DB2TextExtender

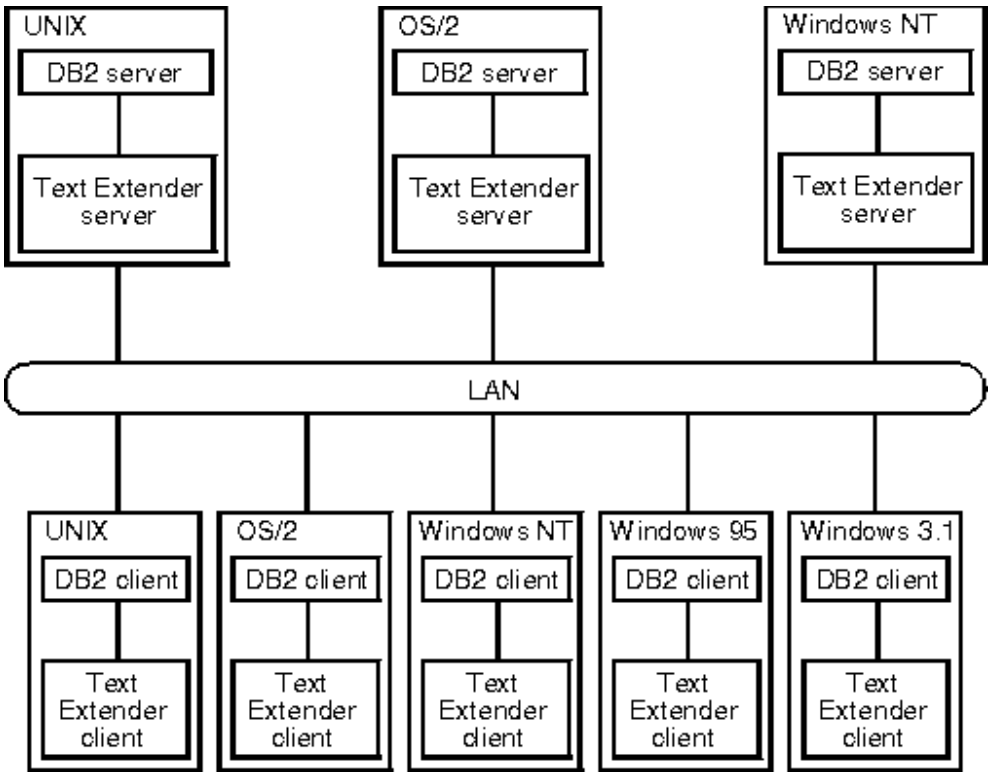
DSV/SU KTH, Susanna Hansson
Helena Lindblad

Introduktion till DB2 - TextExtender

Inledning

DB2 TextExtender är ett tillägg till DB2 Universal Database V5.2. 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.

Bilden nedan visar hur TextExtender är integrerat i DB2s Client/Server miljö.



Text Extender består huvudsakligen av tre delar. Dessa är:

Command line Interpreter. Detta är en kommandoprompt för att utföra kommandon som är specifika för Text Extender (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.

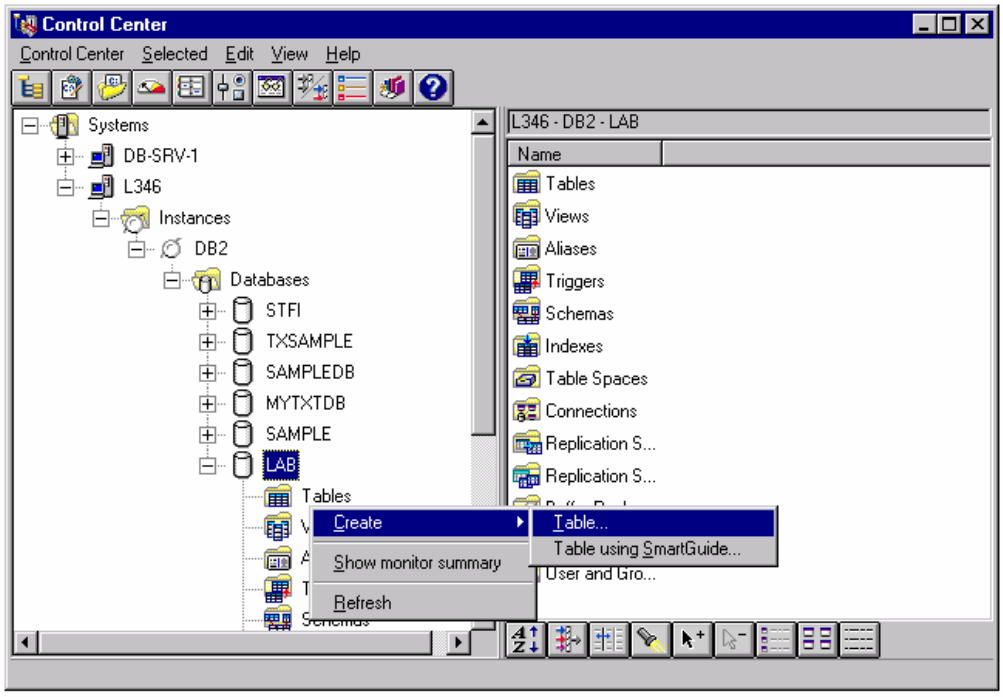
DSV/SU KTH, Susanna Hansson
Helena Lindblad

