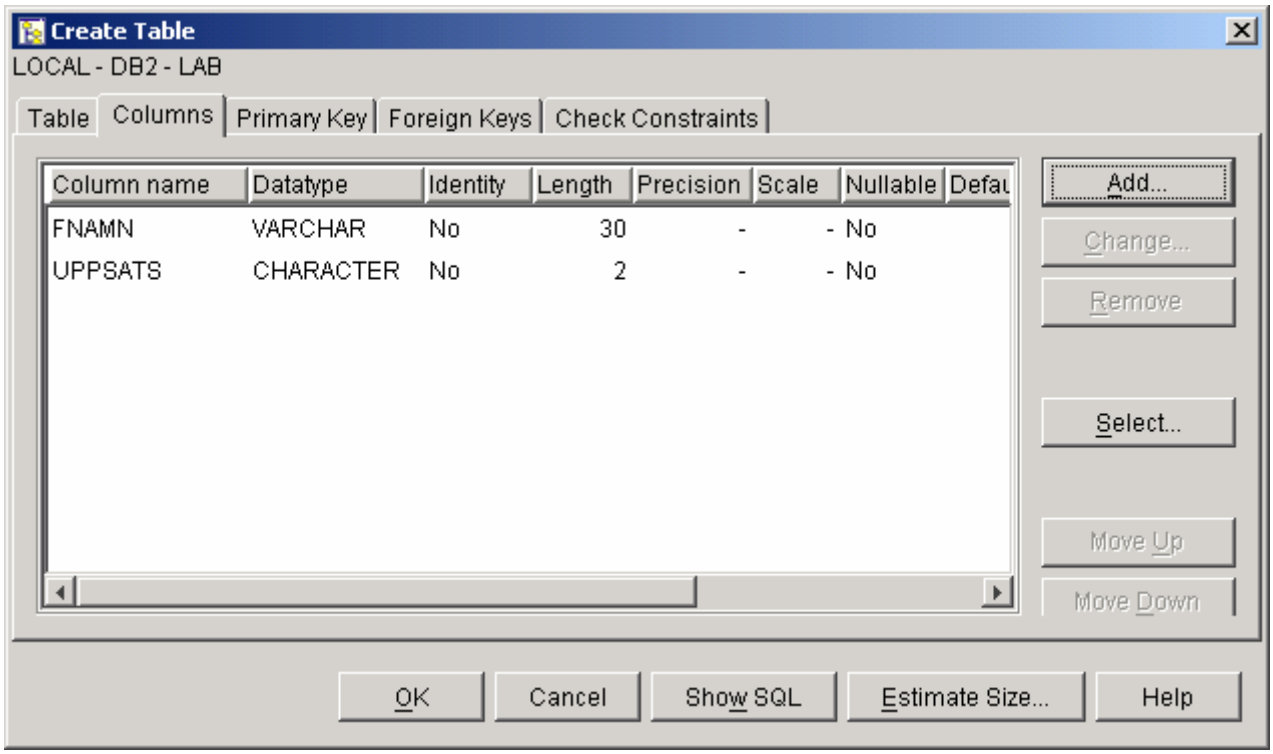
- Definiera kolumnen ID som primärnyckel (i Primary Key tabben):

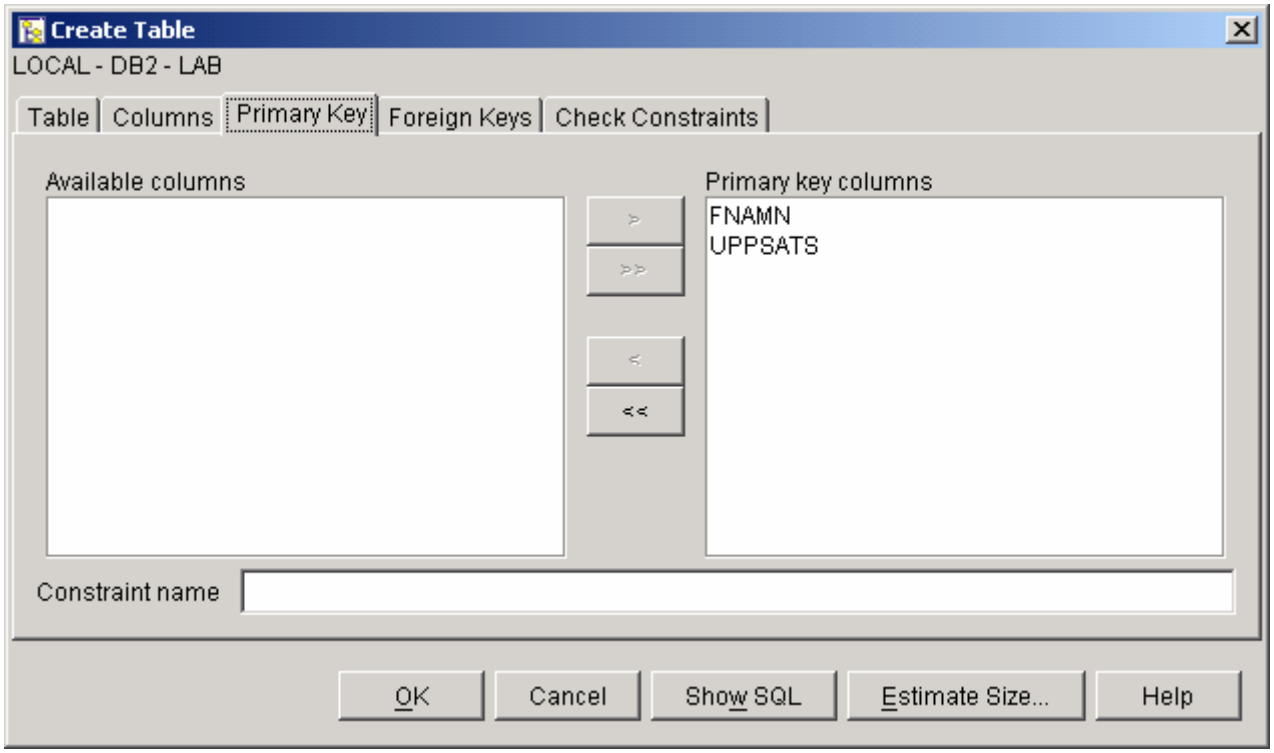


- Klicka OK.

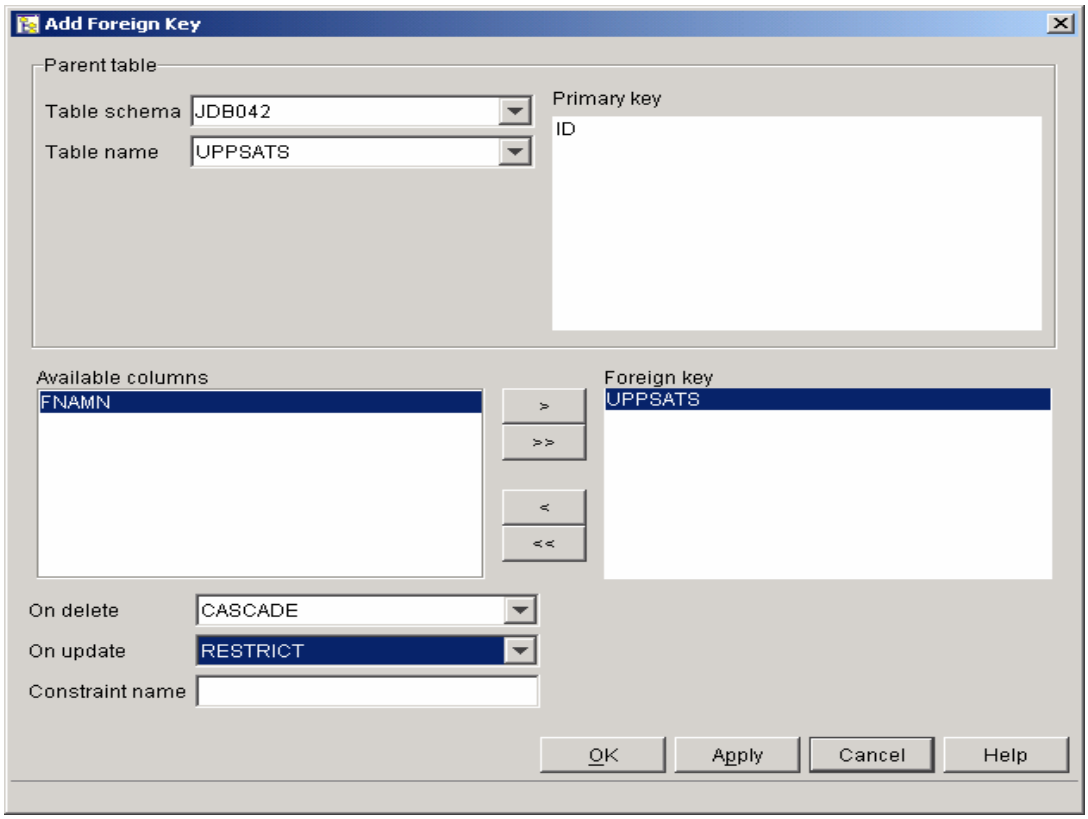
3. Skapa också tabellen forfattare enligt följande:

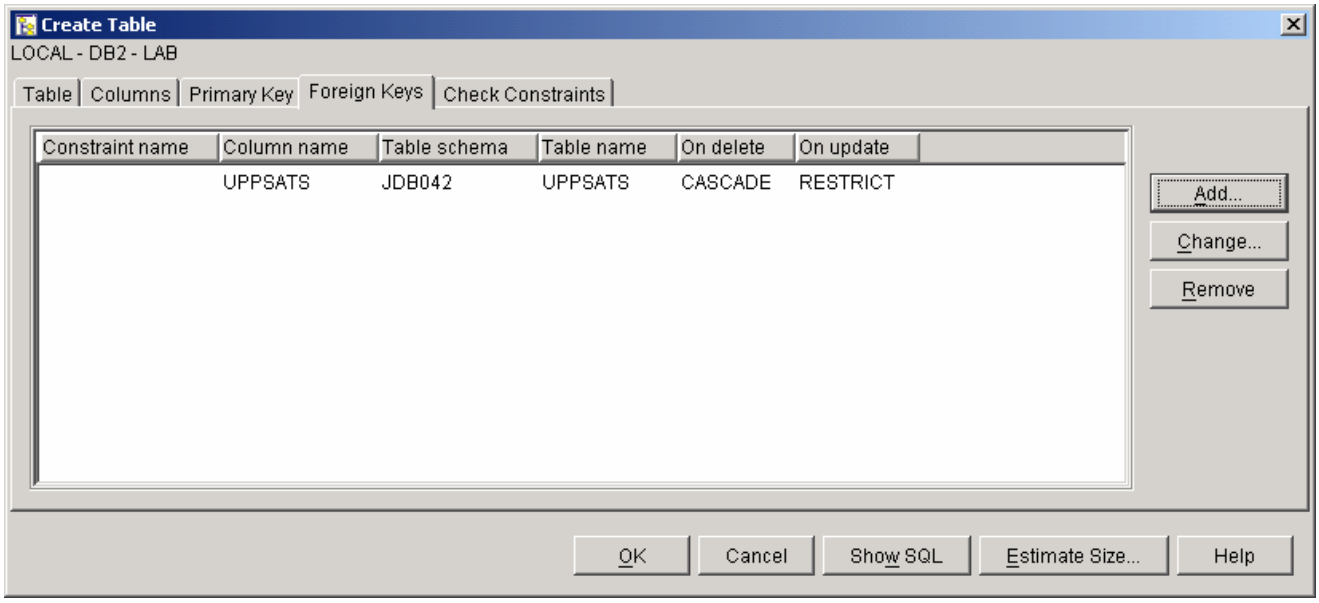

Create Table
LOCAL - DB2 - LAB
Table
Columns
Primary Key
Foreign Keys
Check Constraints
Table schema
JDB042
Table name
forfattare
Table space
Index table space
Long data table space
Comment
Data capture for propagation
OK
Cancel
Show SQL
Estimate Size...
Help


Create Table
LOCAL - DB2 - LAB
Table
Columns
Primary Key
Foreign Keys
Check Constraints
Column name
Datatype
Identity
Length
Precision
Scale
Nullable
Defau
FNAMN VARCHAR No 30 - - No
UPPSATS CHARACTER No 2 - - No
Add...
Change...
Remove
Select...
Move Up
Move Down
OK
Cancel
Show SQL
Estimate Size...
Help



- Lägg till även en främmande nyckel enligt följande:





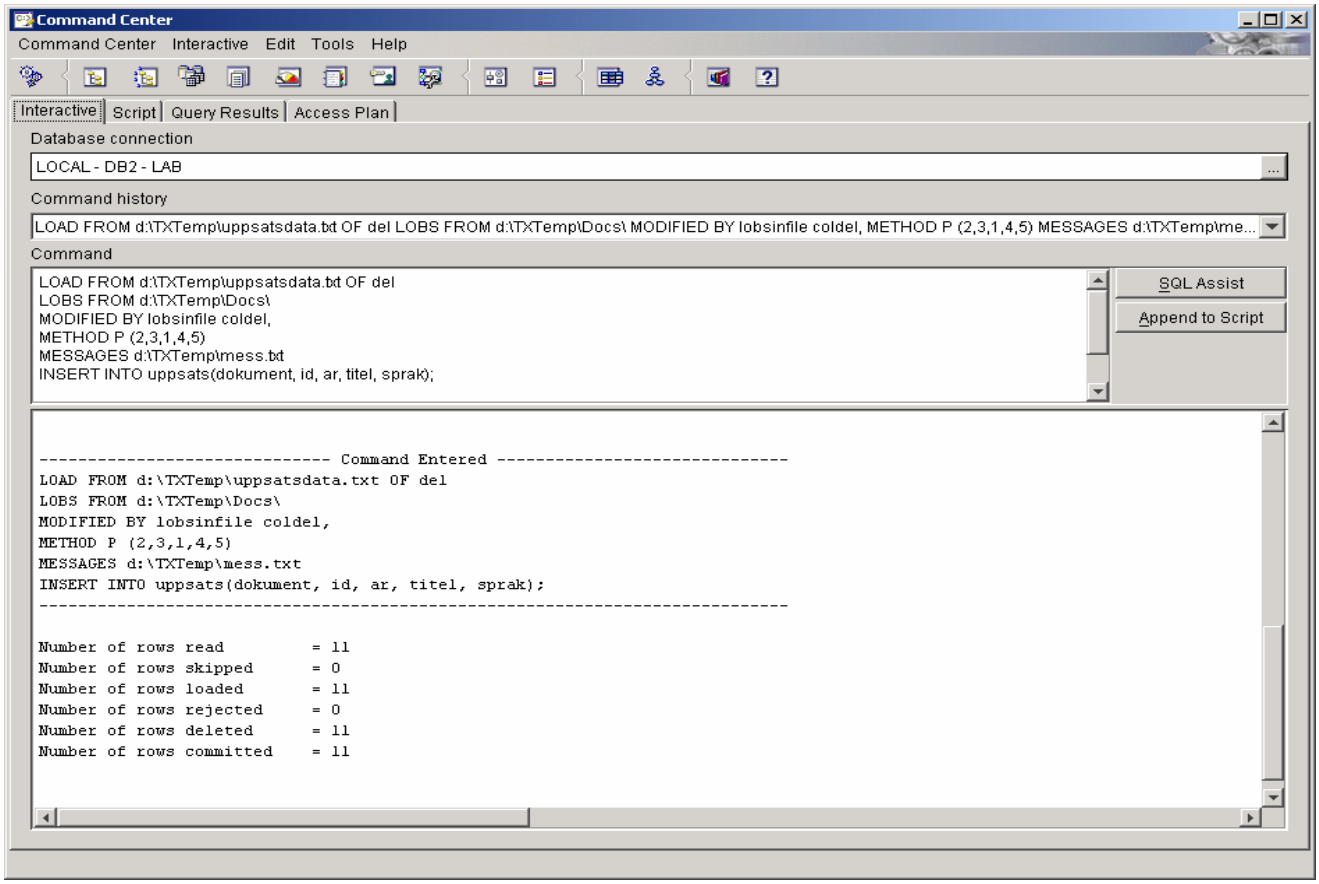
4. Skapa en mapp under er databasprofil på M: som heter *exempel*: **M:*exempel***

