

V6R1 INDEX ENHANCEMENTS

Some very important functions were added to the CREATE INDEX SQL statement in the recently announced V6R1 release. Interestingly enough, they may be more beneficial to RPG programs with native database access than for those using SQL – at least in the short run. While IBM continues to invest almost exclusively in SQL rather than DDS, the new index functions are intended to not only help performance but make it possible to replace logical files with SQL objects.

Prior to V6R1, moving completely from DDS file definitions to SQL was impossible without rewriting at least part of the application. Because logical files had a specific record format, a specific set of fields, and (optionally) specific select/omit criteria, simulating the same function with SQL was not possible. With V6R1 a majority of logical files can now be implemented with an SQL index.

Record Format Definition

Previously, when an SQL index was defined, you could not specify the

record format name or limit the columns from the table that go into that format. In order to allow administrators to replace DDS logical files with SQL indexes, it was necessary to explicitly control the format name as well as the columns in the format so that SQL indexes

would produce the same format identifier as the equivalent logical file. With full control over the format, a significant number of logical files can now be replaced by SQL indexes and, very importantly, be done without requiring the application programs that use those logical files to be recompiled. Here is an example of the new syntax to support the record format and field list:

```
CREATE INDEX custindex1 ON
customer (cust_id asc,
cust_zip asc)
RCDFMT custfmt1 ADD
cust_id, cust_fname,
cust_lname, cust_addr1,
cust_addr2,
cust_zip, cust_st
```

The RCDFMT keyword sets the name of the format and the ADD keyword specifies which columns should be included in that format.

Sparse indexes

Another interesting enhancement is the introduction of “sparse” indexes in SQL. Prior to V6R1, if you wanted to create an index with a subset of the table’s rows represented you had to create a select/omit logical file. There was no equivalent type of index in the SQL world. Internal to the query optimizer, temporary sparse indexes were often created but the database administrator could not create a permanent one.

In the context of SQL, select/omit logical files have been problematic. If the SQL programmer wanted to use a select/omit file directly and “push” the select criteria into the file, the query would be forced to use the old, slower

query optimizer (CQE). While functional, both the performance of the query and the overall system would suffer. Secondly, even if a select statement accessed the physical file directly, the query would by default use CQE because the new query optimizer (SQE) did not know how to handle select/omit logical files.

In V6R1, SQE will process queries even if there are select/omit logical files defined over the table (it will still use CQE if a logical file is accessed in an SQL statement’s FROM clause). The bottom line is that the select/omit logical files can now be converted to use SQL if they are not using DDS functions like zone/digit force or absolute value.

One nice advantage of the new support is that, at least in this author’s opinion, the SQL syntax is easier to understand. Furthermore, because the full WHERE clause provides a much richer set of choices making it possible to define the “logical file” you only wish you could have created in the past.

Unfortunately, the query optimizer still can not take advantage of these new types of indexes to improve performance. Hopefully this will change in the next release. On the other hand, RPG programs can see performance benefit from using an SQL index. Most select/omit logical files have an 8K logical page size. An equivalent SQL index will have a 64K logical page. If RPG programs process the file sequentially, replacing the logical file with an SQL index will reduce disk I/O and thus improve the elapsed time of

Inside this issue:

V6R1 Index Enhancements	1
Run SQL Scripts the oft overlooked solution	2
CTO Memo	3
DB2 Web Query Defining Metadata	3
Visual Explain the history and the practice	7

(Continued on page 12)



Run SQL Script

the oft overlooked solution

Most IBM i users are familiar with the interactive SQL (STRSQL) environment, part of the 5722ST1 DB2 Query Mgr and SQL DevKit product. It is easily accessible to a green-screen user with 5250 emulator, supports context sensitive prompting via F4 key as well as online help via F1 key.

But what if a business policy prevents you from loading that product? Or you want to test your stored procedures returning result sets? Or you want to use a graphical debugger to debug SQL stored procedures? Or you need to tune the performance of a specific SQL statement using Visual Explain? Or you simply want more reliable and easier way to save your SQL statement history?

Enter iNav's Run SQL Script feature.

Invoking Run SQL Scripts

There are multitudes of ways to invoke Run SQL Scripts. I'll offer several methods, and you pick the one that suits you the best. Of course, prerequisite for all of them is that you have iSeries Client Access installed on your PC.

1. Start iNavigator and expand *<system name>* → *Databases* → *<database name>*. At this point you can either right-click on the *<database name>* icon and select *Run SQL Scripts...* menu option or simply left click on the *Run an SQL Script* icon located in the task pane in the lower right corner.
2. Click on *Start* → *Run* → *cwbundbs* → *OK*
3. Hit key combination (*window key* → *R*) → *cwbundbs* → *OK*
4. Create a shortcut to *C:\Program Files\IBM\Client Access\Shared\cwbundbs.exe* and place it on your desktop
5. Use Notepad to create a file with an *.sql* file extension and simply double-click it

Running an SQL statement

Running an SQL statement is very easy with Run SQL Scripts. Here's the most basic example:

- `SELECT * FROM QADBREF`
- `Ctrl` → `Y`

That's it! You've just run a query over system's database cross-reference file and it was displayed on your PC.



Trust. It's **the** key to any decision about who you do business with. Building trust in today's environment is very difficult however, simply because the time to develop relationships that foster trust is so limited. In addition to that, we are also bombarded with messages from the media and from our own e-mail that tell us there are plenty of people in our society not worthy of our trust (how many offers for cheap "Rolex" watches have ended up in your spam folder recently?).

Recognizing the challenge, Centerfield is focused on building the trust of people we work with by:

Listening first and listening carefully to our customers and potential customers. Every iSeries account has a unique set of challenges and opportunities. Our first priority is to understand your needs and focus our discussion on those objectives rather than pushing a product or service you don't need.

