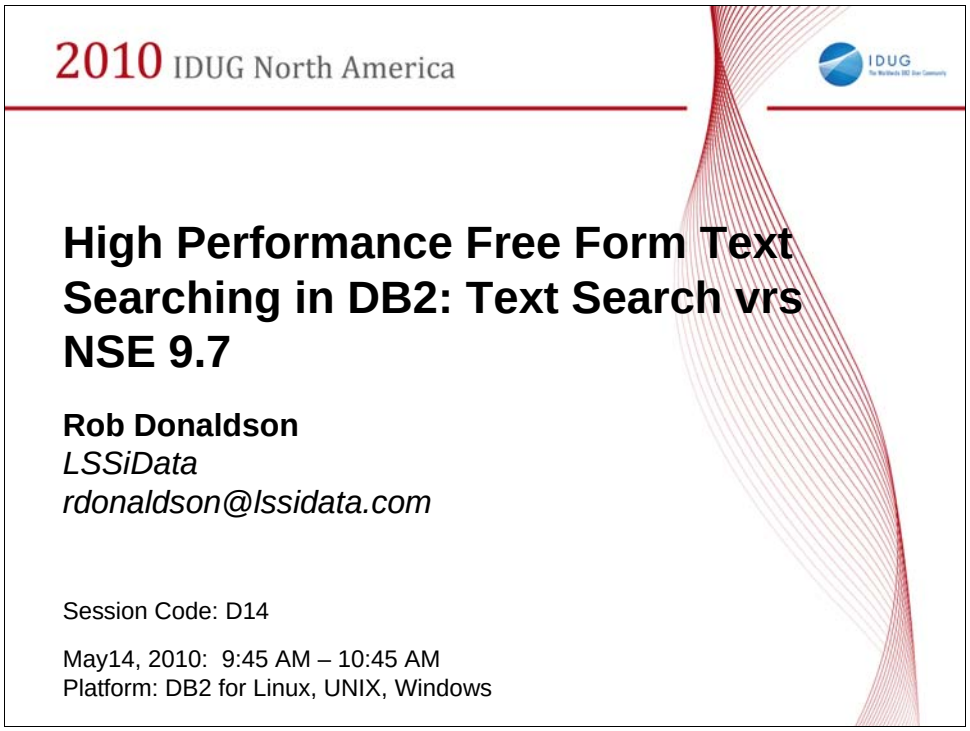


Presentation Abstract

Compares Full Text Search options in DB2 and gives indexing performance comparisons.

Objectives

- Know what free form text search options are available in DB2 UDB 9.1, 9.5, and 9.7.
  - Old NSE (DB2 7.2, 8.x, 9.1, 9.5)
  - Text Search (introduced in DB2 9.5)
  - NSE 9.7
  - The old fashioned word support table method.
- Compare the functionality between the text search options.
- Real world indexing performance and space requirements.
- Text indexing update considerations.
- Things I've learned to watch out for and advise on usage, primarily relating to NSE.

Points of this presentation:

Text Searching: The 3 interfaces available in NSE.

Common problems and Gotchas and how to avoid them.

Performance, performance: Good searches, Bad searches.

Good tuning parameters

Memory usage and considerations

Text Search Basics

- Text Search systems allow you to independently index each word in a document.
- Quickly find documents containing all or a subset of terms.
- Scoring
- Case insensitive by default.
- Documents can be unformatted text or formatted (XML, HTML, Word doc, etc…).
- Thesaurus support for alternate forms of words
- Fuzzy Searching for similar spellings
- Support for Double Byte Character sets or Unicode
- Etc…

DB2 Text Search: Introduced in 9.5

- Based on OmniFind
- Built in Java
- Admin commands are similar to NSE
  - *db2ts* for DB2 Text Search (instead of db2text for NSE)
- SQL searching similar in interface to NSE's Scalar Search interface functions
  - Contains()
  - Score()
  - However, Index Attributes are NOT supported in Text Search
- Does **NOT** support DPF

Installation Notes

DB2 Instance notes:

- When the DB2 instance is created the DB2 Instance owner and the DB2 fenced user **MUST** be members of the same primary group.
  - This is required for both NSE or Text Search introduced in 9.5
- Text Search comes with DB2 9.5 and 9.7 ESE, but is an optional component that must be selected from a custom install.
  - Not included in Warehouse Edition as DPF is not supported.
  - Use db2setup (instead of db2_install) to install and configure the Text Search Server.
- NSE is a plug-in for DB2 and must be installed separately.
  - Install Net Search Extender after DB2 has been installed.
    - It will be installed into the same directory the DB2 installation resides in.
    - Update existing instances using db2iupdt
- You Can have both NSE and Text search installed.
  - However, a given database can only be enabled for one or the other at a time.

