BNL-102146-2014-TECH

RHIC/AP/35;BNL-102146-2013-IR

User Commands Sun Release 4.1

W. MacKay

August 1994

Collider Accelerator Department

Brookhaven National Laboratory

U.S. Department of Energy

USDOE Office of Science (SC)

DISCLAIMER

RHIC/AP/35
8/194
W. MacKay

User Commands

Sun Release 4.1

August 1, 1994

dbtools

Several tools are available for manipulating Sybase tables and for copying tables between a database and SDS files.

Utilities from Sybase which are useful are:

bcp -- a bulk copy utility for copying data between Sybase tables and disk files.

dwb -- a GUI interface for scaning, editing, and messing around with databases.

isql -- a command line interface which uses the Transact SQL language. This can be used with script files. The editor emacs has an SQL mode (sql.el) which can be used with isql.

defncopy
-- a utility for copying out or in definitions of rules, defaults, views, procedures, and triggers.

More about the above programs may be found in the Sybase documentation or by using the syman utility.

-- -- -- -- -- -- -- -- -- --

Other programs and scripts:

db2sds -- a program to copy data out of the database into an SDS file. This program accepts a limited version of the SQL "select" query.

sds2db -- a program to copy data from an SDS object into a database table.

db2gh -- a program to create *.h files for groups of tables and views defined in the "header_info" tables: "header_groups" and "header_tables". It also generates two functions for each group, *_in_sds() and *_out_sds() where * refers to the group name.

dbg2sds -- a program to copy data from a group of tables and views into an SDS file with one object per table or view.

sds2dbg -- a shell script which copies SDS objects into Sybase tables for tables in a group using *sds2db*. It does not try to copy views back into the database; if this is required (and possible), individual views may be copied using *sds2db*.

gselp -- a shell script which grants the select privilege to public for all user defined tables and views within a database.

add_header_info
-- a shell script for adding the "header_info" tables, "header_groups" and "header_tables" to a database.

bcpout -- a shell script to bulk copy all user defined tables in a database into ASCII files.

SEE ALSO

db2sds, sds2db, db2gh, dbg2sds, sds2dbg, gselp, add_header_info, bcpout, dblogins, dbgroups

.dblogins file

Many of the database tools and scripts read the ".dblogins" file from the $HOME/ dirctory in order to find Sybase login information (especially passwords). In order to use some of the programs and many of the scripts you must have a ".dblogins" file in your $HOME directory. This is a file which supercedes the old ".dblogin" file for most of the scripts.

Since the ".dblogins" file contains passwords, read/write protection should be restricted to the user (-rw-------). Each line of the file should have 4 fields separated by white space as follows:

```
  <dbname>  <user>  <pwd>  <server>
```

```
where  <dbname> is the name of a database,
       <user>   is the Sybase username to log into <dbname>,
       <pwd>    is the password for user <user>, and
       <server> is the name of the Sybase server containing
                the database.
```

If the name of a database does not occur in the file, then information on the first line will usually be used as the default information.

By having different login parameters for databases, one person may easily log into several databases as different users. This allows having databases owned by virtual users (e.g. "atr_gddb" is owned by "atr_dude"), so that multiple people (Todd and Waldo) may login as "atr_dude" to create and drop tables, without having to give such access to their private databases to each other.

An example of the file might be:

```
babcom5 garibaldi peekaboo RHICSYB
waldo mackay peekaboo RHICSYB
atr_gddb atr_dude Iseeyou RHICSYB
ags_to_rhic mackay secretsRus SYBASE491
```

Here garibaldi's default database would be babcom5 on RHICSYB with the password "peekaboo".

Aliases can be made for the Sybase utilities (dwb, isql, ...) using awk to read the ".dblogins" file. The following alias may be inserted into your .cshrc file to read the ".dblogins" file and log into the correct account with the "dwb" command, although neither "dwb" or "isql" allow you to specify the database on the command line.

```
alias dwb dwb \`awk "'"'/^\!$/{print " -U " $2 " -P " $3'\
 ' " -S " $4}'"'" $HOME/.dblogins\`
```

SEE ALSO

dbtools, dbgroups, add_header_info, db2gh, dbg2sds, sds2dbg, db2sds, sds2db, bcpout, gselp

NAME

bcpout -- does a bulk copy of all user defined tables into ASCII files.