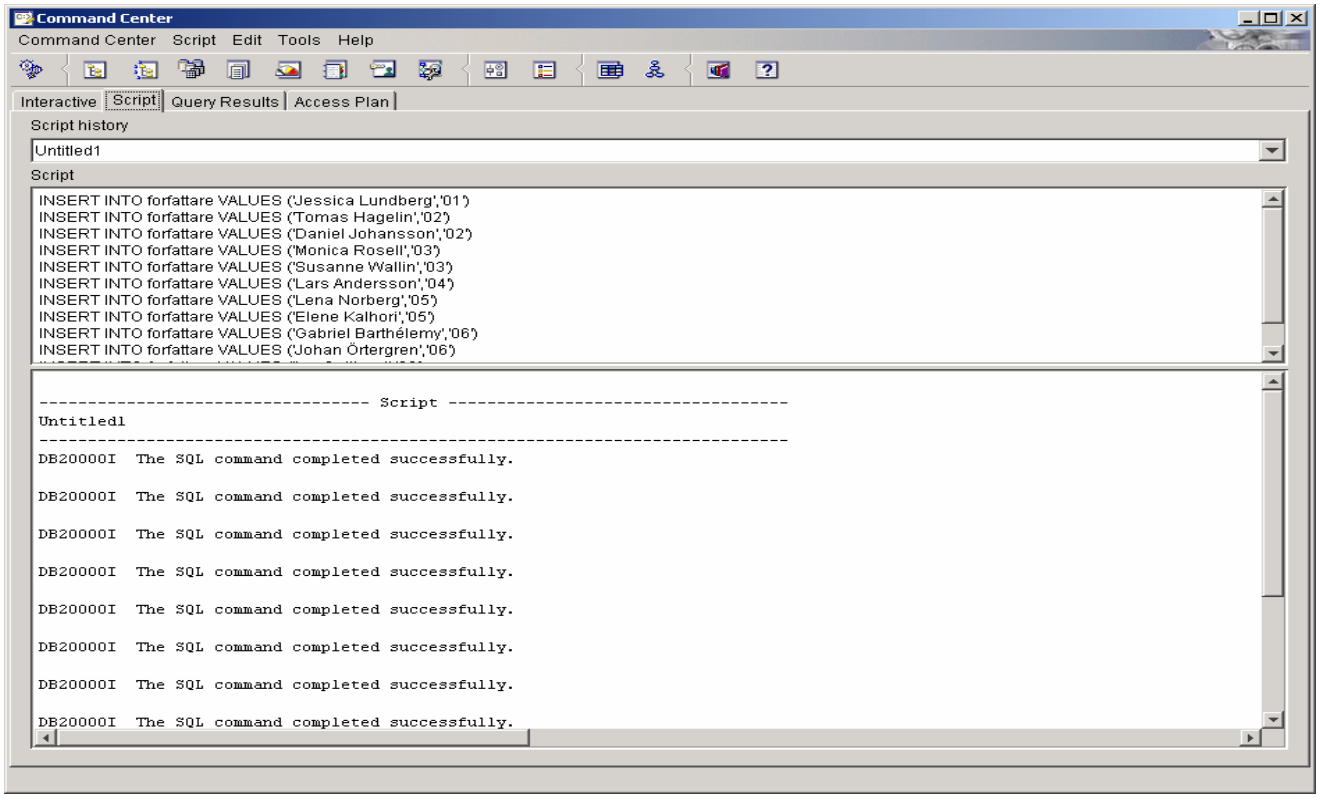
5. Ladda ner uppsatserna (.doc och .rtf filerna) till exempelmappen från \\DB-SRV-1\StudKursInfo\Stjärna62 ht2003\Labb2\Documents

6. Gå in i **Command Center** och anslut till databasen med nedanstående kommando:

CONNECT TO lab

7. Ladda in dokumenten och andra data i de skapade tabellerna genom att köra det speciella kommandot LOAD och sedan INSERT-satserna från populate.txdb.script i **Command Center**:





Förklaringen av LOAD-satsen:

LOAD kommandot används för att ladda in CLOBs i databasen. Huvudsyntaxen är följande:

LOAD FROM *sökväg*
OF *format-typ*
LOBS FROM *CLOB-sökväg*
MODIFIED BY *lista av modifierare*
METHOD *kolumnmappningsmetod*
MESSAGES *filnamn*
INSERT INTO *tabell* (*lista av kolumner*)

- **Den första sökvägen** anger var filen med data (det som skall laddas i tabellen) finns.

- **Format-typen** anger formatet för den ovanspecificerade filen. *DEL* är ett giltigt värde och betyder att filen är av formatet Delimited ASCII.

- **CLOB-sökvägen** anger var CLOB-dokumenten finns. Man förväntas då ha angett bara filnamnet i datafilen som specificerades av den första sökvägen. CLOB-sökvägen läggs då framför dessa filnamn.

- **Lista av modifierare** anger vilka olika typer av ändringar som skall appliceras på den inlästa datafilen (som specificerades av den första sökvägen). Här finns det ett antal olika värden som är tillåtna. Värdena skall vara separerade med blanktecken. Giltiga värden innefattar *LOBSINFILE och COLDEL**x*** där x är det tecken som skiljer värdena i datafilen. Anger man *LOBSINFILE* betyder detta att CLOB:arnas fullständiga namn har modifierats av CLOB-sökvägen som specificerades tidigare. Anger man COLDEL, betyder detta att värdena i datafilen är separerade av ett komma-tecken (,).

- **Kolumnmappningsmetod** specificerar på vilket sätt (och i vilken ordning) man vill läsa in värdena från datafilen. Det finns tre olika metoder. Vi använder här metoden P som betyder att man kan ange kolumnernas ordning genom att ange kolumnernas position i datafilen. T ex P (1,3,2) betyder att man vill läsa in kolumn 1 först, kolumn 3 sedan, och kolumn 2 sist.

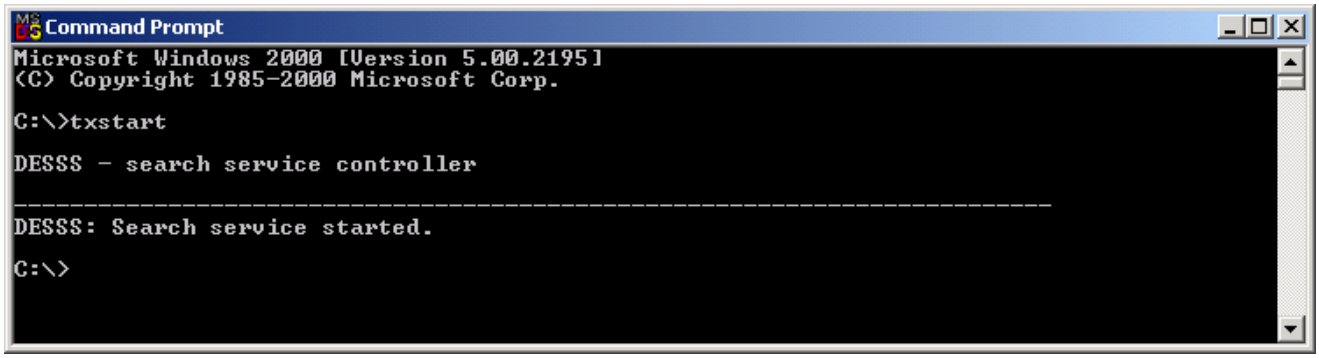
- **Filnamn** anger var DB2 skall skriva meddelanden. Filen måste finnas i förväg (t ex som en tom textfil).

- **Tabell (lista av kolumner)** specificerar vilken tabell och vilka kolumner som de inlästa data skall placeras i. Kolumnernas ordning här mappas mot den kolumnordning som specificerades i METOD-klausulen.

Aktivera databasen för Text Extender

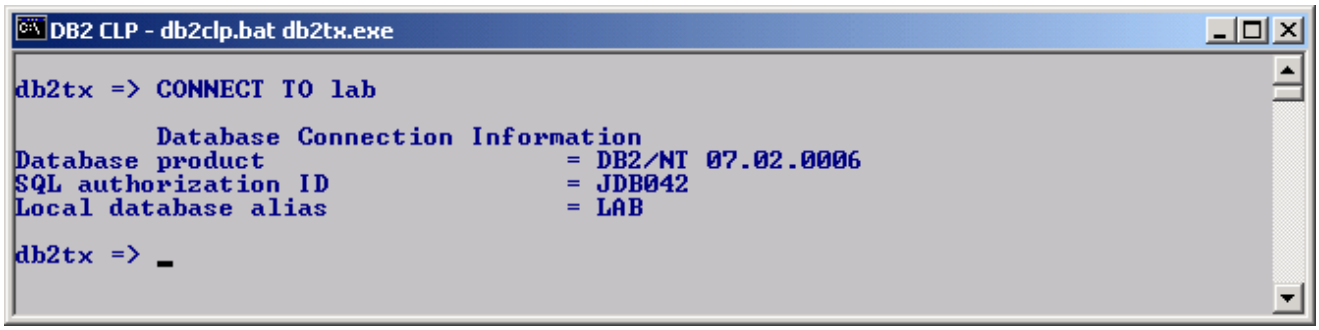
Så långt har man egentligen endast jobbat med standard DB2 funktionalitet. Nu blir det dags att köra igång TextExtender.

8. Öppna en *Command-prompt* fönster och ange kommandot ***txstart***. Detta är en process som måste vara igång för att man skall kunna indexera eller söka i indexerade dokument.

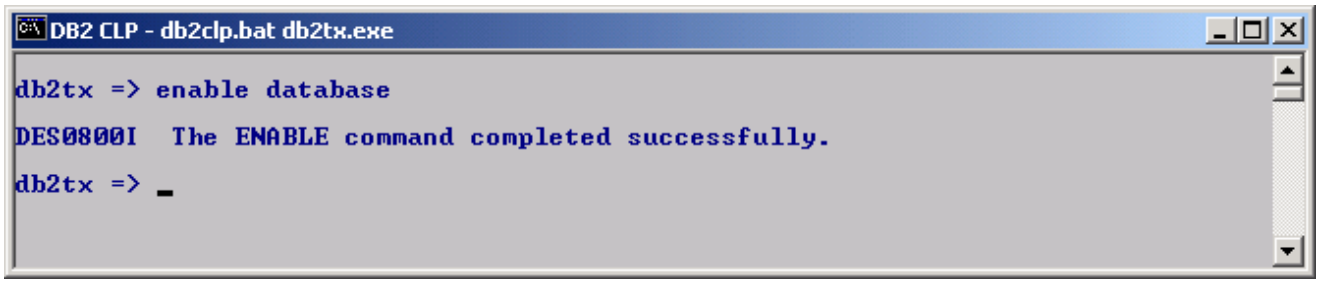


9. Öppna en **DB2TX Command Line Processor** via Start-menyn under DatabaseManagementSystems>IBM-Database-Systems>IBM DB2 Extenders>Text Externder>DB2 Text Extender Command Line Processor.

10. Anslut till databasen med kommandot *CONNECT TO lab.*

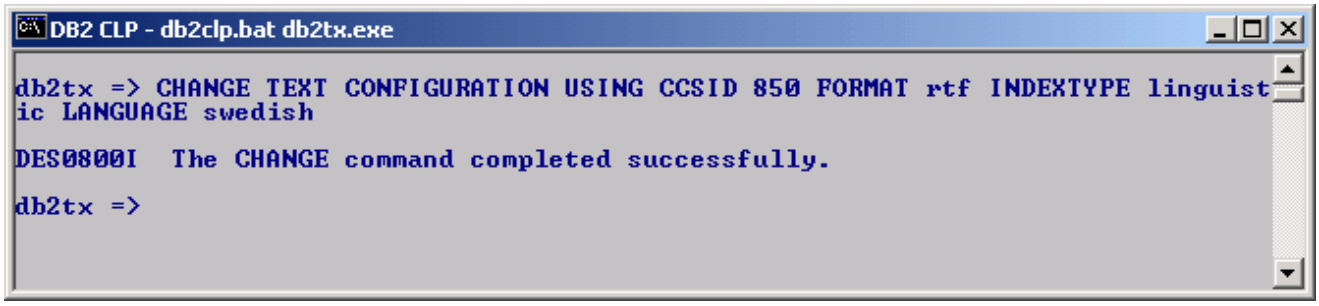


11. Kör kommandot *ENABLE DATABASE* för att aktivera den aktuella databasen för TextExtender. Detta kommando skapar en TextExtender-tabell som heter DB2TX.TextColumns. Denna tabell innehåller information om tabeller och kolumner som är aktiverade för TextExtender.

