DbAccess
The Manual

Interactive Database Access
with Statistics and Graphics

`Version 2.1.1`

D. M. Mastrovito & W. M. Davis

Princeton Plasma Physics Laboratory
Princeton University
P.O. Box 451
Princeton, NJ 08543

April, 2008

Table of Contents

I. Introduction

DbAccess is an X-windows application, written in IDL, which facilitates interaction with the NSTX Microsoft SQL database. It provides a simple interface for many common statistical and graphical needs of NSTX Physicists. Flexible views and joins are possible, including options for complex SQL expressions (similar to “expression tables” in other applications). The integration of a plotting package from General Atomics (GA Plot Objects) adds extensive graphical and interactive capabilities (and documentation!) with very little additional programming. Multiple linear least squares fits, with or without powers, are easy. It runs on PPPL Linux (e.g. portal) cluster.

II. Setting Up DbAccess

To use DbAccess at PPPL you must:

1) Type “module load nstx/mdsplus” (without quotes) at the Linux command line after logging in (or have in your .cshrc or .bashrc file). This sets up IDL, MDSplus, and the environmental variables for Sybase access. If you have never accessed any NSTX databases from the PPPL Linux Cluster, see http://w3.pppl.gov/~bdavis/swdoc/New_NSTX_User_Setup.txt.

2) You will require a database account, which must be created by a database administrator. Send and email to dbadmin@pppl.gov to request a new database account. At the time your database account is created you will also receive a database password.

3) To optionally see some buttons in color, when using Unix or MacOS X as your X-window manager, you need the following lines in your ~/.Xresources file:

```
Idl*colorbuttons*blue*background:blue
Idl*colorbuttons*red*background: red
Idl*colorbuttons*lightblue*background: lightblue
Idl*colorbuttons*green*background: green
Idl*colorbuttons*purple*background: purple
Idl*colorbuttons*yellow*background: yellow
Idl*colorbuttons*white*background: white
Idl*colorbuttons*gold*background: gold
Idl*colorbuttons*black*background: black
Idl*colorbuttons*magenta*background: magenta
Idl*colorbuttons*orange*background: orange
```

You can copy these entries as follows on a PPPL Unix host:

```
cat /p/nstxusr1/util/init_files/idlcolors.Xresources >> ~/.Xresources
```

or, when running dbaccess, select "Load Button Color Resource" from the "File" menu.

You will also need a file in your home directory called <database_name>.sybase_login for every database you wish to use. At present, most NSTX data is in the database called nstxlogs. Therefore, in your nstxlogs.sybase login file you should have the following information:

```
sqlnstx:8080
sqlnstx
nstxlogs
<unix username>
<database password>
```

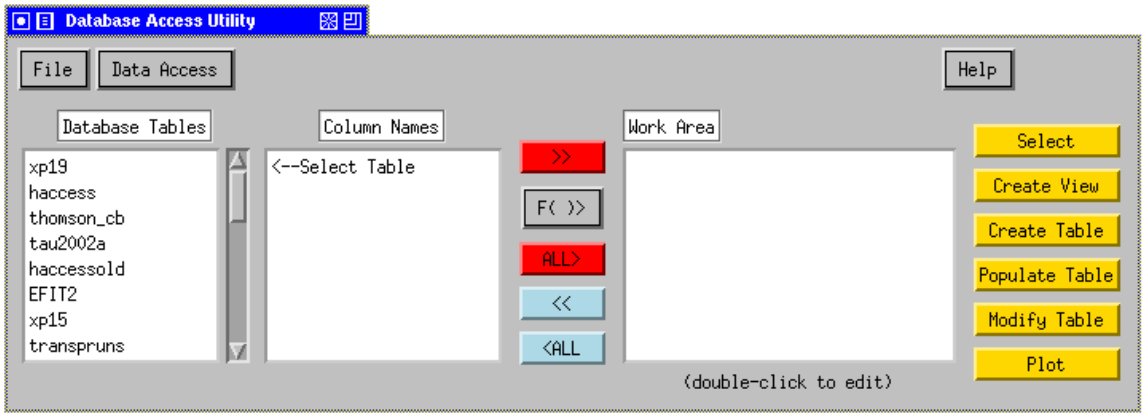
The example above requires an MDSplus server running on the database server. Leave the first line blank, and this won't be necessary.

`Executing /p/nstxusr1/util/setup/overwritedbfiles.sh on a PPPL Unix host should create this file (and others) for you, if it does not exist. However, if you wish to use any other databases you will need to create such a file.`

III. Starting DbAccess

Type IDL at the command prompt. Once inside IDL type dbaccess.

```
portal% idl
IDL> dbaccess
```



When the application is started you should see a list of tables that are available in the nstxlogs database (this is the default database), as well as any views that have been defined. These will be displayed under the heading "Database Tables." Selecting any one of the tables or views in the list will show the list of columns available in that table under the heading "Column Names." You can pull down the "Help" button at the top right at any time to access this document, or get other help.

If you do not see the above window, please send a message to dbadmin@pppl.gov.

IV. Selecting Data From the Database

A. Simple Queries

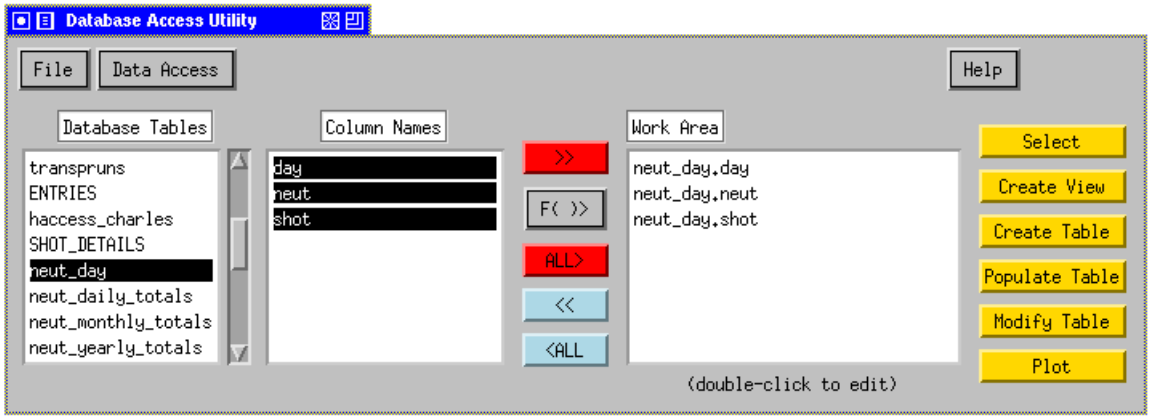
A simple query is one in which you ask for ALL records of ONE table. If you want to query data from more than one table or a subset of one table, see the section entitled "Complex Queries." To select all the data in a particular table:

1) Click on the name of that table in the column labeled "Database Tables." This should cause the columns contained in the selected table to be displayed in the column labeled "Column Names."

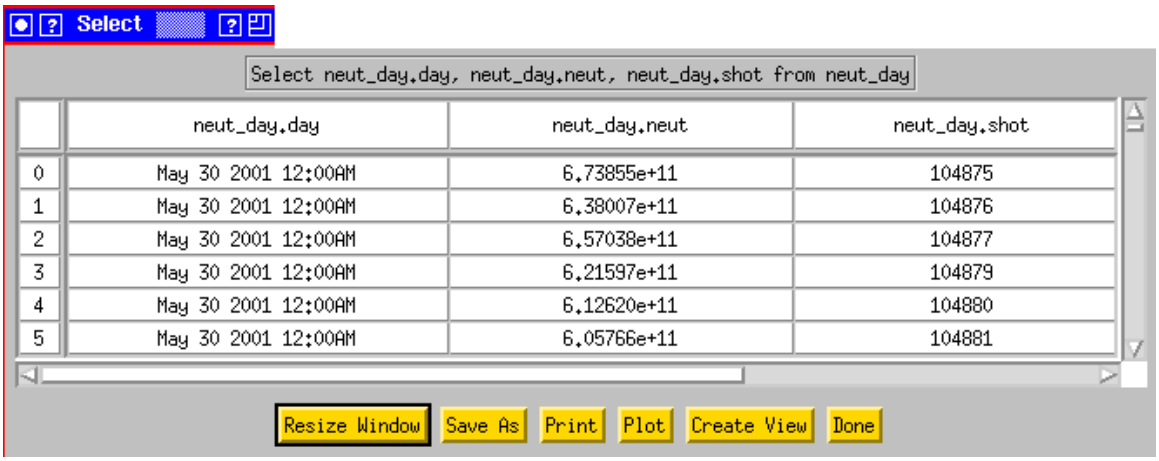
2) Select all of the columns you wish to view in the column labeled "Column Names" with your mouse.

3) Then click the red ">>" button located to the right of the "Column Names" so that the selected columns appear in the column labeled "Work Area."

4) Click the yellow "Select" button.



A new window will appear containing a table with the information you have selected.



B. Complex Queries

i) Using Functions and Arithmetic Operators

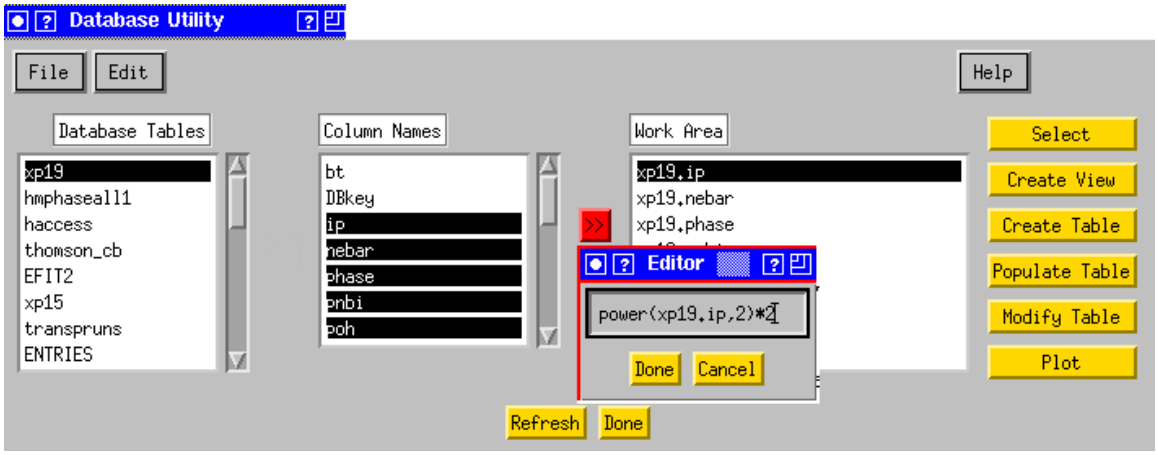
Records are retrieved from SQL databases with "Select" statements. A simple Select statement can include any combination of Functions and Arithmetic Operators supported by SQL (see Appendix I). "Select LOG(Ip), ABS(Bt/1000) from xp19" is a valid Select statement..

These functions perform a calculation, usually based on input values provided as arguments, and return a numeric value.

To use a function or arithmetic operator:

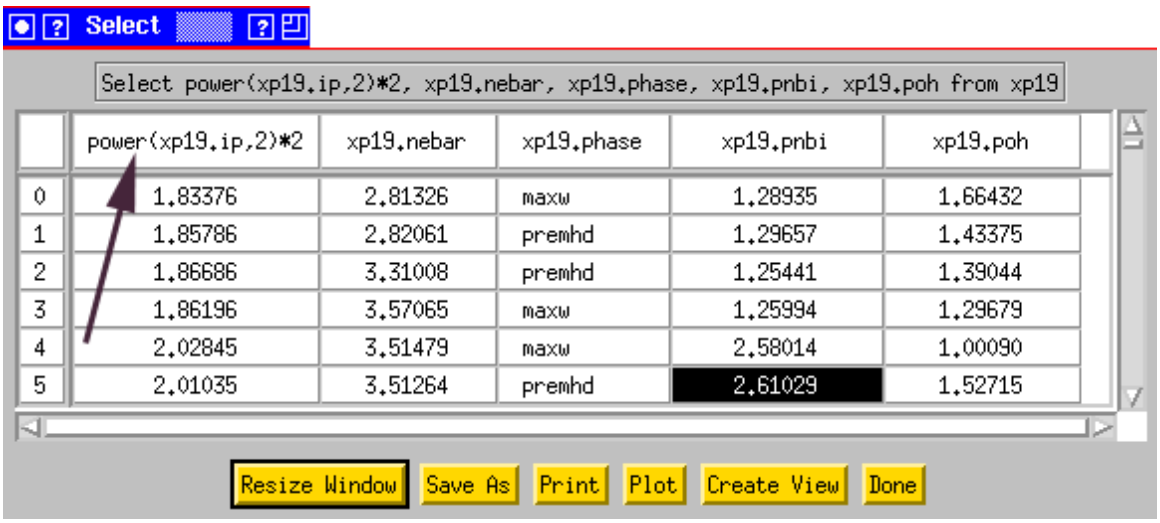
1) Click on the name of the desired table in the column labeled "Database Tables." This should cause the columns contained in the selected table to be displayed in the column labeled "Column Names."
2) Select all of the columns you wish to view in the column labeled "Column Names" with your mouse (either one at a time, or in groups; shift-clicking and control-clicking should work).
3) Then click the red ">>" button located to the right of the "Column Names" causing the selected columns appear to in the column labeled "Work Area."
4) Double-click on the column name in the work area. You will see an Edit window appear.