Skapa en fulltextdatabas

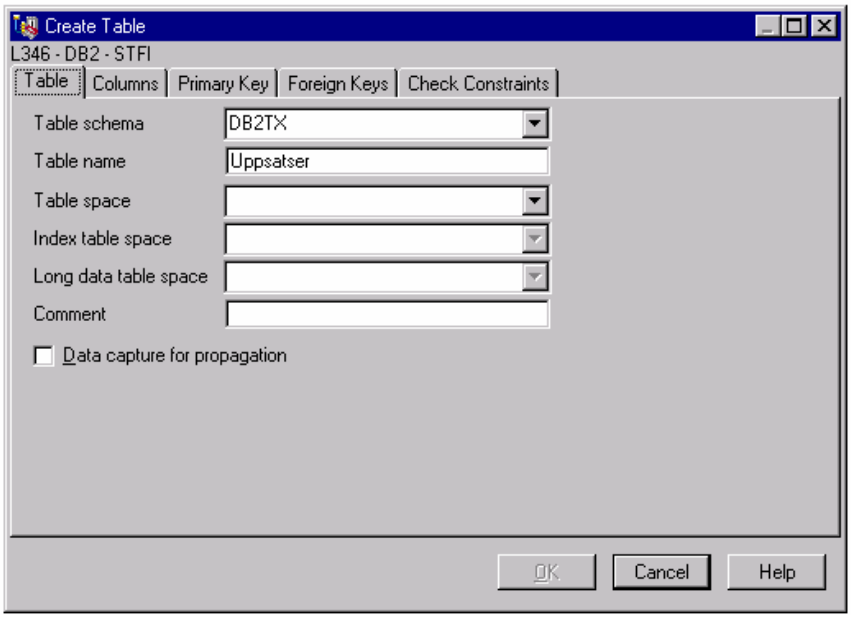
Detta avsnitt beskriver hur ni skapar en fulltextdatabas bestående av magisteruppsatser, skrivna av studenter på DSV, samt uppgifter om dess författare, titel, år och språk. Denna fulltextdatabas kommer sedan att användas för att göra en uppgift som består av att utföra olika typer av sökningar.

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

2. Skapa en tabell som heter *Uppsatser* från DB2 **Control center** enligt följande

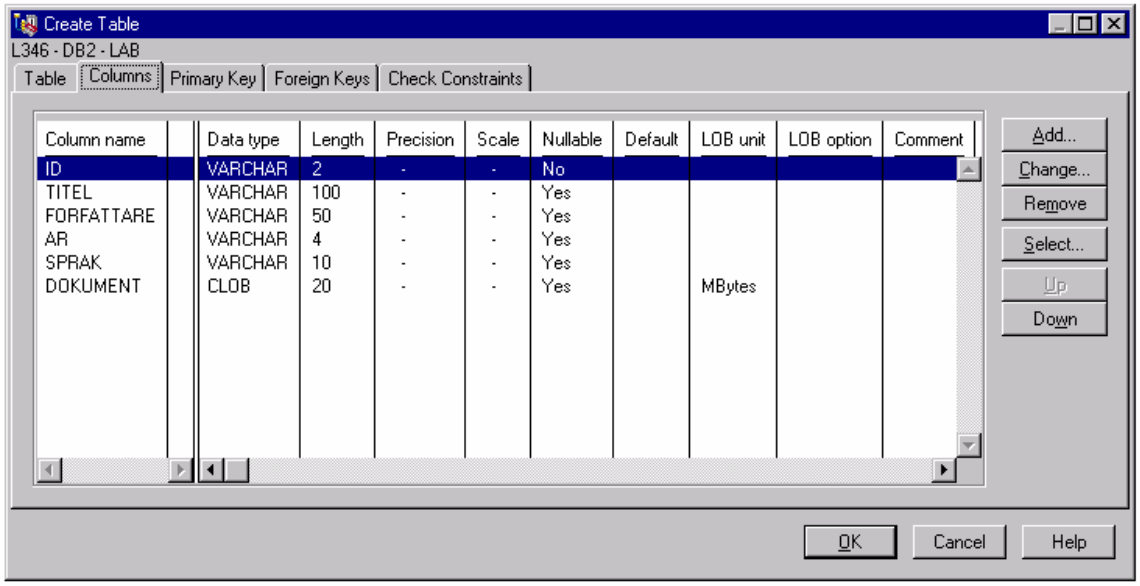


- Ange table schema DB2TX. Se nedan.



- Definiera samma kolumner (via Column knappen, i övre vänstra hörnet) som bilden nedan.

- DOKUMENT kolumnen ska vara av datatypen CLOB (Character Large Object). Det är denna kolumn som ska innehålla RTF-filerna med uppsatser ni senare ska importera. Ändra storleken på CLOB:en (till ca 20Mb) så att du är säker på att ett helt dokument får plats.