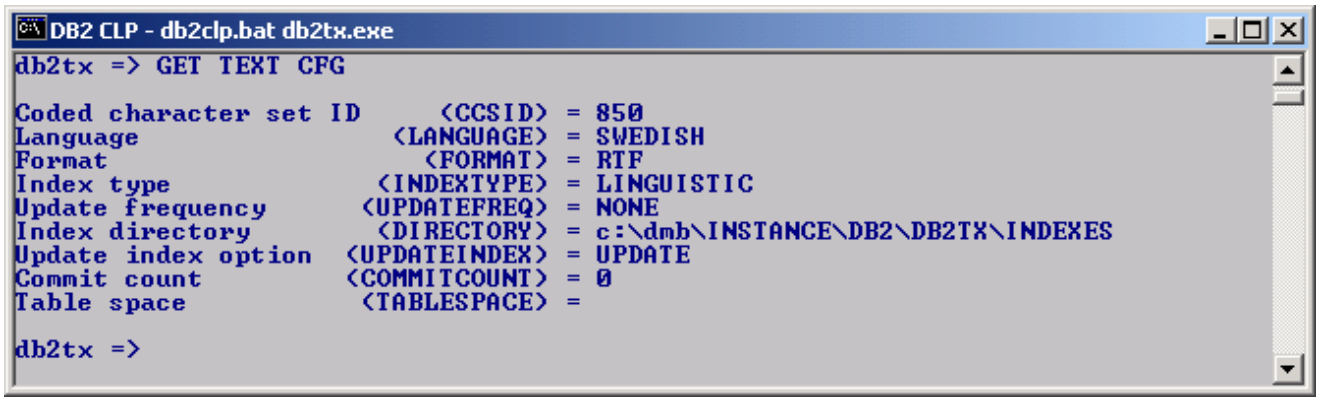


12. Innan vi aktiverar någon kolumn för Text Extender, kan vi ställa in standard inställningar i DB2 så att vi kan minimera behovet av senare ändringar. Vi kan med följande kommando sätta default värden för format, teckenkodning, språk, och indexeringstyp:

CHANGE TEXT CONFIGURATION USING CCSID 850 FORMAT rtf INDEXTYPE linguistic LANGUAGE swedish



Vi antar alltså att dessa värden kommer stämma överens med majoriteten av våra dokument. Vi kan också kontrollera att våra nya inställningar har registrerats med kommandot *GET TEXT CFG*:



Vi valde alltså svenska som default språket, RTF som default filformat och svensk teckenuppsättning. Vi satte också linguistic som default indextyp. Skillnader mellan olika indextyper förklaras senare. Default filformatet har egentligen ingen effekt. DB2 klarar av att känna igen filformatet för varje fil oberoende av vad som är angivet.

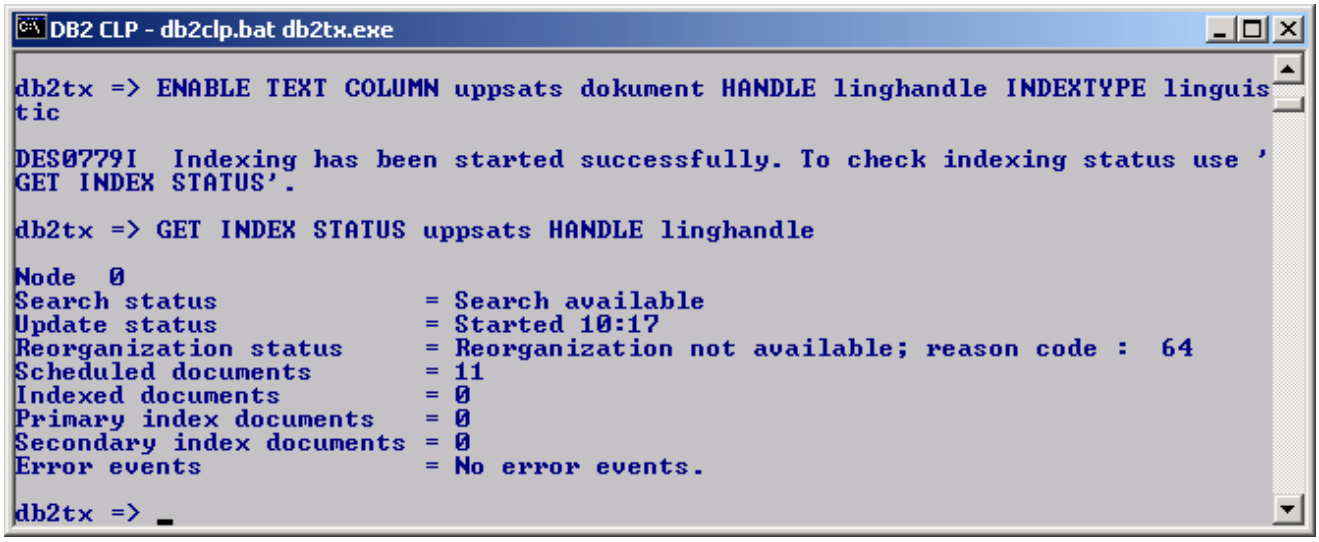
13. Kör kommandot *ENABLE TEXT COLUMN* enligt nedan för att skapa ett lingvistiskt index.

ENABLE TEXT COLUMN uppsats dokument HANDLE linghandle INDEXTYPE linguistic

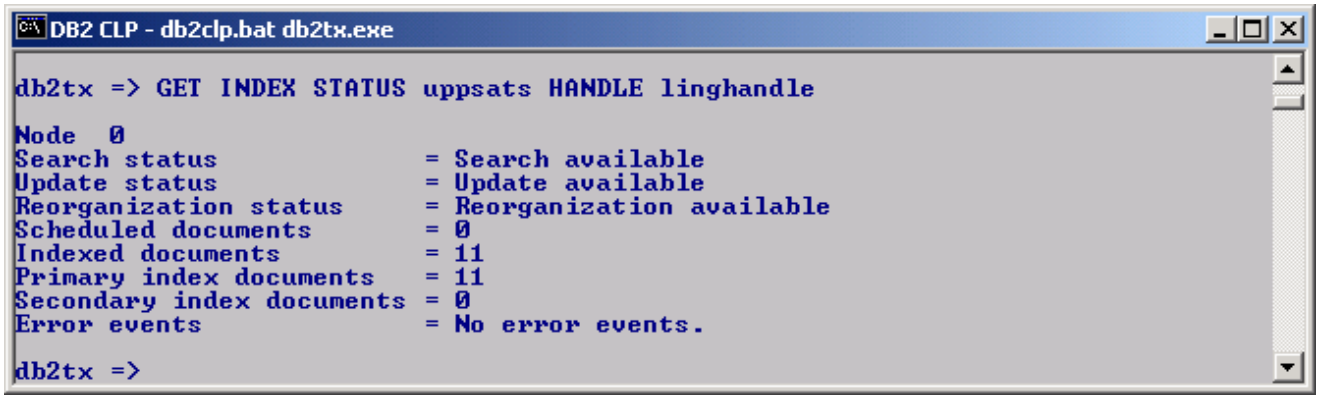
- *uppsats* är namnet på tabellen
- *dokument* är namnet på CLOB kolumnen .
- *linghandle* är namnet på handtaget för det index som skall skapas.
  Handtaget blir då en kolumn i den berörda tabellen. Kolumnen får i det här fallet namnet *linghandle*. Vi kommer att senare se hur man använder handtaget för att komma åt indexet.
- *linguistic* är indextypen som vi vill skall användas för det skapade index. (Observera att man kan utelämna INDEXTYPE-klausulen om man vill använda den indextyp som har definierats som default.)

14. Man kan efter några sekunder köra följande kommando för att se hur långt indexeringen har gått:

GET INDEX STATUS uppsats HANDLE linghandle

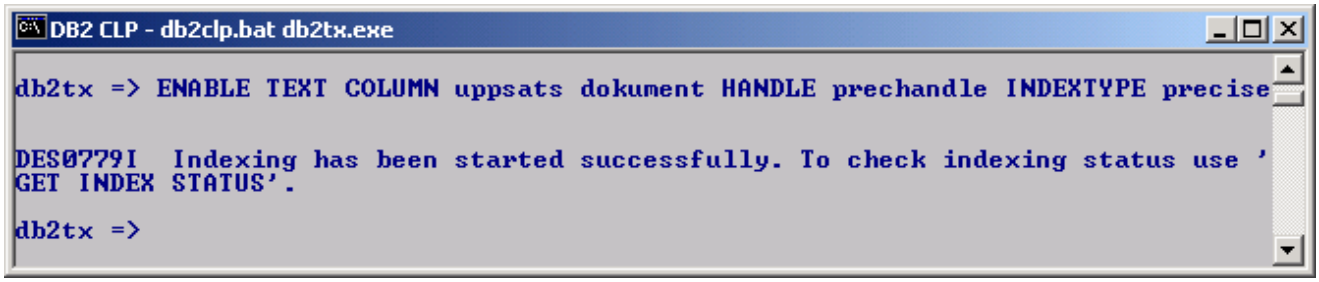


Om indexeringen forfarande pågår kan man vänta lite till och köra kommandot igen. När indexeringen är klar får man följande:



15. Gör nu samma igen för att skapa ett nytt index av indextypen *precise*. Använd följande kommando:

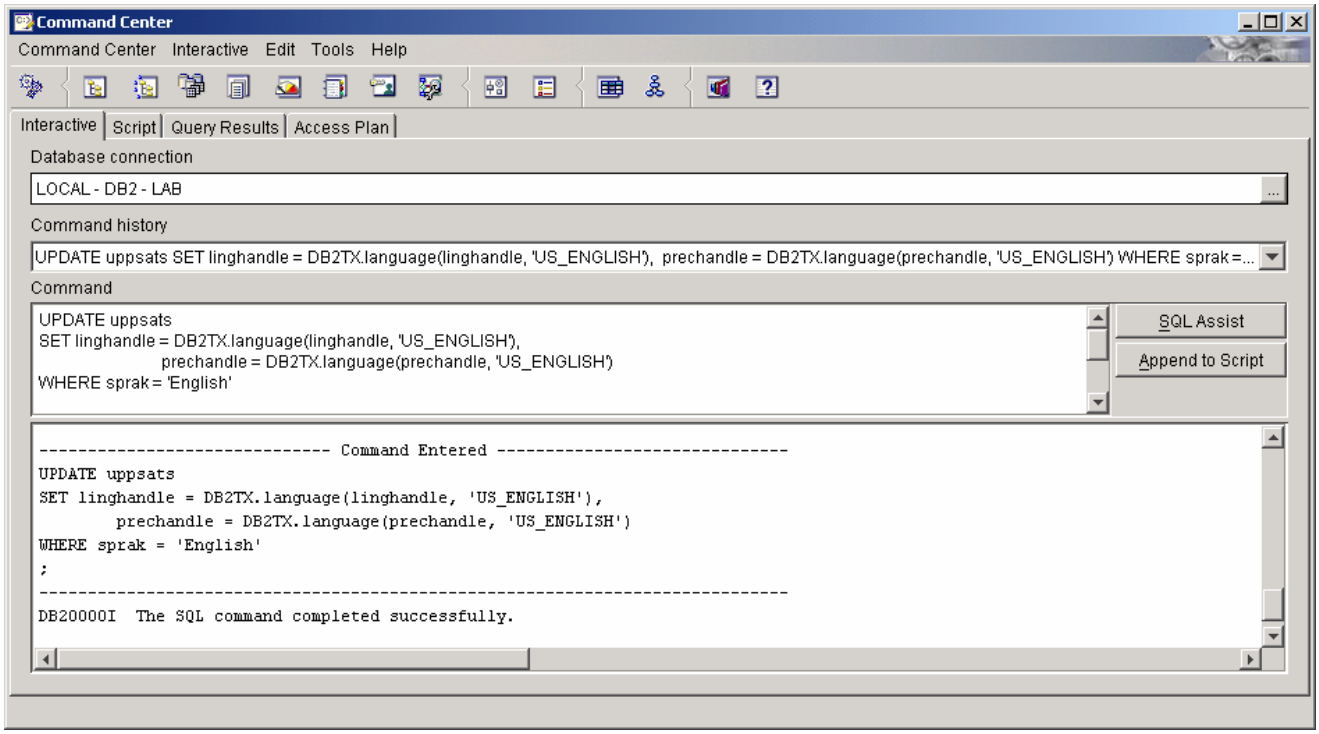
ENABLE TEXT COLUMN uppsats dokument HANDLE prechandle INDEXTYPE precise



16. Eftersom dokumenten inte är av samma språk måste vi nu uppdatera våra handles som nu tror att alla dokument är på svenska (dvs. default språket). Vi kan sätta språket i en handle för varje specifik rad. Vi kan använda följande SQL sats som använder sig av funktionen DB2TX.language(*handle*, *språk*):

UPDATE uppsats
SET linghandle = DB2TX.language(linghandle, 'US_ENGLISH'),

prechandle = DB2TX.language(prechandle, 'US_ENGLISH')
WHERE sprak = 'English'



Alltså, vi har ändrat båda handles på alla rader som innehåller engelska dokument till uppdaterade handles. Funktionen DB2TX.language tar en handle, ändrar språket och returnerar den nya handle.

Ställa frågor mot databasen

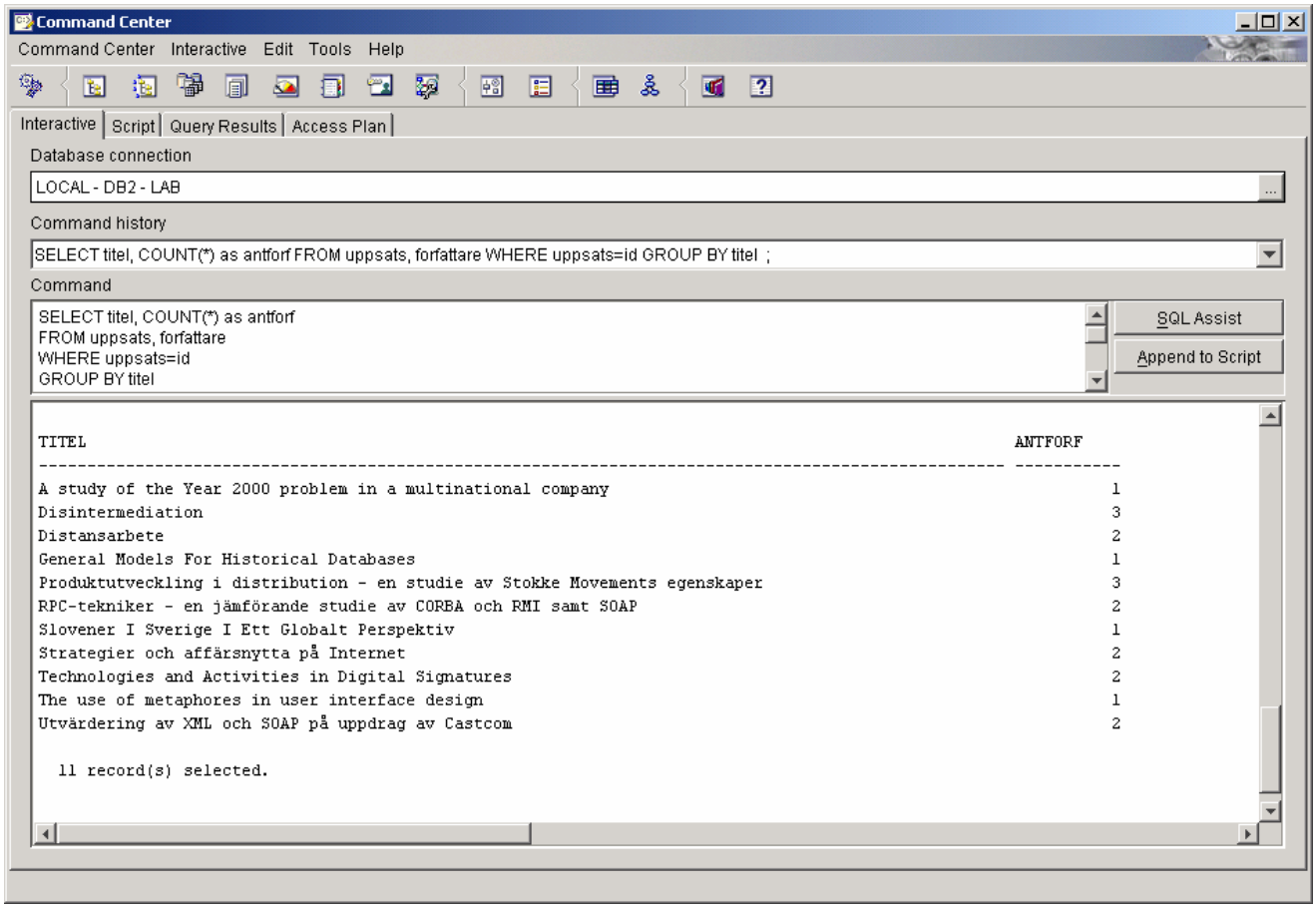
I det här avsnittet kommer vi att ställa frågor mot vår databas. Vi kommer att ställa

- Vanliga SQL frågor
- Frågor om dokumentens meta information
- Frågor om dokumentens textinnehåll (både lingvistiskt och precist)
- Frågor som kombinerar samtliga typer av frågor

Vanliga SQL frågor

1. Vi kan då börja med följande fråga (som illustrerar att vår databas är en helt vanlig databas):
   Visa antal författare per uppsats!

