

Database Goodies in V5R4

While most of you won't be installing the recently announced operating system release any time soon, it is always fun to see what neat features IBM has added to our favorite database. As usual, the hard working folks in Rochester have added a nice mixture of new function, better performance, and lifted limits to allow for more complex SQL queries.

Rather than go into technical detail about how each of the new features work, this article will focus on the benefits of the change and focus on a few of the most interesting additions.

New function

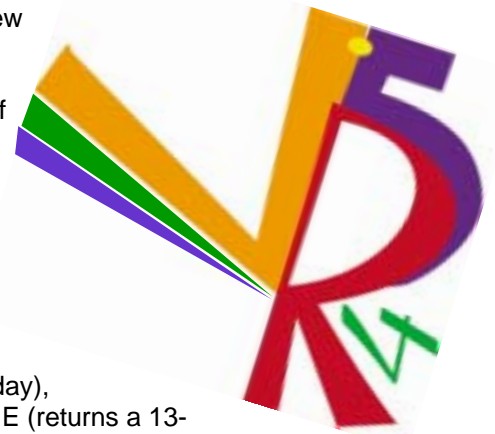
Most new database features surface in the SQL interface since that is where IBM is spending most of its development effort. Here is a short list of the new SQL features:

- Recursive common table expressions
- OLAP functions
- Additional built-in functions
- New special registers

The phrase "**recursive common table expressions**" is a real mouthful. Most people have a difficult time understanding recursion but essentially it means that a part of a query can reference itself. As unusual as this may sound, there are many real-world databases that can make use of this capability. For example, a database with a list of parts may include parts that are made up of other parts. With the new recursive common table expressions you can ask questions like 'Given any part, what are all of the subparts that are needed to build it'. Rather than having to drill into all of the subparts and the parts that make up those subparts in a program, SQL can now gather all of that information for you. Essentially there is yet another way to bury application code in the database so you don't have to write it.

New **OLAP functions** allow queries to be written to rank data values in various ways. Essentially, rows are assigned a number (1-N) based on their ordering relative to other rows. If two rows have the same value they are given the same number. Depending on which of the OLAP functions you use, the number will either be consecutive (i.e. there won't be any numbers missing between 1 and N) or the ranking will skip numbers to account for rows that have the same values. These functions allow queries to be run that can quickly pick out the top or bottom N values in a table or group within that table.

Some of the other new **built-in functions** include **LAST_DAY** (return the last day of the current month), **NEXT_DAY** (returns the date/time of a day after a given date and day of week – for example, tell me the date of the first Monday following today), **GENERATE_UNIQUE** (returns a 13-character unique value), and **ENCRYPT_TDES** (return a value that has been encrypted using the Triple DES algorithm).



The new **special registers** include the following:

SYSTEM_USER	This is the user profile of the person that signed into the system
SESSION_USER	This is the profile currently being used to authenticate.
USER	Same as SYSTEM_USER

These values can come in handy if you want to limit certain operations based on user profile or select certain table data (assuming the user name is in the table).

Performance

On average, every release of DB2 performs better than the last. Performance should improve in V5R4 through a combination of automatic techniques and options that database administrators and

(Continued on page 3)

Inside This Issue:

A Beautiful View
Part 4 of 4 - Page Two

Reader Input
Page Two

Top 5 Reasons to Create an Index
Page Three

STRSQL Conundrum
Page Five

Fun Dates
Page Seven

A Beautiful View

Over the past several newsletters, we've been discussing the benefits of using SQL views. The good news about this series is that the reasons to create and use views run the gamut from making it easier for end-users to query data to improving security for production data. The wide number of ways to use a view means that the work to create them can solve many different problems and save you a lot of time in the long run. Let's summarize the benefits by category.

Ease of use:

SQL views provide a way to make the database easier to understand. This can be accomplished by:

- Only putting columns in the view that are of interest to end-users.
- Providing long names and descriptive text for short, cryptic column names.
- Pre-joining tables together so end-users (and programmers!) don't have to remember how to link data together.
- Convert coded data into human-readable text using CASE or joining in

tables that have descriptive text. Formatting can be done in the view so that it does not have to be done in all reports that need the same information.

