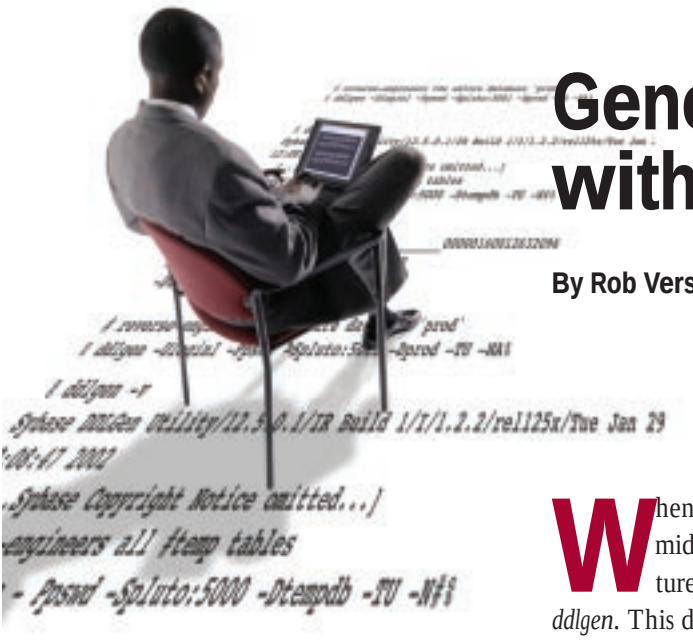

Generating DDL Statements with ddlgen

By Rob Verschoor

Taking a closer looking at ASE's new client tool

Rob Verschoor is a freelance consultant in The Netherlands, specializing in Sybase ASE. He can be reached at rob@sypron.nl.

When ASE 12.5 was released in mid-2001, one of its new features was a new client tool called *ddlgen*. This does what its name suggests: It generates T-SQL DDL statements for the objects in an existing database. In other words, ddlgen reverse-engineers the schema. Remember that "DDL" stands for "Data Definition Language" and is a collective name for statements that manipulate the schema (but not database data) such as **create table**, **create procedure**, etc.). I am sure every DBA will see the potential usefulness of such a reverse-engineering tool, so I won't dwell on it. Instead, I'll look at the technical details of ddlgen.

Where to Find ddlgen

ddlgen is a Java application which is executed from the OS command line. (Do not confuse this with the much-promoted Java-in-ASE feature: ddlgen runs outside ASE). On Unix, ddlgen is located in $SYBASE/sybcent32; on Windows, ddlgen.bat is located in %SYBASE%\Sybase Central 3.2. The JRE (Java Run-time Environment) used by ddlgen is part of the default ASE installation environment.

Although ddlgen is available to DBAs as a stand-alone tool, it is also used by other Sybase tools, including the *sybmigrate* tool (new in 12.5.0.1) and the Java version of Sybase Central.

How ddlgen Works

ddlgen can reverse-engineer individual database objects or entire databases in an ASE server. For example (% indicates the Unix command prompt; "#" indicates a comment):

```
# reverse-engineers the entire database 'prod'
% ddlgen -Ulogin1 -Ppswd -Spluto:5001 -Dprod
```

ddlgen will print all required **create** statements that are needed to recreate the current schema in database "prod" (note that ddlgen does not include any **drop** statements). By default, ddlgen prints the generated DDL statements to the standard output. This quickly becomes a lot of text: Even for a simple 1-column table, ddlgen generates 21 lines of output (partly because it includes additional comment lines for better readability of the DDL).

Please note that we cannot include the results of the examples in this article due to space limitations—readers are warmly invited to try out ddlgen for themselves!

Command Options

ddlgen accepts a number of command-line options which allow only specific objects to be reverse-engineered. The most important options are –D, to specify a database; –T, to specify the type of database object, and –N, to specify a wildcard for matching object names. For example, to generate DDL for all user tables in database "prod" whose names start with a the letter "A":

```
# reverse-engineers the entire database 'prod'
% ddlgen -Ulogin1 -Ppswd -Spluto:5001 –Dprod
–TU –NA%
```

This is the full list of object types that can be used with the –T flag:

C – cache	I – index	RS – remote server
DB – database	KC – primary/unique key constraints	SG – segment
D – default		TR – trigger
DBD – database device	L – login	U – table
DPD – dump device	P - stored procedure	UDD – user-defined datatype
EC – execution class	R – rule	USR – user
EG – engine group	RI – RI constraints	V – view
GRP – user group	RO – role	XP – extended stored procedure

As shown in the above table, server-level objects such as logins and roles can also be reverse-engineered:

```
# reverse-engineers all ASE logins
% ddlgen -Ulogin1 -Ppswd -Spluto:5001 -TL -N%
```

The command options –X and –F allow additional filtering in combination with –T.

The full set of ddlgen command options is as follows:

–U *login_name:* The login name used to connect to ASE

–P *password:* The login password

–S *hostname:port_nr:* Hostname and port number of the ASE server to connect to

–D *db_name:* The database name; if omitted, defaults to the default database of the connecting login

–T *object_type:* The object type to be reverse-engineered (see table above; the default type is DB)

–N *object_name:* Must always be used with –T to specify the object name. This may contain wildcards or may be a fully qualified name (i.e., **db_name.owner.table_name[.index]**). By default, specify –N%.

–XOU or –XOD: These options are used in combination with –TU: with –XOU, only user tables are included; with –XOD, only proxy tables. Specifying just –TU, without –X, includes both user tables and proxy tables.

-F{TR | I | KC | RI | %}: For user tables only, does not generate DDL for triggers (TR), indexes (I), primary/unique key constraints (KC), or RI constraints (RI) for those table(s). These options may be combined with commas. % excludes all of the above; –F alone has no effect.