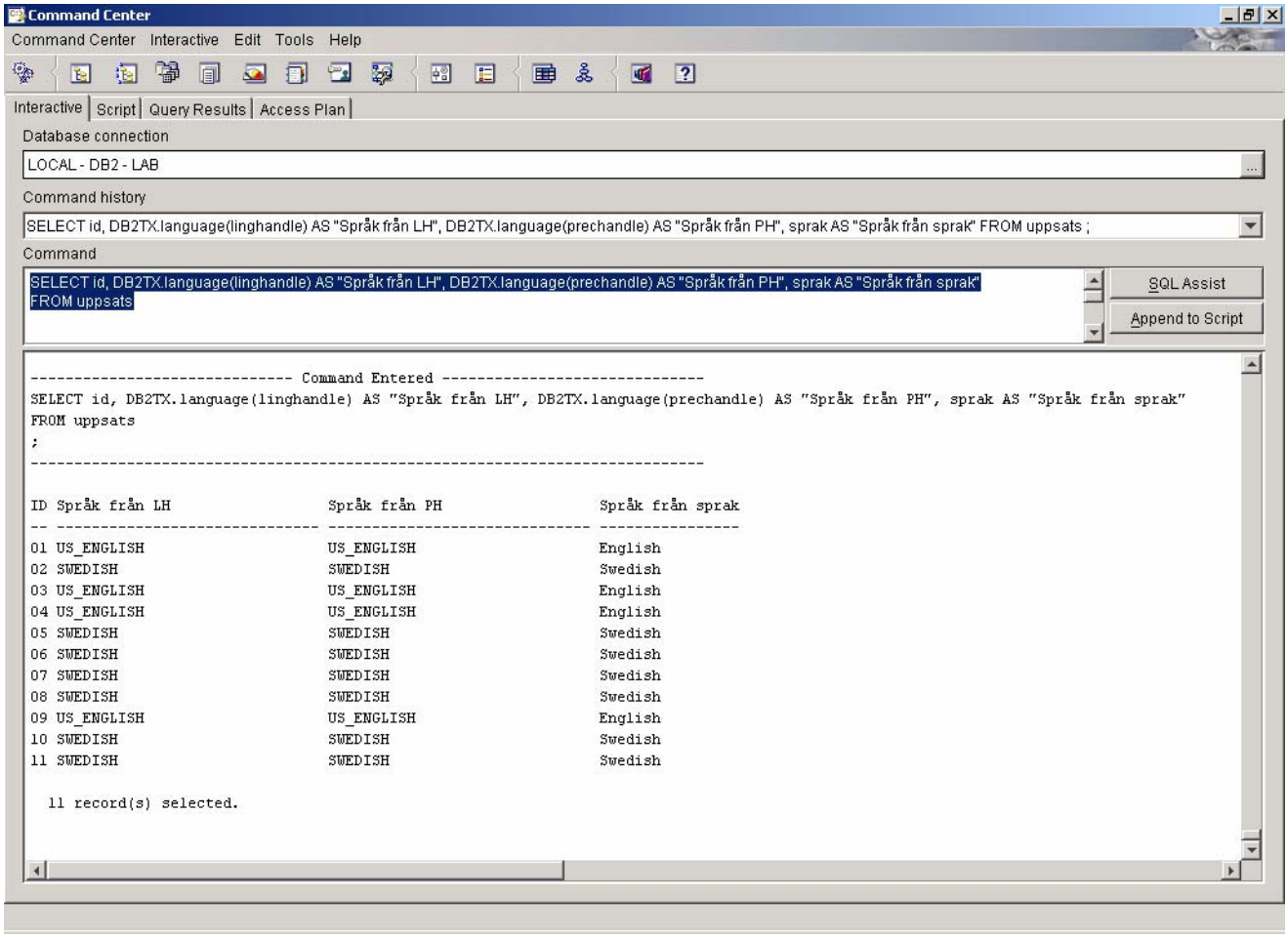
   SELECT titel, COUNT(*) as antforf
   FROM uppsats, forfattare
   WHERE uppsats=id
   GROUP BY titel



Frågor om dokumentens meta-information

1. Visa språket för varje dokument (hämtat från våra handles)! Här kan vi använda oss av funktionen DB2TX.language, som vi använde tidigare. Vi kan också hämta ut språket från kolumnen sprak för att kunna se om de stämmer överens.

   SELECT id, DB2TX.language(linghandle) AS "Språk från LH", DB2TX.language(prechandle) AS "Språk från PH", sprak AS "Språk från sprak"
   FROM uppsats



Frågor om dokumentens textinnehåll (både lingvistiskt och precist)

Det finns ett antal funktioner som kan användas för att ställa frågor mot CLOB-dokuments textinnehåll. Dessa visas i tabellen nedan.

Funktioner (UDF)	Syfte
CONTAINS	Evaluerar ett villkor för varje dokument. Returnerar 0 eller 1.
NO_OF_MATCHES	Returnerar antalet träffar för ett visst villkor för varje dokument.
RANK	Returnerar ett rangordningsvärde per dokument givet ett villkor. Det returnerade värdet ligger mellan 0 och 1, där 1 är bäst.

Villkoren kan bestå av en del olika nyckelord. Dessa fungerar olika på de olika typer av index. Följande tabell sammanfattar de nyckelord vi kommer att använda:

	Indextyp	
Nyckelord	***Linguistic***	***Precise***
PRECISE FORM OF	Icke-tillgänglig	Default
STEMMED FORM OF	Default	Icke-tillgänglig
SYNONYM FORM OF	Tillgänglig	Tillgänglig
SOUNDS LIKE	Tillgänglig	Tillgänglig
IN SAME SENTENCE AS	Tillgänglig	Tillgänglig
IN SAME PARAGRAPH AS	Tillgänglig	Tillgänglig

De första fyra nyckelorden kan följas av språk. Om språket utelämnas används defaultspråket. Det är rekommenderat att man alltid anger språket, för att undvika förvirring. Giltiga värden för språk innefattar *SWEDISH* och *US_ENGLISH*. (Obs! Nyckelorden för språken är case-sensitive!)

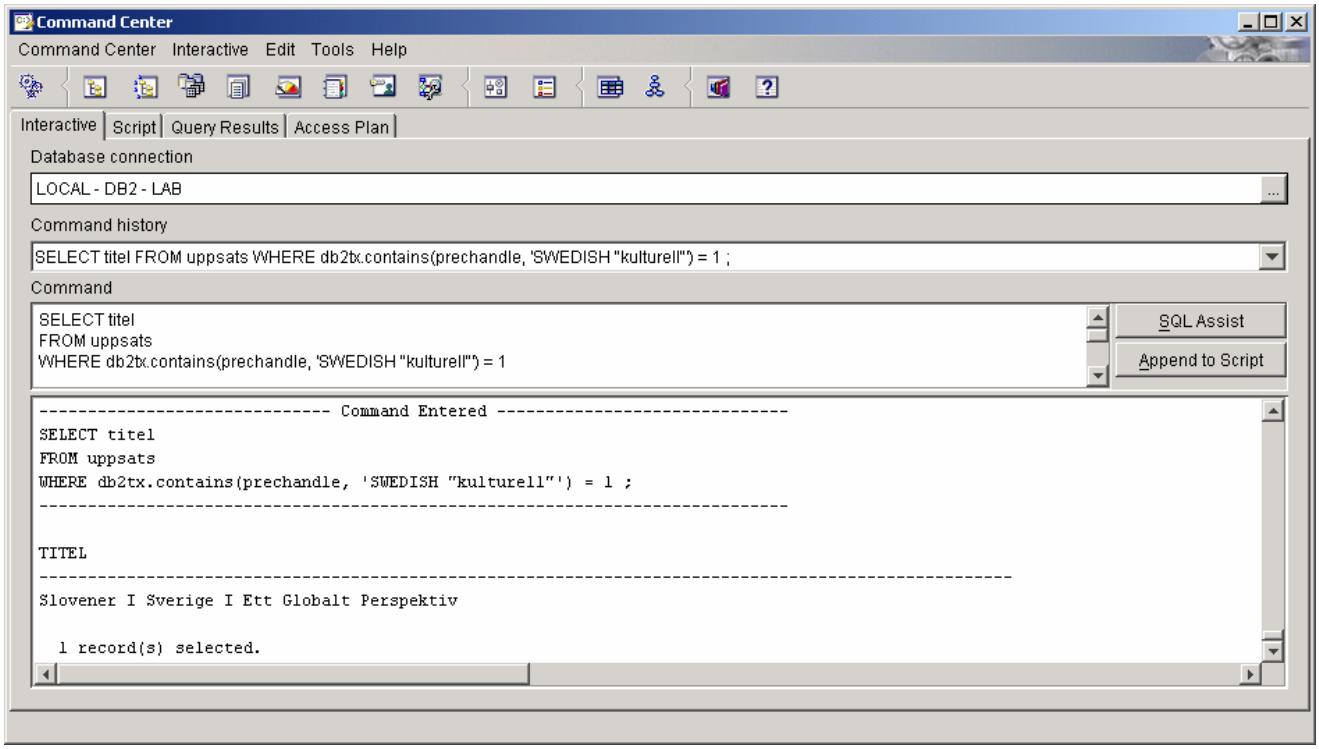
Villkoren kan förstås innehålla även logiska operatorer som *NOT*, *AND (&)* och *OR (|)*. Det finns även möjlighet att använda wild-cards som % och _.

För att exemplifiera de olika funktioners användning kommer vi nu att titta på ett antal exempel.

1. Vilka dokument innehåller det svenska ordet kulturell? I det här fallet vill vi få fram titeln på de dokument som innehåller exakt ordet kulturell.

   Vi kan få svar på ovanstående med följande SQL-sats:

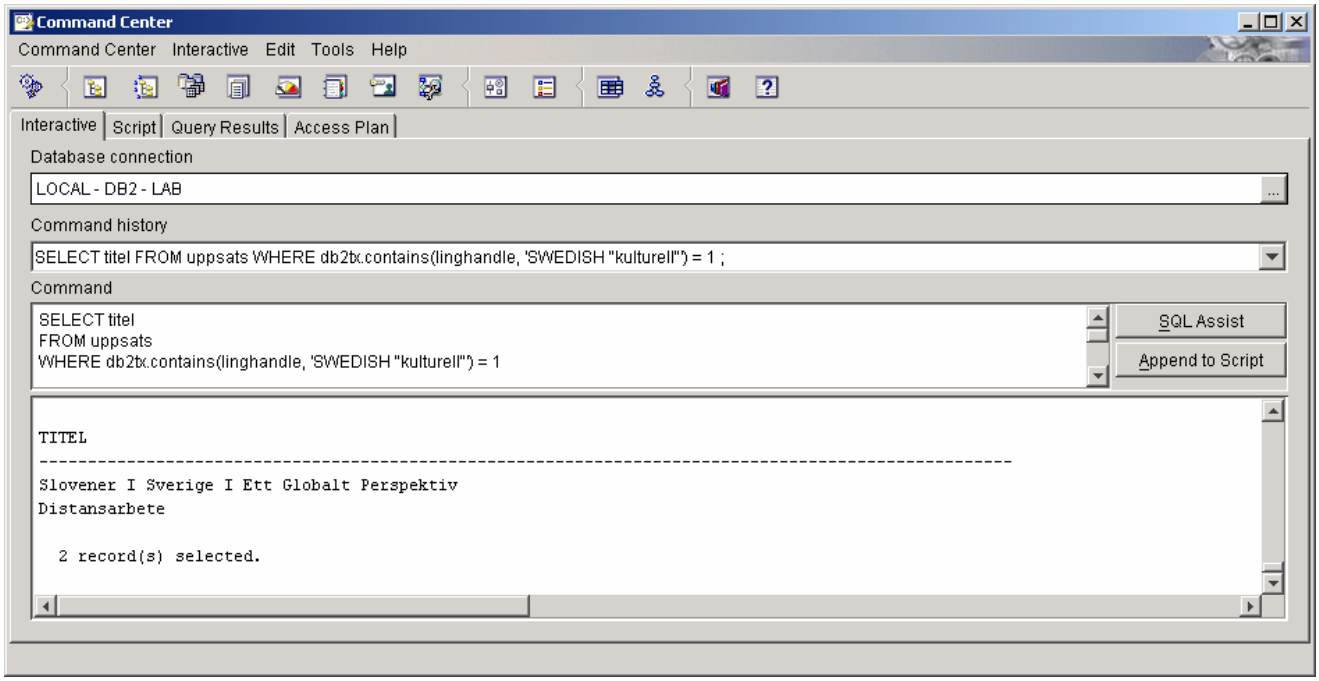
   ```
   SELECT titel
   FROM uppsats
   WHERE db2tx.contains(prechandle, 'SWEDISH "kulturell"') = 1
   ```



Genom att vi använde prechandle för vår sökning så har vi implicit specificerat att vi vill använda nyckelordet PRECISE FORM OF.