Honestly assessing where we can help you and, maybe more importantly, where we can't. We believe that it is in everyone's best interest to be up front about our ability to solve your problems. If we can't address your needs, we'll try to put you in touch with someone who can. Centerfield has a deep contact base in the iSeries world and can more often than not give you great advice on what to do next in your specific situation.

Showing you rather than telling you. The best way to evaluate software or service is to see it in action rather than being told how great it is. It's always fun to hear the phrase – "Wow, I didn't know it was that easy." To achieve the show- rather-than-tell goal, we rely on live internet demonstrations, software evaluations, and ongoing education – all designed to fit into your schedule not ours.

Value-add communication. Marketing material produced by most companies doesn't have a lot of real-world value. At Centerfield, we recognize your time is very valuable and try to provide useful information every time we interact. This newsletter is a good example of that effort – you get great information and we show you the ways that our products and team can help yours.

As you go back to work and feel the list of to-dos getting longer and longer – feel free to call us and discuss the possibility of letting our team help out! We'll work hard to build your trust.

Best Regards,

Mark L. Holm
Chief Technology Officer
Centerfield Technology, Inc.

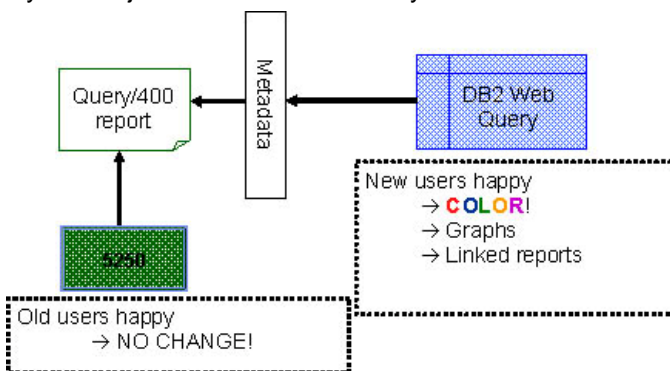
DB2 Web Query for System i

DEFINING METADATA

Many System i shops are looking at the possibility of using DB2 Web Query as a way of to enhance and/or replace their existing Query/400 reports. The purpose of this article is to introduce the topic of importing Query/400 objects into the DB2 Web Query product.

When a shop begins to use DB2 Web Query, the first step will typically be to build on top of their existing Query/400 reports. The simplest way to do this is to import the query definition into the DB2 Web Query product. This is done by defining "metadata", in other words data about data." Essentially the metadata provides information to the DB2 Web Query product so that the Query/400 report can be run and the data displayed the web browser interface instead of a 5250 session.

The following graphic shows the relationship between the Query/400 object and DB2 Web Query.



If there are users that prefer to use their existing menus or commands to run Query/400 reports they can continue to do so. Once the query is imported into Web Query, users can run the same report using the same query definition. The good news is that DB2 Web Query allows the old query to be beautified and enhanced with color, graphics, and drill down reports. The bottom line: users that don't want to change are happy and those that want a newer interface are happy.

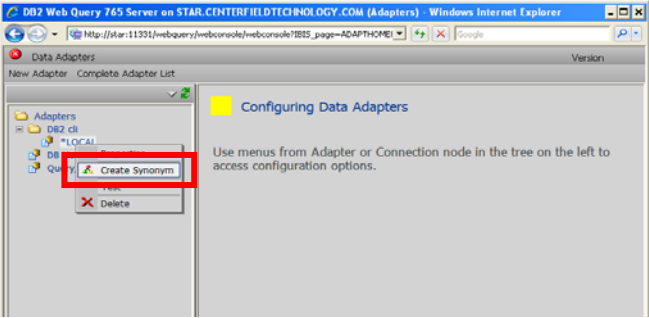
So let's go through the process of importing Query/400 objects (*QRYDFN) into the Web Query product.

After signing into Web Query, you will see a screen similar to the following:

Metadata definition
Import table, view, file definitions
Import Query/400 reports
Define virtual fields, headings, etc. (With developer workbench)

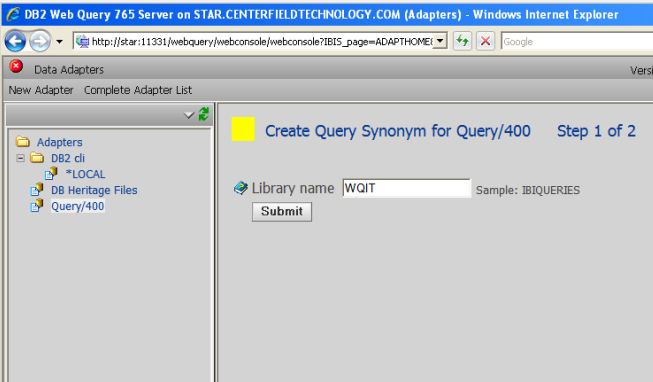
On the left side of the screen are domains (high level containers – like a library) and folders. Right-clicking on the folder will bring up a menu that includes an item called **Metadata**. The metadata function is used to import Query/400 objects but also to define file and field definitions for brand new reports you might create in Web Query.

If you choose the **Metadata** menu item, the following screen will appear.

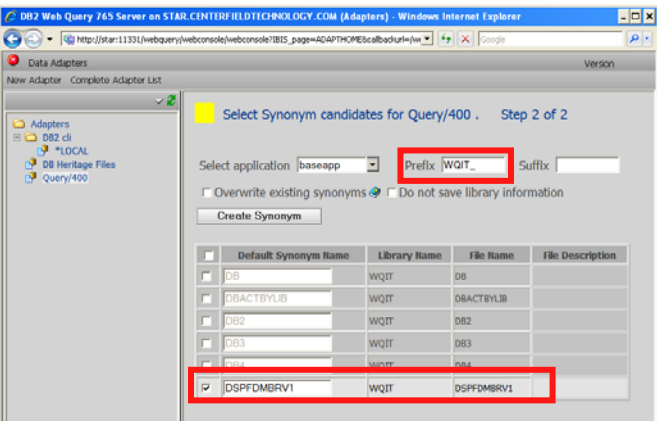


To import a query definition, right click on the Query/400 item in the tree. This will bring up a submenu with **Create Synonym**. Don't be confused by the use of the term "synonym." The words metadata and synonym mean the same thing to DB2 Web Query.

