ArcSDE™ Configuration and Tuning Guide for Microsoft SQL Server®

ArcInfo™ 8.1.2

Contents

CHAPTER 1

Getting started

Installing ArcSDE™ and loading data into a database are arguably simple processes, especially when using tools such as ArcGIS™ software's ArcCatalog™ or Microsoft SQL Server 2000's Enterprise Manager. So, why is there a configuration and tuning guide? While installation and data loading can be relatively straightforward, relational database management systems are dynamic. Relational databases offer highly efficient storage, searching, and modeling mechanisms that can serve many users simultaneously. It takes some effort to build and maintain a database that performs optimally. This book details how to manage your ArcSDE instance within this environment from setting up a two- or three-tiered server to serving your spatial data to multiple users and leveraging SQL Server advanced functionality.

Managing databases and logins

Microsoft SQL Server allows for multiple databases. Databases can be mobile, detached, and reattached to other servers using backup and restore or sp_detach_db and sp_attach_db. Databases can be replicated to other servers, can be used as standby servers, or can employ log shipping to distribute logs in backup environments.

Logins access databases as users. ArcSDE 8.1 has special requirements for managing users and databases. You can use integrated (Windows NT or Windows 2000) logins or SQL Server authenticated logins.

ArcSDE data is stored in SQL Server databases in tables, managed with stored procedures and triggers, and accessed with indexes. These entities are used in conjunction to maximize query performance and minimize database maintenance.

Connecting to Microsoft SQL Server

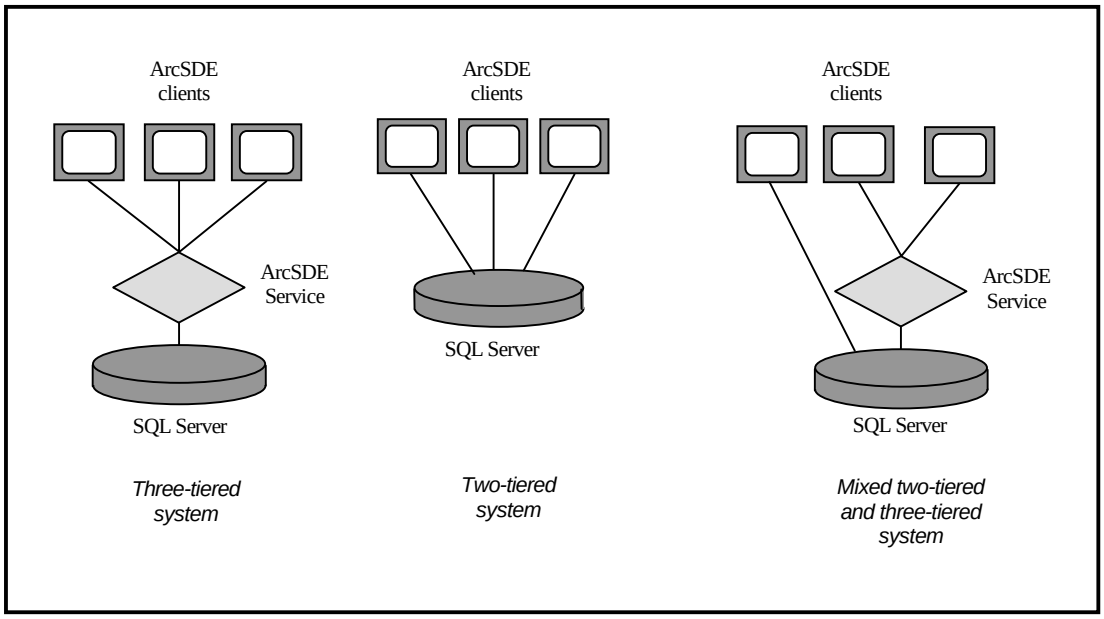
The ArcSDE server provides two different ways to connect to your Microsoft SQL Server 7.0 or 2000 instance. ArcSDE clients may either connect to the ArcSDE service or directly to the database.

The traditional ArcSDE service connection method uses a three-tiered architecture. ArcSDE clients connect to the ArcSDE service, which then spawns a dedicated *gsrvr* process that connects to the database server. The *gsrvr* process brokers the spatial data between the ArcSDE client and the database server. The ArcSDE service and the *gsrvr* processes typically reside on the same machine as the database server, while ArcSDE clients typically operate on remote machines.

The direct connection method uses a two-tiered architecture. The ArcSDE clients connect directly to the Microsoft SQL Server. In this method, the ArcSDE *gsrvr* middle tier has been moved into the ArcSDE client. The ArcSDE clients still run on remote desktops, while the database server runs on a server class machine.

It is possible to use a combination of these two methods on any given database server. With a mixed configuration, some of the ArcSDE clients may be connected directly to a SQL Server server, while others connect to an ArcSDE service.

Figure 1.1: With the traditional three-tiered system the ArcSDE client applications connect to the ArcSDE service, which in turn connects to the server. Within the two-tiered architecture the ArcSDE client applications connect directly to the server. Under a mixed architecture some of the ArcSDE client applications may connect directly to the ArcSDE service, while others connect directly to the server.



Arranging your data

Every table and index created in a database has to be stored somewhere. How you store these tables and indexes can affect how well your database performs under multiuser scenarios and complex queries.

The DBTUNE table

How is the placement of the tables and indexes controlled? ArcSDE uses storage parameters listed in the DBTUNE table that define how and where tables and indexes are created and stored. Storage parameters are grouped into configuration keywords that identify how and where a particular table or index is created.

Prior to ArcSDE 8.1, the configuration keywords were stored in the dbtune.sde file maintained under the ArcSDE etc directory. The dbtune.sde file is still used by ArcSDE 8.1 as the initial source. When the ArcSDE 8.1 sdesetupmssql.exe command executes, the configuration parameters are read from the dbtune.sde file and written to the DBTUNE table. You can import existing dbtune files into the ArcSDE dbtune table using the sdedbtune command. You can make direct edits to this table using the SQL Server Enterprise Manager.

ArcSDE 8.1 simplifies storage parameters by using actual t-sql CREATE TABLE and CREATE PROCEDURE storage parameters.

The ArcSDE 8.1 installation creates the DBTUNE table. If the dbtune.sde file is absent or empty, sdesetupmssql.exe creates the DBTUNE table and populates it with default configuration keywords representing the minimum ArcSDE configuration.

The ArcSDE 8.1 dbtune table has several default settings designed to provide a fast, scalable server “out of the box.” These settings influence clustered indexes, text in row, and network packet size.

In almost all cases, you will populate the table with specific storage parameters for your database. Chapter 3, ‘Configuring DBTUNE storage parameters’, describes the DBTUNE table and all possible ArcSDE storage parameters and default configuration keywords in detail.

National language support

ArcSDE 8.1 supports multiple character sets through the use of the SQL_CHARSET environment variable.

Support for Microsoft SQL Server 2000

Microsoft SQL Server 2000 is fully supported. Some new features of SQL Server 2000 are directly accessible in ArcSDE 8.1 such as text in row, indexable views, and log shipping.

CHAPTER 2

Configuring SQL Server

Microsoft SQL Server is easy to set up and configure. SQL Server has outstanding management tools and wizards to assist you in performing database functions. Due to its ease of use and rich user interface, there are many possible configurations with SQL Server. This chapter guides you through ArcSDE-specific rules for creating and managing databases, logins, and transaction logs.

Installing Microsoft SQL Server

Installing Microsoft SQL Server and starting its database service are very easy. There are only a few general rules you should adhere to when you install:

1. Choose mixed mode security. This specifies that your server supports both Windows NT/Windows 2000 logins and SQL Server Authenticated logins. ArcSDE 8.1 for Microsoft SQL Server supports both modes of security.
2. Place your data files away from your operating system's paging file.
3. Choose a codepage that is appropriate for your language.

Managing databases and filegroups

The ArcSDE installation has the option to create the sde database and user for you. If you choose this option, the sde database will be created within the SQL Server installation directory \data folder (e.g., c:\mssql2k\data). This is acceptable in the beginning, but as your system opens for multiuser access, you'll want to employ some kind of disk striping to reduce disk I/O contention.

This section discusses managing multiple databases for use with ArcSDE, employing disk striping through RAID and Filegroups, and general issues with backup and restore.

NOTE: You must have an 'sde' database.

Using multiple databases

Do you need to create and use multiple databases with ArcSDE? Or can you just use the sde database? The answer depends on the resources you wish to spend to administer your server. Using multiple databases will increase the amount of setup you must perform and can introduce some complexity with connections but makes for a cleaner design and easier maintenance in mature databases.

If you store data in multiple databases, you will deal with smaller files when backing up and restoring data or truncating transaction logs. You can use multiple databases as a security mechanism by separating groups of users and data by database.

While it is recommended to leave the sde database as the sde metadata repository, there are advantages to storing all your data in this database. As ArcSDE returns all spatial data a user has access to in any database upon connection, you can probably speed up your connection and layer list times by leaving all your spatial data in the sde database. Moreover, you will not have to contend with the multidatabase rules of ArcSDE.

ArcSDE multidatabase rules

1. DML (select, insert, update, delete) statements are allowed across databases. For example, you connect as user 'cad' and can select, insert, update, and delete data from any spatially enabled table (a layer or feature class) in any database.
2. DDL (create table, alter table, drop table) statements cannot occur across databases. If you connect as user 'cad' to the 'county_hiways' database, you can only load data (shp2sde, the ArcCatalog shapefile to geodatabase utility) into the 'county_hiways' database.
3. If you do not specify a database when you connect to ArcSDE, you connect to the sde database. The default database is sde, not a user's default database.
4. A connecting user "sees" all feature classes or layers to which they have permission in all databases. Feature classes or layers signify spatially enabled tables.
5. A connecting user sees all tables (regular DBMS tables, even if they are registered with ArcSDE) to which they have permission in the presently connected database. This means that while a user can see all layers in any database when connected to any database, they can only see normal DBMS tables, even if they were created with a ArcSDE tool such as sdetable or the ArcCatalog, in a database they've explicitly connected to. For example, if the cad user has two DBMS tables, income_avg in the sde database and traffic_stats in the county_hiways database, the cad user must explicitly connect to the county_hiways database to view the traffic_stats table.

Filegroups and files

Filegroups and files compose a database. Data resides in files that are grouped by filegroups. The database's primary file of the primary filegroup has a *.mdf extension, while secondary files use *.ndf. Log files are stored in *.ldf files. As mentioned previously, a filegroup is an administrative grouping of files. While files cannot span disks, filegroups can group files across multiple disks.

Files fill proportionately according to their membership in a filegroup. For example, if three files comprise a filegroup, and file A is 20 Mb, file B is 40 Mb, and file C is 60 Mb, one extent is allocated from file A, 2 from B, and 3 from C whenever space is requested from a filegroup.

You are not required to use files and filegroups. You can store all your data in the database's primary filegroup. However, employing files and filegroups offers some advantages, namely simple data striping and parallel queries. By spreading data across multiple disks, you have that many more spindles searching for it during a query. The MS SQL Server query processor will employ a separate thread for each disk storing data on a multi-CPU machine.

Spreading your data across many disks will decrease disk contention as well. When many simultaneous users begin hitting your database, data stored in multiple files in filegroups will help distribute the load of your queries.

When to use filegroups and files

Not all ArcSDE installations will be on multiprocessor server class machines with disk arrays. ArcSDE is certified to scale from Microsoft Data Engine (MSDE) to SQL Server Enterprise Edition. If you are serving data to a few clients on a single processor machine with one or two disk drives, you will achieve no performance benefit by organizing data into files and filegroups.

RAID

Redundant Arrays of Inexpensive or Independent Disks (RAID) boosts performance by striping data into slices across multiple disks in the disk array. By spreading data across multiple disks, all disks share the burden of I/O operations thus reducing the chance of a bottleneck occurring on one disk. RAID's performance increases as you add disks to the array. The operating system and database will see only one volume, a logical representation of the entire disk array.

In a simple configuration, you could create a single disk array of four disks and configure one large data file within that RAID array. Your data would be striped across all four disks evenly, reducing contention. The database's transaction log should not occupy this same array. This solution proves very scalable as well. Additional performance benefits can be gained by adding disks to the array until performance increases begin to decline. More complex configurations would include separate disk arrays for nonclustered indexes, f and s tables, or binary geometry.

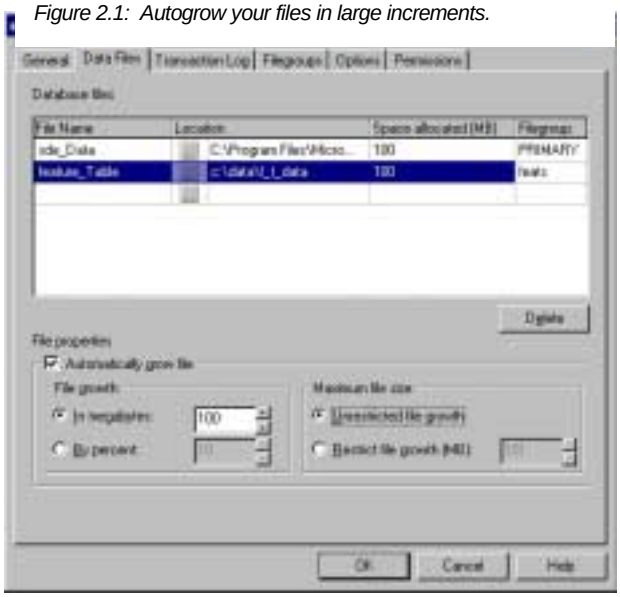
Creating the SDE and other databases

Here are some simple guidelines for creating databases for use with the ArcSDE application. For an example of creating an actual database, see Chapter 5, 'Connecting to SQL Server.'

1. Use files and filegroups because these are SQL Server units of parallelism, can be backed up, and allow you to use dbtune parameters for data placement. Spread your data files across more than one disk. Do not confine all your data to individual disks, but move it across several. There is a point at which too many disks will slow things down.

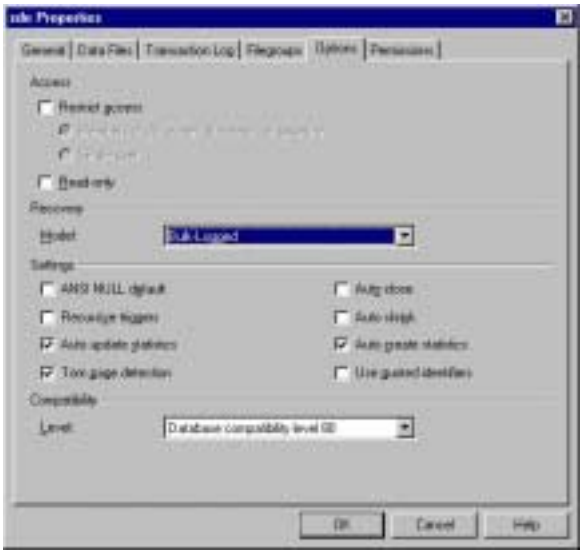
2. Auto-expand your files in large increments. SQL Server will autogrow files, but this can stall queries and fragment your data. Fewer expansions that use more space will result in more contiguous data. Autogrow your transaction log in large increments as well.

Figure 2.1: Autogrow your files in large increments.



3. Leave AUTO_CREATE_STATISTICS and AUTO_UPDATE_STATISTICS enabled. This will prevent your index statistics from becoming stale over repeated edits. Disable auto shrink. Disabling it will prevent the server from attempting to reduce the size of your data files during potentially inopportune times. You will want to manually shrink your database files after data loading and your transaction logs after heavy selections (repeated selections of over 100 features from ArcMAP, for example) or frequent edits.

Figure 2.2: Leave AUTO_UPDATE_STATISTICS and AUTO_CREATE_STATISTICS enabled



4. Create your sde database and transaction logs on a separate disk from the Operating System's paging file.
5. If using RAID 5, keep the tempdb database out of the disk array.

6. If using RAID, consider using different disk arrays for the data files and transaction logs.

7. Refrain from naming your databases with SQL Server reserved words such as "national" or "tran." Keep your database names (and all ArcSDE object names including tables, users, indexes, etc.) under 32 characters in length.

Example: Using SQL to create an sde database

The easiest way to create a database, filegroups, and data files is to use the SQL Server Enterprise Manager. Expand your server, right-click the databases node, and select "New Database." However, you can employ SQL scripts to automate this activity.

This example creates an sde database with one filegroup and one large datafile within a RAID array. The log file is created on a separate disk. The filegroups are created so that the ArcSDE system tables are left on the primary filegroup.

```
Create Database sde
On Primary                  /*created on primary filegroup */
( NAME = sde_dat,
  FILENAME = 'g:\data\sde.mdf',
  SIZE = 40,
  MAXSIZE = UNLIMITED,
  FILEGROWTH = 15% ),
FILEGROUP SpatialData               /*specify    another    filegroup    to
separate*/
( NAME = SpatialData_dat,           /*spatial data*/
  FILENAME = 'g:\data\SpatialData.ndf',
  SIZE = 400,
  MAXSIZE = UNLIMITED,
  FILEGROWTH = 400 )
LOG ON
( NAME = 'SDE_LOG',                 /*place log on different drive*/
  FILENAME = 'F:\logs\sde_log.ldf',
  SIZE = 200,
  MAXSIZE = UNLIMITED,
  FILEGROWTH = 200 )
GO

/*add logins to the server*/
sp_addlogin 'sde', 'esri.sde', 'sde'
go
sp_addlogin 'bob','esri.bob','sde'
go
sp_addlogin 'carto','world.round','sde'
go

/*switch to the sde database and
  add users */
use sde
go
sp_grantdbaccess 'bob'
go
sp_grantdbaccess 'sde'
go
sp_grantdbaccess 'carto'
go

/*permissions*/
grant create table to sde
grant create procedure to sde
grant create view to sde

grant create table to bob
grant create table to carto
```

Managing logins and users

The ArcSDE install program gives you the option of creating the sde database and user. If you allowed it to create the sde user and database, then you can probably start your ArcSDE service. If you did not, you'll need to create an 'sde' login and add it as a user to the sde database.

Microsoft SQL Server has 'logins' and 'users.' A login is a named account that has access rights to the database server but not necessarily access to a database. A user is a login that has been granted access to a database.

There are two types of authentication: SQL Server authentication and Windows NT or Windows 2000 authentication. SQL Server authentication specifies that the named accounts are valid only in SQL Server. Windows NT or Windows 2000 authentication signifies that an operating system account is also valid within the database server.

MS SQL Server also supports roles. Roles are special groupings of users to which a specific privilege is granted. There are three types of roles: fixed server, database, and user-defined. Fixed server roles span the server and apply to all databases. Database and user-defined roles pertain to a specific database.

Once you have installed the software and started the ArcSDE service, or if you want to "preconfigure" your server for ArcSDE, you must create logins and users. We recommend that you use the 'sde' user as the sde application dba and not as a standard user. If you choose to follow this recommendation, you'll need to create new logins and grant them access to your databases.

Logins and users: The rules for ArcSDE

Before you create logins and users for ArcSDE, you must understand the requirements for permissions and database access. Because you can have multiple databases in MS SQL Server and have different types of authentication, you should ensure that your logins adhere to these rules to prevent unexpected client behavior when accessing your spatial data. Remember that a database user is a login that has been granted access to that database.

1. The 'sde' user must have create table, create view, and create procedure permission within the 'sde' database. The sde user must be able to create the metadata tables and stored procedures that manage an ArcSDE instance. If you are using SQL Server 2000, the sde user must also have "create function" permission.

2. The 'sde' login must be granted access to any other database in which you will store spatial data.

3. You must add all logins that will access spatial data as users to the sde database and grant them create table permission. These users must be able to write their logfiles into the sde database and select against the sde metadata. NOTE: The first time a user accesses spatial data managed by ArcSDE, they create two logfile tables in the sde database. As long as these tables are not deleted, it is permissible to revoke their create table permission because they have created their logfile tables.

4. Any user that will own (create) data in any database through the ArcSDE application must have create table and create procedure in that database. They must have create table to create the tables comprising an ArcSDE feature class and create procedure to create objectid generation stored procedures that accompany a feature class.

5. You must explicitly connect to a database to create data in it, drop a table from it, or alter a table within it. In other words, DDL statements can only be issued on data if you have explicitly connected to the database storing that data. If you don't specify a database

name in your connection string, you connect, by default, to the sde database. We ignore a login's default database!

6. You can connect to any database and edit data to which you have permissions in any other database.

7. You should have a separate account for all your application users to reduce contention against the logfile tables in the sde database. The logfile tables are used to store selection ID's in programs such as ArcGIS or ArcView® DBAccess. If you have multiple users logging into ArcSDE as the same account and then selecting features in their client programs, they may see unpredictable results or experience performance problems due to their sharing of a single logfile table.

8. You cannot use a SQL Server reserved word for a login or user name. Examples of reserved words are Max, National, and Count. See the SQL Server books online (BOL) for a complete listing.

ArcSDE users rule matrix

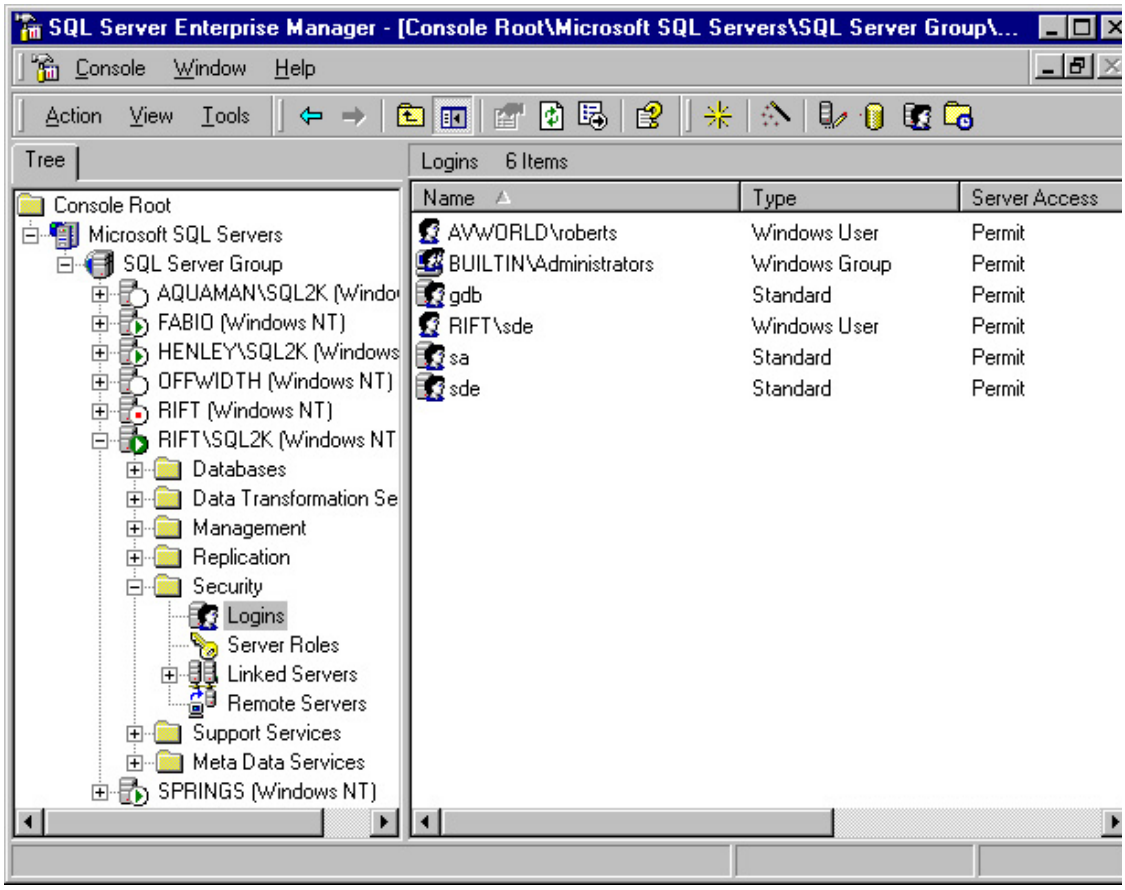
User	In SDE database	In other database
SDE User	create table, create view, create procedure, create function (SQL Server 2000)	must be added as a user
Other User	create table, create procedure if will own data	add as a user if will access data, create table and procedure if will own data

Creating a new SQL Server authenticated login for use in ArcSDE

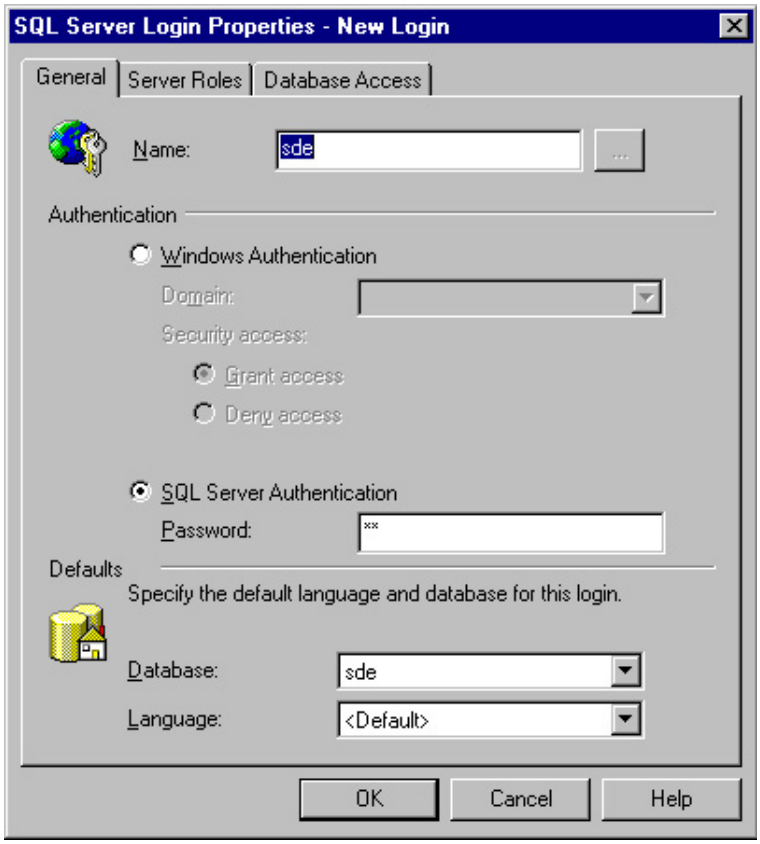
Keep in mind there are multiple ways to do this. SQL Server 7 and 2000 also differ slightly, but the idea is the same: you must first create or add a new login to the relational database management system (RDBMS) before you can access data as that login. Each topic listed below describes the entire process from creating the login to granting it access to a database or databases, and granting that user privileges within a database.

Enterprise Manager

1. Open the SQL Server Enterprise Manager, expand your server's branch in the treeview, expand the "security" node, and select the logins topic.

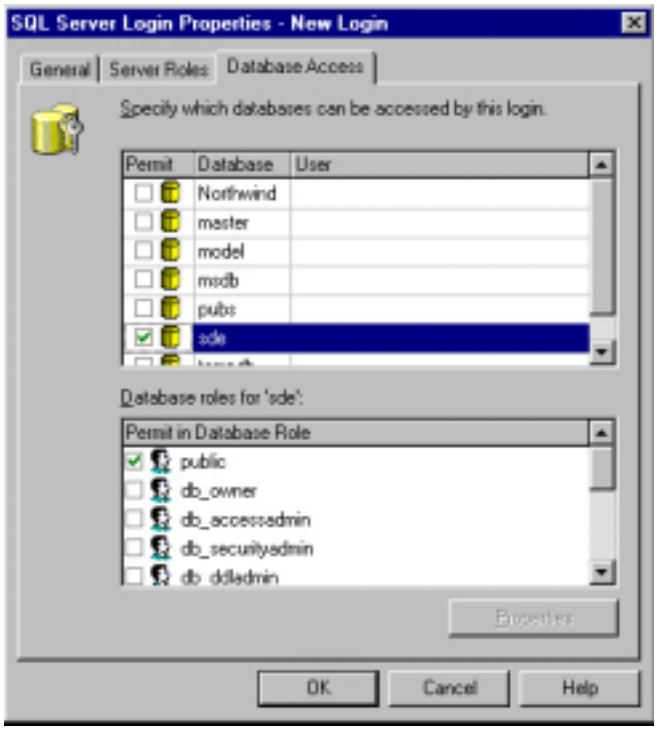


2. In the details panel on the right, right-click and select “new login.”



Select the SQL Server Authentication radio button, name the login, and provide a password. Assign a default database.

3. Now click the Database Access tab. Scroll through the list of databases and select the sde database and any other database you want to add this login to as a user.



4. Click OK and you'll be prompted to confirm your password. The process both creates the new login and grants it access to a database or databases (adds that login as a user to a database).

5. Now expand the databases tree in the treeview and right-click the sde database (or any other database you've added that login to as a user). Select Properties from the popup menu. In the properties form, select the Permissions tab. You'll see a grid of users of this database with columns listing permissions. Check off "create table" and "create procedure" for a user who will own data in this database. Check off "create table" for any user in the sde database. Click OK.

Transact SQL

Scripting the creation of your logins and users is much faster than using the Enterprise Manager. The same rules apply as with the Enterprise Manager. Below is a SQL script that will demonstrate this idea.

```
/*
        Add sde and non-sde login and user
*/

/*
        Step1: Add the logins to the server using
        the sp_addlogin stored procedure.
        sp_addlogin '<login_name>','<pass>','<default_db>'
*/

use master
go
sp_addlogin 'sde','spatial.data','sde'
go
sp_addlogin 'arthur','dent','sde'
go
```

```
/*
        Step2: Grant the login access to the sde database
        using sp_grantdbaccess. Grant the user create table
        and stored procedure. You must switch to the sde
        database to do this.
*/

use sde
go
sp_grantdbaccess 'sde'
go
sp_grantdbaccess 'arthur'
go
grant create table to sde
grant create table to arthur
grant create procedure to sde
grant create view to sde
grant create function to sde  --SQL Server 2k only!
grant create procedure to arthur
go
```

Using Windows 2000 or Windows NT logins with ArcSDE

ArcSDE 8.1 for MS SQL Server supports both SQL Server and Windows authentication.

Windows users are parts of domains and groups. You can add individual windows users or groups to SQL Server and connect with them to ArcSDE data stores. There are issues with this implementation as login names and user names can differ.

To use integrated logins with ArcSDE you must:

1. Add the domain user to a group on the ArcSDE server.
2. Add the domain user to the SQL Server as a new login.
3 Grant the new login access to a database.
4. Grant the login appropriate permissions within the database.

You don't have to perform points 2-4 if your Windows NT user is added to the machine and is a member of the Windows NT administration group on that machine.

NOTE: Windows NT groups are not supported at ArcSDE 8.1.

Using Windows 2000 or NT 'sde' login

You can use an integrated 'sde' login instead of a SQL Server authenticated sde login. Keep in mind that the installer will always create a SQL Server 'sde' authenticated login. However, you don't have to run the postinstaller to set up your sde database.

To use an integrated sde login:

1. Add the domainname\sde login to the SQL Server logins collection.
2. Add the sde login to the sde database (and any other database you intend to use for spatial data storage) as a user.
3. Grant the sde login create table, view, stored procedure, and function (create function is necessary in SQL Server 2000 only).
4. Login to Windows as domainname\sde. Open a DOS prompt and switch directories to %SDEHOME%\bin. Run sdesetupmssql –o install as the sde user. Do not input a username or password.
   sdesetupmssql –o install –s rift –H C:\sde81\sqlexe

5. Or, open a DOS prompt and switch directories to %SDEHOME%\bin. Run sdesetupmssql –o install as the sde user. Input a null username or password if your windows account has admin privileges or use the "sa" account.
   sdesetupmssql –o install –s rift –H C:\sde81\sqlexe
   sdesetupmssql –o install –s rift –H C:\sde81\sqlexe –u sa –p <sa password.
6. Run sdeservice –o create to create an sde service. Specify a null string for your password –p "" argument. Use the minimum options: -s, -l, -d, -H.
   sdeservice –o create –s rift\sql2k –l Fabio –d SQLSERVER –H C:\sde81\sqlexe –i esri_sde –p ""
7. Start the sdeservice: net start esri_sde, alternatively, make a direct connection to it.

Connecting to an ArcSDE service as a Windows NT login

You can connect to an ArcSDE service without using a username or password with some ArcSDE clients. The examples below use the administration commands to demonstrate this functionality.

Sdelayer:

```
C:\>sdelayer -o describe -i sql82k
ArcSDE 8.1 Build 633 Tue Nov 21 22:30:10 PST 2000
Layer Administration Utility
-----------------------------------------------------

-----------------------------------------------------------------
Database : VTEST
Table Owner : VTEST
Table Name : SANDIEGO_NET_JUNCTIONS
Spatial Column : SHAPE
Layer id : 8
Entities : npc
Layer Type : SDE
I/O Mode : NORMAL
User Privileges : SELECT, UPDATE, INSERT, DELETE
Layer Configuration: DEFAULTS
```

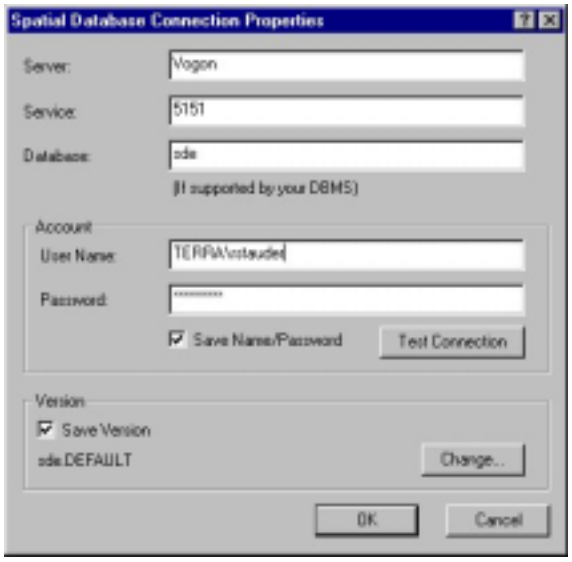
sdetable:

```
C:\>sdetable -o create -t authentication_test -d "colA string" -i sql82k


ArcSDE 8.1 Build 633 Tue Nov 21 22:30:10 PST 2000
Attribute Administration Utility
-----------------------------------------------------
Successfully created table authentication_test.
```

Sdeimport:

```
E:\world_sdex>sdeimport -o create -l test,shp -i sql82k -f cntry94

ArcSDE 8.1 Build 633 Tue Nov 21 22:30:10 PST 2000
SDEX File Import Administration Utility
-----------------------------------------------------
Importing SDEX from cntry94 ...
165 features read.
165 features stored.
```

ArcCatalog:



NOTE: Using integrated logins (Windows 2000 and NT) in direct connect

If you make a direct connection to ArcSDE using a Windows 2000 or NT user, your connection login and password will be taken from your operating system login.

Using roles

Database and user-defined

Adding a database user to a role simplifies administration by reducing the number of things you have to do per user. If you define a role that has create table and procedure and then add all your ArcSDE users to it, you have reduced the number of times you have to grant a privilege to each one. Or you can use a predefined database role such as db_datareader or db_datawriter to accomplish a similar end. Make sure you consult the SQL Server 7 or 2000 Books Online so you understand what each role allows.

You can use the sdelayer -o grant or -o revoke command to grant or revoke a privilege (SELECT, INSERT, UPDATE, DELETE) to a role. For example, you have a role called 'highways_editors' in the sde database. This role contains the user we created above, 'arthur.' You could grant this role SELECT, INSERT, UPDATE, DELETE permissions against an ArcSDE feature class, 'county_hiways.'

```
sdelayer -o grant -U highways_editors -A SELECT, INSERT, UPDATE, DELETE -l
county_hiways,shape -u county -p big.bob -i esri_sde -s vogon
```

Considerations with database role "db_owner"

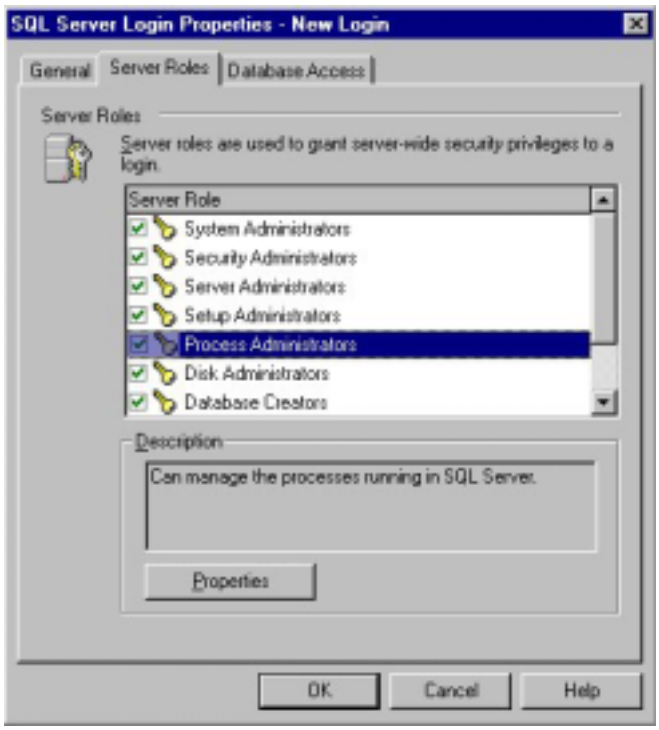
Adding a user to this role makes that user 'dbo' within that database. This means that this user can execute any DDL or DML statement against any fully qualified table in that database. They can also grant other users permissions within that database. However, ownership of their database objects remains in their account name, not dbo. For example, if the user 'cad' is added to the db_owner role and then loads data into the sde database, the created tables are owned by 'cad' and not dbo, even though this user is equivalent to 'dbo' in that database.

If the login 'cad' **creates the database**, the rules change. Now, any object that cad creates is owned by dbo. Since ArcSDE stores ownership names in several tables, inconsistency in qualification can confuse connecting clients.

Fixed server roles

Fixed server roles pertain to the server, and hence, all databases. There are two main things to consider here: if you add a user to the sysadmin fixed server role, anything that user creates in the database is owned by 'dbo' in any database, and you can add a user to multiple, sometimes conflicting, roles. This last point will assuredly cause problems when you connect and attempt to view (or load) your data. It is easy to do this, so make sure you don't. You assign a login (correctly or incorrectly) to one or more fixed server roles upon login creation. Figure 2.5 is a screenshot of SQL Server 2000 that shows this. Be careful you don't do this!

*Figure 2.3: Adding a login to all fixed server roles—**Don't do this!***



Roles: Summary rules

1. Avoid the use of fixed server roles where possible, especially the sysadmin fixed server role.

2. Do not place users into every role. Be very selective about which roles you place them into.

3. Avoid any role that will change object ownership away from a username to 'dbo.'

NOTE: If you are going to use Windows NT or 2000 authentication, you may not be able to conform to rule 3.

If all of your data is owned by 'dbo' yet you loaded it as a regular user, you should check that login's properties to make sure you have not inadvertently added it to multiple roles.

Who should own the data?

You should carefully consider who will own the data or specific datasets in your ArcSDE databases. ArcSDE 8.1 allows you to view and edit (DML) data that you have permission to in any database regardless of your connection database.

For example, the cad user has full permissions to data in the contours database, the county_hiways database, and the sde database. This user can explicitly connect to any of the three databases to view or edit data. Every time the cad user connects to ArcSDE, ArcSDE must figure out what dataset names it must return to the client. This list of dataset names would represent what datasets that cad has permissions to. The more data cad has permissions to, the longer it takes to return this list.

Can the sde user own all the data? Although there is no pressing reason to not have the sde user own all the data, it is cleaner to reserve the sde user as the owner of the sde and gdb system tables, the 'dba' of the sde application.

Data ownership recommendations

1. Have the sde user own only the sde and gdb application system tables and stored procedures.

2. Be careful if you will have 'dbo' own all your data. You may encounter unexpected results when querying for layers.

3. Roles increase the amount of data a user has permissions to.

4. The more data a user has permission to, the longer it will take to connect.

Managing transaction logs

Transaction logs are composed of a physical log, virtual logs, and a logical log. The physical log is a disk space allocation, and virtual logs are its internal segments. The logical log contains all undeleted transaction records and the active portion of this structure.

Transaction logs record transactions, data modifications, extent allocations and deallocations, and creation/destruction of tables and indexes. This record allows you to:

1. ROLLBACK changes made within a transaction.
2. During recovery, roll back incomplete transactions initiated before a database failure.
3. During recovery, roll forward modifications listed in the log but not yet written to disk.

Each record written to the log has a Log Sequence Number (LSN). All new records are written to the end of the log and have a higher LSN than their predecessor. Logs "wrap around," or consume, as much open space as necessary, until the end of the logical log reaches its beginning. This will cause the physical log to autogrow if allowed or throw an error. The log will autotruncate itself when automatic checkpoints occur. However, the log cannot truncate itself past the Minimum Log Sequence Number (MinLSN), which can be the oldest uncommitted transaction listed.

Sooner or later, you'll notice that your transaction logs have grown to considerable size. ArcSDE Transaction logs grow due to data loading, editing, and making selections into sde_logfiles. When you make selections with ArcGIS clients such as ArcMap™, all selections exceeding 100 records are written into a database logfile. The continuous inserting and deleting from this table will make the database transaction log grow.

As transaction logs are critical to recovery, ideally you'll have a series of backed up logs you can use to restore your database to a point in time if you encounter a database failure. Backing up your database and logfile should control the size of the log by truncating it. If you have not made regular backups of the transaction log, you'll need to take action to reduce its size. To reduce the amount of space used within the physical logfile by the logical log, truncate it. To reduce the overall size of the physical logfile, shrink it.

Truncating transaction logs

Truncating a transaction log refers to the process of removing records from the log that are no longer necessary for database recovery. These LSNs of these records would

precede the MinLSN of the active portion of the logical log. Truncating the log will not shrink the size of the physical log.

To truncate your log do one of the following:

1. Backup the database using either the Enterprise Manager or the Backup statement.
2. Backup the logfile using the Enterprise Manager or the Backup statement.
3. Set truncate log on checkpoint. This is an automatic setting that will automatically truncate your database whenever checkpoint occurs. Checkpoints flush dirty pages from the buffer cache and are determined with the Recovery Interval or whenever you:
    a. Issue the CHECKPOINT command.
    b. Shutdown the database.
    c. Issue an alter table statement.

    Setting truncate log on checkpoint has consequences.

Once you've issued the backup statement, all records preceding the MinLSN are marked for reuse, and the reported size of the transaction log's logical log is smaller. The following figures demonstrate this.

Figure 2.4: A transaction log after registering some very large feature classes with the Geodatabase

Figure 2.5: The same log after a database backup. The log has been truncated.

Shrinking the transaction log

Truncating the log marks all nonactive transactions as reusable but does not reclaim any space, only delaying any more growth to the physical log file. In the case shown above in Figure 2.5, the physical log file still consumes 315.8 MB even though only 11.4 MB are active. You may find that you need some of this extra space. Shrinking the transaction log will return allocated space to the operating system.

To shrink the transaction log do one of the following:

1. Use the Enterprise Manager's Shrink database utility: Right-click your database and select all tasks – Shrink Database. Input your desired resulting free space and click OK.
2. Use DBCC SHRINKDATABASE.

In the figure below, we used the Enterprise Manager to shrink the logfile to 49.7 MB.

Figure 2.6: The logfile's new size after shrinking it

Help! It won't shrink!

Under certain circumstances, you will not be able to shrink the size of the logfile. As the logical log progresses through serial lists of Log Sequence Numbers, the active portion of the log will arrive at the end of the physical file. You cannot return space preceding the MinLSN to the operating system. What you must do is advance the MinLSN to the beginning of the physical file by following this process:

1. Backup the database. This will mark the preceding records to the minimum log sequence as reusable. Check the size of the log file.
2. Execute some updates using t-sql. This should move the active portion of the log toward the beginning of the physical file.
3. Execute dbcc shrinkdatabase (dbname) to create a "shrinkpoint."
4. Backup the log file.

At this point you should see more free space in the logfile. Repeat a few times if necessary.

