Out in Left Field
strategic tools for strategic teams
Volume Four, Issue One

SQL JOINS PRIMER

All databases have tables that must be combined to get useful information. The classic example is when two files have a master-detail relationship. For example, an ORDER table may contain general information about a purchase like the customer's name, address, and the date. Details about the order might be put in another table called ORDDTL. It would have a row for each product purchased, the quantity, and maybe a warehouse identifier. To see all of the information about an order the ORDER and ORDDTL tables must be joined together.

Joins can be a complex subject. There are many types of table relationships and ways to join information. This article tries to explain the different types.

Which join syntax to use

Centerfield strongly recommends usage of **explicit** join syntax. By doing so, your intention is clear, the query is easier to maintain, you have more filtering power in the join clause, you're giving more explicit information to the query optimizer (i.e. on the desired join order), you're able to separate join and filtering criteria easily and you'd be compliant with industry recommended standards.

Why bother learning implicit join syntax

Here's the rub, DB2 for i5/OS does not support explicit join syntax in joined UPDATEs or DELETEs. If you're wondering why, it's because it's not in the ANSI SQL standards.

When you need to perform a joined UPDATE (a very common task in the life of an SQL developer), you have to use implicit join syntax; hence my inclusion of implicit join syntax in the table on page 2.

Here are several pieces of good news about implicit join syntax:

- it has been around longer, so it's supported on all database platforms
- performance is the same regardless of the syntax used (the DB2 query optimizer rewrites the query internally in a standard form)

Advanced JOIN concepts

Filtering placement

It is important to note that the placement of filtering conditions does matter (JOIN versus WHERE clause).

For example, let's say we want to enhance our LEFT OUTER JOIN to include another condition:

```
SELECT A.FIELD,B.FIELD
FROM A LEFT OUTER JOIN B ON
A.ID = B.ID AND B.FIELD = 'B1'
```

This is still a LEFT OUTER join and returns all the rows from the primary dial, except that now non-matching rows in secondary field will return NULL for all instances where B.FIELD **does not** equal 'B1'.

There is a common misconception that the following SQL statement has identical function:

```
SELECT A.FIELD,B.FIELD
FROM A LEFT OUTER JOIN B ON
A.ID = B.ID
WHERE B.FIELD = 'B1'
```

Filtering is done **AFTER** the join has completed and will weed out all of the rows in the result set where B.FIELD does not equal 'B1'. This in essence turns the join into a modified INNER join.

Multiple joins

You can join more than one table using different types of join conditions. Note that parentheses are optional with explicit join syntax, since multiple joins are evaluated in left-to-right order.

Self join

Since most System i shops own databases that have evolved over 2-3 decades rather than being designed, it is not unusual to find tables that contain "everything" in them. There are no satellite (a/k/a detail or dimension) tables to join to, instead an SQL developer may have to join the table back to itself. This is possible in SQL and is often referred to as self-join.

To join a table to itself simply specify the same table two or more times in the FROM clause and differentiate them with their own table alias. As illustrated here:

```
SELECT X.*, Y.* FROM A X INNER
JOIN A Y ON X.FIELD1 = Y.FIELD2
```

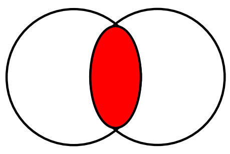
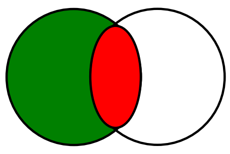
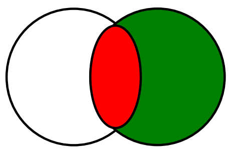
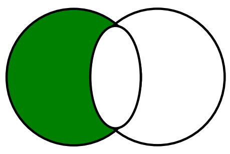
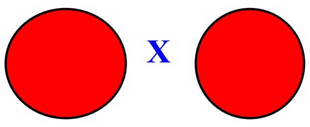
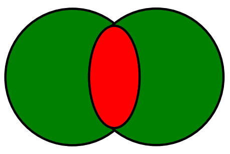
Indexing