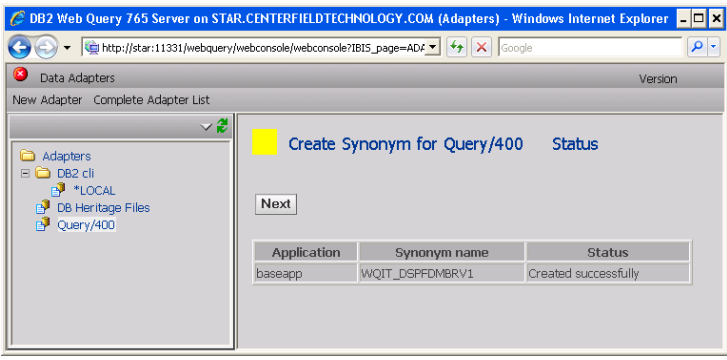
The next screen will let you choose which libraries contain your Query/400 objects. Simply type in a name and click on **Submit**.



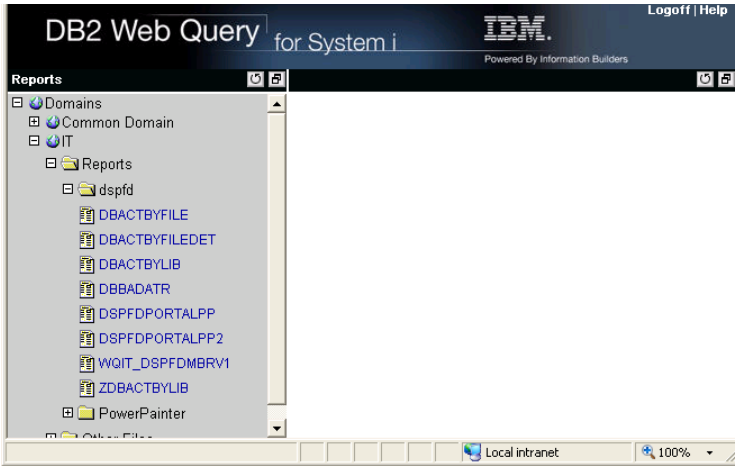
The next screen allows you to choose which reports to import into Web Query. The ones you choose to import should be checked as seen with DSPFDMBRV1 below. The other item that is filled in is the **Prefix** box. The prefix string will be pre-pended to the Query/400 name. While it is an optional part of the process, it is a good idea to add a prefix using the library name so that at a later time you know where the source query object is located. If you leave the prefix off, the query will be imported correctly but it may be difficult to identify the original query if you have duplicate query names on your system.



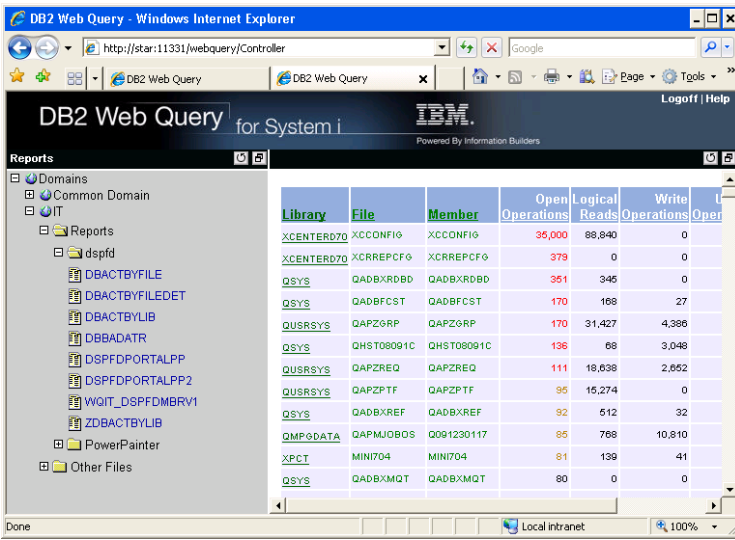
The final screen will show the results of the import:



As you can see the **WQIT_DSPFDMBRV1** synonym has been created. At this point the query can be run from the user interface and it will return the same data as the Query/400 report. [NOTE: There are certain functions that are not imported directly, but simple reports work fine. See the latest IBM documentation for a list of the features that can not be imported from Query/400.]



If the WQIT_DSPFDMBRV1 report is clicked the Query/400 object will be run and the data put into the window on the right similar to the following:



As you can see, it is relatively simple to import an existing Query/400 object into DB2 Web Query.

This example query has been modified to have color, drill through, and conditional styling (stop lighting). These are ways to enhance the existing query without any changes to the *QRYDFN. We'll cover more of these techniques in future articles.

Mark

Modifying connection details

Most experienced IBM i users are used to *SYStem naming, rather than Run SQL Scripts default of *SQL naming. If you'd like to alter that frequently requested setting, click on:

Connection →JDBC Setup... →Format →alter the Naming convention: dropdown to System (*SYS)

You can of course alter other settings, such as to translate data tagged with 65535 CCSID (*HEX).

By modifying the naming convention you will avoid the error on qualified objects. Alternative is to qualify the object with naming appropriate separators (dot for *SQL or slash for *SYS naming).

Running multiple statements

In order to run multiple statements in the script, simply separate them using the semicolon (;) separator. You can have as many statements as you'd like in this script. For example:

```
SELECT * FROM QSYS/QADBXREF;
SELECT * FROM QSYS2/SYSTABLES;
```

To execute the entire script, press key combo *Ctrl* →*R* or click on *Run* →*All* menu option.