CHAPTER 3

Configuring DBTUNE storage parameters

DBTUNE storage parameters allow you to control how ArcSDE clients create objects within a SQL Server database. They determine such things as which filegroups a table or index is created in. Specific SQL Server tuning parameters are stored in the DBTUNE file, which is used by ArcSDE to create the SDE_dbtune table.

The DBTUNE table

The dbtune storage parameters are maintained in the DBTUNE metadata table. The DBTUNE table along with all other metadata tables is created during the setup phase that follows the installation of the ArcSDE software. The DBTUNE table has the following definition.

Column Name	Data Type	Length	Allows Null
keyword	varchar	32	no
parameter_name	varchar	32	no
config_string	varchar	2048	yes

The keyword field stores the keywords. Within each keyword, there are a number of parameters. The names of these are stored in the parameter_name field. Each parameter has a configuration string associated with it, and this is stored in the config_string field. After creating the DBTUNE table, the setup phase of the ArcSDE 8.1 installation populates the table with the contents of the dbtune.sde file, which it expects to find under the etc directory of the SDEHOME directory. If the DBTUNE table already exists, the ArcSDE setup phase will not alter its contents, should you decide to run it again.

Editing the DBTUNE table

Although users are free to edit the contents of the DBTUNE table using a SQL interface such as Query Analyzer, the sdedbtune administration tool has been provided to enable the export of the contents of this table to a file. The file can then be edited in a Windows NT file-based editor such as "WordPad". After updating the file, users can then repopulate the DBTUNE table using the import operation of the sdedbtune command. In the following example the DBTUNE table is exported to the dbtune.out file, the file is edited with Windows WordPad, and then it is imported after making changes to the file.

```
C:\>sdedbtune -o export -f dbtune.out -i sql82k -s rhiannon -D sde -u sde -p go
ArcSDE   8.1      Build 664 Tue Dec 26 22:32:22 PST 2000
Attribute      Administration Utility
------------------------------------------------------
     Successfully exported to file SDEHOME\etc\dbtune.out
C:\>write %sdehome%/etc/dbtune.sde

C:\>sdedbtune -o import -f dbtune.out -i sql82k -s rhiannon -D sde -u sde -p go
ArcSDE   8.1      Build 664 Tue Dec 26 22:32:22 PST 2000
Attribute      Administration Utility
------------------------------------------------------
Import DBTUNE Table. Are you sure? (Y/N): y

     Successfully imported from file SDEHOME\etc\dbtune.out
```

The sdedbtune administration tool always exports the file in the etc directory of the ArcSDE home directory. You cannot relocate the file to another directory with a qualifying pathname. By not allowing the relocation of the file, the sdedbtune command ensures that they are located in the ArcSDE software's etc directory under the ownership of the ArcSDE administrator.

Arranging storage parameters by keyword

Parameters of the DBTUNE table are grouped by keyword. When the contents of the dbtune table are exported to a file, the keywords are prefixed by two pound signs "##". The 'END' clause terminates each keyword. Keywords define the storage configuration of simple objects such as tables and indexes and complex objects such as feature classes, network classes, and raster columns. ESRI client applications and some ArcSDE administration tools assign DBTUNE keywords to these objects. The pound signs '##' are not included when the keywords are assigned.

DEFAULTS keyword

Each DBTUNE table has a fully populated DEFAULTS keyword. The DEFAULTS keyword can be selected whenever you create a table, index, feature class, or raster column. If you do not select a keyword for one of these objects, the DEFAULTS keyword is used. If you do not include a parameter in a keyword you have defined, ArcSDE substitutes the parameter from the DEFAULTS keyword. The DEFAULTS keyword relieves you of the need to define all the parameters for each of the keywords you define. The parameters of the DEFAULTS keyword should be populated with values that represent the average storage configuration of your data. During installation, if the ArcSDE software detects a missing DEFAULTS keyword parameter in the dbtune.sde file, it automatically adds the parameter. If you import a dbtune file with the sdedbtune command, it will automatically add any parameters that are missing. ArcSDE will detect the presence of the following list of parameters and insert the parameter and the default configuration string.

```
##DEFAULTS
NUM_DEFAULT_CURSORS -1
UI_TEXT    ""
A_CLUSTER_ROWID          0
A_CLUSTER_SHAPE          1
A_CLUSTER_STATE_ROWID          0
A_CLUSTER_STATEID       0
A_CLUSTER_USER 0
A_INDEX_ROWID   "WITH FILLFACTOR = 75"
A_INDEX_SHAPE   "WITH FILLFACTOR = 75"
A_INDEX_STATE_ROWID   "WITH FILLFACTOR = 75"
A_INDEX_STATEID"WITH FILLFACTOR = 75"
A_INDEX_USER     "WITH FILLFACTOR = 75"
A_STORAGE          ""
B_CLUSTER_ROWID          0
B_CLUSTER_SHAPE          1
B_CLUSTER_USER 0
B_INDEX_ROWID   "WITH FILLFACTOR = 75"
B_INDEX_SHAPE   "WITH FILLFACTOR = 75"
B_INDEX_USER     "WITH FILLFACTOR = 75"
B_STORAGE          ""
B_TEXT_IN_ROW   256
D_CLUSTER_ALL   0
D_CLUSTER_DELETED_AT 1
D_CLUSTER_STATE_ROWID          0
D_INDEX_ALL       "WITH FILLFACTOR = 75"
D_INDEX_DELETED_AT     "WITH FILLFACTOR = 75"
D_INDEX_STATE_ROWID   "WITH FILLFACTOR = 75"
D_STORAGE          ""
F_CLUSTER_FID     1
F_INDEX_AREA      "WITH FILLFACTOR = 75"
F_INDEX_FID       "WITH FILLFACTOR = 75"
F_INDEX_LEN       "WITH FILLFACTOR = 75"
F_STORAGE          ""
F_TEXT_IN_ROW   256
S_CLUSTER_ALL    1
S_CLUSTER_SP_FID          0
S_INDEX_ALL       "WITH FILLFACTOR = 75"
S_INDEX_SP_FID    "WITH FILLFACTOR = 75"
S_STORAGE          ""
END
```

Setting the system table DATA_DICTIONARY keyword

During the execution of the install operation of the sdesetupmssql administration tool, the ArcSDE and geodatabase system tables and indexes are created with the storage parameters of the DATA_DICTIONARY keyword. Users may customize the keyword in the dbtune.sde file (found in the %SDEHOME%\etc directory) prior to running the sdesetupmssql tool. In this way you can change default storage parameters of the DATA_DICTIONARY keyword. Edits to all of the geodatabase system tables and most of the ArcSDE system tables occur when schema change occurs. As such, edits to these system tables and indexes usually happen during the initial creation of an ArcGIS database with infrequent modifications occurring

whenever a new schema object is added. Four of the ArcSDE system tables, SDE_version, SDE_states, SDE_state_lineages, and SDE_mvtables_modified, directly participate in the ArcSDE versioning model and record events resulting from changes made to multiversioned tables. If your site makes extensive use of a multiversioned database, these tables and their associated indexes are highly active. Separating these objects into their own filegroups allows users to position their data files on disks that experience low I/O activity. This will avoid as much disk I/O contention as possible. If the dbtune.sde file does not contain the DATA_DICTIONARY keyword, or if any of the required parameters are missing from the keyword, the following records will be inserted into the DATA_DICTIONARY when the table is created.

```
##DATA_DICTIONARY
B_CLUSTER_ROWID          0
B_CLUSTER_USER 0
B_INDEX_ROWID   "WITH FILLFACTOR = 75"
B_INDEX_USER      "WITH FILLFACTOR = 75"
B_STORAGE          ""
MVTABLES_MODIFIED_INDEX "WITH FILLFACTOR = 75"
MVTABLES_MODIFIED_TABLE ""
STATE_LINEAGES_INDEX    "WITH FILLFACTOR = 75"
STATE_LINEAGES_TABLE    ""
STATES_INDEX    "WITH FILLFACTOR = 75"
STATES_TABLE    ""
VERSIONS_INDEX  "WITH FILLFACTOR = 75"
VERSIONS_TABLE  ""
END
```

Changing the appearance of DBTUNE keywords in the ArcInfo user interface

ArcSDE 8.1 introduces two new parameters that will support the ArcInfo user interface UI_TEXT and UI_NETWORK_TEXT. ArcSDE administrators can add one of these parameters to each keyword to communicate to the ArcInfo™ schema builders the intended use of the keyword. The configuration string of these parameters will appear in ArcInfo interface dbtune keyword scrolling lists. The UI_TEXT parameter should be added to keywords that will be used to build tables, feature classes, and indexes. UI_NETWORK_TEXT parameter should be added to parent network keywords.

LOGFILE keywords

Logfiles are used by ArcSDE to maintain temporary and persistent sets of selected records. Whenever a user connects to ArcSDE for the first time, the SDE_logfiles and SDE_logfile_data tables and indexes are created. Administrators may create a keyword for each user that begins with the LOGFILE_<username>. For example, if the user's name is STANLEY, ArcSDE will search the dbtune table for the LOGFILE_STANLEY keyword. If this keyword is not found, ArcSDE will use the parameters of the LOGFILE_DEFAULTS keyword to create the SDE_logfiles and SDE_logfile_data tables. ArcSDE always creates the DBTUNE table with a LOGFILE_DEFAULTS keyword. If this keyword is not specified in the dbtune file that is imported with the sdedbtune command, ArcSDE will populate the dbtune

table with a default LOGFILE_DEFAULTS parameter. In addition, if the dbtune file contains some of the LOGFILE_DEFAULTS keyword parameters, ArcSDE will supply the rest. Therefore, the LOGFILE_DEFAULTS keyword is always fully populated. If a user-specific keyword exists, but some of the parameters are not present, the parameters of the LOGFILE_DEFAULTS keyword are used. If some of the parameters are not set in either a user-specific keyword or the LOGFILE_DEFAULTS keyword, the SQL Server defaults are used. Creating a logfile keyword for each user makes it possible to separate their logfiles onto different devices by specifying the filegroup the logfile tables and indexes are created in. Most installations of ArcSDE will function well using the LOGFILE_DEFAULTS parameters supplied with the installed dbtune.sde file. However, for applications that make heavy use of logfiles, such as ArcInfo, ArcEditor™, ArcIMS®, and ArcView, it may help performance by spreading the logfiles across separate filegroups. Typically, logfiles are updated whenever a selection set that exceeds 100 records. If the imported dbtune file does not contain a LOGFILE_DEFAULTS keyword, or if any of the logfile parameters are missing, ArcSDE will insert the following records:

```
##LOGFILE_DEFAULTS
LD_CLUSTER_ALL 0
LD_CLUSTER_DATA_ID      0
LD_CLUSTER_ROWID        1
LD_INDEX_ALL      "WITH FILLFACTOR = 75"
LD_INDEX_DATA_ID          "WITH FILLFACTOR = 75"
LD_INDEX_ROWID "WITH FILLFACTOR = 75"
LD_STORAGE          ""
LF_CLUSTER_DATA_ID      0
LF_CLUSTER_ID     0
LF_CLUSTER_NAME          0
LF_INDEX_DATA_ID          "WITH FILLFACTOR = 75"
LF_INDEX_ID         "WITH FILLFACTOR = 75"
LF_INDEX_NAME   "WITH FILLFACTOR = 75"
LF_STORAGE          ""
UI_TEXT     ""
END
```

Index parameters

Index parameters define the storage configuration of a SQL Server index. The index parameter is appended to a SQL Server CREATE INDEX statement during its creation by ArcSDE. Valid entries into an ArcSDE index parameter include all SQL Server CREATE INDEX statement parameters to the right of the statement's column list.

The business table parameter

A business table is any SQL Server table created by an ArcSDE client, the sdetable administration command, or the ArcSDE C application programming interface (API) SE_table_create function. Use the DBTUNE table's B_STORAGE parameter to define the storage configuration of a business table.

The business table index parameters

Three index parameters exist to support the creation of business table indexes. The B_INDEX_USER parameter holds the storage configuration for user-defined indexes created with the C API function SE_table_create_index and the create_index operation of the sdetable command. The B_INDEX_ROWID parameter holds the storage configuration of the index ArcSDE creates on the register table's object ID column, commonly referred to as the ROWID. A registered table can be created with the alter_reg operation of the sdetable command or from the ArcCatalog interface. The B_INDEX_SHAPE parameter holds storage configuration of the spatial column index ArcSDE creates when a spatial column is added to a business table. This index is created by the ArcSDE C API function SE_layer_create. This function is called by ArcInfo when it creates a feature class and by the add operation of the sdelayer command.

Multiversioned table parameters

Registering a business table or feature class as multiversioned allows multiple users to maintain and edit their copy of the object. At appropriate intervals each user merges the changes they have made to their copy with the changes made by other users and reconciles any conflicts that arise when the same features are modified. ArcSDE creates two tables—the adds table and the deletes table—for each table that is registered as multiversioned. The A_STORAGE parameter maintains the storage configuration of the adds table. Four other parameters hold the storage configuration of the indexes of the adds table. The adds table is named A<n>, where <n> is the registration ID listed in the SDE.TABLE_REGISTRY system table. For instance, if the business table ROADS is listed with a registration ID of 10, ArcSDE creates the adds table as A10. The A_INDEX_ROWID parameter holds the storage configuration of the index ArcSDE creates on the multiversion object ID column, commonly referred to as the ROWID. The adds table ROWID index is named A<n>_ROWID_IX1, where <n> is the business table's registration ID that the adds table is associated with. The A_INDEX_STATEID parameter holds the storage configuration of the index ArcSDE creates on the adds table's SDE_STATE_ID column. The SDE_STATE_ID column index is called A<n>_STATE_IX2, where <n> is the business table's registration ID that the adds table is associated with. The A_INDEX_SHAPE parameter holds the storage configuration of the index ArcSDE creates on the adds table's spatial column. If the business table contains a spatial column, the column and the index on it are duplicated in the adds table. The adds table's spatial column index is called A<n>_IX1_A, where <n> is the layer ID of the feature class as it is listed in the SDE.LAYERS table. The A_INDEX_USER parameter holds the storage configuration of user-defined indexes ArcSDE creates on the adds table. The user-defined indexes on the business tables are duplicated on the adds table. The D_STORAGE parameter holds the storage configuration of the deletes table. Two other parameters hold the storage configuration of the indexes ArcSDE creates on the deletes table. The deletes table is named D<n>, where <n> is the registration ID listed in the SDE.TABLE_REGISTRY system table. For instance, if the business table ROADS is listed with a registration ID of 10, ArcSDE creates the deletes table as D10. The D_INDEX_ STATE_ROWID parameter holds the storage configuration of the D<n>_IDX1 index ArcSDE creates on the deletes table's SDE_STATE_ID and SDE_DELETES_ROW_ID columns. The D_INDEX_DELETED_AT parameter holds the storage configuration of the D<n>_IDX2 index ArcSDE creates on the deletes table's SDE_DELETED_AT column. **Note:** If a keyword is not specified when the registration of a business table

is converted from single-version to multiversion, the adds and deletes tables and their indexes are created with the parameters of the configuration keyword the business table was created with.

Feature class parameters

A feature class created with a compressed binary storage (image datatype) format adds two tables to the SQL Server database—the feature table and the spatial index table. Three indexes are created on the feature table, and two indexes are created on the spatial index table. The storage parameters for these tables and indexes follow the same pattern as the B_STORAGE and B_INDEX_* parameters of the business table. The F_STORAGE parameter holds the SQL Server CREATE TABLE storage configuration of the feature table. The feature table is created as F_<n>, where <n> references the layer ID of the table's feature class found in the SDE_layers table. The F_INDEX_FID parameter holds, the SQL Server CREATE INDEX storage configuration of the feature table's spatial column index. The spatial column is created as F_<n>_IX1, where <n> references the layer ID of the index's feature class found in the SDE_layers table. The F_INDEX_AREA parameter holds the SQL Server CREATE INDEX storage configuration of the feature table's area column index. The spatial column is created as F_<n>_AREA, where <n> references the layer ID of the index's feature class found in the SDE_layers table. The F_INDEX_LEN parameter holds the SQL Server CREATE INDEX storage configuration of the feature table's length column index. The spatial column is created as F_<n>_LEN, where <n> references the layer ID of the index's feature class found in the SDE_layers table. The S_STORAGE parameter holds the SQL Server CREATE TABLE storage configuration of the spatial index table. The spatial index table is created as S_<n>, where <n> references the layer ID of the spatial index table's feature class found in the SDE_layers table. The S_INDEX_ALL parameter holds the SQL Server CREATE INDEX storage configuration of the spatial table first index. The spatial index table is created as S_<n>_IX1, where <n> references the layer ID of the index's feature class found in the SDE_layers table. The S_INDEX_SP_FID parameter holds the SQL Server CREATE INDEX storage configuration of the spatial table second index. The spatial index table is created as S_<n>_IX2, where <n> references the layer ID of the index's feature class found in the SDE_layers table.

Network class composite keywords

The composite keyword is a unique type of keyword designed to accommodate the tables of the ArcGIS network class. The network table's size variation requires a keyword that provides configuration parameters for both large and small tables. Typically, the network descriptions table is very large in comparison with the others. To accommodate the vast difference in size of the network tables, the network composite keyword is subdivided into elements. A network composite keyword has three elements: the parent element defines the general characteristic of the keyword and the junctions feature class, the description element defines the configuration of the DESCRIPTIONS table and its indexes, and the network element defines the configuration of the remaining network tables and their indexes. The parent element does not have a suffix, and its keyword looks like any other keyword. The description element is demarcated by the addition of the ::DESC suffix to the parent element's keyword, and the network element is demarcated by the addition of the ::NETWORK suffix to the parent element's keyword.

```
##NETWORK_DEFAULTS::DESC
A_CLUSTER_ROWID          1
A_CLUSTER_STATE_ROWID          0
A_CLUSTER_STATEID        0
A_CLUSTER_USER 0
A_INDEX_ROWID   "WITH FILLFACTOR = 75"
A_INDEX_STATE_ROWID   "WITH FILLFACTOR = 75"
A_INDEX_STATEID"WITH FILLFACTOR = 75"
A_INDEX_USER     "WITH FILLFACTOR = 75"
A_STORAGE          ""
B_CLUSTER_ROWID          1
B_CLUSTER_USER 0
B_INDEX_ROWID   "WITH FILLFACTOR = 75"
B_INDEX_USER     "WITH FILLFACTOR = 75"
B_STORAGE          ""
B_TEXT_IN_ROW   256
D_CLUSTER_ALL   0
D_CLUSTER_DELETED_AT1
D_CLUSTER_STATE_ROWID          0
D_INDEX_ALL       "WITH FILLFACTOR = 75"
D_INDEX_DELETED_AT     "WITH FILLFACTOR = 75"
D_INDEX_STATE_ROWID   "WITH FILLFACTOR = 75"
D_STORAGE          ""
END

##NETWORK_DEFAULTS::NETWORK
A_CLUSTER_ROWID          1
A_CLUSTER_STATE_ROWID          0
A_CLUSTER_STATEID        0
A_CLUSTER_USER 0
A_INDEX_ROWID   "WITH FILLFACTOR = 75"
A_INDEX_STATE_ROWID   "WITH FILLFACTOR = 75"
A_INDEX_STATEID"WITH FILLFACTOR = 75"
A_INDEX_USER     "WITH FILLFACTOR = 75"
A_STORAGE          ""
B_CLUSTER_ROWID          1
B_CLUSTER_USER 0
B_INDEX_ROWID   "WITH FILLFACTOR = 75"
B_INDEX_USER     "WITH FILLFACTOR = 75"
B_STORAGE          ""
B_TEXT_IN_ROW   256
D_CLUSTER_ALL   0
D_CLUSTER_DELETED_AT0
D_CLUSTER_STATE_ROWID          0
D_INDEX_ALL       "WITH FILLFACTOR = 75"
D_INDEX_DELETED_AT     "WITH FILLFACTOR = 75"
D_INDEX_STATE_ROWID   "WITH FILLFACTOR = 75"
D_STORAGE          ""
END
```

The NETWORK_DEFAULTS keyword

The NETWORK_DEFAULTS keyword contains the default parameters for the ArcGIS network class. If the user does not select a network class composite keyword from ArcCatalog interface, the ArcGIS network is created with the parameters within the NETWORK_DEFAULTS keyword. Whenever a network class composite keyword is selected, its parameters are used to create the feature class, table, and indexes of the network class. If a network composite keyword is missing any parameters, ArcGIS substitutes the parameters of the DEFAULTS keyword rather than the NETWORK_DEFAULTS keyword. So the parameters of the NETWORK_DEFAULTS keyword are only used in the event that no network composite keyword is selected. If a NETWORK_DEFAULTS keyword is not present within a dbtune file that is imported into the DBTUNE table, the following NETWORK_DEFAULTS keyword is created:

```
##NETWORK_DEFAULTS
UI_NETWORK_TEXT          "The network default configuration"
COMMENT "The base system initialization parameters for
NETWORK_DEFAULTS"
A_CLUSTER_ROWID          0
A_CLUSTER_SHAPE          1
A_CLUSTER_STATE_ROWID          0
A_CLUSTER_STATEID        0
A_CLUSTER_USER 0
A_INDEX_ROWID   "WITH FILLFACTOR = 75"
A_INDEX_SHAPE    "WITH FILLFACTOR = 75"
A_INDEX_STATE_ROWID   "WITH FILLFACTOR = 75"
A_INDEX_STATEID"WITH FILLFACTOR = 75"
A_INDEX_USER     "WITH FILLFACTOR = 75"
A_STORAGE          ""
B_CLUSTER_ROWID          0
B_CLUSTER_SHAPE          1
B_CLUSTER_USER 0
B_INDEX_ROWID   "WITH FILLFACTOR = 75"
B_INDEX_SHAPE    "WITH FILLFACTOR = 75"
B_INDEX_USER     "WITH FILLFACTOR = 75"
B_STORAGE          ""
B_TEXT_IN_ROW   256
D_CLUSTER_ALL    0
D_CLUSTER_DELETED_AT 1
D_CLUSTER_STATE_ROWID          0
D_INDEX_ALL       "WITH FILLFACTOR = 75"
D_INDEX_DELETED_AT     "WITH FILLFACTOR = 75"
D_INDEX_STATE_ROWID   "WITH FILLFACTOR = 75"
D_STORAGE          ""
F_CLUSTER_FID     1
F_INDEX_AREA      "WITH FILLFACTOR = 75"
F_INDEX_FID       "WITH FILLFACTOR = 75"
F_INDEX_LEN       "WITH FILLFACTOR = 75"
F_STORAGE          ""
F_TEXT_IN_ROW   256
S_CLUSTER_ALL    1
S_CLUSTER_SP_FID          0
```

```
S_INDEX_ALL       "WITH FILLFACTOR = 75"
S_INDEX_SP_FID    "WITH FILLFACTOR = 75"
S_STORAGE         ""
END
```

Raster table parameters

A raster column added to a business table is actually a foreign key reference to raster data stored in a schema consisting of four tables and five supporting indexes. By default, these parameters are not exposed in the default SDE_dbtune table.

The RAS_CLUSTER_ID parameter holds the SQL Server CREATE TABLE storage configuration of the RASTER table.

The RAS_INDEX_ID parameter holds the SQL Server CREATE TABLE storage configuration of the RASTER table index.

The BND_CLUSTER_COMPOSITE parameter holds the SQL Server CREATE TABLE storage configuration of the BAND table index.

The BND_INDEX_ COMPOSITE parameter holds the SQL Server CREATE INDEX storage configuration of the BAND table's composite column index.

The BND_INDEX_ID parameter holds the SQL Server CREATE INDEX storage configuration of the BAND table's rid column index.

The BND_CLUSTER_ID parameter holds the SQL Server CREATE INDEX storage configuration of the BAND table's rid column index.

The AUX_CLUSTER_COMPOSITE parameter holds the SQL Server CREATE TABLE storage configuration of the AUX table.

The AUX_INDEX_COMPOSITE parameter holds the SQL Server CREATE INDEX storage configuration of the AUX table's index.

The BLK_CLUSTER_COMPOSITE parameter holds the SQL Server CREATE TABLE storage configuration of the BLOCK table.

The BLK_INDEX_COMPOSITE parameter holds the SQL Server CREATE TABLE storage configuration of the BLOCK table's index.

The RAS_STORAGE parameter holds the placement of the sde_ras <rastercolumn_id> table.

The BND_STORAGE parameter holds the placement of the sde_bnd <rastercolumn_id> table.

The BLK_STORAGE parameter holds the placement of the sde_blk <rastercolumn_id> table.

The IMS METADATA keywords

The IMS METADATA keywords control the storage of the IMS Metadata tables. These keywords are a standard part of the dbtune table. If the keywords are not present in the dbtune file when it is imported into the DBTUNE table, ArcSDE applies software defaults. The software defaults have the same settings as the keyword parameters listed in the dbtune.sde table that is shipped with ArcSDE. For more information about installing IMS Metadata and the associated tables and indexes refer to ArcIMS Metadata Server documentation.

The IMS metadata keywords are as follows:

The IMS_METADATADEFAULTS keyword controls the storage of the ims_metadatarelationships business table. ArcSDE creates the following default IMS_METADATADEFAULTS keyword in the DBTUNE table if the keyword is missing from the dbtune file when it is imported

```
IMS_METADATADEFAULTS   B_CLUSTER_ROWID          1
IMS_METADATADEFAULTS   B_CLUSTER_USER          0
IMS_METADATADEFAULTS   B_INDEX_ROWID           WITH FILLFACTOR = 75
IMS_METADATADEFAULTS   B_INDEX_USER            WITH FILLFACTOR = 75
IMS_METADATADEFAULTS   B_STORAGE
IMS_METADATADEFAULTS   UI_TEXT
```

The IMS_METADATAFEATURES keyword controls the storage of the ims_metadatarelationships business table. ArcSDE creates the following default IMS_METADATAFEATURES keyword in the DBTUNE table if the keyword is missing from the dbtune file when it is imported

```
IMS_METADATAFEATURES   B_CLUSTER_ROWID          0
IMS_METADATAFEATURES   B_CLUSTER_SHAPE          1
IMS_METADATAFEATURES   B_CLUSTER_USER          0
IMS_METADATAFEATURES   B_INDEX_ROWID           WITH FILLFACTOR = 75
IMS_METADATAFEATURES   B_INDEX_SHAPE           WITH FILLFACTOR = 75
IMS_METADATAFEATURES    B_INDEX_USER           WITH FILLFACTOR = 75
IMS_METADATAFEATURES   B_STORAGE
IMS_METADATAFEATURES   B_TEXT_IN_ROW           256
IMS_METADATAFEATURES   F_CLUSTER_FID           1
IMS_METADATAFEATURES   F_INDEX_AREA            WITH FILLFACTOR = 75
IMS_METADATAFEATURES   F_INDEX_FID             WITH FILLFACTOR = 75
IMS_METADATAFEATURES   F_INDEX_LEN             WITH FILLFACTOR = 75
IMS_METADATAFEATURES   F_STORAGE
IMS_METADATAFEATURES   F_TEXT_IN_ROW           256
IMS_METADATAFEATURES   S_CLUSTER_ALL           1
IMS_METADATAFEATURES   S_CLUSTER_SP_FID        0
IMS_METADATAFEATURES   S_INDEX_ALL             WITH FILLFACTOR = 75
IMS_METADATAFEATURES   S_INDEX_SP_FID          WITH FILLFACTOR = 75
IMS_METADATAFEATURES   S_STORAGE
```

The IMS_METADATARELATIONSHIPS keyword controls the storage of the ims_metadatarelationships business table. ArcSDE creates the following default IMS_METADATARELATIONSHIPS keyword in the DBTUNE table if the keyword is missing from the dbtune file when it is imported.

```
IMS_METADATARELATIONSHIPS   B_CLUSTER_ROWID        0
IMS_METADATARELATIONSHIPS   B_CLUSTER_USER        1
IMS_METADATARELATIONSHIPS   B_INDEX_ROWID         WITH FILLFACTOR = 75
IMS_METADATARELATIONSHIPS   B_INDEX_USER          WITH FILLFACTOR = 75
IMS_METADATARELATIONSHIPS   B_STORAGE
```

The IMS_METADATARELATIONSHIPS2 keyword controls the storage of the ims_metadatarelationships business table. ArcSDE creates the following default IMS_METADATARELATIONSHIPS2 keyword in the DBTUNE table if the keyword is missing from the dbtune file when it is imported.

```
IMS_METADATARELATIONSHIPS2   B_CLUSTER_USER        0
IMS_METADATARELATIONSHIPS2   B_INDEX_USER          WITH FILLFACTOR = 75
```

The IMS_METADATATHUMBNAILS keyword controls the storage of the ims_metadatathumbnails business table. ArcSDE creates the following default IMS_METADATATHUMBNAILS keyword in the DBTUNE table if the keyword is missing from the dbtune file when it is imported.

```
IMS_METADATATHUMBNAILS   B_CLUSTER_ROWID          0
IMS_METADATATHUMBNAILS   B_CLUSTER_USER           1
IMS_METADATATHUMBNAILS   B_INDEX_ROWID            WITH FILLFACTOR = 75
IMS_METADATATHUMBNAILS   B_INDEX_USER             WITH FILLFACTOR = 75
IMS_METADATATHUMBNAILS   B_STORAGE
IMS_METADATATHUMBNAILS   B_TEXT_IN_ROW           256
```

The IMS_METADATAVALUES keyword controls the storage of the ims_metadatavalues business table. ArcSDE creates the following default IMS_METADATAVALUES keyword in the DBTUNE table if the keyword is missing from the dbtune file when it is imported.

```
IMS_METADATAVALUES   B_CLUSTER_ROWID              0
IMS_METADATAVALUES   B_CLUSTER_USER               1
IMS_METADATAVALUES   B_INDEX_ROWID                WITH FILLFACTOR = 75
IMS_METADATAVALUES   B_INDEX_USER                 WITH FILLFACTOR = 75
IMS_METADATAVALUES   B_STORAGE
```

The IMS_METADATAWORDINDEX keyword controls the storage of the ims_metadatawordindex business table. ArcSDE creates the following default IMS_METADATAWORDINDEX keyword in the DBTUNE table if the keyword is missing from the dbtune file when it is imported.

```
IMS_METADATAWORDINDEX   B_CLUSTER_ROWID           0
IMS_METADATAWORDINDEX   B_CLUSTER_USER            1
IMS_METADATAWORDINDEX   B_INDEX_ROWID             WITH FILLFACTOR = 75
IMS_METADATAWORDINDEX   B_INDEX_USER              WITH FILLFACTOR = 75
IMS_METADATAWORDINDEX   B_STORAGE
```

The IMS_METADATAWORDINDEX2 keyword controls the storage of the ims_metadatawordindex business table. ArcSDE creates the following default IMS_METADATAWORDINDEX2 keyword in the DBTUNE table if the keyword is missing from the dbtune file when it is imported.

```
IMS_METADATAWORDINDEX2   B_CLUSTER_USER           0
IMS_METADATAWORDINDEX2   B_INDEX_USER             WITH FILLFACTOR = 75
```

The IMS_METADATAWORDS keyword controls the storage of the ims_metadatawords business table. ArcSDE creates the following default IMS_METADATAWORDS keyword in the DBTUNE table if the keyword is missing from the dbtune file when it is imported.

```
IMS_METADATAWORDS   B_CLUSTER_ROWID               1
IMS_METADATAWORDS   B_CLUSTER_USER                0
IMS_METADATAWORDS   B_INDEX_ROWID                 WITH FILLFACTOR = 75
IMS_METADATAWORDS   B_INDEX_USER                  WITH FILLFACTOR = 75
IMS_METADATAWORDS   B_STORAGE
```

Text in row parameters

SQL Server 2000 supports the ability to store image values in a data row. Unless the text in row option is specified, the images are stored outside the data row. Users can set a text in row option for tables containing image columns. They can also specify a text in row option limit, from 24 through 7,000 bytes. With this option image strings are stored directly in the data row if the length of the string is shorter than the specified limit and there is enough space available in the data row to hold the string. This can eliminate a page access when reading or writing the string, which speeds processing. The text in row option is set to the SQL Server default of 256 bytes. This has been chosen to allow maximum performance by allowing small images to be stored directly in the row but not large enough that the number of rows per page is reduced. This value should not be set below 72 bytes or above 7,000 bytes.

Fill factor parameters

The "WITH FILLFACTOR = 75" parameter specifies how full each page in the leaf level of an index should be. SQL Server uses a default value of 0, which means that the leaf pages of an index are almost full but that the nonleaf pages have room for at least two more rows. If the fillfactor is 100, all pages are completely full. With a fill factor of 75, each clustered index is 75 percent full.

Editing the storage parameters

To edit the storage parameters, the sdedbtune administration command allows you to export the DBTUNE table to a file located in the %SDEHOME%\etc folder on Windows servers. It is an ArcSDE configuration file that contains SQL Server table and index creation parameters. These parameters allow the ArcSDE service to communicate with SQL Server such things as where a filegroup, table, or index will be created, as well as other parameters that can be set on either the CREATE TABLE or CREATE INDEX statement.

CHAPTER 4

Managing tables, feature classes, rasters, and views

A fundamental part of any database is creating and loading the tables. Tables with spatial columns are called standalone feature classes. Attribute only (nonspatial) tables are also an important part of any database. This chapter will describe the table and feature creation and loading process and provide some examples demonstrating how to populate your ArcSDE database.

Setting up the dbtune table

Previous versions of ArcSDE and Spatial Database Engine™ (SDE®) used the dbtune.sde file to identify data spaces for your data. Loading your data onto specified data spaces allows you to distribute I/O in multiuser configurations. The basic idea is to spread your data and indexes across many disks so that disk spindles can find it. Placing all data in one location on a single disk will create a "hotspot." A hotspot is a location on a disk of frequently accessed data by multiple readers or writers.

ArcSDE 8.1 pushes the dbtune file into a database table. This allows for all types of connections, three-tier or two-tier, to access the information. Dbtune tables are composed of keywords, parameters, and configuration parameters. Keywords identify specific loading parameters. Parameters represent ArcSDE objects such as tables and indexes. Configuration parameters specify how something should be loaded.

For further information on the dbtune table, see Chapter 3, 'Configuring DBTUNE storage parameters'.

Data creation

There are numerous applications that can create and load data within an ArcSDE SQL Server database. These include:

1. ArcSDE administration commands located in the bin directory of SDEHOME:
    - sdelayer manages feature classes including creating a schema
    - sdetable manages tables including creating a schema
    - sdeimport takes an existing sdeexport file and loads the data into the database
    - shp2sde loads an ESRI® shapefile into the database
    - cov2sde loads a coverage, Map LIBRARIAN™ layer, or ArcStorm™ layer into the database

- tbl2sde loads an attribute-only dBASE or INFO file into the database
- sdegroup—a specialty feature class creation command that provides a way to "group" or tile features from an existing feature class into another feature class that displays background or reference data. The generated feature classes are used for rapid display of a large amount of geometry data. The attribute information is not retained, and spatial searches cannot be performed on these feature classes.

These are all run from the operating system prompt. Command references for these tools are in the ArcSDE developer help.

Other applications include:

ArcGIS Desktop—use ArcCatalog or ArcToolbox™ to manage and populate your database.

ArcInfo Workstation—use the Defined Layer interface to create and populate the database.

ArcView GIS 3.2—use the Database Access extension.

MapObjects®—custom COM applications can be built to create and populate databases.

ArcSDE CAD Client extension—for AutoCAD and MicroStation users.

Other third party applications built with either C or Java APIs.

This document focuses primarily on the ArcSDE administration tools but does provide some ArcGIS Desktop examples as well. In general, most people prefer an easy-to-use graphic user interface like the one found in ArcGIS 8.1 Desktop. For details on how to use ArcCatalog or ArcToolbox (another desktop data loading tool), please refer to the ArcGIS 8.1 books:

Know your data

Before you begin loading your data, get to know it. SQL Server 2000 has approximately 174 reserved words and 135 future keywords, words that may be listed as reserved in future releases. The use of reserved words for table and column names is not supported. See the SQL Server Books Online for more information.

The use of nonstandard identifiers is also unsupported. Such an identifier could be naming a table "3d." You cannot begin an ArcSDE tablename with a number. Nor can you use pound signs or symbols. The maximum length of an ArcSDE table is 32 characters.

Watch the length and naming of your columns as well. Do not employ column names with spaces in them. Long column names can also cause problems if they exceed the SQL Server accepted range. Finally, be aware of your data ranges, especially with date data types. SQL Server cannot accept datetime values preceding January 1, 1753. If you attempt to load a table into SQL Server through ArcSDE with values preceding this range, the load will fail.

Creating and populating a feature class

The general process involved with creating and loading a feature class is:

1. Create the business table.
2. Add the feature class to the ArcSDE system tables.

3. Switch the feature class to load_only_io mode (optional step to improve bulk data loading performance. It is OK to leave the feature class in normal_io mode to load data.).
4. Insert the records (load data).
5. Switch the feature class to normal_io mode (builds the indexes).
6. Version the data (optional).
7. Grant permissions on the data (optional).

In the following sections, this process is discussed in more detail and illustrated with some examples of ArcSDE administration commands usage and ArcInfo data-loading utilities through the ArcCatalog and ArcToolbox interfaces.

Creating a feature class "from scratch"

There are two basic ways to create a feature class. You can create a feature class from scratch (generally regarded as more effort), or you can create a feature class from existing data such as a coverage or ESRI shapefile. Both methods are reviewed below with the "from scratch" method being first.

Creating a business table

You may create a business table with either the SQL CREATE TABLE statement or with the ArcSDE sdetable command. The sdetable command allows you to include a dbtune configuration keyword which contains the storage parameters of the table.

Although the table may be up to 256 columns, ArcSDE requires that only one of those columns be defined as a spatial column.

In this example, the sdetable command is used to create the 'roads' business table.

```
sdetable -o create -t roads -d 'road_id integer, name string(32), shape
integer' -k roads -u beetle -p bug
```

The table is created using the dbtune configuration keyword (-k) 'roads' by user beetle.

The same table could be created with a SQL CREATE TABLE statement using the SQL Server Query Analyzer.

```
create table roads
(road_id     integer,
name         varchar(32),
shape        integer)
on beetle_data
```

At this point you have created a table in the database. ArcSDE does not yet recognize it as a feature class. The next step is to add a feature class.

Adding a feature class

After creating a business table, you must add an entry for it in the ArcSDE system tables before the ArcSDE server can reference it. Use the sdelayer command with the "-o add" operation to add the new feature class. This will add a spatial column, update the ArcSDE SDE_layers and SDE_geometry_columns tables, and add the required system table entries.

In the following example, the roads feature class is added to the ArcSDE database. Note that to add the feature class, the roads table name and the shape spatial column are combined to form a unique feature class reference. To understand the –e, –g, and –x options, refer to the sdelayer command reference in the ArcSDE developer help.

```
sdelayer -o add -l roads,shape -e l+ -g 256,0,0 -x 0,0,100 -u beetle -p
bug -k roads
```

If the spatial column of the feature class is stored in either LONG RAW or BLOB compressed binary format, the feature and spatial index tables are created at this time. The feature class tables and indexes are stored according to the storage parameters of the **roads** configuration keywords in the DBTUNE table. Upon successful completion of the previous sdetable command—to create a table—and the sdelayer command—to add the feature class to the ArcSDE system tables—you have an empty feature class in normal_io mode.

Switching to load-only mode

Switching the feature class to load-only mode drops the spatial index and makes the feature class unavailable to ArcSDE clients. Bulk loading data into the feature class in this state is much faster due to the elimination of index maintenance. Use the sdelayer command to switch the feature class to load-only mode by specifying the "-o load_only_io" operation.

```
sdelayer -o load_only_io -l roads,shape -u beetle -p bug
```

Note: A feature class, registered as multiversioned, cannot be placed in the load-only I/O mode. However, the grid size can be altered with the -o alter operation. The alter operation will apply an exclusive lock on the feature class, preventing all modifications by ArcInfo until the operation is complete.

Inserting records into the feature class

Once the empty feature class exists, the next step is to populate it with data. There are several ways to insert data into a feature class, but probably the easiest method is to convert an existing shapefile or coverage or import a previously exported ArcSDE sdeexport file directly into the feature class. A more "from scratch" method would be to add the data with an editor such as ArcMap.

In this first example, shp2sde is used with the init operation. The init operation is used on newly created feature classes or can be used on feature classes when you want to "overwrite" data that's already there. Be careful when using the init operation on feature classes that already have data loaded in them. Here, the shapefile, "rdshp", will be loaded into the feature class, "roads". Note that the name of the spatial column (shape in this case) is included in the feature class (-l) argument.

```
shp2sde -o init -l roads,shape -f rdshp -u beetle -p bug
```

Similarly, we can also use the cov2sde command:

```
cov2sde -o init -l roads,shape -f rdcov -u beetle -p bug
```

Switching the table to normal_io mode

After data has been loaded into the feature class, you must switch the feature class to normal_io mode to re-create all indexes and make the feature class available to clients. For example:

```
sdelayer -o normal_io -l roads,shape -u beetle -p bug
```

Now your feature class is ready for use by ArcSDE client applications.

Versioning your data

Optionally, you may enable your feature class as multiversioned. Versioning is a process that allows for multiple representations of your data without requiring duplication or copies of the data. ArcMap requires data to be multiversion enabled in order to edit it. For further information on versioning data, refer to the *Building a Geodatabase* book.

In this example, the feature class called 'states' will be registered as multiversioned using the sdetable alter_reg operation.

```
sdelayer -o alter_reg -t states -c ver_id -C SDE -V multi -k GEOMETRY_TYPE
```

Granting permissions on the data

Once you have the data loaded, it is often necessary for other users to have access to the data for update, query, insert, or delete operations. Initially, only the user who has created the business table has access to it. In order to make the data available to others, the owner of the data must grant permissions to other users. The owner can use the sdelayer command to grant permissions. Privileges can be granted to either another user or to a role.

In this example, a user called "beetle" gives a user called "spider" SELECT privileges on a feature class called 'states'.

```
sdelayer -o grant -l states,feature -U spider -A SELECT -u beetle -p bug
```

The full list of -A keywords are:

SELECT. The user may query the selected object(s) data.

DELETE. The user may delete the selected object(s) data.

UPDATE. The user may modify the selected object(s) data.

INSERT. The user may add new data to the selected object(s) data.

If you include the -I grant option, you also grant the recipient the privilege of granting other users and roles the same privilege.

Creating and loading feature classes from existing data (recommended method)

This next section reviews how to create feature classes from existing data. This method is simpler since the creation and load process is completed at once.

Creating new feature classes and loading data all in one step

Each of the ArcSDE administration commands, shp2sde,cov2sde, and sdeimport, includes an "-o create" operation, which allows you to create a new feature class within the ArcSDE database. The create operation does all of the following:

- Creates the business table using the input data as the template for the schema
- Adds the feature class to the ArcSDE system tables
- Puts the feature class into load-only mode
- Inserts data into the feature class
- When all the records are inserted, puts the feature class into normal_io mode

shp2sde

The shp2sde command converts shapefiles into ArcSDE feature classes. The spatial column definition is read directly from the shapefile. You can use the shpinfo command to display the shapefile column definitions. In this example, the –k roadsys switch identifies a dbtune parameter.

```
shp2sde -o create -f rdshp -l roads,shape -k roadsys -u beetle -p bug
```

cov2sde

The cov2sde command converts ArcInfo coverages, ArcInfo Librarian™ library feature classes and ArcStorm library feature classes into ArcSDE feature classes. The create operation derives the spatial column definition from the coverage's feature attribute table.

Use the ArcInfo describe command to display the ArcInfo data source column definitions.

In this example, an ArcStorm library, "roadlib", is converted into the feature class, "roads".

```
cov2sde -o create -l roads,shape -f roadlib,arcstorm -g 256,0,0 -x 0,0,100
-e l+ -u beetle -p bug
```

sdeimport

The sdeimport command converts ArcSDE export files into ArcSDE feature classes. In this example, the roadexp ArcSDE export file is converted into the feature class 'roads'.

```
sdeimport -o create -l roads,shape -f roadexp -u beetle -p bug
```

After using these commands to create and load data, you may optionally need to enable multiversioning on the feature class and modify the permissions of the feature class.