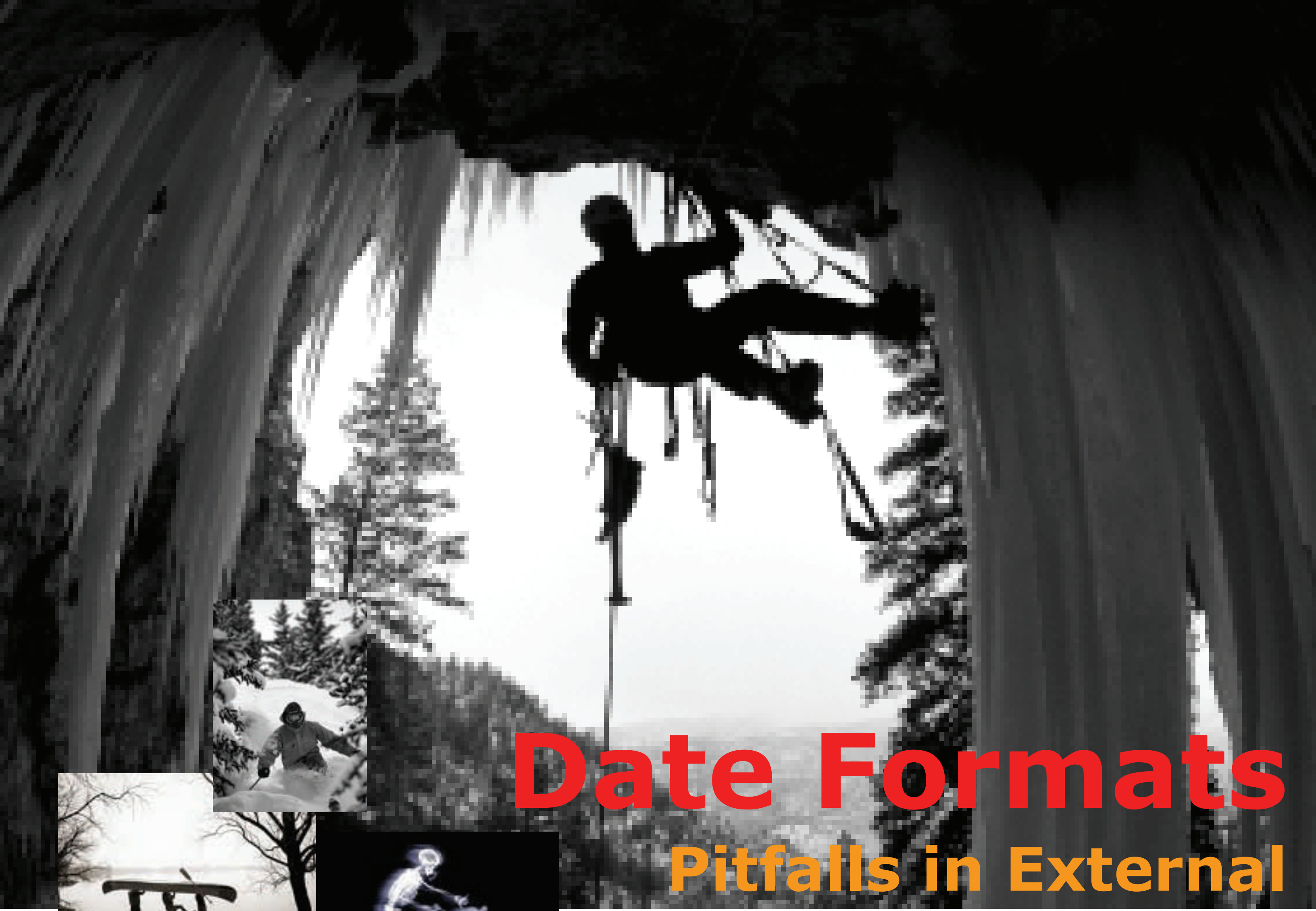
Excuse my candor: It is **not** the responsibility of the DBA to build indexes over join columns. This is the duty of the DB Architect as well as any "creative" SQL developer who decides to join two tables on columns that are not already indexed.

DB2 will run any query including a join on non-keyed columns. The problem is that frequently, when join columns are not keyed, a Cartesian Product join is performed and then further search criteria filtering is applied. As I've already stated, Cartesian Product is the worst-performing type of join processing and can take a VERY long time to complete, as well as chew up significant system resources. *Elvis*

Inside this issue:

NOTE: I am taking liberties with Venn diagrams here. In essence, I assume 1-1 correspondence on the ID field used in the examples. In the real world, there are often 1-N and even N-N relationships in the data. Nevertheless, I believe that this type of primer is long overdue in System i arena and will help more people understand the JOIN concept despite omitting examples to cover N-N sets.