Note: Text Search runs an Omni-find Text Indexing dedicated server as a separate process on it's own TCP/IP port. When you install and configure Text Search, it will create an Omni-Find server running under the same user ID as the DB2 Instance owner. The DB2 Server communicates with it on a dedicated TCP/IP port that is specified at the time Text Search is installed and setup in db2setup.

NSE and Text Search Index Notes

- Text indexes are stored on a file system, NOT in a tablespace.
  - NSE indexes are stored in:
    - Default directory:
      ${HOME}/sqllib/db2ext/indexes
    - Specify the storage directory with: INDEX DIRECTORY
  - Text Search indexes are stored in:
    - Default directory:
      ${DBPATH}/NODExxxx/SQLxxxx/db2collections/index identifier/data
    - Specify the storage directory with: COLLECTION DIRECTORY

- NSE and Text Search indexes are buffered by the native OS disk file system cache, NOT in DB2 bufferpools.
  - Make sure you have enough Memory free for OS disk file system caching. (Don't configure your database to use all available RAM.)
  - If possible, create your NSE indexes on a dedicated file system so you can monitor i/o.

Enabling DB2 For Text Search Compared to NSE

Text Search

- Start DB2 Text by executing: (As the DB2 instance owner)
  db2ts START FOR TEXT
  - db2ts is NOT started automatically when the instance is started.
- Enable your database for text:
  db2ts ENABLE DATABASE FOR TEXT CONNECT TO SAMPLE

NSE

- Start NSE by executing: (As the DB2 instance owner)
  db2text start
  - db2text is NOT started automatically when the instance is started.
- Enable your database for text:
  db2text ENABLE DATABASE FOR TEXT CONNECT TO SAMPLE

You Can have both NSE and Text search installed, but a given database can only be enabled for one or the other. (The same database CAN NOT be enabled for both at the same time.)

Disabling DB2 For Text Search Compared to NSE

- Removing Text Search from a Database:
  - Disable your database for text:
    db2ts DISABLE DATABASE FOR TEXT FORCE CONNECT TO SAMPLE

- Removing NSE from a Database:
  - Disable your database for text:
    db2text DISABLE DATABASE FOR TEXT FORCE CONNECT TO SAMPLE

Creating a Text Search Index (OmniFind)

- Sample Table:

```
create table dbadmin.LISTINGS (
        LRECID bigint NOT NULL,
        NPA smallint,
        GAC char(4),                    -- 4 character code (numeric string. i.e. '0123')
        NSEXMLIDX varchar(1024),        -- XML Document for this listing
        PRIMARY KEY (LRECID))           -- Table MUST have a primary key for NSE or Text Search
```

- Sample Create Index: *This defines the index object only

```
db2ts "CREATE INDEX dbadmin.LISTINGS_NSEXMLIDX
        FOR TEXT ON dbadmin.LISTINGS (NSEXMLIDX)
        FORMAT XML
        COLLECTION DIRECTORY /db2/TextIndexdsk/MYDB/LISTINGS
        CONNECT TO MYDB"
```

- Update the index: *You must update the index after creation

```
db2ts "UPDATE INDEX dbadmin.LISTINGS_NSEXMLIDX FOR TEXT CONNECT TO MYDB"
```

Primary Key requirement

Must update the index after creation

Note: NSE supports Attributes but Text Search does not. Similarly, Document Models are not supported by Text Search.

Creating an NSE Text Index

- Sample Table:

```
create table dbadmin.LISTINGS (
        LRECID bigint NOT NULL,
        NPA smallint,
        GAC char(4),                    -- 4 character code (numeric string. i.e. '0123')
        NSEXMLIDX varchar(1024),        -- XML Document for this listing
        PRIMARY KEY (LRECID))           -- Table MUST have a primary key for NSE or Text Search
```

- Sample Create Index: *This defines the index object only

```
db2text "CREATE INDEX dbadmin.LISTINGS_NSEXMLIDX
        FOR TEXT ON dbadmin.LISTINGS (NSEXMLIDX)
        ATTRIBUTES (CAST(NPA as DOUBLE) AS NPA,
                        CAST(CAST(GAC as INT) as DOUBLE) AS GAC)
        FORMAT XML DOCUMENTMODEL listingsmodel IN /db2/model/listingsmodel
        INDEX DIRECTORY /db2/NSEindexdsk/MYDB/LISTINGS
        CONNECT TO MYDB"
```