Adding comments

As any good software engineer would tell you, you need to document your code. SQL scripts are code as well and you can document your intent by prefixing your comments with double dashes (--) or if you have a multiline comment you can use traditional start comment (/*) and end comment (*/) enclosures.

```
-- this is a single line comment
/* this is a
multiline comment
*/
SELECT * FROM QSYS/QADBXREF;
```

Running CL commands

You can run CL commands by prefixing them with a *CL:* prefix. Here's an example:

```
CL: DSPFD FILE(QSYS/QADBXREF) TYPE(*MBR) OUTPUT
(*OUTFILE) OUTFILE(QTEMP/OUTFILE);
SELECT MBNRCD FROM QTEMP/OUTFILE;
```

If you run the script above, you'll get the count of files on your system as output.

Prompting

One of the chief reasons people like STRSQL command is its interactive prompting feature. Prompting has taught many a

newbie the correct SQL syntax to use in their statement. Run SQL Scripts also supports prompting, both for SQL statements and CL commands!

SQL prompting is supported by multiplatform visual component called SQL Assist. CL prompting is supported by generic JT400 CL command prompter component.

Hit F4 function key on either SQL or CL command and the appropriate prompt assistant will pop-up and take you through the steps of completing that command.

Saving the script

I've written before about the difficulty of saving the statement history using STRSQL (see "STRSQL Conundrum" article) and retrieving it in a reliable and an easy to use fashion.

With Run SQL Script you have a totally intuitive and easy to use interface for saving your scripts. Simply click *File* →*Save* or hit *Ctrl* →*S* key combo as you do with any other Windows application. Script is saved with .SQL file extension which can be opened by simply double-clicking on the file in any Windows Explorer folder.

Scripts can be saved to your PC drive or any other shared network drive (i.e. IFS of your IBM i system).

Testing Stored Procedures

Have you've ever found yourself in a situation where you wanted to test a stored procedure returning a result set that you or your developers have written and didn't know where to go? STRSQL does not work, RPG can't handle them (other than through not so simple CLI methods), nor can RUNSQLSTM, STRQMQRV or any other variety of SQL tools normally used on the IBM i. There are 3rd party tools, but you need to download and install them and they still require an ODBC or JDBC or .NET driver be installed using iSeries Client Access install. Since you have to have some sort of data source provided by iSeries Client Access, why not use (free to you) Run SQL Scripts?

It outputs the result set to a grid in the Run SQL Scripts window. You can configure it to instantiate the output grid in a separate window. You can copy/paste the result set to Excel using *Ctrl* →*C* and *Ctrl* →*V* keyboard combos. Here's a sample SQL stored procedure that you could use in your experimentation phase:

```
CREATE PROCEDURE sampleProc (library VARCHAR(10))
RESULT SETS 1
LANGUAGE SQL
BEGIN
  DECLARE myCursor CURSOR FOR SELECT * FROM
QSYS/QADBXREF WHERE DBXLIB = library;
  OPEN myCursor;
END;
CALL sampleProc('QSYS');
```

(Run SQL Script, the oft overlooked solution — Continued from page 5)

Debugging

As much as I'd like to say that when you write SQL stored procedures you will never make a mistake, it's simply not true. So when you do encounter a bug in your stored procedure, how do you debug it? You don't want to go through the computer generated C program on the IBM i server, do you? I know I don't want to. That's where the graphical iSeries System Debugger comes in.

First, you'll need to add the following option to your CREATE PROCEDURE script and recreate your stored procedure:

SET OPTION DBGVIEW=*SOURCE

You should place it just before the BEGIN indicator. I normally compile all of my stored procedures with this option set, so I don't have to recreate or ALTER them later when I need to debug them.

Next, click on the *Run* → *Debugger* or simply hit *Ctrl* → *D* key combination.

The beauty of this debugger is that it is fully visual (graphic) and allows you to step through the SQL statements instead of generated C code. As you step through the SQL statements, you can examine the variable values and figure out what's wrong with the stored procedure logic.

Visual Explain = Performance Improvement

Hands down, the best tool for tuning a specific SQL statement is IBM's Visual Explain (Centerfield tools and staff excluded). Once you have your SQL statement in the Run SQL Scripts window and you're not happy with its performance, simply click on *Visual Explain* → *Explain* or hit *Ctrl* → *E* key combo to invoke Visual Explain.

Visual Explain uses database monitor support on the server, the best and most detailed source of query optimizer costing decision statistics, and draws a visual diagram representing the access plan query optimizer created for this SQL statement. All of the steps are described in great detail.

First thing you should look at are what query engine is your SQL statement dispatched too (SQE or CQE). If CQE, you need to figure out why, as that query engine is in maintenance mode only and all of the cool performance and functional enhancements for the past 10 years have been in strategic SQE query engine.

Next thing you should look at is if there are any advised indexes by clicking on *Actions* → *Advisor* menu option or associated shortcut icon.

If after removing any SQE restrictions and building advised indexes your SQL statement still does not perform to your specification, consider modifying your SQL statements slightly, perhaps to leverage Common Table Expressions (CTEs), in an effort to influence the query optimizer to recreate its access plan.

V6R1 enhancements

Latest version of the IBM i operating system brings us some enhancements that are worth mentioning. The inherent grid used in all of the iNavigator output now has options to export data to Excel file, as well as other output options.

More importantly, RUNSQLSTM IBM i operating system command has been enhanced to accept IFS stream file input. That means that you can now build and test an entire SQL script using Run SQL Scripts, save it to the IFS and schedule a job to execute it on a regular basis using RUNSQLSTM command!

IBM iNav Run SQL Scripts offers a number of unique functions that may aid you in your daily administrative duties. Hopefully this summary exposes you to some functions that you need and you were not aware are available to you at no additional cost.

Elvis

Free Offer

Ask for your free copy today
jen@centerfieldtechnology.com

2008 COMMON EDITION
DB2 Web Query

call Centerfield 888.357.8119

New Services for DB2 Web Query

Phase 1: **ICV Service (Install, Configure, Verify)**
 Basic service - Introductory Priced

Phase 2: **The Query/400 Conversion Service**
 We take up to 20 Query/400 reports and integrate them into the DB2 Web Query environment to leverage the web interface.