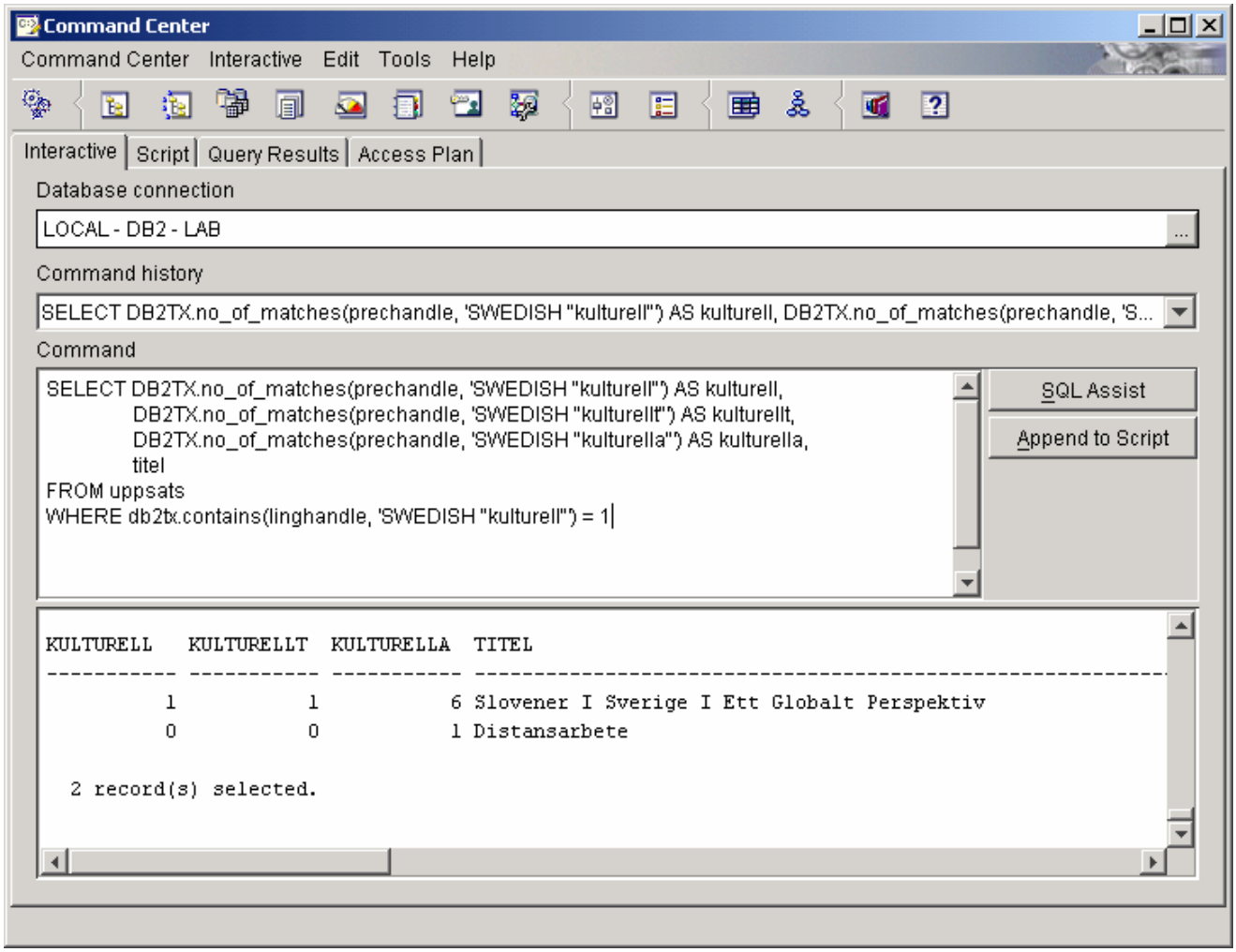
2. Om vi nu istället vill ha fram alla dokument som innehåller någon böjning av ordet kulturell kan vi istället använda oss av vår linghandle:

   SELECT titel
   FROM uppsats
   WHERE db2tx.contains(linghandle, 'SWEDISH "kulturell"') = 1



3. Vi kan då modifiera frågan så att vi även kan se vilka böjningar av ordet som finns i varje dokument! Vi kan inkludera i SELECT-klausulen en kolumn för varje böjning och genom att använda funktionen DB2TX.no_of_matches se antal förekomster i varje dokument.

```
SELECT DB2TX.no_of_matches(prechandle, 'SWEDISH "kulturell"') AS kulturell,
       DB2TX.no_of_matches(prechandle, 'SWEDISH "kulturellt"') AS kulturellt,
       DB2TX.no_of_matches(prechandle, 'SWEDISH "kulturella"') AS kulturella,
       titel
FROM uppsats
WHERE db2tx.contains(linghandle, 'SWEDISH "kulturell"') = 1
```

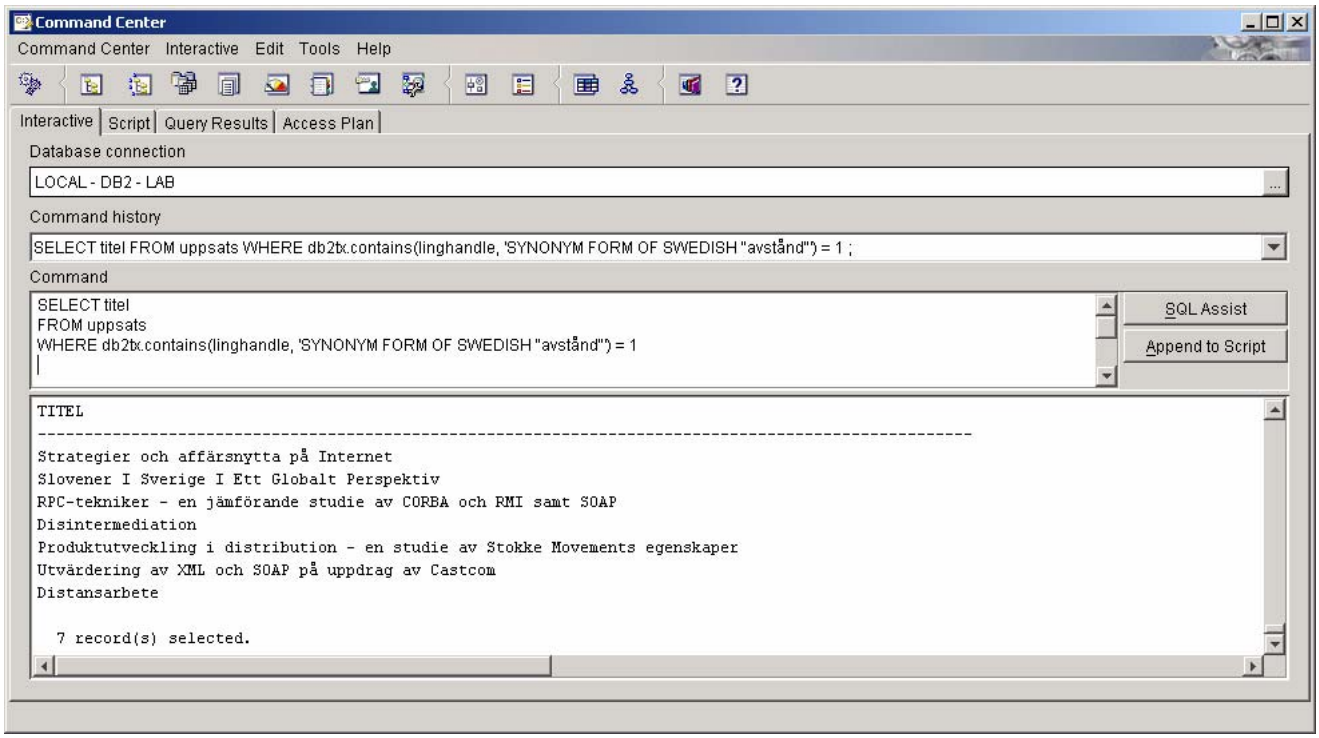


Så vi kan se att uppsatsen "Distansarbete" innehåller den böjda formen *kulturella* 1 gång, vilket gör att den klarar av det här villkoret, men inte villkoret om den precisa formen *kulturell*.

Man kan ibland vilja kolla efter en företeelse och inte ett visst ord. Då kan man ha nytta av synonymer:

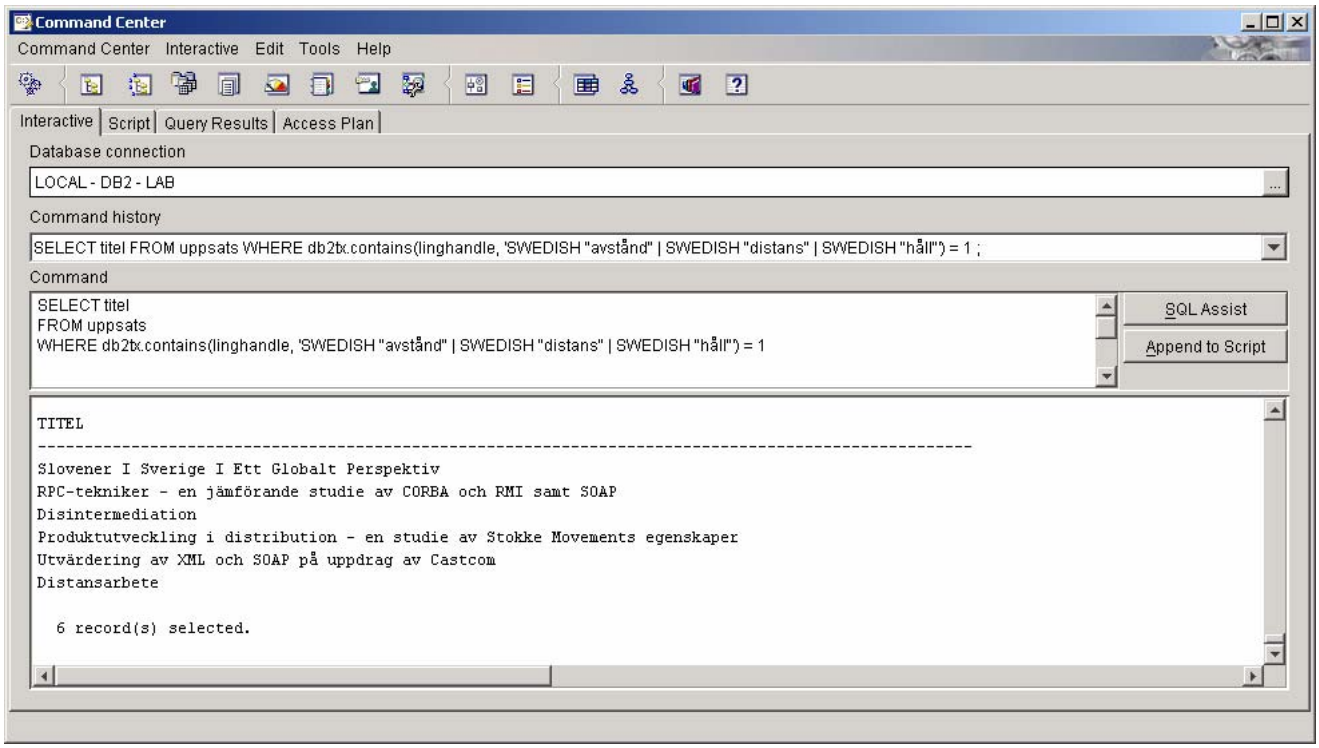
4. Visa alla dokument som omnämner avstånd i någon form.

```
SELECT titel
FROM uppsats
WHERE db2tx.contains(linghandle, 'SYNONYM FORM OF SWEDISH "avstånd"') = 1
```



5. Om vi nu vill kontrollera endast vissa synonymer som t ex distans, avstånd och håll men inte resten kan det vara fördelaktigt att använda logiska operatorn OR (|):

SELECT titel
FROM uppsats
WHERE db2tx.contains(linghandle, 'SWEDISH "avstånd" | SWEDISH "distans" | SWEDISH "håll"') = 1



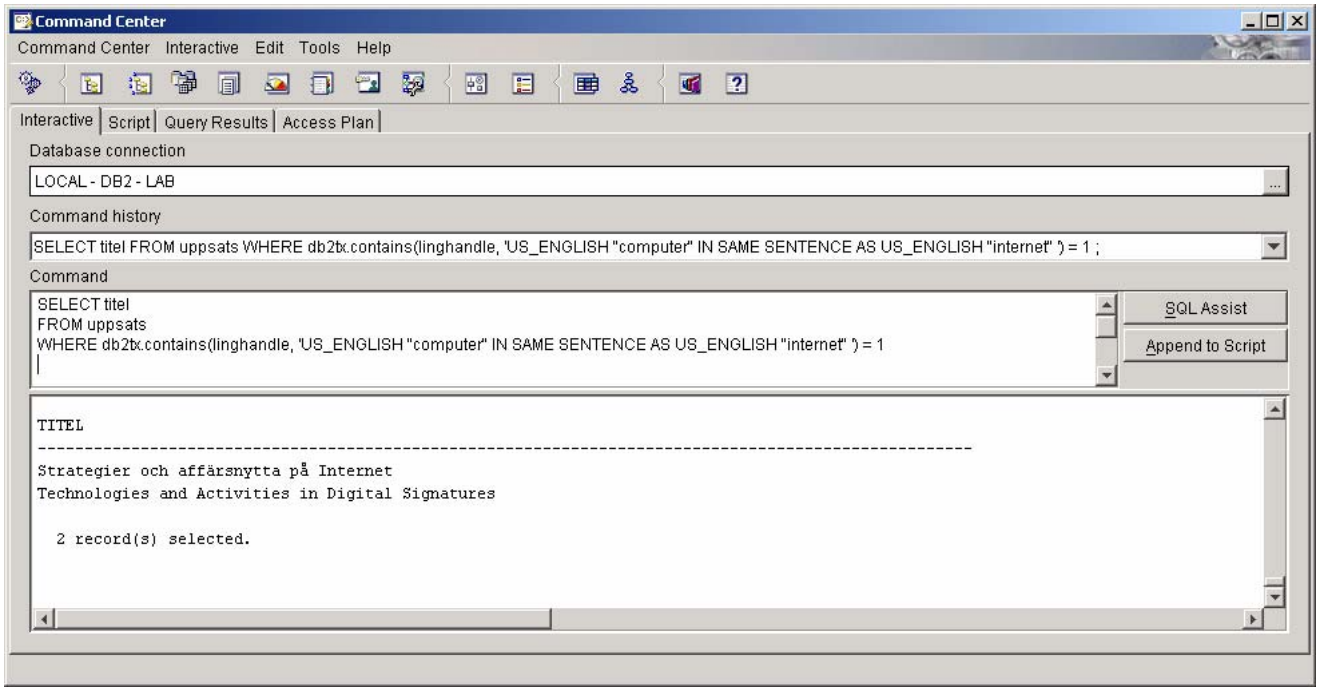
Man kan då observera att uppsatsen "Strategier och affärsnytta på Internet" har bortfallit då den inte innehåller någon av våra tre synonymer (Den innehåller antagligen en annan synonym till *avstånd*).