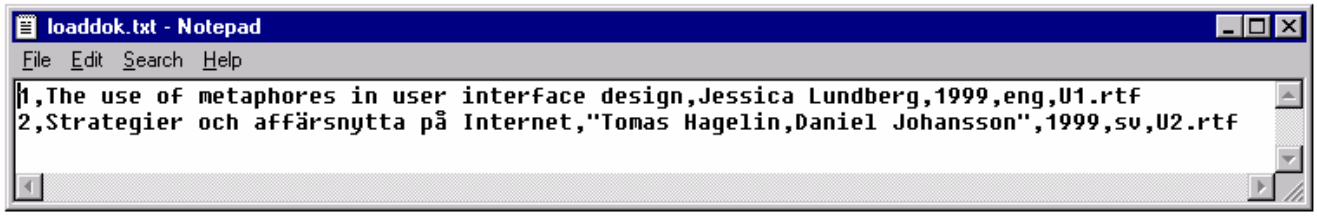


- Klicka OK.

3. Skapa en mapp under er databasprofil på M: som heter *exempel*. Om man har "kontot" jdb024 så blir det: **M:\jdb024*exempel***

4. Ladda ner RTF-filerna med uppsatser från ***Network Neighborhood, dbsrv-1, StudKursInfo, x62HT1999, DB2TextExtender, TextExtDokuments*** till exempelmappen.

5. Skapa ett textdokument som heter *loaddok.txt* i Notepad och sparas i exempelmappen ni just skapat. Denna textfil ska innehålla den data ni vill importera till respektive kolumn. Värdena skrivs i den ordning som kolumnerna ligger i tabellen. Värdet för CLOB-kolumnen utgörs av namnet på RTF-filen (ex *U1.rtf*). Nedan följer ett exempel på två rader som ska laddas in till tabellen (ni ska emellertid ladda in alla uppsatserna):



- För de övriga dokumenten gäller följande:
  (OBS, som ID kan du välja vilket tvåsiffrigt unikt värde som helst. Alla årtal stämmer inte riktigt överens med verkligheten utan är påhittade av oss för att utsökningarna ska bli lite roligare.)

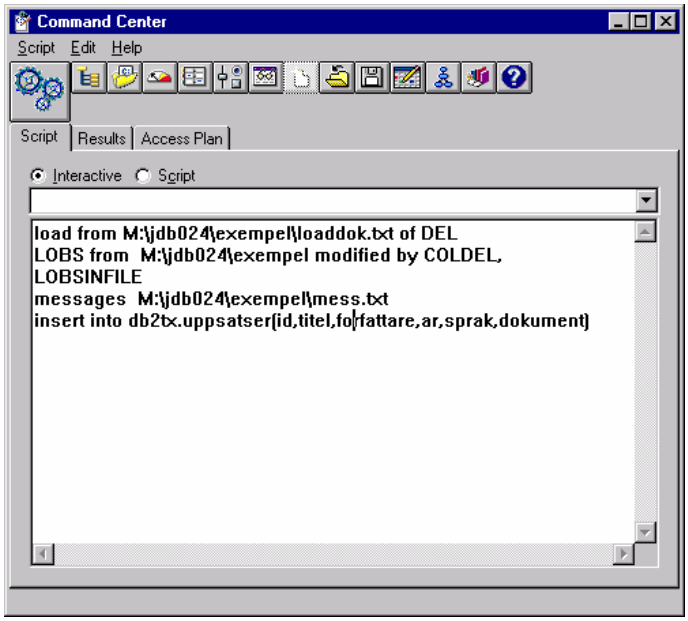
  – Dokument U3.rtf är ett engelskt dokument, skrivet 1998 av Monica Rosell och Susanne Wallin, med titeln *Technologies and Activities in Digital Signatures.*
  – Dokument U4.rtf är ett engelskt dokument, skrivet 1997 av Lars Andersson, med titeln *A study of the Year 2000 problem in a multinational company*.
  – Dokument U5.rtf är ett svenskt dokument, skrivet 1999 av Lena Norberg och Elene Kalhori, med titeln *Distansarbete*.

- Kolumnvärdena separeras med kommatecken. Om kommatecken används inom ett kolumnvärde så måste hela kolumnvärdet omges av citationstecken (ex. ”Tomas Hagelin,Daniel Johansson”).

- Rader skiljs åt med radslutstecknet Enter (OBS! Tryck ej Enter efter sista raden.)

6. Gå in i **Command center** och ”connecta” till databasen med nedanstående kommando:
Connect to lab

7. Ladda in dokumenten och annan data genom att från **Command center** ange följande load kommando (OBS, notera kommatecknet efter COLDEL):



- **Den första sökvägen** (M:\jdb024\exempel\loaddok.txt) anger vart din skapade Notepadfil ligger.