- Update the index: *You must update the index after creation

```
db2text "UPDATE INDEX dbadmin.LISTINGS_NSEXMLIDX FOR TEXT CONNECT TO MYDB"
```

Primary Key requirement

Must update the index after creation

Note the cast to double for attributes

Updating Text Search Indexes (OmniFind)

- Text indexes are **not** updated by DB2 as part of the transaction when a row is updated.
- Maintenance before you update your Text Indexes:
  - Clear your Event tables before updating to remove old messages
    db2ts "CLEAR EVENTS FOR INDEX dbadmin.LISTINGS_NSEXMLIDX for TEXT CONNECT TO MYDB"
  - Clear Text Search locks to prevent lock timeouts from blocking the index update
    db2ts "CLEAR COMMAND LOCKS FOR INDEX dbadmin.LISTINGS_NSEXMLIDX FOR TEXT CONNECT TO MYDB "
- Update the Text Index:
  db2ts "UPDATE INDEX dbadmin.LISTINGS_NSEXMLIDX FOR TEXT CONNECT TO MYDB"

Failure to clean your index events log tables will eventually burn you. Had a database run out of space once because of that.

Note: You don't need to worry about incremental updates and when to do an index rebuild. Text Search will do incremental updates and rebuild the complete index when necessary automatically. (Unlike NSE.)

Updating Your NSE Text Indexes

- Text indexes are **not** updated by DB2 as part of the transaction when a row is updated.
- Maintenance before you update your Text Indexes:
  - Clear your Event tables before updating to remove old messages
    db2text "CLEAR EVENTS FOR INDEX dbadmin.LISTINGS_NSEXMLIDX for TEXT CONNECT TO MYDB"
  - Clear NSE locks to prevent NSE lock timeouts from blocking the index update
    db2text "CONTROL CLEAR ALL LOCKS FOR DATABASE MYDB INDEX dbadmin.LISTINGS_NSEXMLIDX"
- Two types of updates:
  - Update: Usually faster but index is fragmented with holes
    db2text "UPDATE INDEX dbadmin.LISTINGS_NSEXMLIDX FOR TEXT CONNECT TO MYDB"
    - Marks deleted index entries and appends new index entries
    - Usually faster than reorganize, but may take longer **or hang** if there are too many updates
  - Update reorganize rebuilds the index:
    db2text "UPDATE INDEX dbadmin.LISTINGS_NSEXMLIDX FOR TEXT REORGANIZE CONNECT TO MYDB"
    - Time consuming and CPU intensive
    - Improved search performance

Failure to clean your index events log tables will eventually burn you. Had a database run out of space once because of that.

Dropping Text Indexes

- **Note:** Always drop text indexes **before** dropping a table

Text Search

- Clear your Event tables before updating to remove old messages
  db2ts "CLEAR EVENTS FOR INDEX schema.LIST_IDX for TEXT CONNECT TO MYDB"
- Drop the Text index:
  db2text "DROP INDEX schema.LIST_IDX for TEXT CONNECT TO MYDB"

NSE:

- Clear NSE locks to prevent NSE lock timeouts/deadlocks from blocking the index drop:
  db2text "CONTROL CLEAR ALL LOCKS FOR DATABASE MYDB INDEX schema.LIST_IDX "
- Drop the Text index:
  db2text "DROP INDEX schema.LIST_IDX for TEXT CONNECT TO MYDB"

Unlike normal indexes, text indexes are NOT dropped automatically when a table is dropped. This is because Text indexes are stored outside the database. Wouldn't it be nice if DB2 had a trigger for 'on drop table'…

Text Search Query Examples

Both Text Search and NSE share query syntax for Contains() and Score()

- **Exceptions:** Text Search does not support **Attributes** or **Sections**

Contains() = 1

- Select * from DEALERS where Contains(XMLDOCCOL, ' "Computer" | "PC" | "Desktop" | "Intel" | "AMD" ') =1

Score()

- Select VENDOR,
  Score(XMLDOCCOL, ' "Computer" | "PC" | "Desktop" | "Intel" | "AMD" ') as SCORE
  from DEALERS where
  Contains(XMLDOCCOL, ' "Computer" | "PC" | "Desktop" | "Intel" | "AMD" ') =1 and
  Score(XMLDOCCOL, ' "Computer" | "PC" | "Desktop" | "Intel" | "AMD" ') > 0.8

Both contains() and score() are supported by NSE and Text Search.