SYNOPSIS

bcpout *dbname*

DESCRIPTION

This shell script uses **isql** to query the database table "sysobjects" in *dbname* for user defined tables and then executes the **bcp** command to bulk copy them into ASCII files in the current directory. Each file is named *dbname..tablename* where *tablename* is the name of the table in *dbname*. A comma is used for a column separator.

The user must have select privilege on each of the tables to be copied, as well as on the "sysobjects" table. It requires that the login information for the user is defined in **$HOME/.dblogins**

SEE ALSO

dbtools, Sybase documentation for bcp

NAME

gselp -- grants select privilege to all user defined tables and views in a Sybase database.

SYNOPSIS

gselp *dbname*

DESCRIPTION

This shell script uses "isql" to query the database table "sysobjects" in *dbname* for database objects with type= "U" or "V" and then grants the select privilege to public on each of these objects. It requires that the user have select access to the "sysobjects" table and that he have grant select privilege on the tables. This is typically only useful for the database owner. It also requires that the login information for the database is defined in **$HOME/.dblogins**

SEE ALSO

dblogins, dbtools

Definition of groups of tables and views in a Sybase database.

Groups of tables within Sybase may be formed with the help two user defined tables: **header_groups** and **header_tables.**

Both these tables may be created by the database owner with the following command:

```
add_header_info <dbname>
```

where <dbname> corresponds to the name of the database.

The table **header_groups** lists the defined groups by name, with extra columns for the name of a C header (*.h) file for the group and a comment containing a description of the group. The comment is written into the header file. The SQL create command definition of the **header_groups** table is given by:

```
create table header_groups
(
  groupname char(30) primary key,
  filename varchar(40) not null,
  comment varchar(255) default ""
)
```

The "filename" column should contain the valid name for the header file (e.g. "header_info.h"). The program **dbg2sds** will copy a group of tables and views into an SDS file, with an SDS object corresponding to each table or view in the group. The program **db2gh** will generate the C header files for each defined group, as well as input and output functions for for reading and writing the SDS files. The script **sds2dbg** will copy tables from a group in an SDS file into the database; views will not be copied back in, since there may not be enough information in a view to specify how to fill the tables.

The tables and views contained in a given group are specified in the **header_tables** table, which is constructed with the following create command in SQL:

```
create table header_tables
(
  groupname char(30) references header_groups(groupname),
  sequence int not null,
  tablename char(30) not null,
  dbname char(30) default "'$1'",
  host char(30) default "'${dbserv}'",
  prefix varchar(10) default "",
  unique clustered (groupname, sequence)
)
```

where '$1' is the current database name, and
    '${dbserv}' is the current server name.

The column "groupname" specifies a row entry for entering a table or view in a given group. The name of the table or view should be entered into the "tablename" column. For each group, the tables must be ordered by the "sequence" number entry, given in an order which the tables must be filled with respect to referential integrity constraints. The combination of (groupname, sequence) must form a unique entry for each row. The "dbname" entry should be the name of the database containing the table; at present tables should reside within the same database as the group; however, this may change in the future. The "host" entry should specify which Sybase host should be accessed to read the table (e.g., RHICSYB or SYBASE491); at present, this should be the same as the database of the group (i.e., the current database). The "prefix" entry defaults to a null string, but may be filled with a string which will be prefixed onto the structure definitions in the C header file in order to resolve clashes with similarly named tables in other databases. If two such tables tables with identical names in different databases have the identical structure, then it is not necessary to use the "prefix" entry, since the structure definitions will be identical.

EXAMPLE

Executing the script

```
add_header_info waldo
```

also builds the group "header_info" in the database "waldo" and defines the table entries for the "header_info" group:

```
header_groups:
  groupname, filename, comment:
  "header_info", "Dbapps/header_info.h", "comment blah"

header_tables:
  groupname, sequence, tablename, dbname, host, prefix:
  "header_info", 1, "header_tables", "waldo", "RHICSYB", ""
  "header_info", 2, "header_groups", "waldo", "RHICSYB", ""
```

If the command

```
db2gh waldo header_info
```

is then given in a directory with the subdirectory "Dbapps/", a header file will be written with the file specification "Dbapps/header_info.h"; if the subdirectory is not there, then a bunch of error messages will appear, and the header file will not be written. Two other files besides the _.h file will be created:
"header_info_in_sds.c" containing the function

```
"header_info_in_sds(sds_handle,
                         struct header_info_ptrs *)"
```

and
"header_info_out_sds.c" containing the function

```
"int header_info_out_sds(sds_handle, char *,
                         struct header_info_ptrs)"
```

SEE ALSO

add_header_info, db2gh, dbg2sds, sds2dbg, dbtools, dblogins

NAME

add_header_info - adds the "header_info" group definition tables to a database.