- **of DEL** betyder att du ska använda formatet Delimited ASCII när du ska ladda in Notepadfilen. Detta format styr hur du separerar kolumnvärden och rader i din Notepadfil (se punkt 3 ovan)

- **Den andra sökvägen** (M:\jdb024\exempel) anger vart RTF-dokumenten som ska laddas in i CLOB kolumnen ligger.

- **Lobsinfile** betyder att RTF-dokumenten ligger i separata filer utanför Notepad-filen.

- **Messages** beskriver vart DB2 skall skicka meddelande om, och i så fall hur, load kommandot genomfördes. Ni måste själva skapa detta dokument (tomt).

- **Insert** kan självklart bytas ut mot replace, update etc. beroende på vad man vill göra.

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. Gå till **DB2TX Command Line Processor** (via Start/Programs/DB2Extenders/DB2TX Command Line Processor) och ”connecta” till databasen.

db2tx => *connect to lab*

10. Kör kommandot *enable database*. Detta kommando skapar en TextExtender-katalog som heter DB2tx.TextColumns. Denna katalog innehåller information om tabeller och kolumner som är ”enablade” för Text Extender. Informationen i denna ändras varje gång man ger kommandot ”enable” för en tabell, kolumn eller extern fil.

11. Kontrollera default inställningar med hjälp av kommandot *get text configuration (cfg)* så att de stämmer överens med nedanstående.

db2tx => GET TEXT CFG

Coded character set ID (CCSID) = 850
Language (LANGUAGE) = SWEDISH
Format (FORMAT) = RTF
Index type (INDEXTYPE) = DUAL

CCSID	Varje DB2 databas använder en viss ”code page” när den lagrar data. TextExtender anväder sig av samma ”code page” som databasen. I vårt fall använder sig DB2 av CCSID 850 vilket gör att du måste se till att TextExtender använder sig av samma.
Language	TextExtender måste veta vilket språk de importerade dokumenten är skrivna i för

	att veta vilket ”dictionary” som skall användas vid indexeringen
Format	Anger formatet på de dokument som skall importeras.
Index type	Anger vilken typ av index som skall skapas. Se bilaga om indexering.

- För att ändra de värden som skiljer sig åt ange följande kommando:

db2tx => *change text configuration using ccsid 850 format rtf indextype dual language swedish*

12. Kör kommandot *enable text column* för kolumnen där RTF-filerna ligger.

db2tx => *enable text column db2tx.uppsatser dokument handle commenthandle*

- *dokument* är namnet på CLOB kolumnen .

- *handle* skapar ett handtag av typen DB2TEXTFH eller DB2TEXTH beroende på om det är en extern fil eller ett dokument lagrat i databasen som skall indexeras. I detta fallet skapas ett hantag av typen DB2TEXTH. Handtaget innehåller följande information som behövs för att kunna indexera ett dokument:

  - Dokument ID
  - Namnet på servern där texten skall bli indexerad
  - Namnet på indexet
  - Information om textdokumentet

- Handtaget lagras i en kolumn som skapas när kommandot körs. Namnet på denna kolumn namnger du efter ordet *handle* (dvs. i detta fall *commenthandle*)

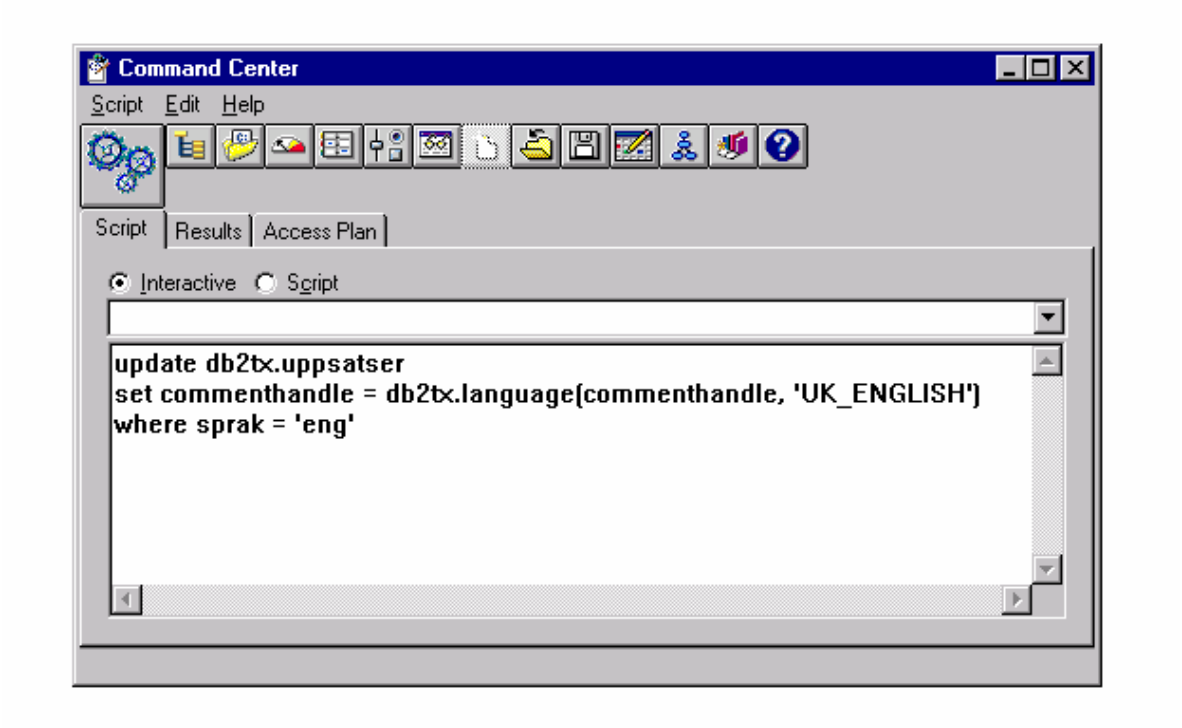
- Kommandot *disable text column* raderar handtaget och således all information det innehåller (bl a indexet). Hur syntaxen för detta kommandot ser ut kan du urskilja genom att skriva ? (följt av mellanslag) före kommandot. Om man har tänkt att ”droppa” en tabell måste detta kommando alltid köras innan.