Type of JOIN	Explicit join syntax	Implicit join syntax	Description
INNER join (aka Equi-join aka JOIN) 	SELECT A.FIELD, B.FIELD FROM **A INNER JOIN B ON A.ID = B.ID** or SELECT A.FIELD, B.FIELD FROM **A INNER JOIN B USING(ID)**	SELECT A.FIELD,B.FIELD FROM A, B **WHERE A. ID = B. ID**	Retrieve all data from the primary dial (file on the left) that has a matching ID with the secondary dial (the file on the right). This is the most common type of join used in the industry. NOTE: The INNER keyword is optional.
LEFT OUTER JOIN (aka LEFT JOIN) 	SELECT A.FIELD,B.FIELD FROM **A LEFT OUTER JOIN B ON A.ID = B.ID** or SELECT A.FIELD,B.FIELD FROM **A LEFT OUTER JOIN B USING(ID)**	SELECT A.FIELD,B.FIELD FROM A,B WHERE A.ID = B.ID **UNION ALL** SELECT A.FIELD,' ' FROM A WHERE NOT EXISTS (SELECT FIELD FROM B WHERE A.ID = B.ID)	LEFT join returns all data from primary dial, even if there is no match in the secondary dial. For rows that don't have a match, any projected fieldsfrom the secondary dial are substituted with a NULL special value (note use of blanks in implicit join syntax to simulate NULL output). NOTE: The OUTER keyword is optional
RIGHT OUTER JOIN (aka RIGHT JOIN) 	SELECT A.FIELD,B.FIELD FROM **A RIGHT OUTER JOIN B ON A.ID = B.ID** or SELECT A.FIELD,B.FIELD FROM **A RIGHT OUTER JOIN B USING(ID)**	SELECT A.FIELD,B.FIELD FROM A,B WHERE A.ID = B.ID **UNION ALL** SELECT ' ',B.FIELD FROM B WHERE NOT EXISTS (SELECT FIELD FROM A WHERE A.ID = B.ID)	RIGHT join is **rarely used** in practice as it's simply a reverse of LEFT join so developers usually just switch the table order and use LEFT join instead. This is a well accepted practice in the industry and I include RIGHT join here only for the sake of completeness. NOTE: The OUTER keyword is optional
EXCEPTION JOIN (aka non-matching records join) 	SELECT A.FIELD,B.FIELD FROM **A EXCEPTION JOIN B ON A.ID = B.ID** Or SELECT A.FIELD,B.FIELD FROM **A EXCEPTION JOIN B USING(ID)**	SELECT A.FIELD,' ' FROM A WHERE NOT EXISTS (SELECT * FROM B WHERE A.ID = B.ID)	An EXCEPTION join is unique to DB2. It returns rows from the primary dial that have no match in the secondary dial. Obviously, any fields projected from the secondary dial would return NULL. You can use LEFT & RIGHT qualifiers with this join type as well.
CROSS join (aka Cartesian Product join) 	SELECT * FROM **A CROSS JOIN B**	SELECT * FROM **A, B**	This is the worst kind of join from a performance standpoint. If you have 1000 records in each table, end result would contain their product (1,000,000 records). Often CROSS JOINS occur in **error** as well, since many SQL developers don't realize that an implicit join occurs if no criteria is specified and there is more than one file in the FROM clause.
FULL OUTER JOIN 	SELECT A.FIELD, B.FIELD FROM A **LEFT OUTER JOIN** B USING(ID) **UNION ALL** SELECT A.FIELD,B.FIELD FROM B **EXCEPTION JOIN** A USING(ID)	SELECT A.FIELD,B.FIELD FROM A,B WHERE A.ID = B.ID UNION ALL SELECT A.FIELD,' ' FROM A WHERE NOT EXISTS (SELECT * FROM B WHERE A.ID = B.ID) UNION ALL SELECT ' ',B.FIELD FROM B WHERE NOT EXISTS (SELECT * FROM A WHERE A.ID = B.ID)	A FULL OUTER join returns all rows from the primary dial as well as any non-matching rows in the secondary dial. DB2 for i5/OS **does not support** FULL OUTER join syntax since it can easily be simulated using a combo of LEFT OUTER join and EXCEPTION join. Use of COALESCE function can help remedy cases where values projected from seondary dial return NULL even though they don't technically have to (i.e. COALESCE(A.FIELD,B.FIELD)).