En annan möjlighet som finns är att kunna testa om två eller flera subvillkor tillfredsställs inom samma mening eller stycke:

6. Visa vilka uppsatser som innehåller engelska orden *computer* och *internet* i samma mening!

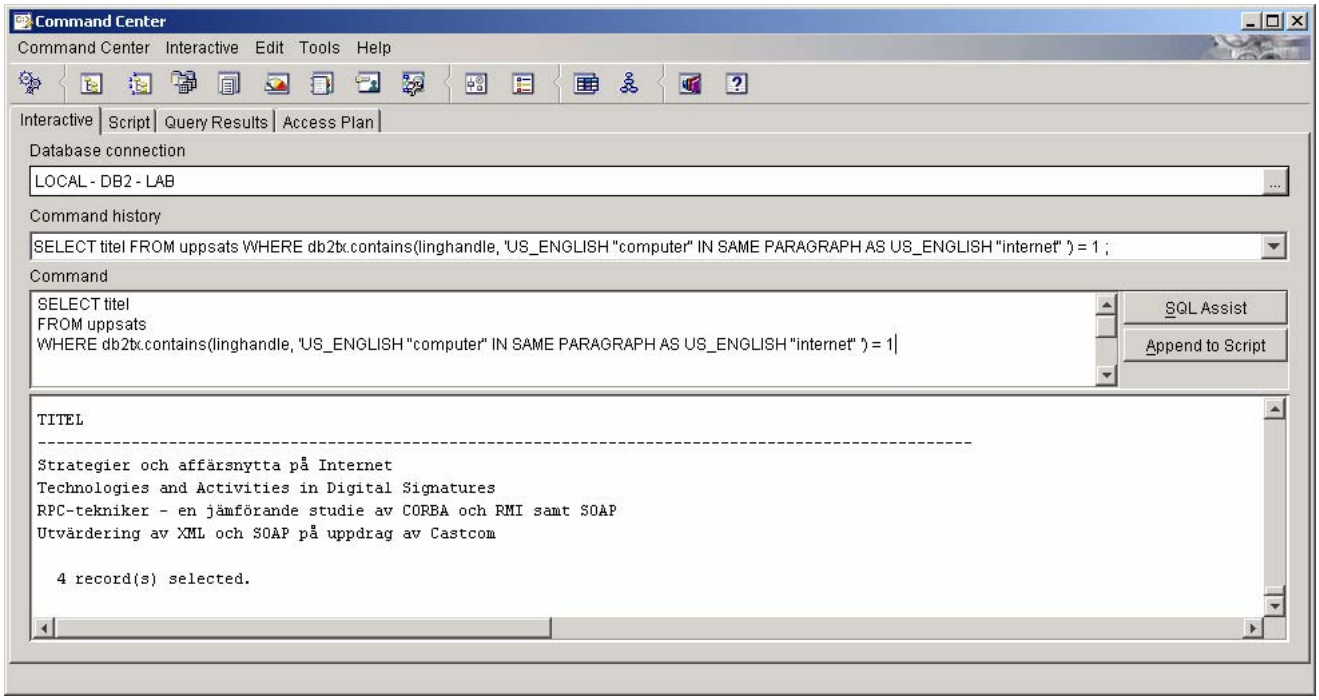
   Det kan göras med följande SQL sats:

   SELECT titel
   FROM uppsats
   WHERE db2tx.contains(linghandle, 'US_ENGLISH "computer" IN SAME SENTENCE AS US_ENGLISH "internet" ') = 1



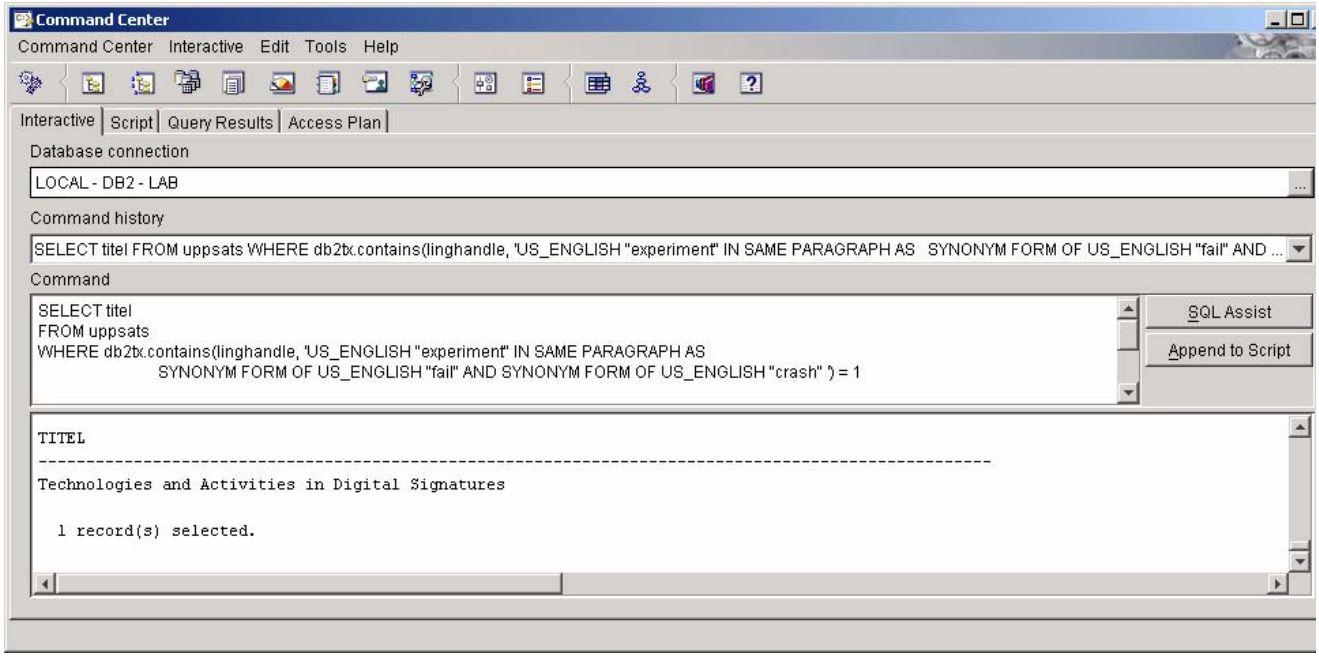
7. Visa nu vilka uppsatser som innehåller engelska orden *computer* och *internet* i samma stycke istället! (För att verkligen se att det blir skillnad!)

   SELECT titel
   FROM uppsats
   WHERE db2tx.contains(linghandle, 'US_ENGLISH "computer" IN SAME PARAGRAPH AS US_ENGLISH "internet" ') = 1



8. Man kan också kontrollera följande: Vilka uppsatser har i samma stycke ordet *experiment* (eller böjning av) och synonymer för orden *fail* och *crash.*

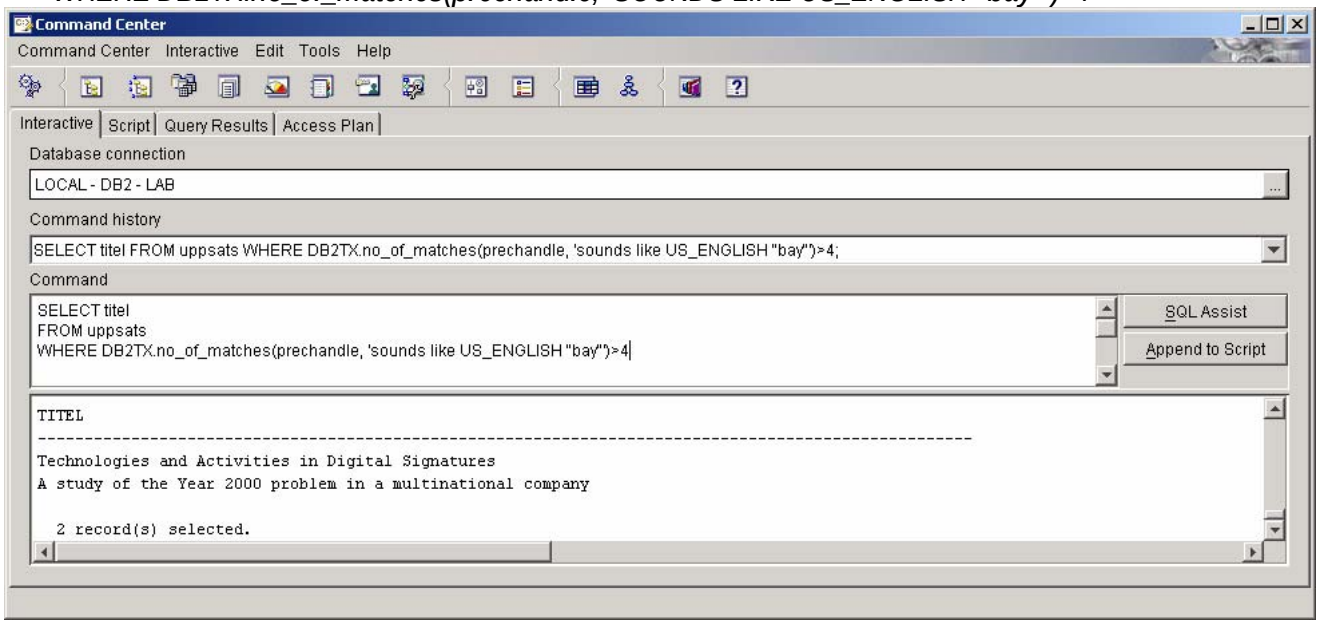
   SELECT titel
   FROM uppsats
   WHERE db2tx.contains(linghandle, 'US_ENGLISH "experiment" IN SAME PARAGRAPH AS SYNONYM FORM OF US_ENGLISH "fail" AND SYNONYM FORM OF US_ENGLISH "crash"') = 1



Skulle man vilja leta efter ett ord som man inte riktigt vet hur det stavas, kan man ha nytta av nyckelordet *SOUNDS LIKE*:

9. Hitta alla dokument som innehåller minst 5 förekomster av ord som låter som engelska ordet *bay*!

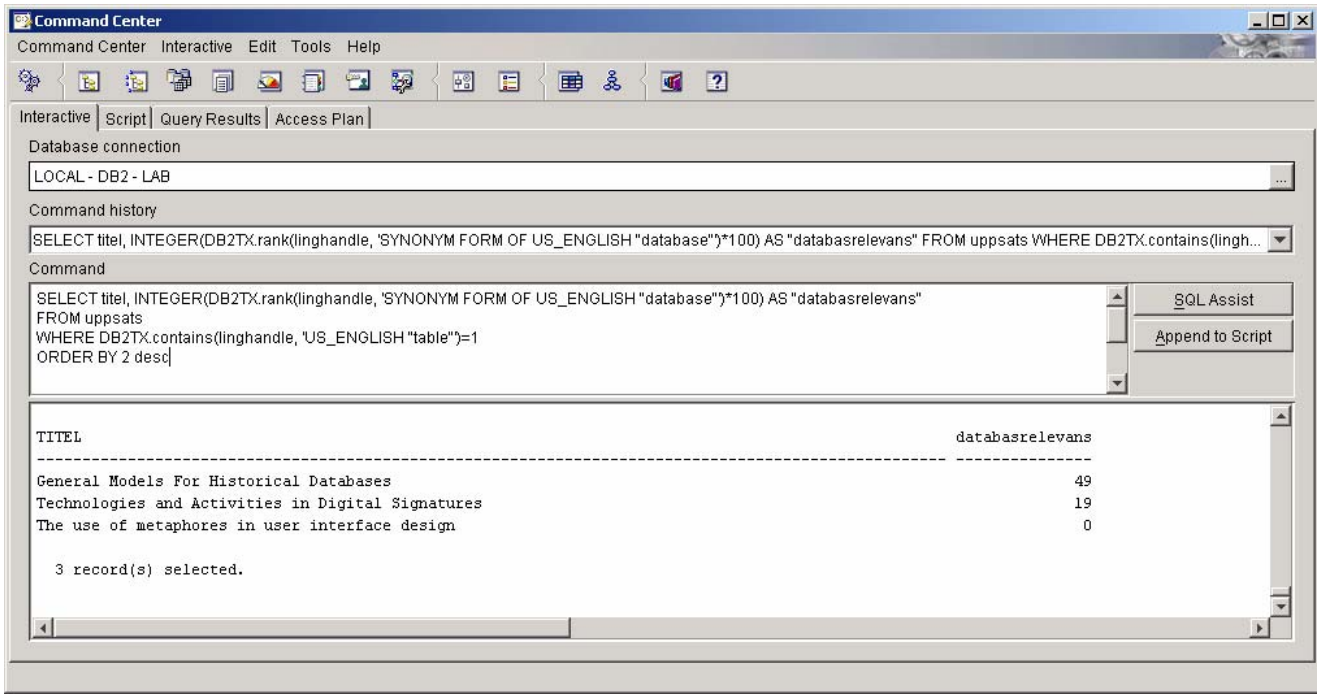
```
SELECT titel
FROM uppsats
WHERE DB2TX.no_of_matches(prechandle, 'SOUNDS LIKE US_ENGLISH "bay"')>4
```



Man kan använda sig av funktionen DB2TX.rank för att få databashanteraren att beräkna ett rangordningsvärde per dokument givet ett villkor:

10. Ta fram alla dokument som innehåller engelska ordet *table* och rangordna dem efter relevans till ordet *database*!

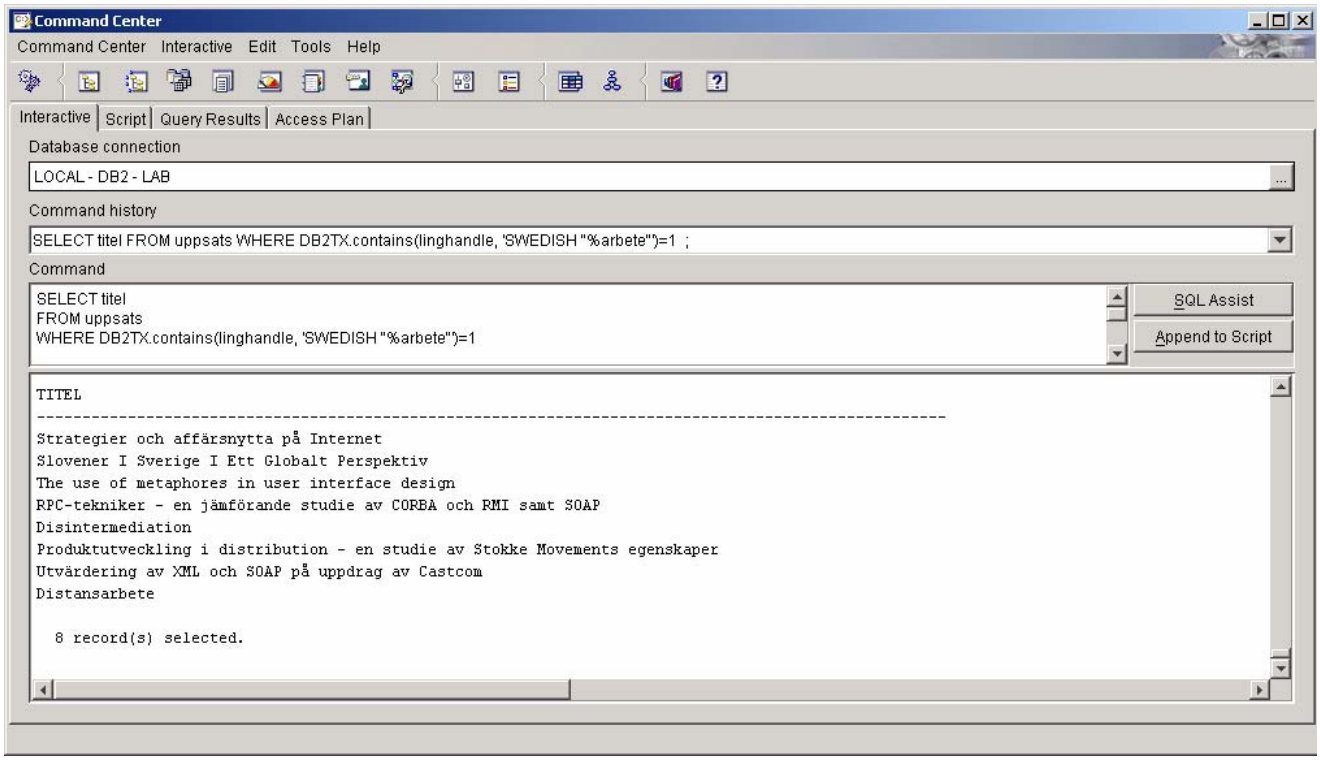
```
SELECT titel, INTEGER(DB2TX.rank(linghandle, 'SYNONYM FORM OF US_ENGLISH
"database"')*100) AS "databasrelevans"
FROM uppsats
WHERE DB2TX.contains(linghandle, 'US_ENGLISH "table"')=1
ORDER BY 2 desc
```



Sist, men inte minst kan vi titta på ett exempel på användning av wild-cards.