Business consistency

Views can be used as a basis for financial or other reporting. If all reports in a certain category can use the same basic views, this will result in reports with consistent information. For example, if two different financial reports generate different numbers, the cause can often be traced back to the fact they use slightly different formulas to calculate the "same" number. If you put the calculation in a view that both reports use, then the number will be consistent. If it turns out the formula used to calculate a value needs to change, it only needs to be changed on one spot rather than a myriad of individual reports.

Security

Views can be used in a variety of ways to

provide an additional degree of security. Rather than expose your raw data to inquisitive users, you can secure the physical tables and build views to:

- Hide columns of data you don't want exposed (e.g. social security numbers)
- Hide rows of data that should not be seen (e.g. only showing regional sales to people in that region)
- Ensure the underlying tables cannot be updated by making the view read-only.
- Allow the use of the views to be audited to avoid having to audit the physical tables, which may be referenced by many application modules. Provide selective decryption support based on user with built-in SQL functions and special registers.

SQL views provide a powerful, yet very straightforward way to make the life in your IT shop easier and more productive.

Here's to your Beautiful View!

Reader Input

Just as an FYI, users should be careful when doing a CRTDUPOBJ on the QAQQINI file from library QSYS. This file contains system triggers which cannot be added or removed in a High Availability Environment. We found this out in our HA Environment and needed to remove the file from the Mirroring Software. An error occurs from the Operating System indicating that System Triggers cannot be added or removed.

KK- System Administrator

In your article "A Beautiful View / Format Away" you used a combination of CONCAT and TRIM. But instead of nesting CONCAT functions, you can use CONCAT like you'd use ||. But in contrary to || concat works with all EBCDIC-Codes (for example I'd to use !! instead of || in the German version).

I think it's more readable. Your solution:

```
SELECT CONCAT(CONCAT(
    CONCAT(TRIM>Last_Name) , ' '),
    CONCAT(TRIM(First_Name) , ' '),
    TRIM(Middle_Name))
FROM customer
```

New solution:

```
Select Trim>Last_Name) concat ' , ' concat
    Trim(First_Name) concat ' ' concat
    Trim(Middle_Name)
>From Customer
```

Viele Grüße / Best regards

BH—Database Administrator

Top 5 Reasons to Create an Index

Five

The fifth reason to create an index is to allow referential constraints to be defined. Essentially referential integrity makes DB2 responsible for making sure that data does not get 'orphaned'. For example, you may have one table that contains customer information such as name, address, and phone number. A second table might contain all of the orders for those customers. Obviously it would be a bad idea to delete a customer record if that customer had an outstanding order. If referential integrity has been set up correctly, when a request to delete the customer with an outstanding order happens, DB2 will use the index over the ORDERS table to quickly determine if removing the customer information would orphan the data and make it impossible for



the shipping department to determine where to send the box.

In addition to making sure that data does not get orphaned, the indexes created for referential constraints allow DB2 to implement something called cascading

deletes. This function pushes the responsibility for deleting all data associated with a particular record by deleting the 'root data'. For example, you may have a business rule that says: "when a customer is deleted from your database; all information related to that customer should also be deleted." With a cascading rule defined, DB2 will do all of the work. In other words, when the master customer record is removed, the system will locate the 'children' rows in the associated indexes, and delete those rows as well.

Four

A fourth reason to create an index is to make sure that duplicate values are not put into your table if those values are supposed to be unique. On the iSeries

(Continued on page 6)

(Database Goodies in V5R4 — Continued from page 1)

programmers can leverage. The most important performance change in V5R4 is that the query engine will be utilizing the new query optimizer to a much greater extent. This change will happen automatically since the SQE (the newer optimizer) can now process a much wider range of statements.

Some other interesting performance options include:

- Index page size options
- SET CURRENT DEGREE syntax for SQL
- Materialized Query Tables (MQT) awareness

The **index page size** option, exposed on the CREATE INDEX statement, allows you to specify the size of the pages used by the index manager. This allows the database administrator greater control over the memory, disk space, and I/O characteristics of a given index. In general, if the index is processed in sequential order a larger page size can be of great benefit. On the other hand a smaller page size is better if the index is processed randomly, only used occasionally or built over a table that is changed very often.