SYNOPSIS

add_header_info <dbname>
where <dbname> is the name of a database

DESCRIPTION

This shell script generates two tables in the given database for defining "groups" of tables and views. This command should be run by the database owner to create and initialize the tables: "header_groups" and "header_tables".

SEE ALSO

dbgroups, dblogins, dbtools, db2gh, dbg2sds, sds2dbg

DEFICIENCIES

NAME

db2gh - generate C header files (*.h) for "groups" of tables within a SYBASE database.

SYNOPSIS

```
db2gh <dbname> [<groupname> | -]
where <dbname> is the name of a database
      <groupname> is the name of a group defined in the
                  table "header_groups"
```

DESCRIPTION

This program generates C header files for groups of SYBASE tables and views as defined in the tables "header_groups" and "header_tables" of a database. If these tables do not exist, then db2gh will not work correctly. *db2gh* additionally creates two files for each group: *_in_sds.c and *_out_sds.c, where * is the group name. These files are useful for reading and writting an SDS file with the table data from the groups. The commands *dbg2sds* and *sds2dbg* may be used for copying groups between the database and an SDS file. (Note that *sds2dbg* only copies tables from the SDS file into the database, NOT views.)

If no groupname is given, then header files and functions are written for all groups within the database. An additional header file is written which contains #include statements for all the other header files; the name of this file is

   <dbname>_hi.h

where <dbname> is again the database name. (Note that there should be subdirectories in the current directory for any groups which have specified directories in the "filename" column of the "header_groups" table, e.g. the "header_info" group requires the subdirectory "Dbapps".)

If a minus (-) sign is given as the groupname, then a special header file is written containing structures for all tables not yet defined in groups. This feature is intended for helping the database designer develop groups.

DATA Conversions

The following table gives the mapping of types for each piece of data in the Sybase database, SDS file, and C-program structures:

Sybase type	SDS type	C struct type
char(#)	SDS_STRING	char ...[#+1]
varchar(*)	SDS_STRING	char ...[#+1]
binary(#)	SDS_BYTE	char ...[#]
varbinary(#)	SDS_BYTE	char ...[#]
tinyint	SDS_BYTE	char
smallint	SDS_WORD	short
int	SDS_LONG	long
real	SDS_REAL	float
double precision	SDS_DOUBLE	double
float	SDS_DOUBLE	double
datetime	SDS_TIME	long ...[2]
smalldatetime	SDS_UNIX_TIME	long

The character string types "char(#)" and "varchar(#)" define strings which are not null terminated within Sybase. In order to keep C-programs from having segmentation faults when a user tries to print one of these strings from the SDS file with the usual assumption that strings are null terminated, we have added an extra character to the length of the char* in the corresponding C-structure. This extra character is always null. Additionally, any excess whitespace at the end of the string is truncated; this should simplify tests such as used in functions like strcmp(). When copying the string back into the database, any excess string length will be truncated at the limit of the column's length.

The time formats of the database and SDS files are different. The SDS_TIME format is assumed to be like the usual "struct timeval" definition of two long integers with the first element containing the number of seconds since 1 Jan 1970, and the second long containing the fraction of a second in microseconds, i.e., Unix time given in Universal Coordinated Time (UTC). The SDS_UNIX_TIME a single long integer which has the same definition as the first part of the SDS_TIME. (See the man pages for the C-function "localtime" for useful time functions.)

The Sybase "datetime" is two long integers: the first being the number of days since 1 Jan 1900, and the second being the number of 1/300's of a second from midnight. The time is assumed to be in the local time zone of the database server. This means that any true conversion will be screwed up at least twice a year, if the server resides in a part of the world where daylight savings time is used during part of the year.

The Sybase "smalldatetime" is stored as two short integers: the first being the number of days since 1 Jan 1900 in local time, and the second being the number of minutes since midnight.