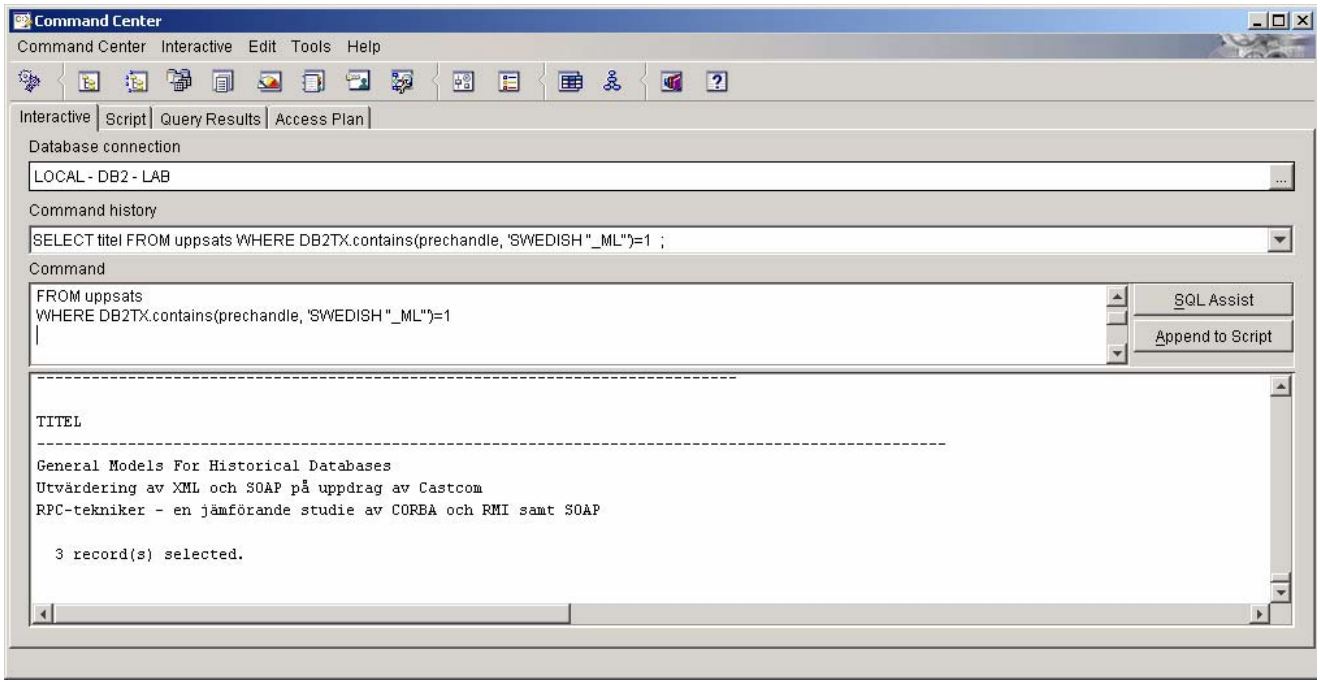
11. Visa alla dokument som innehåller något ord som slutar med *arbete*!

```
SELECT titel
FROM uppsats
WHERE DB2TX.contains(linghandle, 'SWEDISH "%arbete"')=1
```



12. Visa vilka dokument som innehåller en förkortning på tre tecken som slutar med ML!

```
FROM uppsats
WHERE DB2TX.contains(prechandle, 'SWEDISH "_ML"')=1
```

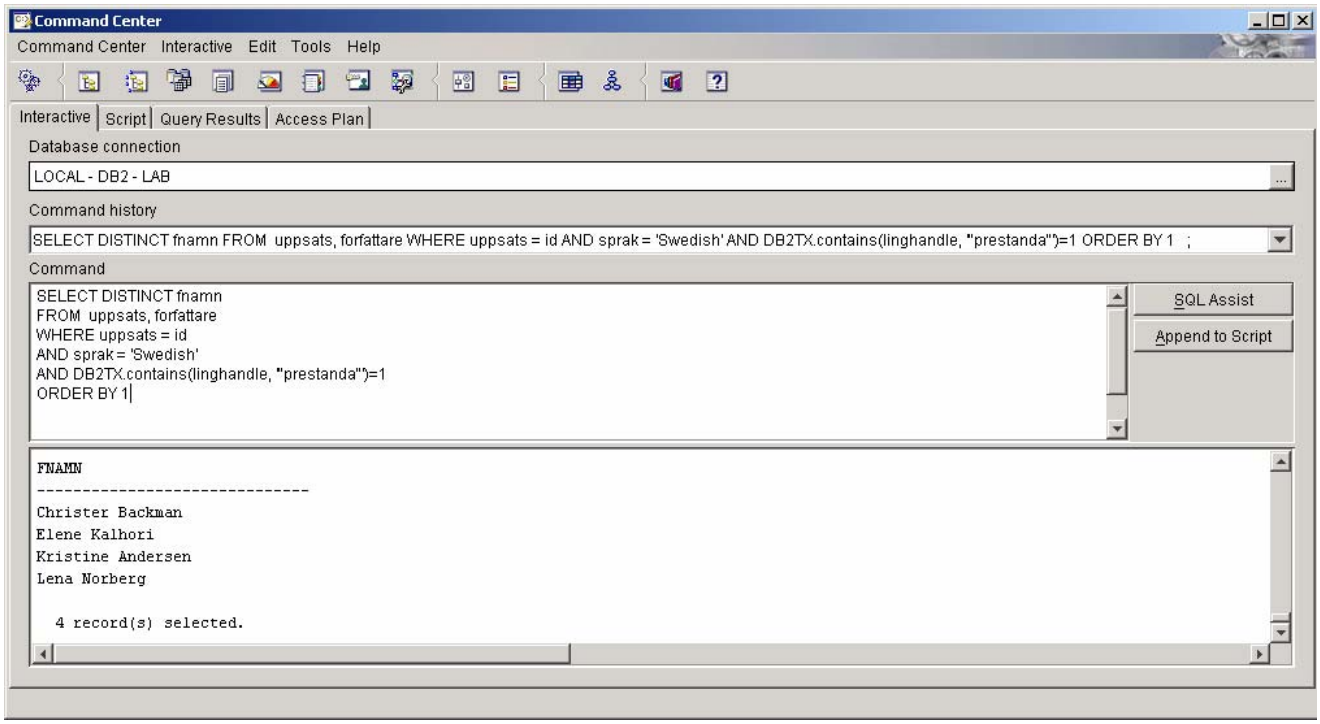


(Två av dem innehåller XML och en DML!)

Frågor som kombinerar samtliga typer av frågor

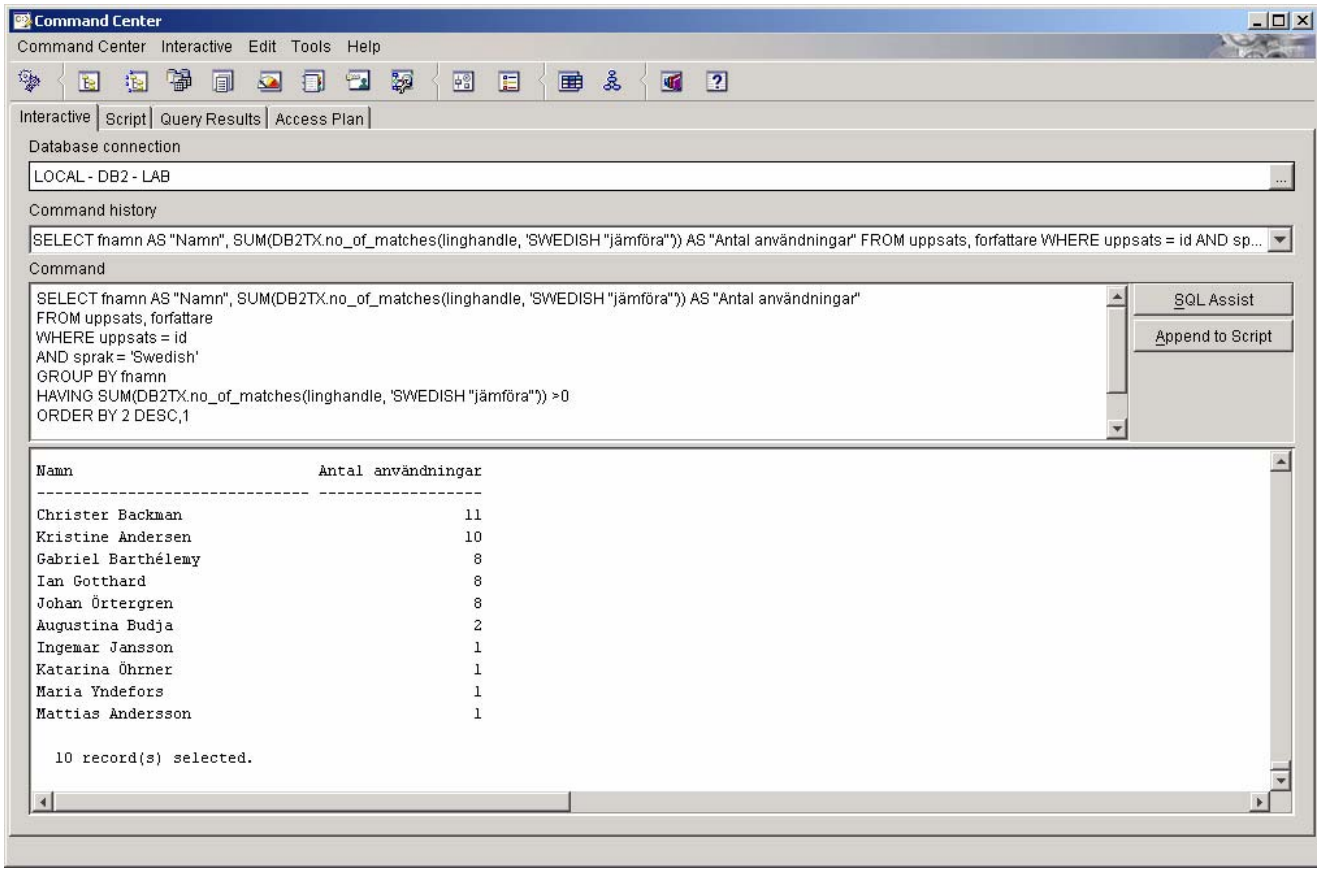
1. Vilka författare har skrivit uppsats på svenska som innehåller ordet *prestanda*? Sortera på namn!

```
SELECT DISTINCT fnamn
FROM  uppsats, forfattare
WHERE uppsats = id
AND sprak = 'Swedish'
AND DB2TX.contains(linghandle, '"prestanda"')=1
ORDER BY 1
```



2. Visa för varje författare namn och antal gånger denna har använt ordet *jämföra* (i någon form) i svenska uppsatser! Författare som inte har använt ordet skall inte visas i resultatet. Den som har använt ordet mest skall komma först. Om flera har använt ordet lika många gånger skall de sorteras på namn.

   SELECT fnamn AS "Namn", SUM(DB2TX.no_of_matches(linghandle, 'SWEDISH "jämföra"')) AS "Antal användningar"
   FROM uppsats, forfattare
   WHERE uppsats = id
   AND sprak = 'Swedish'
   GROUP BY fnamn
   HAVING SUM(DB2TX.no_of_matches(linghandle, 'SWEDISH "jämföra"')) >0
   ORDER BY 2 DESC,1

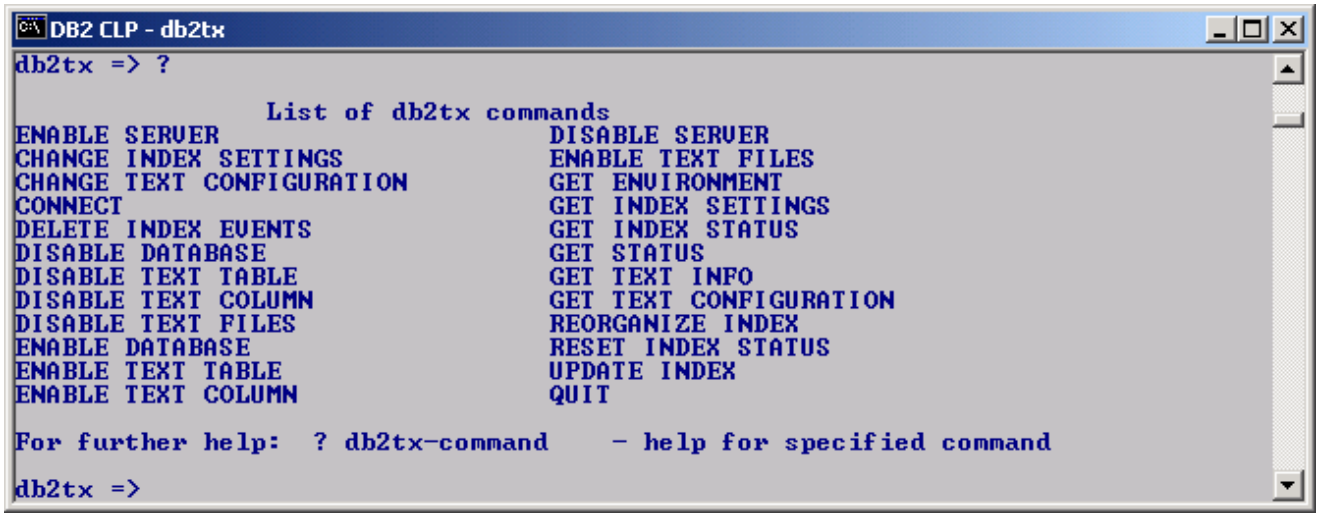


När fel har uppstått

Här hittar man en lista med de kommandon man kan använda för att återställa databasen när fel uppstår. Följande tabell visar vilka kommandon som kan användas för att återställe ett annat kommando (INTE SQL kommandon!).