5) Enter the function and any necessary parameters with parentheses around the column name.



6) Click "Done." You will see that the changes you have made to the column value are shown in the work area.
7) You may now click the "Select" button, "Plot" button, etc.

Rather than double-clicking to add a function, you may include a function while moving the item from "Column Names" to the "Work Area" by pulling down to the desired line from the F( )> button.
In the table returned to you from the "Select" button, you should see a column labeled with the function you entered in the work area:



ii) Complex query without a join

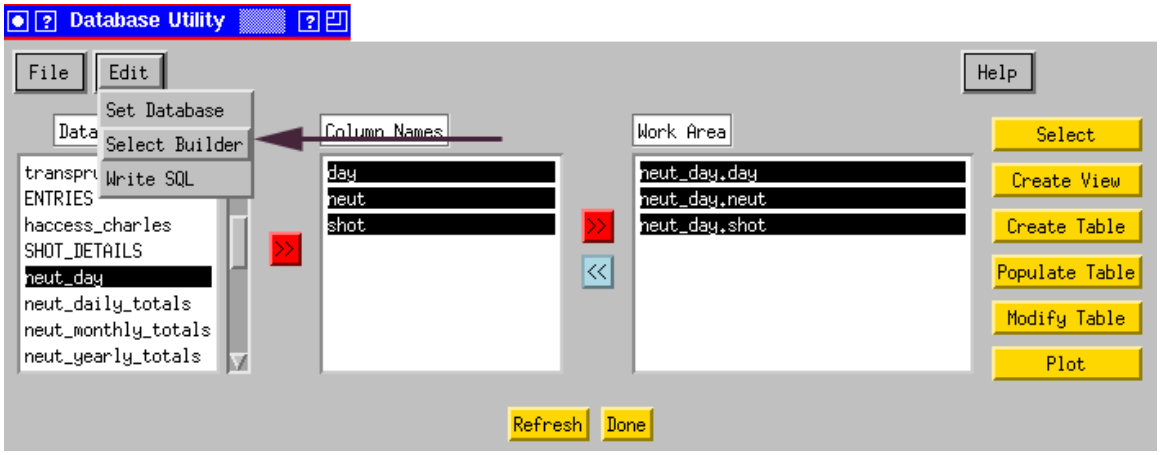
To execute a complex query without a join:

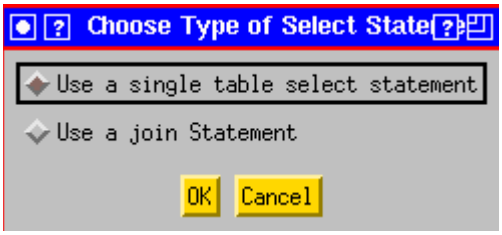
1) Click on the name of that table in the column labeled “Database Tables.” This should cause the columns contained in the selected table to be displayed in the column labeled “Column Names.”

2) Select all of the columns you wish to view in the column labeled “Column Names” with your mouse (shift-clicking and control-clicking should work).

3) Then click the red “>>” button located to the right of the “Column Names” so that the selected columns appear in the column labeled “Work Area.” (You can add functions and mathematical operators to any items placed in the “Work Area” by double clicking on the line or by pulling down to the desired line from the F( )> button, while moving the item.)

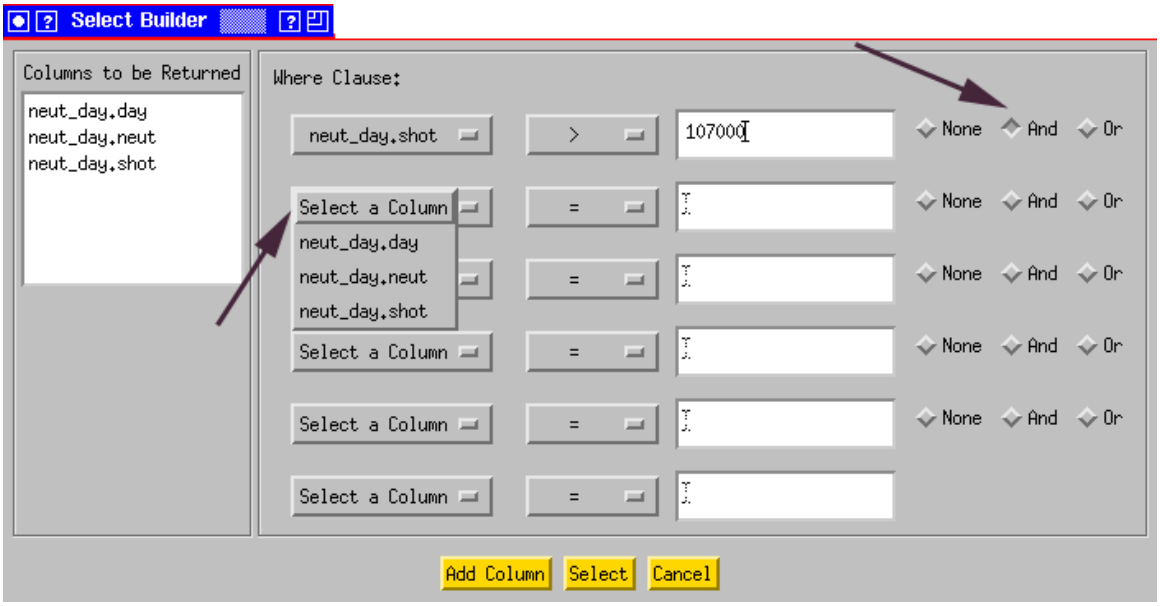
4) Once the desired columns are in the “Work Area,” choose “Constrain Data Set” from the Edit menu.





5) Select “Use a single table select statement” (the default option) and click “OK.”

A new “DbAccess Constrain Data Set” window will appear. The first column contains the items from the “Work Area” in the main window. If you do not explicitly select any of these items all of them will appear in the results of your query. Each of the drop-down boxes in the second column contains all of the items that you placed in the “Work Area” in the main window. To limit a column to a particular range of values you would:



6) Select the column you wish to limit from the drop down list. (E.g., neut_day.shot)

7) Then select a Boolean from the third column. (E.g., ">")

8) Enter a value in the 4th column. (E.g., 107000)

   Completing these steps with the example values will result in the following select statement:

```
Select neut_day.day, neut_day.neut,neut_day.shot from neut_day where
neut_day.shot >107000
```

   Or, continuing, you could:

9) Select "And" or "Or" from the last column (e.g., "And").

10) Select a column from the second drop-down box (e.g., select neut_day.shot again)

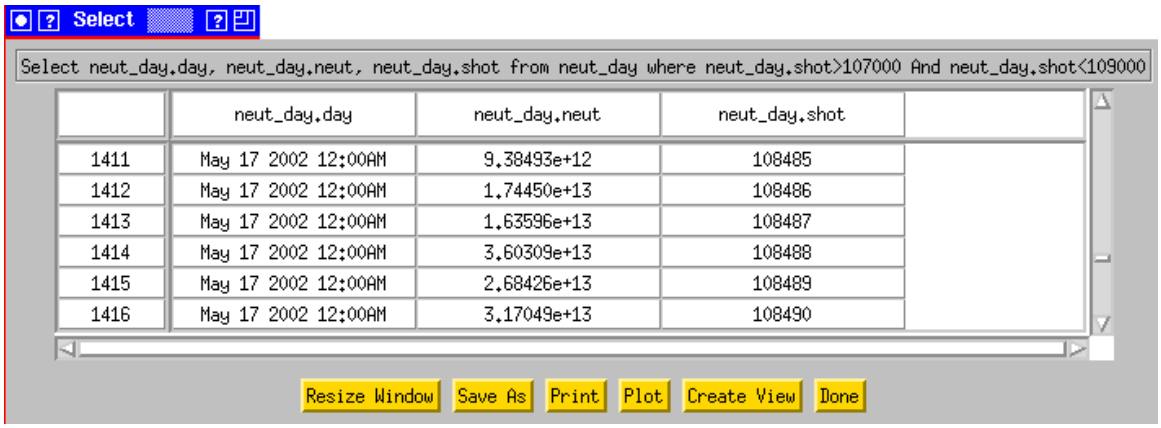
11) Select a Boolean from the drop-down in the 3rd column (e.g., "<")

12) Enter another value in the 4th column. (E.g., 109000)

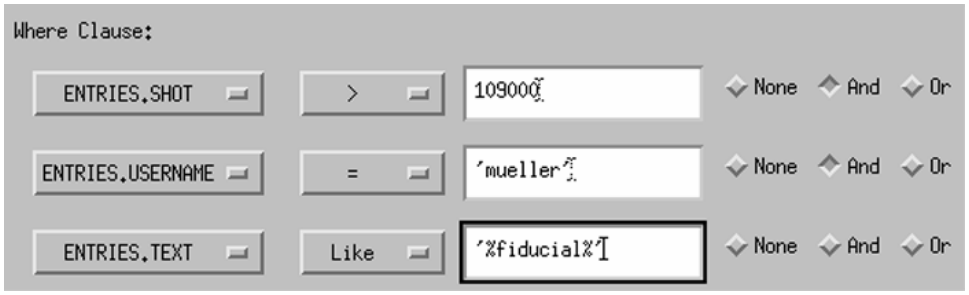
   This would result in the following select statement:

```
Select neut_day.day, neut_day.neut,neut_day.shot from neut_day
where neut_day.shot >107000 and neut_day.shot < 109000
```

13) Click on the yellow "Select" button at the bottom of the window. The "DbAccess Constrain Data Set" window will disappear and another window containing the results you requested will appear. The example above will return all the records in the table neut_day having shot values between 107000 and 109000, exclusively:



To limit a column that is a string data type, such as logbook comments in the "Entries" table or username in the logbook you must use SINGLE QUOTES. This indicates to IDL that the value you are entering is a string.



The above example will select items from the logbook where 1) the shot is greater than 109000, 2) entered by Dennis Mueller, and 3) with a comment containing the word 'fiducial' preceded by or followed by any number of characters. The SQL statement generated would be:

```
Select ENTRIES.SHOT, ENTRIES.TEXT, ENTRIES.USERNAME from ENTRIES
where ENTRIES.SHOT>109000 And ENTRIES.USERNAME='mueller' And
ENTRIES.TEXT Like '%fiducial%'
```

iii) Complex query with a join

Creating a query with a join means that you would like to select data from more than one table based on some criteria common to both tables. A common example would be selecting all records of two tables where the shot numbers in each table are equal to one another. Note that queries with joins can be much slower than queries from one table.

To execute a complex query with a join:

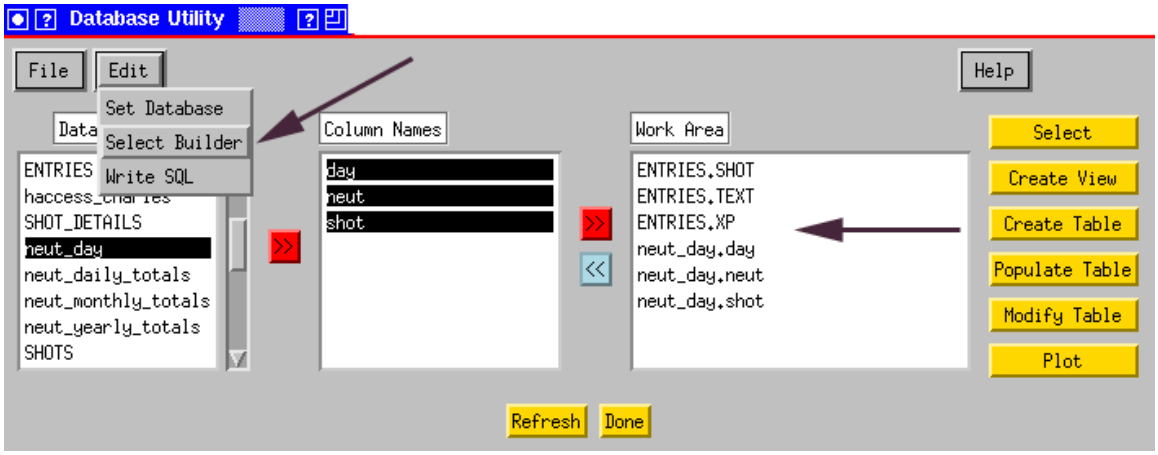
1) Click on the name of that table in the column labeled "Database Tables." This should cause the columns contained in the selected table to be displayed in the column labeled "Column Names."

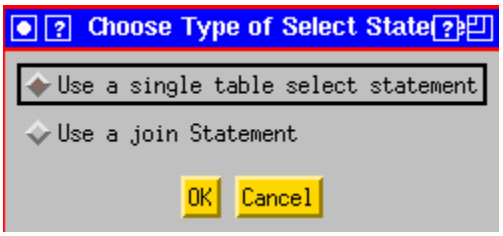
2) Select all of the columns you wish to view in the column labeled “Column Names” with your mouse (shift-clicking and control-clicking should work).