13. För att kontrollera att indexeringen har fungerat kan du använda dig av kommandot *get index status* i db2tx. Skriv *? get index status* för att få den fullständiga syntaxen. Om indexeringen fungerat felfritt får man följande meddelande (förutsatt att två dokument ligger i tabellen, om ni laddat in fler dokument är det detta antal som ska stå efter *Indexed documents* =).

- Indexeringen kan ta lite tid, så ge er till tåls innan antalet dokument ni indexerat anges efter *Indexed documents* =. Innan indexeringen är färdig så står istället antalet dokument efter *Scheduled documents*=. Detta betyder att dessa dokument står i kö för att indexeras.

Node 0
Search status = Search available
Index status = Update available
Scheduled documents = 0
Indexed documents = 2
Error events = No error events.

14. Eftersom några av dokumenten som skall lagras är skrivna på engelska måste språket ändras för dessa dokument för att ett korrekt index skall kunna skapas. Detta görs enligt nedanstående.

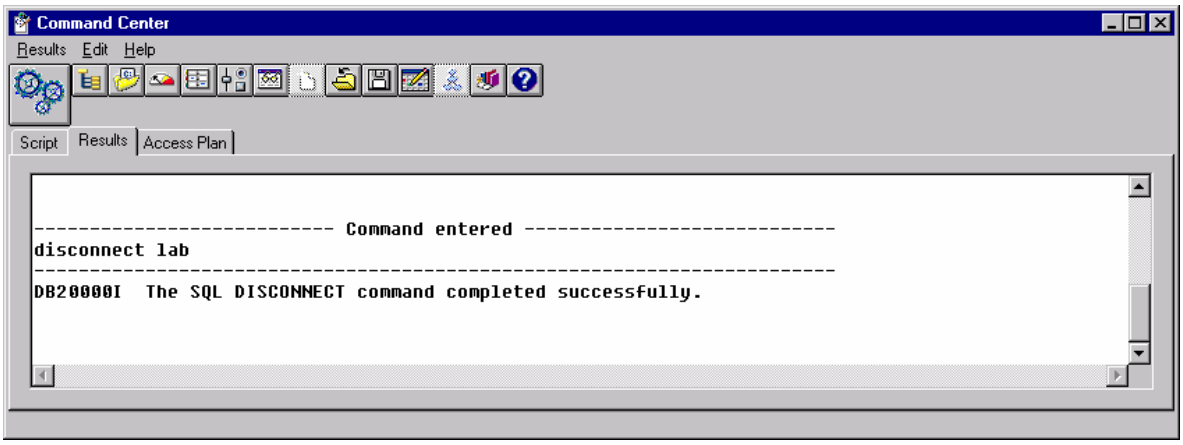


15. För att se om språket har ändrats till engelska kan du skriva följande i **Command center**:
select distinct db2tx.language(commenthandle) from db2tx.uppsatser

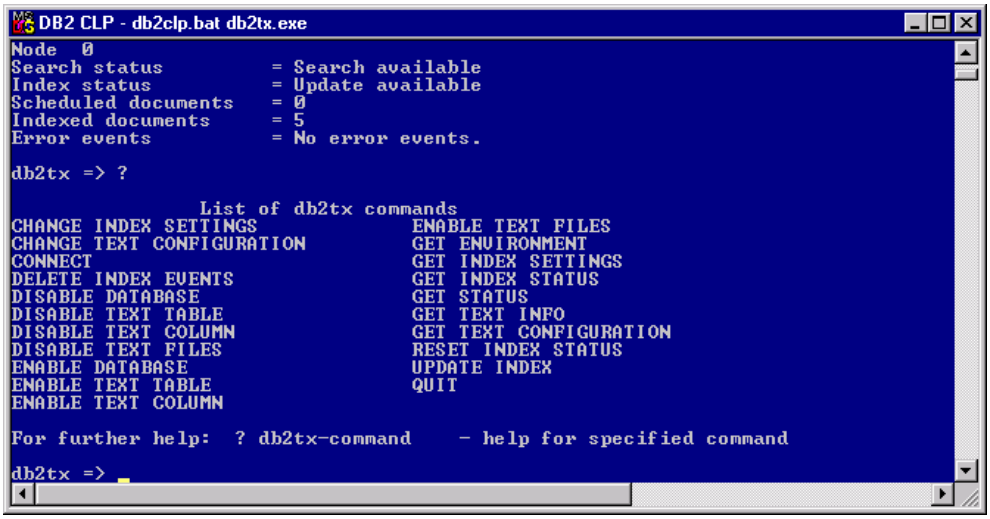
16. OM ALLT HAR GÅTT BRA ÄR DATABASEN REDO FÖR SÖKNING!