However, NSE's contains clause supports the use of ATTRIBUTES() which Text Search does not.

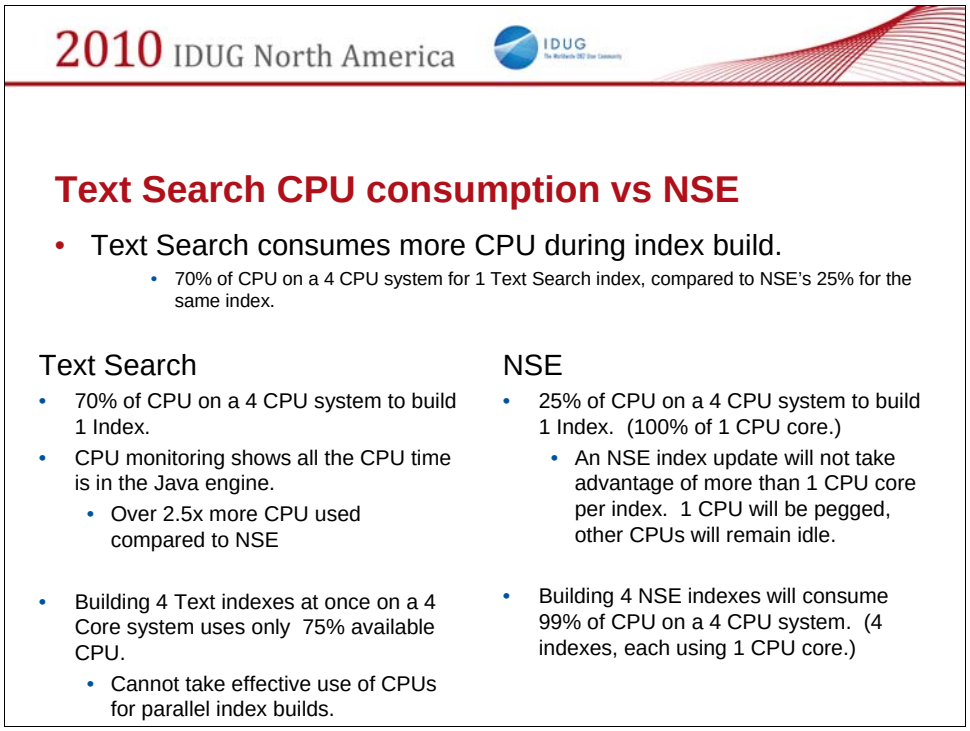
NSE supports the keyword 'SECTION' to limit word matches to specific XML Tag path. Text Search does not.

(An attribute() is used to index a double precision float value (or a data type that can be cast to a double) in a text index so it can be searched along with the text strings to reduce the results set returned by NSE.)

Text Search Indexing Performance vs NSE

- Indexing Speed is not a Text Search advantage…
  - Note: My Text columns are small (average 220 bytes).
  - If you have larger text data Text Search indexing speed is better, but still slower than NSE.

Test	NSE Indexing time	Text Search Indexing time	Text Search performance
AIX: 4 CPU p690 LPAR (old Power 4) Index 1 Table: 3.4 Million rows	16 min, 45 sec	96 min, 34 sec	5.7x slower
AIX: 4 CPU p690 LPAR (old Power 4) Index 4 Tables in parallel: 3.4 Million rows per table	18 min, 38 sec	4 hours, 50 min	15.6x slower
Linux: 2 CPU dual Core (new Intel) Index 1 Table: 3.4 Million rows	4 min, 25 sec	19 min, 9 sec	4.3x slower
Linux: 2 CPU dual Core (new Intel) Index 4 Tables in parallel: 3.4 Million rows per table	4 min, 35 sec	61 min, 54 sec	13.5x slower



Unfortunately, not only does Text Search take 4-5 times longer to index text, it also consumes 2.5 times more CPU to do it at the time.

My advice is to stick with NSE for now, but keep an eye on the next version of Text Search.

NSE Performance: Document Model

- If you have a formatted document (XML, HTML, Word, etc) specify a document model so NSE knows how to index it.

```
db2text "CREATE INDEX dbadmin.LISTINGS_NSEXMLIDX FOR TEXT ON dbadmin.LISTINGS (NSEXMLIDX)
        FORMAT XML DOCUMENTMODEL listingsmodel IN /db2/model/listingsmodel
        INDEX DIRECTORY /db2/NSEindexdsk/MYDB/LISTINGS  CONNECT TO MYDB"
```

  - Prevents NSE from indexing format tags as if they were data.
  - Document Model allows you to specify which fields in your structured document are to be indexed and which are to be excluded from the index.
  - By removing data from the index that will not be searched, you can improve performance both indexing and searching.