Phase 3: **Report Enhancement Service**
 Build on advanced DB2 Web Query function and improved performance with the new query engine; while giving visibility to upper management with dynamic, graphical and powerful reports.

Centerfield announces new DB2 WQ services and free tool

Centerfield Technology has unveiled a DB2 Web Query consulting service

*****DBIT 10005

CENTERFIELD TECHNOLOGY
 3131 SUPERIOR DRIVE NW
 ROCKFORD, MN 55955
 WWW.CENTERFIELDTECHNOLOGY.COM

With Centerfield, DB2 Web Query Installation is Fast and Affordable

VISUAL EXPLAIN

the practice and the history

Back in 1998, Centerfield Technology Inc. developed a suite of tools called Database Essentials. Part of that suite was a tool called SQL Visual Explain. They say that “imitation is the sincerest form of flattery” and lo and behold, couple of years later IBM came out with a component of iSeries Navigator also called Visual Explain. Navigator’s Visual Explain provided similar function to Centerfield’s version, but was built with the idea of showing much more technical detail.

Unfortunately, because IBM’s Visual Explain was built into iSeries Navigator, the adoption rate was relatively slow (the first users encountered performance and functional issues which discouraged widespread use). For those reasons, as well as general unfamiliarity with the tool, the use of iSeries Navigator was, and still is, surprisingly low.

I am here to tell you that the time to adopt Visual Explain is now!

Invoking the tool

As usual, there are a number of ways to invoke Visual Explain:

- 1) from Run SQL Scripts iNav feature, place a cursor on the statement of interest and click on *Visual Explain* → *Explain* or alternatively, *Visual Explain* → *Run and Explain* menu options
- 2) from SQL Performance Monitor data (aka imported database monitor file) by right-clicking on the monitor/file and selecting *Show Statements* → *Retrieve* → *click on statement of interest to highlight it* → *Run Visual Explain*
- 3) right-click on *<database name>* → *SQL Plan Cache* → *Show Statements...* → *Retrieve* → *click on statement of interest to highlight it* → *Run Visual Explain*

There may be other methods to invoke it, but those are the chief ones I use on a regular basis.

In the following sections I will outline three straightforward, real-world, practical scenarios of Visual Explain utilization to remedy an SQL performance issue. My hope is that by giving you the taste of how easy it can be to correct a performance issue, you’ll realize just what a powerful tool Visual Explain can be.

Set up a sample database

IBM provides a nice stored procedure to create a sample database used for documentation and examples. If you feel like trying out some of the scenarios I’m going to outline in this article, complete following two steps:

- 1) click on *Start* → *Run* → *cwbundbs* → *OK* to invoke Run SQL Scripts iNav tool
- 2) `CALL QSYS/CREATE_SQL_SAMPLE('VEXPLAIN');`
- 3) Click on *Run* → *Selected* or *Run* → *All* (since there is only one statement in the script at this time)

This will create an SQL schema (aka library) with all of the objects necessary for IBM illustrations, along with inherent SQL catalogues and journal created for the SQL schema.

Note that I use VEXPLAIN as a schema name, but you are free to use whatever name suits you.

Do NOT close the Run SQL Script window for the duration of this illustration.

Practical scenario #1 --- missing index

You've found a poorly performing SQL statement. Now you want to analyze it in Visual Explain and see if there is anything you could do about it, with goal being to improve its performance.

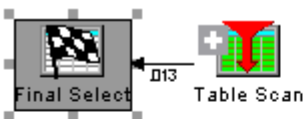
The first (obvious) choice is to see if there are any query optimizer advised indexes you could build and test to see if the performance gets better or not.

My poorly performing statement is:

```
SELECT * FROM VEXPLAIN/EMPPROJECT WHERE EMENDATE = DATE('1982-09-15');
```

I put a cursor anywhere on this statement and click *Visual Explain* → *Run and Explain*. In this case I prefer *Run and Explain* over just *Explain* since I know that sample database is small and I won't have to wait long. Also, *Run and Explain* gives me more accurate statistics since the statement is truly executed, rather than relying on query optimizer estimates alone.

I receive the following diagram:



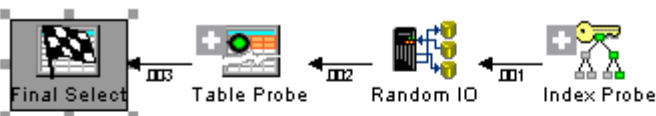
Final Select	
Timestamp for Creation of Monitor Entry	2008-04-22-15.46.40.028447
Run Time, in Microseconds	42384
Rows Fetched	2

Even though I have selection criteria in my statement (the WHERE clause), DB2 opted to use a table scan implementation. I think that query optimizer could utilize an index to weed out the rows that don't satisfy the selection criteria, so the first thing I am going to check is if the query optimizer has advised any indexes. I click on *Actions* → *Advisor* menu option and am pleasantly surprised with index advice over the EMENDATE column:

The screenshot shows the 'Index and Statistics Advisor - Star(Star)' window. The 'Index Advisor' tab is selected. It displays the recommendation: 'It is recommended that the following indexes be created:'. Below this is a table with the following data:

Create	Table Name	Schema	Index Type	Columns
<input checked="" type="checkbox"/>	EMPPROJECT	VEXPLAIN	Binary Radix	EMENDATE

Next, I click on the Create button, type in the name of the index as EMENDATE_INDEX and click OK. After that, I re-invoke the Visual Explain over my original SQL statement and receive the following plan:



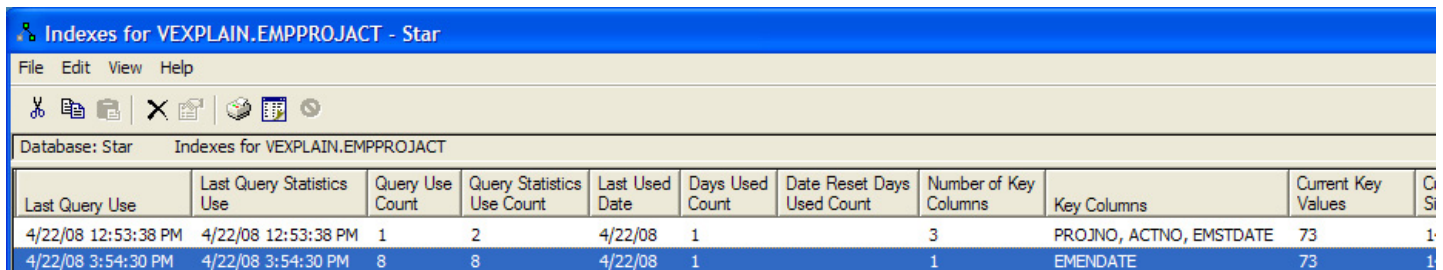
Final Select	
Timestamp for Creation of Monitor Entry	2008-04-22-15.50.07.029706
Run Time, in Microseconds	12168
Rows Fetched	2

Yes! I built an index and query optimizer is actually using it. Furthermore, total runtime dropped by a factor of **3.5!**

If I wanted to assure myself and my colleagues that this index is indeed being used by applications querying this table, I could use yet another iNav tool to verify that - Show Indexes. To invoke it follow these steps:

- 1) under Databases, *right-click on Schemas* → *Select Schemas to Display* → *VEXPLAIN* → *Add* → *OK*
- 2) expand *Schemas* → *VEXPLAIN* → *Tables* → *right-click on EMPPROJECT table* → *Show Indexes*

Here are some of the select statistics informing you how many times the index was used for query implementation and/or query statistics. Notice that in my example I'm already up to the count of 8, so I know that my index is being used by other processes as well!



Last Query Use	Last Query Statistics Use	Query Use Count	Query Statistics Use Count	Last Used Date	Days Used Count	Date Reset Days Used Count	Number of Key Columns	Key Columns	Current Key Values	Ci
4/22/08 12:53:38 PM	4/22/08 12:53:38 PM	1	2	4/22/08	1		3	PROJNO, ACTNO, EMSTDATE	73	1
4/22/08 3:54:30 PM	4/22/08 3:54:30 PM	8	8	4/22/08	1		1	EMENDATE	73	1

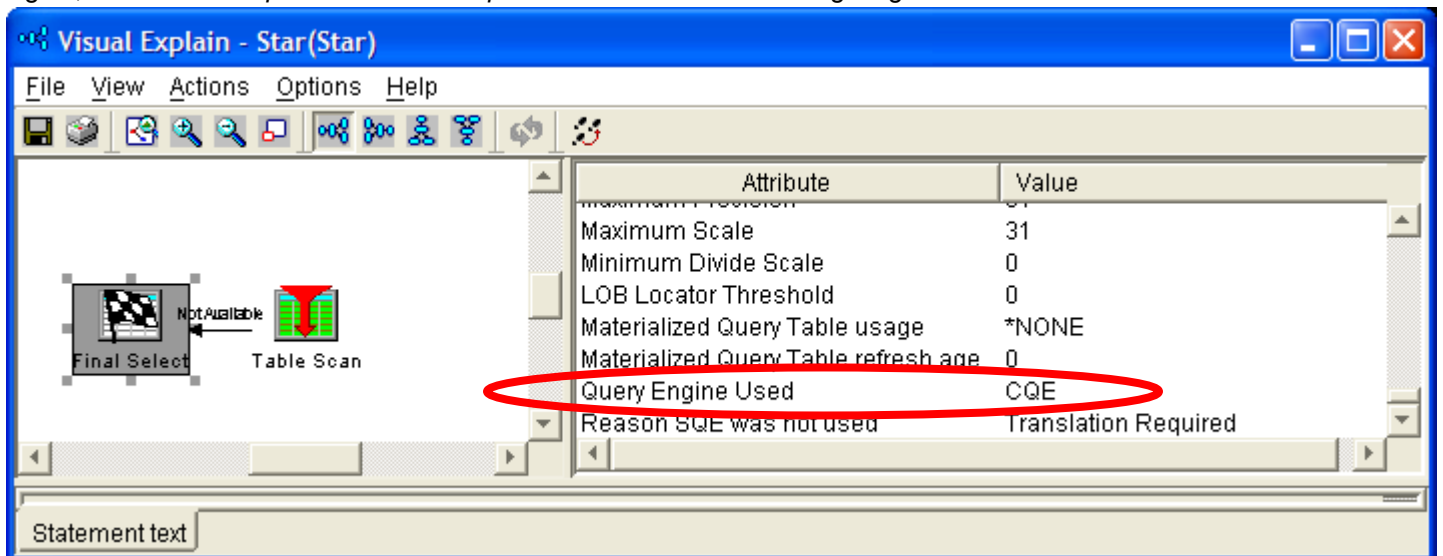
There you go, the case of the missing index is closed to everyone's satisfaction.

Practical scenario #2 --- the old query engine is being used

I knocked off the case of the missing index and am off to another poorly performing SQL statement:

```
SELECT * FROM VEXPLAIN/DEPARTMENT WHERE LOWER(DEPTNAME) = 'branch office f2';
```

Again, I use *Visual Explain* → *Run and Explain* and receive the following diagram:



The screenshot shows the 'Visual Explain - Star(Star)' window. On the left, there is a diagram with a 'Final Select' icon and a 'Table Scan' icon. An arrow points from 'Table Scan' to 'Final Select', with the text 'Not Available' above it. On the right, there is a table with 'Attribute' and 'Value' columns. The 'Query Engine Used' attribute is circled in red, with the value 'CQE'. Below it, the 'Reason SQE was not used' attribute has the value 'Translation Required'.