Appending data to an existing feature class

A common requirement for data management is to be able to append data to existing feature classes. All the data loading commands described thus far have a –o append operation for appending data. A feature class must exist prior to using the append operation. If the feature class is multiversioned, it must be in an "open" state. It is also advisable to change the feature class to load-only I/O mode and pause the spatial indexing operations before loading the data to improve the data loading performance. The spatial indexes will be re-created when the feature class is put back into normal I/O mode. Because the feature class has already been defined, the metadata already exists and is not altered by the append operation.

In the shp2sde example below, a previously created 'roads' feature class gets appended features from a shapefile, "rdshp2." All existing features, loaded from the rdshp shapefile, will remain intact, and the feature class will simply be updated with the new features from the rdshp2 shapefile.

```
sdelayer -o load_only_io -l roads,shape -u beetle -p bug
shp2sde -o append -f rdshp2 -l roads,shape -u beetle -p bug
sdelayer -o normal_io -l roads,shape -u beetle -p bug
sdetable -o update_dbms_stats -t roads -u beetle -p bug
```

Note the last command in the sequence. The update_dbms_stats operation on sdetable updates the table and index statistics.

Creating and populating raster tables

Raster columns are created from the ArcGIS Desktop using ArcCatalog or ArcMap. To create a raster column, you will first need to convert the image file into a format acceptable to ArcSDE . Then after the image has been converted to the ESRI raster file format, you can convert it into a raster column.

For more information on creating raster columns using either ArcCatalog or ArcToolbox, refer to *Building a Geodatabase*. ArcCatalog and ArcToolbox have a raster to Geodatabase loader for loading rasters into ArcSDE.

Consult Chapter 3 'Configuring DBTUNE storage parameters,' for dbtune specific parameters for loading rasters. ArcSDE 8.1 for SQL Server 2000 supports a text_in_row option for raster columns: blk_text_in_row.

Creating a Raster Image Catalog

An Image Catalog allows you to group many images by simply listing them in a table. ArcGIS clients such as ArcCatalog or ArcMap display the images as a group by reading the entries in the table. The table must contain five columns: Image, xmin, ymin, xmax, and ymax. The Image column contains a fully qualified image name, and the remaining four describe the extent of each image. The table does not have to be registered with either ArcSDE or the Geodatabase and does not have to be multiversioned. Here's an example:

Image	xmin	ymin	xmax	ymax
sde.DBO.A106	322169.2	4094888	411902.3	4151957
sde.DBO.E106	323452.5	4150579	412586.2	4207658
sde.DBO.A107	233002.1	4096472	323537.9	4154628
sde.DBO.E107	234912.9	4151847	324666.3	4210094
sde.DBO.A108	676712.7	4090675	770814.6	4154551
sde.DBO.E108	675530.7	4146368	769195.7	4210173
sde.DBO.A106_8	324599.5	4206098	413235.3	4263136
sde.DBO.E106_8	325340.9	4261470	413692.2	4318609
sde.DBO.A107_8	236598.8	4207336	325702	4265705
sde.DBO.E107_8	239302.9	4262352	327095.4	4321168
sde.DBO.A105_9	413273.8	4316886	500802.2	4372878

If you have less than nine images, then all the images will display. If you have more, your images will not display until you begin to zoom in.

Exporting data

As with importing data, there are also client applications that export data from ArcSDE as well. With ArcSDE , the following command line tools exist:

sdeexport—creates an ArcSDE export file to easily move feature class data between Oracle instances and to other supported RDBMSs.

sde2shp—creates an ESRI shapefile from an ArcSDE feature class

sde2cov—creates a coverage from an ArcSDE feature class

sde2tbl—creates a dBASE or INFO file from an RDBMS table

Schema modification

There will be occasions where it is necessary to modify the schema of some tables. You may need to add or remove columns from a table. ArcCatalog offers an easy-to-use tool for this and other schema operations such as modifying the spatial index (grids) and adding and dropping column indexes. To drop a column with ArcSDE command tools, create an sde view of the layer without the column, export it, then reimport it. To add a column using the sde admin tools, create a table from your business table using a select into statement, then add your new column.

Using ArcInfo software's ArcCatalog and ArcToolbox applications

So far the discussion has focused on ArcSDE command line tools that create feature class schemas and load data into them. While robust, these commands can be daunting for the first-time user. In addition, if you are using ArcInfo Desktop, you may have to use ArcCatalog to create feature datasets and feature classes within those feature datasets to use specific ArcInfo 8 Desktop functionality. For that reason, we provide a glimpse of how to use ArcToolbox and ArcCatalog to load data. Please refer to the ArcInfo Digital Books on ArcCatalog, ArcToolbox, and the Geodatabase for a full discussion of these tools.

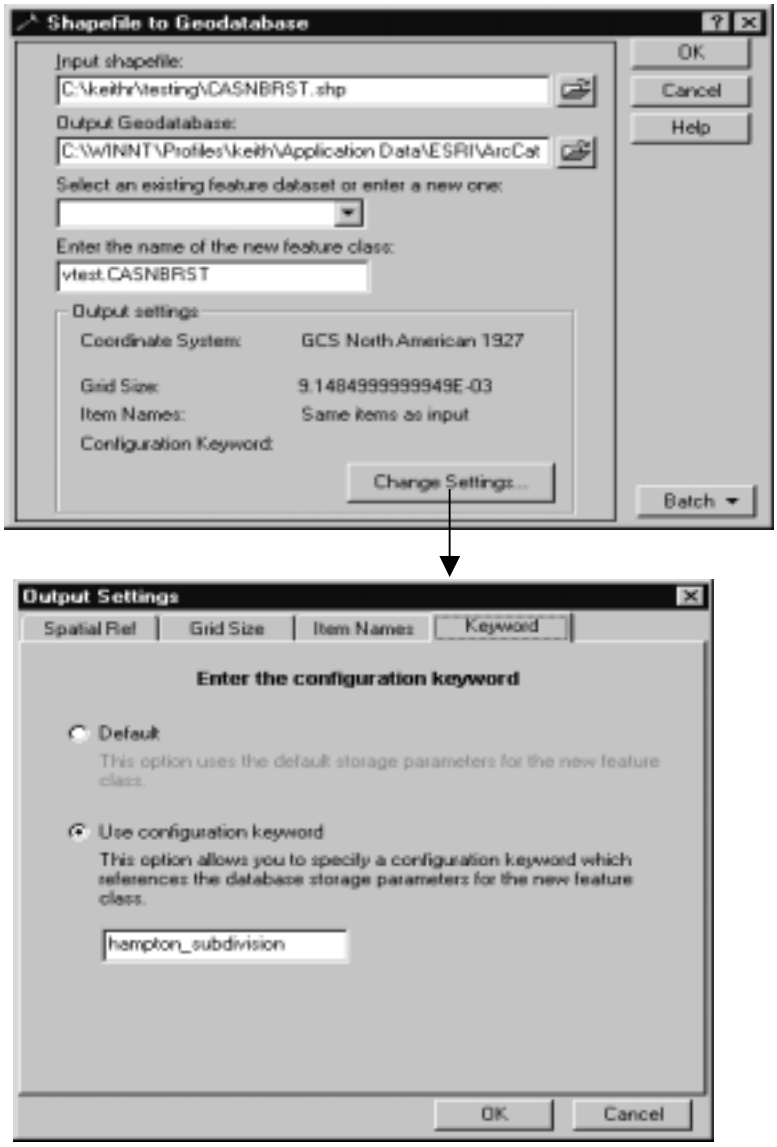
Loading data

You can convert ESRI shapefiles, coverages, Map LIBRARIAN layers, and ArcStorm layers into Geodatabase feature classes with the ArcInfo ArcToolbox and ArcCatalog applications. ArcToolbox provides a number of tools that enable you to convert data from one format to another.

ArcToolbox operations, such as the ArcSDE administration commands shp2sde, cov2sde, and sdeimport, accept configuration keywords. By using a configuration keyword from the dbtune table, you can employ customized data loading options such as fill factors of indexes, text_in_row sizes, or filegroups for storing tables or indexes.

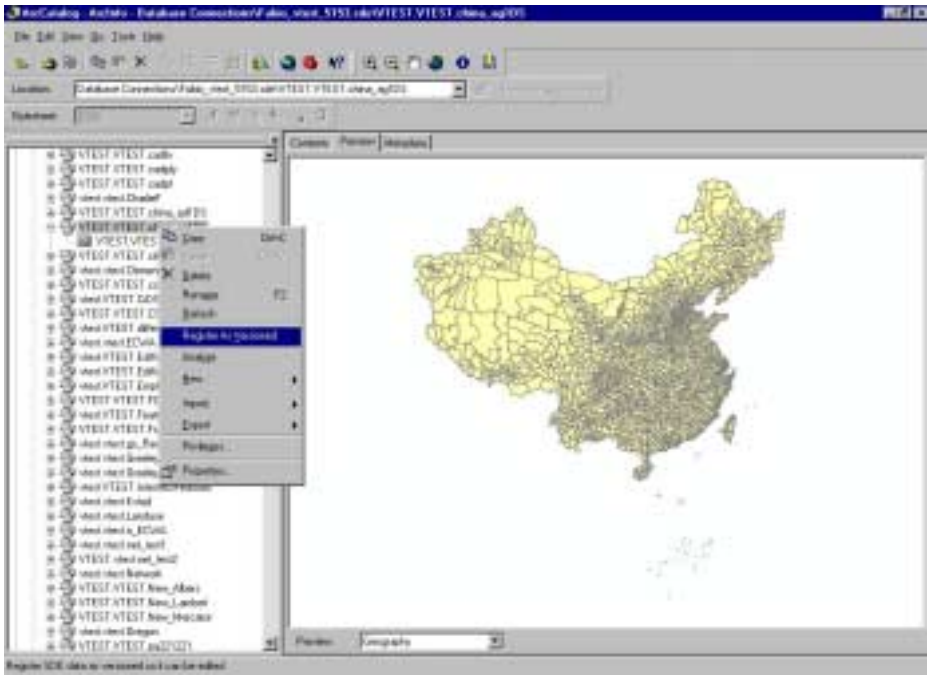
In the ArcToolbox Shapefile to Geodatabase wizard, we can see that a configuration keyword has been specified for the loading of the hampton_streets shapefile into the Geodatabase. Since the Geodatabase is maintained by an ArcSDE 8.1 service operating on a SQL Server database, we can store the resulting feature class using the parameters identified by the Hampton_subdivision dbtune keyword.

The shapefile CASNBRST.shp is converted to a feature class vtest.CASNBRST using ArcToolbox.



Versioning your data

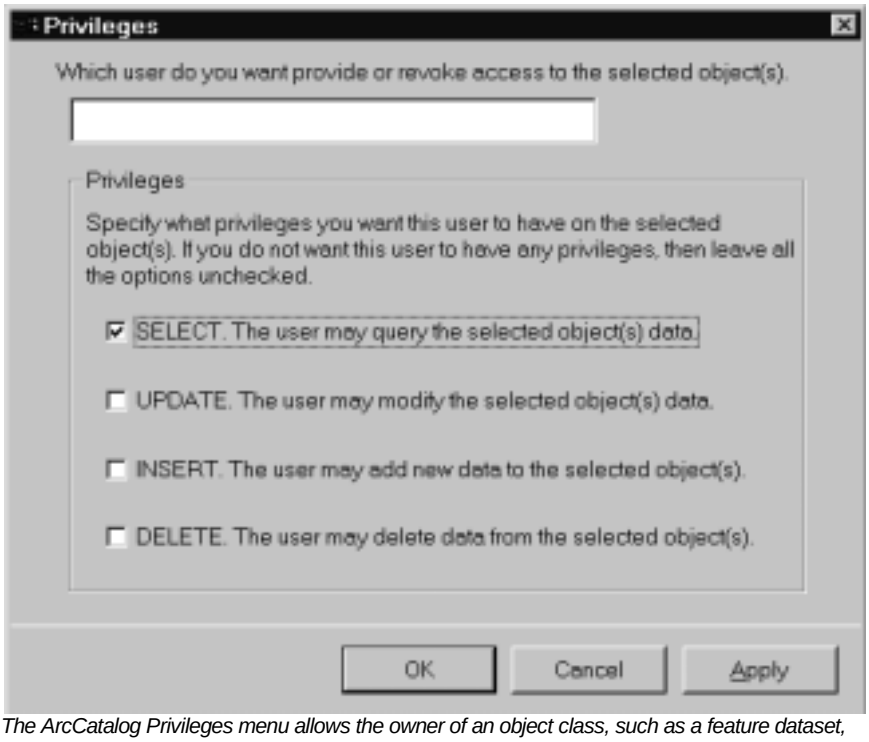
ArcCatalog also provides a means for registering data as versioned. Simply right-click the feature class to be registered as versioned and select the Register As Versioned context menu item. You can register standalone feature classes or all feature classes of a feature dataset.



A feature class is registered as multiversioned from within ArcCatalog.

Permissions

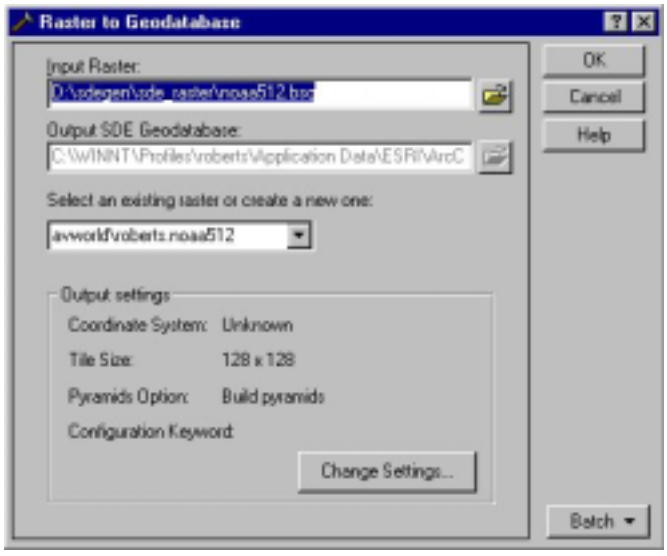
Using ArcCatalog, right-click on the data object class and click on the Privileges context menu. From the Privileges context menu assign privileges specifying the username and the privilege you wish to grant to or revoke from a particular user.



The ArcCatalog Privileges menu allows the owner of an object class, such as a feature dataset, feature class, or table, to assign privileges to other users or roles.

Loading a raster into ArcSDE with ArcCatalog

Using ArcCatalog, right-click on the database connection, point to Import, and click on Raster to Geodatabase. Navigate to the raster file to import. Click Change Settings if you want to change the coordinate reference system, tile size, pyramids option, or configuration keyword. Click OK to import the raster file into the SQL Server database.



Defining ArcSDE views

You can create spatial views of your data much the same way you create tabular views. Views are stored queries or virtual tables. The select statement is stored in the database instead of the table. SQL Server 2000 allows users to create unique clustered indexes on views and partition views across multiple servers.

Indexed views are stored in the database in the same manner as a table, while nonindexed views store only the SQL statements composing them.

With ArcSDE, it is possible to define views between two feature classes or between a feature class and a table or create more complex views containing subqueries or that span databases. ArcSDE views are created with the sdetable –o create_view command. When you create an ArcSDE view, three views are created, one of the business table, feature table, and spatial index.

Creating an ArcSDE view

Use the sdetable –o create_view command to create a view and the sdetable –o delete command to remove one. The syntax of sdetable –o create is:

```
sdetable -o create_view
    -T <view_name>
    -t <table1,table2...tablen>

    -c <table_col1,table_col2...table_coln>

    [-a <view_col1,view_col2...view_coln>]
    [-w “where_clause">]
    [-i <service>]
    [-s <server_name>]
    [-D <database>]
    -u <DB_User_name> [-p <DB_User_password>] [-N] [-q]
```

The syntax of sdetable –o delete is:

```
    sdetable -o delete       -t <table>
            [-i <service>] [-s <server_name>] [-D <database>]
          -u <DB_User_name> [-p <DB_User_password>] [-N] [-q]
```

You must list the columns (-c) you want in the view as well as a where clause (–w) within the sdetable command. Be sure to include the business table’s spatial column to make the view spatial. You must have ‘create view’ permissions to execute this command.

This example creates a view of all U.S. counties with a 1990 population per square mile greater than 50. The –s option specifies a server.

```
sdetable -o create_view -T CountyPopView -t us_counties -c
shape,name,state_name -w "pop90_sqmi > 50" -s bigman -u beetle -p bug
```

This example uses a subquery to select the state whose capital city has the largest male population.

```
sdetable -o create_view -T ManView -t us_states -c
us_states.shape,us_states.state_name --w "us_states.state_name = (select
us_capitals.state_name from us_capitals where p_male = (select max(p_male)
from us_capitals))" -u beetle -p bug -s bigman
```

Cross database views

It is possible to create views between tables and sde feature classes that do not reside in the same database. When you do this, you must correctly qualify the tables involved in your queries and use the –D (database) switch to identify which database contains the data on which to create a view.

This is illustrated in the following example, where there is a feature class in database "a." and a standalone, unregistered table in database "b.", and you want to make an sde view from these two entities in database "a."

1. Fully qualify the standalone table and its columns in your sdetable –o create_view statement. Or you can use table aliases. You must enclose table names in quotes if you do.

2. Qualify only to owner.tablename for the sde feature class.

3. Use the -D <database> switch for the "a" database.

4. Run the command as the owner of the sde layer.

The SQL statements to execute this would be:

```
CREATE VIEW crossDBView
AS SELECT  us_statesview.feature, us_statesview.state_name,
vtest.vtest.bob.sub_region
FROM  us_statesview, vtest.vtest.bob
WHERE us_statesview.state_name=vtest.vtest.bob.state_name
```

So your command line would look like this:

```
C:\>sdetable -o create_view -T bigbob -t
vtest.us_statesview,vtest.vtest.bob
-c
us_statesview.feature,us_statesview.state_name,vtest.vtest.bob.sub_region
-w us_statesview.state_name=vtest.vtest.bob.state_name -i sql82k -D sde -u
vtest -p go

Explanation:
-T bigbob = name of the view

-t vtest.us_statesview,vtest.vtest.bob = us_statesview resides in the
'sde'
database, bob in 'vtest' database

-c
us_statesview.feature,us_statesview.state_name,vtest.vtest.bob.sub_region
= column list fully qualified for stand alone table.

-w us_statesview.state_name=vtest.vtest.bob.state_name = where clause,
fully
qualified for stand alone table.

-D sde = database context
```

Example using table aliases:

```
C:\sde81\etc>sdetable -o create_view -T crossdbview -t
vtest.us_statesview,"vtest.vtest.bob x" -c
us_statesview.feature,us_statesview.state_name,x.sub_region -w
"us_statesview.state_name=x.state_name and x.pop1996<1000000" -i sql82k -D
sde
-u vtest -p go
```

Creating views across linked servers

This is similar to cross database views with the following exceptions. You are forced to use aliases in your column lists because SQL Server will not allow four- or one-part qualified names in a column list. You must also define a linked server first and make

sure your DTC service is running. See the SQL Server Books Online for the exact usage of 'sp_addlinkedserver.'

1. Define a linked server in SQL Server Query Analyzer. This example was executed by a sysadmin user in SQL Server 2000.

```
sp_addlinkedserver
@server='foo',
@srvproduct='',
@provider='SQLOLEDB',
@datasrc='fabio\sql2k',
@location='',
@provstr = 'DRIVER={SQL
Server};SERVER=fabio\sql2k;UID=vtest;PWD=go;database=vtest',
@catalog='vtest'
```

2. Start the Distributed Transaction Coordinator (MSDTC). Open the Services panel and scroll down to the MSDTC service. If it is not running, start it.

3. Run the sdetable command with aliased table names:

```
C:\sde81\etc>sdetable -o create_view -T viewRemoteData -t
vtest.us_statesview,"foo.vtest.vtest.bob x" -c us_statesview.feature,
us_statesview.state_name,x.sub_region -w "us_statesview.state_name =
x.state_name and x.pop1996<1000000" -i sql82k -D sde -u vtest -p go
```

The SQL statements to execute this would be:

```
CREATE VIEW viewRemoteData AS SELECT
us_statesview.feature, us_statesview.state_name, x.sub_region
FROM  vtest.us_statesview, foo.vtest.vtest.bob x
WHERE us_statesview.state_name=x.state_name
```

Creating indexed views (SQL Server 2000)

Indexed views are a new feature of SQL Server 2000 and are advantageous when your view definition employs a complex join. They are a good choice on static data as they are schemabound to their source. Also, they are stored in the same way as a regular table with a clustered index. Before you try this, make sure you are familiar with the SQL Server Books Online documentation covering indexed views.

Creating an indexed view in SQL Server 2000 requires extra work and has many conditions. These conditions can be summarized as follows:

- The view must be created with Schemabinding.
- The view must be created with QUOTED_IDENTIFIER on.
- The view cannot reference other views.
- Tables referenced in views must be two-part.
- You cannot reference all columns of a table with *.

There are additional rules and conditions. Refer to the SQL Server Books Online under the topic 'Creating an Indexed View' for more information.

As ArcSDE does not create schema bound views by default, follow this process:

1. Create an ArcSDE view with sdetable –o create_view.
2. Open the SQL Server Query Analyzer and drop the business table view with drop view <view name>. Before you do this, make sure you copy the view definition.

3. Re-create the business table view with the create view statement, but make sure that you use the WITH SCHEMABINDING option and qualify your view name and table name with the owner.<table or view name> convention.
4. Create a unique clustered index on the view with the create unique clustered index statement.

This is an example that creates an indexed view. This example was done in SQL Server 2000 using a SQL Server authenticated user.

1. Create the view with sdetable –o create_view:

```
C:\sde81\etc>sdetable -o create_view -T vwWithIndx -t cagis_cent -c
parcelid,object__id,x_coord,y_coord,book,page,parcel,shp -w
"parcel>0393" -u vtest -p go –i sql82k -s fabio -D vtest
```

2. Query Analyzer—copy the view statement and then drop the view of the business table.

```
drop view vwWithIndx
```

3. Re-create the view. Make sure you qualify the view and table names with the owner name and use the WITH SCHEMABINDING statement.

```
CREATE VIEW vtest.vwWithIndx
WITH SCHEMABINDING
AS SELECT  parcelid, object__id, x_coord, y_coord, book, page, parcel,
shp
FROM  vtest.cagis_cent
WHERE parcel>0393
```

4. Create a unique clustered index on this view:

```
create unique clustered index shpIdxView on vtest.vwWithIndx (shp)
```

Visualizing data with sdegroup

If you have very large datasets, you'll want to discourage your users from drawing these at full extent. There is no use in drawing a million polygons at full extent, as you can see no detail. However, your users may want to visualize what the overall shape or extent of their data looks like. The command sdegroup takes the existing shapes from a feature class and groups them into multipart shapes. While this may not be useful to analysis, it does allow rapid drawing of large amounts of data because it reduces the number of records. Polygons are generalized into polylines so as to not destroy topological relationships.

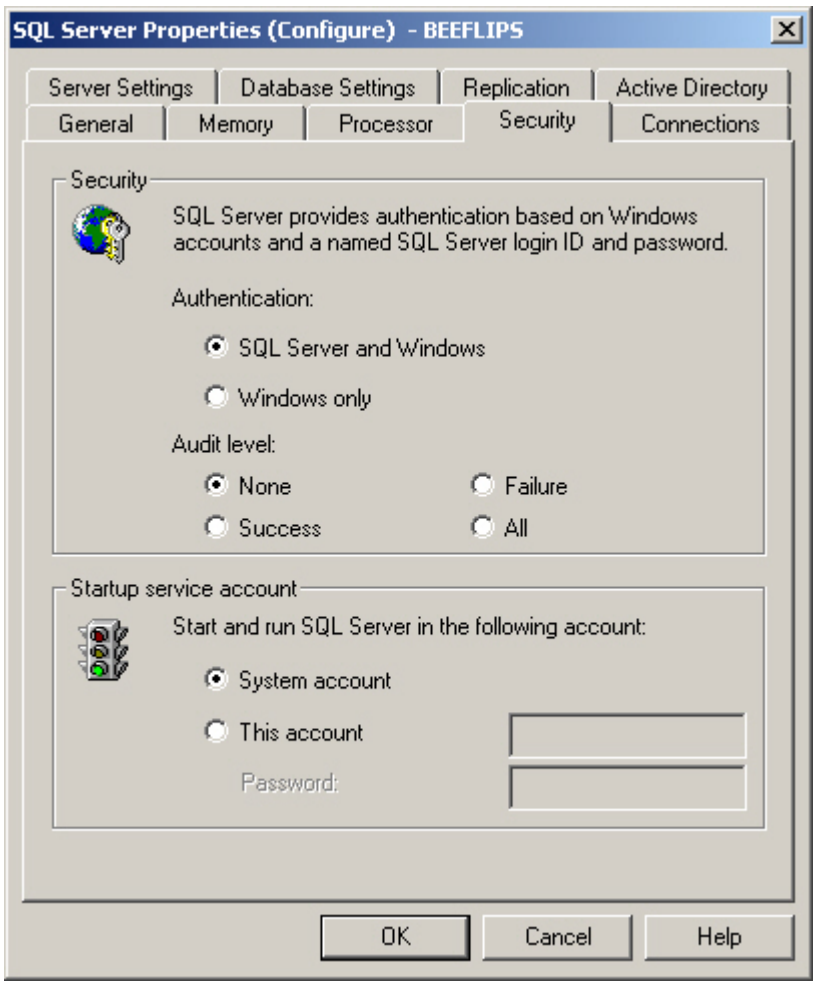
CHAPTER 5

Connecting to SQL Server

You can operate ArcSDE under three different configurations: three tiered with your clients connecting to a middle tier (giomgr) service; two tiered with your clients making direct connections to the database; and mixed, where your server accepts both direct and three-tier connections. This chapter covers how to make your initial connection to ArcSDE under a three-tiered configuration, how to connect in a two-tier configuration, and how to operate only in a two-tier environment without installing ArcSDE.

Making your first connection

VERY IMPORTANT NOTE: By default, SQL Server 2000 uses Windows authentication. In order to create the sde database and user, you must change this setting to include SQL Server authentication by right-clicking the server name in the Enterprise Manager and selecting Properties Security tab. Select, under Authentication, SQL Server and Windows. Then click OK to accept this setting. This is illustrated in the following panel. This procedure should be performed before installation of the ArcSDE software or creation of the sde database and sde user. After confirming that this option is correct, proceed with this manual.


SQL Server Properties (Configure) - BEEFLIPS
Server Settings
Database Settings
Replication
Active Directory
General
Memory
Processor
Security
Connections
Security
SQL Server provides authentication based on Windows accounts and a named SQL Server login ID and password.
Authentication:
SQL Server and Windows
Windows only
Audit level:
None
Failure
Success
All
Startup service account
Start and run SQL Server in the following account:
System account
This account
Password:
OK
Cancel
Help

If you have your ArcSDE service started, then you have already successfully made a connection to the data store (as the sde user). To successfully connect to an ArcSDE data store as a non-sde user, these conditions must be true:

1. The connecting user has been added as a user to the sde database. The connecting user cannot use a reserved word for a name. Examples of reserved words are max, min, national, and select. Refer to the SQL Server Books Online, under "reserved keywords," for more information.

2. The connecting user has at least create table permissions in the sde database.

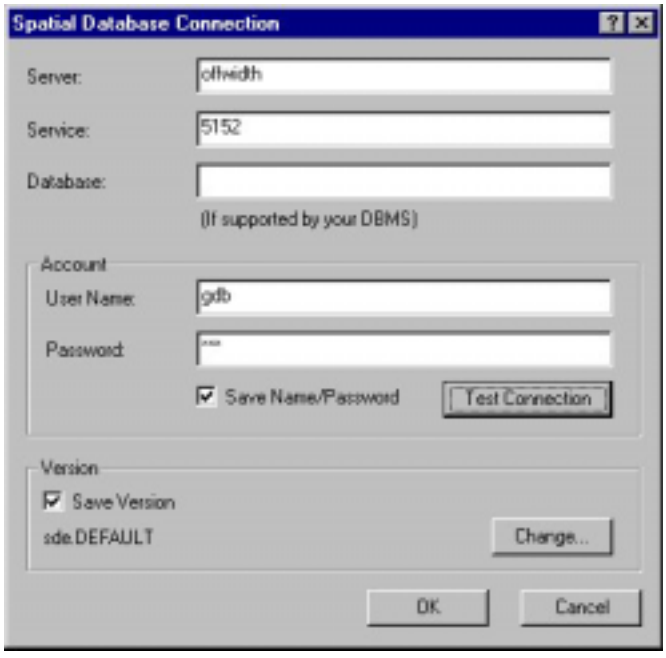
3. If the connecting user will own spatial data—that is, load spatial data with the ArcGIS data loaders such as ArcToolbox or the admin tools such as shp2sde or sdeimport—that user must have create table and create procedure permissions in the database that will store the loaded data.

4. You must explicitly connect to a database to perform DDL statements (create table, alter table, drop table). For example, if you want to load data into a database named county_hiways, you must connect to it by specifying a database name in your connection. ArcSDE does not use a login's default database!

Example: Connecting to ArcSDE from ArcCatalog

Open ArcCatalog. Navigate to the **Database Connections** node. Expand it and select 'Add Database Connection.' Double-click this to bring up the connection properties dialog box. Under server, enter the name of the ArcSDE server. For Service, input either your service instance name or your tcp/ip port (e.g., 5151). For Database, leave blank to connect to the sde database or input another database name. Input a user name and password for the remaining text boxes.

Figure 5.1: Connecting to a server through ArcCatalog



Example: Connecting to ArcSDE through ArcObjects™

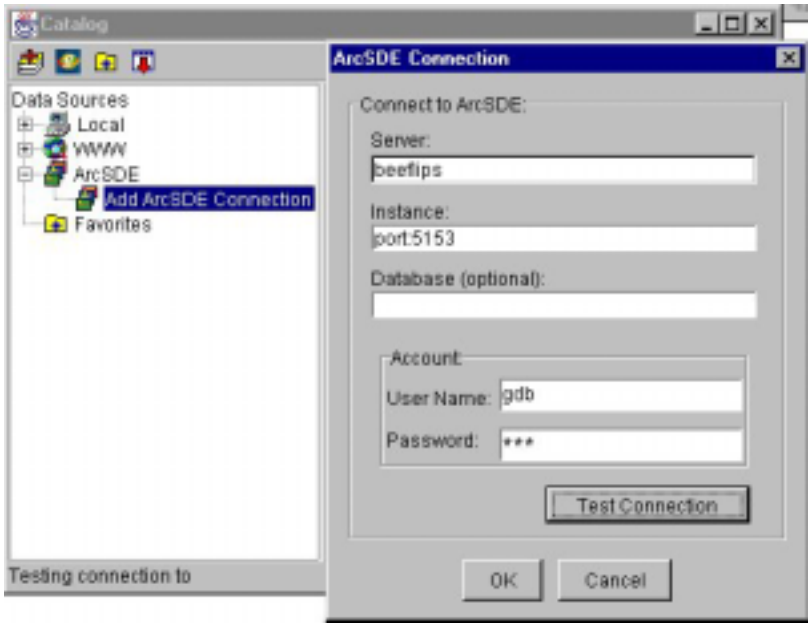
To use this sample, you must have the esricore tlb referenced in your MS Visual Basic development project.

```
Dim pPropSet as iPropertySet
Dim pWorkspaceFactory as iWorkspaceFactory
Dim pWorkspace as iWorkspace

set pPropSet = new PropertySet
set pWorkspaceFactory = new SDEWorkspaceFactory
with pPropSet
        .setProperty "Server" = "vogon"
        .setProperty "Instance" = "5151"
        .setProperty "Database" = "contours"
        .setProperty "User" = "arthur"
        .setProperty "Password"="dent"
        .setProperty "version" = "sde.DEFAULT"
end with
set pWorkspace=pWorkspaceFactory.Open(pPropSet)
if pWorkspace is nothing then
        msgbox "Connection Failed"
else
        msgbox "Success"
end if
```

Example: Connecting to ArcSDE through ArcExplorer™

Open the Catalog, expand the ArcSDE servers node, then double-click the Add ArcSDE Connection entry. In the ArcSDE Connection panel, fill out your server, instance, database (if you leave this blank, you'll connect to the 'sde' database), user name, and password arguments, then click "Test Connection." If the connection is successful, (grayed out), then click OK.

Figure 5.2: Connecting to ArcSDE from ArcExplorer



Using the ArcSDE Direct Connect Driver

Beginning with ArcSDE 8.1, ArcSDE client applications can connect to either an ArcSDE service or directly to an MS SQL Server instance. The direct connection capability, added during the ArcSDE 8.1 release, was achieved by embedding the ArcSDE application server technology into the ArcSDE client software. Any application programmed with the functionality of the ArcSDE 8.1 C API library can connect directly to an MS SQL Server instance.

ArcSDE client applications that connect to an ArcSDE service send spatial requests to the service. The ArcSDE service converts the spatial requests into SQL statements and submits them to the SQL Server instance. Upon receiving results from the server, the ArcSDE service converts the result into spatial data recognizable to the ArcSDE client. At ArcSDE 8.1, the functionality of the ArcSDE service has been linked into the ArcSDE C API, allowing ArcSDE client applications to connect directly to MS SQL Server instances. Spatial requests are converted to SQL statements by the ArcSDE client applications instead of the ArcSDE service.

Before you can connect to an ArcSDE service, you must install and configure the ArcSDE product. The sde user must be created and the sde setup program (sdesetupmssql.exe) must be run to create and populate the ArcSDE system tables and required stored procedures. In addition, ArcSDE must be able to reference an ArcSdeServer license from a license manager accessible through the network. Since a direct connection to an MS SQL Server instance does not require the presence of an ArcSDE service, the ArcSDE product does not need to be installed. You are, however, required to run the ArcSDE setup program, which creates the necessary ArcSDE system tables in the sde user's schema. Also, to obtain a read–write connection to the MS SQL Server database, an ArcSDE client application must reference a valid ArcSdeServer license. If an ArcSdeServer license cannot be referenced, access is restricted to read-only.

The ArcSDE setup program is located in the bin directory of all ESRI products capable of connecting to ArcSDE. If you do not already have one, an ArcSdeServer license can be obtained from ESRI Customer Support. To connect directly to an MS SQL Server instance, you must have Microsoft Data Access Components 2.5 installed on your client machine. This MDAC is installed by ArcGIS, so you may already have it. The sections that follow provide details on how to set up the client and server machines to perform a direct connection to an MS SQL Server instance. For more information regarding an ArcSDE service, refer to the *ArcSDE Installation Guide* for installation instructions and the *Managing ArcSDE Services* for configuration and connection instructions.

All ArcSDE client applications, such as ArcMap or ArcCatalog, function the same way regardless of whether a user connects to an ArcSDE service or directly to an MS SQL Server instance. The only difference occurs during the entry of the connection information.

Setting up your server without installing ArcSDE

ArcSDE requires an 'sde' database and at least an 'sde' user with create table, view, and stored procedure permissions. The two most common ways of doing this are with the Enterprise Manager for SQL Server 7 or SQL Server 2000 or with a t-sql script. After creating your sde database and user, you must run the sdesetupmssql in the \bin directory where you have the gsrvrsql81.dll; the sdesqlsvr81.dll; and three client libs sg81.dll, pe81.dll, and sde81.dll. Finally, set either SDEHOME or ARCHOME to the directory containing the bin folder holding these files.

1. Create an ArcSDE database and user

Either run this script in the SQL Server Query Analyzer or see the screenshots from the SQL Server Enterprise Manager below:

NOTE: If you run this script, it will be necessary to change your paths where indicated.

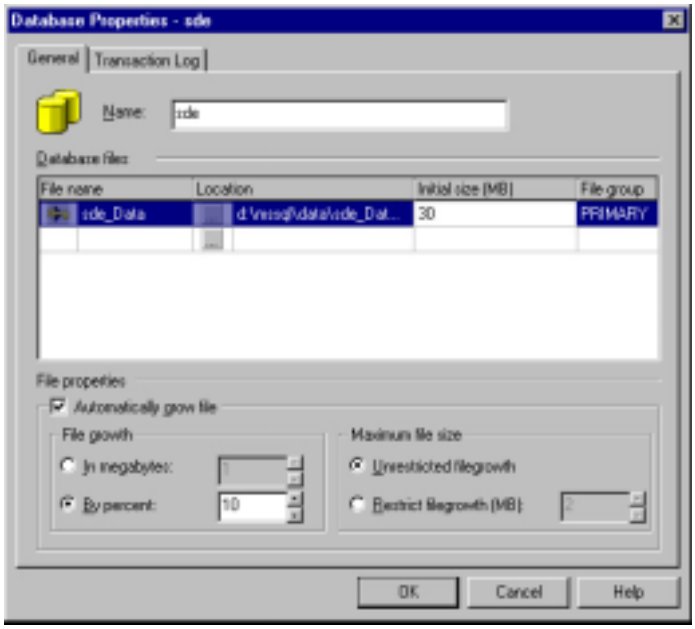
```
/*
Script: setupSDEdb.sql
purpose:
1. Creates sde database if it doesn't exist
2. Creates sde and gdb logins if they don't exist
3. adds logins to sde database if they are not there
4. grants create table view and procedure to users in sde.

NOTE: Change the paths in the filename statements below
These files will be written to your disk
*/

use master
go

if exists (select name from sysdatabases where name = 'sde')
 begin
  print 'SDE database already exists, attempting to create login and user'
 end
else
 begin
  CREATE DATABASE sde
  on
  ( name='sde',
  --change paths here
  --to a valid location on
  --your disk
  filename='d:\sdedb\sql\sde.mdf',
  size=30,
  maxsize=UNLIMITED,
  filegrowth=30 )
  LOG ON
  ( name='sdelog',
  --change paths here
  --to a valid location on
  --your disk
  filename='d:\sdedb\sql\sdelog.ldf',
  size=10mb,
  maxsize=UNLIMITED,
  filegrowth=10mb)
 end

/*
now attempt to create logins and users
if they don't exist
*/

if exists (select name from syslogins where name = 'sde')
 begin
  print 'sde login exists, attempting to create test user'
 end
else
 begin
  --create sde login
  --password is 'go'
  --default db = sde
  exec sp_addlogin 'sde', 'go', 'sde'
 end

--create a test user
if exists (select name from syslogins where name = 'gdb')
 begin
  print 'gdb test login already exists'
 end
else
 begin
  --create a test login whose name = gdb
  --and password = gdb
  exec sp_addlogin 'gdb','gdb','sde'
```

```
 end

/*
switch db context to sde
check for users in sde
add if necessary
grant create table, view and procedure to
both users
*/
go
use sde
go

--sde login
if exists (select name from sysusers where name = 'sde')
 begin
  print 'sde user already added to sde database'
 end
else
 begin
  exec sp_adduser sde
 end

--gdb login
if exists (select name from sysusers where name = 'gdb')
 begin
  print 'gdb user already added to sde database'
 end
else
 begin
  exec sp_adduser gdb
 end
go

grant create table to sde
grant create table to gdb
grant create view to sde
grant create view to gdb
grant create procedure to sde
grant create function to sde –-SQL SERVER 2K ONLY!!
grant create procedure to gdb

/*
make sure you run sdesetupmssql -o install
from either %sdehome%\bin or %archome%\bin
*/
```

2. Create the SDE database and user with the SQL Server Enterprise Manager

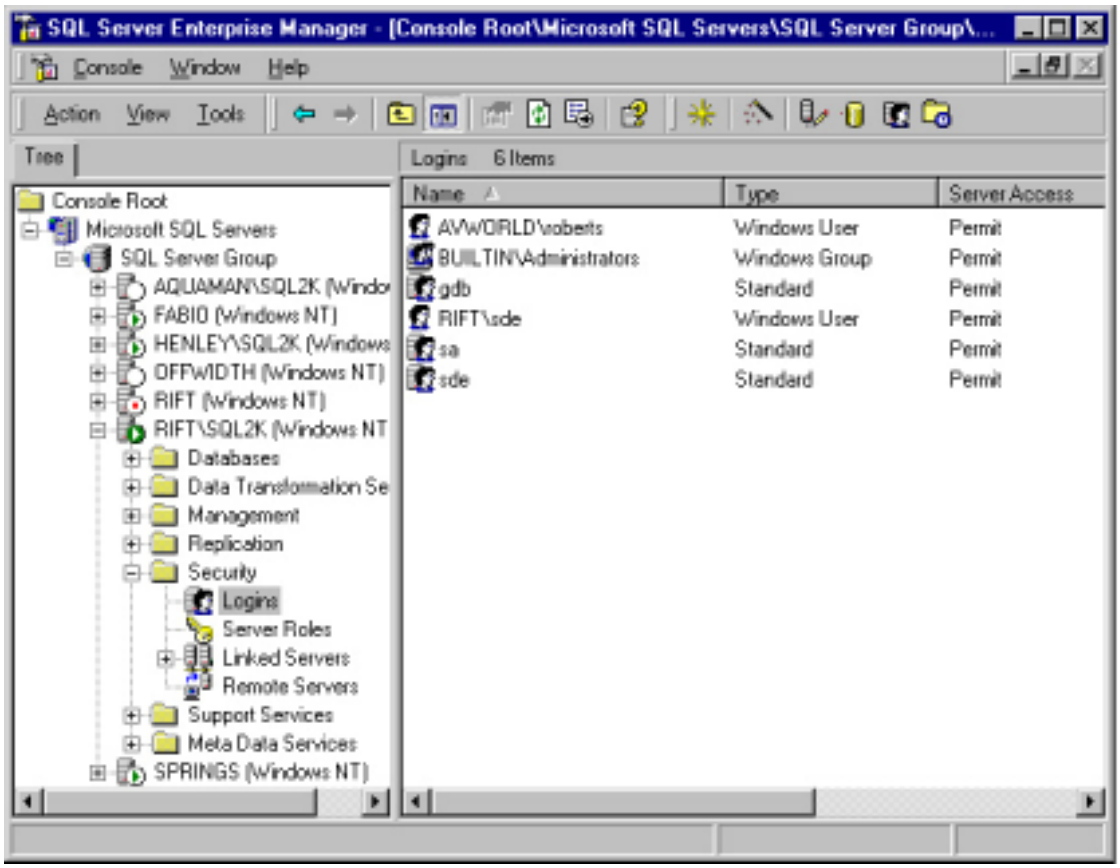
Create the sde database

Open Enterprise Manager, expand your "Microsoft SQL Servers" node, expand your server, right-click Databases, and select New Database.

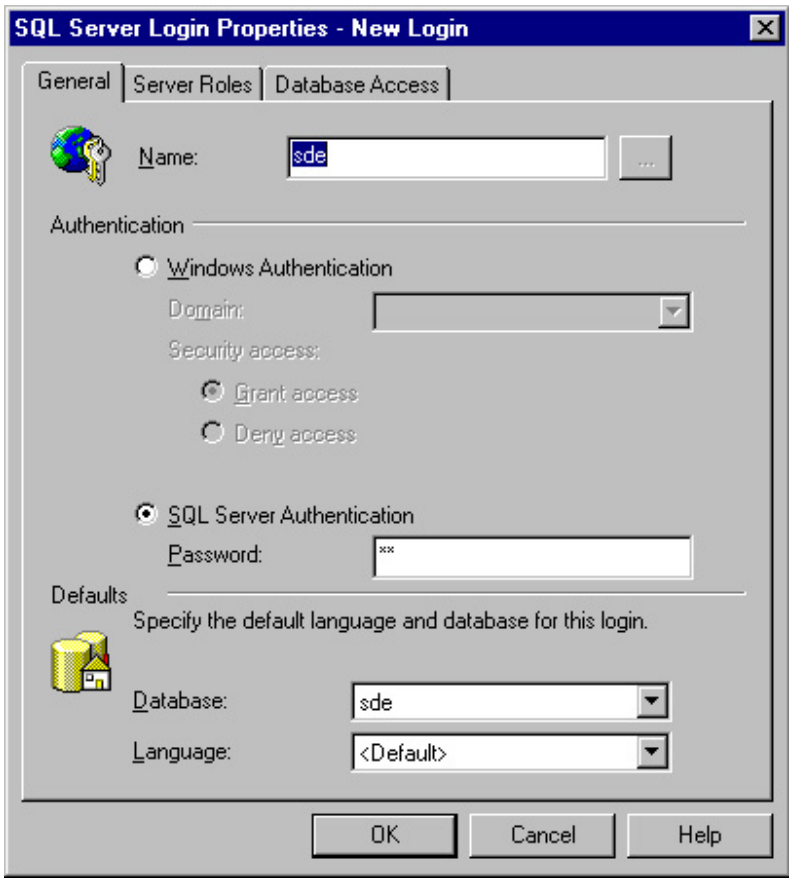


You can take the defaults here or fill out the initial size, location, etc. Do the same for the transaction log. Click OK and the sde database will be created.