The **SET CURRENT DEGREE** syntax allows the programmer more control over the use of database parallelism without having to resort to calling a CL program that sets the value with the CHGQRYA command. With this new support, individual queries within a program or stored procedure can utilize multiple processors while limiting others.

Materialized Query Tables (MQT) are now part of the base support in V5R4 and have been enhanced over what was available in V5R3 via PTFs. Essentially MQT support provides a way for common query results to be pre-calculated so the query optimizer can bypass a large amount of work in order to return the results to the user.

SQL Limits

To eliminate certain constraints that limit application portability and prohibit tool-generated SQL from working on the iSeries, IBM has raised a bunch of limits. These include:

- SQL statements can now be 2MB
- 128 byte column names
- 1000 tables in a single SQL statement
- 32K keys and ORDER BY length totals

These new limits may seem pretty large (when was the last time you typed in a 2MB SQL statement?), but there are situations where very large SQL statements are generated. These new limits allow the tools that create these statements to be run on V5R4 without being rewritten. This should also allow more SQL-based applications to be ported to the iSeries.

Summary

The team here at Centerfield is excited about the new functions in V5R4 and we'll be enhancing our products to take advantage of the new DB2 support (while continuing to enhance our products for V5R2 and V5R3 as well). Stay tuned!

How do you like your new PDA?

**Couldn't be out here without it.
Lots of disk spikes on the iSeries.**

**Why don't you get disk/HUNTER?
That's what we put on.**

**We did too!
Now I leave
my PDA in
the car.
Now hurry
up and putt.**



It is common knowledge that industrial strength iSeries® systems with lots of users running critical applications require industrial strength notification tools. And those tools need to be monitored and set up to notify you of critical availability events. Almost all large iSeries shops have these.

But what do you do once you are notified of a critical system event — except have your golf game interrupted? Wouldn't it be nice to have a tool that does both?

Notification AND diagnostics?

When it comes to availability-threatening disk spikes on the iSeries, disk/HUNTER from Centerfield Technology in Rochester, Minnesota fills the bill.

For more information, visit us at www.centerfieldtechnology.com



IBM System Storage – Storage for an on demand world.

Centerfield Technology, Inc. is an IBM® Business Partner who has demonstrated success in delivering storage solutions to meet the needs of our customers. disk/HUNTER runs on IBM System Storage™, which includes disk storage systems, Storage Area Networks (SAN), tape backup solutions, Network Attached Storage (NAS), infrastructure management and virtualization software. IBM offers the right mix of storage products, solutions, and services using new innovative technology and open standards to help improve your business continuity, infrastructure, simplification and information lifecycle management. To find out how you can leverage IBM System Storage products visit www.ibm.com/servers/storage.



The IBM logo and the IBM Business Partner emblem, and IBM System Storage are trademarks or registered trademarks of International Business Machines Corporation in the United States, other countries, or both.

STRSQL Conundrum

All of you know about it and most of you have used it at some point in time – the interactive SQL editor that comes as part of SQL Development Kit, an IBM charged feature.

Recently I received a question as to the location of the session statement history once you exit STRSQL and if it's possible to prune it out so only certain, good SQL statements are kept. I knew a little bit about this but wanted to give a more informed answer so I went all gung-ho and researched it in greater detail. Boy, was I in for an unexpected ride!

Where and if your interactive SQL session attributes get saved depends upon which option you choose when you exit STRSQL. When you hit F3 to exit interactive SQL you will be prompted with 4 options:

- 1=Save and exit session
- 2=Exit without saving session
- 3=Resume session
- 4=Save session in source file

Let's describe each one of them in greater detail.