Document Model Example

Given XML Document stored in DB2:

```
<?xml version="1.0"?>
<Customer>
    <Name>Sparky Automotive Inc. </Name>
    <Address>
      <Street>555 Main St</Street>
      <Locality>Charlotte</Locality>
      <State>NC</State>
      <Zip>28210<Zip>
    </Address>
    <Phone>800-555-5555</Phone>
    <Description>
      …
    </Description>
    <References>
       <Name>Billy Joe </Name>
      <Name>Bobbie Sue </Name>
    </References>
</Customer>
```

Document model for NSE indexing:

```
<?xml version="1.0"?>
<XMLModel>

    <XMLFieldDefinition
        name="$(PATH)"
        locator="/Customer"
        exclude="YES" />
    <XMLFieldDefinition
        name="$(PATH)"
        locator="/Customer/Name"
        exclude="No" />
    <XMLFieldDefinition
        name="$(PATH)"
        locator="/Customer/Address"
        exclude="No" />
    …
    … Every XML field should be defined …
    …
    <XMLFieldDefinition
        name="$(PATH)"
        locator="/Customer/References"
        exclude="Yes" />
    … <!--  Don't forget to exclude child elements-->
</XMLModel>
```

3 Search Interfaces Into NSE

- SQL Scalar Search Functions
  - Contains() = 1
  - Score() <-- Can be in select and/or where clause
- SQL Table-Valued Function
  - Table (DB2EXT.TEXTSEARCH())
  - Table (DB2EXT.HIGHLIGHT())
- Stored Procedure Search Functions
  - Call db2ext.TextSearch(…)

SQL Scalar Search Functions

Contains() and Score()

Advantages:

- Most functionality
- Good performance
- Can be used in arbitrary SQL statements

Disadvantages:

- Incurs a join against the primary key column to fetch rows. (Large results sets hurt performance.)
- Does not support text indexes on a view. For tables only.

SQL Table-Valued Function

Table (DB2EXT.TEXTSEARCH()) Table (DB2EXT.HIGHLIGHT())

Advantages

- Supports searching text indexes on views
- Can be used in arbitrary SQL statements

Disadvantages

- Must write query as a join against the primary key column to fetch rows. (Large results sets hurt performance.)

Stored Procedure Search

Call db2ext.TextSearch(…)

Advantages

- Fastest text search method. Performance and concurrency are insane.
- No join against primary key column. (Join is pre-calculated and cached with data.)
- Supports searching text indexes on views

Disadvantages

- No arbitrary SQL queries. Query runs against predefined cache table.
- Requires enough RAM to buffer entire selectable data set.
- Cache size limit: Selectable data set MUST be under 2 Gig total.

  Maximum Cache size depends on OS:

  AIX (32 and 64 bit): 1536 Meg
  Windows 32 bit: 1024 Meg
  Solaris (32 and 64 bit): 2048 Meg
  Linux (32 bit): 2048 Meg
  HP-UX (32 and 64 bit): 2048 Meg

Scalar Function Search Examples

Contains() = 1

- Select * from DEALERS where Contains(XMLDOCCOL, 'section ("PRODUCT") "Computer" | "PC" | "Desktop" | "Intel" | "AMD" ') =1

Score()

- Select VENDOR,
  Score(XMLDOCCOL, 'section ("PRODUCT") "Computer" | "PC" | "Desktop" | "Intel" | "AMD" ') as SCORE
  from DEALERS where
  Contains(XMLDOCCOL, 'section ("PRODUCT") "Computer" | "PC" | "Desktop" | "Intel" | "AMD" ') =1 and
  Score(XMLDOCCOL, 'section ("PRODUCT") "Computer" | "PC" | "Desktop" | "Intel" | "AMD" ') > 0.8

Multiple NSE Index Updates

- Don't update more indexes at one time than you have CPUs.
  Why?:
  - Index updates are resource intensive and will peg a CPU. Having more index updates than CPUs will NOT make the indexing faster but will increase the likely hood of NSE resource contention and index update failures.