Create the sde login and user: From within your server's node, expand the security node and select logins.

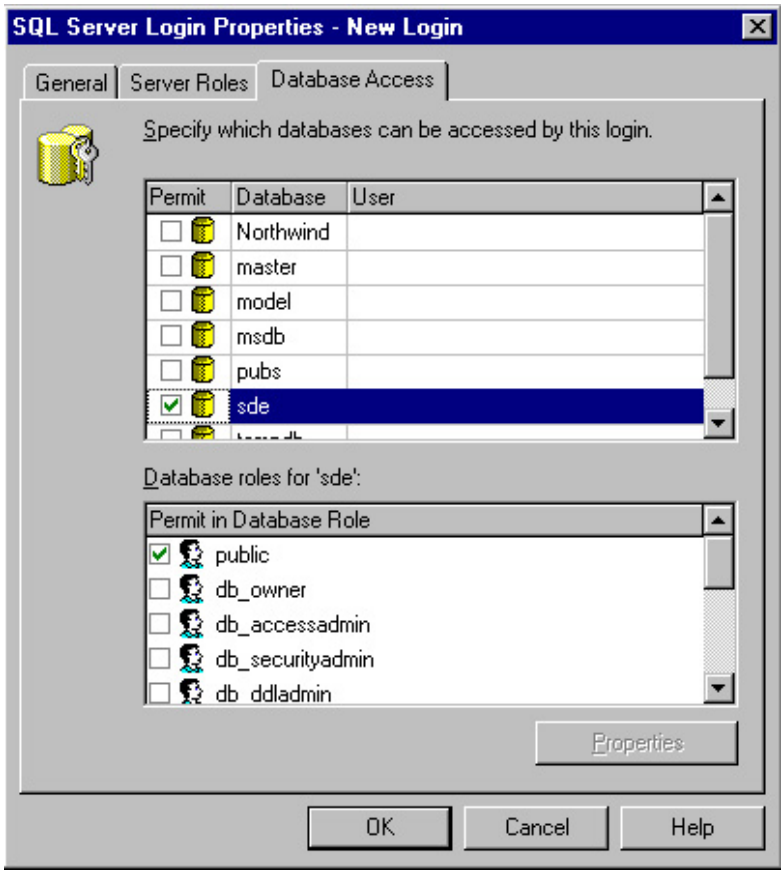


Now right-click in the right-hand panel. Select New Login.



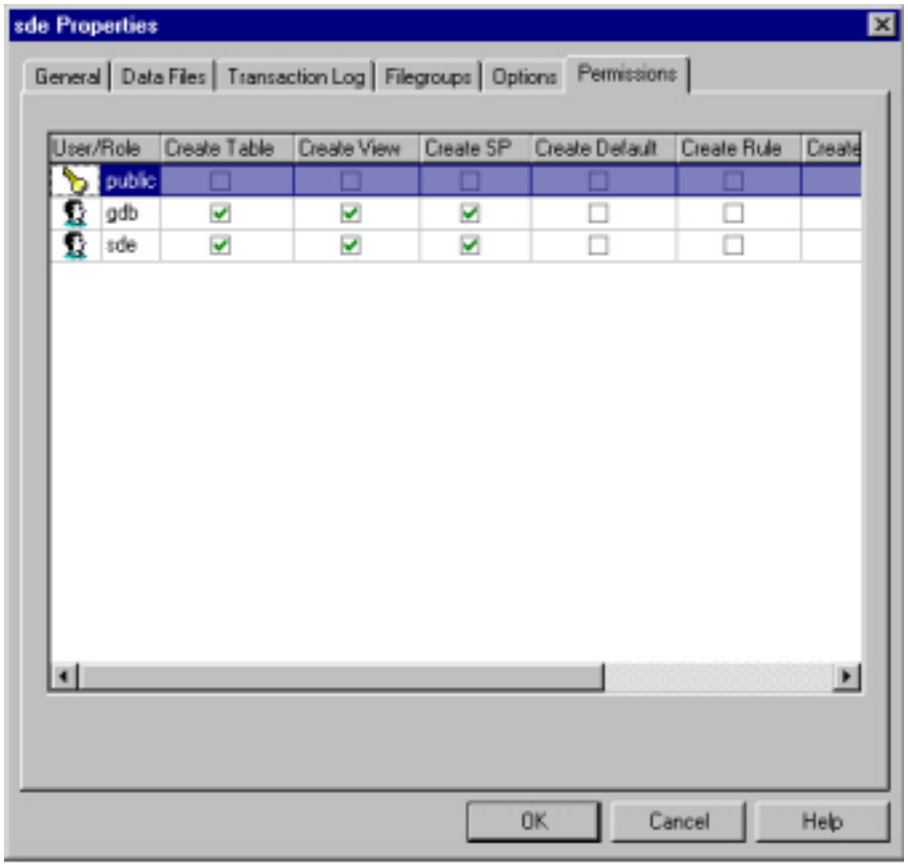
Fill in the login name property and password.

The default database should be ‘sde.’ Now click the ‘Database Access’ tab and click the sde database check box.



Now grant this new user create table, view and stored procedure in the ‘sde’ database. Go back to the database’s node, right-click the ‘sde’ database, and select Properties. When the Properties dialog appears, click the Permissions tab.

Check the create table, view, and sp (stored procedure) check boxes. If you are using SQL Server 2000, you must also grant create function to sde. Click OK.



RUN sdesetupmssql

You are almost done. Now from a command prompt, navigate to the directory that contains gsrvrsql81.dll, sdesqlsvr81.dll, sg81.dll, sde81.dll, and pe81.dll with a cd (chdir) command. There should also be an executable in this directory called sdesetupmssql.exe. In this example, these executables are located in the ArcGIS installation folder. This executable creates the sde and Geodatabase metadata. The syntax for this command is listed below.

```
C:\ArcGIS\arcexe81\Bin>sdesetupmssql
Usage for sdesetupmssql:
-?
-h
-o upgrade [-H <sde_directory>] [-u <DB_Admin_user>] [-p
<DB_Admin_password>]
[-s datasource] [-N] [-q]
-o list [-H <sde_directory>] [-u <DB_Admin_user>] [-p <DB_Admin_password>]
[-s datasource] [-q]
-o install [-H <sde_directory>] [-u <DB_Admin_user>] [-p
<DB_Admin_password>]
[-s datasource] [-N] [-q]
```

In this example below, sdesetupmssql will be run against a data source called “rift\sql2k,” which is a SQL Server 2000 instance name.

```
C:\ArcGIS\arcexe81\Bin>sdesetupmssql -o install -u sde -p go -s rift\sql2k
-H c:\ArcGIS\arcexe81
```

The user should confirm that the –H switch is entered properly.

How do you know what your data source is? The data source will almost always be the name of your SQL Server host. However, if you have both SQL Server 7 and SQL Server 2000 installed, the SQL Server 2000 instance will have a different name that you assigned to it when you did the installation. If this is the case, look in the Enterprise Manager. You should see both your instances listed here.

Making a direct connection

Whether or not you have an ArcSDE service running or have installed the ArcSDE software, you are ready to make a direct connection. Here's what you must do:

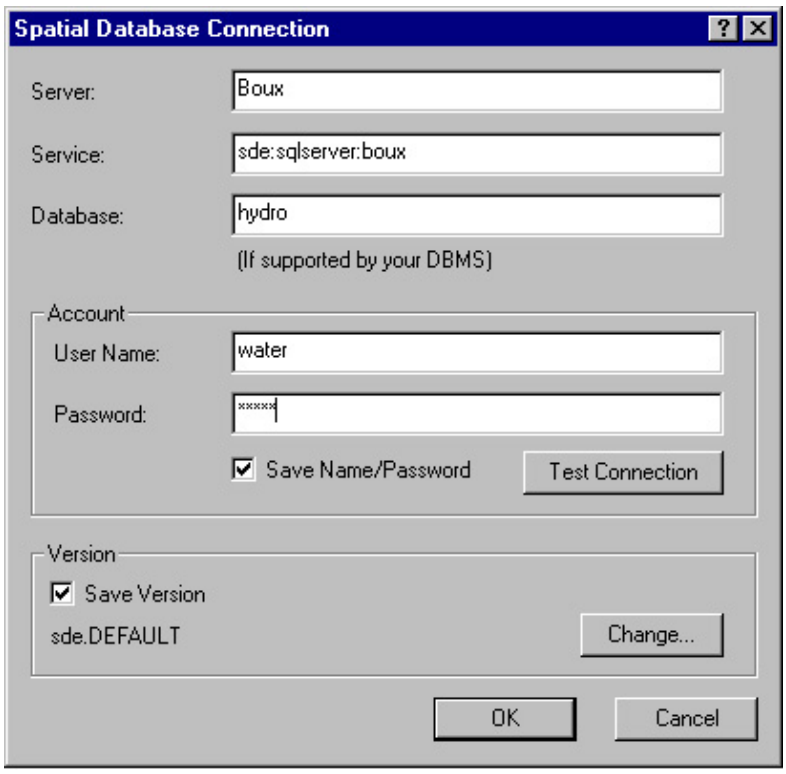
You must have sdehome set and pointing to the folder that contains the bin directory holding the gsrvrsql81.dll; the sdesqlsvr81.dll; and the client libraries sg81.dll, sde81.dll, and pe81.dll.

- You can use substitute SDEHOME with ARCHOME.
- Or set neither and let the search default to HKEY_LOCAL_MACHINE-SOFTWARE-ESRI-ArcInfo-Desktop-8.0-SourceDir - BIN.

For example: Presume you are attempting a direct connection from ArcCatalog. You have installed ArcGIS into c:\arcgis\arcexe81. The bin folder beneath the arcexe81 directory holds the gsrvrsql81.dll, the sdesqlsvr81.dll, and the three client libs. You can set SDEHOME to c:\arcgis\arcexe81 and attempt your direct connection.

If you have one setup, try to connect. To connect:
- Server name = ArcSDE Server (or any string)
- Instance = sde:sqlserver:<data source>
- Database = a database you want to explicitly connect to
- User = any valid DBMS/ArcSDE user
- Password = any valid DBMS/ArcSDE user's password

You must have at least MDAC 2.5 installed. How can you tell? Check the file version of msado15.dll in \program files\common files\system\ado. If its version is 2.5 or greater, you are OK. You can also use the component checker from Microsoft. This tool will allow you to remove MDAC as well. Beware of installing an MDAC onto Windows 2000; it may already have it installed.

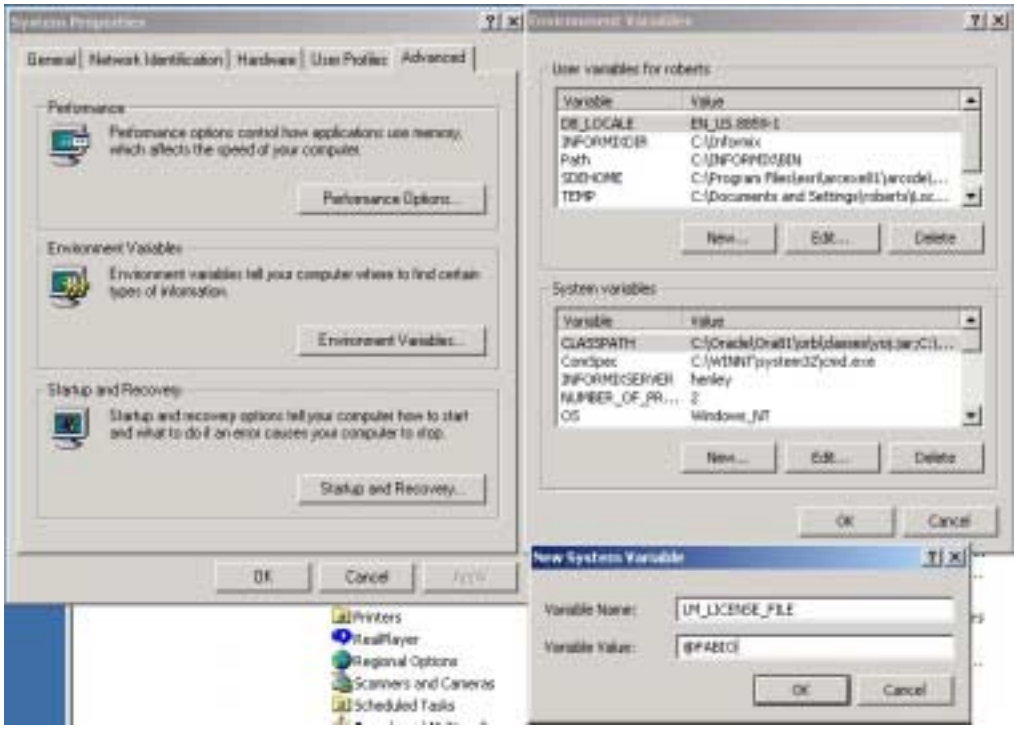


Licensing Direct Connect clients

If you do not have a valid ArcSDE license service, you will still be able to connect to any ArcSDE data store. However, your connections will be read-only. If you do have a license server, you can set an environment variable, LM_LICENSE_FILE, to access those licenses. Make sure you set it with the host name prefixed with an @ sign. For example, setting this variable in DOS would look like this:

C:\sde>SET `LM_LICENSE_FILE=@FABIO`

Setting this variable in Windows 2000 would look like the following panel.


General
Network Identification
Hardware
User Profiles
Advanced
Performance
Performance options control how applications use memory, which affects the speed of your computer.
Performance Options...
Environment Variables
Environment variables tell your computer where to find certain types of information.
Environment Variables...
Startup and Recovery
Startup and recovery options tell your computer how to start and what to do if an error causes your computer to stop.
Startup and Recovery...
OK
Cancel
User variables for roberts
Variable
Value
DB_LOCALE
EN_US.8859-1
INFORMIXDIR
C:\Informix
Path
C:\INFORMIX\BIN
SDEHOME
TEMP
New...
Edit...
Delete
System variables
CLASSPATH
ComSpec
INFORMIXSERVER
henley
NUMBER_OF_PR...
2
OS
Windows_NT
New System Variable
Variable Name:
LM_LICENSE_FILE
Variable Value:
Printers
RealPlayer
Regional Options
Scanners and Cameras
Scheduled Tasks

CHAPTER 6

National language support

Storing data in an ArcSDE SQL Server database using collation sets other than the SQL Server default, Latin1_General, requires some extra configuration on both the client and the server. This section provides guidelines for configuring both the SQL Server database and the ArcSDE client environment to enable the use of collation sets other than Latin1_General.

SQL Server database collation designator

The default collation designator for SQL Server 2000 databases is Latin1_General. The collation is selected when SQL Server 2000 is installed and cannot be changed afterwards. The default collation will support U.S. English and most Western European languages. Change this default only if you are not using a language from this group. After installation is complete, to change a collation designator the rebuild master utility must be run and the data reloaded. Consult the SQL Server Books Online to determine the collation designator that is appropriate for your data. Also, select how data will be sorted. Sort order can be binary, case sensitive, accent sensitive, kana sensitive, or width sensitive. If the binary option is not selected, SQL Server 2000 follows sorting and comparison rules as defined in dictionaries for the associated language or alphabet.

It is also possible to choose an alternative collation if backward compatibility with previous installations of SQL Server 7 or earlier is desired. The choice here will then depend on the collation choice made in the earlier instance of SQL Server. If the previous installation accepted the defaults, select Dictionary Order, Case Insensitive. This is backward compatible with the 1252 character set, which was the default in SQL Server 6.5 and SQL Server 7.0.

Setting the SDE_CHARSET variable for Windows NT clients

Exercise care in setting the SDE_SQLCHARSET on the Windows NT platform. In the Windows NT environment there are two different codepage environments, Windows American National Standard Institute (ANSI) and Original Equipment Manufacturer (OEM). Windows applications such as ArcInfo and ArcView run in the ANSI codepage environment. ArcSDE administration tools and C API applications invoked from the MS–DOS Command Prompt run in the OEM codepage environment.

If you use both Windows applications and MS–DOS applications together then set the SDE_SQLCHARSET variable for MS–DOS applications by opening the MS–DOS Command Prompt and issuing a MS–DOS SET command. Initially verify that the data to be loaded is in either ANSI or OEM format. From the MS–DOS Command Prompt issue one of the following commands:

For sdeimport or sdeexport:commands:

If the data is in ANSI format: **set SDE_SQLCHARSET=cp1257**

If the data is in OEM format: **set SDE_SQLCHARSET=cp437**

In the above ANSI example, cp1257 would be for Baltic languages. In the OEM example, cp775 is for Western European languages. The user should insert the correct number for the language that they are using. A complete list of code pages may be found in SQL Server books online. The user should also not place any blank spaces at the end of the command.

For sde2shp, shp2sde, cov2sde, or sde2cov commands:

If the code is running from an MS–DOS Command Prompt: **set SDE_SQLCHARSET=cp775**

If the code is running from a UNIX platform: **setenv SDE_SQL_CHARSET iso_13**

Again, the user should insert the appropriate number for their language in these commands and should not place a blank space after the command.

CHAPTER 7

Standby servers and log shipping

The use of a standby server that can be brought online if the primary production server fails can avert the loss of data or allow users to continue working with the database. SQL Server 7 and SQL Server 2000 both support the use of standby servers. A standby server contains a copy of the databases on the primary server and can be brought online if the primary production server fails.

Implementing a standby server

Implementing a standby server involves three phases:

1. Creating the database and ongoing transaction log backups on the primary server.
2. Setting up and maintaining the standby server by backing up the database on the primary server and restoring it on the standby server.
3. Bringing the standby server online if the primary server fails.

If the use of a standby server is necessary, all users will have to log onto the standby server. User processes are not transferred automatically to the standby server. In addition, transactions are not maintained. It will be necessary to run sp_change_users_login once the switch to the standby server is made.

To test ArcSDE 8.1 in a Standby Server configuration, the following steps were implemented. For more information, see the instructions in SQL Server Books Online.

1. Create the full database backups on the Primary Server. Create a transaction log backup of each database that is duplicated. The frequency of these transaction log backups will depend on the amount of transaction changes that users make.

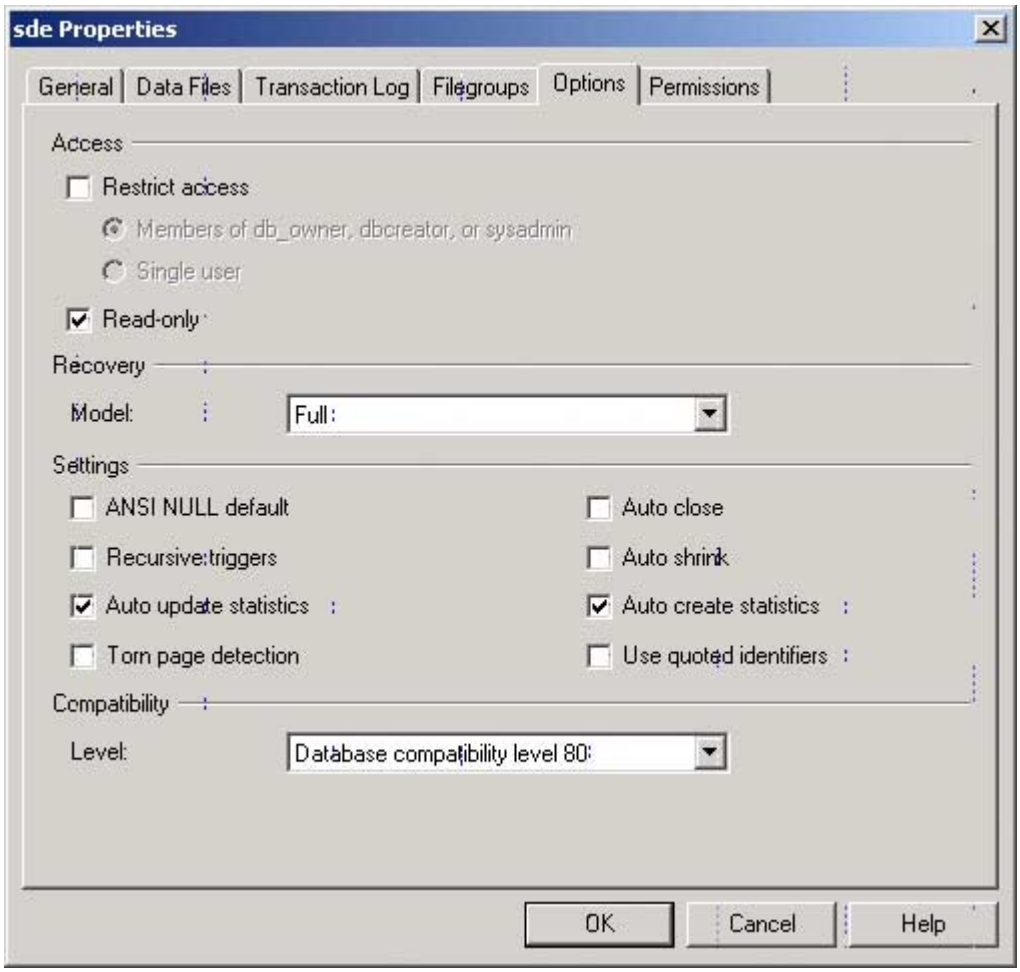
2. Restore the database backups onto the standby server that is in standby mode. Specify one undo file per database. Periodically apply the transaction logs from the primary server to the databases on the standby server. Specify the same undo file used in previous steps.

3. If the primary server should become unavailable, apply to the standby server any transaction logs that have not yet been applied to it. Recover the databases on the standby server.

4. The standby server is now available for users to make changes to the database. Configuration of the standby server can be done in either Enterprise Manager or Query Analyzer.

The following steps illustrate the creation of a standby server in Enterprise Manager.

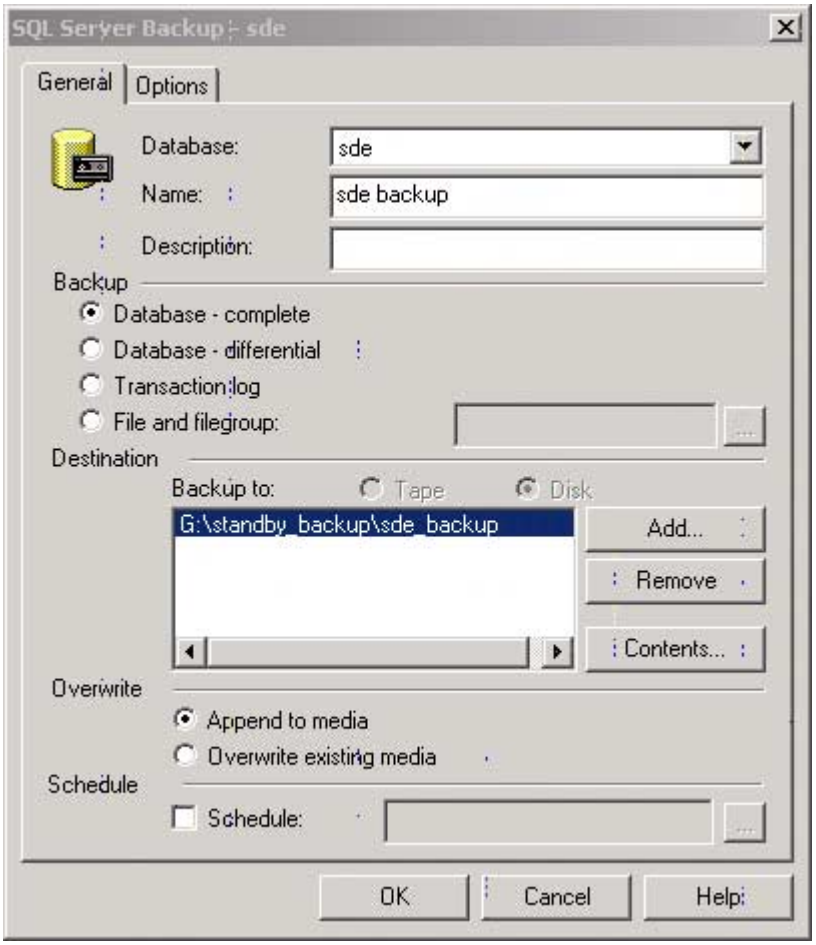
Confirm that the database is in Full Recovery Model.



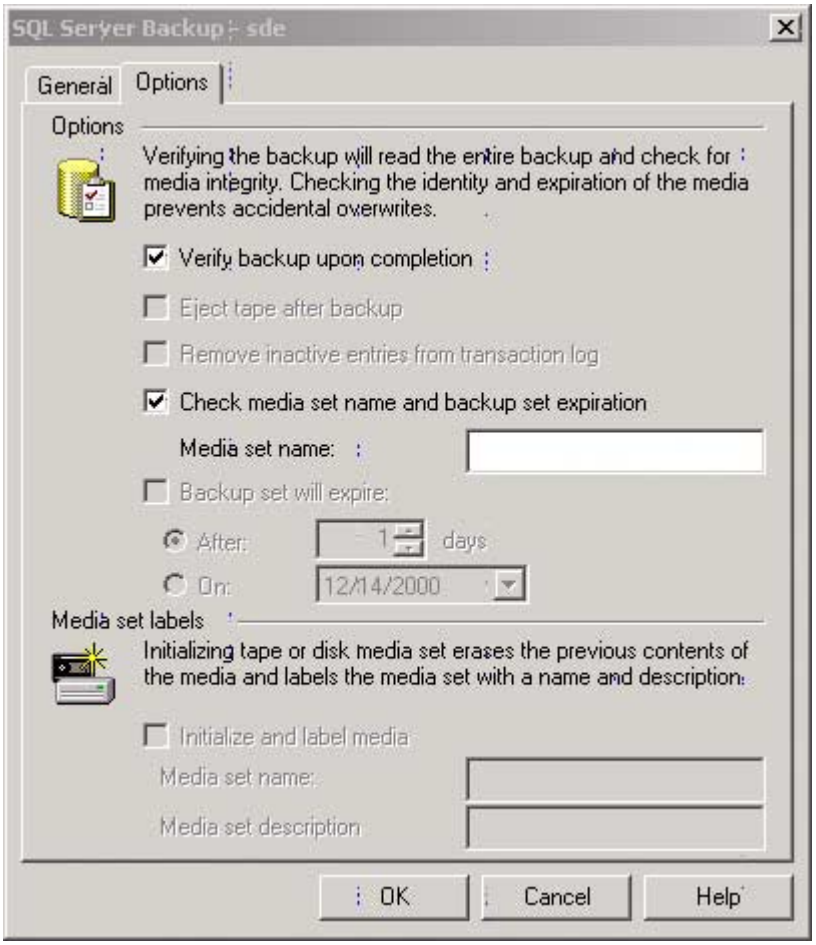
In order to implement transaction log backups, the database must be in Full Recovery Model. This is the default setting for a new database created with the Standard and Enterprise Editions of SQL Server 2000. To change the Recovery Model in Enterprise Manager, right-click on a database and choose Properties. Click the Options tab and change the recovery model to Full. Click OK to continue.

Backing Up Databases:

In Enterprise Manager, right-click on the database to backup, choose all tasks, Backup Database.

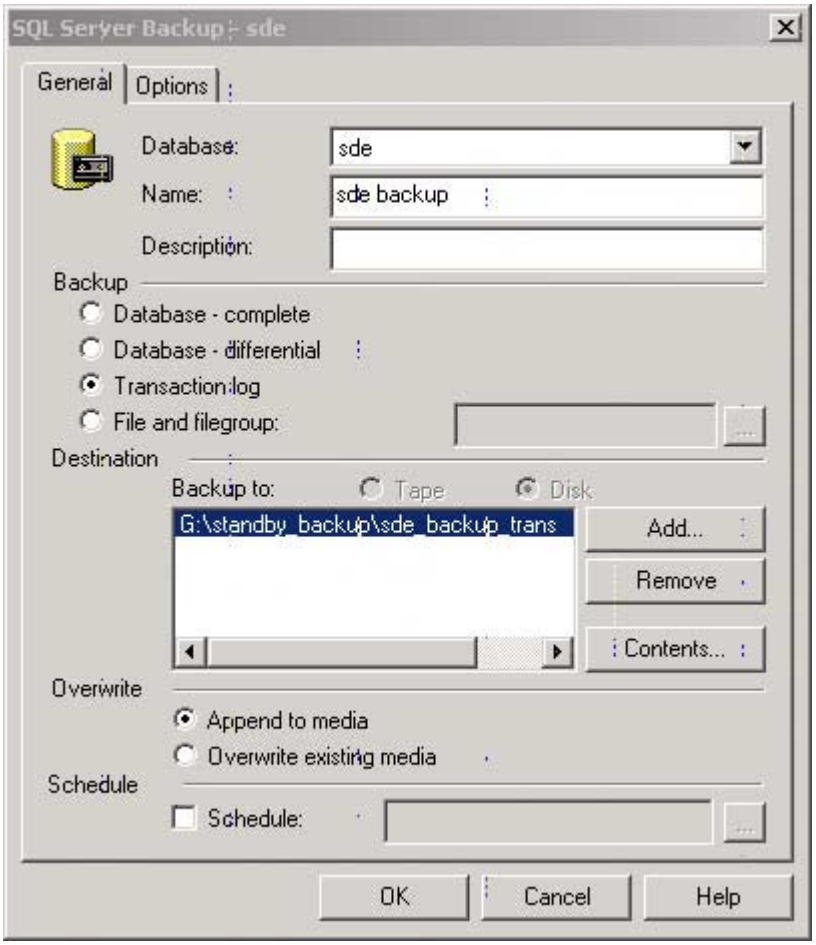


Choose Backup Database Complete and select the location in which to store the backup. Click on the Options tab.



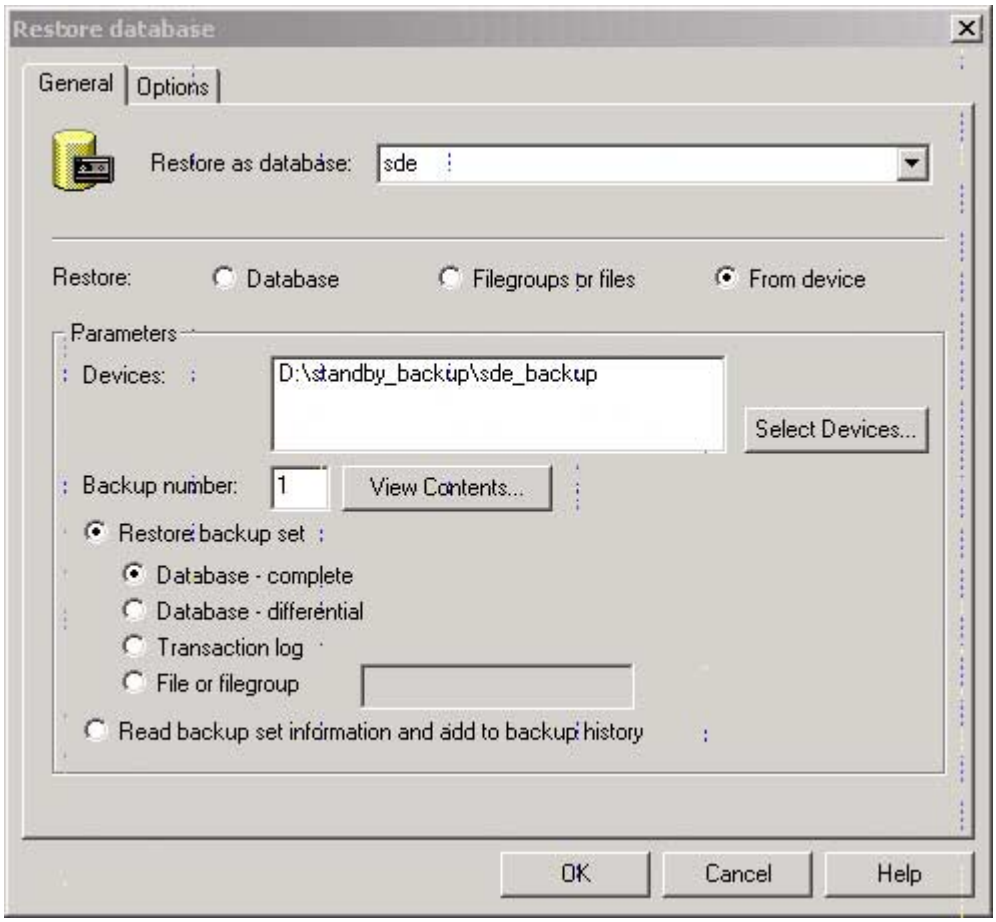
Click Verify backup upon completion and click OK to continue.

To back up the transaction log, right-click the database, choose all tasks, Backup Database.

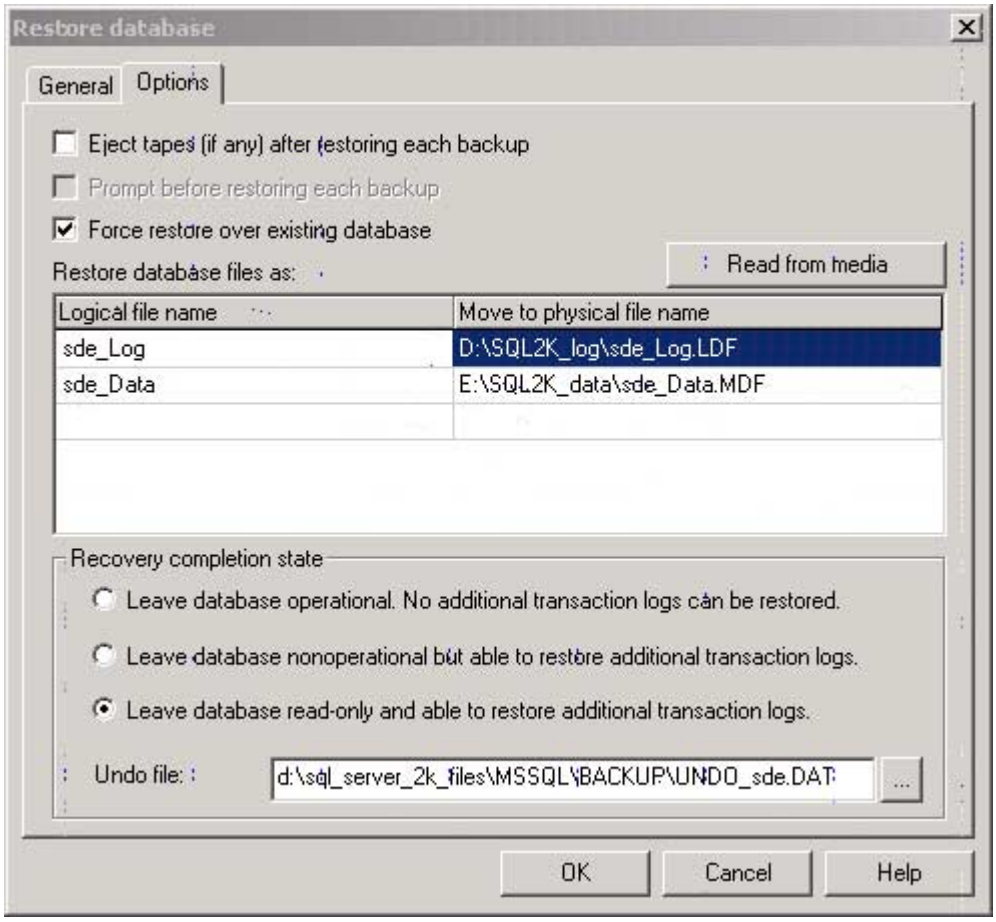


Click on Backup Transaction log. Choose the destination for the backup file. As before, click on the Options tab and click on Verify backup upon completion. Click on OK to continue.

To restore the database on the standby server, right-click on the database to restore, choose all tasks, restore database.



Choose Restore backup set Database - complete. Select Restore From device and then select the file to restore. Click on the Options tab.



In the Recovery completion state, select Leave database read-only and able to restore additional transaction logs. Choose the location and filename of the undo file. Click OK to continue. The database will be restored and will be in read-only mode. Continue this process for each transaction log to be restored on the standby server.

If the production server should become unavailable, apply all remaining transaction logs to the standby server. From Query Analyzer recover the standby server database without restoring. Execute the following statement with the name of the database to be recovered.

```
-- Restore database using WITH RECOVERY.
RESTORE DATABASE sde   WITH RECOVERY
```

The standby server will now be in the same state as the production server from the last transaction log. All changes after the last backup of the transaction log will be lost.

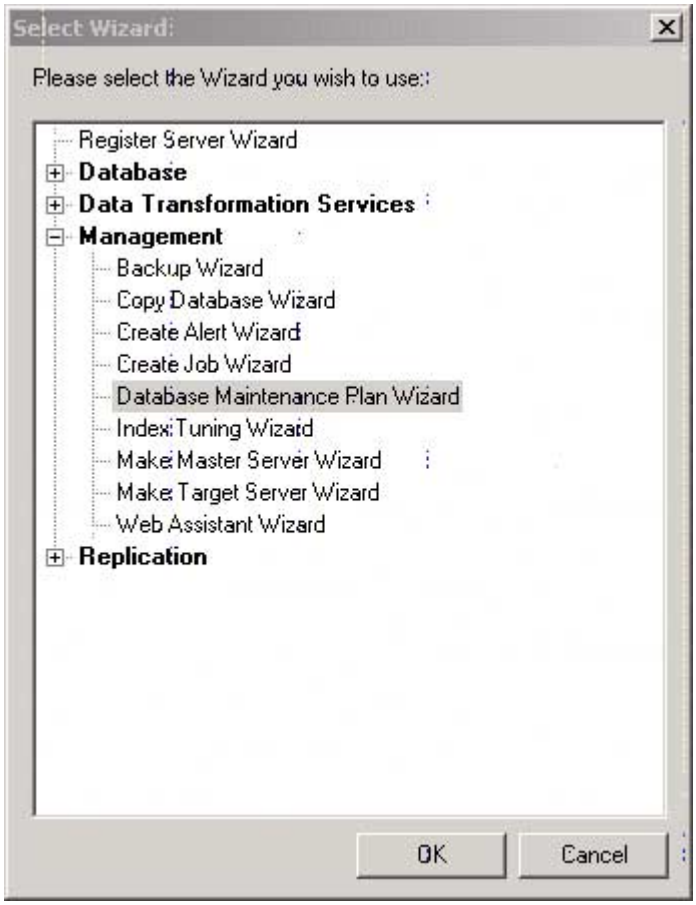
Log shipping

In SQL Server 2000 Enterprise Edition, log shipping can be used to send transaction logs from one database to another on a scheduled basis. This will allow synchronization of the source and the destination database. The use of this is very efficient with standby servers. Log shipping may be configured in Enterprise Manager. Complete instructions for standby server and log shipping are covered in SQL Server Books Online.

This example will use SQL Server 2000 Enterprise Edition to log ship transactions between two servers.

The initial procedure is the same as configuring a standby server. There should be a primary and standby server. The standby server should have the backups restored from the primary server and should be in read-only mode.

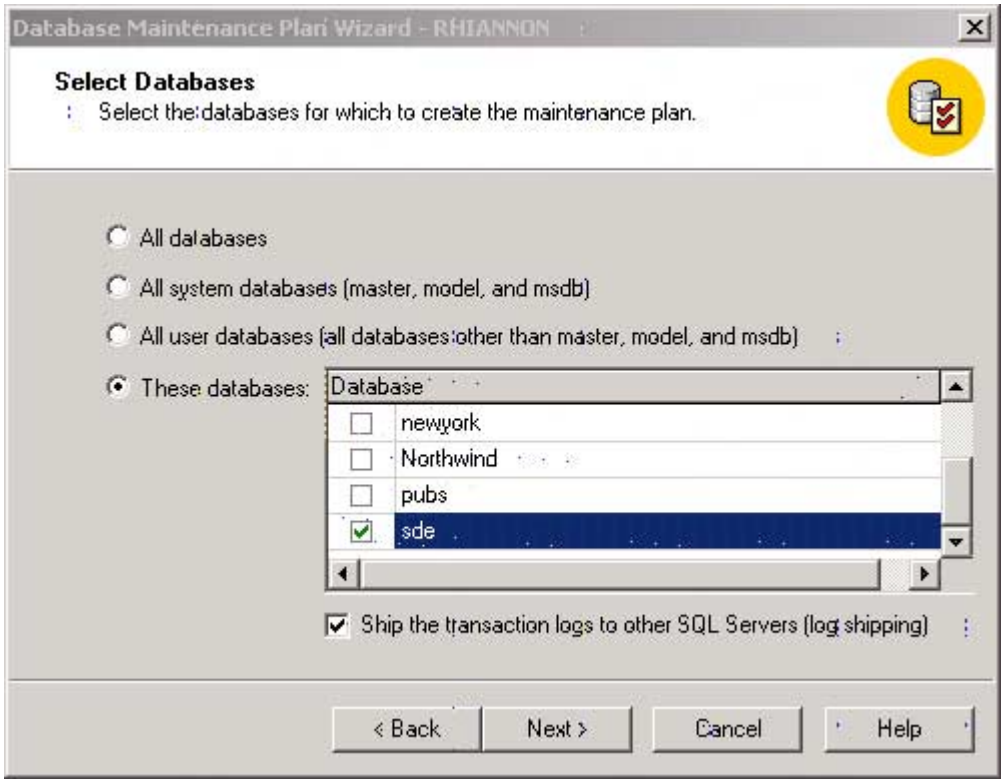
From the primary server, select Tools, Wizards, Management, Data Maintenance Plan Wizard.



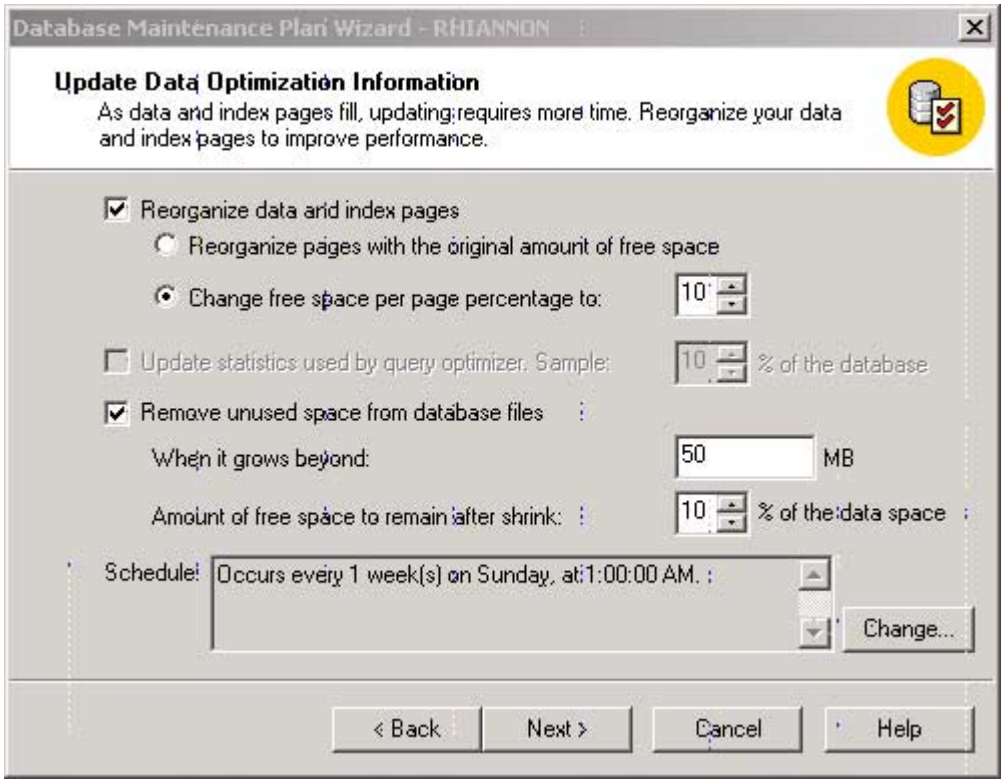
Click OK to start the wizard.