3) Click the red “>>” button located to the right of the “Column Names” column so that the selected columns appear in the column labeled “Work Area.” (As in the previous section “Using Functions and Arithmetic Operators,” you can add functions and mathematical operators to any items placed in the “Work Area.”)

4) Repeat the above steps for columns from another table.

5) Then chose “DbAccess Constrain Data Set” from the “Edit” menu.





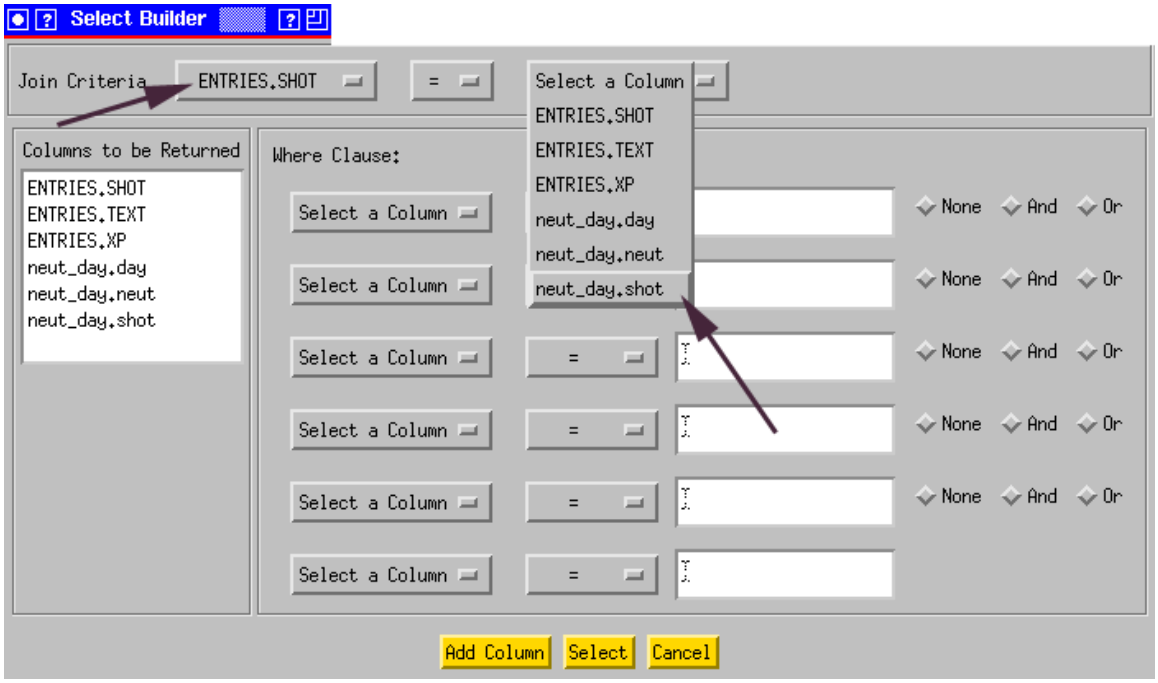
6) Select “Use a join statement” and click “OK.”

A new “DbAccess Constrain Data Set” window will appear. In the first column of the “DbAccess Constrain Data Set” window you will see the items that you placed into the “Work Area” in the main window, e.g., xp19.bt is the Bt column from the xp19 table. If you do not explicitly select any of these items all of them will appear in the results of your query. You will also see that in the section labeled “Join Criteria” each of the drop-down boxes contain all of the items you placed in the “Work Area.” Additionally, all drop-down boxes in the section labeled “Where Clause:” also contain all the items previously placed in the “Work Area” in the main program window. To join the neut_day table with the entries table on the shot column, for example, you would:

7) In the section labeled “Join Criteria” select from the first drop-down box “Entries.shot.”

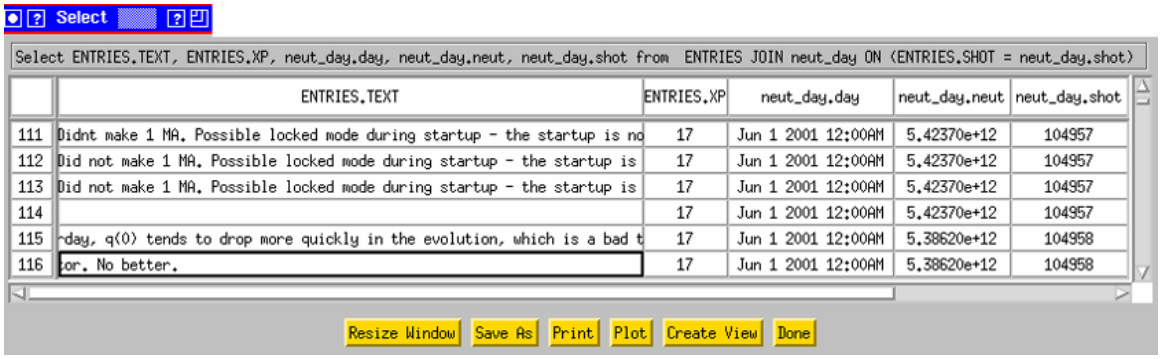
8) Since we are performing an equa-join leave the “=” in the second drop-down box.

9) In the third drop-down box choose “neut_day.shot”



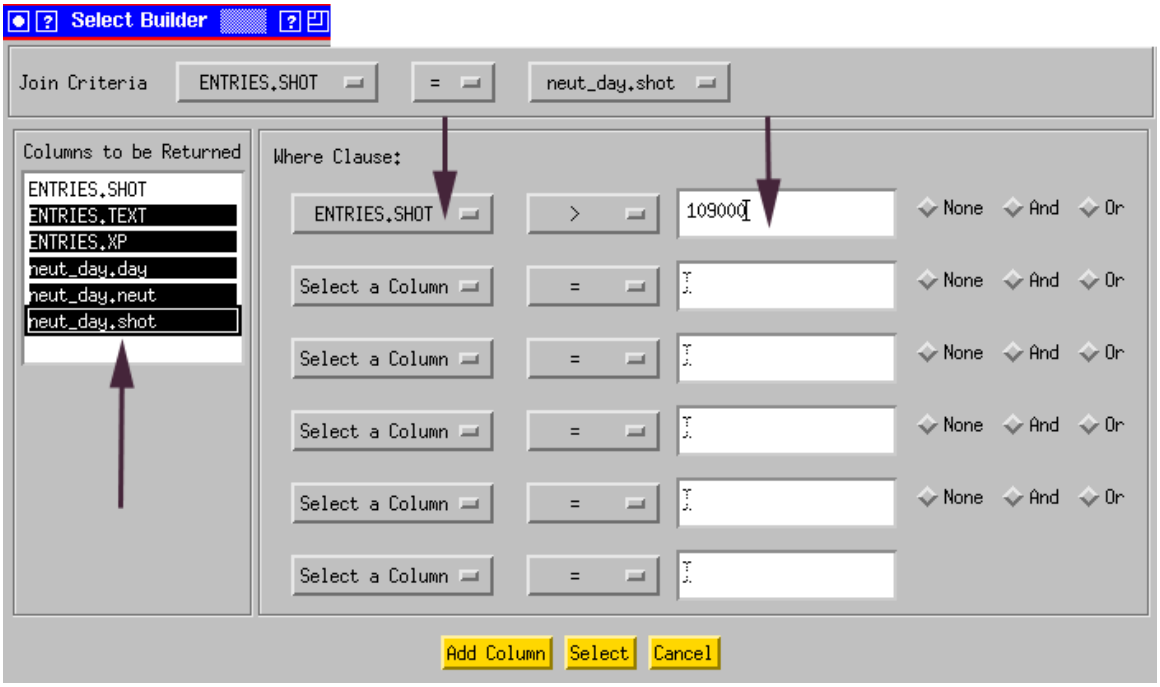
Clicking the yellow “Select” button at this point would result in the following statement being executed:

```
Select ENTRIES.SHOT, ENTRIES.TEXT, ENTRIES.XP, neut_day.day,
neut_day.neut, neut_day.shot from  ENTRIES JOIN neut_day ON
(ENTRIES.SHOT = neut_day.shot)
```



You would then see your results, the text, XP, day and the number of neutrons/day for every shot that is listed in both the entries and neut_day tables.

Alternatively, this selection could be further limited by doing the following:



10) In the area labeled “Where Clause:” Select “Entries.shot.”

11) Then select the Boolean “>” from the second column.

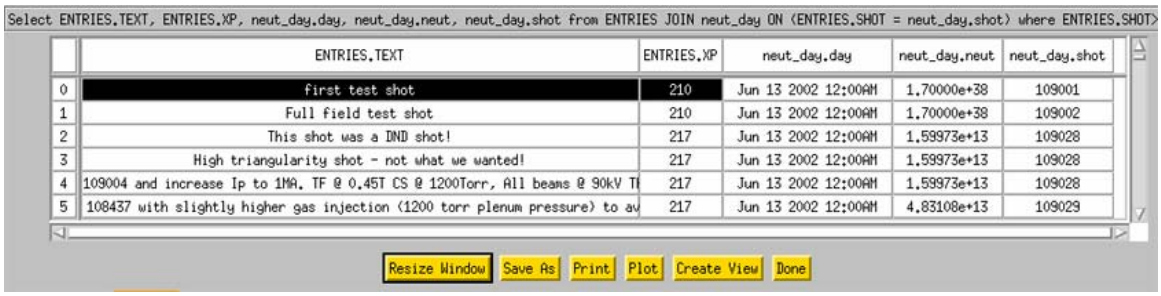
12) Enter a value in the 3rd column (109000).

13) Then in the section columns to be returned select several columns. (Since it is redundant to select both nuet_day.shot and entries.shot when joining on those columns, I have not selected entries.shot)

After completing these steps, the following SQL statement would be executed:

```
Select ENTRIES.TEXT, ENTRIES.XP, neut_day.neut, neut_day.shot from
ENTRIES JOIN neut_day ON (ENTRIES.SHOT = neut_day.shot)  where
ENTRIES.SHOT>109000
```

The following results would be returned:



iv) Adding brackets for logical expressions

For example, if "Or" was checked for any data constraints, an input box will prompt you for any parentheses you wish to add, e.g.,

```
where (xp19.shot>105000 And xp19.shot<105100) Or (xp19.shot>105200
And xp19.shot<105300) Or xp19.shot>105600
```

C. Null Values

When a column has no value for a particular record SQL stores that value as NULL. IDL does not have an equivalent NULL value. Therefore, when data is returned to you, you may see strange values returned when a record has a NULL value in the database. Fortunately, these values are predictable by the datatype of the data returned:

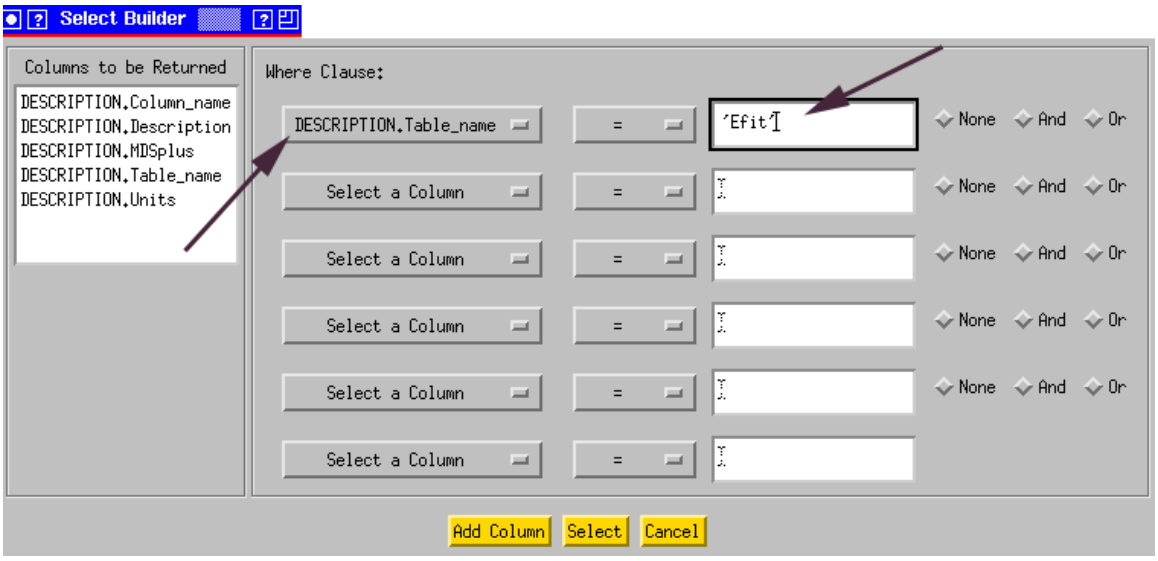
SQL varchar/char:		SQL float:	1.7000000e+38
SQL int:	2147483647	SQL datetime:	Jan 01 1970 12:00:00:000AM

D. Finding Descriptions for a Particular Table

There is a special table in the "nstxlogs" database only called "Description." This table should contain information about every column in the "nstxlogs" database, including the MDSplus definition, the units, and a textual description of each parameter. To find information on the definitions of entries in the EFIT table, for example:

1) Click on the name "Description" in the "Database Tables" column on the main program window. This should cause the columns of the "Description" table to be displayed in the column labeled "Column Names."