DEFICIENCIES

Support for the following Sybase types does not (yet?) exist

    bit
    text
    image
    numeric(p,s)
    decimal(p,s)
    smallmoney
    money

Some of these will be addressed in the future, but they are not high priority items.

I'm not sure what happens with the types:

    nchar
    nvarchar

Unless somebody loads a different character set, these should be useless anyway.

Just as a note of warning: the "float" datatype may be 4 or 8 bytes, depending on the server. The datatype "real" is 4 bytes long, and "double precision" is 8 bytes long. Both the CCD (SYBASE491) and RHIC (RHICSYB) servers seem to treat "float" as 8 bytes.

Currently, the program *sds2db* copies SDS objects into database tables via the SQL "insert" command. There are two drawbacks to this: transfers are rather slow, and transactions are logged causing the transaction log to fill up rather quickly. In the future, we may try to either convert **sds2db** or make a parallel program which will use bulk copying.

Conversions of dates and times may be slightly off by an hour for a few hours around the time of transition between standard and daylight savings times.

Precision of the Sybase "datetime" is limited to 1/300's of a second.

SEE ALSO

dbgroups, dblogins, dbtools, add_header_info, dbg2sds, sds2dbg
.TH dbg2sds 1 "27 Jul 1994"

NAME

dbg2sds - copies a group of tables and views from a database to an SDS file. database.

SYNOPSIS

```
dbg2sds <dbname> <groupname>

where
        <dbname> is the name of the database
        <groupname> is the name of the group to be copied into
       an SDS file.
```

DESCRIPTION

This program copies the group <groupname> of tables and views from a Sybase database <dbname> to an SDS file. Each table or view becomes an SDS object within the SDS file. The object name is identical to the corresponding table or view name. It searches the "$HOME/.dblogins" file for login information to the database.

Groups are defined by the two tables: "header_groups" and "header_tables", which can be created by the **add_header_info** command.

DATA Conversions

The following table gives the mapping of types for each piece of data in the Sybase database, SDS file, and C-program structures:

Sybase type	SDS type	C struct type
char(#)	SDS_STRING	char ...[#+1]
varchar(*)	SDS_STRING	char ...[#+1]
binary(#)	SDS_BYTE	char ...[#]
varbinary(#)	SDS_BYTE	char ...[#]
tinyint	SDS_BYTE	char
smallint	SDS_WORD	short
int	SDS_LONG	long
real	SDS_REAL	float
double precision	SDS_DOUBLE	double
float	SDS_DOUBLE	double
datetime	SDS_TIME	long ...[2]
smalldatetime	SDS_UNIX_TIME	long

The character string types "char(#)" and "varchar(#)" define strings which are not null terminated within Sybase. In order to keep C-programs from having segmentation faults when a user tries to print one of these strings from the SDS file with the usual assumption that strings are null terminated, we have added an extra character to the length of the char* in the corresponding C-structure. This extra character is always null. Additionally, any excess whitespace at the end of the string is truncated; this should simplify tests such as used in functions like strcmp(). When copying the string back into the database, any excess string length will be truncated at the limit of the column's length.

The time formats of the database and SDS files are different. The SDS_TIME format is assumed to be like the usual "struct timeval" definition of two long integers with the first element containing the number of seconds since 1 Jan 1970, and the second long containing the fraction of a second in microseconds, i.e., Unix time given in Universal Coordinated Time (UTC). The SDS_UNIX_TIME a single long integer which has the same definition as the first part of the SDS_TIME. (See the man pages for the C-function "localtime" for useful time functions.)

The Sybase "datetime" is two long integers: the first being the number of days since 1 Jan 1900, and the second being the number of 1/300's of a second from midnight. The time is assumed to be in the local time zone of the database server. This means that any true conversion will be screwed up at least twice a year, if the server resides in a part of the world where daylight savings time is used during part of the year.

The Sybase "smalldatetime" is stored as two short integers: the first being the number of days since 1 Jan 1900 in local time, and the second being the number of minutes since midnight.

DEFICIENCIES

Support for the following Sybase types does not (yet?) exist

```
bit
text
image
numeric(p,s)
decimal(p,s)
smallmoney
money
```

Some of these will be addressed in the future, but they are not high priority items.

I'm not sure what happens with the types:

```
nchar
nvarchar
```

Unless somebody loads a different character set, these should be useless anyway.