Database Maintenance Plan Wizard - RHIANNON
Welcome to the Database Maintenance Plan Wizard
This wizard helps you create a maintenance plan that the SQL Server Agent can run on a regular basis. With this wizard you can:
Run database integrity checks.
Update database statistics.
Perform database backups.
Ship transaction logs to another server (Enterprise Edition).
< Back
Next >
Cancel
Help

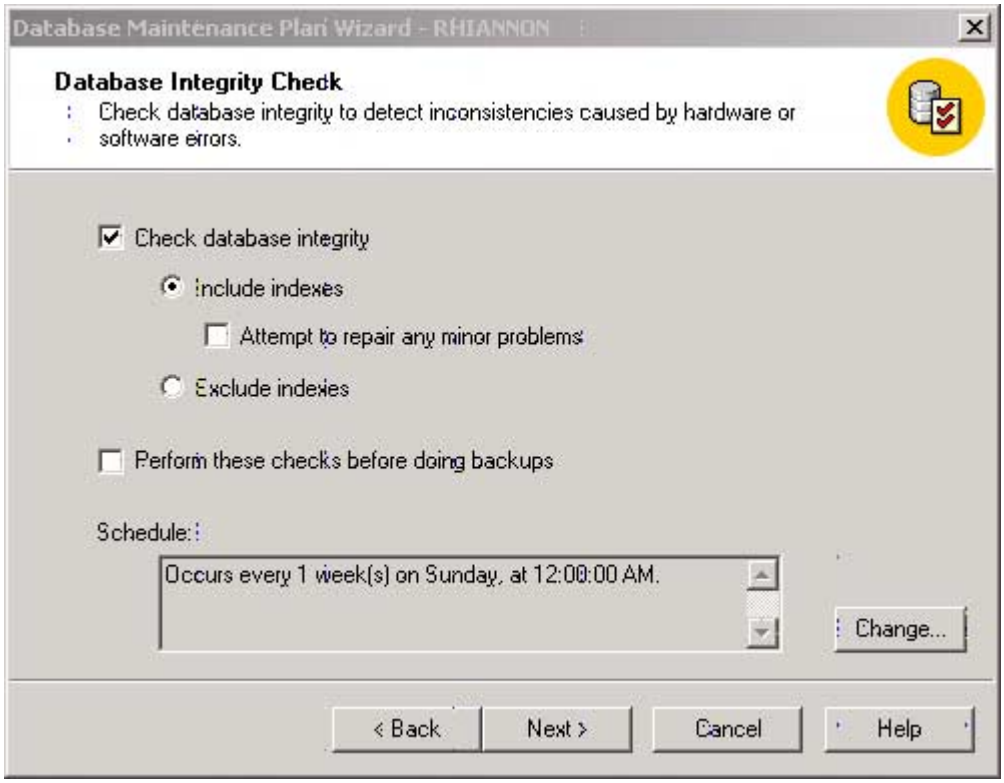
Click Next>.



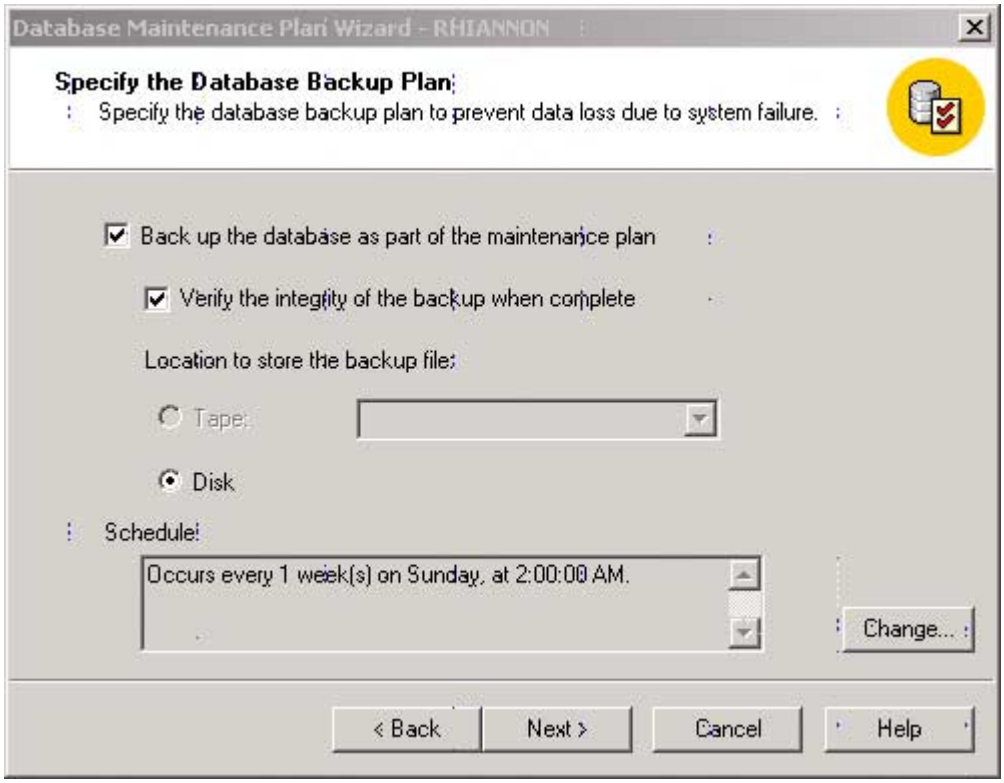
Select the database that the transaction logs are to be shipped from. Click the Ship the transaction logs to other SQL Servers check box. Click Next>.



As part of the update Wizard, tasks other than log shipping are performed. All defaults for these tasks will be accepted. Appropriate entries should be made for your installation. Click Next>.

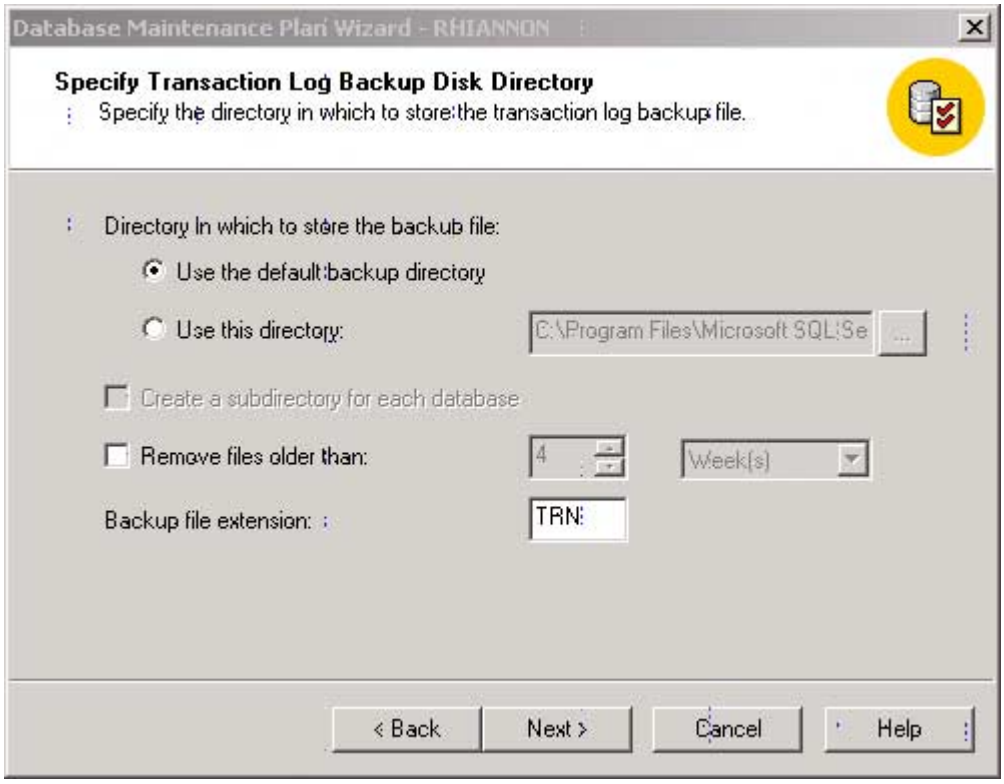


Default selections are shown. Selecting Attempt to repair any minor problems will put the database in single-user mode so that availability will be restricted. Click Next>.

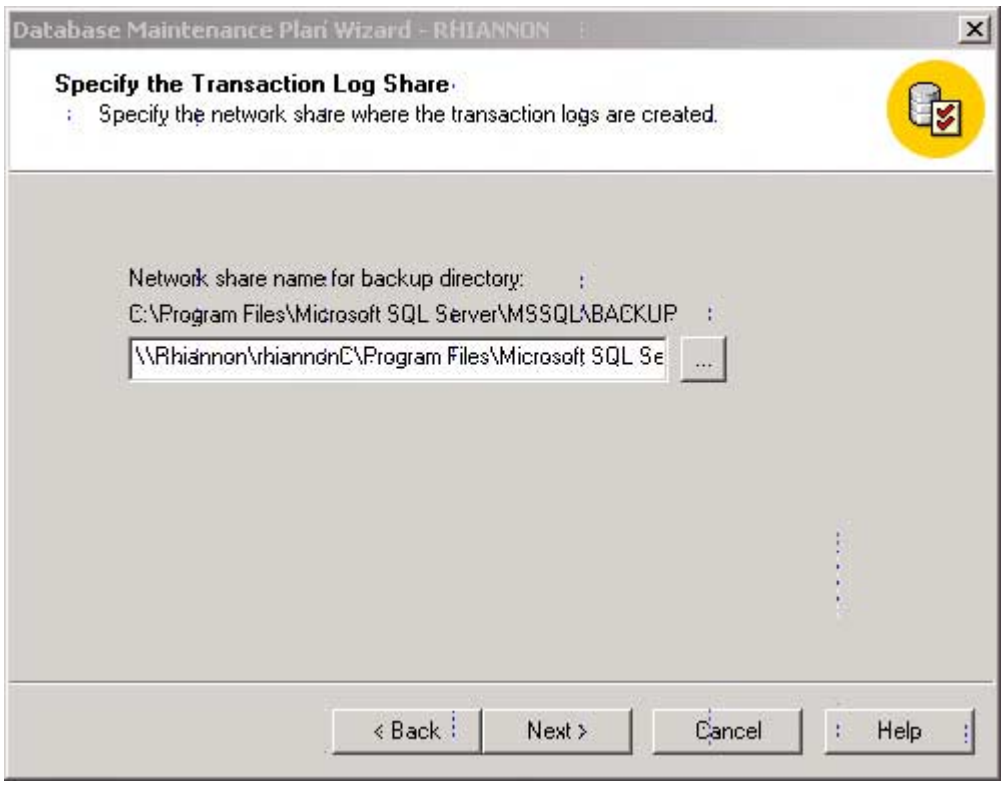


Select the database backup options and click Next>.

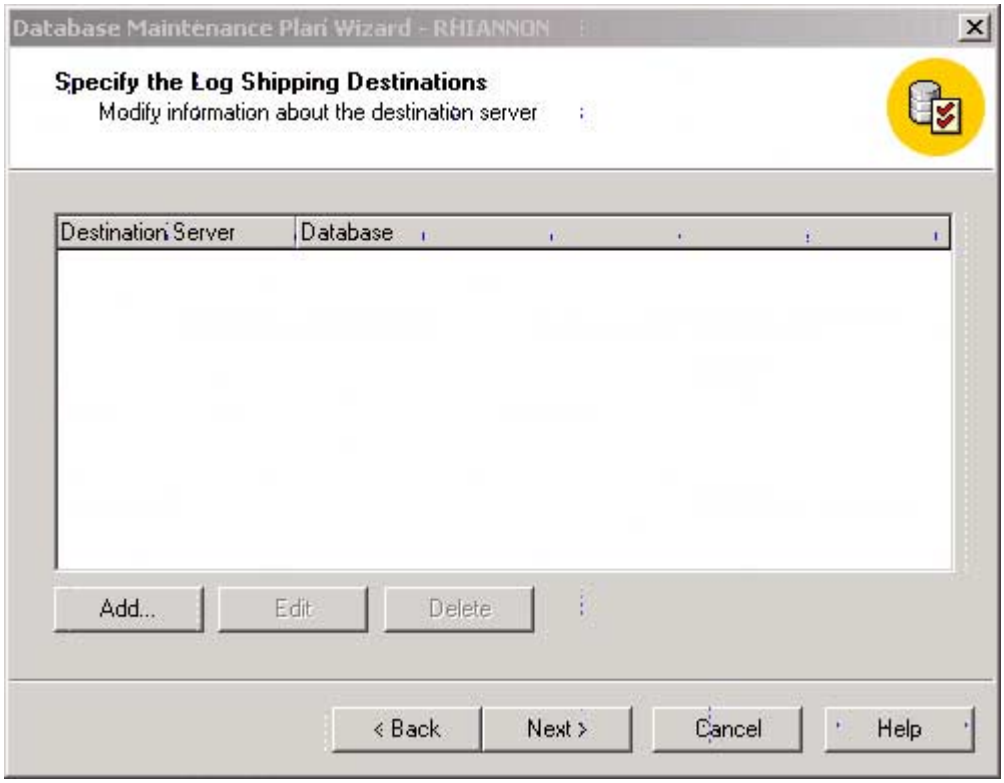
Specify the location to store the backup file. Click Next>.



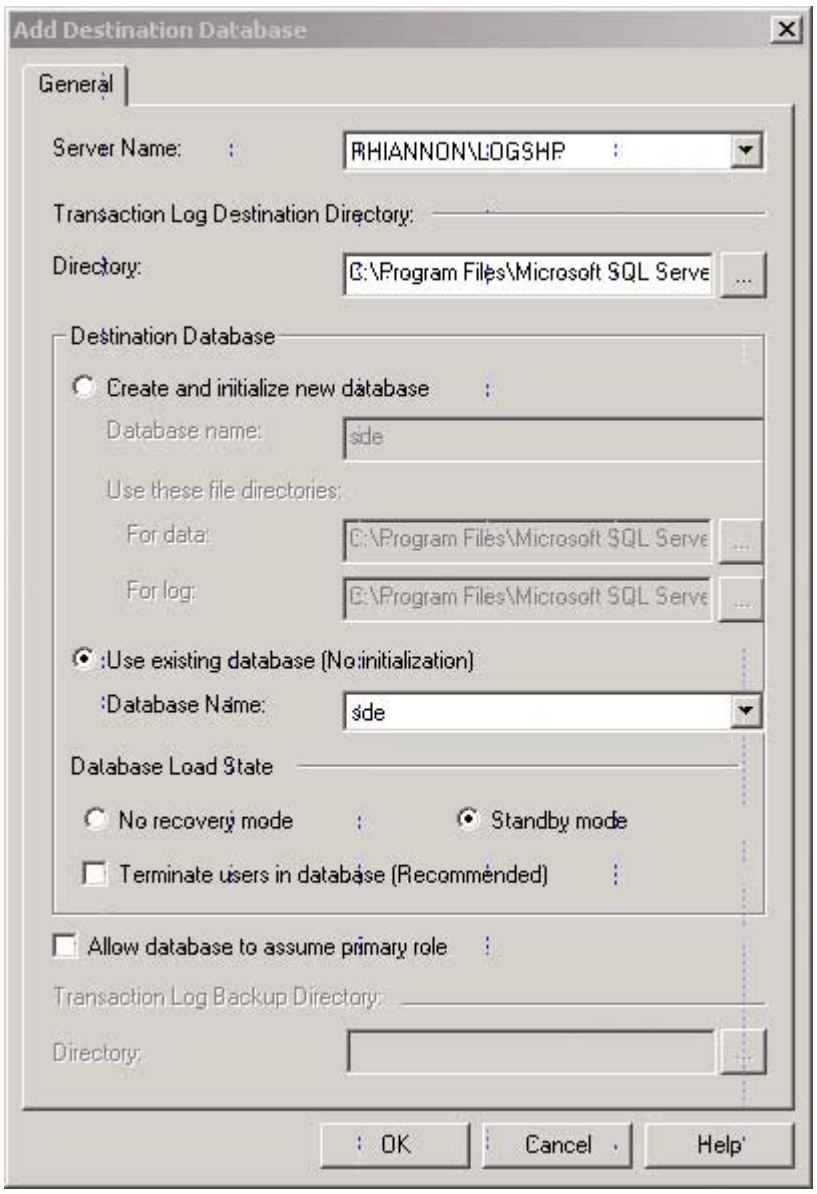
Specify the location to store the transaction backup file. Click Next>.



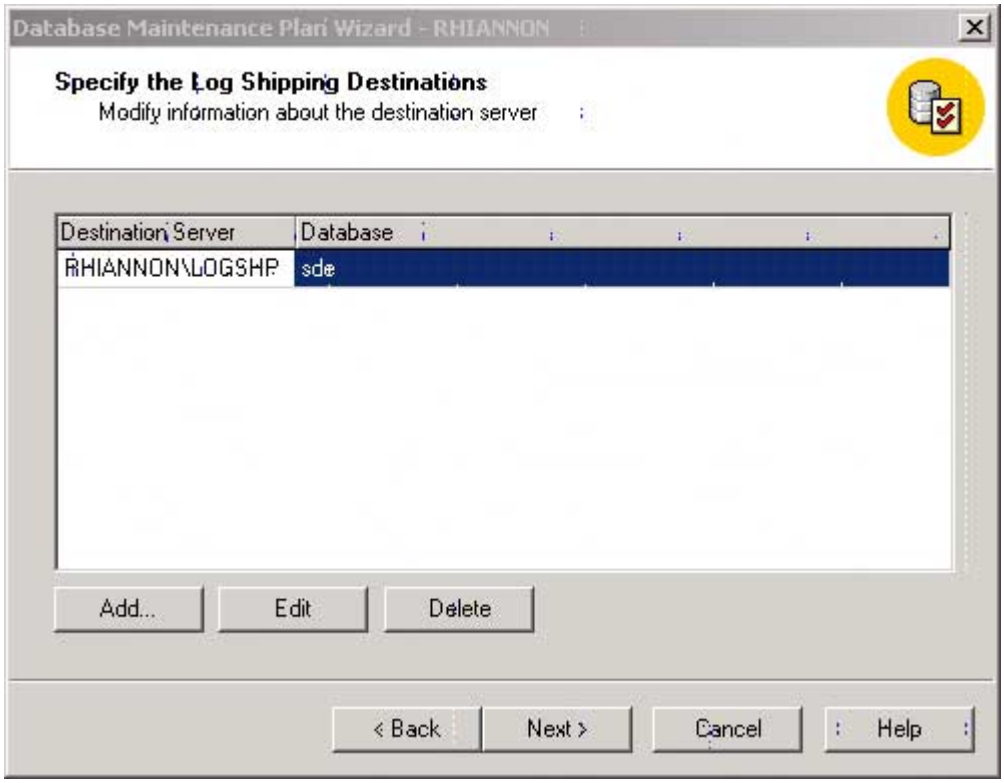
In order to run log shipping, a share on the primary server drive that stores the transaction log must be created. Enter the location of this drive and click Next>.



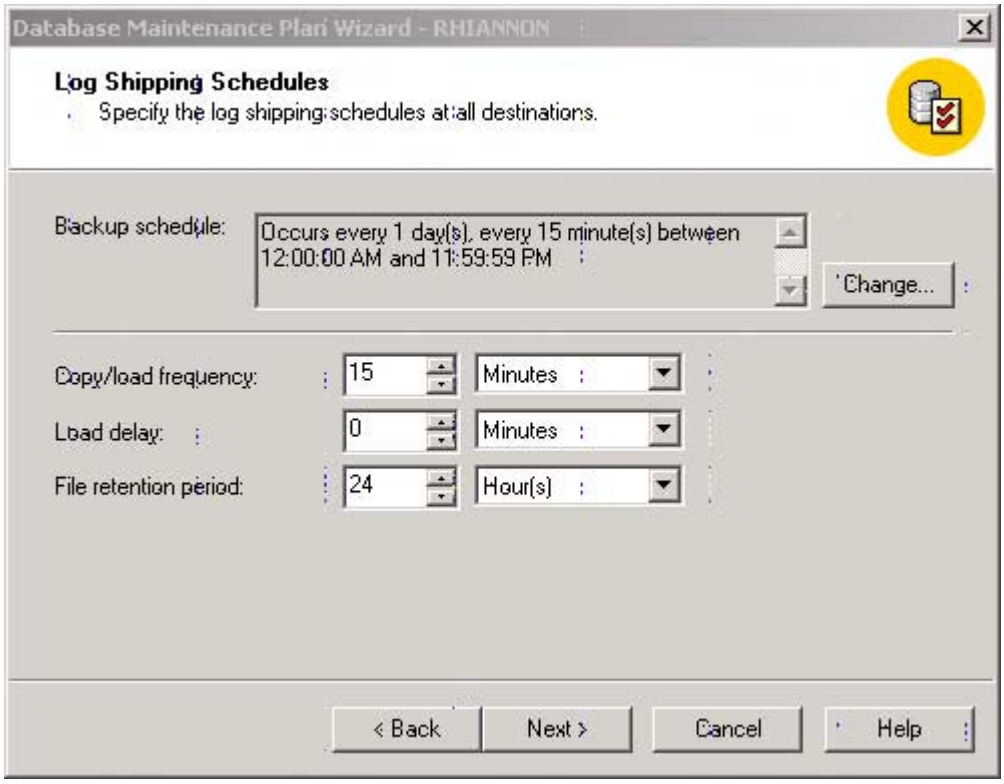
Click Add to specify the log shipping destination server.



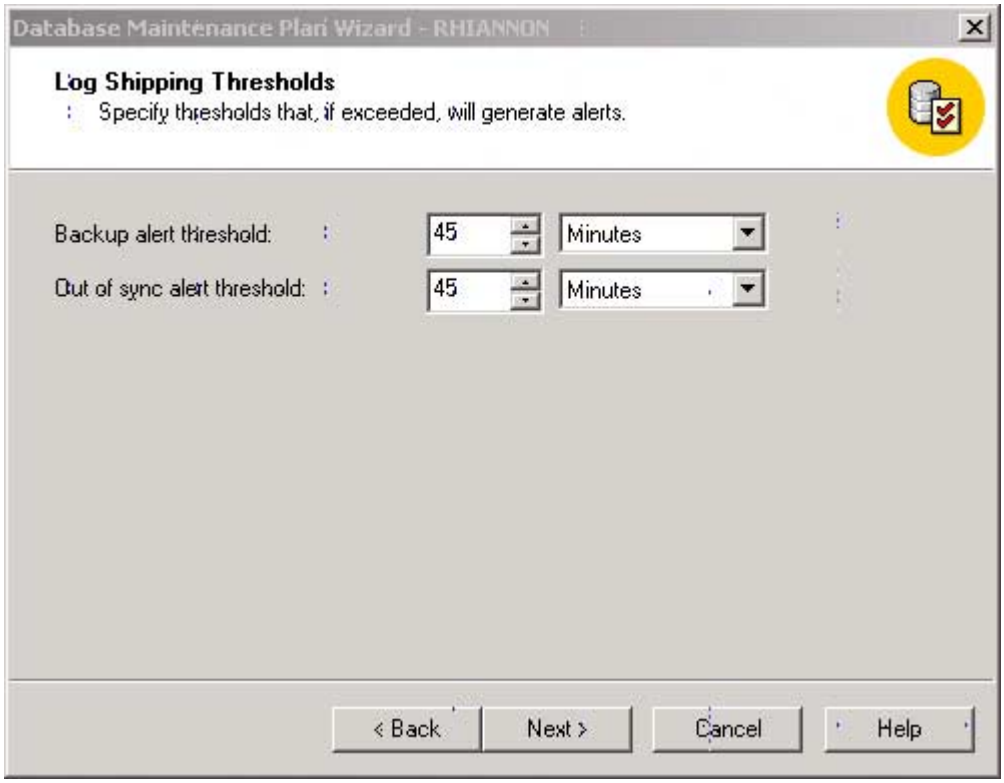
Enter the server name for the destination of the transaction logs. Here an existing database was used to receive the transaction logs. Click OK.



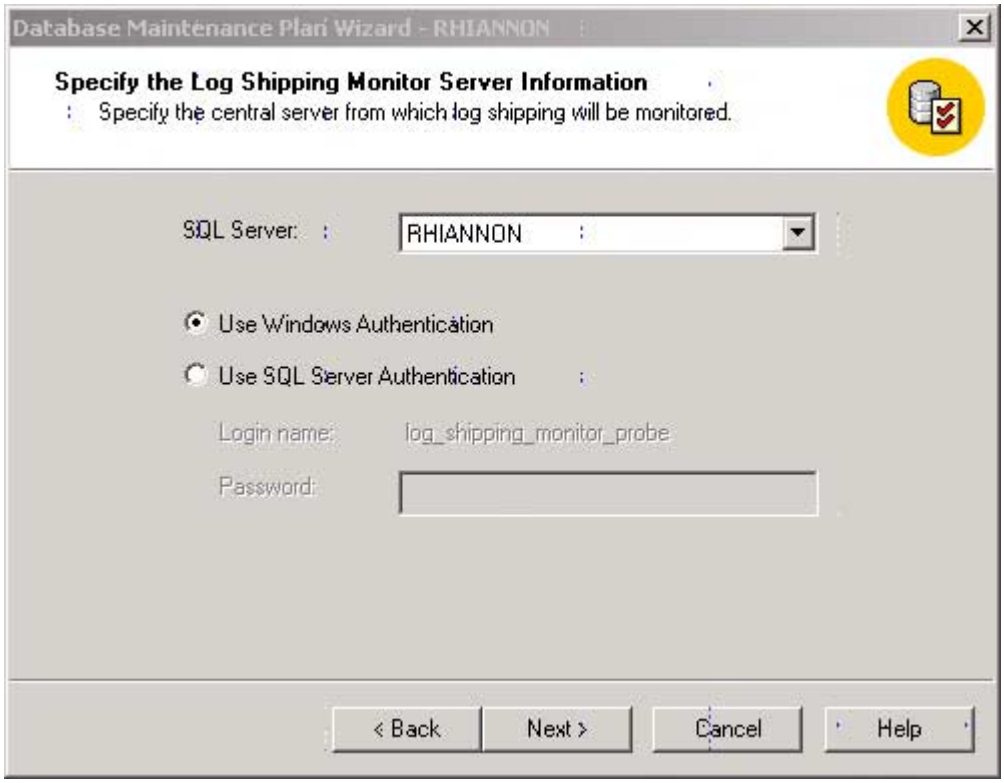
Confirm that the destination server and database are correct and click Next>.



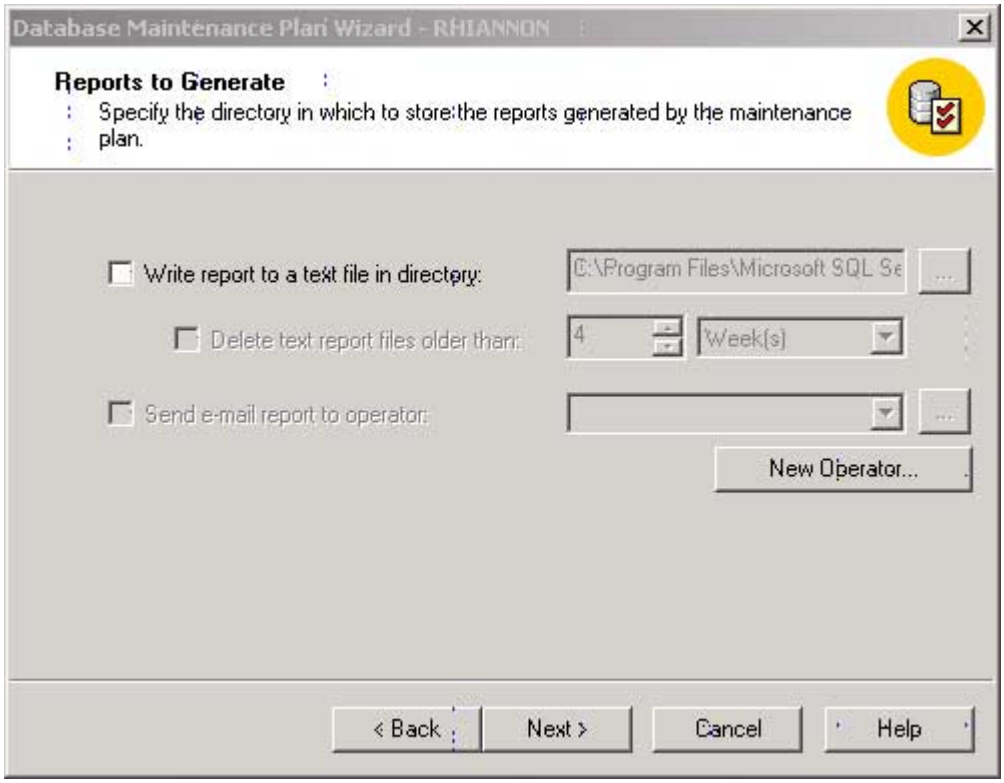
Enter the backup schedule for the log shipping and click Next>.



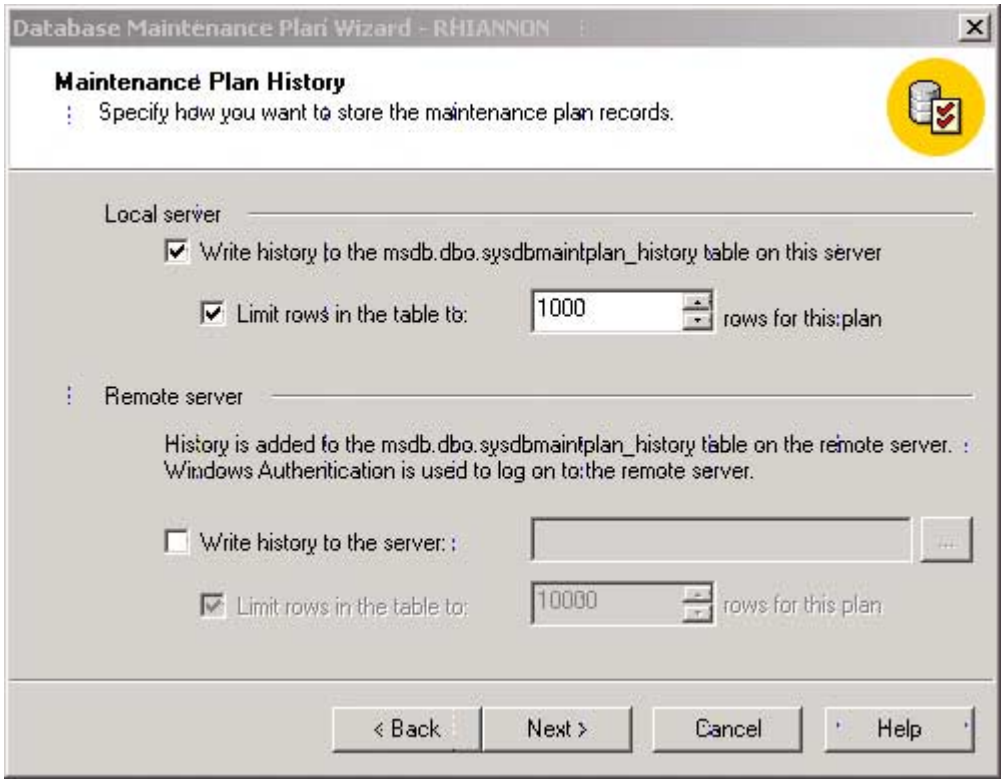
Specify the log shipping thresholds and click Next>.



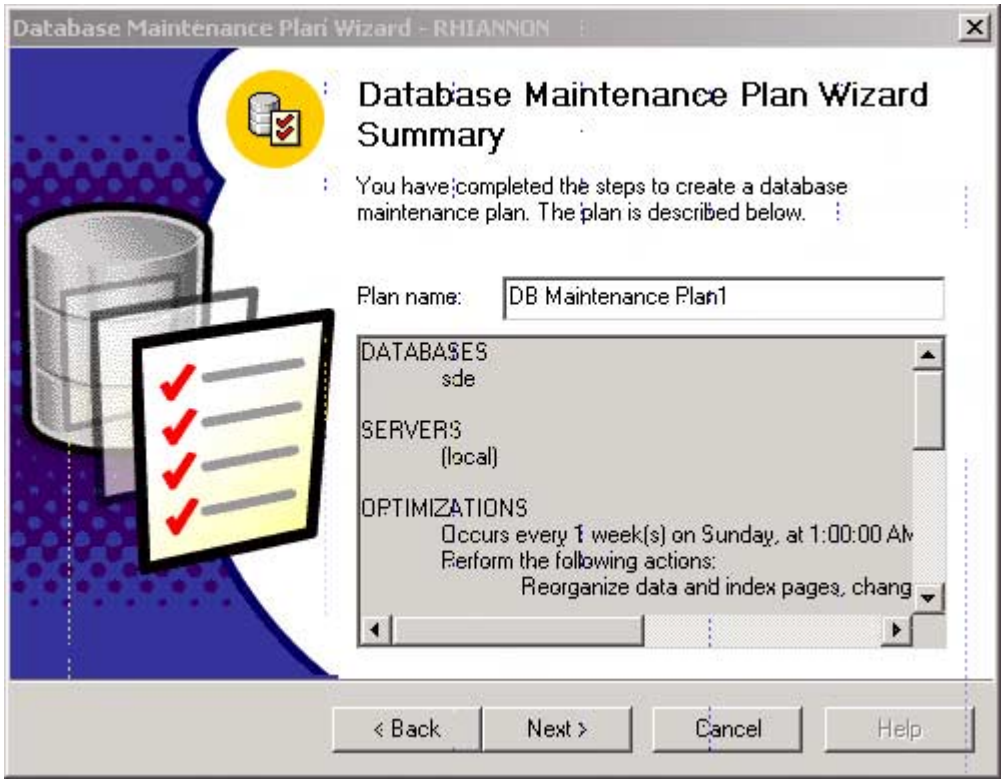
Specify the server to monitor the log shipping. Click Next>.



Specify any reports to generate. Click Next>.



Specify how to store the maintenance plan records. Click Next>.



Name the database maintenance plan. Here the default DB Maintenance Plan1 was accepted. Confirm the options selected and click Next>.

Click Finish to create a database maintenance plan.

It will be necessary to start the SQL Server Agent. In Enterprise Manager, open Management and right-click SQL Server Agent to start. The log shipping will be performed at the times specified in the log shipping schedules. The steps to restore the standby server are the same as above. If the production server should become unavailable, apply all remaining transaction logs to the standby server. From Query Analyzer recover the standby server database without restoring. Execute the following statement with the name of the database to be recovered.

```
-- Restore database using WITH RECOVERY.
RESTORE DATABASE sde
   WITH RECOVERY
```

The standby server will now be in the same state as the production server from the last transaction log. All changes after the last backup of the transaction log will be lost.

CHAPTER 8

Backup and recovery

MS SQL Server allows you to easily move databases between servers. This section describes ways you can distribute your ArcSDE databases through simple backup and restore operations. Other ways to distribute data among servers are replication and standby servers, both discussed in this guide, and data transformation services.

Introduction

MS SQL Server allows you to backup a database from one server and restore it to another. You can also detach and reattach databases between servers. This functionality allows you an easy way of distributing your data to other servers without having to worry about reloading it and rebuilding relationships or networks.

If you restore a database backup from one server to another, the code pages and sort orders of those two servers must be identical. For example, you cannot restore a database built with Binary Sort Order (BSO) to a server configured for default (dictionary) sort order. You can ascertain your server's sort order by running the stored procedure 'sp_helpsort' in the master database.

In this example, this server is configured for default, or dictionary, sort order.

```
Server default collation
Latin1-General, case-insensitive, accent-sensitive, kanatype-insensitive,
width-insensitive for Unicode Data, SQL Server Sort Order 52 on Code Page
1252 for non-Unicode Data
```

A backup operation also backs up the database's system tables including sysusers. When you apply a backup to a different server, the users listed in the database may differ from those users listed in the new server's sysxlogins table in the master database. This will prevent you from logging into the restored database until you correct the login mapping between the master database's sysxlogins table and the restored database's sysusers table. To synchronize the two tables, run sp_change_users_login.

Moving data using backup and restore

This section assumes you are going to backup an ArcSDE database and apply that backup to another server that does not have a preexisting ArcSDE installation.

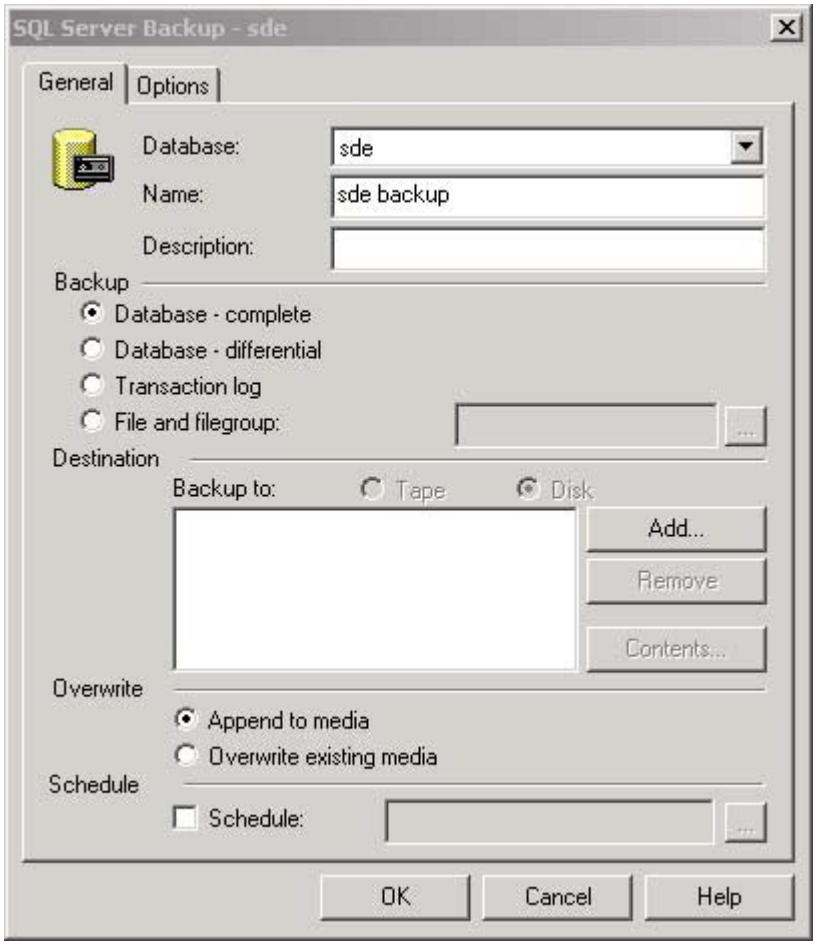
1. Backup the SDE database and any other database you wish to apply to another server.
2. Copy the backup file(s) to the new server.
3. Restore the backup file(s) to the new server.

4. Run sp_change_users_login to synchronize the sysuser's table in each newly restored database with the master database's sysxlogins table.

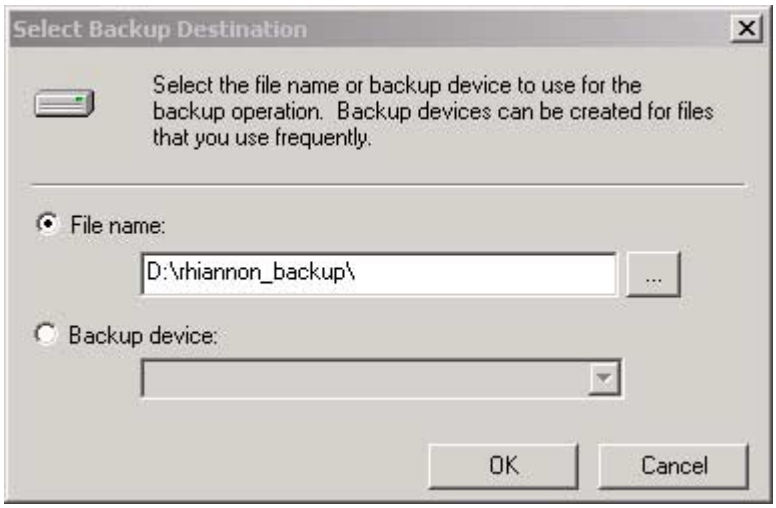
1. Backup the SDE database

Use either t-sql's BACKUP command or the Enterprise Manager's backup utility to backup the sde and any other spatial database you have. In this example, we use the Enterprise Manager to backup the sde database.

Expand Enterprise Manager's Server Group, then your server host name. Right-click your database, select all tasks backup database.



The backup database form will appear. You'll need to add a destination with the 'add' button if none appears. Click Add and choose a directory to backup to.



Click OK when you are done. See the SQL Server Books Online for exact information on backup types and options.

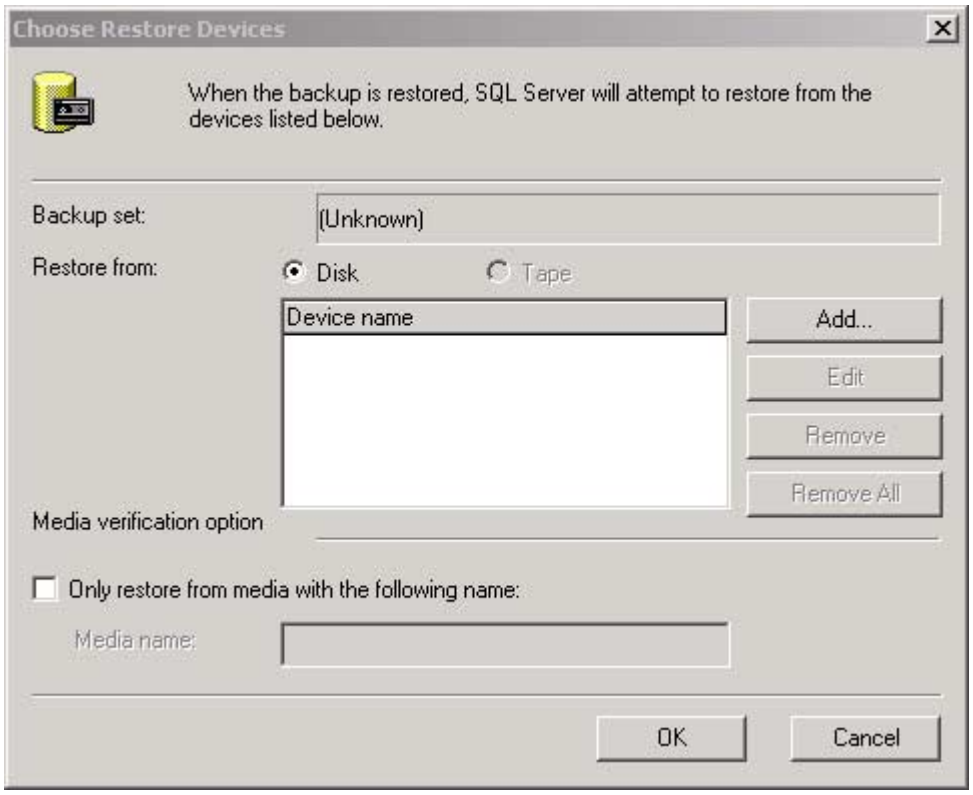
2. Copy the backup file(s) to the new server

In the example above, the backup file is located on D:\rhiannon_backup. Copy the backup file to the new server.