Date Formats

Pitfalls in External User Defined Functions

By **Birgitta Hauser**

Question: We want to register an RPG function stored in a service program as an external User Defined Function (UDF). The problem we are faced is, one of the parameters is a date and the RPG function expects this date being passed in *EUR (European) date format. But if we register the UDF with a date parameter, this date will be passed in the *ISO date format and consequentely the execution of the RPG function will fail. Changing the RPG function to expect the date in date format *ISO is not allowed and setting the date format with SET OPTION DATFMT = *EUR is not possible, either.

Is it possible to predefine the date format in the external function or is there a way to make RPG accept a date in any format?

Before we solve the problem, let us have a look how RPG and SQL work with dates.

A date is not stored in ISO format or any other date format. A date is stored as 4 byte binary value (Scaliger Number) representing the number of days since some fixed date.

RPG

RPG always converts the binary date values into character strings using the date formats specified in either the D- or H-specifications. As soon as the date will be used for a calculation or stored in a physical file or SQL table, the character string will be reconverted into the binary date value. Due to the conversion, the date must be checked to make sure it is in a valid range. If a date format with a 2 digit year is used, the valid range is between 01/01/1940 and 12/31/2039. Any date outside this valid range will cause an error.

When calling a program or procedure, the character string representing the date (or a pointer to the converted character string)

(Continued on page 6)

CTO MEMO

Those of us in the Midwest have just experienced a month of blustery, cold weather – but it is what we've learned to expect from the season. Being an election year, we've all endured a similar amount of bluster from those trying to get into office – but again, it's just what you'd expect from the political season.

Despite the season, we hope this newsletter comes to your e-mail with plenty of substance and a minimum of hyperbole.

To start off, we've got a couple of exciting announcements. The first is next week's availability of HomeRun Version 6.1 with some nice features I'm sure you will like. See the article in this newsletter for more information.

The second announcement is Centerfield's support of DB2 Web Query for System i. We believe that this product will be a huge hit and are introducing a service to help iSeries shops get the product installed, configured, and verified with almost no effort on their part. We expect to introduce other services and software tools for users of DB2 Web Query and help them leverage this very powerful business intelligence product.

You'll find an interesting article about Web Query in this issue of our newsletter as well as some good reading here:
http://www.mcpressonline.com/february-8-2008-vol-5-issue-6/db2/with-perseverance-and-luck-you-can-conquer-the-db2-web-query-install.html

http://www.itjungle.com/fhs/fhs021208-story05.html

As usual, enjoy our mix of technical articles. For example, I think you'll enjoy the exploration of table joins and some of the new V6R1 database features.

As always, let us know what you think and if you have suggestions for future articles.

Best Regards,

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

HomeRun 6.1 is GA Now

Since the HomeRun 6.0 was announced and released last fall, the team here at Centerfield has been collecting customer comments and making a series of fine-tuning changes to make it even better. Next week, Version 6.1 will be available for download and use by our existing customers and for evaluation for iSeries shops in need of tools to optimize the performance of SQL and DB2. Let's cover the highlights in this new release.

Improved data collection function

It is always important to minimize the amount of data collected by a performance tool, especially if a large amount of information can be gathered. The database monitor support built into DB2 is a good example. On busy systems a database monitor can collect gigabytes of data in a few minutes, imposing a significant CPU and disk load. On already busy systems, this overhead must be minimized by targeting just the jobs that need to be tuned.

To help accomplish that goal, new filters have been added to our data collection component. Now you can select a particular user or group profile so that just the jobs running under those profiles are monitored. While HomeRun has had the ability to filter by job name for a very long time (where job name was often the same as the active user profile), it has not had the ability to selectively collect data for current profiles. The good news is that this support has been made available for shops that are still using V5R2 and V5R3. On V5R4, HomeRun leverages built-in i5/OS support to collect for current profiles.

Support has also been added to collect data based on IP address. Many multi-tier applications connect to the iSeries through Wintel servers. By isolating jobs that originate from a particular server, a specific application or set of users can be monitored. Furthermore, if you have a particular user at a known IP address the new filtering will allow you to collect for just that user.

Enhanced AutoDBA advice

Starting in Version 6.0, our strategic platform for DB2 and SQL advice is our AutoDBA function. With 6.1, we've fine-tuned the way we produce advice to make it easier and more effective to implement. Here are a several of the changes:

- Prioritization of index advice is very important because it means that indexes with the most benefit are created before others that have less value. We've improved the way we prioritize index advice to take more factors into account so that the priority more closely reflects the ultimate value that will come from creating the object.

- The "clustered index" recommendation (which tells you the optimum order of the rows in the table), has been changed to make it easier to identify which index should be the target of a concurrent reorganization.