17. För att avsluta en session måste du göra följande:

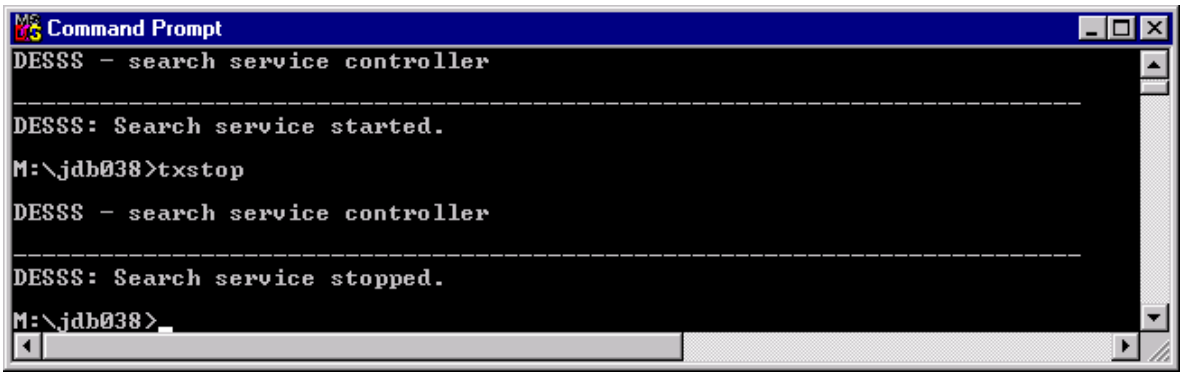
1) Från Command Center ges kommandot: disconnect lab



2) För att lämna DB2TX Command LineProcessor: Ange kommandot quit



3) Från Command Prompt fönstret ges kommandot: txstop



Uppgift

1. Innan ni börjar med uppgiften, starta processen *txstart* i en *Command Promt* (om den inte redan är igång).
2. Frågorna skrivs i Control Center, så glöm ej att göra *connect to lab* i Control Center innan ni börjar.
3. För att kunna lösa uppgiften rekommenderar vi er även att läsa igenom referensmaterialet om sökning och testa de exempel som beskrivs där.

För att få en inblick i hur SQL kombineras med UDF:er i er tabell följer tre små exempel:

1. Select titel from db2tx.uppsatser
   Where db2tx.contains(commenthandle,'"term"')=1

Detta exempel visar den vanligaste UDF:en CONTAINS. Provkör gärna, resultatet ska visa titlarna på de dokument som innehåller ordet term

2. Select titel from db2tx.uppsatser
   Where db2tx.contains(commenthandle,'"term"')=1
   and sprak='eng' and ar>'1997'

Detta exempel visar hur man kombinerar UDF med vanliga SQL villkor.

3. With temptable(titel,rank)
   as (select titel,db2tx.rank(commenthandle,'"term"')
   From db2tx.uppsatser)

```
Select * from temptable where
Rank>0
Order by rank desc
```

Här används UDF:en RANK för att rangordna träffarna efter relevans. Jämför gärna resultatet med resultatet från det första exemplet, det borde skilja sig åt.

Frågor

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 börjar antingen på bokstaven D eller S.

3. Vilka engelska dokument (titlar) är skrivna efter 1997 och innehåller ordet *term*? Ordna träffarna efter relevans på ”fritextsökningen”.

4. Kör nedanstående fråga:
   Select titel from db2tx.uppsatser where db2tx.contains(commenthandle, ’”term”’)=1

   - Notera antalet träffar
   - Utgå från frågan och lägg till villkoret att dokumenten även måste innehålla ordet *uppsats*. På detta sätt så kan man minska antalet träffar från en tidigare definierad fråga
   - TIPS: Läs i referensmaterialet, det finns en speciell UDF för just detta ändamål

5. Ta reda på vad det blir för skillnad i träffresultatet på följande villkor:

   - Ta fram titel, ar, sprak för de dokument som innehåller orden *Internet* och *dator* men inte ordet *uppsats* och inte den exakta formen av ordet *metod*.

   - Ta fram titel, ar, sprak för de dokument som har orden *Internet* och *dator* i en och samma mening och där ordet *uppsats* och den exakta formen av ordet *metod* inte förekommer en enda gång i hela dokumentet.