2) Then click the red ">>" button located to the right of the "Column Names" so that these selected columns appear in the column labeled "Work Area."



3) Then choose “DbAccess Constrain Data Set” from the “Edit” menu.

4) Since we are only working with the “Description” table, select “Use a single table select statement” and click “OK.”

5) In the first drop-down box select “DESCRIPTION.Table_name.”

6) Leave the “=” in the second column.

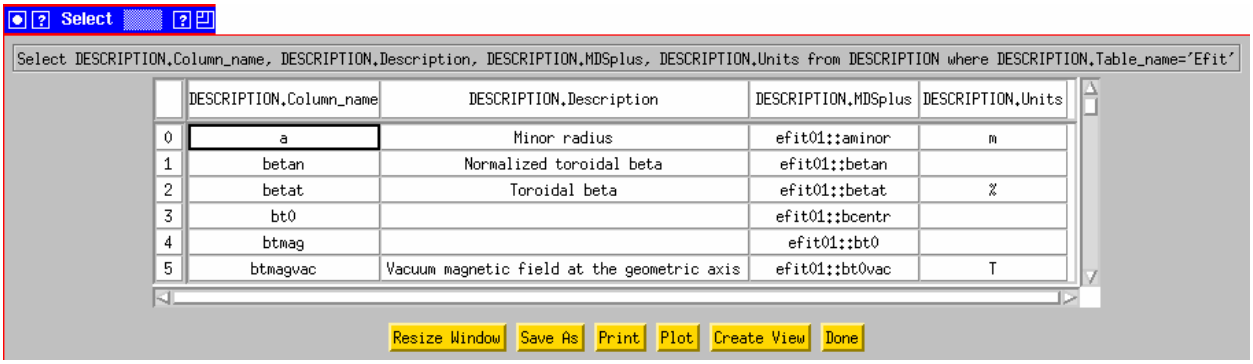
7) Then put ‘Efit’ in the field as shown above.

8) Click the yellow “Select” button at the bottom of “DbAccess Constrain Data Set” window.

   After completing these steps, the following SQL statement would be executed:

```
Select DESCRIPTION.Column_name, DESCRIPTION.Description,
DESCRIPTION.MDSplus, DESCRIPTION.Table_name, DESCRIPTION.Units from
DESCRIPTION where DESCRIPTION.Table_name='Efit'
```

   And the following results will be returned:



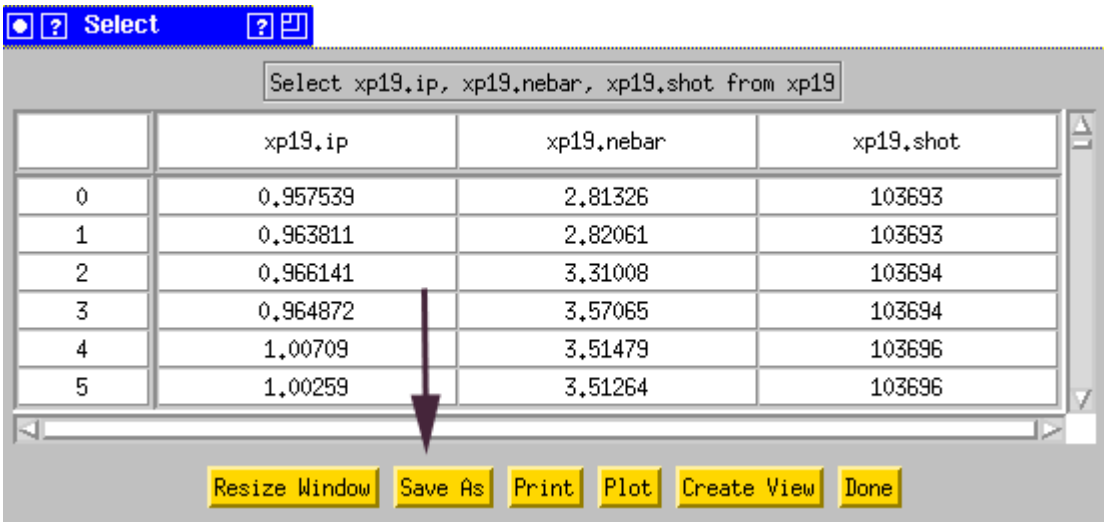
Substantial effort has been put forth to keep this table up to date, however if you are unable to find the information you are looking for please send a message to dbadmin@pppl.gov.

E. Resizing the window

Most windows in DbAccess will reorganize the display when they are resized with the mouse. (In the window manager shown, twm, you would click and drag on the right most square on the blue title tab.) You may also drag the column lines to resize the columns.

F. Saving the Results

At the bottom of any window with returned results from a select statement you will notice that there is a yellow button at the bottom labeled “Save As.”

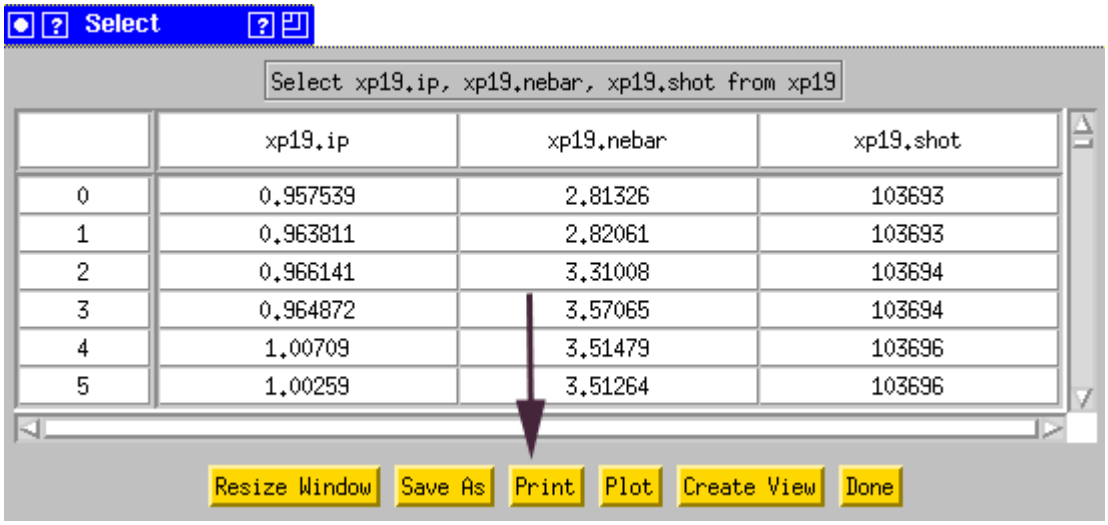


Clicking this button will cause the usual “Save As” dialog box to appear. You can choose the location and name of your file. The data will be saved in a comma-delimited file. Therefore, if you save your file as a .csv file (e.g. myfile.csv) you will be able to

open your file using Microsoft Excel and work with the results of your query using standard spreadsheet functionalities.

G. Printing the Results

At the bottom of any window with returned results from a select statement you will notice that there is a yellow button at the bottom labeled "Print."



Clicking this button will cause the usual "Pick a Printer" dialog box to appear. You can choose a printer from the list or enter the full name of another printer not in the list and your results will be printed in labeled unformatted columns.

V. Creating a View (Expression Table)

A view is similar in concept to the Expression Tables that were in use in the older TFTR INGRES Database. A view can be as simple or as complex as you wish. It represents a set or subset of data contained in other tables or views within the current database. The columns of a view can be based on any combination of columns/views in other tables and any combination of functions and/or arithmetic operators (see section entitled "Using Functions and Arithmetic Operators") as long as they can be described in one SQL statement. An example of a view statement actually in use on the NSTX database is:

```
CREATE VIEW dbo.haccess_charles AS SELECT shot, [time], DBkey, bt0, ip,
0.054 * POWER(nebar_ts * 1e-14, .49) * POWER(bt0, .85) * POWER(psurfa,
.84) AS pth, psurfa, rsurf, pnbi, prad, ptot, a, poh, r0, nebar_ts, 0.65
* POWER(nebar_ts * 1e-14, .93) * POWER(bt0, .86) * POWER(rsurf, 2.15) AS
pthmr, .10 * POWER(nebar_ts * 1e-14, .84) * POWER(bt0,.63) *
POWER(psurfa, .95) * POWER(a / rsurf, .46) AS pthaspect, .54 *
POWER(nebar_ts * 1e-14, .49) * POWER(bt0, .85) * POWER(psurfa, .84) AS
pthsurfa, .65 * POWER(nebar_ts * 1e-14, .93) * POWER(bt0, .86) *
POWER(rsurf, 2.15) AS pthmajor, 1.73 * POWER(nebar_ts * 1e-14, .63) *
POWER(bt0, .72) * POWER(a, .82) * POWER(rsurf, .99) AS pthryter, phase,
```

```
wbdot, wpdot, 0.050 * POWER(nebar_ts * 1.0E-14, 0.46) * POWER(bt0, 0.87)
* POWER(psurfa, 0.84) AS pth2, 1.67 * POWER(nebar_ts * 1.0E-14, 0.61) *
POWER(bt0, 0.78) * POWER(a, 0.89) * POWER(rsurf, 0.94) AS pthryter2,
(pnbi + poh - wpdot) / 1E6 AS ploss FROM  dbo.haccess
```

You don't need to understand the syntax of this statement, but its good to observe that you can have a column in your view that is identical to one in another table, such as "bt0" in the first line of the above statement. Or you can have one that is based on a mathematical combination of several columns, such as:

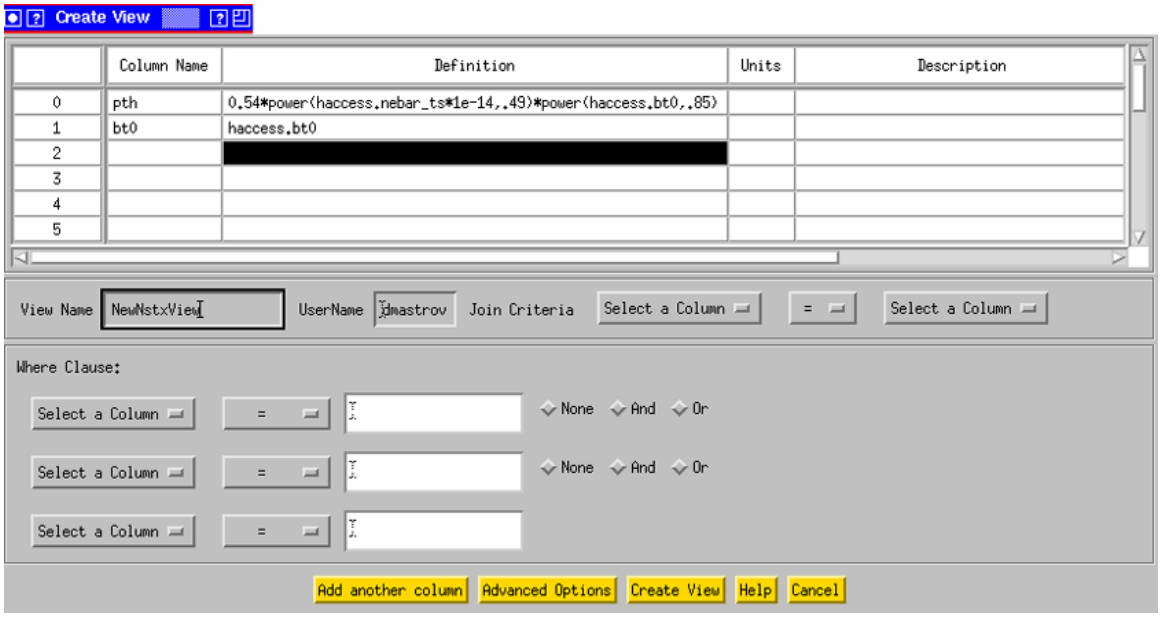
```
0.054 * POWER(nebar_ts * 1e-14, .49) * POWER(bt0, .85) * POWER(psurfa,
.84) AS pth
```

where "pth" is the column name in your view and is defined as a mathematical combination of "nebar_ts," "bt0," and "psurfa," which are columns in the "haccess" table.

Notice also, that as the "CREATE VIEW dbo.haccess_charles AS SELECT" syntax indicates, the view is nothing more than a saved database query. Every time you request data from your view the select statement that represents your view is executed. For this reason, your view will always have the most up to date information from the database.

The SQL statement that is used to define your view can also contain information from multiple tables using join syntax and/or a subset of one or more tables by limiting shot number or another parameter, etc. (see section entitled "Selecting Data from the Database"). To create your own view:

1) If you want to include a join in the statement for your join or you want your join to look at a subset of data from another table, bring the relevant column names into the "Work Area" column on the main program window. At this point you can add any functions or mathematical operators you wish (see section entitled "Using Functions and Arithmetic Operators"). Click the yellow "Create View" button on the right side of the main program window.