- AIX with NSE 9.5 or lower: Don't update more than 4 indexes at once even if you have more than 4 CPUs.
  Why?:
  - NSE resource contention. Index update failures ***will*** occur.
  - 9.7 Resolves that issue

- Avoid using 'UPDATE-FREQUENCY' in you create index command if you have a lot of text indexes. (You should update your indexes manually if you have more than a few text indexes.)
  Why?:
  - Could cause NSE to fail to come on line after db2text start if too many indexes try to update as NSE starts up.

Search Performance

- Avoid multiple contains clauses if possible
- Score(): Always use Contains() when using Score()
  - Use the same clause within Contains() that you use for score()
- Numeric Attributes can be fast when numeric data needs to be searched with text.
- Limits can be double edged:
  - RESULT LIMIT (If you don't need all results)
    - Warning: Sometimes results in an un-indexed hash join if value is large so use cautiously and *watch* your access plans.
  - EXPANSION LIMIT (Can help you if small… or kill you if large)
    - Large expansion limits over can cause heavy memory usage. Exceed 100,000 with caution. Over 500,000 will cost you dearly in paging space and performance.

Expansion Limit: Wildcarding for example. A%

Attribute Usage

- If you are going to include a numeric non-NSE predicate in your where clause, consider creating an NSE attribute for it.

  In this example we want to find any LCD High Definition TVs in a price range between $500 and $2000 in our products table:

```
select * FROM products WHERE contains (NSEXMLIDX, 'SECTIONS ("/Product/Description") "HDTV  & "LCD" ') =1  AND
    PRICE between 500 and 2000;
select * FROM products WHERE contains (NSEXMLIDX, 'SECTIONS ("/Product/Description") "HDTV  & "LCD" &
    ATTRIBUTE "Price" between 499 and 2001 ') =1;
```

- Attribute Notes:
  - Attribute names are CaSe SenSitive
  - Syntax requires a between clause (no =, <, or >)
  - Maximum of 260,000 unique values matched by attribute clause. (This maximum is on unique values hit. It is *not* a row limit.)
  - Attribute values are **non-inclusive** so use -1 and +1 to desired value range. --This is different than the DB2 'between' operator.

Table Advise, Performance

- Primary Key
  - NSE requires a primary key exist on the table in order to create an NSE index. This is because NSE functions return a list of primary key elements which are then used by DB2 to fetch the rows matching the text search.
  - The primary key should be a binary based data type and should **NOT** be a composite index. (INTEGER or BIGINT are recommended.)
  - If your table does not have a column suitable for being a primary key consider an auto generated integer or bigint.

    Ex) create table test (RECID integer generated by default as identity, …)

- Text column to be indexed
  - Use Varchar data type when possible. (up to 32,672 bytes) Don't use CLOB or long varchar unless you absolutely have to as they will slow NSE indexing.

    Exception: If your document is for searching **ONLY** and will not be fetched, then you may gain search performance at the cost of indexing speed by using CLOB or long varchar…

Additional Composite Indexes

- Creating additional composite indexes that include the primary key's column may be advisable if where clauses include column predicates other than Text search.

  create table parts (PARTNUM bigint NOT NULL, PARTNAME varchar(20), PARTMAKE varchar(10), DESCRIPTION varchar(4000))

  select PARTNUM, PARTNAME from parts where PARTMAKE = 'FORD' and contains(DESCRIPTION, ' "Passenger" & "door" & "window" ')=1

  This select would benefit from an index on (PARTNUM, PARTMAKE):

  create index PARTSNUMMAKE on PARTS (PARTNUM, PARTMAKE)
  allow reverse scans

  Why?:

  - The contains clause will return a list of primary key elements (PARTNUM) to be fetched. Since the predicate PARTMAKE = 'FORD' must also be applied, the optimizer will generally select an index containing both PARTNUM and PARTMAKE columns if it exists, has up to date statistics, and decent cardinality.

Large Operation Considerations

- Beware: 'Update index' is incremental on NSE indexes
  - Incremental index update only when relatively small updates are being done to the base tables. <200k. Much beyond that and the incremental update may actually take longer than rebuilding the entire NSE index from scratch with an update index.
  - Never do an incremental NSE index update if you have had close to a million or more updates/inserts. Not only will the update take longer, it may never finish and hang. Use 'REORGANIZE' in the update index command for large updates.
- Large volumes of updates over time when using Update Incremental:
  - Regularly recreate your indexes using 'REORGANIZE' which actually does a complete index rebuild. This will clean up the index fragmentation that occurs with incremental index updates. This will improve search performance.
- If you are doing massive inserts or updates, drop the NSE index before processing the updates if you can. Then recreate and update the index after the updates finish.
  - Creating an NSE index on a table results in a trigger being created on the table as well when the first NSE update index is done.
  - Updating the base table results in the trigger firing. Small performance impact with some extra logging. (May or may not be a concern for large operations.)