–O *output_file:* Output file name for the generated DDL (default=*stdout*)

–E *error_file:* File to log errors into

–J *character_set:* Client character set

–v: Displays ddlgen version information

Reverse-Engineering #temp Tables

Although this capability is undocumented, ddlgen will generate DDL for #temporary tables owned by other sessions. To do this, specify the *tempdb* database and use a wildcard or specify the full table name:

```
# reverse-engineers all #temp tables
% ddlgen -Usa - Ppswd -Spluto:5000 -Dtempdb -TU -N#%

# reverse-engineers only #t_____00000160012632096
% ddlgen -Usa - Ppswd -Spluto:5000 -Dtempdb -TU -
N#t_____00000160012632096
```

Apart from writing custom queries against the tempdb system tables, there are hardly any other methods to reverse-engineer a #temporary table. This functionality may not immediately be relevant, but I remember at least one situation where it would have been useful to quickly see the schema of a #temp table. For more information about #temp tables, see an earlier article on this topic at www.sypron.nl/temptab.html.

Caveats and Gotchas

1) The first version of ddlgen was shipped as part of ASE 12.5. Unfortunately, this first version contained some disturbing bugs in the generated DDL. These bugs are fixed in later releases, and I therefore recommend everyone to upgrade to at least 12.5.0.1.

   To check your version of ddlgen, run **ddlgen –v** (**NT: ddlgen.bat –v**). The 12.5 GA version of ddlgen will print a single line saying:

```
% ddlgen -v
Product Release Version: 1.0, for ASE 12.5
```

   The 12.5.0.1 version of ddlgen prints a more complete version string as well as the standard Sybase copyright notice:

```
 % ddlgen -v
Sybase DDLGen Utility/12.5.0.1/IR Build 1/I/1.2.2/rel125x/Tue Jan 29
12:08:47 2002
 [...Sybase Copyright Notice omitted...]
```

2) The Java background of ddlgen is clearly visible: As illustrated by the examples in this article, ddlgen requires that the ASE server be specified as a host address and port number (i.e., JDBC-style). This is a bit inconvenient as ASE DBAs are used to specifying an ASE servername

instead. Let's hope this will be fixed soon. In the meantime, the quickest way to find the ASE server's port number is to run the query **select * from master..syslisteners** (when you're using TLI—on Solaris for example—you need a more complex query instead: See www.sypron.nl/quiz2002a.html#may02 for details).

3) For some reason, ddlgen does not tolerate any spaces between a command parameter flag and its value. For example, **ddlgen -U sa** (with a space between **-U** and **sa**) will result in an error: You *must* specify **-Usa** without a space.
4) When using the –T option to specify an object type, –N% must always be specified as well. Omitting –N% will cause an error message to be printed.

What ddlgen Does Not Do

Despite its usefulness, it is no surprise that ddlgen has some limitations as well:

- ddlgen cannot reverse-engineer compiled objects whose source code has been encrypted with sp_hidetext.
- For logins and roles, ddlgen does not reverse-engineer the passwords (which are stored in an encrypted form).
- ddlgen does not reverse-engineer external logins or remote logins.
- ddlgen does not always take all object dependencies into account. See the next section for more information.

Dependencies Between Objects

Ideally, the generated DDL should be able to execute without any errors. A classic problem for reverse-engineering tools are the dependencies between database objects. For example, when procedure P reads from table T, the DDL for table T should be generated before the DDL for procedure P—otherwise an error will result when the DDL for P is executed because table T does not yet exist.

ddlgen does a reasonable job of taking object dependencies into account, but it's not perfect. When your generated DDL gives you errors, the simplest remedy is to run the DDL again. The objects that could not be created the first time because other objects had yet not been created, will be created the second time (this is possible because ddlgen does not include drop statements in the generated DDL). The disadvantage of this method is that it's messy. During the second execution of the DDL, for each table, index, stored procedure, etc., there will be an error message complaining that the object already exists. This may make it more difficult to determine whether all objects have finally been created correctly.

Alternatively, you may also explicitly reverse-engineer the database object by type using the –T and –F command options (see above).

The Classic "Procedure Table" Problem

A classic problem is stored procedures referring to a table created outside the procedure. ddlgen does not solve this problem. Example:

```
create table #t1 (a int)
go
create procedure p as
select * from #t1
go
```

The **create table** statement *must* be executed first, or creating the procedure will fail because table #t1 is missing.

ddlgen will happily reverse-engineer procedure "p," but this will not include the **create table** statement. This means that when the generated DDL for procedure "p" is executed, it will result in an error because table #t1 does not exist. The background is that ddlgen retrieves the SQL text from syscomments, which does not include the **create table** statement because it is not part of the SQL text for the stored procedure. DBAs should handle such cases by maintaining their own DDL scripts containing both the **create table** and the **create procedure** statements.

Backward Compatibility

As mentioned in the introduction, ddlgen was introduced in ASE 12.5. However, ddlgen also works on earlier ASE versions, albeit with certain limitations. Please note that the information below has been determined empirically. ddlgen is formally only supported on ASE 12.5 or later.

As far as I have been able to determine, ddlgen works fine on ASE 12.0 for all types of objects.

On ASE 11.9 and 11.5, some objects can be reverse-engineered, but others can't. For example, user tables cannot be reverse-engineered in pre-12.0 servers because ddlgen expects the column *sysindexes.identitygap* to exist (so the ddlgen query fails). Other objects, such as indexes, logins, roles, or caches can be reverse-engineered successfully in 11.9 and 11.5 servers.

I have not been able to run ddlgen against ASE 11.0; connectivity problems seem to stop ddlgen from successfully connecting. ■