2) Using the example above, I have named my columns “pth” and “bt0” and put in definitions. Notice that you must specify table names as in “haccess.bt0,” which specifies that the column “bt0” is in table “haccess.” This is important as there may be more then one column named “bt0” in the database.

3) You should see that your Username is already filled in for you. You need to enter a name for your view in the field labeled “View Name.”

4) You can put in join criteria and column restraints as in the section “Selecting Data from the Database.”

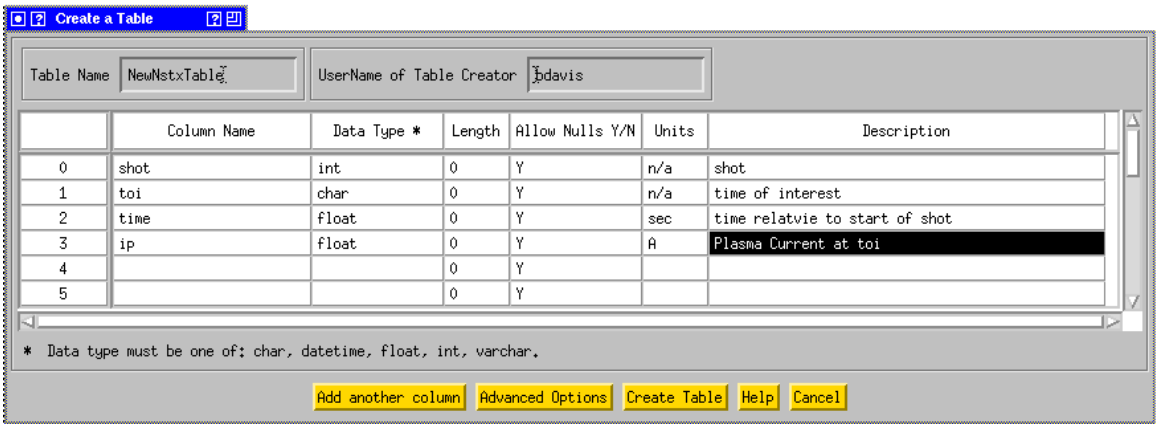
5) Click the yellow “Create View” button at the bottom of the window.

6) Click the yellow “Refresh” button at the bottom of the main program window. You should see that your view appears in the list of tables under “Database Tables.” You can now interact with this view as if it were a regular table.

VI. Creating a Table

The first few times you create a table you may want to familiarize yourself with the procedure or perhaps test a few different versions of a script to load them. If so you may want to create your table the first time in the test database. If so, see the section entitled “Working with a Different Database.” To create a new table:

1) Click the yellow “Create Table” button on the main program window. This will cause a new “Create a Table” window to appear.



2) Enter a unique name for your new table in the field labeled “Table Name.”

3) You should see that your username is already in the field labeled “UserName of Table Creator.” In future versions of DbAccess this will be used to implement database security.

4) Enter names for the columns of your table in the column labeled “Column Name.”

5) Enter the SQL data type of each column in the column labeled “Data Type.” The choices are:

char

Fixed-length non-Unicode character data with length of *n* bytes. *n* must be a value from 1 through 8,000. Storage size is *n* bytes. The SQL-92 synonym for **char** is **character**. Use **char** when the data values in a column are expected to be consistently close to the same size.

varchar

Variable-length non-Unicode character data with length of *n* bytes. *n* must be a value from 1 through 8,000. Storage size is the actual length in bytes of the data entered, not *n* bytes. The data entered can be 0 characters in length. The SQL-92 synonyms for **varchar** are **char varying** or **character varying**. Use **varchar** when the data values in a column are expected to vary considerably in size.

datetime

Date and time data from January 1, 1753 through December 31, 9999, to an accuracy of one three-hundredth of a second (equivalent to 3.33 milliseconds or 0.00333 seconds). Values are rounded to increments of .000, .003, or .007 seconds. Example: 2002-06-13 08:08:07.080

int

Integer (whole number) data from -2^31 (-2,147,483,648) through 2^31 - 1 (2,147,483,647). Storage size is 4 bytes. The SQL-92 synonym for **int** is **integer**.

float

Is a floating point number data from - 1.79E + 308 through 1.79E + 308. *n* is the number of bits used to store the mantissa of the **float** number in scientific notation and thus dictates the precision and storage size. *n* must be a value from **1** through **53**.

6) The "Length" column is optional. If specified it will define the precision of the data type chosen in the previous step. If not specified the default precision for each data type will be used:

Default Precisions	
Char	1
Varchar	1
Datetime	Cannot be set 23
Float	53
Int	Cannot be set 10

7) In the "Allow Nulls" column you should specify with a Y (meaning this column will allow NULL values) or N (meaning this column will not allow NULL values). This is an important consideration especially if you will be loading your table by an automatic script (see section entitled "Populating a Table"). SQL will not create a record for you if you try to create one with an empty or unspecified value for a column that does not allow NULL values and this could result in no entry for that particular shot. If you have a column named shot, for example, this column should likely not allow NULL values. On the other hand, if it is possible that a particular value will not be able to be read from MDSplus for any given shot, it is safer to allow NULL values for that column. The default is Y (allow NULL values). If you do not want a given column to allow NULL values you will need to change this to N for that column.

8) In the "Units" column enter the units for the given column. If there are no units for a column (such as shot number) enter "N/A." This information will be stored automatically in the "Description Table." (See section entitled "Finding Descriptions for a Particular Table")

9) In the "Description" column enter a textual description of what your column will represent. This is very useful for other people who may want to make use of your table, and will also serve as a good reminder for you. This information will be stored automatically in the "Description Table," as was the contents of the "Units" column.

10) Click the yellow “Create Table” button at the bottom of the window. If you have left any items blank you will receive an error. Otherwise you should receive a message stating that your table has been created successfully.
11) Click the yellow “Refresh” button on the main program window and you should see the addition of the table you have just created.

In the “Create Table” window, by default, there are 20 rows, which allows you to create a table with 20 or less columns. If your table needs more columns click the yellow “Add another column” button at the bottom of the window, which will create more rows, giving you room to define more columns.

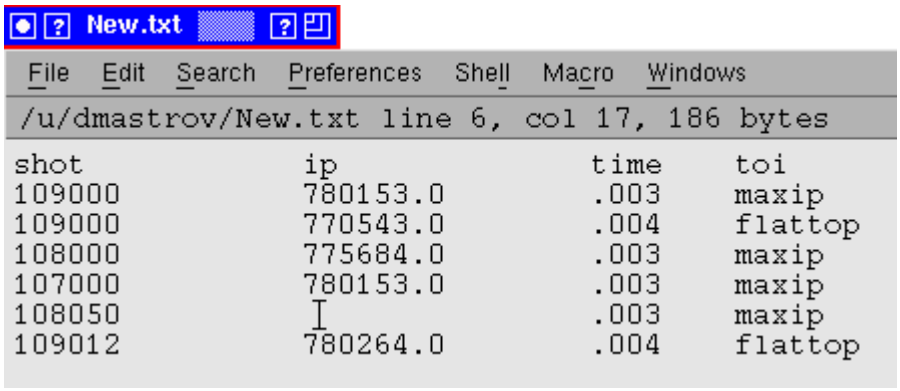
All tables are created with one additional column called “dbkey” which is created as a primary key for your table (a column that cannot have any NULL values and is always unique). This column is mainly used for house-keeping and to allow future capability of table value modifications from within DbAccess. For this reason, it is not necessary for you to create a primary key for your table.

VII. Populating a Table

A. From a File

To populate an already existing table from a file:

Create a text file with the data you wish to enter into your table. The first line of your file should be the same as the column headings in the database table (order is not important). The columns in your file should be tab delimited. Using the example of the table we created in the previous section, your data file might be the following:

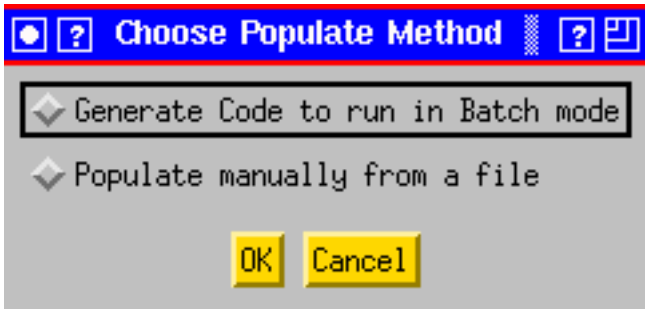


Notice that there is one value for ip that is missing. In the column with the missing value there must be a space “ .” Since the values are parsed by tabs, the space will indicate to the application that this value is meant to be empty. Notice also that there are two entries for shot 109000. This will not cause a problem, because when you created your

table a unique key was automatically created for you. Furthermore, since we did not specify a length for the char column "toi" when we created the table, this column was created with the default length of 1 character. Therefore you will see when you retrieve the values from this table that only one character appears for each entry (i.e. instead of "maxip" the database contains only "m"). To change the length of the column "toi" or to add or remove columns from your table you will need to modify it (see section entitled "Modifying a Table").

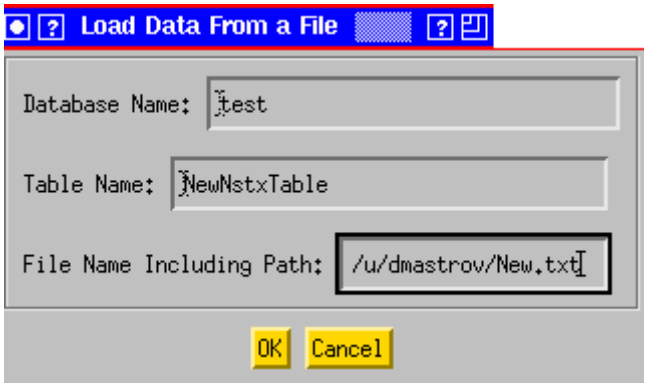
Select the table you would like to populate from the "Database Tables" column on the main program window. You should see the columns contained in that table listed in the "Column Names" column.

Click the yellow "Populate Table" button on the right side of the main program window. You will see the following dialog:



Choose "Populate manually from a file" and click "OK."

The Database Name and Table Name should appear in the "Load Data From a File" Dialog. Enter the file name and path to your text file containing your table data in the "File Name Inclucing Path:" field. Click "OK"



Note that if you change the database name in this dialog the database context will be changed and the application will attempt to look for the NewNstxTable in a different database. This will also effect the database context when you return to the main window.

You could confirm this by clicking the "Refresh" button. To change back to a different database see section "Working with a Different Database."

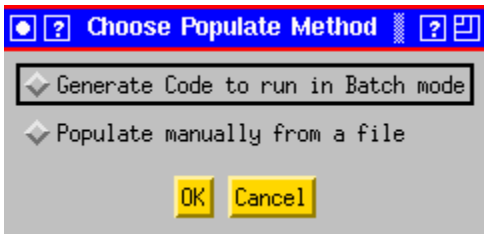
B. From a Script

Populating a table from a Script does not actually put any values into the database for you. What it allows you to do is define the values that you want in the database in terms of some calculation on a particular MDSplus node and it generates a script for you based on those definitions that is robust and can be run on a shot by shot basis automatically to load your data into the database. It does, however, make an assumption about the structure of the table you have created, namely that it has a shot number and a time of interest, at which certain values calculated from MDSplus signals are entered. If this is not the structure of your table you will probably need to edit the script that is generated for you or in some cases it might be best to modify an already existing script. If you need help you should send a message to dbadmin@pppl.gov .

To generate a script for loading your table:

1) Select the table you would like to populate from the "Database Tables" column on the main program window. You should see the columns contained in that table listed in the "Column Names" column.

2) Click the yellow "Populate Table" button on the right side of the main program window. You will see the following dialog:



3) Select "Generate Code to run in Batch mode" and click "OK."

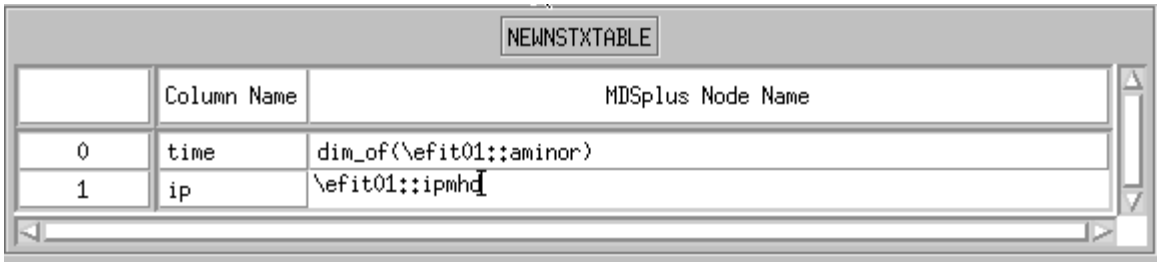
4) This will open a new "Create a Table Batch File" window. The top part of this window (shown below) is used to define an MDSplus node to associate with each of the parameters or columns in your table. You will see that the names of the columns are read from the database and filled in for you, excluding the special columns shot and toi, since shot has a well-known meaning and the times of interest will be defined on the bottom part of the window. It is not required that you supply an MDSplus Node Name, and in fact if there is a complicated set of commands that will be used to calculate a particular node you should NOT enter anything in the MDSplus Node Name column. However if the column value is simply an MDSplus signal at a time of interest you should enter the MDSplus node here. In this example, time is defined by the MDSplus tdi command

'dim_of(\efit01::aminor'. This means that every shot your script will make one entry per time of interest like:

```
time=mdsvalue('dim_of(\efit01::aminor)')
```

The value placed in the database for each record would be time[toi_index]

The code that is generated at the end of this process has built into it the ability to fully customize these calculations.