- Suggestions made by AutoDBA now have a new column of information that contains the CL command that can be used to implement the advice. This allows you, as the user, to extract that command and execute it in whatever fashion you desire – even when HomeRun does not allow you to automate a particular piece of advice. The bottom line is more flexibility and control.

Thanks to our customers who have given us valuable feedback the last few months. We appreciate your help to make HomeRun even better!

Flirting with DB2 Web Query for System i

I've been flirting with *DB2 Web Query for System i* since early January because I was curious just how it compared to Query/400 and because it's positioned as the successor to this tried-and-true product.

Our team loaded DB2 Web Query for System i on our V5R4 system, added me as an authorized user and off I went reading the "Getting Started" Redbook. [Since this product's name is such a mouthful, I'll refer to it as Web Query for the rest of this article.]

To be honest, getting started didn't go as smoothly as I would have liked. To begin with, new terminology is introduced which initially confused me. This in turn caused me to go down a couple of dead-ends, but as with anything new, some hiccups were to be expected.

Here's my summary judgment: Web Query is nothing like Query/400!

- ***Don't compare it to Query/400***

Throughout the course of my career I've met people who could draw parallels between any two (unrelated) things. Doing so between these two products would be a great mistake. Though the objective for both products is to create reports, Web Query's design and architecture are so radically different from Query/400, you would do yourself a disservice by thinking you (or your end-users) could simply jump into it and continue doing the same-old thing.

In fact, *we can do so much more with DB2 Web Query*; I will stop making any comparisons with Query/400 and put it into its rightful place – the *legacy hospice*. Sure, you can continue using Query/400, but once you dip your toes into the Web Query water, you won't find a reason to go back to it any longer.

- ***What helped me get started***

What truly helped me is that I have experience building Crystal Reports (now owned by Business Objects which in turn has been purchased by SAP at the end of last year). Crystal Reports (CR) is such a comprehensive product, I know several people whose job is to do nothing else but write new and exciting reports for their company. It turns out there is a lot of overlap between CR and Web Query.

One key difference is that Web Query is designed to be used in a web browser, while CR is designed to be used as a thick (rich) GUI client. This is a very significant advantage to IT personnel as anyone familiar with managing PC's knows. Rich clients do have their place however. IBM, in fact, offers a thick client called Developer Workbench to help report writers create and change queries. I haven't used it yet but have seen at least one feature I personally would use – writing ad-hoc SQL statements to be used as synonyms (think of a synonym as your data source – result set which gives you the base data for the report). As it stands today, I have to resort to writing SQL views to mask some of the database complexity and leverage advanced SQL syntax and built-ins.

So what else have I done besides reading the "Getting Started" Redbook and drawing correlations between CR and Web Query? I've watched Jackie Jansen's online demo which is very nicely done and amply illustrates Web Query's potential. See it yourself at this link:

http://www-03.ibm.com/systems/i/software/db2/webquery/flashdemo/

- I was privy to another demo by Gene Cobb (IBM's Mr. Web Query)
- I've went through IBI's DB2 Web Query for System i eLearning module (it was helpful that I could learn at my own pace)
- I've created a meaningful report from scratch, using number of features I've learned about in my introductory lessons

- ***Some things I've learned***

✓ IBM has done a very good job advancing Web Query as a replacement for Query/400. It'll do that and much more. True BI (Business Intelligence) is now at your fingertips, without spending exorbitant amounts of money.

✓ While it may be revolutionary for green-screen diehards, I believe that base offering is truly just a beginning and just for beginners. Once you see them in action, you WILL want to add Active Reports, OLAP functions, Developer Workbench and if IBM offers it, the ReportCaster add-on as well.

✓ Active Reports rock (offline, dynamic analysis that includes data in the report as well as advanced filtering capabilities combined into a single, easily distributable object). You can email them to all of your system analysts and they can then traverse the data on their laptops - without a need for direct connection to the System i.

✓ OLAP allows you to dynamically slice and dice the data. This means that reports aren't static any longer. You don't have to write a report for each request by the analyst. You provide a base report and let them slice and dice it any way they want.

✓ Don't underestimate the learning curve. While you can probably go with quick and dirty approach and show a proof-of-concept relatively quickly, to truly leverage the power of Web Query you'll need to open your mind to new design concepts and new way of doing things.