Referensmaterial

Sökning (kap 5 i manualen)

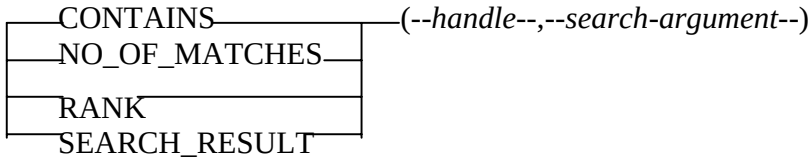
När man vill söka i dokumenten så går man som vanligt in i Control Center och skriver sin fråga. För sökning i dokumenten används en form av ”utökad” SQL. Denna består dels av vanlig SQL och dels av funktioner, s.k. UDF:er (User Defined Functions). Dessa deklareras automatiskt när man kör kommandot *enable database*. Nedanstående tabell visar vilka UDF:er som finns. I filen txsamle.udf (C:\dmb\samples\txsample.udf) finns olika exempel på hur dessa kommandon används vid sökning.

UDFs

Search function (UDF)	**Purpose**
CCSID	Returns the CCSID from handle
CONTAINS	Makes a search for text in a particular document
FILE	Returs or changes the path and name of a file in an existing handle
FORMAT	Returns the format of a document
HANDLE	Returns a handle from a list of handles
HANDLE_LIST	Searches and returns a list of handles
INIT_TEXT_HANDLE	Returns a partially initialized handle containing information such as format and language settings
LANGUAGE	Returns or changes the language setting in a handle
NO_OF_ DOCUMENTS	Returns the number of documents listed in a handle list
NO_OF_MATCHES	Searches and returns the number of matches found
RANK	Searches and returns the rank value of a found text document
REFINE	Takes a search argument and a refining search argument and returns a combined search argument
SEARCH_RESULT	Returns an intermediate table with the search result of the specified search string

Samtliga UDF:er tar ett handtag som argument, man skickar således inte med dokumentkolumnen utan man skickar med kolumnen där handtaget ligger. Nedan följer ett utdrag ur manualen som visar hur dessa UDF:er fungerar.

Searching for text



This section describes how to use the UDFs provided with Text Extender to search in DB2 databases containing text. It tells you how to:

- Make a query
- Determine how many matches were found in a text document
- Get the rank of a found text document.

The use of SEARCH_RESULT is described in improving search performance. Each of these UDFs searches in the text index for occurrences of the search argument. If there are, say, 100 000 text documents in the table, the CONTAINS, RANK, or NO_OF_MATCHES UDF is called 100 000 times. But the text index is not searched 100 000 times. Instead, the first time the UDF is called, an internal list of all the documents containing the search term is created; subsequent calls of the UDF determine if the document concerned is in the list.

Tip
When you use the Text Extender UDFs to search in a table, be sure to pass the handle column to the UDF, rather than the text column. If you try to search in a text column, SQL responds with a message indicating that the data type is wrong, for example: No function by the name "CONTAINS" having compatible arguments was found in the function path.

If you search for text immediately after issuing the ENABLE TEXT TABLE or ENABLE TEXT COLUMN command, an error RC_SE_EMPTY_INDEX can occur which indicates that the index being created by the command does not yet exist. The time taken for an index to be created depends on factors such as the number of documents being indexed, and the performance of the system doing the indexing. It can vary from several minutes to several hours, and should be done when the system is lightly loaded, such as over night.

If this message occurs, try searching again later, or use GET INDEX STATUS to check whether indexing errors have occurred.

Making a query

This example demonstrates how the CONTAINS function searches for text in documents identified by a handle. It returns 1 if the text satisfies the search argument, otherwise it returns 0.

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

In this example, you search for the term compress in the text referred to by the handles in the column COMMENTHANDLE. The handles in the COMMENTHANDLE column indicate where the COMMENT text is indexed.

Tip
If you have created mixed-case identifiers for tables or columns, remember that these names must be enclosed in double quotes. For example: SELECT DATE, SUBJECT FROM **"DB2TX.Sample"** WHERE DB2TX.CONTAINS (COMMENTHANDLE, '"compress"') = 1

Number of matches found

Use the NO_OF_MATCHES function to determine how often the search criteria are found in each text document. This function returns an integer value.

```
WITH TEMPTABLE(DATE, SUBJECT, MATCHES)
    AS (SELECT DATE, SUBJECT,
    DB2TX.NO_OF_MATCHES(COMMENTHANDLE,'"compress"')
    FROM DB2TX.SAMPLE)
    SELECT *
        FROM TEMPTABLE
        WHERE MATCHES > 0
```

Rank of a found text document

RANK is an absolute value that indicates how well the document met the search criteria relative to other found documents. The value indicates the number of matches found in the document in relation to the document's size. This function returns a DOUBLE value between 0 and 1.

You can get the rank of a found document by using the RANK UDF.
Here is an example:

```
WITH TEMPTABLE(DATE, SUBJECT, RANK)
    AS (SELECT DATE, SUBJECT,
    DB2TX.RANK(COMMENTHANDLE,'"compress"')
    FROM DB2TX.SAMPLE)
    SELECT *
        FROM TEMPTABLE
        WHERE RANK > 0
        ORDER BY RANK DESC
```

Specifying search arguments

Search arguments are used in CONTAINS, NO_OF_MATCHES, RANK, and HANDLE_LIST. 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.".