Just as a note of warning: the "float" datatype may be 4 or 8 bytes, depending on the server. The datatype "real" is 4 bytes long, and "double precision" is 8 bytes long. Both the CCD (SYBASE491) and RHIC (RHICSYB) servers seem to treat "float" as 8 bytes.

Currently, the program *sds2db* copies SDS objects into database tables via the SQL "insert" command. There are two drawbacks to this: transfers are rather slow, and transactions are logged causing the transaction log to fill up rather quickly. In the future, we may try to either convert **sds2db** or make a parallel program which will use bulk copying.

Conversions of dates and times may be slightly off by an hour for a few hours around the time of transition between standard and daylight savings times.

Precision of the Sybase "datetime" is limited to 1/300's of a second.

SEE ALSO

dbgroups, dblogins, dbtools, add_header_info, db2gh, sds2dbg
 .TH sds2dbg 1 "26 Jul 1994"

NAME

sds2dbg - copies tables from a group SDS file back into the database.

SYNOPSIS

```
sds2dbg <dbname> <groupname> <filename>
where <dbname> is the name of a database
    <groupname> is the name of a group
    <filename> is the name of the SDS file containing the group.
```

DESCRIPTION

This shell script queries the database <dbname> for all tables in the given group, <groupname>. It ignores views, since views usually don't have enough information to reverse the process of a select statement. Each table from the SDS file <filename> is copied back into the database with the program *sds2db*.

Each table will be truncated before the copying is done. It has been assumed that the table has already been created; if not, then the table may be created via the *sds2db* command.

Each table to be copied via *sds2dbg* should have an SDS object whose name in the SDS file is identical to the name of the database table.

DATA Conversions

The following table gives the mapping of types for each piece of data in the Sybase database, SDS file, and C-program structures:

Sybase type	SDS type	C struct type
char(#)	SDS_STRING	char ...[#+1]
varchar(*)	SDS_STRING	char ...[#+1]
binary(#)	SDS_BYTE	char ...[#]
varbinary(#)	SDS_BYTE	char ...[#]
tinyint	SDS_BYTE	char
smallint	SDS_WORD	short
int	SDS_LONG	long
real	SDS_REAL	float
double precision	SDS_DOUBLE	double
float	SDS_DOUBLE	double
datetime	SDS_TIME	long ...[2]
smalldatetime	SDS_UNIX_TIME	long

The character string types "char(#)" and "varchar(#)" define strings which are not null terminated within Sybase. In order to keep C-programs from having segmentation faults when a user tries to print one of these strings from the SDS file with the usual assumption that strings are null terminated, we have added an extra character to the length of the char* in the corresponding C-structure. This extra character is always null. Additionally, any excess whitespace at the end of the string is truncated; this should simplify tests such as used in functions like strcmp(). When copying the string back into the database, any excess string length will be truncated at the limit of the column's length.

The time formats of the database and SDS files are different. The SDS_TIME format is assumed to be like the usual "struct timeval" definition of two long integers with the first element containing the number of seconds since 1 Jan 1970, and the second long containing the fraction of a second in microseconds, i.e., Unix time given in Universal Coordinated Time (UTC). The SDS_UNIX_TIME a single long integer which has the same definition as the first part of the SDS_TIME. (See the man pages for the C-function "localtime" for useful time functions.)

The Sybase "datetime" is two long integers: the first being the number of days since 1 Jan 1900, and the second being the number of 1/300's of a second from midnight. The time is assumed to be in the local time zone of the database server. This means that any true conversion will be screwed up at least twice a year, if the server resides in a part of the world where daylight savings time is used during part of the year.

The Sybase "smalldatetime" is stored as two short integers: the first being the number of days since 1 Jan 1900 in local time, and the second being the number of minutes since midnight.

DEFICIENCIES

Support for the following Sybase types does not (yet?) exist

bit
text
image
numeric(p,s)
decimal(p,s)
smallmoney
money

Some of these will be addressed in the future, but they are not high priority items.

I'm not sure what happens with the types:

nchar
nvarchar

Unless somebody loads a different character set, these should be useless anyway.

Just as a note of warning: the "float" datatype may be 4 or 8 bytes, depending on the server. The datatype "real" is 4 bytes long, and "double precision" is 8 bytes long. Both the CCD (SYBASE491) and RHIC (RHICSYB) servers seem to treat "float" as 8 bytes.