1)
Being that the default on this option is 1, let's discuss what happens when you simply hit Enter after hitting F3 to exit interactive SQL. The system will save your session in a Permanent Miscellaneous Space MI Object in the QRECOVERY library (type 19 subtype EE). This is not a file, and is not designed to be accessed by an end-user (i.e. you). That said, if you really want to see what's in this space, you could do the following:

```
DMPYSOBY OBJ(ISQLST<usrprf>*) CONTEXT  
( QRECOVERY) TYPE(19) SUBTYPE(EE)
```

If your user profile is TEST you would replace <usrprf> with TEST (i.e. ISQLSTTEST*). This will create a spooled file QPSRVDMP in your library. If you search this spooled file on term "NAME-" you may get multiple hits. This will occur if you ever work in interactive SQL on more than one AS400 workstation device. What this means is that you'll have multiple space objects in the QRECOVERY library. The naming convention IBM follows for these internal space objects based on byte positions is:

- 01-06 :: ISQLST
- 07-16 :: User profile starting the session.
- 17-26 :: Workstation device name for session selection
- 27-30 :: Session number for that workstation ID <0000 is first>

If you saved only one interactive SQL session by using option 1 (Save and exit session) on the

Exit Interactive SQL display, you may resume that session at any workstation. However, if you use option 1 to save two or more sessions on different workstations, interactive SQL will first attempt to resume a session that matches your workstation. If no matching sessions are available, then interactive SQL will increase the scope of the search to include all sessions that belong to your user ID. If no sessions for your user ID are available, the system will create a new session for your user ID and current workstation.

For example, you saved a session on workstation 1 and saved another session on workstation 2 and you are currently working at workstation 1. Interactive SQL will first attempt to resume the session saved for workstation 1. If that session is currently in use, interactive SQL will then attempt to resume the session that was saved for workstation 2. If that session is also in use, then the system will create a second session for workstation 1.

However, suppose you are working at workstation 3 and want to use the ISQL session associated with workstation 2. You then may need to first delete the session from workstation 1 by using option 2 (Exit without saving session) on the Exit Interactive SQL display.

2)
I already mentioned option 2 earlier. It could be useful if you wish to delete some or all of these space objects to conserve disk consumption on your system. Just sign on to the workstation device name that you usually use for your interactive SQL work and select option 2. This will delete the space object maintained in QRECOVERY. If you keep reentering STRSQL and selecting option 2 upon exit, you'll delete all existing space objects.

3)
All the "Resume session" option does is bring you back to the STRSQL editor. Think of it as Cancel, i.e. you changed your mind and don't want to exit interactive SQL.

4)
This is where you control your destiny by explicitly telling the system that you want to save SQL statements from the interactive SQL session you've run. By default these sessions will then be saved in the QSQLSESS source physical file but you can select the library, file and member in which you want the session information saved. This is a regular, easily read source member and could prove useful if you want to archive your STRSQL experiments.

(Continued on page 7)

(Top 5 Reasons to Create an Index—

Continued from page 3)



(System i5) the only way to enforce uniqueness at the database level is to create an index.

There are several different ways to create a unique index. The first approach is to define the column as unique at the physical file level. This can be done via DDS or SQL. With SQL, you can use the syntax of UNIQUE or PRIMARY KEY. Either way, the data is guaranteed to be unique since the database will prevent duplicate data from being added by an INSERT or UPDATE. A second approach is to define the unique attribute with a logical file or SQL index. Which method you use depends on your environment and database maintenance needs. If the unique attribute is defined at the physical table level, there is no way to bypass the unique requirement. If you define the uniqueness at the logical file or index level, you have the flexibility to disable the unique constraint if you need to remove it temporarily to do table changes or to populate it with new data that may not be 'clean'.

Three

This subtle reason to create an index has nothing to do with traditional programming. The DB2 query optimizer can use indexes to select and process records, and it can also use the index to get statistical information that helps it make good decisions.

When a query is submitted to DB2, it is examined and the optimizer tries to determine how to implement the query. For simple queries, there are at least two alternatives and for complex ones there can be hundreds of choices. These choices are made based on the estimated number of rows that will be processed at each step in the query. The index structure can be examined to help the query optimizer get accurate estimates that will lead to efficient queries.

One way the optimizer uses an index is to take values from the query and estimate

how many values in the table will be returned. For example, a query may want to retrieve all part numbers between 35539 and 35699. If an index exists for the part number column, the query optimizer can efficiently determine the number of part numbers that fall within this range. If the number is relatively small, the index will likely be used. If the number is high, then the optimizer will likely try to use another index or try a non-index access method.