Använd det här kommandot	För att göra motsatsen till det här kommandot
TXSTOP	TXSTART
QUIT och sedan DB2TX	CONNECT TO *databas*
DISABLE DATABASE …	ENABLE DATABASE …
DISABLE TEXT COLUMN …	ENABLE TEXT COLUMN …

För mer information kan man ange ? i **DB2TX Command Line Processor:**



Uppgifter

Följande uppgifter löses och sedan antingen redovisas vid dator eller SQL frågorna och exekveringsresultat lämnas in.

1. Vilka dokument (titeln på uppsatserna) innehåller orden *Internet* eller *hårddisk* och inte orden *intranet* eller *databas*?

2. Ta fram titeln på de uppsatser som innehåller en exakt överrensstämmelse med ordet *dator* (d.v.s. datorer, datorn etc räknas inte) och vars titel innehåller ordet *och*.

3. Vilka svenska dokument (titlar) är skrivna efter 1999 och innehåller ordet *term*? Visa bara de uppsatser som har skrivits av minst 2 författare!

4. Visa alla dokument som innehåller minst 3 förekomster av synonymer till ordet *population* och som har ordet (synonymen) i samma stycke som ordet *kunskap*

5. Vilka dokument är minst lika relevanta till ordet *teknik* (eller böjningar) som medelvärdet för de svenska dokument som innehåller exakt ordet *protokoll* och exakt frasen *höga krav*? Visa titeln, året och antal författare! Alltså: Om svenska dokument som innehåller *protokoll* och *höga krav* har en relevans till ordet *teknik* på *X*, så skall resultatet bestå av dokument som har en relevans till ordet *teknik* som är högre än *X*.

Referensmaterial

Specifying search arguments

Search arguments are used in CONTAINS, NO_OF_MATCHES and RANK. This section uses the CONTAINS function to show different examples of search arguments in UDFs.

Searching for several terms

You can have more than one term in a search argument. One way to combine several search terms is to connect them together using commas, like this:

```
SELECT DATE, SUBJECT
   FROM DB2TX.SAMPLE
   WHERE DB2TX.CONTAINS (COMMENTHANDLE,
    '("compress", "compiler", "pack", "zip", "compact")') = 1
```

This form of search argument finds text that contains any of the search terms. In logical terms, the search terms are connected by an OR operator.

Searching with the Boolean operators

Search terms can be combined with other search terms using the Boolean operators "&" (AND) and "|" (OR). For example:

```
SELECT DATE, SUBJECT
  FROM DB2TX.SAMPLE
  WHERE DB2TX.CONTAINS (COMMENTHANDLE,
              '"compress" | "compiler"') = 1
```

You can combine several terms using Boolean operators:

```
SELECT DATE, SUBJECT
  FROM DB2TX.SAMPLE
  WHERE DB2TX.CONTAINS (COMMENTHANDLE,
              '"compress" | "compiler" & "DB2"') = 1
```

If you use more than one Boolean operator, Text Extender evaluates them from left to right, but the logical AND operator (&) binds stronger than the logical OR operator (|). For example, if you do not include parentheses,

```
   "DB2" & "compiler" | "support" & "compress"
```

is evaluated as:

```
   ("DB2" & "compiler") | ("support" & "compress")
```

So in the following example you must include the parentheses:

```
   "DB2" & ("compiler" | "support") & "compress"
```

If you combine Boolean operators with search terms chained together using the comma separator, like this:

```
   ("compress", "compiler") & "DB2"
```

the comma is interpreted as a Boolean OR operator, like this:

```
   ("compress" | "compiler") & "DB2"
```

Searching for variations of a term

If you are using a **precise** index, Text Extender searches for the terms exactly as you type them. For example, the term media finds only text that contains "media". Text that contains the singular "medium" is not found.

If you are using a **linguistic** index, Text Extender searches also for variations of the terms, such as the plural of a noun, or a different tense of a verb.
For example, the term drive finds text that contains "drive", "drives", "driving", "drove", and "driven.".

```
SELECT DATE, SUBJECT
FROM DB2TX.SAMPLE
    WHERE DB2TX.CONTAINS (COMMENTHANDLE,
            'PRECISE FORM OF "utility"') = 1
```

By contrast, this example finds occurrences of "utility" and "utilities":

```
SELECT DATE, SUBJECT
FROM DB2TX.SAMPLE
    WHERE DB2TX.CONTAINS (COMMENTHANDLE,
            'STEMMED FORM OF "utility"') = 1
```

Searching for parts of a term (character masking)

Masking characters, otherwise known as "wildcard" characters, offer a way to make a search more flexible. They represent optional characters at the front, middle, or end of a search term. They increase the number of text documents found by a search.

Tip
If you use masking characters, you cannot use the SYNONYM FORM OF keyword.

Masking characters are particularly useful for finding variations of terms if you have a precise index. If you have a linguistic index, many of the variations found by using masking characters would be found anyway.

Note that word fragments (words masked by wildcard characters) cannot be reduced to a base form. So, if you search for passe%, you will not find the words "passes" or "passed", because they are reduced to their base form "pass" in the index. To find them, you must search for pass%.
Text Extender uses two masking characters: underscore (_) and percent (%):

- % represents **any number of arbitrary characters**. Here is an example of % used as a masking character at the front of a search term:

```
SELECT DATE, SUBJECT
FROM DB2TX.SAMPLE
  WHERE DB2TX.CONTAINS (COMMENTHANDLE, '"%name"') = 1
```

This search term finds text documents containing, for example, "username", "filename", and "table-name". % can also represent a **whole word**: The following example finds text documents containing phrases such as "graphic function" and "query function".

```
SELECT DATE, SUBJECT
FROM DB2TX.SAMPLE
    WHERE DB2TX.CONTAINS (COMMENTHANDLE, '"% function"') = 1
```

- _ represents **one character** in a search term: The following example finds text documents
containing "CLOB" and "BLOB".

```
SELECT DATE, SUBJECT
FROM DB2TX.SAMPLE
    WHERE DB2TX.CONTAINS (COMMENTHANDLE, '"_LOB"') = 1
```

Searching for terms that already contain a masking character

If you want to search for a term that contains the "%" character or the "_" character, you must precede the character by a so-called *escape* character, and then identify the escape character using the ESCAPE keyword.

For example, to search for "10% interest":

```
SELECT DATE, SUBJECT
FROM DB2TX.SAMPLE
WHERE DB2TX.CONTAINS (COMMENTHANDLE, '"10!% interest" ESCAPE "!"') =
1
```

The escape character in this example is "!".

Searching for terms in any sequence

If you search for "hard disk" as shown in the following example, you find the two terms only if they are adjacent and occur in the sequence shown, regardless of the index type you are using.

```
SELECT DATE, SUBJECT
FROM DB2TX.SAMPLE
    WHERE DB2TX.CONTAINS (COMMENTHANDLE, '"hard disk"') = 1
```

To search for terms in any sequence, as in "data disks and hard drives", for example, use a comma to separate the terms:

```
SELECT DATE, SUBJECT
FROM DB2TX.SAMPLE
   WHERE DB2TX.CONTAINS (COMMENTHANDLE, '("hard", "disk")') = 1
```

Searching for terms in the same sentence or paragraph

Here is an example of a search argument that finds text documents in which the search terms occur in the same sentence:

```
SELECT DATE, SUBJECT
FROM DB2TX.SAMPLE
```

```
WHERE DB2TX.CONTAINS (COMMENTHANDLE,
   '"compress" IN SAME SENTENCE AS "decompress"') = 1
```

You can also search for more than two words occurring together. In the next example, a search is made for several words occurring in the same paragraph:

```
SELECT DATE, SUBJECT
FROM DB2TX.SAMPLE
   WHERE DB2TX.CONTAINS (COMMENTHANDLE,
         '"compress" IN SAME PARAGRAPH AS "decompress"
                  AND "encryption"') = 1
```

Searching for synonyms of terms

For a linguistic or a dual index, you can make your searches more flexible by looking not only for the search terms you specify, but also for words having a similar meaning. For example, when you search for the word "book", it can be useful to search also for its synonyms. To do this, specify:

```
SELECT DATE, SUBJECT
FROM DB2TX.SAMPLE
   WHERE DB2TX.CONTAINS (COMMENTHANDLE,
            'SYNONYM FORM OF "book"') = 1
```

When you use SYNONYM FORM OF, it is assumed that the synonyms of the term are connected by a logical OR operator, that is, the search argument is interpreted as:
"book" | "article" | "volume" | "manual"
The synonyms are in a dictionary that is provided with Text Extender. The default dictionary used for synonyms is always US_ENGLISH, not the language specified in the text configuration settings.

You can change the dictionary for a particular query by specifying a different language. Here is an example:

```
SELECT DATE, SUBJECT
FROM DB2TX.SAMPLE
   WHERE DB2TX.CONTAINS (COMMENTHANDLE,
         'SYNONYM FORM OF UK_ENGLISH "programme"') = 1
```

Tip
You cannot use the SYNONYM keyword if there are masking characters in a search term, or if NOT is used with the search argument.

Making a linguistic search

Text Extender offers powerful linguistic processing for making a search based on the search terms that you provide. The linguistic functions are applied when the index is linguistic.

An example of this is searching for a plural form, such as "utilities", and finding "utility". The plural is reduced to its base form utility, using an English dictionary, before the search begins.

The English dictionary, however, does not have the information for reducing variations of terms in other languages to their base form. To search for the plural of a term in a different language you must use the dictionary for that language.

If you specify GERMAN, for example, you can search for "geflogen" (flown) and find all variations of its base form "fliegen" (fly)--not only "geflogen", but also "fliege", "fliegt", and so on.

```
SELECT DATE, SUBJECT
FROM DB2TX.SAMPLE
    WHERE DB2TX.CONTAINS (COMMENTHANDLE,
        'STEMMED FORM OF GERMAN "geflogen"') = 1
```

Tip
When searching in documents that are not in U.S. English, specify the language in the search argument *regardless of the default language.*

If you always specify the base form of a search term, rather than a variation of it, you do not need to specify a language.

To understand why, consider what happens when the text in your database is indexed. If you are using a linguistic index, all variations of a term are reduced to their base form before the terms are stored in the index. This means that, in the DB2TX.SAMPLE table, although the term "decompress" occurs in the first entry in the COMMENT column, "decompression" occurs in the second entry, the index contains only the base form "decompress" and identifies this term (or its variations) as being in both entries.

Later, if you search for the base form "decompress", you find all the variations. If, however, you search for a variation like "decompression", you cannot find it directly. You must specify an appropriate dictionary for the search, so that the variation can first be converted to its base form.

Searching with the Boolean operator NOT

You can use the Boolean operator NOT to exclude particular text documents from the search. For example:

("compress", "compiler") & NOT "DB2"

Any text documents containing the term "DB2" are excluded from the search for "compress" or "compiler".

You cannot use the NOT operator in combination with IN SAME SENTENCE AS or IN SAME PARAGRAPH AS, neither can you use it with SYNONYM FORM OF . You can use the NOT operator only with a search-primary, that is, you cannot freely combine the &, |, and NOT operators.

Example of the use of NOT that is **not** allowed:

NOT("compress" & "compiler")
Allowed is:
NOT("compress" , "compiler")

Searching for similar-sounding words

"Sound" search finds words that sound like the search argument. This is useful when documents can contain words that sound alike, but are spelled differently. The German name that is pronounced my-er, for example, has several spellings.

```
SELECT DATE, SUBJECT
FROM DB2TX.SAMPLE
    WHERE DB2TX.CONTAINS (COMMENTHANDLE,
          'SOUNDS LIKE "Meyer"') = 1
```

This search could find occurrences of "Meyer", "Mayer", and "Maier".

Indexering

Ett IR-system (information retrieval system) utför sökningar genom att matcha sökargumentet mot ord som förekommer i ett på förhand skapat index. Det skulle nämligen ta alldeles för långt tid att vid varje sökning sekventiellt skanna igenom alla ord i dokumenten. Indexet består av relevanta ord/termer som har extraherats från textdokumenten och varje ord/term lagras tillsammans med information om de dokument där de förekommer. På så sätt är det lätt att lokalisera de dokument som matchar sökargumentet.

Med relevanta ord/termer menas sådana som är typiska för dokumentet. Ord som inte är typiska för dokumentet (d.v.s. ofta förekommande ord som prepositioner och pronomen - *och, är, med, utan etc.)* kommer sålunda inte att indexeras. Dessa ord finns i en *Stop Word List* som används för att filtrera bort irrelevanta ord innan ord/termer lagras i indexet.

Det finns lite olika typer av index man kan använda. Dessa påverkar b la vilka olika typer av sökningar man kan använda sig av. De vanligaste indextyperna är *precise, linguistic*, och *ngram*.

Linguistic index

Innan ord/termer lagras i ett index genomgår dokumenttexten en lingvistisk analys. Detta sker även för sökargumentet innan en sökning. De viktigaste delarna av denna process kan delas upp i fyra steg:

1. **Basic text analysis:**
   - Texten analyseras för att ord som innehåller icke alphanumeriska tecken ska lagras i indexet som 1 term (ex ”mother-in-law”, ”$12,234”).
   - Versaler ändras till gemener (ex ”About” ändras till ”about”).
   - Dokumenttexten analyseras för att identifiera var varje mening börjar och slutar. Detta för att man ska kunna söka efter termer som förekommer inom samma mening
2. **Reduction to base form:** Alla ord omvandlas till grundform
3. **Decomposition:** Sammansatta ord (ex ”jobberbjudande”) indexeras dels som helhet, dels som dess beståndsdelar (”jobb” och ”erbjudande”).
4. **Stop-word filtering:** Irrelevanta ord filtreras bort genom att dokumenttexten/sökargumentet jämförs med en Stop Word List innehållandes ofta förekommande ord.

Nedan är en sammanställning av indexeringsprocessen för *linguistic index*.

Table 1. Term extraction for a linguistic index

Document text	Term in index	Linguistic processing
Hatt	hatt	**Basic text analysis** (normalization)
möss simmade stal	mus, simma stjäla	**Reduction to base form**
systembaserad Väderprognos	systembaserad, system, bas väderprognos, väder, prognos	**Decomposition**
En rapport om djur	rapport, djur	**Stop-word filtering**. Stopporden är: en, om

Precise index

Ord/termer lagras i indexet exakt som de förekommer i dokumenttexten. Detta gäller även för sökargumentet, d.v.s. endast exakt överrensstämmelse mellan det indexerade ordet och sökargumentet resulterar i en träff. Fördelen med denna typ av index är att träffen blir mer exakt och att indexering och sökning går snabbare.

Analysprocessen för detta typ av index vid indexering och sökning består av följande delar:

- **Word and sentence separation.** Ord och meningar identifieras genom analys.
- **Stop-word filtering.**

Nedan följer en sammanställning av indexeringsprocessen för *precise index*

Table 2. Term extraction for a precise index

Document text	Term in index	Linguistic processing
Hatt	Hatt	**No normalization**
möss simmade stal	möss simmade stal	**No reduction to base form**
En rapport om djur	rapport, djur	**Stop-word filtering.** Stoppord är: en, om
Systembaserad Väderprognos	Systembaserad Väderprognos	**No decomposition**