✓ I haven't had a chance to take advanced education offerings from IBM or IBI so I'm not exactly sure how Web Query report developers debug wrongly construed FOCUS (4GL scripting language) requests. I've struggled debugging some of the issues I created, and sincerely hope they offer some sort of friendly debug environment in the Developer Workbench.

With my bias for superior database performance, I am especially glad that customers that are still using Query/400 can finally get off the old Classic Query Engine (CQE) and convert to new, better performing SQL Query Engine (SQE). Performance between the two query engines can differ by orders of magnitude.

and not the binary date value will be passed. As long as prototypes are used and the date format is specified in the prototype, there will be no problem. Parameters passed by value (keyword VALUE) or by constant reference (keyword CONST) will be converted into the expected date format. If the parameter will be passed by reference and a date with a different date format is used, the module will compile successfully. When using parameter lists, different date formats will also cause a program to be ended abnormally.

SQL

Contrary to RPG SQL does not convert dates into character strings. Date formats are only used to make the binary date value readable. Which date format is used depends on the environment or job in which the SQL statement is executed.

To change or set the date format in interactive SQL (STRSQL) the following steps are needed:

Press function key F13
Select Option 1
Change the option DATFMT to the format you want

In iSeries Navigator's Run an SQL script, the date format can be set or changed with the following steps:

Open Connection
Open JDBC Setup
Select Format and change the date format to the format you want

In embedded SQL the used date format is predefined at compile time. The date format can be specified in the compile command (CRTSQLxxxI) with the option DATFMT. The default value for the compile option DATFMT is *JOB and the job's date format usually has a 2 digit year. If you change the date format at compile time you have to remember this modification each time the module must be recompiled. A better solution is to add a SET OPTION statement at the beginning of the source code (immediately after the D-Specifications). Beginning with release V5R4 the SET OPTION statement must preceed all other SQL statements specified in your source. That means a SET OPTION statement specified in the *INZSR subroutine that is located at the end of the source code, will prevent the program from being created.

```
C/EXEC SQL  Set Option DatFmt = *ISO
C/END-EXEC
```

Example 1: SET OPTION - Statement

A SET OPTION – Statement can also be used in SQL triggers, stored procedures and UDFs to set compile options.

Date values out of the valid range are displayed with ++++++++++, but do not cause any error. In this way it is possible to insert a row and initialize a date field with '01/01/0001', even though a date format with a 2 digit year is used in the current job.

Furthermore SQL is able to identify character stings in one of the following formats as DATE independently of which date format is currently used.

YYYY-MM-DD (*ISO – format)
MM/DD/YYYY (*USA – format)
DD.MM.YYYY (*EUR – format)

Casting the character string into a real date is only mandatory if date calculations must be performed.

If a date is passed between two RPG procedures this date will be converted into a character string. This conversion will also occur if an RPG procedure is registered as SQL stored procedure or UDF and is called by any SQL-Interface.

How is it possible to pass a date from SQL to RPG in the expected date format?

In many company databases, dates are only stored as numeric values. Converting a numeric date into a real date and vice versa with SQL is rather complicated. In RPG only a single statement is needed. The following example shows a RPG function where the built-in-function %DEC() is used to convert a real date with European format into a numeric date in *ISO format (YYYYMMDD). This function works with release V5R3 and higher. If you are stuck with a lower release, you either have to use the OpCode MOVE or a combination of the built-in-functions %INT() and %CHAR() to convert the date.

(Continued on page 7)

```
H NoMain Debug DateEdit(*DMY) DatFmt(*EUR)
*************************************************************************
P CvtDateToNum    B                   Export

D CvtDateToNum    PI             8P 0
D   DateEur                       D   DatFmt(*Eur)
 *-------------------------------------------------------------------------
 /Free
    Return %Dec(DateEur *ISO);
 /End-Free
P CvtDateToNum    E
```

Example 2: Converting a date in European format into a numeric date in ISO

This function CvtDateToNum is integrated in the Service-Program CVTDATNUM. When registering this RPG function as UDF with a parameter of type DATE, the SQL date will be passed as a character string in ISO format (YYYY-MM-DD). To avoid the automatic conversion into a character string, the UDF can be registered with a parameter with a CHARACTER/CHAR data type as shown in the example below:

```
Create Function MYSCHEMA/CvtDateToNum(
       ParDate  CHAR(10)
       Returns  DECIMAL(8, 0)
       Language RPGLE
       Specific MYSCHEMA/CVTDAT00001
       Deterministic
       No SQL
       Returns NULL on NULL Input
       No External Action
       Not Fenced
       External Name 'MYSCHEMA/CVTDATNUM(CVTDATETONUM)'
       Parameter Style General;
```

Example 3: Registering the external function with a character data type parameter.