Two

A second reason to create an index also relates to performance but in a different way. Besides selecting rows, queries often group, order and join tables together. Indexes assist the query optimizer in performing these tasks in an efficient manner. For example, let's say you have a query that gets the total sales by state for the past month. To do this work, DB2 has to find all of the rows in the table that have the same state and add up the individual sales amounts. If there is an index over the state column, the keys in the index are already ordered so that each state is together. This allows the database engine to gather up the sales figures for each state and return that result row before working on the next state.

Joining tables together is another common task that requires an index to perform well in a very similar way that random access becomes faster with an index. Essentially, the column(s) from the first table is used to build a key and then do a random probe into the index to locate the rows that meet the joining criteria. Without an index this process would be prohibitively expensive.

Returning rows in a sorted order is the traditional way for RPG programs to process data in a predictable manner. DB2 queries that use an ORDER BY clause can also take advantage of this fact and utilize an existing index. The index removes the need to sort the data after the selected rows are found.

One caveat to this set of reasons relates to the "20% rule". If the query or program needs to process more than one-fifth of the table or tables, non-index techniques

are sometimes better. If you have one of those situations with an SQL statement, the query optimizer is responsible for determining if using the index is better than a non-index approach. If you are coding the joins, ordering, or grouping in an RPG program it is the programmer's responsibility to choose the algorithm that will be most efficient.

One

The most obvious reason to create an SQL Index (or keyed logical file if you are an RPG programmer) is to speed up access to rows (records) that meet narrow selection criteria. The most extreme example is when there is a column (field) that is unique, such as a customer identifier. Whether SQL or RPG is used to locate information efficiently, it is essential if a program or query wishes to select a single customer that an index exists on the customer identifier. For example, if you have a table with 1,000,000 customers, without an index a program or query would have to examine 500,000 rows on average to find a single customer's data. With an index, on average that same customer could be found by examining less than 20 keys.

So what does 'narrow selection criteria' really mean? In our example above, it meant that one row out of a whole table was being located. As a general rule of thumb, if a query selects less than 20% of a table an index is potentially justified to speed up access. As you get above that threshold, it is normally more efficient to use a table scan technique to find the data.

More....

We've covered a list of reasons to build an index. As you can see there are performance and functional reasons why an index can help you accomplish a given goal. When it comes to queries, it is best to experiment with different combinations of index creations to get the right balance of good performance with minimal performance drag.

Fun Dates

No, it's not what you think. I am not going to be playing Mr. Hitch today and give you dating advice. Much to your chagrin, I have picked a rather mundane subject instead, namely SQL dates and associated built-in functions. At the offset, let me just say that I have tested these on V5R3 and there is no guarantee that all of the built in functions I have used are supported on earlier releases. Now that we have cautions out of the way, let's delve into fun immediately!

Build this view on your system:

```
CREATE VIEW FUNDATES AS (  
  SELECT  
    CURRENT_DATE - 7 DAYS AS LAST_WEEK,  
    CURRENT_DATE - 1 DAYS AS YESTERDAY,  
    CURRENT_DATE      AS TODAYS_DATE,  
    CURRENT_DATE + 1 DAYS AS TOMORROW,  
    CURRENT_DATE + 7 DAYS AS NEXT_WEEK,  
    CURRENT_DATE + 3 MONTHS AS IN_3_MTHS,  
    CURRENT_DATE + 1 MONTH - DAY(CURRENT_DATE + 1 MONTH) DAY AS LAST_CUR_MONTH,  
    CURRENT_DATE + 1 DAY - DAY(CURRENT_DATE) DAY AS FST_CUR_MONTH,  
    CURRENT_DATE - (DAYOFYEAR(CURRENT_DATE) - 1) DAYS AS STARTYEAR,  
    CURRENT_DATE - (DAYOFYEAR(CURRENT_DATE)) DAYS + 1 YEAR AS ENDEYEAR,  
    DAYOFYEAR(CURRENT_DATE) AS DAY_OF_YEAR  
  FROM SYSIBM/SYSDUMMY1)
```

(Continued on page 8)

(STR SQL Conundrum—Continued from page 5)

What a conundrum! So HOW can one save just frequently run queries in the interactive SQL session object?

Here comes the bad news. Assuming you don't have MI programming experts available to you, there really is no easy alternative you can use.