If you are using a **dual** index, you can choose to search for word variations or not. For example, the following query finds only occurrences of "utility":

```
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. If you use a dual index type, the masked search is case-sensitive.

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 or when a dual index is used with STEMMED FORM OF parameter.

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 or a dual 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")
```

Respecting word-phrase boundaries

"Bound" search has been developed for the Korean language. It ensures that Text Extender respects word boundaries during the search. For example:

```
SELECT DATE, SUBJECT
FROM DB2TX.SAMPLE
    WHERE DB2TX.CONTAINS (COMMENTHANDLE,
          'BOUND "korean-expression"') = 1
```

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 myer, 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".

Free-text and hybrid search

"Free-text search" is a search in which the search term is expressed as free-form text. A phrase or a sentence describes in natural language the subject to be searched for. The sequence of words in a free-text query are not relevant. Furthermore, so-called *lexical affinities* are supported. In retrieval, these are certain pairs of words occurring in a free-text query term, and occurring in the document collection, with a certain minimal frequency and a certain minimal distance. The distance for English documents is five words.

Note that the masking of characters or words is not supported for search strings in a free-text argument.
For example:

```
SELECT DATE, SUBJECT
FROM DB2TX.SAMPLE
    WHERE DB2TX.CONTAINS (COMMENTHANDLE,
            'IS ABOUT "everything related to AIX installation"') = 1
```

Hybrid search is a combination of Boolean search and free-text search. For example:

```
SELECT DATE, SUBJECT
FROM DB2TX.SAMPLE
    WHERE DB2TX.CONTAINS (COMMENTHANDLE,
            '"DB2" & IS ABOUT "everything related to AIX installation"') = 1
```

Refining a previous search

When a search argument finds too many occurrences, it can often be useful to narrow, or *refine*, the search by combining the initial search argument with a second search argument in a Boolean-AND relationship.

You can refine search results without using the REFINE function, by storing the results in a table and making the next search against this table. However, depending on the number of qualifying terms, this method is less efficient than that of storing the latest search argument and using REFINE.

The following steps show how to make a search, and then refine it using the REFINE function. The REFINE function returns a search argument that is a Boolean-AND combination of its two input parameters. The combined search argument returned by REFINE is a value of type LONG VARCHAR.

1. **Create a table for old search arguments**.
   Create a table PREVIOUS_SEARCHES to hold the search arguments of searches that have already been made.

```
CREATE TABLE PREVIOUS_SEARCHES (step INT,
            searchargument LONG VARCHAR)
```

2. **Search for the first search argument**.
   Search for the word "compress" in the sample table.

```
SELECT COMMENT
 FROM DB2TX.SAMPLE
 WHERE DB2TX.CONTAINS (COMMENTHANDLE, '"compress"') = 1
```

3. **Insert the search argument into the previous searches**
   Insert the search argument into the PREVIOUS_SEARCHES table for use by further steps.

```
INSERT INTO PREVIOUS_SEARCHES
  VALUES (1, '"compress"')
```

4. **Refine the search.**
   Assuming that the search returns too many text documents, refine the search by combining the previous search term with the word "compiler" using the REFINE function.

```
WITH LAST_STEP(STEP_MAX)
    AS (SELECT MAX(STEP)
       FROM PREVIOUS_SEARCHES),
    LAST_SEARCH(LAST_SEARCH)
    AS (SELECT SEARCHARGUMENT
       FROM PREVIOUS_SEARCHES,LAST_STEP
       WHERE STEP = STEP_MAX)
SELECT COMMENT
    FROM DB2TX.SAMPLE, LAST_SEARCH
    WHERE DB2TX.CONTAINS(COMMENTHANDLE,
            DB2TX.REFINE(LAST_SEARCH, '"compiler"')) = 1
```

5. Insert the refined search argument into the PREVIOUS_SEARCHES

Insert the refined search argument into the PREVIOUS_SEARCHES table for use by further steps.

```
INSERT INTO PREVIOUS_SEARCHES
  WITH LAST_STEP(STEP_MAX)
     AS (SELECT MAX(STEP)
        FROM PREVIOUS_SEARCHES)
  SELECT STEP_MAX+1, DB2TX.REFINE(SEARCHARGUMENT, '"compiler"')
    FROM PREVIOUS_SEARCHES, LAST_STEP
```

You can repeat this step until the number of text documents found is small enough.

Indexering (kap 2&3 i manualen)

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 scanna 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 *dual*.

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
Systembasera Väderprognos	Systembaserad Väderprognos	**No decomposition**

Dual index

Detta index är en kombination mellan *precise* och *linguistic*. Vid sökning kan man välja om man vill söka med en *precise* textanalys eller *linguistic* textanalys. Nackdelen med detta index är att den tar mycket diskutrymme samt att sökningen är långsammare än de övriga indextyperna.