Attribute	Value
Maximum Scale	31
Minimum Divide Scale	0
LOB Locator Threshold	0
Materialized Query Table usage	*NONE
Materialized Query Table refresh age	0
Query Engine Used	CQE
Reason SQE was not used	Translation Required

With this statement there is no index advised, so the next place I look at is what query engine it's been dispatched to. I click the Final Select icon to activate it and scroll down to the bottom of the Attribute-Value pane on the right. There I find that the query engine being used is CQE and the reason is that Translation was Required.

As you may be aware, the CQE (Classic Query Engine) has been in maintenance mode for some time and all of the functional and performance enhancements have been concentrated in the SQE (SQL Query Engine) only. SQE is IBM's strategic query engine so if at all possible we want to make sure our queries run under SQE engine.

I can't really force the query optimizer to use one engine or the other since a component of the optimizer called the Query Dispatcher makes that decision at runtime. I can however make sure any SQE restrictions are not in place at the time I run my query.

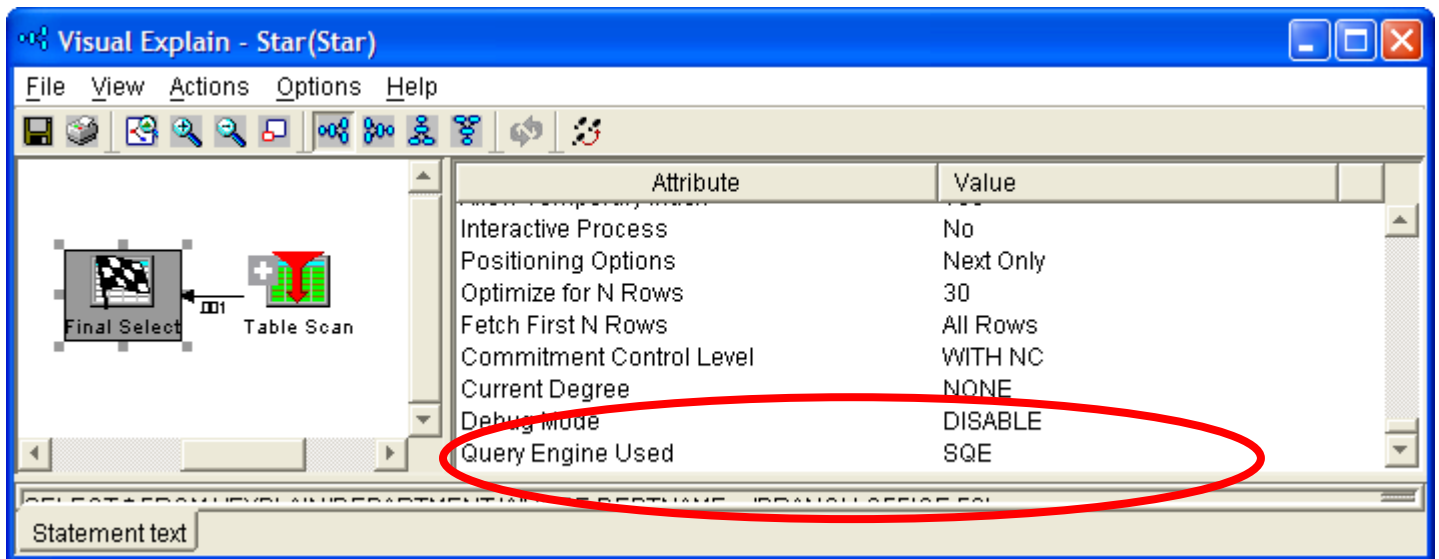
I look again at my query and deduce that the only possible translation that could be occurring is through the use of IBM's LOWER built-in function. Is it possible to remove the use of that function and still get the results I want?

I examine the data in the DEPTNAME column of DEPARTMENT table and am pleasantly surprised that all of the department names are inserted using capital letters.

I rewrite my statement as such:

```
SELECT * FROM VEXPLAIN/DEPARTMENT WHERE DEPTNAME = 'BRANCH OFFICE F2';
```

As usual, I re-explain the statement and receive the following diagram:



Notice that the query engine being used this time is SQE!

Not only that, when I check for advised indexes, I find one was advised over DEPTNAME column and so I build it. After explaining the statement again, I notice the implementation has not changed from the table scan. I look at the Show Indexes information and notice that the index I built was considered by the query optimizer and used for query statistics. However, with number of rows in the table being less than 30 (it was 14), the query optimizer wisely decided that table scan is still a better implementation option. However, if this table was to grow considerably in the future, I am certain that this query would start using the index I built for query implementation as well.

There are a number of other reasons that can cause the query dispatcher to route your query to CQE. Chief among them is specifying a logical file in the FROM clause of your SQL statement. Even at V6R1 this will route your query to CQE. Fortunately the remedy is easy. Simply target the physical file in your FROM clause and let the query optimizer figure out which indexes (aka keyed LFs) to use for implementation. In fact, this is so important that I am going to implore you to put it in your "best practices" bucket – always query physical files, never logical files in SQL statements.

Another common SQE restriction is the existence of Select/Omit logical files on physical file being queried. With V5R3 and V5R4 you could instruct the query optimizer to ignore these Select/Omit LFs by setting the IGNORE_DERIVED_INDEX QAQQINI value to *YES. The good news with V6R1 is that IBM is setting that option by default.

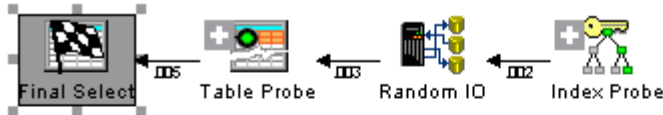
Bottom line is, after removing any existing SQE restrictions you can watch your queries fly!