Dropping A Table Having An NSE Index

- **Always** drop your NSE indexes first, ***before*** you drop a table!
- NSE indexes are NOT dropped automatically when a table is dropped. (Unlike traditional indexes.)
- If you drop a table with an NSE index, the Text index will remain eating up both disk space and preventing you from creating a new NSE index on that table if the table is recreated.

- Clean up if you drop a table with an NSE index on it:
  - **Option 1:** Disable the database for text using *'**force**'* and manually delete the index data on the file system. --This will remove **ALL** NSE index definitions. Yuck!
  - **Option 2:** Call DB2 Extender support. They can give you an undocumented NSE command to clean up a specific index.

NSE Error Handling

Many NSE errors report a generic message and refer you to an event view to get the error details.

- The following index update is one of 54 index updates executed serially:

  db2text UPDATE INDEX dbadmin.LISTINGS_NSEXMLIDX FOR TEXT REORGANIZE CONNECT TO MYDB

- The first 50 index updates succeeded but the 51st returned this error:

  ERROR: CTE0192 Errors occurred in an update index operation. Check event view "DB2EXT"."EVENTIX013001" and db2diag.log for details.

- Lets get the error message from the index's event table:

  Select TIME, MESSAGE from DB2EXT.EVENTIX013001 order by time

```
TIME                       MESSAGE
-------------------------- ------------------------------------------------
2006-02-18-20.30.02.252910 CTE0118 All available lock space for indexes on a database
                           is used. Change the configuration.
```

- Cause: sqllib/db2ext/db2extlm.cfg default of: maxIdxPerDb = " 50" was too small.

Finding the Right Event Table

Given the table dbadmin.LISTINGS how do we find the correct event table for the NSE index?

- Two views hold NSE text index information:
  - DB2EXT.TEXTCOLUMNS
  - DB2EXT.TEXTINDEXES

- DB2EXT.TEXTCOLUMNS:

```
Select INDEXSCHEMA, INDEXNAME, EVENTVIEWSCHEMA, EVENTVIEWNAME
  from  DB2EXT.TEXTCOLUMNS where
    TABLESCHEMA = 'DBADMIN' and TABLENAME = 'LISTINGS';
INDEXSCHEMA INDEXNAME     EVENTVIEWSCHEMA EVENTVIEWNAME
_____ _____ _____ _____

DBADMIN   LISTINGS_NSEXMLIDX DB2EXT      EVENTIX013001
```

Infinite Error Needs A Flush

Some errors wont go away:

*select * from dbadmin.LISTINGS where contains(NSEXMLIDX, 'section("/nse/Name") "tools" & "Sears"')=1*

Returns Error:

SQL0443N Routine "DB2EXT.TEXTSEARCH_1K16" (specific name "CTE21") has returned an error SQLSTATE with diagnostic text "CTE0198 No corresponding text index.". SQLSTATE=38798

Ops, no index (*duh*) so we create it:

db2text "CREATE INDEX dbadmin.LISTINGS_NSEXMLIDX FOR TEXT ON dbadmin.LISTINGS (NSEXMLIDX) …… CONNECT TO MYDB"

db2text "UPDATE INDEX dbadmin.LISTINGS_NSEXMLIDX FOR TEXT REORGANIZE CONNECT TO MYDB"

Re-execute our select and…

What? SAME ERROR?!!!

Why?:

- DB2 Package cache has cached the NSE error response and is returning the cached value. It is *not* actually re-executing the search.

Solution:

Flush the package cache after fixing index errors:

db2 flush package cache dynamic

NSE Configuration Parameters of Interest

Index resource parameters:

{HOME}/sqllib/db2ext/db2extlm.cfg

- Default:
  maxIdxPerDb = " 50"
  maxLocksPerIdx = "100"
- Rob's:
  maxIdxPerDb = "200" *How many indexes do *you* have
  maxLocksPerIdx = "600"

{HOME}/sqllib/db2ext/cteixcfg.ini

- Can be tuned depending on your environment for slight performance gain during indexing and searching. Some cannot be changed while indexes exit.
  RespectCase=OFF *If you don't need case sensitivity.
  MaskResolutionLimit= *May need to increase if you do wild carded searches

Memory Issues

- NSE relies on OS i/o cache for Text index caching
  - Make sure you have unallocated memory available for OS to cache i/o
  - Watch your server's paging space.
- NSE 9.1 AIX Paging space growth
  - Due to AIX default memory allocator
  - NSE APAR IC46495
    - Not a true fix, but it helps significantly
  - Later fixed in an NSE Fixpack by changing how NSE allocates memory.