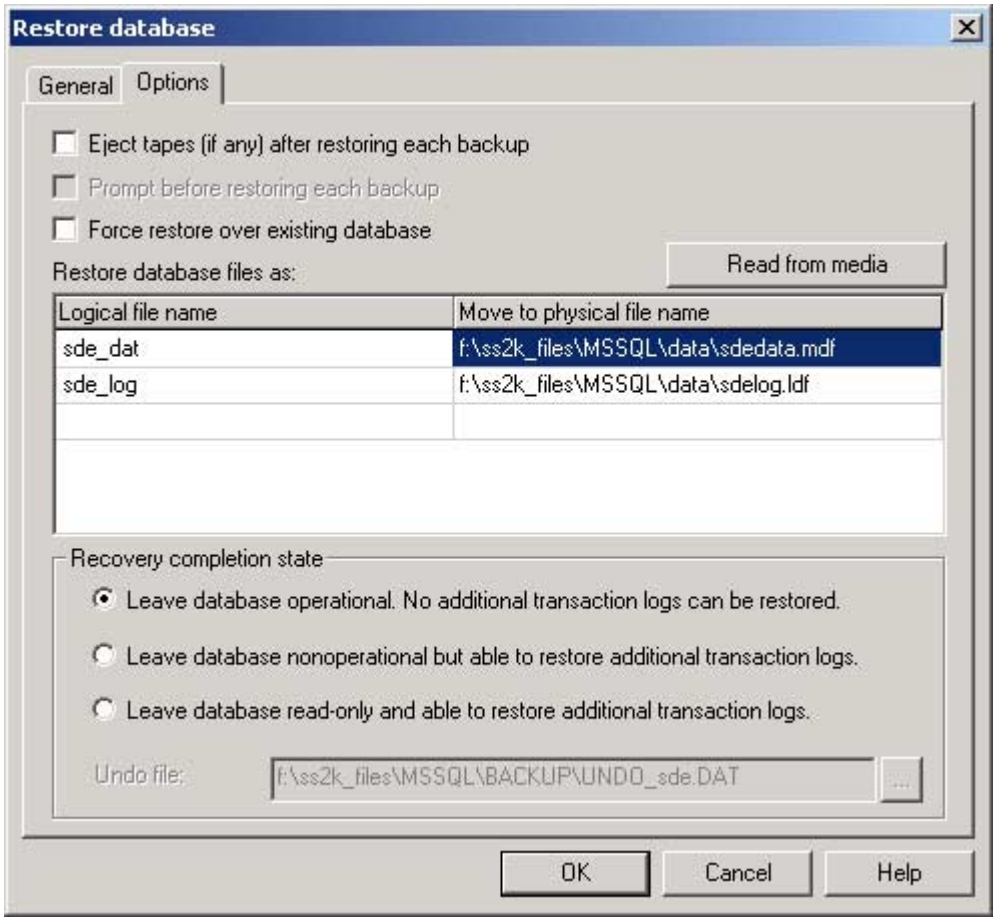
3. Restore the backup file(s) to the new server

Expand the new server in the Enterprise Manager's server group. Right-click 'databases' and select all tasks restore database.

The Restore database dialog box will appear. Click the From device radio button and then the “Select Devices” button.



If you want to restore the database with a different name, change its name in the Restore as database combo box. NOTE: If you do this, you may cause inconsistencies in the sde or gdb metadata tables. Navigate to the backup file and select it. Click OK all the way back to this form and select the Options tab.



In the middle of this form, you'll see a grid control with paths in it. You'll want to verify that the paths are correct for your restore file. You can restore these files to any location you like, as long as it exists on disk. If you check Force restore over existing database, you'll restore this database over an existing one. If you don't and you have an existing database with the same name, the restore will fail. Click OK.

Once the restore finishes, open the database in the Enterprise Manager and verify your tables are there. Now you must synchronize the database users with their logins located in master.dbo.sysxlogins.

4. Run sp_change_users_login

When you restore a database from a backup created on another machine, there is a chance that a database user's identification number (either an SID or a GUID) does not match the same entry in the master.dbo.sysxlogins table. There is also a chance that the database user does not exist in the master.dbo.sysxlogins table. Since a restore also restores a database's sysusers table, it is possible for a record to exist here but not be present in master.dbo.sysxlogins. Hence, there are two scenarios you need to consider:

- The user exists as a login on server but has a different identification (SID or GUID).

- The user does not exist (added as a login) on the server.

Scenario 1: The user exists on both servers but has a different identifier

In this example, both servers have a login called 'sde.' After you restore the sde database onto the new server, the sde user's SID differs in the sde database from the sde login's SID in the master.dbo.sysxlogins table.

```
Use sde
go
Select SID from sysusers where name = 'sde'
Use master
go
select SID from sysxlogins where name = 'sde'

SID
------------------------------------------------------------
0x76695419BFAED41184FD00C04F8D0451

(1 row(s) affected)

SID
------------------------------------------------------------
0xEDDFCA8E56B0D411850000C04F8D0451

(1 row(s) affected)
```

To synchronize the two accounts, run sp_change_users_login. For exact usage, see the SQL Server Books Online. In this example, we use the "update_one" option to update just the sde user. You must run sp_change_users_login within the database you restored (or want to change the login in).

```
Use sde
go
sp_change_users_login 'update_one','sde','sde'
```

Rerun the SID query above to verify your results:

```
0xEDDFCA8E56B0D411850000C04F8D0451
0xEDDFCA8E56B0D411850000C04F8D0451
```

Scenario 2: The user does not exist on the destination (restore) server

This scenario differs from the first one in that a user exists in the database but not as a login in the master.dbo.sysxlogins table. This can happen when you create a login on a server, backup the databases on that server, then restore the databases to another server that does not have a record for this login. In this example, the sde database is restored onto a machine that never had an 'sde' login.

```
Use sde
go
Select SID from sysusers where name = 'sde'
Use master
go
select SID from sysxlogins where name = 'sde'

SID
------------------------------------------------------------
0x76695419BFAED41184FD00C04F8D0451

(1 row(s) affected)

SID
------------------------------------------------------------

(0 row(s) affected)
```

To rectify this, you must add an ‘sde’ login to the server but not grant them access to the sde database. To add a login, use the sp_addlogin stored procedure or use the logins tool with the Enterprise Manager:

```
sp_addlogin ‘sde’,’go’,’sde’
```

Now run sp_change_users_login to synchronize the SID values in master.dbo.sysxlogins and the restored database’s sysusers table. You must run sp_change_users_login in the sde database since its sysusers table is out of sync with the master.dbo.sysxlogins table.

```
use sde
go
sp_change_users_login ‘update_one’,’sde’,’sde’
```

Now verify that the SIDs are equal by rerunning our SID query.

```
Use sde
Go
Select SID from sysusers where name = ‘sde’
Use master
Go
select SID from sysxlogins where name = ‘sde’
SID
------------------------------------------------------------
0xF6DFCA8E56B0D411850000C04F8D0451

(1 row(s) affected)

SID
------------------------------------------------------------
0xF6DFCA8E56B0D411850000C04F8D0451

(1 row(s) affected)
```

Once you’ve synchronized the users, you are ready to either start the ArcSDE service or make direct connections to it.

CHAPTER 9

Configuring snapshot replication

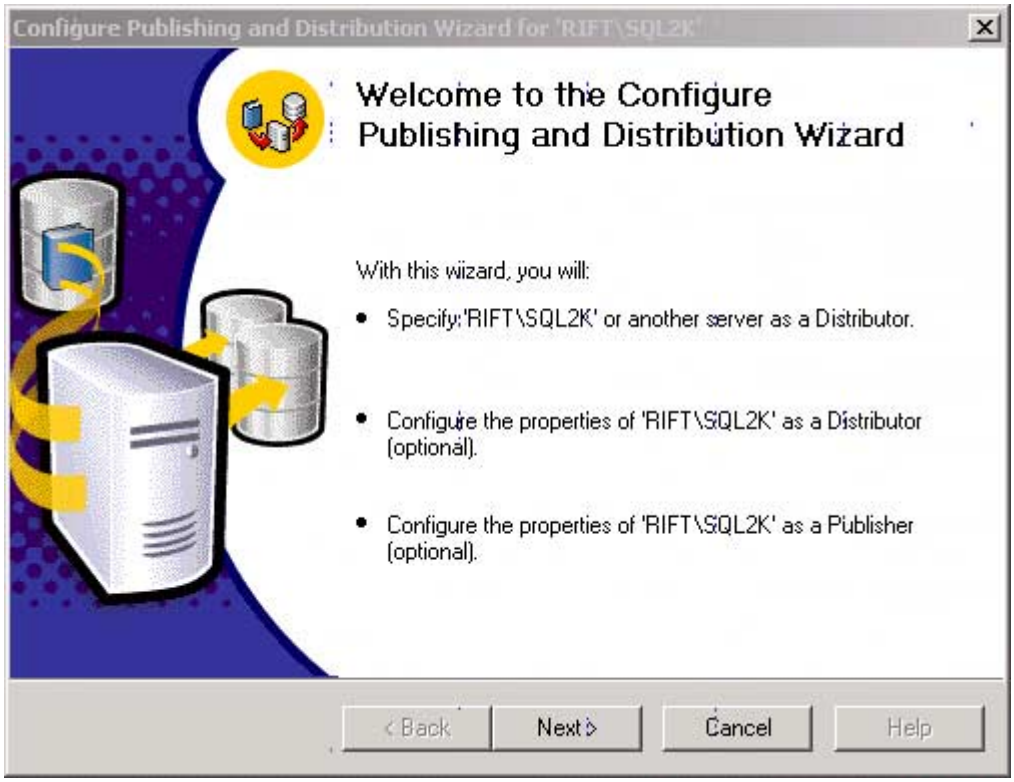
Replication is used to create multiple copies of the same data that can be distributed to multiple other databases throughout an organization.

Implementing snapshot replication

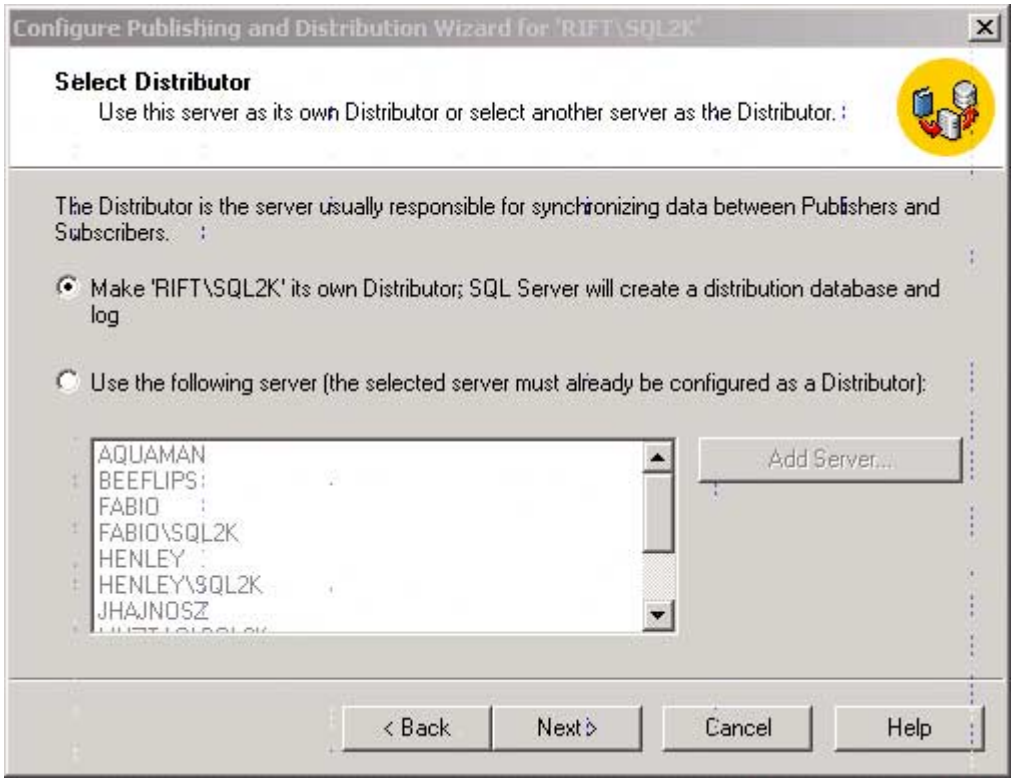
Snapshot replication is used to distribute an entire copy of the information from the distributing database to the subscribing database. In SQL Server 7 or SQL Server 2000, it is the easiest type of replication to setup and maintain. It is most appropriate for read-only data, and there is a high degree of latency in this process. Any changes that are made to the data at the subscriber will be overwritten once the next snapshot image is downloaded from the distributor. Management of the distribution of the snapshot can be performed from the distribution server, a push subscription, or from the subscription server, a pull subscription. The use of a push or pull subscription is dependent on the particular circumstances of each organization. A push subscription can be centrally managed but will require a greater storage premium on the distribution server.

Implementation of snapshot replication, in this example replication, will be between two instances of SQL Server 2000. The procedure will be similar for SQL Server 7.

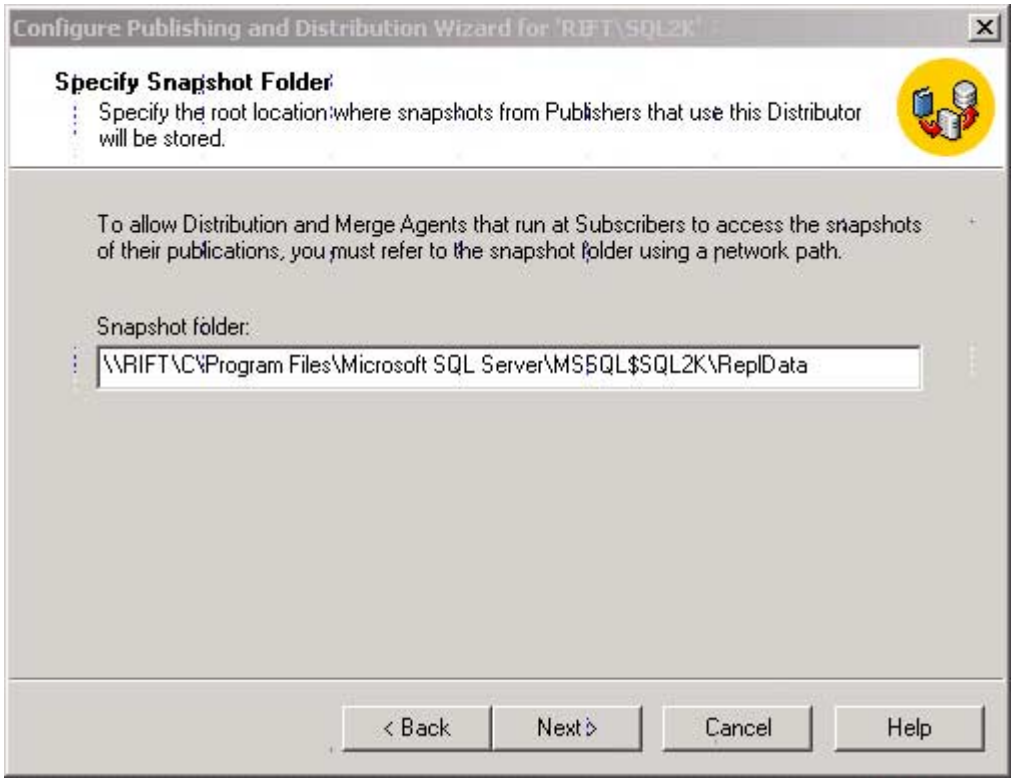
From Enterprise Manager, for the SQL Server instance that you wish to replicate from, choose TOOLS; Replication; and Configure Publishing, Subscribers, and Distribution. This will begin the process of configuring the distribution server.



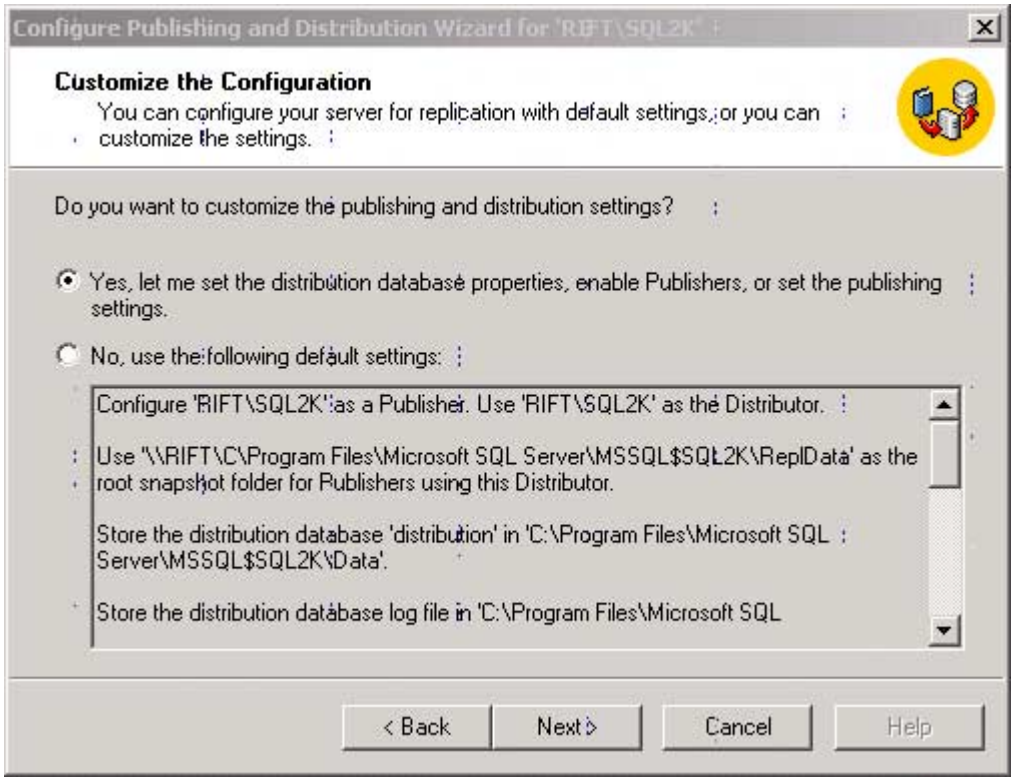
Click Next>.



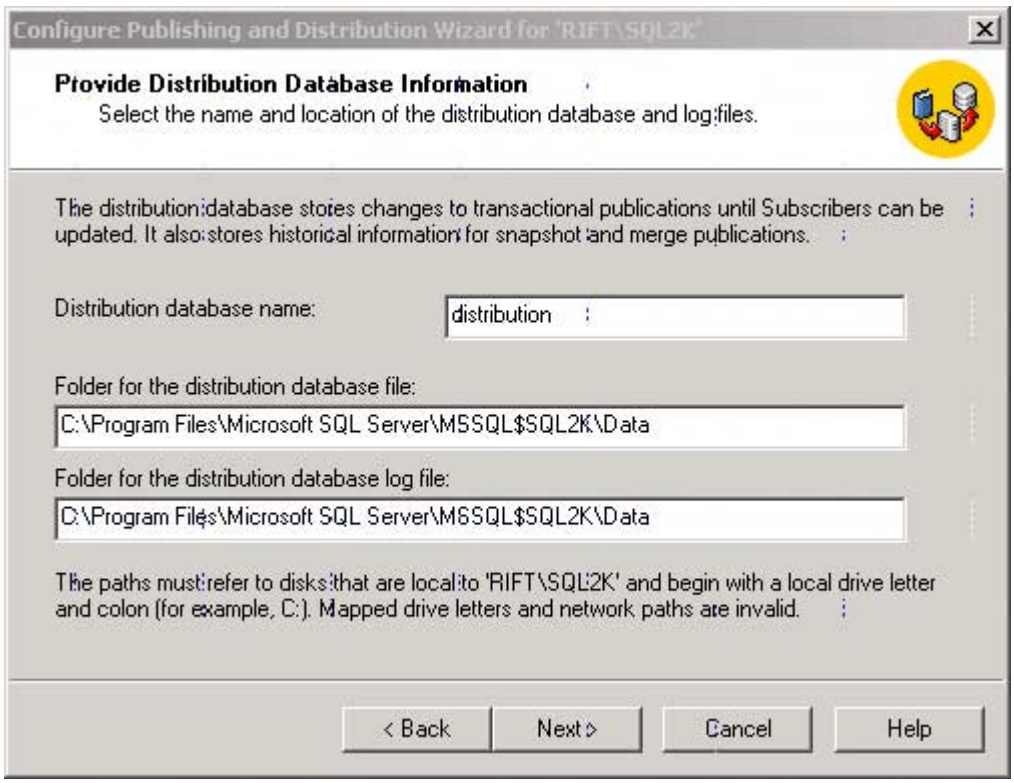
Choose the server to act as the distributor, then click Next>.



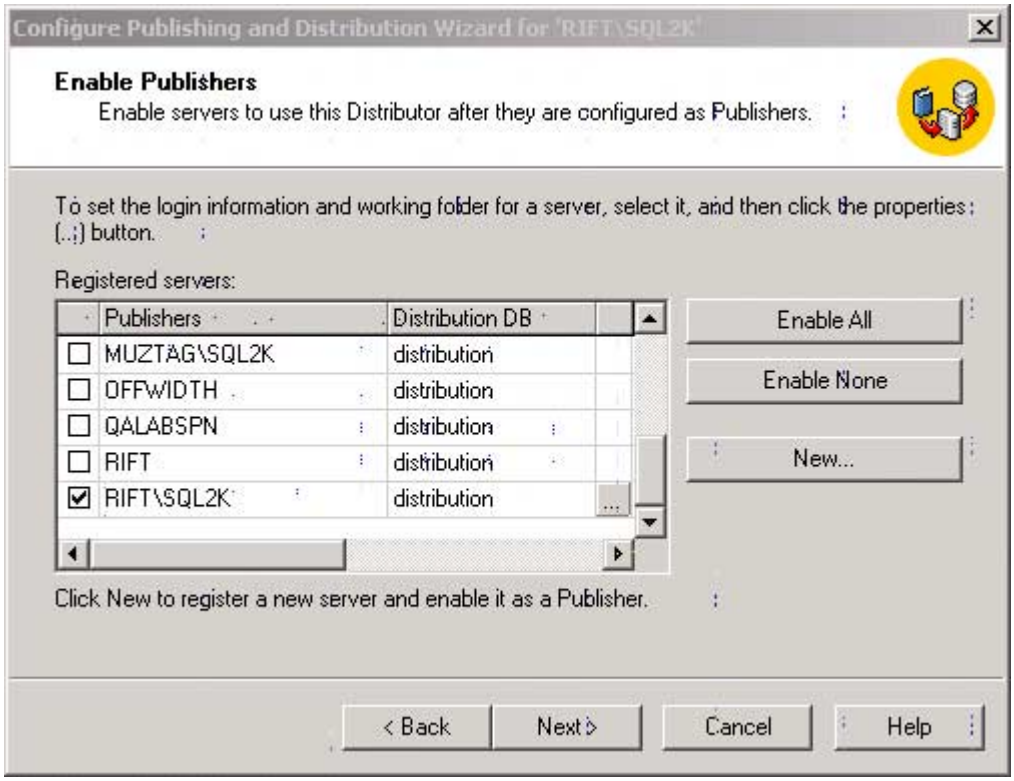
Specify the path to the replication snapshot folder, then choose Next>.



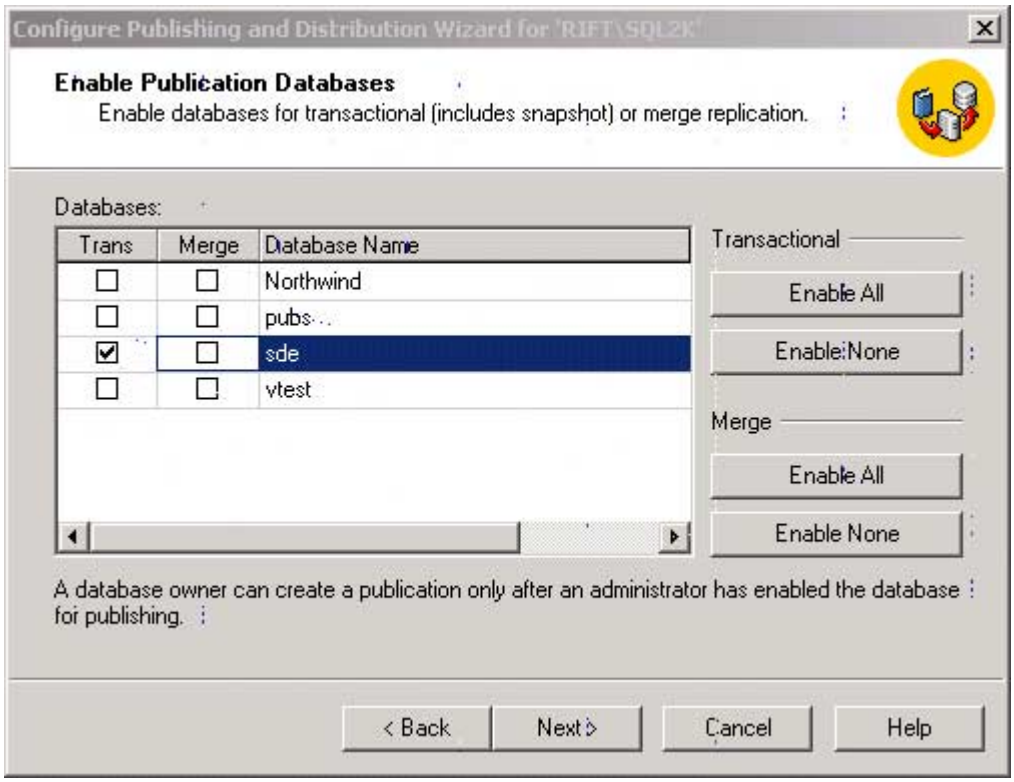
You may accept the default configuration from SQL Server, but for this example we will choose to customize the publishing and distribution settings. Click Next>.



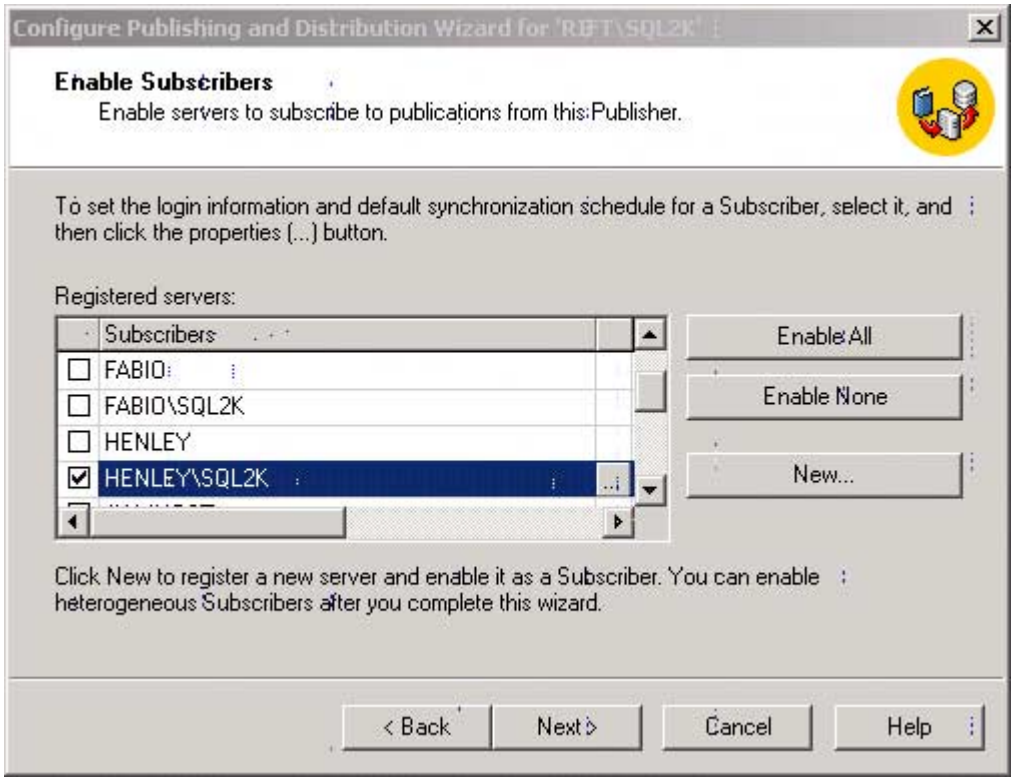
Select the name and location of the distribution database and logfiles. Then choose Next>.



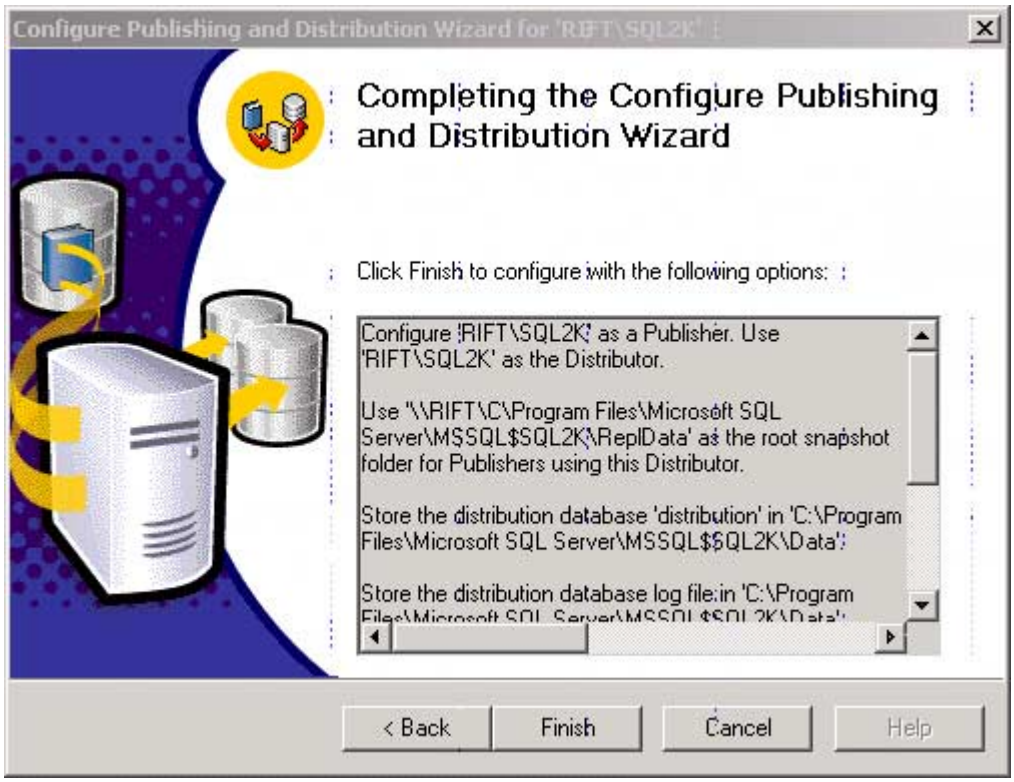
Enable the servers to use this distributor after they are configured as Publishers. Then choose Next>.



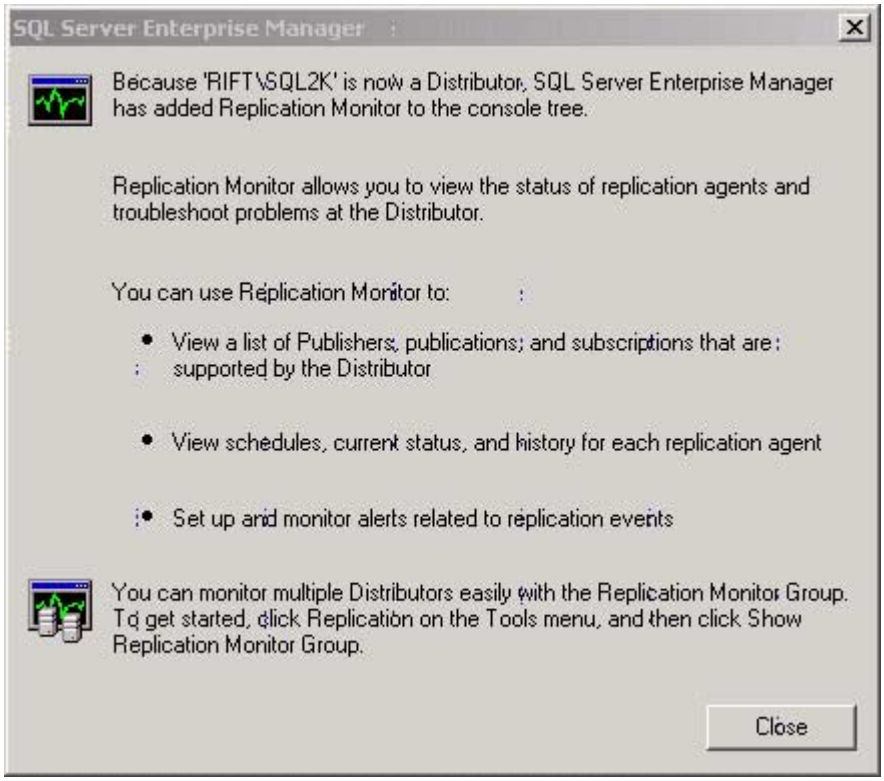
Enable the databases that are to be used for replication. Here choose Transactional, which includes snapshot. Choose Next>.



Enable the subscribing server to receive publications from the distributor. Then choose Next>.

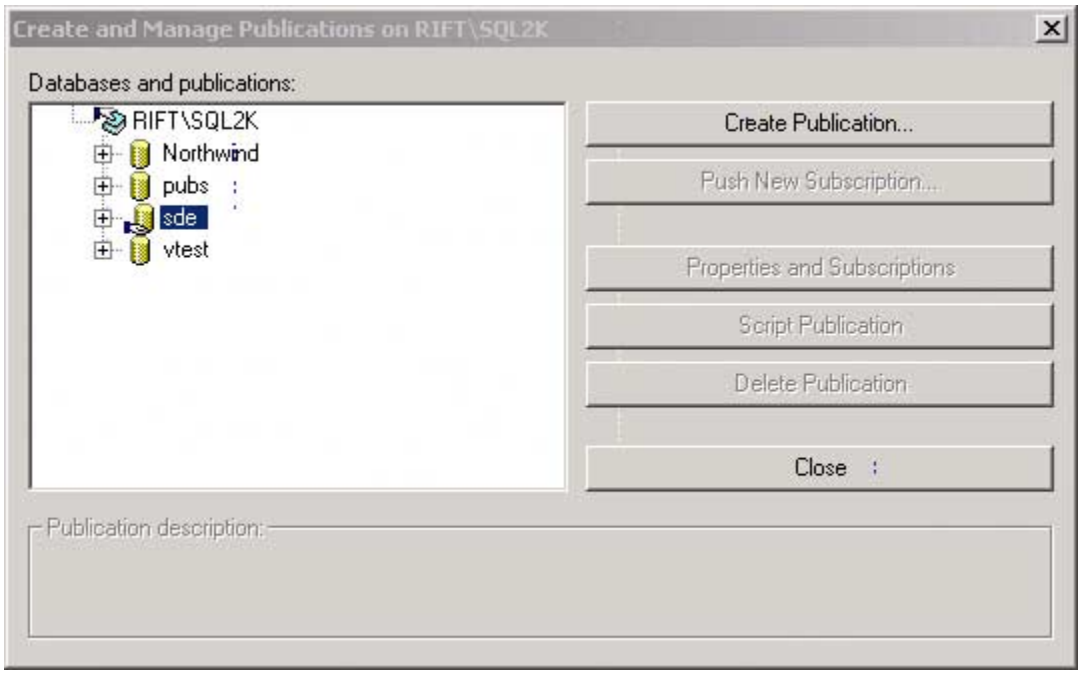


Choose Finish to complete the configuration of the distribution server.

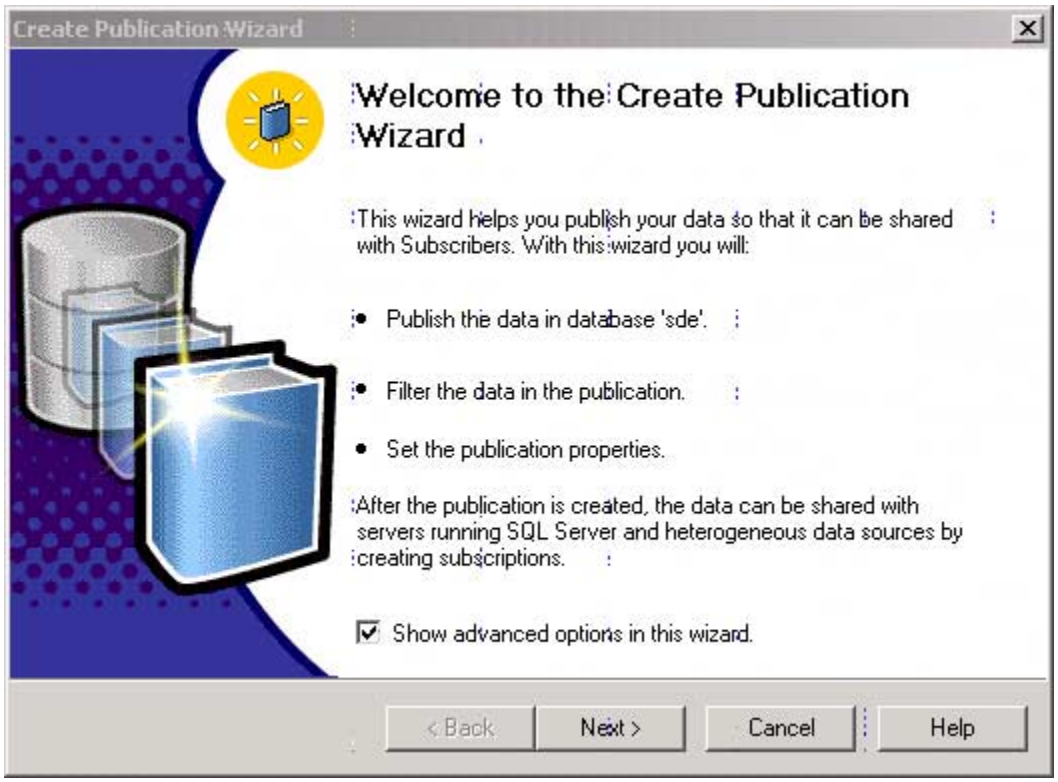


Choose Close to complete the distribution configuration process.

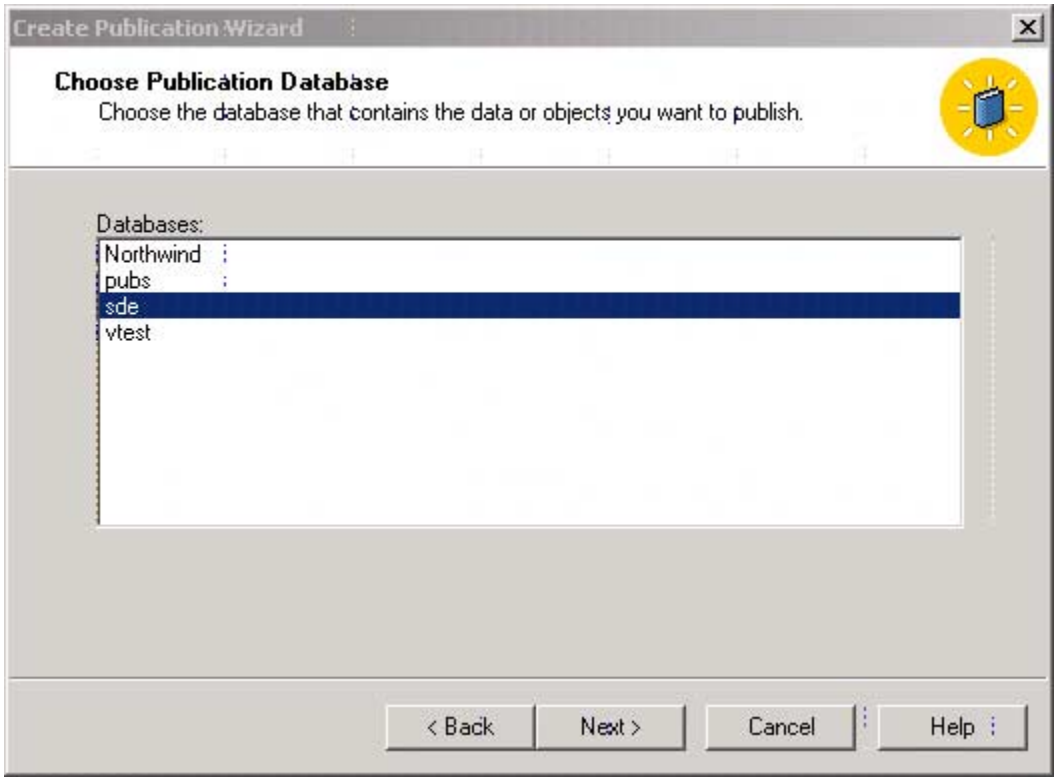
To create and manage a publication in Enterprise Manager, choose Replication, then choose Create and manage publications.



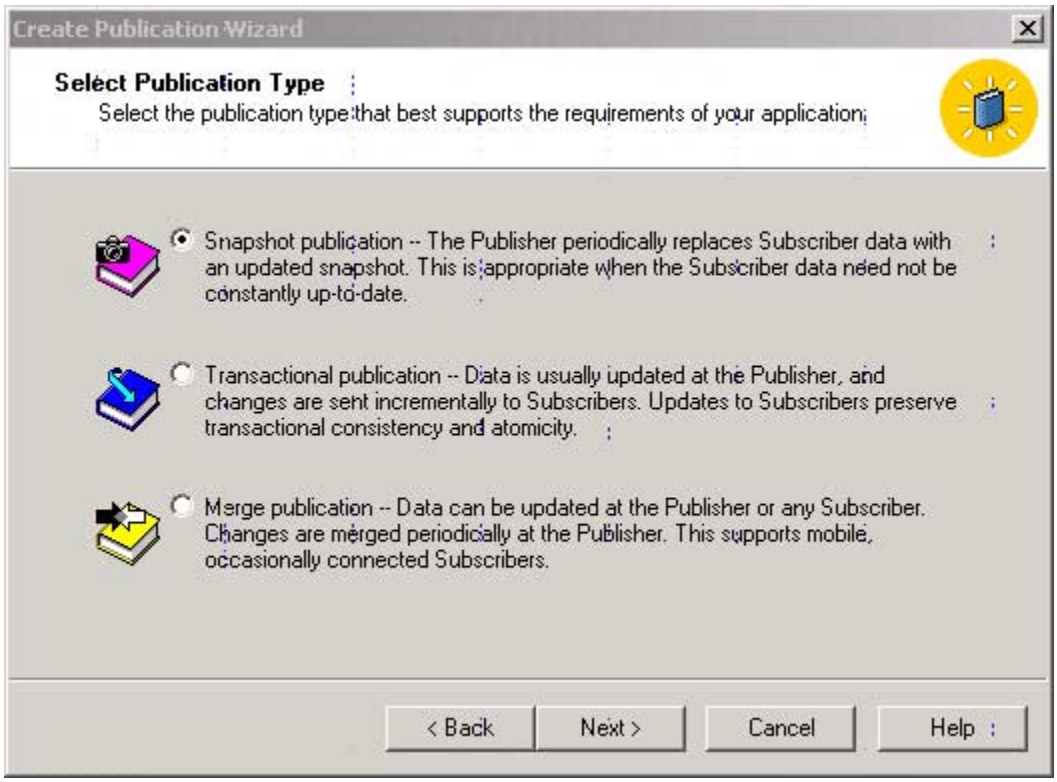
Highlight the source of the publication and click Create Publication.



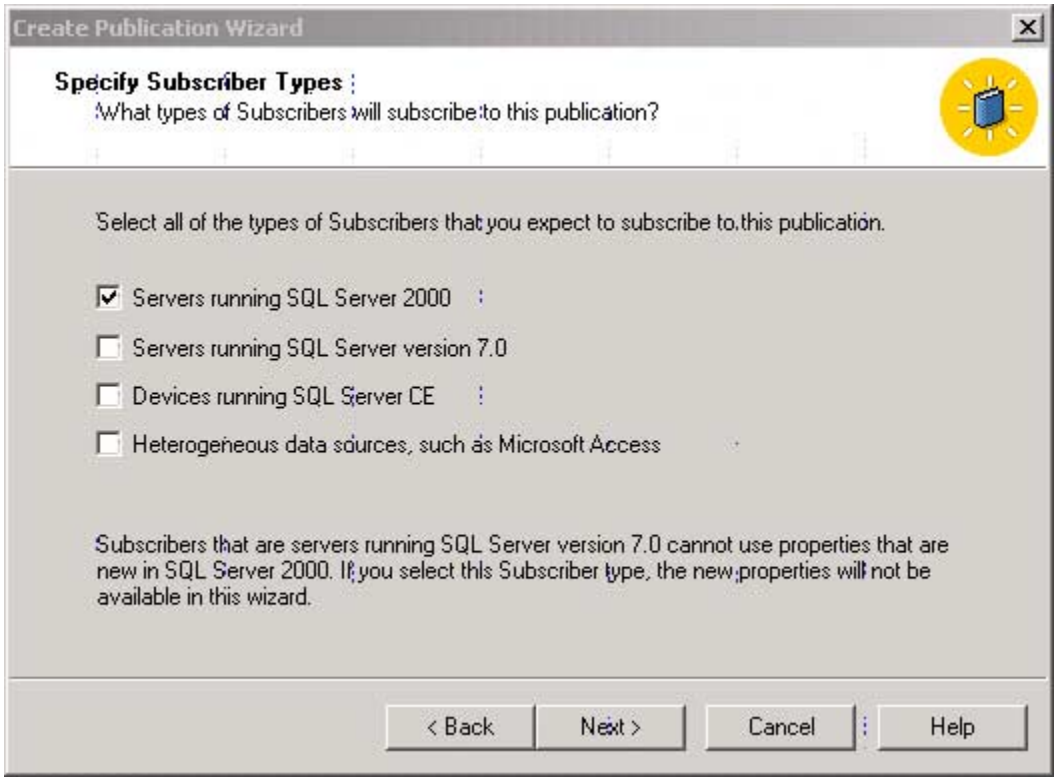
The Welcome to the Create Publication Wizard will appear. We will run this example with the Show advanced options in this wizard configured. Choose Next> to continue.