Practical scenario #3 --- implicit query rewrite causing unforeseen issues

Similar to scenario #1, I Explain the following query:

```
SELECT * FROM VEXPLAIN/EMPLOYEE WHERE EDLEVEL = 12;
```

I find that an index was advised over the EDLEVEL column so I build it. Then I re-explain the query and as I expected, query is using an index this time:



Final Select	
Timestamp for Creation of Monitor Entry	2008-04-22-16.53.08.077575
Run Time, in Microseconds	18896
Rows Fetched	3

The Index Probe icon is expected, but I wanted to verify that query optimizer is using my index directly, without resorting to any sort of internal casting. To verify that:

- 1) click on *Index Probe* icon
- 2) scroll down to the bottom of the Attribute-Value pane
- 3) double-click *Statement Text* value to see the detail representation of the internal query rewrite for that particular step

Much to my chagrin, I find that the query engine is indeed casting my column values from a 2 byte column data type (SMALLINT) to an INT:

```
SELECT DselDField(EDLEV00001) FROM Radix Index(VEXPLAIN/EDLEVEL_INDEX)  
WHERE (Cast(EMPLOYEE_1.EDLEVEL, 2-byte int) AS 4-byte int)=12
```

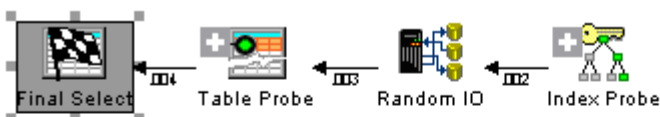
The reason this is happening is that DB2 implicitly converts any numeric literals without decimal points to an integer. As a result the literal 12 is interpreted as an integer value. Then our artificial intelligence friend, the query optimizer, decides to cast each value in EDLEVEL column to an integer, instead of doing the reverse (cast 12 to a SMALLINT).

While I know query optimizer is very smart, this is an example of why humans still have a place in this business.

I rewrite my query to suit the query optimizer better (in general, the more information you can provide the query optimizer, the better):

```
SELECT * FROM VEXPLAIN/EMPLOYEE WHERE EDLEVEL = SMALLINT(12);
```

I re-explain the statement, and receive an identical diagram, except notice the runtime difference – it's faster by a factor of **2.5!**



Final Select	
Timestamp for Creation of Monitor Entry	2008-04-22-17.05.40.692152
Run Time, in Microseconds	7432
Rows Fetched	3

(Visual Explain the practice and the history — Continued from previous page)

I look at the Index Probe internal statement:

```
SELECT DseIdField(EDLEV00001) FROM Radix Index(VEXPLAIN/EDLEVEL_INDEX)
WHERE EMPLOYEE_1.EDLEVEL=(Cast(12, 4-byte int) AS 2-byteint)
```

Obviously, cast is taking place in the right place this time, on a numeric literal rather than the column value.

The reason I bring up this scenario is that I have seen implicit casting disallow the use of an index in the past. You can imagine the runtime difference between a table scan implementation and index implementation over a multimillion row table. Needless to say, this little tweak caused a dramatic improvement and allowed my customer to get another feather in his cap.

Visual Explain is the best tool to use for performance tuning of a specific SQL statement. It offers a wealth of information, as long as you're able to intelligently interpret it. I have been using it with great success to improve performance of our commercial applications as well as to perform SQL and database performance consulting on customer systems.

For a graphical demonstration of Visual Explain in action, click on Visual Explain link on this webpage:

http://www-03.ibm.com/systems/i/software/db2/prod_admin.html

For more reading, refer to:

SQL Performance Diagnosis on IBM DB2 UDB - <http://www.redbooks.ibm.com/abstracts/sg246654.html?Open>

OnDemand SQL Performance Analysis Simplified on DB2 for i5/OS in V5R4 -

<http://www.redbooks.ibm.com/abstracts/sg247326.html?Open>

Database performance and query optimization whitebook (most detailed), as well as Preparing and Tuning the V5R2 SQL Query Engine on DB2 UDB for iSeries - <http://publib.boulder.ibm.com/infocenter/iseries/v5r4/topic/rzajq/>

(V6R1 Index Enhancements—Continued from page 1)

the job.

Here is an example of the new syntax building on the previous example:

```
CREATE INDEX custindex1 ON
customer (cust_id asc, cust_zip
asc)
WHERE cust_st IN ('CA', 'NY',
'FL', 'TX', 'MN')
RCDFMT custfmt1 ADD cust_id,
cust_fname, cust_lname,
cust_addr1, cust_addr2,
cust_zip, cust_st
```

Expressive keys

Another major change is that keys can be expressions and not just plain old columns. For example, you may want to build an index over a calculated key field such as QUANTITY * PRICE AS TOTALSALE you can do so. This allows ordering by values that are not in the database itself and avoids having to calculate a separate column for the

calculation and build it over that column. Previously it was not possible to define a calculated field in SQL.

Here is an example of a derived key index that calculates the total sale amount and puts the column in ascending order:

```
CREATE INDEX total_sale_ix ON
order_det (quantity * price
ASC )
```

So why create an index with a derived key? The answer is performance. Unlike sparse indexes, the optimizer can take advantage of the fact that a value has already been calculated by the index and stored as a key. Let's take the following select statement as an example:

```
SELECT order_id, cust_id,
quantity * price AS Total FROM
order_det
WHERE quantity * price >= 1500
Because an index exists that contains a
derived column of quantity * price, there is
```

no need to scan every row and recalculate the value. The index can be used to access the rows directly and avoid a table scan. [Note: the SQL statement shown above may still use a table scan if the optimizer sees that more than 20% of the sales total to more than \$1500. It will determine this by looking at the derived index before the query runs to get an estimate so it can decide whether to use the index or not.]

If you have a very common expression used in a WHERE clause, it makes sense to consider building an index with that expression. You may be surprised at the performance gain.

These three enhancements to the CREATE INDEX statement are a strong indication that IBM not only wants you to migrate to SQL but is providing the needed functions to make it easier to do so. Furthermore, these enhancements can provide some immediate performance benefit for both SQL and RPG programs.