Currently, the program *sds2db* copies SDS objects into database tables via the SQL "insert" command. There are two drawbacks to this: transfers are rather slow, and transactions are logged causing the transaction log to fill up rather quickly. In the future, we may try to either convert **sds2db** or make a parallel program which will use bulk copying.

Conversions of dates and times may be slightly off by an hour for a few hours around the time of transition between standard and daylight savings times.

Precision of the Sybase "datetime" is limited to 1/300's of a second.

SEE ALSO

dbgroups, dblogins, dbtools, add_header_info, db2gh, dbg2sds, sds2db
.TH db2sds 1 "26 July 1994"

NAME

db2sds - creates an SDS file from a database via an SQL select statement.

SYNOPSIS

```
db2sds [-i] [-h <host>] [-u <user>] [-p <pwd>]          <dbname> [<sql_filename>]
```

where

i: generate insert sds template (header file)
u: log in as <user>
p: log in with passowrd <pwd>
h: log in to dataserver <host>

DESCRIPTION

This copies data from the database <dbname> using an SQL statement, which is either typed in interactively, or specified in the file <sql_filename>.

DATA Conversions

The following table gives the mapping of types for each piece of data in the Sybase database, SDS file, and C-program structures:

Sybase type	SDS type	C struct type
char(#)	SDS_STRING	char ...[#+1]

varchar(*)	SDS_STRING	char ...[#+1]
binary(#)	SDS_BYTE	char ...[#]
varbinary(#)	SDS_BYTE	char ...[#]
tinyint	SDS_BYTE	char
smallint	SDS_WORD	short
int	SDS_LONG	long
real	SDS_REAL	float
double precision	SDS_DOUBLE	double
float	SDS_DOUBLE	double
datetime	SDS_TIME	long ...[2]
smalldatetime	SDS_UNIX_TIME	long

The character string types "char(#)" and "varchar(#)" define strings which are not null terminated within Sybase. In order to keep C-programs from having segmentation faults when a user tries to print one of these strings from the SDS file with the usual assumption that strings are null terminated, we have added an extra character to the length of the char* in the corresponding C-structure. This extra character is always null. Additionally, any excess whitespace at the end of the string is truncated; this should simplify tests such as used in functions like strcmp(). When copying the string back into the database, any excess string length will be truncated at the limit of the column's length.

The time formats of the database and SDS files are different. The SDS_TIME format is assumed to be like the usual "struct timeval" definition of two long integers with the first element containing the number of seconds since 1 Jan 1970, and the second long containing the fraction of a second in microseconds, i.e., Unix time given in Universal Coordinated Time (UTC). The SDS_UNIX_TIME a single long integer which has the same definition as the first part of the SDS_TIME. (See the man pages for the C-function "localtime" for useful time functions.)

The Sybase "datetime" is two long integers: the first being the number of days since 1 Jan 1900, and the second being the number of 1/300's of a second from midnight. The time is assumed to be in the local time zone of the database server. This means that any true conversion will be screwed up at least twice a year, if the server resides in a part of the world where daylight savings time is used during part of the year.

The Sybase "smalldatetime" is stored as two short integers: the first being the number of days since 1 Jan 1900 in local time, and the second being the number of minutes since midnight.

DEFICIENCIES

Support for the following Sybase types does not (yet?) exist

bit
text
image
numeric(p,s)
decimal(p,s)
smallmoney
money

Some of these will be addressed in the future, but they are not high priority items.

I'm not sure what happens with the types:

nchar
nvarchar

Unless somebody loads a different character set, these should be useless anyway.

Just as a note of warning: the "float" datatype may be 4 or 8 bytes, depending on the server. The datatype "real" is 4 bytes long, and "double precision" is 8 bytes long. Both the CCD (SYBASE491) and RHIC (RHICSYB) servers seem to treat "float" as 8 bytes.

Currently, the program *sds2db* copies SDS objects into database tables via the SQL "insert" command. There are two drawbacks to this: transfers are rather slow, and transactions are logged causing the transaction log to fill up rather quickly. In the future, we may try to either convert **sds2db** or make a parallel program which will use bulk copying.

Conversions of dates and times may be slightly off by an hour for a few hours around the time of transition between standard and daylight savings times.

Precision of the Sybase "datetime" is limited to 1/300's of a second.