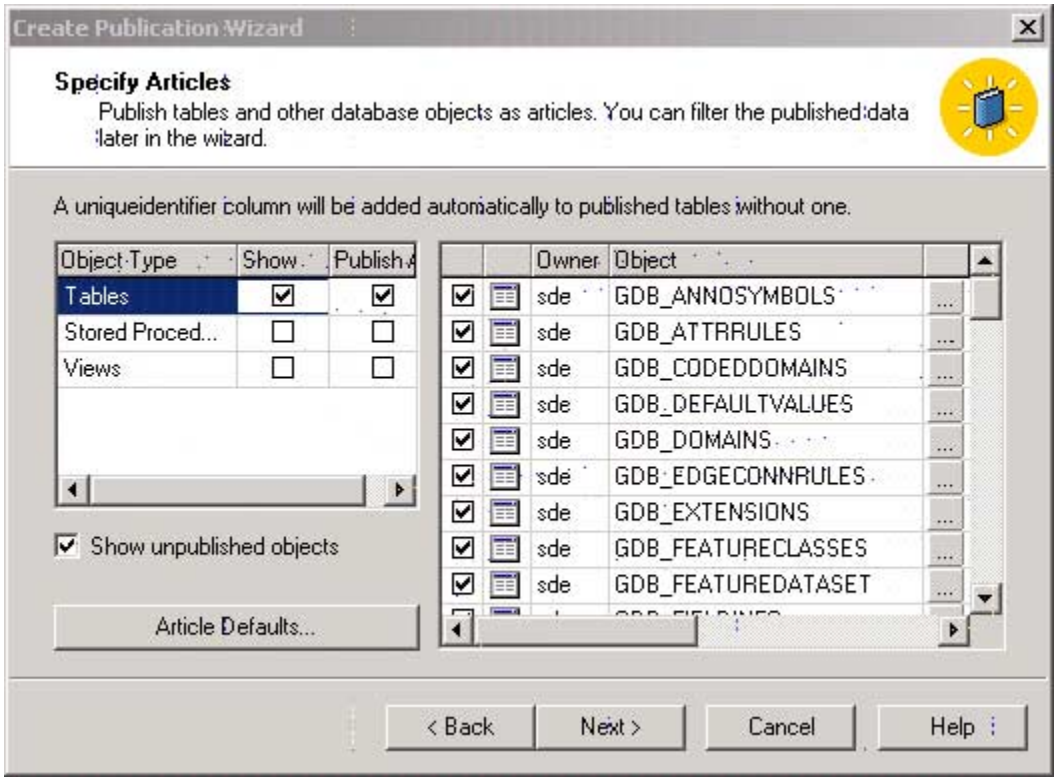
Highlight the database that contains the objects to be published, then click Next>.



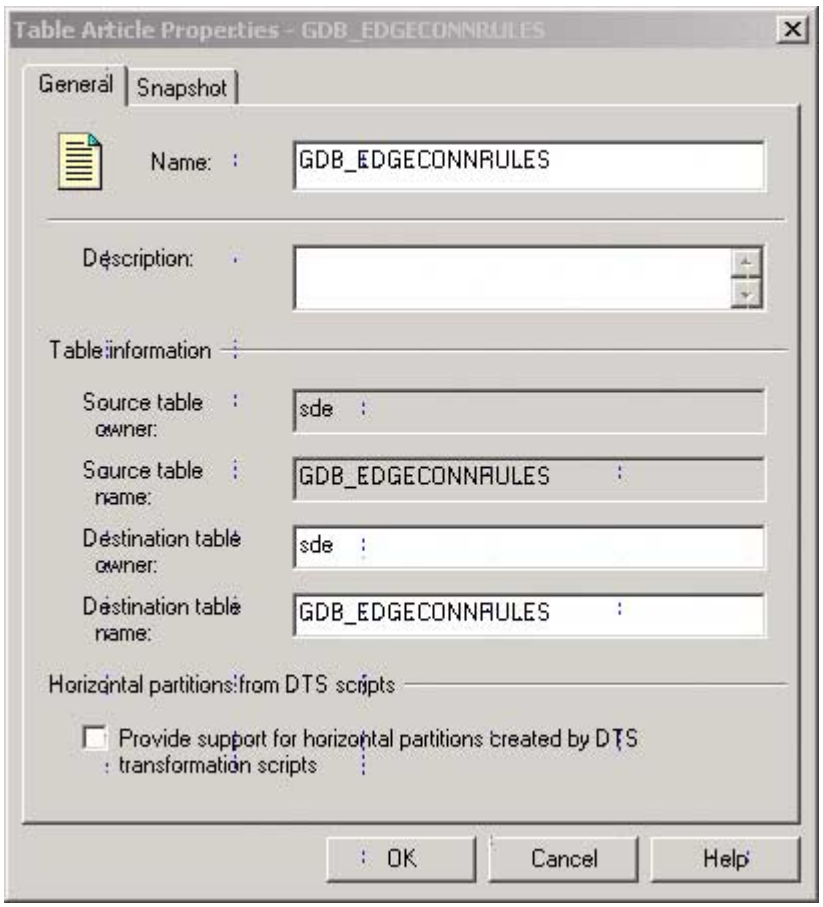
Select Snapshot publication, then click Next>.



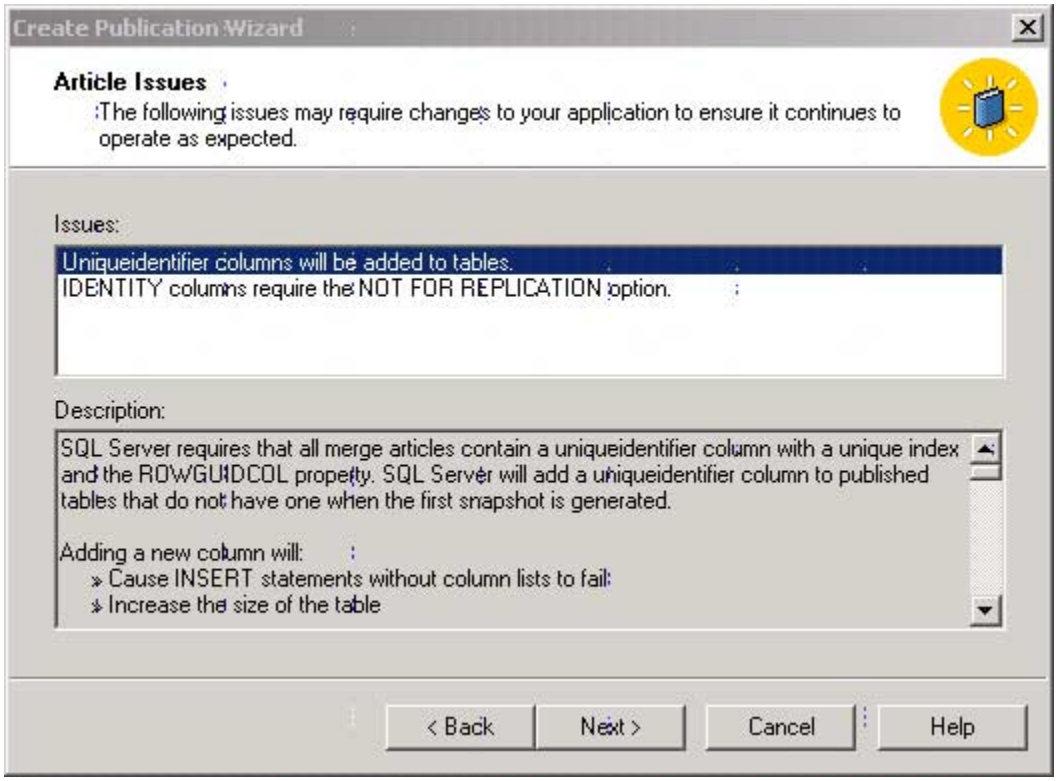
For this example, SQL Server 2000 was run. ArcSDE currently supports SQL Server 2000 and SQL Server 7. Click Next>.



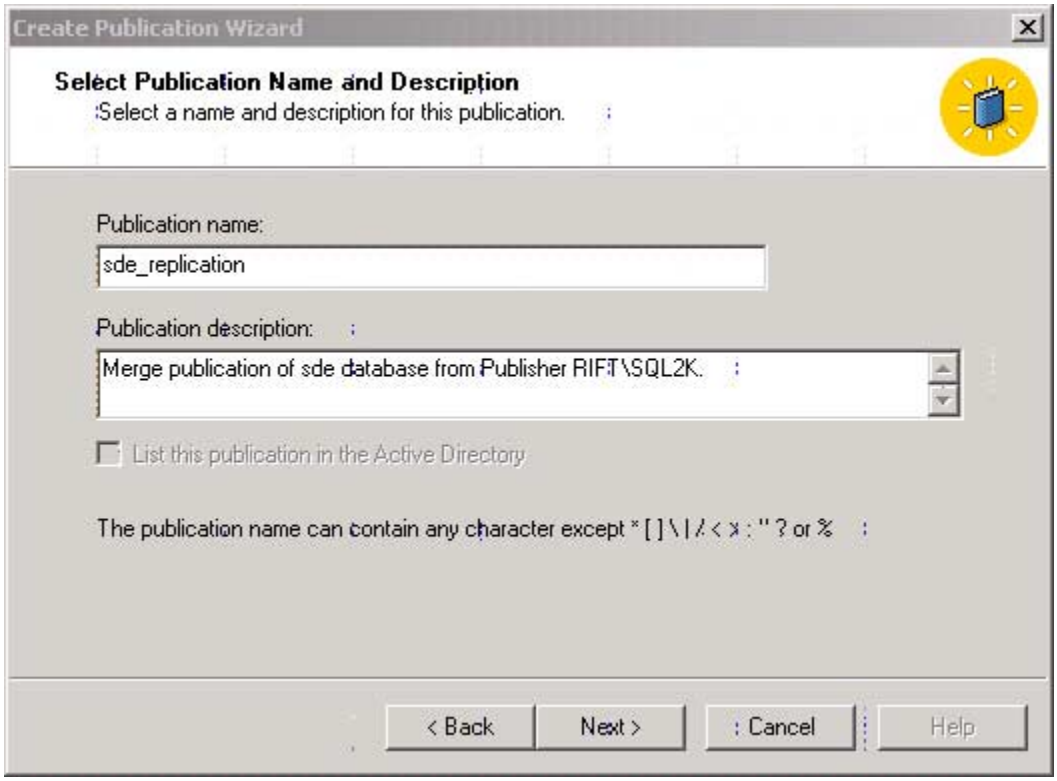
Choose the data to transfer; here all tables in sde will be replicated. It is necessary here to ensure that the tables are also replicated with the correct owner on the subscribing database. For each table that is replicated, click the ellipsis next to the Object and the following screen will appear. Enter the correct owner of the table for the subscribing database.



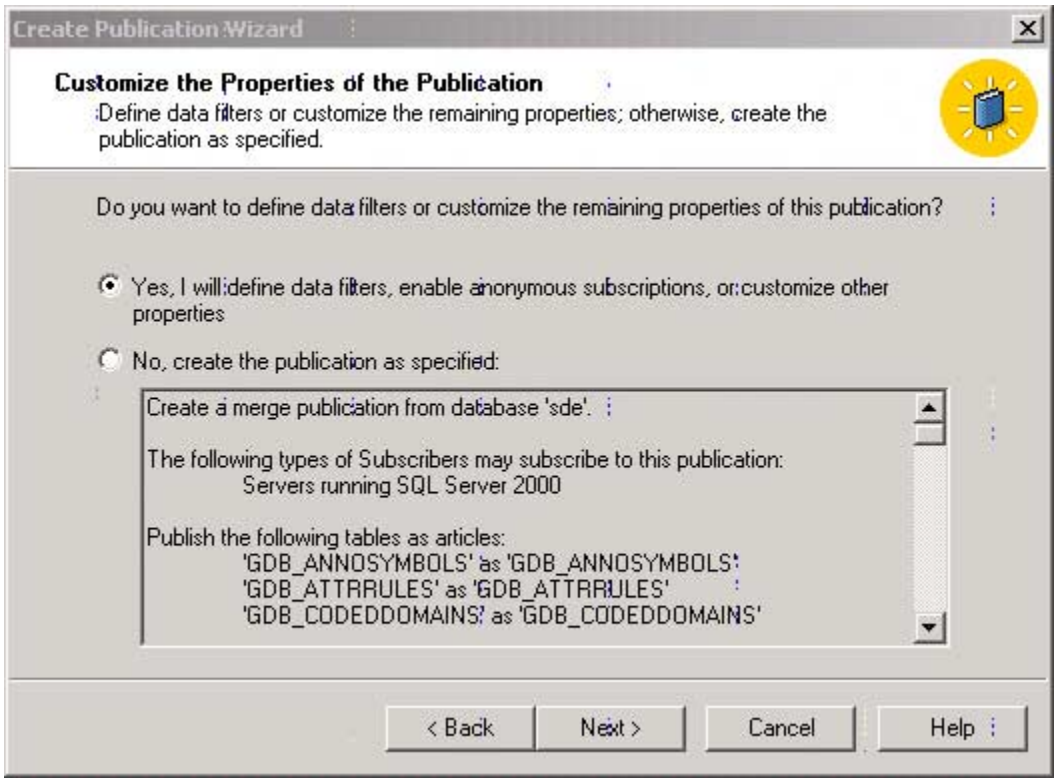
Confirm that the source table owner and the destination table owner are correct, then click OK to return to the previous screen. Click Next> at that screen to continue.



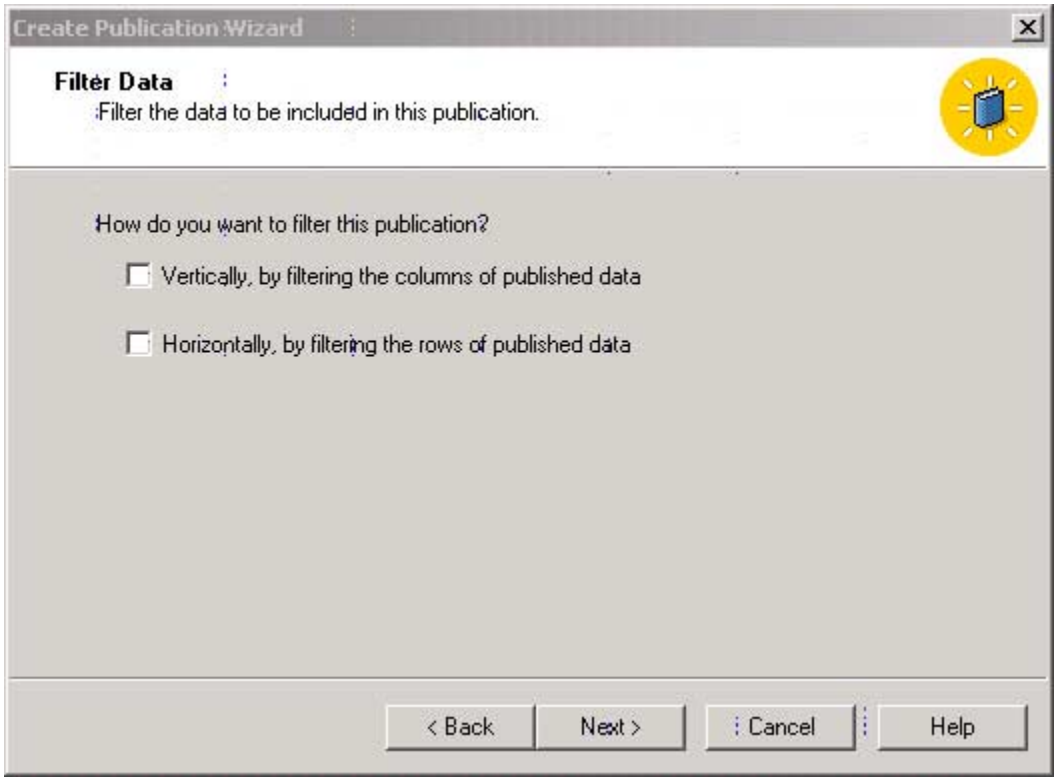
ROWGUIDCOL is a unique identifier that is added to tables used in merge replication. This makes it possible to track the lineage of changes made to a record. Click Next>.



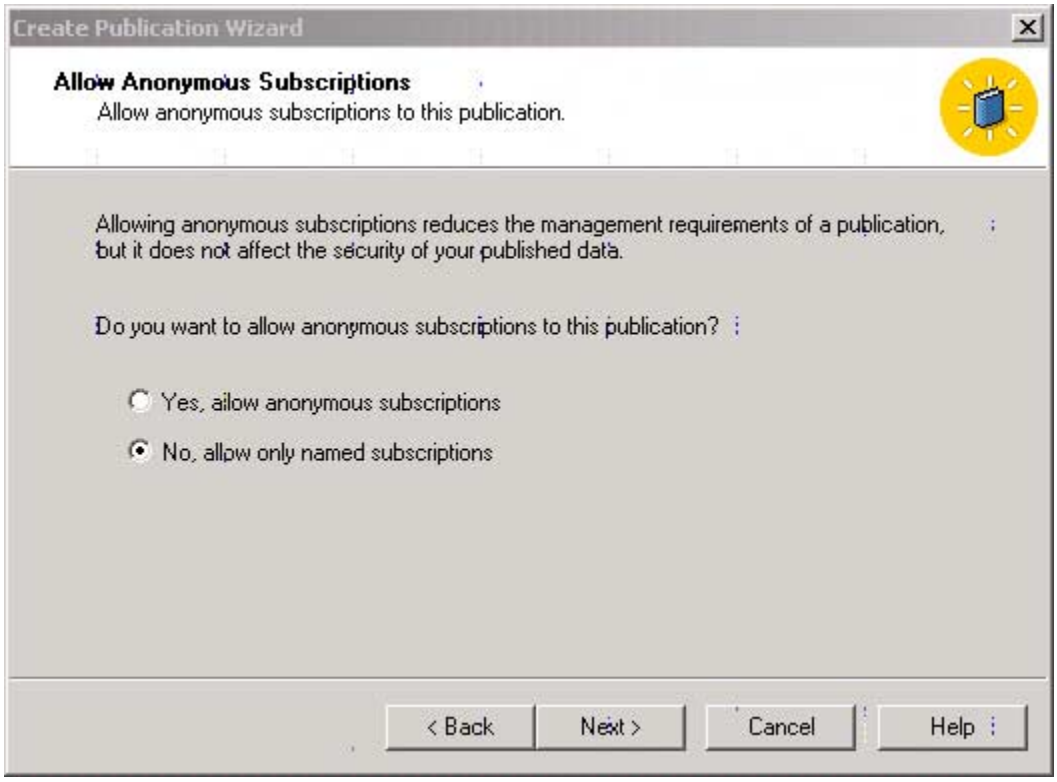
Enter a publication name and publication record and click Next>.



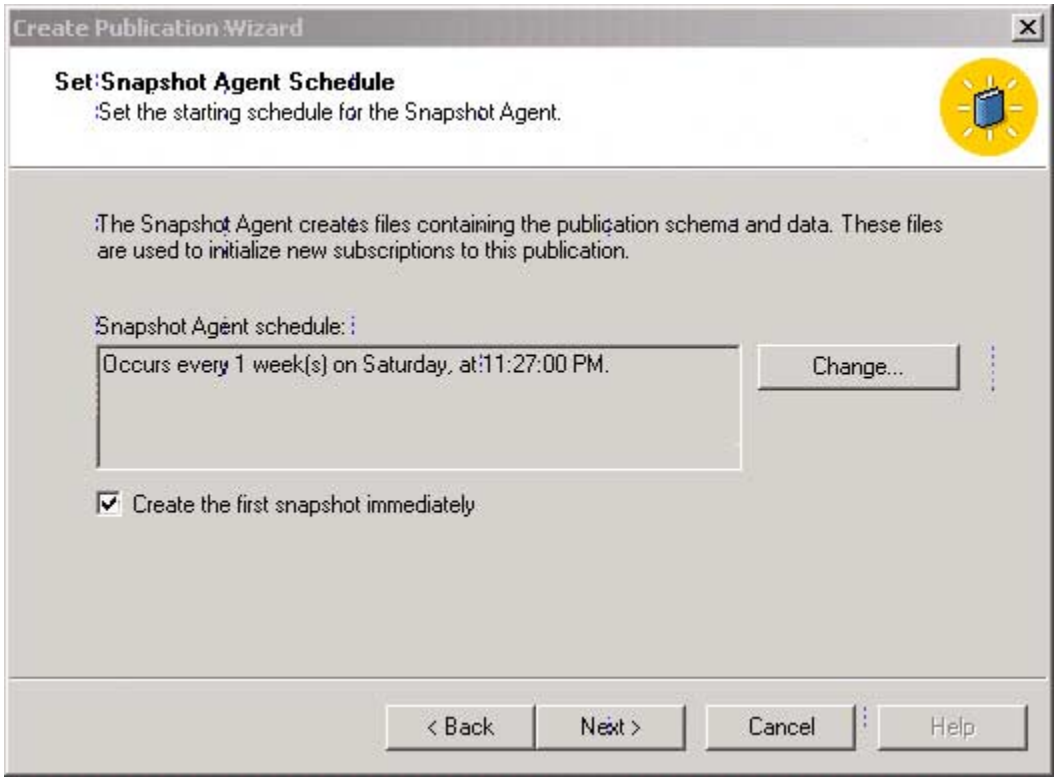
For this example, we will choose to customize the publication properties. Click Next>.



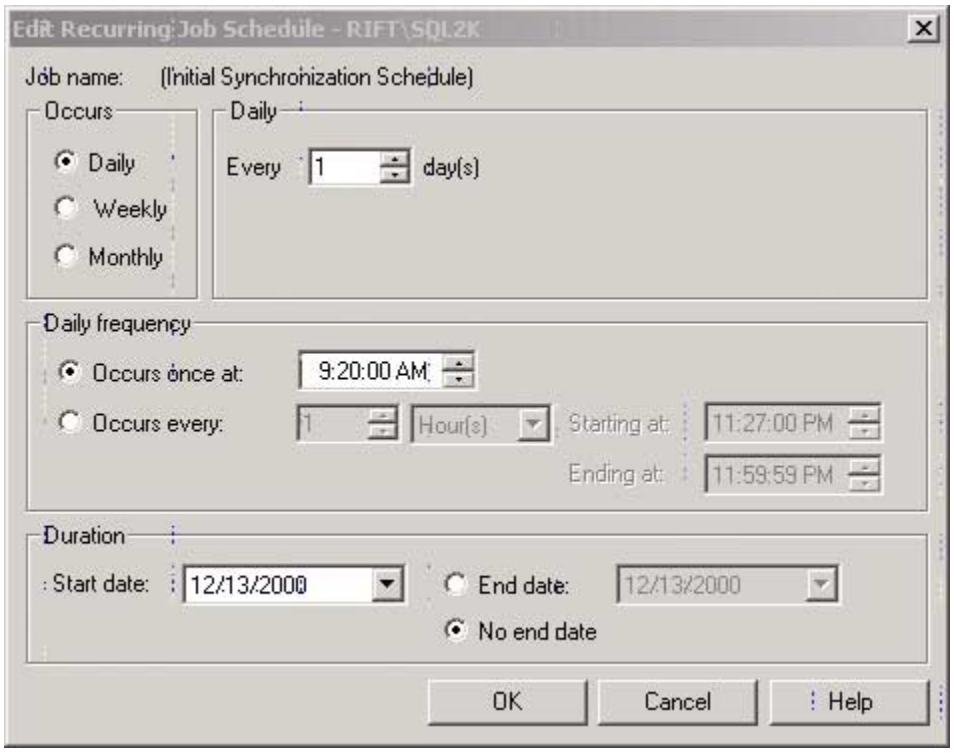
No filtering of this publication will be chosen. Click Next>.



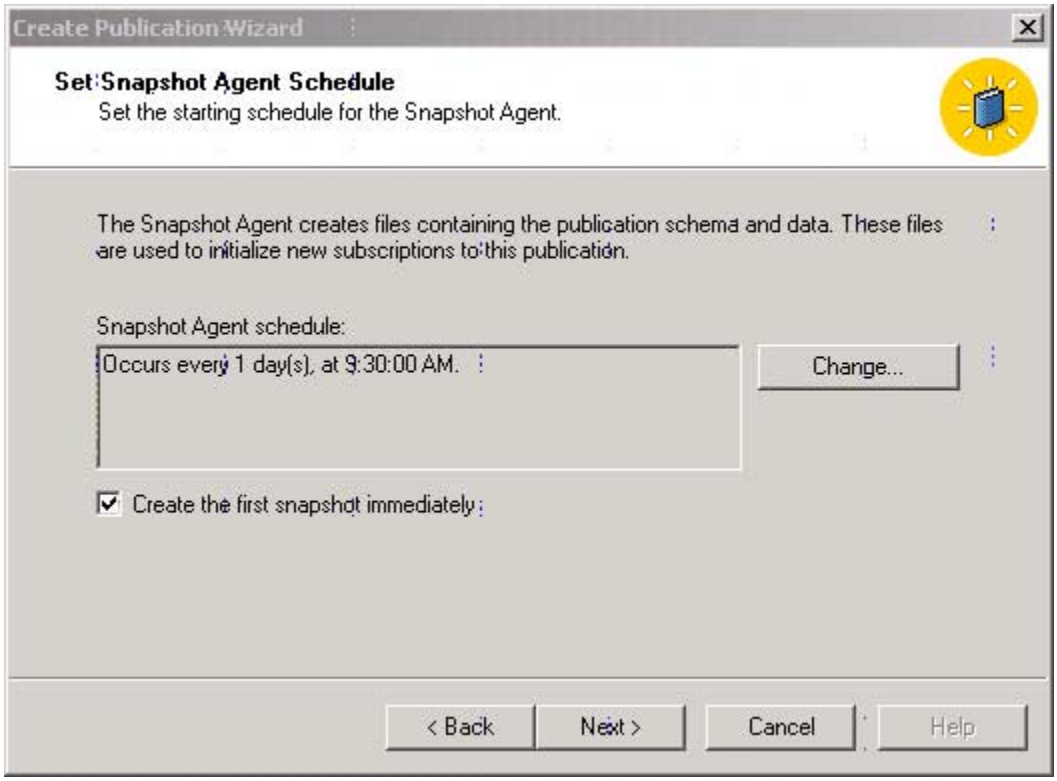
Only named subscriptions will be allowed in this example. Click Next>.



The snapshot agent schedule will be changed. Click Change.

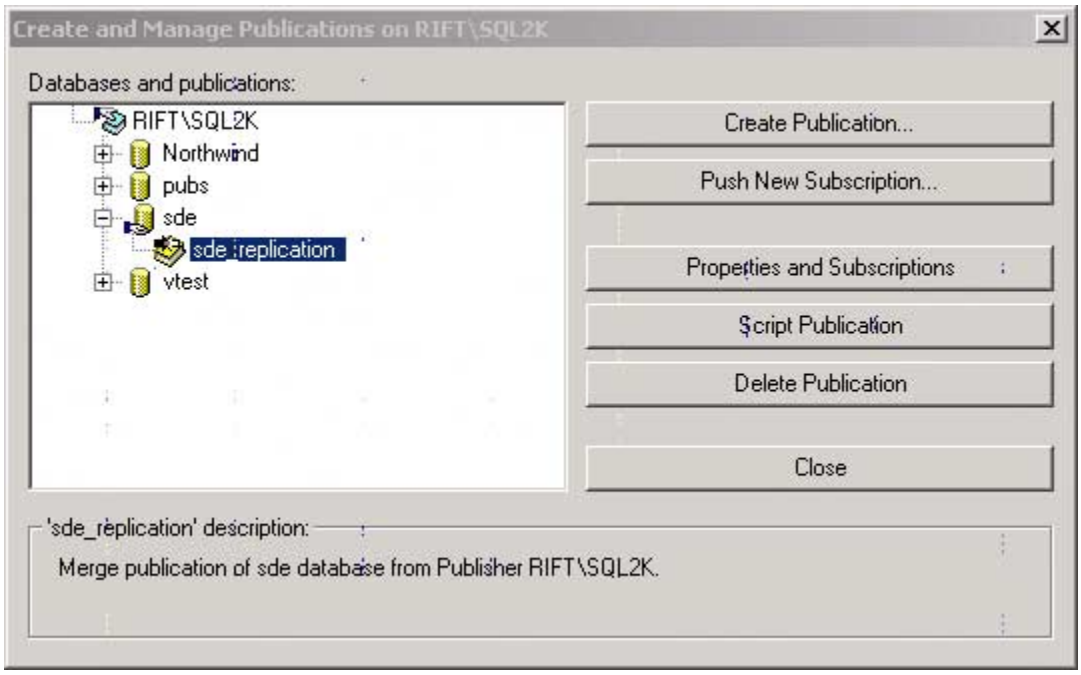


Set the parameters to make the snapshot. Click OK. You will return to the previous screen.


Create Publication Wizard
Set Snapshot Agent Schedule
Set the starting schedule for the Snapshot Agent.
The Snapshot Agent creates files containing the publication schema and data. These files are used to initialize new subscriptions to this publication.
Snapshot Agent schedule:
Occurs every 1 day(s), at 9:30:00 AM.
Change...
Create the first snapshot immediately
< Back
Next >
Cancel
Help

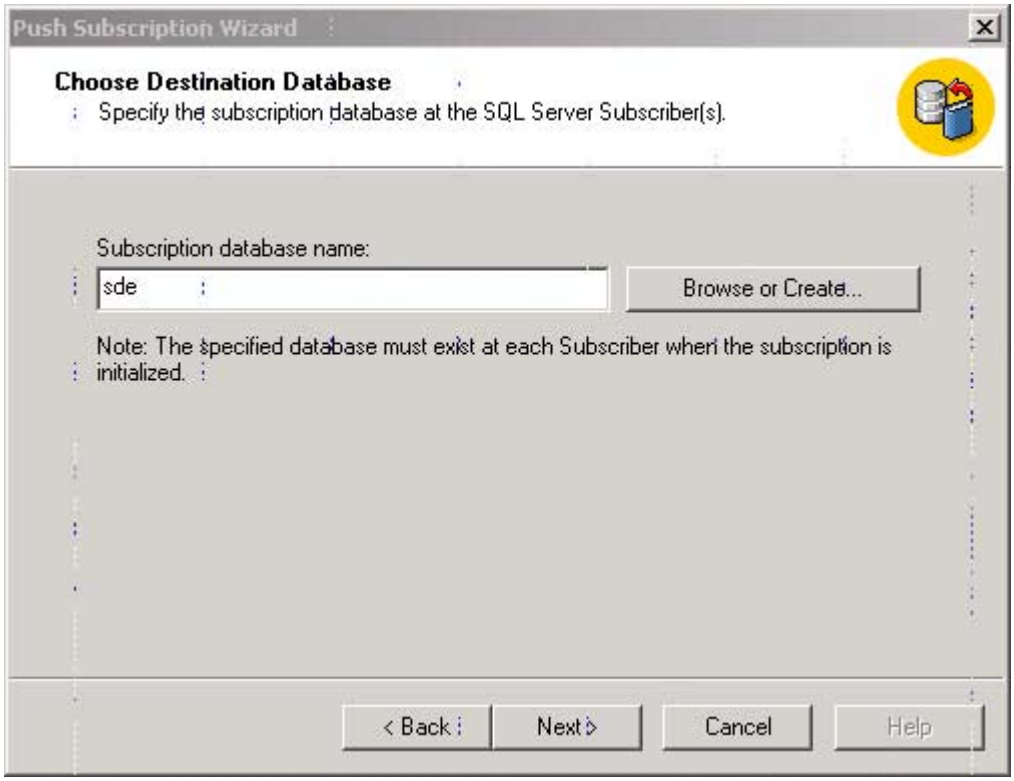
Click Next>.

Click Finish to complete the creation of the publication. In this example we will create a push subscription of the publication.

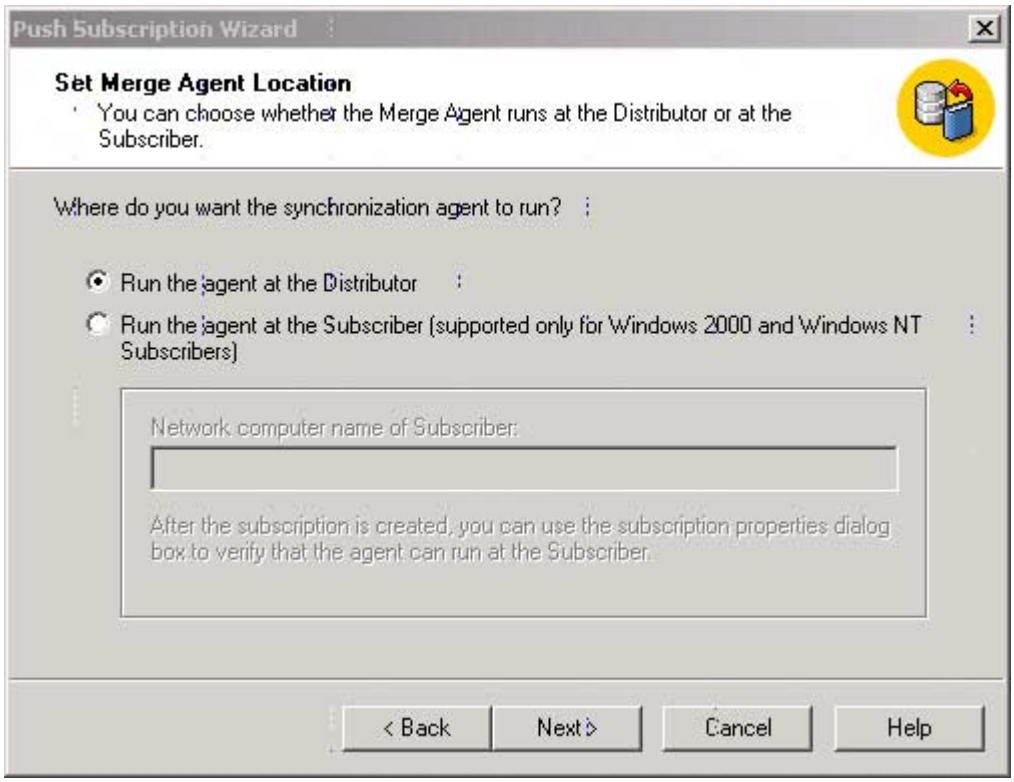


Highlight the publication and select Push New Subscription.

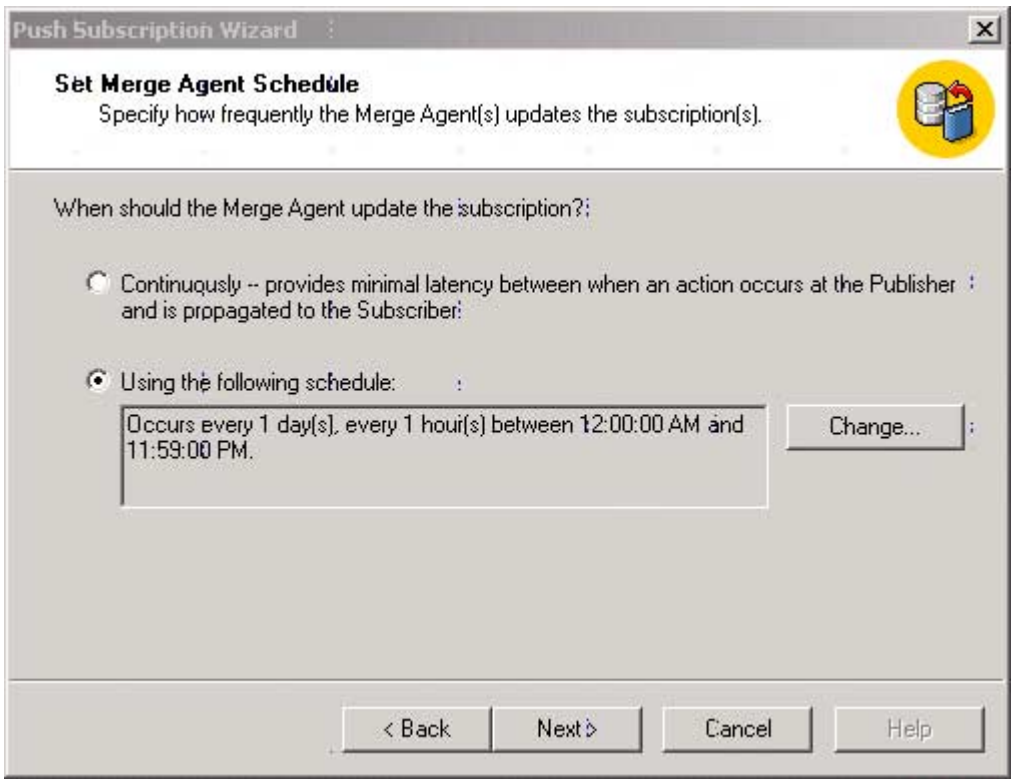
The Welcome to the Push Subscription Wizard will appear. This example will select advanced options. Click Next>.



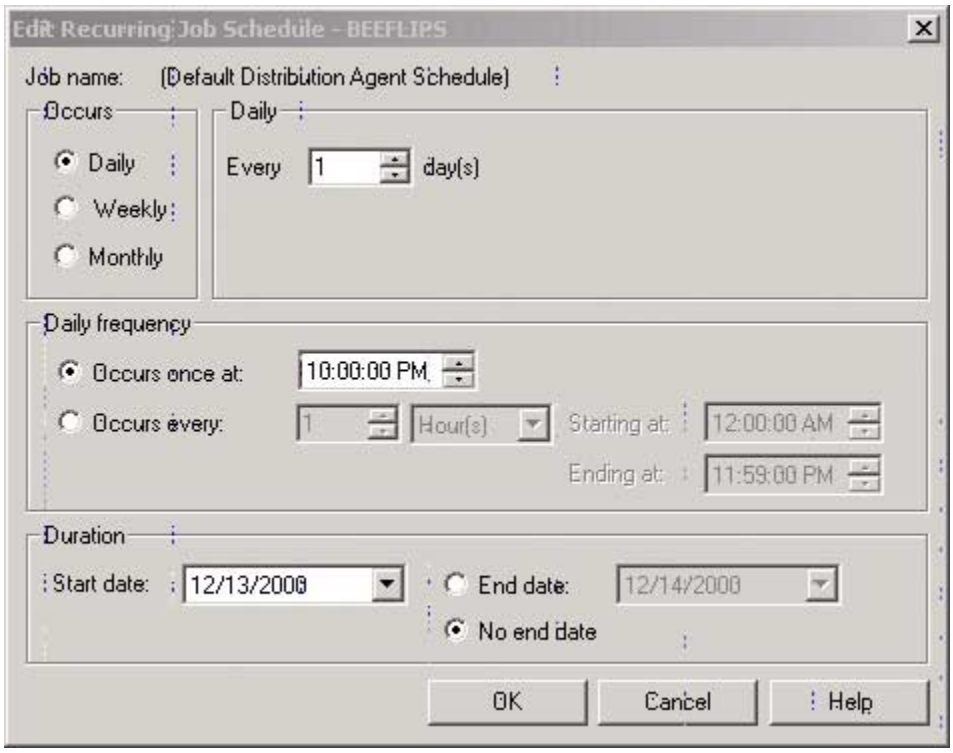
Browse to or create the database on the subscribing database that will receive the publication. Click Next>.



For this example, the agent will be run at the Distributor. Click Next>.

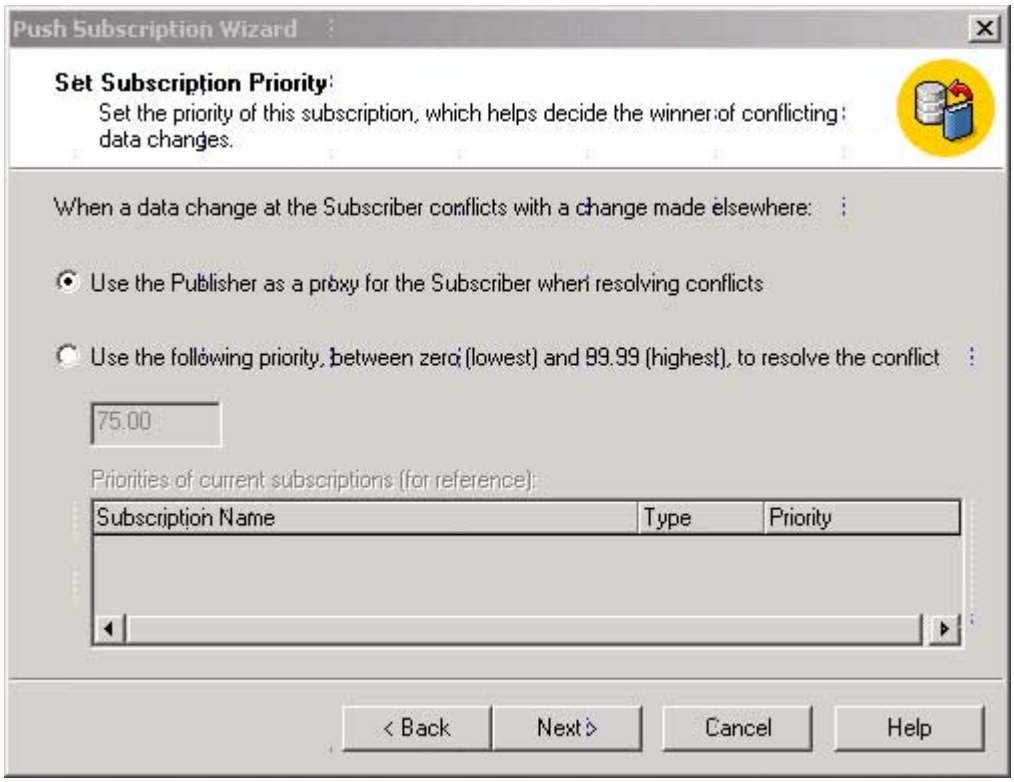


This example will change the Merge Agent schedule rather than run continuously. Click Change.

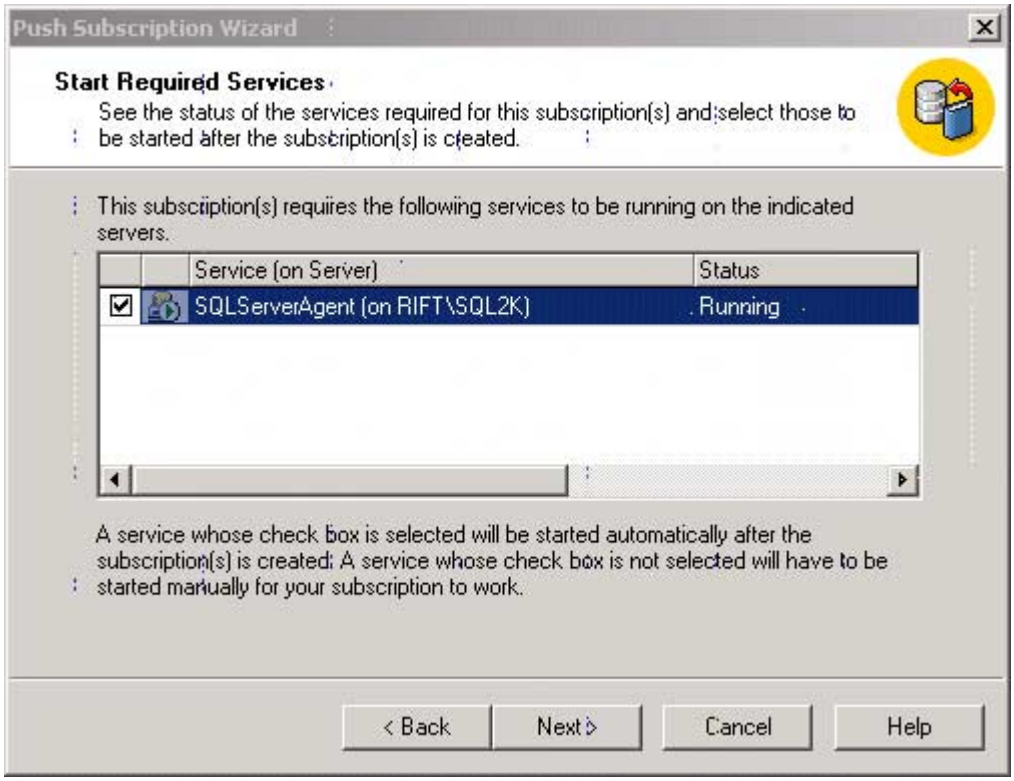


Select the parameters to run the Merge Agent schedule. Click OK.