Some paging space usage is to be expected when using NSE. However, if the system starts thrashing, you need to bounce DB2, bounce NSE. You should free additional memory to prevent it from happening again, or at least delay it.

Other Known issues

- NSE 9.5
  - Fixed in NSE Fixpack 1
    - Separate download and install from DB2 Fixpack 1
    - Uninstall NSE 9.5, then install NSE 9.5 Fixpack 1
- NSE 9.7
  - AIX Performance issue due to bad cardinality estimate coming back from NSE in explain.

Hacks Around Unsupported Features

Hacks: These generally require you create a special version of your document for indexing purposes

Position sensitive searching when you need to make sure only the first word is matched.

- Special character prefix on first word.

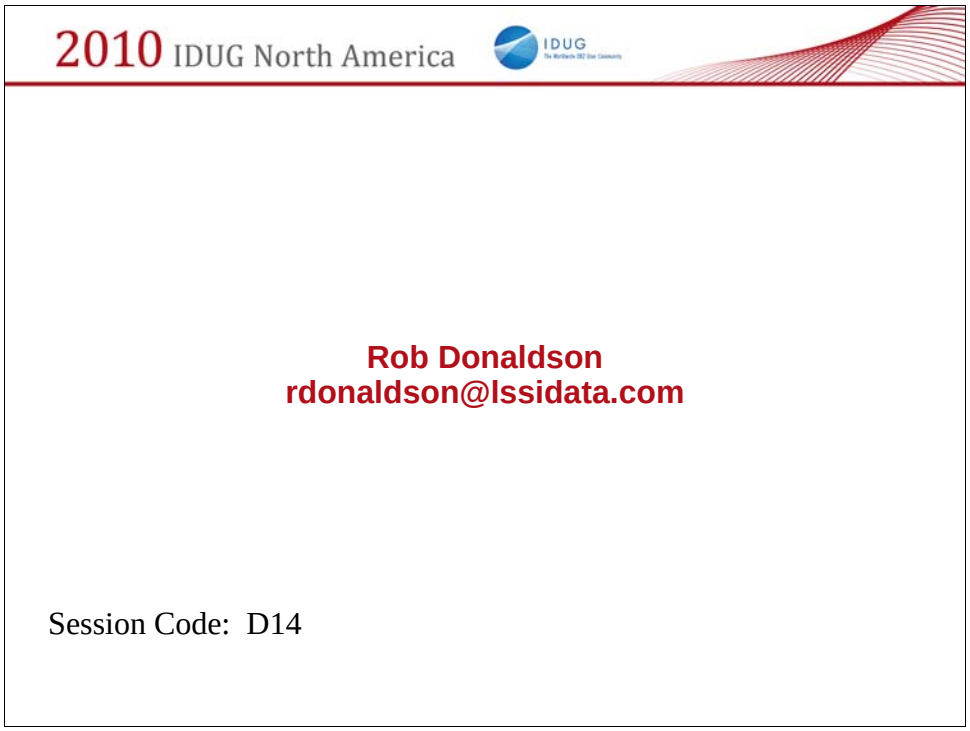
  ",International Business Machines"

  - Since NSE indexes words with and without special characters you search the above with "International" or ",International" and find it both ways.

NULL data matching to emulate 'is NULL'

- Populate a null indicator to indicate no data. Search on that indicator when you want NULL.

  "~"



Speaker Biography

Have worked with relational databases for over 13 years, predominately DB2 on UNIX, with occasional forays into Mainframe and Windows.9 Years experience working with DB2 text search products (including Net Search Extender).I've been at LSSiData for 9 years. I am a Database Architect with responsibility for the design an implementation of many of our Relation Database systems. I also have development responsibility for several of the C++ applications and gateways that accesses these databases. My background includes C/C++ software development, Database Administration, and UNIX System administration. Officer of TriDUG (Research Triangle, NC DB2 users group), IBM Data Champion 2009.Prior Speaker at IDUG and IBM IOD conferences. My hobbies include saltwater fish, wood working, and photography though not all at the same time.