5) The second part of the window is used to define the times of interest. These are the times at which all the other signals in the table will be put into the database. In the following example two times of interest have been defined. The first is called 'maxip' and is defined as the time point *previous* to where '\efit01::ipmhd' was at a maximum, since the offset is set to –1. The value put in the database for column "toi" would be 'maxip' and its index would be used to calculate all other signals defined in the previous section. The second is called 'maxwtot' and is defined as the time where the signal '\efit::wmhd' is closest to 220,000. The entry in the database for toi would be 'maxwtot' in this case and its index would be used to calculate all the other signals as defined in the previous section.

Clicking the "Populate Table" button on the main window will generate an IDL script for you named <yourtalbename>db.pro and place it in your home directory. In this case /u/dmastrov/NewNstxTabledb.pro

This script is rather complicated as it contains error checking, etc. However, the framework is already present in the script for you to do much more complex calculations for both your times of interest and any signals. Send a message to dbadmin@pppl.gov if you need assistance in modifying this script. After the script is complete it will need to be moved onto VMS where it will be run via a batch queue at the end of every shot. Alternatively, you can run this script yourself from inside idl:

```
IDL>for shot=109200,11000 do NewNstxTabledb, shot
```

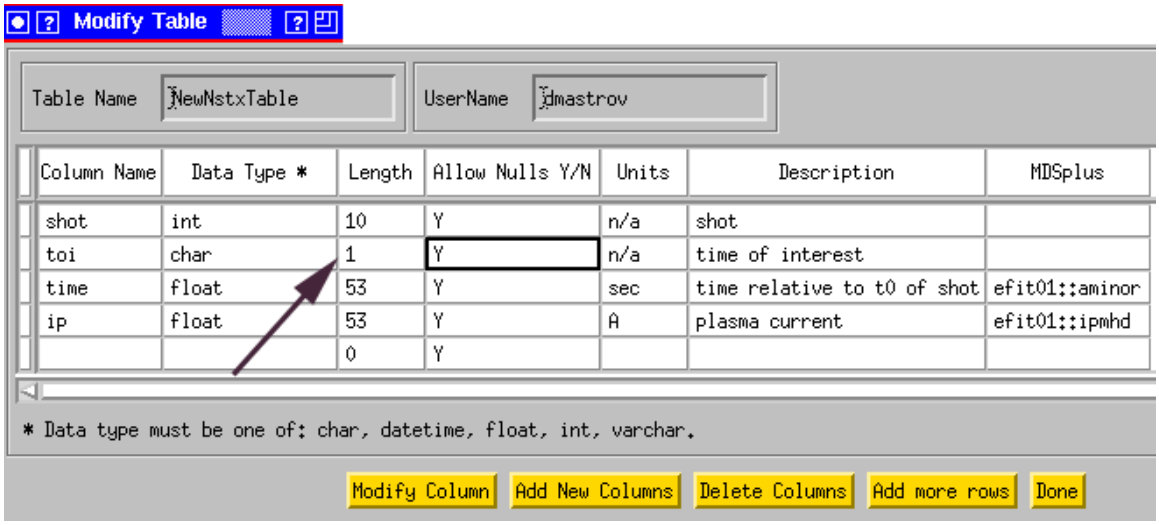
If you want your script to run in batch mode, you should send a message to dbadmin@pppl.gov in order to have your script put into an NSTX shot cycle queue.

VIII. Modifying a Table

Modifying a table in the context of this application involves changing the structure of a table and not changing the values of particular records (in the future this will be possible from DbAccess). Lets say you have created a table and would like to add a new column or change the data type of a particular column. Perhaps you simply want to change how the units are listed in the description table. All these things are done using the "Modify" functionality of DbAccess. To modify a table, start off on the main program window:

1) Select the table you would like to modify from the "Database Tables" column. Click the yellow "Modify" button on the right side of the main program window.

2) You will see a new "Modify Table" window appear and in it you will see the relevant information for the table you have chosen, which is read from the database.



3) Using as an example the table we created in the section entitled "Creating a Table," you will see the data types and lengths of the columns you previously defined. As pointed out the length of the toi column is indeed 1. Now is a good time to change it to accommodate our full time of interest names. Change the 1 in the Length column to 10 or 15. Being sure that some portion of the row for column "toi" is selected, click the "Modify Column" button.

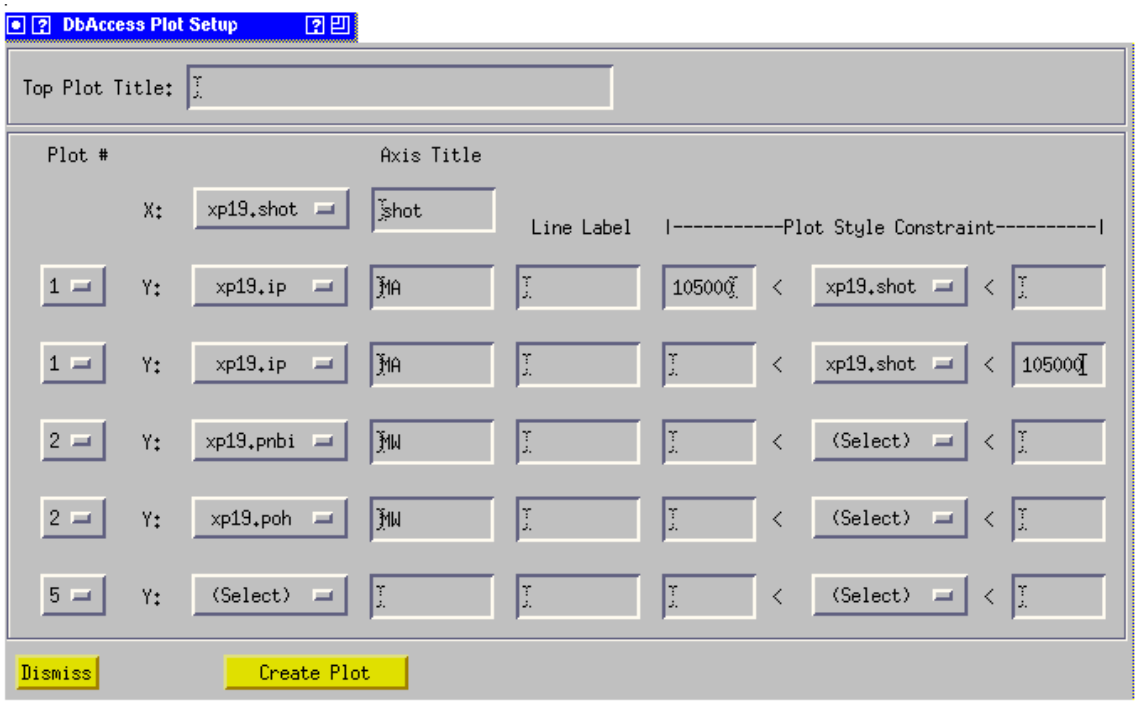
4) Also here, you can change a column to disallow null values. For instance, we will never want to have an entry in the database with a NULL shot number so we should change the Y to an N in the "Allow Nulls Y/N" column for shot. Being sure that

some portion of the row for column “shot” is selected, click the “Modify Column” button.

5) It is also possible to add new columns here by simply typing in the information for the new column below those already existing. Make sure you have some part of the row containing the column you are adding selected and clicking the yellow “Add New Columns” button at the bottom of the window. It is, however, important to realize that if your table already contains data, the new column you create will be filled with NULL values for all previous records and you will need to take some course of action to fill that column for all previous records. For this reason, if your table already contains data, it does not makes sense to add a new column with an “N” in the “Allow NULL Y/N” column and doing so will cause and error and the column addition will not take place. Also, if you already have a script created to load your table on a shot-by-shot basis you will need to modify this script to accommodate the new column you have added.

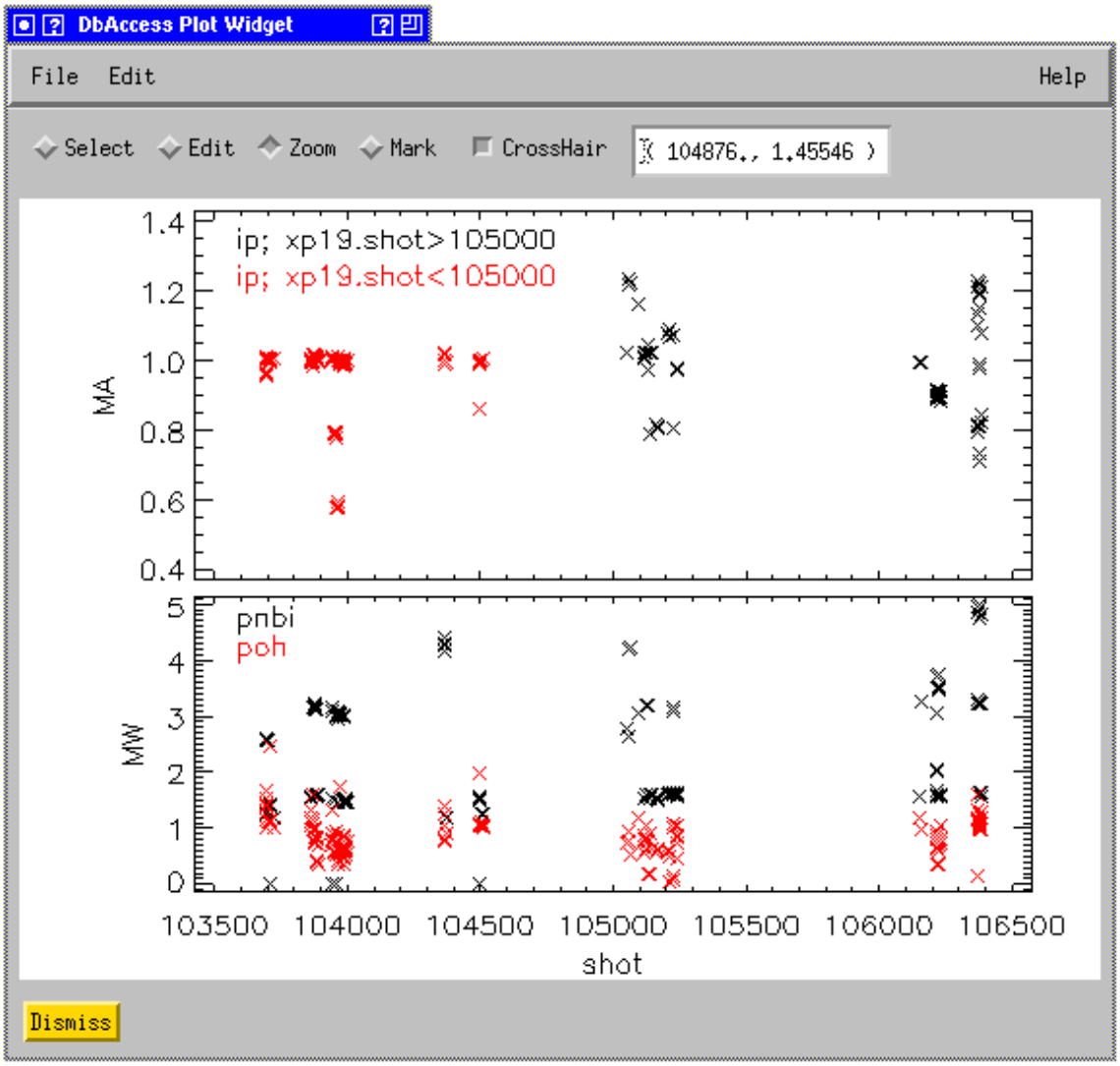
IX. Plotting

When you click on the Plot button from the Database Access Utility window or the DbAccess Selection window, you will get a dialog window that lets you configure plots from the variables you have placed in the Work Area column of the main widget.