SEE ALSO

sds2dbg, dbgroups, dbg2sds, sds2db, dbtools, dblogins, dbgroups

NAME

sds2db - copies an SDS object into a Sybase database table.

SYNOPSIS

```
sds2db [-c][-t][-o <object>] [-h <hostname>]        [-u <user> -p <password>] \
        <input_file> <database_name> <tablename>

where: -c will create the table
     -h sets a SYBASE dbms called <hostname>
     -f sds contains flat arrays,not structure
     -s will take input sds from shared memory
     -t will truncate the table before copying
     -o creates the table from SDS object <object>
        rather than the default object, named the same as
          the table
     -u logs you in as user <user>
     -p ...with password <password>
```

DESCRIPTION

This copies a single SDS object into a database table. The various options allow the creation of new tables, as well as the truncation of an existing table before the copying begins. If no login information is specified then the ".dblogins" file will be searched. To maintain compatability with earlier versions, if the ".dblogins" file does not exist, then it looks older formated file ".dblogin", first in the user's HOME directory.

DATA Conversions

The following table gives the mapping of types for each piece of data in the Sybase database, SDS file, and C-program structures:

Sybase type	SDS type	C struct type
char(#)	SDS_STRING	char ...[#+1]
varchar(*)	SDS_STRING	char ...[#+1]
binary(#)	SDS_BYTE	char ...[#]
varbinary(#)	SDS_BYTE	char ...[#]
tinyint	SDS_BYTE	char
smallint	SDS_WORD	short
int	SDS_LONG	long
real	SDS_REAL	float
double precision	SDS_DOUBLE	double
float	SDS_DOUBLE	double
datetime	SDS_TIME	long ...[2]
smalldatetime	SDS_UNIX_TIME	long

The character string types "char(#)" and "varchar(#)" define strings which are not null terminated within Sybase. In order to keep C-programs from having segmentation faults when a user tries to print one of these strings from the SDS file with the usual assumption that strings are null terminated, we have added an extra character to the length of the char* in the corresponding C-structure. This extra character is always null. Additionally, any excess whitespace at the end of the string is truncated; this should simplify tests such as used in functions like strcmp(). When copying the string back into the database, any excess string length will be truncated at the limit of the column's length.

The time formats of the database and SDS files are different. The SDS_TIME format is assumed to be like the usual "struct timeval" definition of two long integers with the first element containing the number of seconds since 1 Jan 1970, and the second long containing the fraction of a second in microseconds, i.e., Unix time given in Universal Coordinated Time (UTC). The SDS_UNIX_TIME a single long integer which has the same definition as the first part of the SDS_TIME. (See the man pages for the C-function "localtime" for useful time functions.)

The Sybase "datetime" is two long integers: the first being the number of days since 1 Jan 1900, and the second being the number of 1/300's of a second from midnight. The time is assumed to be in the local time zone of the database server. This means that any true conversion will be screwed up at least twice a year, if the server resides in a part of the world where daylight savings time is used during part of the year.

The Sybase "smalldatetime" is stored as two short integers: the first being the number of days since 1 Jan 1900 in local time, and the second being the number of minutes since midnight.

DEFICIENCIES

Support for the following Sybase types does not (yet?) exist

```
bit
text
image
numeric(p,s)
decimal(p,s)
smallmoney
money
```

Some of these will be addressed in the future, but they are not high priority items.

I'm not sure what happens with the types:

```
nchar
nvarchar
```

Unless somebody loads a different character set, these should be useless anyway.

Just as a note of warning: the "float" datatype may be 4 or 8 bytes, depending on the server. The datatype "real" is 4 bytes long, and "double precision" is 8 bytes long. Both the CCD (SYBASE491) and RHIC (RHICSYB) servers seem to treat "float" as 8 bytes.

Currently, the program *sds2db* copies SDS objects into database tables via the SQL "insert" command. There are two drawbacks to this: transfers are rather slow, and transactions are logged causing the transaction log to fill up rather quickly. In the future, we may try to either convert **sds2db** or make a parallel program which will use bulk copying.

Conversions of dates and times may be slightly off by an hour for a few hours around the time of transition between standard and daylight savings times.

Precision of the Sybase "datetime" is limited to 1/300's of a second.

SEE ALSO

sds2dbg, dbgroups, dbg2sds, db2sds, dbtools, dblogins, dbgroups