I suppose you can try F13->3 to clear all the entries from your current session, run your 'useful' statements and then save them. However, next time you're in STRSQL you'll again save any newly run statements in addition to the original statements. Perhaps that's not that bad if you save your original session using option 4 first and repeat the clear/save procedure occasionally.

The only other STRSQL alternative I see is to reserve some workstation device for special purpose queries. Then you'd only sign in to those workstations when you want to run special purpose queries. You'd use other workstation devices for general ad-hoc querying.

I can already hear you groaning, "Please, stop the pain!"

Being that I didn't really offer you any easy alternatives for interactive SQL sessions, let me emphasize that IBM's strategic development direction is not to improve STRSQL but rather to enhance the RunSqlScript feature of iSeries Navigator. In this interface you could run both CL commands by prefixing them with "CL:" prefix as well as any SQL commands. You can save these sessions in a PC file and name them anything you want.

You can save as many sessions as your PC can fit. This really is a better and more modern interface and I encourage you to try and use it more. I think IBM will agree with me on this point.

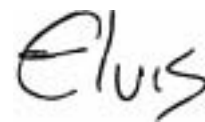
To get access to the RunSqlScript function, you need to have iSeries Navigator installed on your PC. This is a free tool from IBM and comes bundled with iSeries Client Access feature. Check your iSeries installation CDs for it or talk to your system administrator (setup.exe may already be available on the shared drive in the IFS of your iSeries).

Once you have iSeries Navigator installed, you can follow these instructions to get to RunSqlScript function:

iSeries Navigator->[system name]->Databases->[database name]->click "Run an SQL script" in the lower window pane

OR you can invoke its executable directly:
Start->Run->cwdbundbs->OK

Good luck and have fun!



Elvis Budimlic
Development Director

(Fun Dates—Continued from page 7)

This is a dynamic view, meaning that the output values will change depending on the day you are running it. Another thing to note is the use of IBM file SYSDUMMY1. This file is part of standard installations nowadays and is frequently used for illustrating built in functions as it is guaranteed to contain a single row, hence guaranteeing we get a single row output, which is what we want.

Now query it simply by doing:
SELECT * FROM FUNDATES

You'll get data like what was the date a week ago, yesterday, today, tomorrow, in a week, in 3 months, last date of this month, first date of this month, start date of this year, end date of this year and current day of the year.

You can expand on this view by either building other views over it or by simply querying directly. Let's say you want to see how many days are remaining in the year:

```
SELECT
(DAYS(ENDYEAR)-DAYS(STARTYEAR)+1) - DAY_OF_YEAR AS DAYS_LEFT_IN_YEAR
FROM FUNDATES
```

Or if you wanted to test if this year is a leap year:

```
SELECT
CASE (DAYS(ENDYEAR)-DAYS(STARTYEAR)+1)
WHEN 366 THEN 'YES'
ELSE 'NO'
END LEAP_YEAR
FROM FUNDATES
```

Or simply you want to know what day of the week is the first and the last day of the month:

```
SELECT DAYNAME(FST_CUR_MONTH) AS FIRST_DAY_OF_MONTH,
DAYNAME(LAST_CUR_MONTH) AS LAST_DAY_OF_MONTH
FROM FUNDATES
```

Having fun with dates? Lets have some fun times now.

```
CREATE VIEW FUNTIMES AS (
SELECT
MIDNIGHT_SECONDS(NOW())/3600 HOURS_SINCE_MIDNIGHT,
MOD(MIDNIGHT_SECONDS(NOW()),3600)/60 MINUTES_SINCE_MIDNIGHT,
MOD(MIDNIGHT_SECONDS(NOW()),60) SECONDS_SINCE_MIDNIGHT
FROM SYSIBM/SYSDUMMY1)
```

So, why would anyone create a FUNTIMES view as I've outlined if they can simply invoke SELECT CURRENT_TIME FROM SYSIBM/SYSDUMMY1? Well, what's the fun in that?

There are many other goodies IBM has added over time. Take a look at all of them at this link:
<http://publib.boulder.ibm.com/infocenter/iseres/v5r3/topic/db2/rbafzmstch2func.htm>

See you at Spring COMMON in Minneapolis!

Elvis