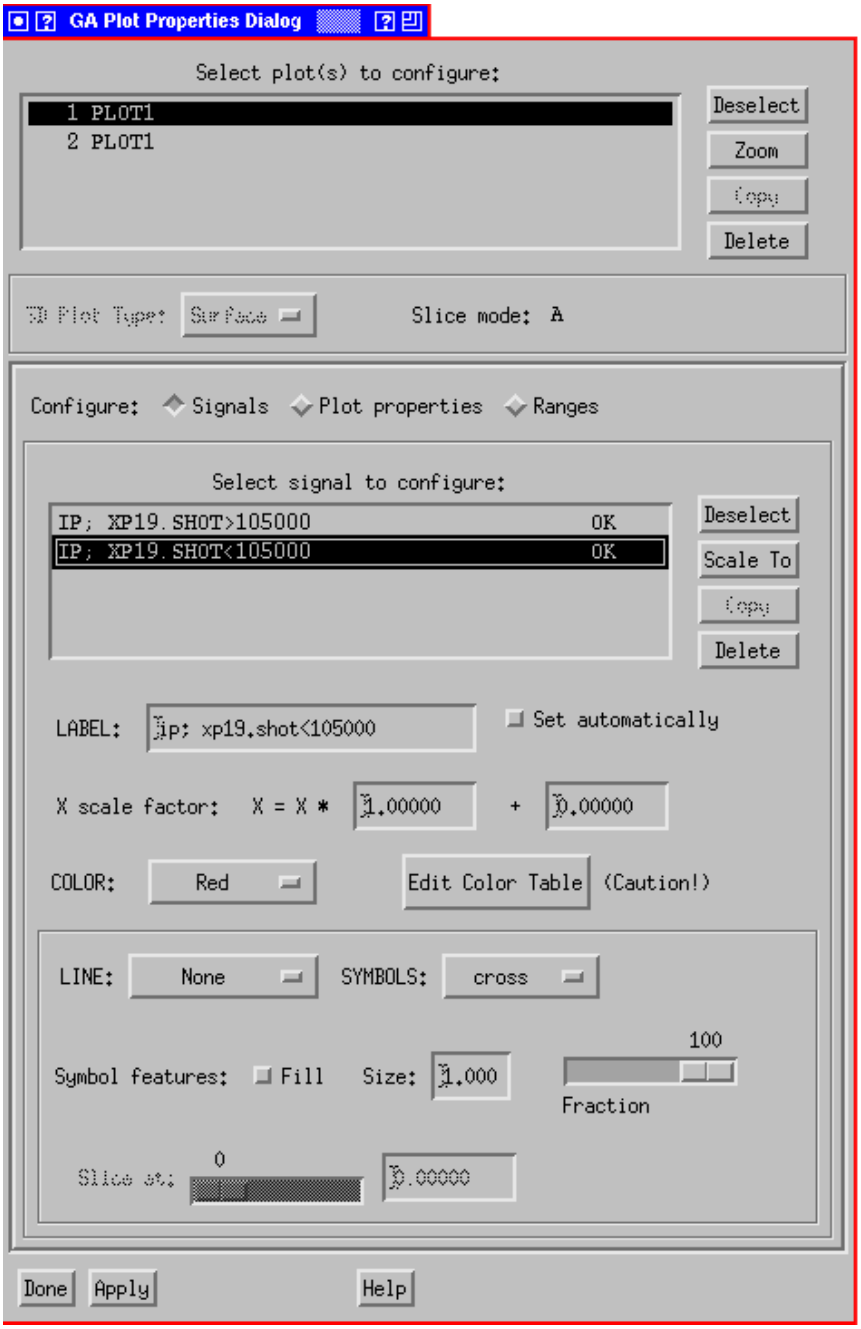
In the example above, plasma current (Ip) would be plotted on one axis for shots less than or equal to 105000, and greater than or equal to 105000, as different data sets. These will be plotted with different colors (different line styles and plot symbols may also be

specified). In the second plot frame, Neutral Beam Power and OH Power will be overlayed for all shots.



GA Plot Objects (like those in ReviewPlus) are used for graphics. When a plot window is brought up, you will see a set of radio-buttons that indicate the modes for mouse operations, and a check-button for turning on and off the cursor tracking. When the Zoom button is checked, you may draw a zoom box with the left mouse button. A single click with the middle button returns to autoscaling. When the Mark button is checked, and a point is clicked on, the corresponding row in the data table (if opened) is highlighted. Right clicking on a plot will bring up a SCOPE-like configuration options.

To change symbols, colors, character sizes, etc., of any data set, select “Set Plot Appearance” under the edit menu. This should bring up the “GA Plot Properties Dialog” window:

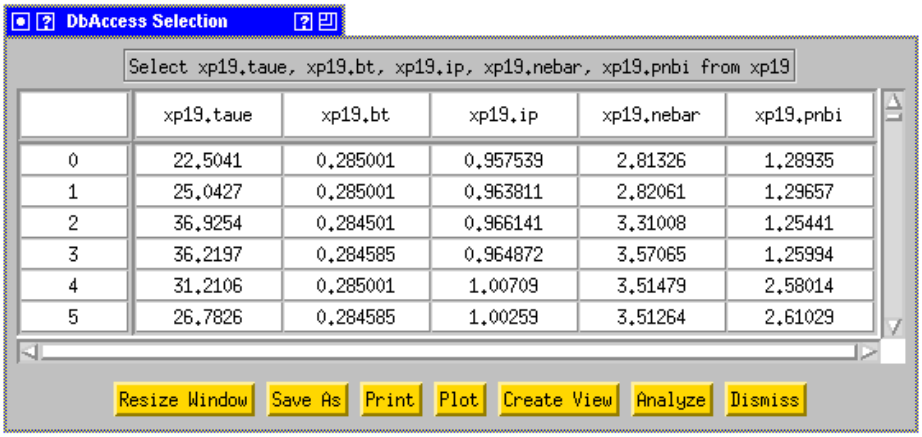


See http://web.gat.com/comp/analysis/uwpc/reviewplus/manual/ for more detail.

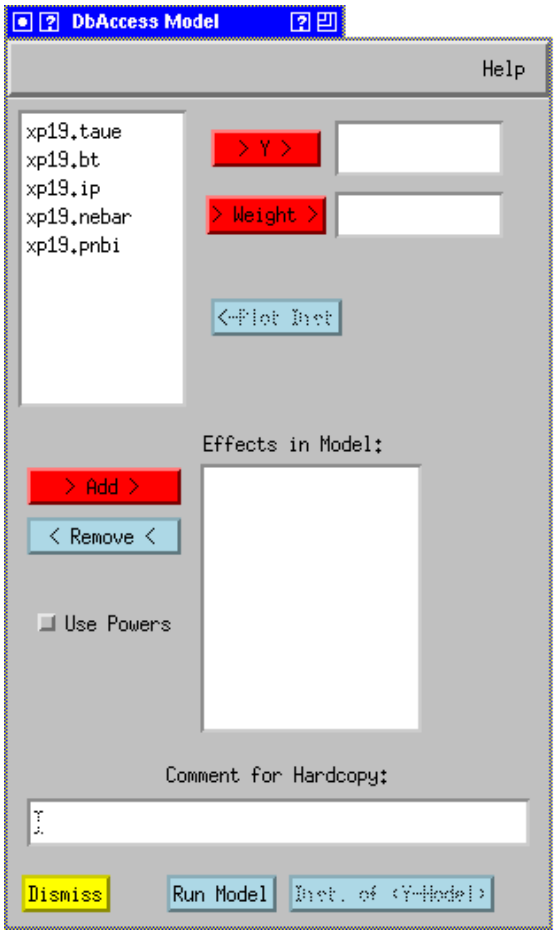
You may wish to use a commercial PC or Macintosh product, such as Excel, for plotting (see the section on "Saving your Results").

X. Statistical Features

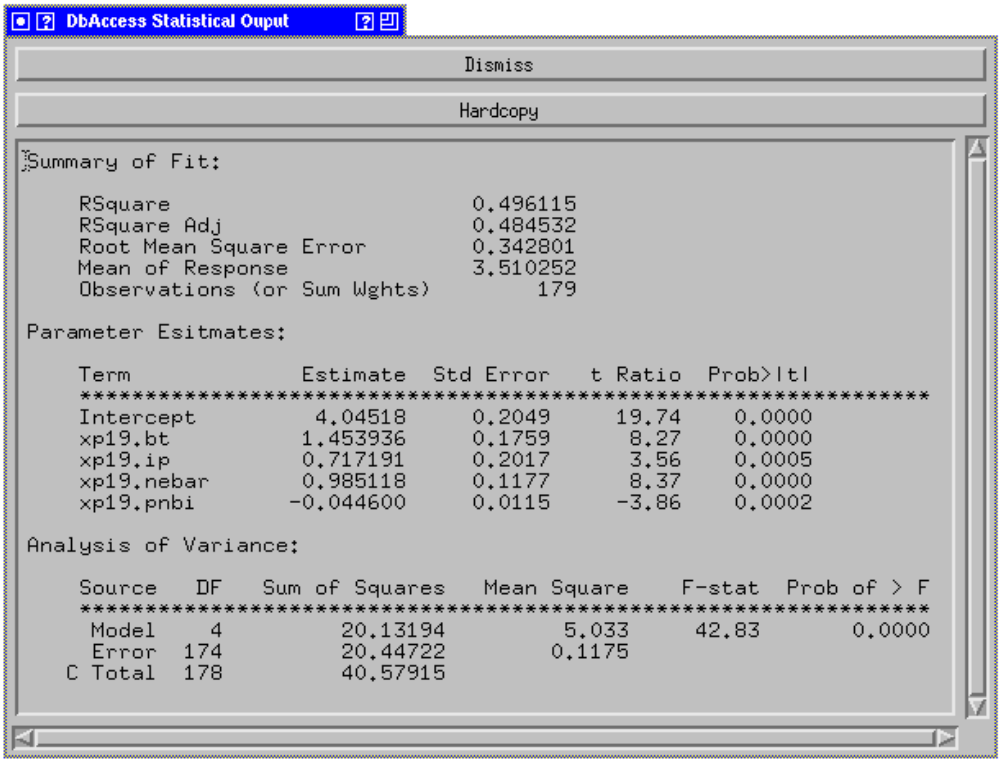
After moving the desired columns into the Work Area, click on the Select Button to bring up the DbAccess Selection window:

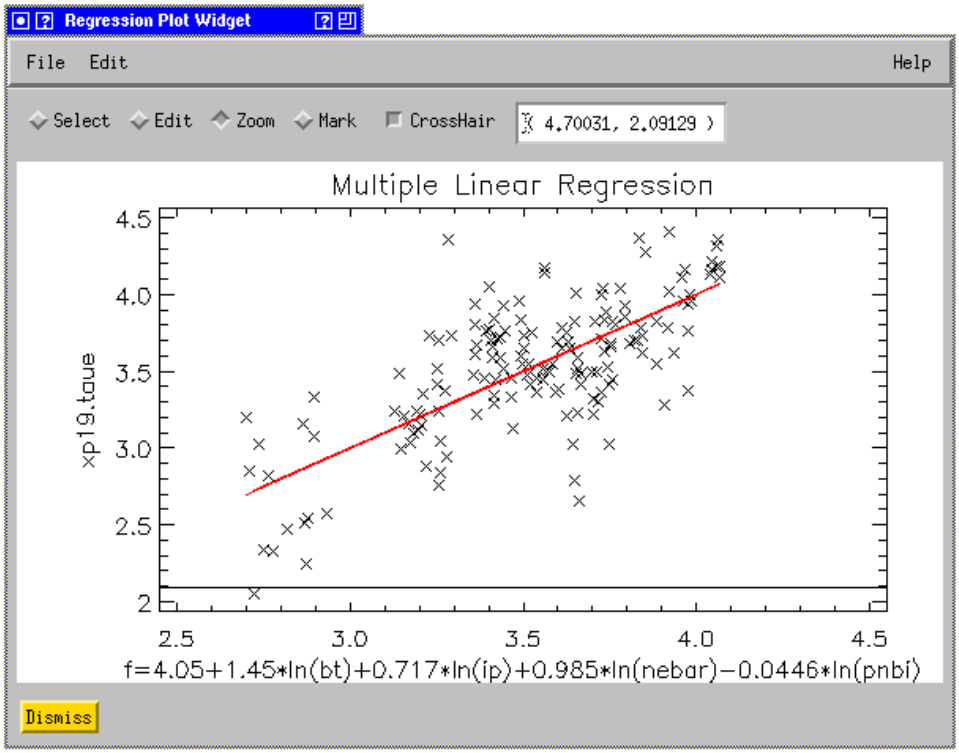


Click on the “Analyze” button and you should see:

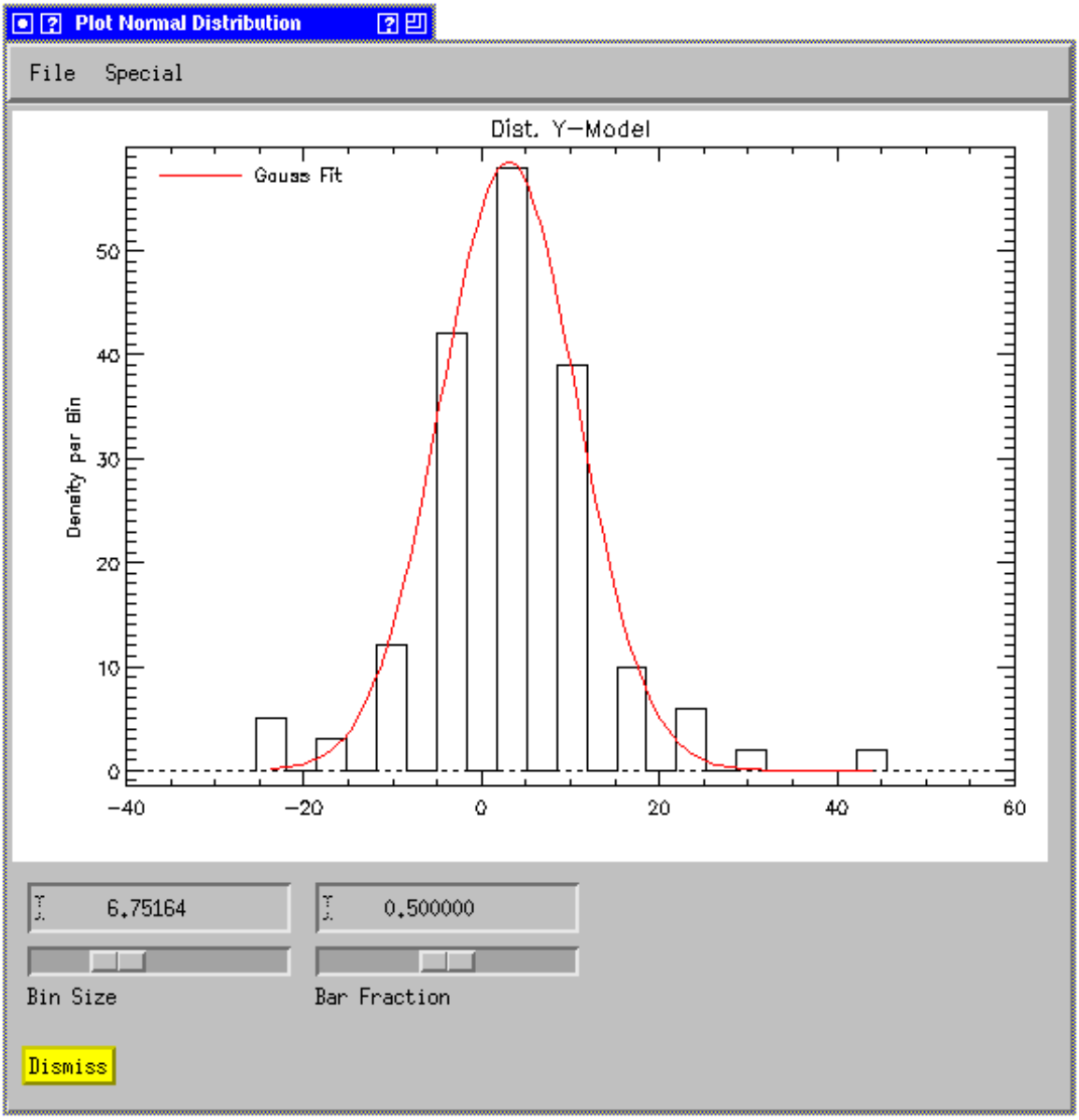


After choosing a Y value (e.g., xp19.taue) and Effects, (e.g., drag the mouse across the remaining variables, and click on “> Add >”), check the “Use Powers” button and click on “Run Model.” Two windows should appear:





After you have run a model, the button “Dist. of Y-Model” button will become active. Clicking it will show you the distribution of points about the mean:

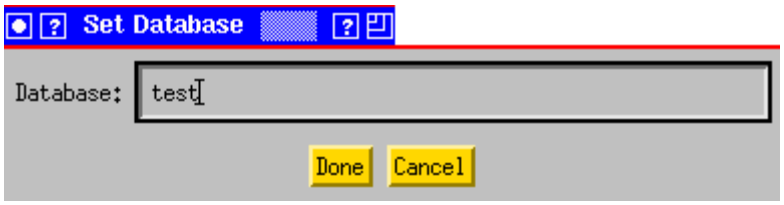


You can create this plot for individual parameters by clicking on a parameter in the "DbAccess Model" window, and clicking the "<-Plot Distribution" button.

XI. Working with a Different Database

There may be instances when you would like to work with the contents of another database. For example if you would like to test table creation you may wish to use the "test" database or you may wish to browse EFIT runs from the "code rundb" database. To begin working with another database:

1) Choose "Set Database" from the "Edit" menu on the main program window.



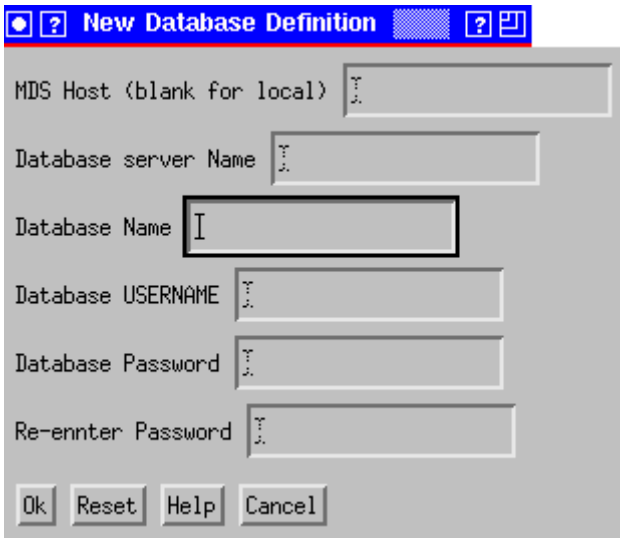
2) Enter the name of the database and click “Done.”

3) Click the “Refresh” Button on the main program window. You should see the table names on the main window refresh.

   As you can see you are required to know the name of the database you would like to use. In addition, you will need a <database name>.sybase_login file in your home directory for any database you will use of the form:

```
eagle:8080
eagle
<database name>
<unix username>
pfcworld
```

   If you do not create this file before attempting to change databases you will be prompted with a “New Database Definition" window:



   where MDS Host is eagle:8080
   Database server Name is eagle
   Database name <database name>
   Username <unix username>
   Database Password <database password>

   The <database>.sybase_login file will then be created for you. The next time you change to using this database you should not see this window.

If you have any problems or questions regarding the use of DbAccess or would like to see any additional functionality added, please send a message to dbadmin@pppl.gov

Appendix I. SQL Functions and Arithmetic Operators

ABS Returns the absolute, positive value of the given numeric expression. **Syntax** ABS **(** numeric_expression **)**	DEGREES Given an angle in radians, returns the corresponding angle in degrees. **Syntax** DEGREES **(** numeric_expression **)**	RAND Returns a random **float** value from 0 through 1. **Syntax** RAND **(** [*seed*] **)**
ACOS Returns the angle, in radians, whose cosine is the given **float** expression; also called arccosine. **Syntax** ACOS **(** *float_expression* **)**	EXP Returns the exponential value of the given **float** expression. **Syntax** EXP **(** *float_expression* **)**	ROUND Returns a numeric expression, rounded to the specified length or precision. **Syntax** ROUND **(** *numeric_expression* **,** *length* [**,** *function*] **)**
ASIN Returns the angle, in radians, whose sine is the given **float** expression (also called arcsine). **Syntax** ASIN **(** *float_expression* **)**	FLOOR Returns the largest integer less than or equal to the given numeric expression. **Syntax** FLOOR **(** *numeric_expression* **)**	SIGN Returns the positive (+1), zero (0), or negative (-1) sign of the given expression. **Syntax** SIGN **(** *numeric_expression* **)**

ATAN Returns the angle in radians whose tangent is the given **float** expression (also called arctangent). **Syntax** ATAN **(** float_expression **)**	LOG Returns the natural logarithm of the given **float** expression. **Syntax** LOG **(** *float_expression* **)**	SIN Returns the trigonometric sine of the given angle (in radians) in an approximate numeric (**float**) expression. **Syntax** SIN **(** *float_expression* **)**
ATN2 Returns the angle, in radians, whose tangent is between the two given **float** expressions (also called arctangent). **Syntax** ATN2 **(** *float_expression* **,** *float_expression* **)**	LOG10 Returns the base-10 logarithm of the given **float** expression. **Syntax** LOG10 **(** *float_expression* **)**	SQUARE Returns the square of the given expression. **Syntax** SQUARE **(** *float_expression* **)**
CEILING Returns the smallest integer greater than, or equal to, the given numeric expression. **Syntax** CEILING **(** *numeric_expression* **)**	PI Returns the constant value of PI. **Syntax** PI **( )**	SQRT Returns the square root of the given expression. **Syntax** SQRT **(** *float_expression* **)**
COS A mathematic function that returns the trigonometric cosine of the given angle (in radians) in the given expression. **Syntax** COS **(** *float_expression* **)**	POWER Returns the value of the given expression to the specified power. **Syntax** POWER **(** *numeric_expression* **,** *y* **)**	TAN Returns the tangent of the input expression. **Syntax** TAN **(** *float_expression* **)**
COT A mathematic function that returns the trigonometric cotangent of the specified angle (in radians) in the given **float** expression. **Syntax** COT **(** *float_expression* **)**	RADIANS Returns radians when a numeric expression, in degrees, is entered. **Syntax** RADIANS **(** *numeric_expression* **)**	

Arithmetic operators perform mathematical operations on two expressions of any of the data types of the numeric data type category.

Operator	Meaning
+ (Add)	Addition.
- (Subtract)	Subtraction.
* (Multiply)	Multiplication.
/ (Divide)	Division.
% (Modulo)	Returns the integer remainder of a division.

Appendix II. Pattern Matching Syntax

LIKE determines whether or not a given character string matches a specified pattern. A pattern can include regular characters and wildcard characters. During pattern matching, regular characters must exactly match the characters specified in the character string. Wildcard characters, however, can be matched with arbitrary fragments of the character string. Using wildcard characters makes the LIKE operator more flexible than using the = and != string comparison operators. If any of the arguments are not of character string data type, Microsoft® SQL Server™ converts them to character string data type, if possible.

Syntax

match_expression [NOT] LIKE *pattern* [ESCAPE *escape_character*]

Arguments

match_expression

Is any valid SQL Server expression of character string data type.

pattern

Is the pattern to search for in *match_expression*, and can include these valid SQL Server wildcard characters.

Wildcard character	Description	Example
%	Any string of zero or more characters.	WHERE title LIKE '%computer%' finds all book titles with the word 'computer' anywhere in the book title.
_ (underscore)	Any single character.	WHERE au_fname LIKE '_ean' finds all four-letter first names that end with ean (Dean, Sean, and so on).
[ ]	Any single character within the specified range ([a-f]) or set ([abcdef]).	WHERE au_lname LIKE '[C-P]arsen' finds author last names ending with arsen and beginning with any single character between C and P, for example Carsen, Larsen, Karsen, and so on.
[^]	Any single character not within the	WHERE au_lname LIKE 'de[^l]%' all author

	specified range ([^a-f]) or set ([^abcdef]).	last names beginning with de and where the following letter is not l.

escape_character

Is any valid SQL Server expression of any of the data types of the character string data type category. *escape_character* has no default and must consist of only one character.

Result Types

Boolean

Result Value

LIKE returns TRUE if the *match_expression* matches the specified *pattern*.

Remarks

When you perform string comparisons with LIKE, all characters in the pattern string are significant, including leading or trailing spaces. If a comparison in a query is to return all rows with a string LIKE 'abc ' (abc followed by a single space), a row in which the value of that column is abc (abc without a space) is not returned. However, trailing blanks, in the expression to which the pattern is matched, are ignored. If a comparison in a query is to return all rows with the string LIKE 'abc' (abc without a space), all rows that start with abc and have zero or more trailing blanks are returned.

Pattern Matching with LIKE

It is recommended that LIKE be used when you search for **datetime** values, because **datetime** entries can contain a variety of dateparts. For example, if you insert the value 19981231 9:20 into a column named **arrival_time**, the clause WHERE *arrival_time* = 9:20 cannot find an exact match for the 9:20 string because SQL Server converts it to Jan 1, 1900 9:20AM. A match is found, however, by the clause WHERE *arrival_time* LIKE '%9:20%'.

Using the % Wildcard Character

If the LIKE '5%' symbol is specified, SQL Server searches for the number 5 followed by any string of zero or more characters.

To see all objects that do not begin with 5, use NOT LIKE '5%'.

Examples

D. Use the [ ] wildcard characters

Examples Using Like:

```
select shot,phase from haccess where shot < 107320
```

returns:

```
107300  DI

107302  LH

107303  LH
```

```
107305  LL

107306  LH

107307  LH

107308  DI

107311  LH

107312  DI

107313  LH

107314  LH

107315  HS

107316  HS

107317  HS
```

To select records where 'phase' begins with 'L':

```
select shot,phase from haccess where shot < 107320 and phase like 'L%'

107302 LH

107303 LH

107305 LL

107306 LH

107307 LH

107311 LH

107313 LH

107314 LH
```

To select records where 'phase' does not contain an 'H':

```
select shot,phase from haccess where shot < 107320 and phase not like '%h%'
```

```
107300 DI

107305 LL

107308 DI

107312 DI
```

To select records where 'phase' begins with 'L' or 'H':

```
select shot,phase from haccess where shot < 107320 and phase like '[hl]%'
```

```
107302 LH

107303 LH

107305 LL

107306 LH

107307 LH

107311 LH

107313 LH

107314 LH

107315 HS

107316 HS

107317 HS
```