To call this UDF, the date first must be converted into the character representation in European format. A date can be converted into a character string by using the SQL scalar function CHAR. The scalar function CHAR accepts a second parameter, where the date format can be passed in which the date must be converted. The following example shows how the UDF CvtDateToNum is

(Continued on page 8)

(Date Formats, Pitfalls in External User Defined Functions—Continued from page 7)

called from SQL with a date that is converted into the character representation of a date.

```
select CvtDateToNum(Char(Current_Date, EUR))
```

```
From SysIbm/SysDummy1;
```

Example 4: Calling the registered external function CvtDateToNum with character date

The use of an additional scalar function is rather annoying and may be forgotten with the next execution. It would be more desirable if the call to the extra scalar function could be avoided.

SQL functions can be overloaded, that means several functions with the same name, but with different parameter definitions (a different number of parameters and/or parameters with different data types) can coexist in the same schema. In this way it is possible to define another UDF also called CvtDateToNum where the date parameter is passed as real date. The following example shows the source code for this additional function. The newly created UDF will receive the date, will convert it into the character representation of a European date and finally will call the external function CvtDateToNum.

```
Create Function MYSCHEMA/CvtDateToNum(
       ParDate  DATE)
       Returns  DECIMAL(8, 0)
       Language SQL
       Specific MYSCHEMA/CVTDAT00002
       Deterministic
       Reads SQL Data
       Returns NULL on NULL Input
       Disallow Parallel
       Not Fenced
   Return CvtDateToNum(Char(ParDate , EUR));
```

Example 5: UDF CvtDateToNumwith a date parameter

The following example shows how to call these functions. In the first call the current date is converted into a character representation and the external function is called after while in the second call the current date is passed directly.

```
select CvtDateToNum(Char(Current_Date, EUR)) as ParAlpha,
       CvtDateToNum(Current_Date) as ParDate
From SysIbm/SysDummy1;
```

Example 6: Calling the overladed UDFs CvtDateToNumwith character and date parameters

In this way, problems with passing parameters as dates with other date formats than ISO can be circumvented.

Have fun when trying it out!

Birgitta Hauser has been a Software Engineer since 1996, focusing on RPG and SQL development on iSeries at Lunzer + Partner GmbH in Germany. She graduated with a degree in business economics, and started programming on the AS/400 in 1992. She is responsible for the complete RPG, ILE, and Database programming concepts for Lunzer + Partner's Warehouse Management Software Package. She also works in education as a trainer for RPG and SQL developers. Since 2002, she has frequently spoken at the COMMON User Group in Germany. In addition, she is chief editor of the iSeries Nation Network (iNN, the German part of iSeries Nation), eNews, and the author of several papers focusing on RPG and SQL.

Data Types in V6R1

Much of the power of SQL databases comes from its very nice data definition language (DDL). By using advanced DDL, applications can be smaller and less complex because common duties like data validation, managing table relationships, and auditing can be turned over to DB2. Let's take a closer look at the new data definition functions recently announced as part of V6R1.

Change Timestamps

The timestamp data type has existed for many releases. One of the uses of a timestamp column is to keep track of when data in the row was last changed. The problem with this approach is that the application had to be coded to update the timestamp column and a developer could forget to do it. Furthermore, if the data was changed outside of the application (e.g. via STRSQL) there was no guarantee that the last change timestamp would get updated as well.

The DB2 development team has solved that problem in V6R1 with the introduction of something called the "change timestamp column". This type of column is automatically updated with the current date/time every time the row is changed. Here is how this type of column is defined:

```
create table MyTable
   (CUSTID integer,
    LNAME varchar(20),
    FNAME varchar(20),
    LASTCHG timestamp generated
always for each row on update
as row change timestamp)
```

In this example, the LASTCHG column would get initialized with the current timestamp when a row is inserted into MyTable. If that row is later updated, LASTCHG would also implicitly get updated by DB2 even if it was not specified in the UPDATE statement. Specifying the data type of the column as "timestamp" is optional since it is redundant with the rest of the column's declaration.

While it isn't clear from the documentation, I don't believe that the timestamp will get updated if the UPDATE statement doesn't really change any columns (i.e. the same data that was read from the row is used on a subsequent update thus leaving the row unchanged). One other note of caution; if you are doing blocked inserts the initial timestamp is not guaranteed to be unique. Another obvious restriction is that a change timestamp column can not contain null values.

This is a very nice feature and eliminates the disadvantages of managing this information at the application level.

Hidden columns

There are times as a database designer that you may want to put information into a table, but don't necessarily want users to see it. An example might be values used by an application to track the state of a transaction or process. While useful for the application, these state values would clearly not be useful to people querying the data or for reporting purposes.

To accommodate columns like this, they can be defined with a hidden attribute in V6R1. So what does "hidden" really mean? The answer is that hidden columns are still visible to anyone writing an SQL statement, but they must be explicitly specified. Here is the syntax for defining hidden columns:

```
create table MyTable
   (CUSTID integer,
    LNAME varchar(20),
    FNAME varchar(20),
    APPSTATE integer implicitly
hidden)
```

If the following select statement is now run, the APPSTATE column would not appear in the output:

```
SELECT * FROM MyTable
```

As you would expect, the default attribute for columns is NOT HIDDEN. Given that columns with this attribute can still be accessed, clearly this function should not be used for security purposes.

One other interesting note; you can alter a column's definition so it becomes a hidden column (or visa versa). This is nice if you have columns you'd like to suppress by default but can't recreate the table. If you have compiled programs that depend on seeing these columns however, you should examine them before altering their IMPLICITLY HIDDEN attribute.

Decimal Floating Point

A new numeric type has been introduced in V6R1 called decimal floating point. The advantage of this data type over traditional packed and zoned number is greater precision. Here is a bit of text from the V6R1 SQL reference manual:

A decimal floating-point number is an IEEE 754R number with a decimal point. The position of the decimal point is stored in each decimal floating-point value. The maximum precision is 34 digits. The range of a decimal floating-point number is either 16 or 34 digits of precision, and an exponent range of 10^{-383} *to* 10^{384} *or* 10^{-6143} *to* 10^{6144} *respectively.*

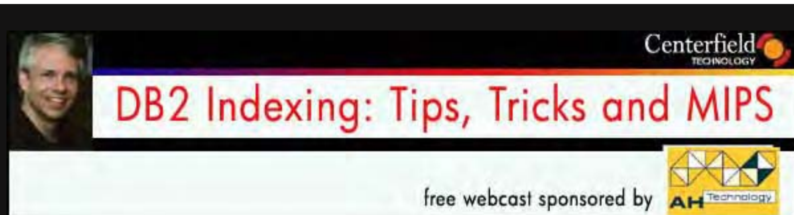
Most applications won't need this extra precision but if needed, this new data type could be very important.

We'll discuss other new V6R1 database features in upcoming newsletters.

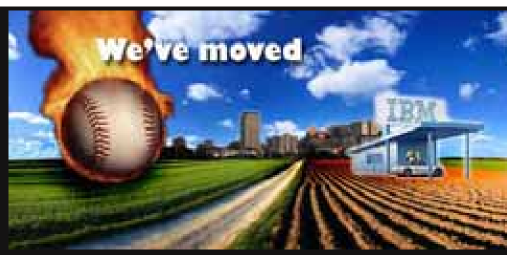
centerfield.technology Events

20th Omni Technical Conference, Tuesday, February 19, 2008
Drury Lane Conference Center - Oakbrook Terrace, Illinois

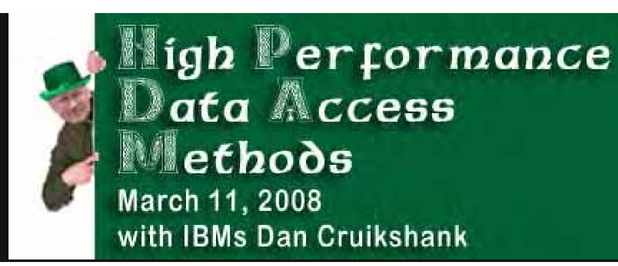
DB2 Indexing: Tips, Tricks & MIPS, Tuesday, February 19 (Australia & New Zealand)
http://www.ahtech.com.au/invite_19feb08_agenda.html



You're cordially invited to our Open House, Wednesday, February 27, 3:00pm – 7:00pm
3131 Superior Drive, Rochester, Minnesota



High Performance Data Access Methods , Tuesday, March 11, 2008 12:00 PM
https://www1.gotomeeting.com/register/845948652



Wisconsin Midrange Computer Professional Association
23rd Annual Spring Technical Conference - Lake Geneva, WI,   April 8th - 10th, 2008