For this example, choose Yes, initialize the schema and data. Click Start the Snapshot Agent to begin the initialization process immediately, based on the parameters to start earlier. Click Next>.



Select the subscription priority to resolve conflicts, then click Next>.



Click Next> to start the required services.

Click Finish to complete the replication procedure.

CHAPTER 10

Configuring transactional replication

ArcSDE 8.1 for MS SQL Server supports transactional and snapshot replication. The transactional model allows a replication distributor/publisher to update its data and then propagate those edits downstream to its subscribers. The snapshot model creates entire snapshots of data and delivers the entire snapshot to its subscribers. Snapshot replication works well with smaller, read-only data. Transactional replication proves better for frequent updates to dynamic data.

Before implementing transactional or snapshot replication, you should have a well-developed understanding of the SQL Server replication architecture. Make sure to read the section "Replication" in the SQL Server Books Online. This section demonstrates how to configure transactional and snapshot replication from a SQL Server 2000 publisher/distributor to a SQL Server 2000 replication subscriber.

Implementing transactional replication

Basic components of transactional replication with ArcSDE data

1. Create primary keys on all ArcSDE business tables. We recommend that you use a column that is an sde-maintained rowid. Use either the ObjectID field of data loaded with ArcCatalog or ArcToolbox or add a field with sdetable -o alter_reg and use the -C sde switch.
2. Create primary keys on five Geodatabase metadata tables (see the table below).
3. Define your publication: replicate all ArcSDE tables and stored procedures **except** SDE_process_information, SDE_state_locks, and SDE_table_locks. Also, it is not necessary to replicate the logfiles, SDE_logfiles and SDE_logfile_data. **Do not** replicate the execution of the SDE stored procedures. You must manually remap ownership of replication articles; SQL Server, by default, remaps all article ownership to 'sa.'
4. Create Subscriptions to each replication publication.

5. Deliver your snapshot to your replication subscribers. You can do this by using the SQL Server Distribution Agent, backup and restore, or sp_detach_db and sp_attach_db. Using the Distribution Agent is easiest but not efficient for moving huge quantities of data across your network. Both backup and restore and sp_detach_db and sp_attach_db are also easy but will require you to run sp_change_users_login for each data owner and possibly regrant table permissions to all nonowners. Don't forget to copy the SDE_process_information, SDE_state_locks, and SDE_table_locks tables to the replication subscriber if you have used the Distribution Agent to move your data to the subscriber machines.
6. Connect to the Subscribers.

Step 1: Create primary keys

Transactional replication requires a primary key on each replication article. All ArcSDE tables except business tables will have a primary key already created. Some Geodatabase metadata tables do not have the primary key implemented; these will have to be created manually.

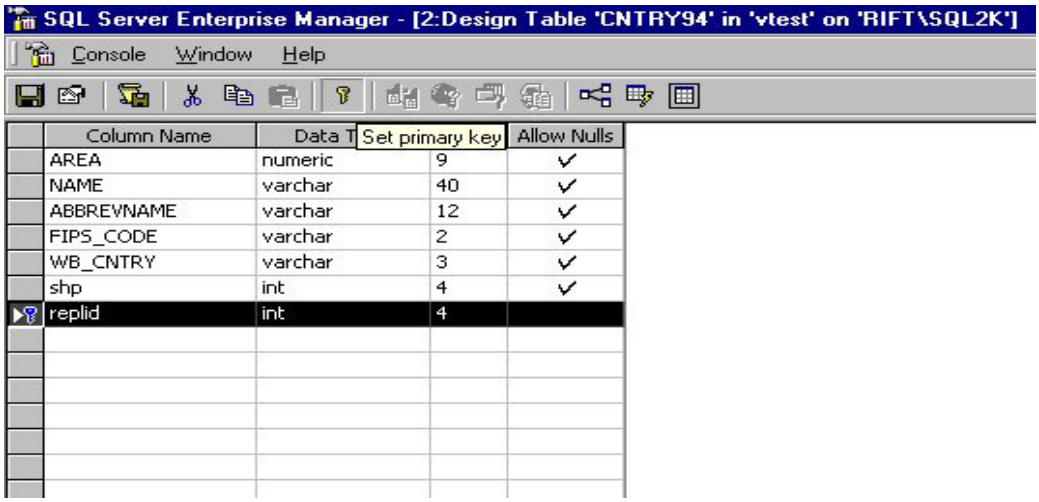
Create primary keys on your business tables

We recommend that you add a primary key to a column that has an sde-maintained row-id. Examples of such columns are OBJECTID columns created by loading data with ArcGIS tools ArcCatalog or ArcToolbox and columns set or created with sdetable -o alter_reg. Do not run sdetable -o alter_reg on data loaded with ArcCatalog/ArcToolbox for the purpose of creating a primary key. Use the existing objectid field instead. If you did not load your database with these tools but used shp2sde, sdeimport, etc., use the alter_reg command to create your field with the sde-maintained row-id column.

```
F:\world_sdex>sdetable -o alter_reg -i sql82k -t cntry94 -C sde -c replid
-u vtest -p go -D vtest

ArcSDE 8.1 Build 633 Tue Nov 21 22:30:10 PST 2000
Attribute Administration Utility
--------------------------------------------------
Alter registration for table cntry94. Are you sure? (Y/N): y
Table cntry94's registration successfully altered.
```

To add the primary key constraint to this table, use the design table utility within the SQL Server Enterprise Manager. Find your business table, right-click it, and select "design table." In the design table form, select the column on which you'll create the primary key. Now click the "Key" button.

Figure 10.1: In the Design Table dialog, select your primary key field and click the Key button.



Create primary keys on five Geodatabase metadata tables

This only applies to Geodatabase-dependent clients. If you are not using ArcGIS clients or did not load your data with ArcCatalog or ArcToolbox, skip this step. Otherwise, create a primary key on the following tables on the listed column(s).

Table	Column
GDB_FEATURECLASSES	ObjectClassID
GDB_NETWEIGHTASOCS	NetworkID
GDB_RELEASE	Any column
GDB_DEFAULTVALUES	ClassID,FieldName,Subtype
GDB_FIELDINFO	ClassID,FieldName

Step 2: Configure the Replication Publisher and Distributor

This step demonstrates using the SQL Server 2000 replication wizards. The process is similar in SQL Server 7. You can also use the replication stored procedures to set everything up. We recommend you use the wizards first to get familiar with the process, then either write out the stored procedures yourself or let the database script them for you.

Configure publishing and distribution

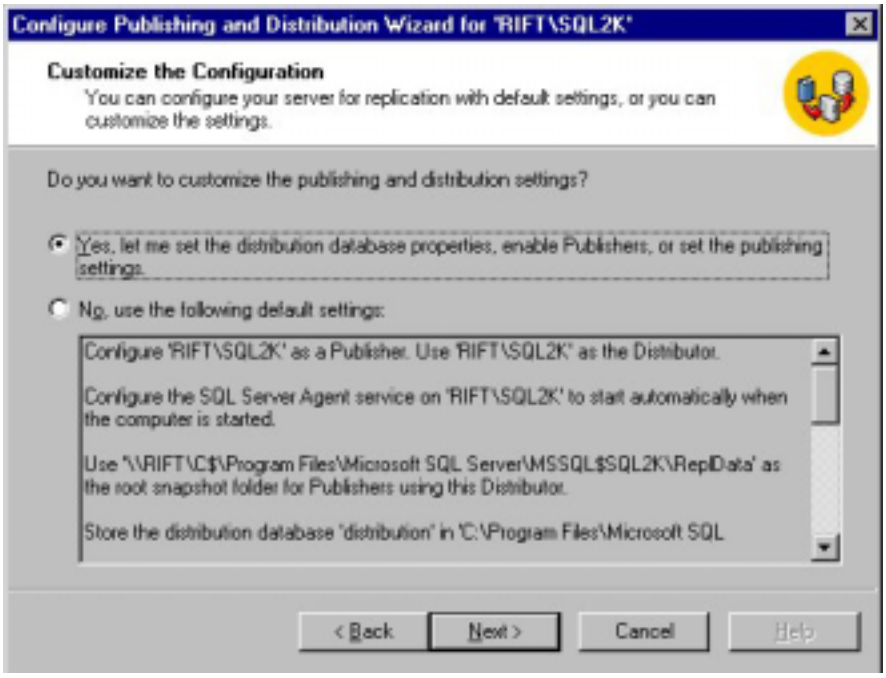
Start the **Configure Publishing and Distribution** Wizard by selecting and then right-clicking your server in the SQL Server Enterprise Manager. Choose View taskpad. This will repaint your details window with the SQL Server 2000 taskpad. Now click the Wizards tab. Under Setup Replication click Configure Publishing and Distribution

Wizard. You can click **Next** throughout this form as long as your server is listed in the entries for:

```
Specify '<server name>' or another server as a Distributor.
Configure the properties of '<server name>' as a Distributor.
Configure the properties of '<server name>' as a Publisher.
```

1. In the next form, **Select Distributor**, specify a distribution server if you don't want to take the default.
2. Click **Next** through the following form **Configure SQL Server Agent**. (NOTE: This agent may have to be owned by a domain account.)
3. Click **Next** through the following form **Specify Snapshot Folder**. (NOTE: You may get a warning message regarding sharing this folder.)
4. In the next form, **Customize the Configuration**, select "Yes, let me set the distribution database properties..." unless you are absolutely certain the defaults will work for you. In this case, we clicked Yes to customize settings regarding publication databases and enabling subscribers. Click **Next.**

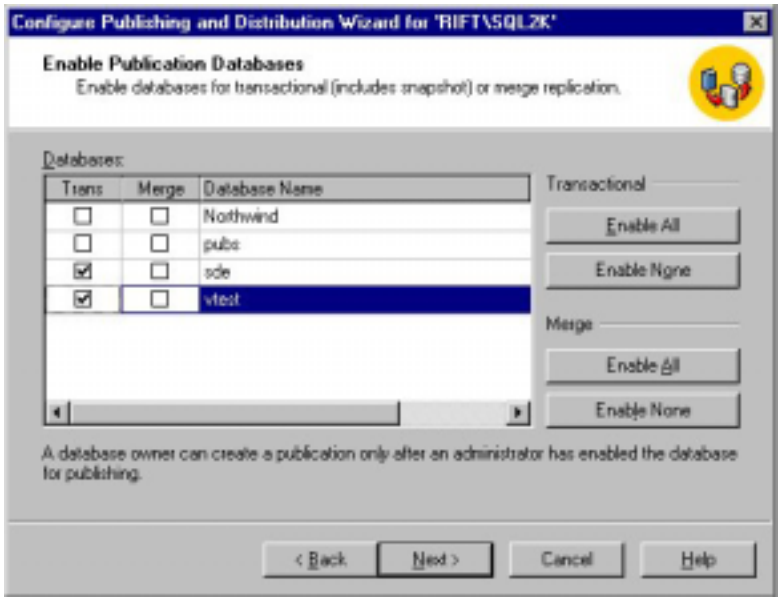
Figure 10.2: Click Yes to customize the settings.



5. Click Next through the following form: Provide Distribution Database Information.
6. In the ensuing form, Enable Publishers, confirm that you have the correct publisher selected. In most cases, the default will be adequate. Click Next.
7. The next form, Enable Publication Databases, requires you to select which databases you want to enable for publication. In this case, we will enable two, the "sde" and "vtest" databases. You do this by selecting a type of replication per database. Since ArcSDE 8.1 does not support merge replication, you must only choose the "trans" option. Click Next.

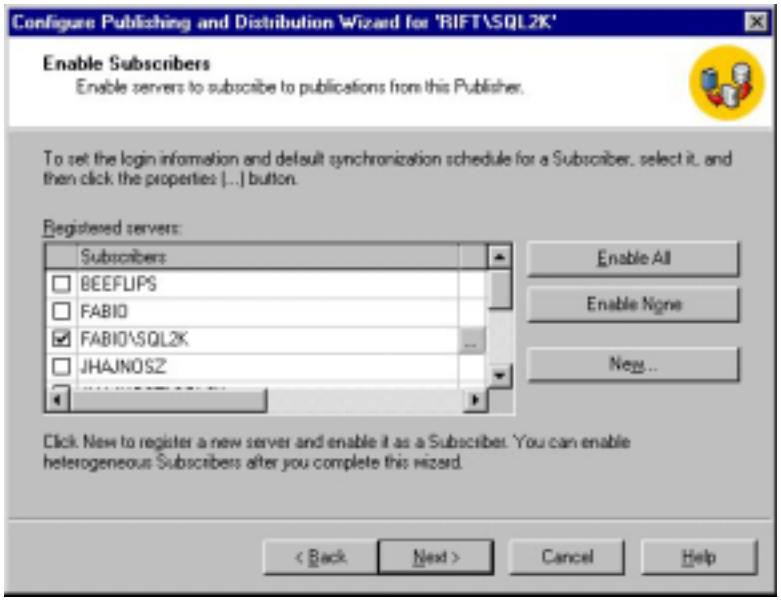
.

Figure 10.3: Check the boxes in the trans column for each database you wish to enable for replication.



8. Now you'll see the **Enable Subscribers** form. All the servers you have registered in your Enterprise Manager will be listed here. You may not want to enable them all for replication. In this case, we deselect all but one SQL Server 2000 instance (FABIO\sql2k) on a machine called "Fabio." Click **Next**.

*Figure10.4: Deselect all servers you **do not want** to enable for replication subscription.*



You've arrived at the end of the Configure Publishing and Distribution Wizard. Congratulations! Review the values listed to make sure they are correct. Now click **Finish**. The configuration process will start, step through each point, and finish with this dialog if all went well.

Figure10.5: Successful enabling of the distributor and publisher

Step 3: Create a publication

The MS SQL Server 2000 Books Online define a publication as:

A collection of one or more articles from one database. This grouping of multiple articles makes it easier to specify a logically related set of data and database objects that you want to replicate together.

Return to the taskpad view of your server and click the Wizards tab. Select the Create Publication topic under Setup Replication. In the ensuing form, Create and Manage Publications on <YourServerName>, you should see your databases listed in the tree view with the "hand" icon enabled to their left. Select the sde database and click the Create Publication button. The **Create Publication** Wizard will start. Click **Next**.

1. The **Choose Publication Database** wizard appears. Your should see your sde database highlighted in this box. Click **Next**.

2. The **Select Publication Type** dialog is next. Click the Transactional Replication radio button. Click **Next**.

3. Now the **Specify Subscriber Types** form displays. In this case, you have enabled one other server, a SQL Server 2000 machine, for transactional replication. Select all your platforms that apply here. You can replicate to SQL Server 2000 and SQL Server 7.0 or heterogeneous data sources such as DB2 or MS Access. Click **Next**.

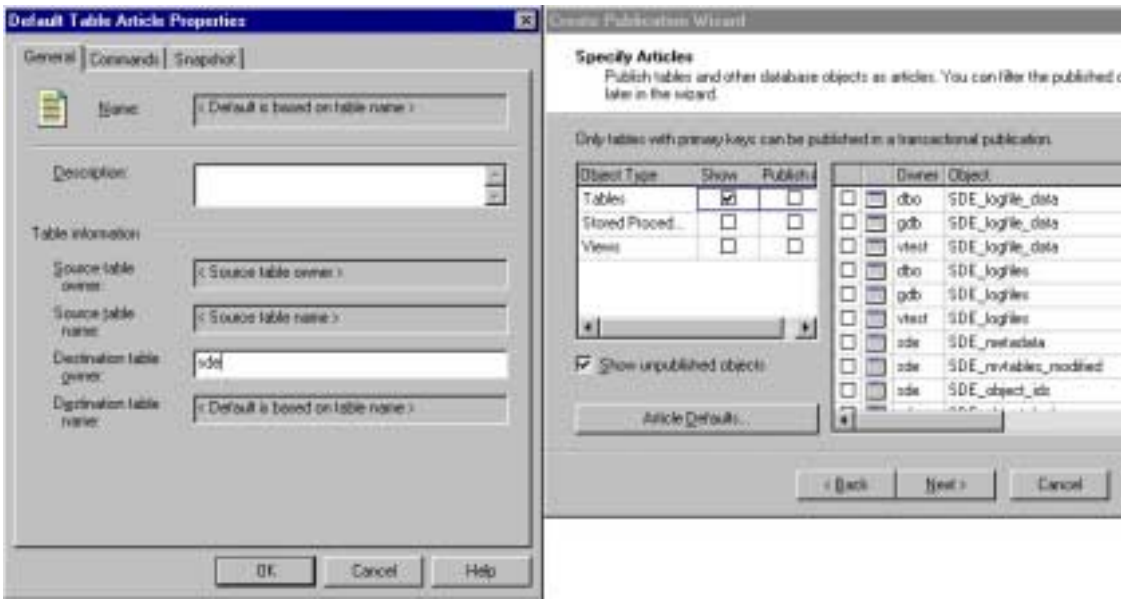
4. You should now see the **Specify Articles** form. A replication "article" is any table, stored procedure, or view you wish to replicate. In this form, you'll have to specify which articles you want to replicate and who should own those articles on the subscriber servers.

NOTE: You must ensure that your articles are correctly mapped to their proper owners. SQL Server, by default, will change all table, view, and stored procedure ownership to 'sa.' This will prevent the sde service from starting on the subscriber machine or client direct connections from connecting to the data store.

There are several ways to remap the article permissions. This example describes one method but assumes this condition: all replicated tables, stored procedures, and views in the 'sde' database are owned by the sde user, and everything in the 'vtest' database is owned by user 'vtest.' This condition allows the use of the **Article Defaults** dialog in the SQL Server 2000 **Specify Articles** form.

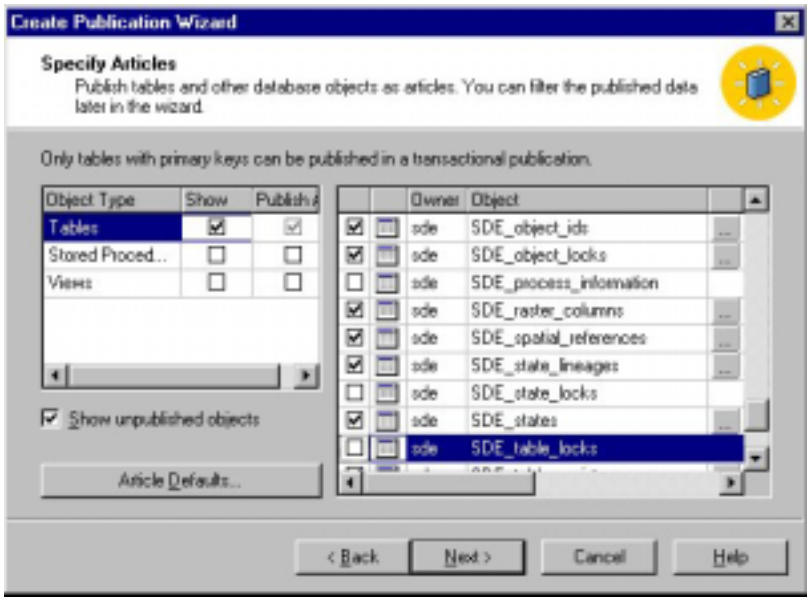
With focus on the Tables row of the Object Type column in the grid control on the left of this form, click the **Article Defaults** button. Select Table Defaults, OK. The **Default Table Article Properties** form shows. For Destination table owner input 'sde.' Click **OK** to return to the **Specify Articles** form.

Figure 10.6: Click the Article Defaults button to open the Default Table Article Properties Form. Input 'sde' into the Destination table owner field.



Back in the **Specify Articles** form, click the Publish All check box in the Tables Object Type record in the grid control on the left. In the right-hand grid, deselect all the logfile tables, the SDE_process_information, SDE_table_locks, and SDE_object_locks tables.

Figure 10.7: Deselect all logfile tables, the SDE_process_information, SDE_table_locks, and SDE_object_locks tables.



Now shift focus to the Stored Procedures record in the **Specify Articles** form by clicking it. Click the **Article Defaults** button to show the **Default Stored Procedure Article Default Properties** form and input 'sde' into the owner text field. Click the "Other" tab. Ensure that this screen has the "Never; replicate the schema only" radio button checked (see Figure 10.8).

Figure 10.8: Ensure that the "Never; replicate the schema only" radio button is checked.

Now perform step B for the stored procedures, only do not deselect anything.

Redo steps A and B for views, only do not deselect anything.

Check a few of the stored procedures and tables to ensure they have the proper ownership by clicking the small button on the right

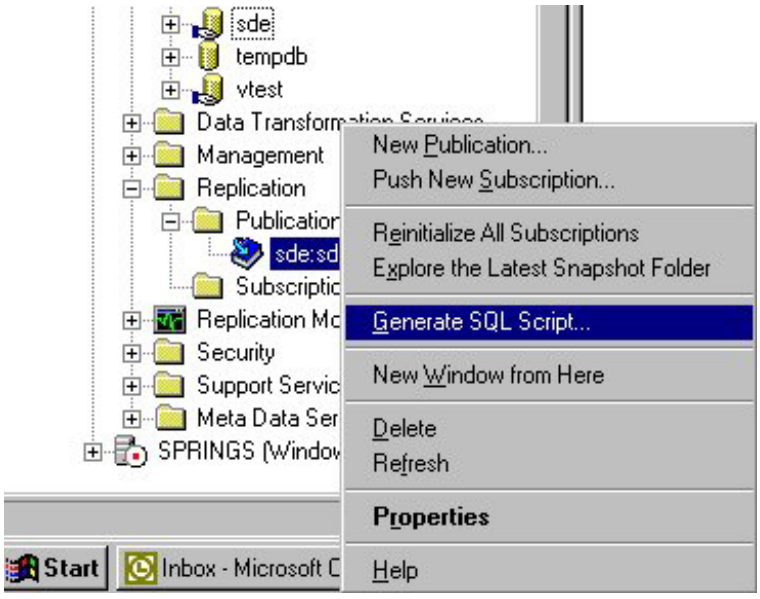
Click **Next**.

5. Click **Next** all the way to the end of the wizard.

Shortcut for point 4:

The steps in point 4 may seem a bit cumbersome. If you feel comfortable with publications in general, take all the defaults. This will remap all of your table, stored procedure, and view ownership to sa. When you've created the publication, script it to SQL by right-clicking the publication in the Enterprise Manager and select Generate SQL Scripts (see Figure 10.9).

Figure 10.9: Generate SQL scripts for your publication.



Save the SQL Script and delete your publication. Open the script in the SQL Server Query Analyzer and replace all instances of the string @destination_owner = N'<**some owner Name will be here**>' with the proper owner. For example:

```
...@destination_owner = N'terra\slim'
...@destination_owner = N'sde'
```

Now run the entire script to re-create the publication with the proper ownership of your replicated articles.

Create a publication for each database to be replicated. This means you'll have to rerun steps 1–5 for each database you want to replicate.

Step 4: Create subscriptions

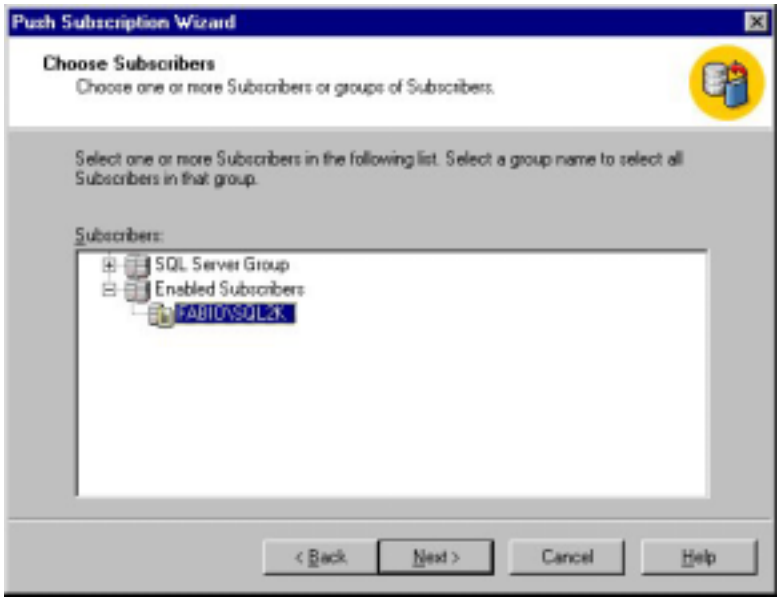
The SQL Server Books Online lists a Subscriber as:

Servers that receive replicated data. Subscribers subscribe to publications, not to individual articles within a publication.

In this step, we will create a new push subscription on a SQL Server 2000 instance named "Fabio\sql2k." Start the **New Subscription** Wizard by right-clicking the publication in the Enterprise Manager's Replication branch publication node. Select "Push New Subscription." Click **Next** in the Welcome screen.

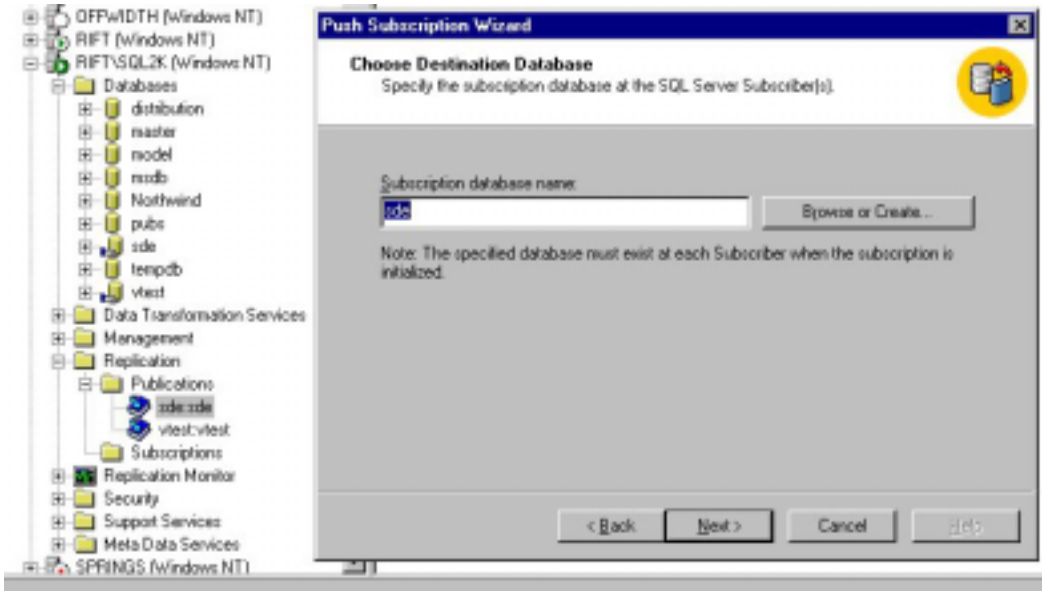
The Choose Subscribers form appears. You should see at least one enabled subscriber listed here. If you don't, you must go back and enable a subscriber for replication. Select one or more of the listed subscribers. See Figure 10.10.

Figure 10.10: You should see a list of enabled subscribers. Select one or more.



Click **Next** to proceed to the **Choose Destination Database** form. You'll see a text box with a selected database name here. If you right-clicked the 'sde:sde' publication, then you should see the word 'sde' written here. See Figure 10.11.

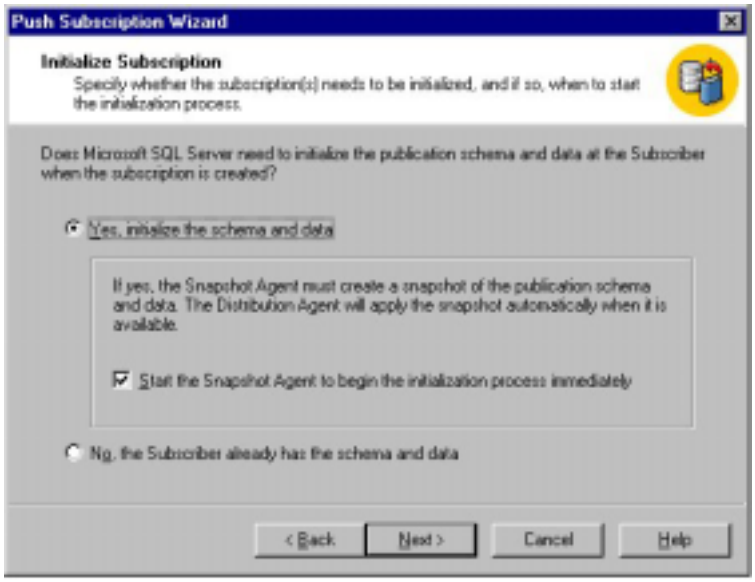
*Figure 10.11: Detail of **Choose Destination Database** form. Note the selected publication in the Enterprise Manager 'sde:sde.'*



Click **Next>**. Now the **Set Distribution Agent Schedule** appears. This form controls latency between the publisher and the subscribers. If you must maintain close synchronization between the publisher and subscribers, then take the default, continuous updates. You can set the schedule to update as you please. In this case, we take the defaults. Click **Next**>.

The **Initialize Subscription** form opens. This form controls the subscription initialization on the subscriber. If you want the Snapshot and Distribution Agents to initialize your schema and deliver your data to the subscriber machines, then take the defaults. If you've moved or will move your data to the subscriber yourself, click the radio button labeled "No, the Subscriber already has the schema and data." In this example, take the default and check the "Start the Snapshot Agent to begin the initialization process immediately" check box. This means that the Snapshot and Distribution Agents will initialize the schema and move all the articles to the subscriber.

Figure 10.12: Check the "Start the Snapshot Agent." check box.



Click **Next**>.

The **Start Required Services** Form displays. Make sure the status of your SQL Server Agent is "Running." Click **Next**>.

You've reached the last screen in the **Push Subscription** Wizard. Click **Finish** to start the agents.

Do points 1–7 for each publication you have. In this case, we will have to redo 1 to 7 for the 'vtest' publication.

Step 5: Deliver your snapshot to your replication subscribers

If you have followed these steps exactly, then most of your data has been moved to the subscribers. However, you may find that the Agents are too slow in initializing your schema and moving your data. You can do this manually by backing up your data to be replicated and restoring it to the subscriber machine. See point 4 of Step 4 for more information on what to do in the **Create Publication** Wizard. The next section explains how to manually distribute your replication snapshot.

Manually initializing your replication snapshot

Again, if you have followed these instructions verbatim, you don't have to do this. This only applies to scenarios that want to distribute data manually, outside the scope of the distribution and snapshot agents.

Move your data to the subscriber machine. You can do this with backup and restore, sp_detach_db and sp_attach_db, or Data Transformation Services (DTS).

If you've ran backup and restore or sp_detach_db and sp_attach_db, you must run sp_change_users_login to synchronize your SQL Server logins and users.

Moving the SDE_process_information, SDE_state_locks, and SDE_table_locks tables to the subscribers

You must move three ArcSDE tables to the replication subscriber, the SDE_process_information, SDE_state_locks, and SDE_table_locks tables. The easiest way to do this is to use the 'DTS.' Make sure that these tables, when moved to the subscribers, are still owned by the 'sde' user.

Special considerations for MS SQL Server 7.0

SQL Server 7.0 and 2000 have some differences with regard to replication. The most important one has to do with replicating stored procedures. With SQL Server 2000, you can replicate the schema of the stored procedures but not their execution. This option is not available with SQL Server 7.0. SQL Server 7.0 will replicate the stored procedures to the subscribers only if you allow their execution. This option is not supported in ArcSDE 8.1. **For SQL Server 7.0 replication of ArcSDE 8.1 data, you'll have to move all the stored procedures to each subscriber manually.**

Step 6: Connect to the subscribers

Now you should be able to make connections to the subscriber machines serving ArcSDE data. You can either start an SDE service on each subscriber or make direct connections from ArcSDE clients such as ArcGIS or both. If you want to start an ArcSDE service on each subscriber, then you must have the ArcSDE application installed on those servers. However, you don't have to install the ArcSDE application on a server setup for Direct Connections. See Chapter 5, 'Connecting to SQL Server', in this manual for more information on this.

APPENDIX A

ArcSDE compressed binary

This chapter describes the state of ArcSDE with no configuration changes made to the server or any spatial databases. This chapter also details how the ArcSDE application manages your data with regard to indexes, triggers, and specific SQL Server configuration settings. This chapter will help you understand how to manage spatial data.

Compressed binary

After the client verifies the geometry, it compresses it and sends it to the server, where it is stored in compressed binary format in a feature table. Compressing the geometry on the client offloads the task from the ArcSDE server and reduces the transmission time to send the geometry to the ArcSDE server. Storing compressed geometry data reduces the space required to store data by as much as 40 percent.

Compressed binary metadata tables

A compressed binary feature class comprises three tables: the business table, the feature table, and the spatial index table.

The business table contains attributes and a spatial column. The spatial column is a key to the feature and spatial index tables.

The relationship between the business table and the feature table is managed through the spatial column and the FID column. This key, which is maintained by ArcSDE, is unique.

NAME	DATA TYPE	Allow Nulls
fid	INTEGER(4)	NOT NULL
numofpts	INTEGER(4)	NOT NULL
entity	SMALLINT(2)	NOT NULL
eminx	FLOAT(8)	NOT NULL
eminy	FLOAT(8)	NOT NULL
emaxx	FLOAT(8)	NOT NULL
emaxy	FLOAT(8)	NOT NULL
eminz	FLOAT(8)	NULL
emaxz	FLOAT(8)	NULL
min_measure	FLOAT(8)	NULL
max_measure	FLOAT(8)	NULL
area	FLOAT(8)	NOT NULL
len	FLOAT(8)	NOT NULL
points	IMAGE	NULL
anno_text	VARCHAR(255)	NULL

Feature table schema

The feature table stores the geometry, annotation, and CAD in the image type POINTS column. The internal SQL Server datatype is listed in the table above. The ArcSDE datatype for each column is defined below.

- fid (SE_INTEGER_TYPE)—contains the unique ID that joins the feature table to the business table
- entity (SE_INTEGER_TYPE)—the type of geometric feature stored in the shape column (e.g., point, linestring)
- numofpts (SE_INTEGER_TYPE)—the number of points defining the shape
- eminx, eminy, emaxx, emaxy (SE_FLOAT_TYPE)—the envelope of the shape
- eminz (SE_FLOAT_TYPE)—the minimum Z value in the shape
- emaxz (SE_FLOAT_TYPE)—the maximum Z value in the shape
- min_measure (SE_FLOAT_TYPE)—the minimum measure value in the shape
- max_measure (SE_FLOAT_TYPE)—the maximum measure value in the shape
- area (SE_FLOAT_TYPE)—the area of the shape
- len (SE_FLOAT_TYPE)—the length or perimeter of the shape
- points (SE_SHAPE_TYPE)—contains the byte stream of point coordinates that define the shape's geometry
- anno_text (SE_STRING_TYPE)—contains the feature annotation string.

NAME	DATA TYPE	Allow Null?
sp_fid	INTEGER(4)	NOT NULL
Gx	INTEGER(4)	NOT NULL
Gy	INTEGER(4)	NOT NULL
Eminx	INTEGER(4)	NOT NULL
Eminy	INTEGER(4)	NOT NULL
Emaxx	INTEGER(4)	NOT NULL
Emaxy	INTEGER(4)	NOT NULL

Spatial index table schema

The spatial index table defines the grid range and extent of all geometry in an ArcSDE feature class.

- sp_fid (SE_INTEGER_TYPE)—contains the unique ID that joins the feature table to the business table
- gx/gy (SE_INTEGER_TYPE)—defines the feature's extent in grid cells
- eminx/eminy/emaxx/emaxy (SE_INTEGER_TYPE)—defines the extent of the feature in system units

In this example the FEATURE-ID column from the WELLS business table references features from the feature and spatial index tables:

WELL_ID	DEPTH	ACTIVE	FEATURE-ID
1	30029	Yes	101
2	13939	No	102
3	92891	No	103
…	…		…

FID	AREA	LEN	EMINX,EMINY,…	POINTS
101				<compressed feature>
102				<compressed feature>
103				<compressed feature>
…				**…**

SP_FID	GX	GY	{EMINX,EMINY,EMAXX,EMAXY}
101	70	100	
102	70	100	
103	71	100	
…			

A business/feature/spatial index key reference

The spatial grid index

The spatial grid index is a two-dimensional index that spans a feature classsuch as the reference grid you might find on a common road map. You may assign the spatial grid index one, two, or three grid levels, each with its own distinct cell size. The mandatory first grid level has the smallest cell size. The optional second and third grid cell levels are disabled by setting them to 0. If enabled, the second grid cell size must be at three times larger than the first grid cell size, and the third grid cell size must be three times larger than the second grid cell size.

The spatial index table (S<feature class_id>) has seven integer columns that store the grid cell values, feature envelopes, and corresponding feature IDs. Adding a feature to a feature class adds one or more grid cells to the spatial index table. The number of records added to the spatial index table depends on the number of grid cells the feature spans.

The spatial index table contains two indexes. One index is on the SP_FID column, which contains the feature ID. The other is a clustered composite index that includes all of the columns of the spatial index table. Since all of the columns of the spatial index table are indexed, the values of the table are read from the leaf blocks of the index and not the table data blocks. The result is less I/O and better performance. In addition, the spatial index table is not accessed whenever the feature class is queried. Therefore, when considering how to position the tables and indexes to reduce disk I/O contention, you should be concerned about the positioning of the indexes of the spatial index table but not the table itself.

Building the spatial index

Every time a feature class is added to a business table, a persistent spatial index is built for it.

The ArcSDE server manages the spatial index throughout the life of the feature class. As features are inserted, updated, or deleted, the spatial index is automatically updated.

A load-only mode disables spatial index management until loading is completed. This boosts loading performance substantially and is imperative for bulk loading efforts (no queries are allowed in the load-only mode except native SQL-based queries).

Once loading has been completed, the spatial index is enabled by returning it to normal I/O mode. The conversion from normal I/O mode to load-only I/O mode reconstructs the spatial index.

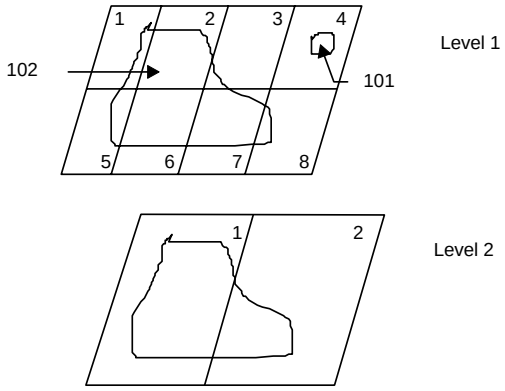
Inserting, updating, or deleting a feature updates the spatial index when the feature class is in normal I/O mode.

ArcSDE overlays the extent of each feature onto the lowest grid level to obtain the number of grid cells. If the feature exceeds four cells, ArcSDE promotes the feature to the next highest grid level, if you have defined one. ArcSDE will continue to promote the feature until it fits within four cells or less or until the highest defined grid level is reached. On the highest defined grid level, geometries can be indexed by more than four grid cells.

ArcSDE adds the feature's grid cells to the spatial index table with their corresponding shape ID and feature envelope. The grid level is encoded with each grid cell.

In the example below, the feature class has two grid levels. Area shape 101 is located in grid cell 4 on level 1. A record is added to the spatial index table because the feature resides within four grid cells (in this case it is one). Area feature 102's envelope is located in cells 1 through 8 on level 1. Because the feature's envelope resides in more than four grid cells, the feature is promoted to level 2, where its envelope fits within two grid cells. Feature 102 is indexed at level 2, and two records are added to the spatial index table.

Figure A.1: Shape 101 is indexed on grid level 1, while shape 102 is indexed on grid level 2 where it is in only two grid cells.



Spatial queries and the spatial index

Spatial queries, like finding all the lakes within a state boundary, use the spatial index. A spatial index is used unless the search order has been set to SE_ATTRIBUTE_FIRST in the SE_stream_set_spatial_constraints function. When the search order is set to SE_ATTRIBUTE_FIRST, ArcSDE ignores the spatial index, and the criteria of the attribute where clause determines which records of the feature class the query returns.

Whenever the spatial index is used, the ArcSDE service generally uses the following decision process when performing the query:

1. Define the envelope. The envelope could be defined directly by the application such as the extent defined by the ArcMap zoom in tool. Alternatively, the envelope may be defined as the envelope of another feature.
2. Join the spatial index table with the feature table and return all features whose grid cells intersect the envelope.
3. Join the feature table with the business table and apply the criteria of the attribute where clause to further refine features returned.

Grid cell size impacts the size of the spatial index table. Setting up the spatial index means balancing the cell sizes—smaller cell sizes mean more cells per shape, which requires more entries in the spatial index table.

Guidelines for tuning the spatial index

Because client applications and spatial data profiles vary from one system to another, no single solution fits all. Experienced users of ArcSDE often experiment with the spatial index, trying different cell sizes and different grid level configurations.

The sdelayer command has several operations that can help you optimize the spatial index by changing the grid cell sizes and adding new grid levels with the 'alter' operation. The 'stats' and 'si_stats' operations profile your spatial data and current spatial index.

The following guidelines can help improve the performance of spatial queries.

- Consider how many grid levels are needed and remember that the ArcSDE server scans the spatial index table once per grid level. Often a single grid level

is the best solution for a feature class, despite the notion of distributing geometries evenly across many grid levels to minimize the spatial index entries.

- Use one grid level for pure point type feature class and consider making the cell sizes large. Spatial queries generally process point geometries faster than other geometry types.
- Monitor the spatial index. Tuning a spatial index is difficult if the data changes frequently. Tuning depends on the structure of the spatial data. Periodically assess the spatial index as your spatial data changes.
- Base the spatial index on the application. Match the spatial index grid cell sizes to the extent of the application window. By doing so, the application is probably viewing exact entries in the spatial index table. This helps to size the spatial index table suitably and reduces the amount of processing because fewer candidate feature IDs must be evaluated against the feature table (see 'Spatial queries and the spatial index' above).
- For unknown or variable application windows, start by defining one grid level with a cell size three times the average feature extent size. Query the feature table to obtain the average feature size with the following SQL statement:

  ```
  select (avg(emaxx - eminx) + avg(emaxy - eminy)) / 2
  from f<N>;

  (where <N> is the layer number of the feature class)
  ```

  Such spatial index configuration minimizes the number of rows in a spatial index table while maintaining the proficiency of the index because the majority of features can be referenced by less than one or two grid cells.
- Which application loaded your data? The dataloaders used in ArcCatalog and ArcToolbox will not calculate an optimal spatial index. The ArcSDE command tool shp2sde will generally calculate a better grid size. An easy test is to load the same data with both toolsets (e.g., ArcCatalog and shp2sde) and compare calculated grid sizes and drawing times for each. Alternatively, load the data with ArcCatalog or ArcToolbox and run the query listed immediately above. Use sdelayer –o alter to change your grid size.
- Design the feature class around spatial data categories such as type, geometry size, and distribution. Sometimes a carefully designed feature class, using these categories, can substantially boost the performance of spatial queries.

Displaying spatial index statistics

The sdelayer command's spatial indexstatistics operation, 'si_stats', can help you determine optimum spatial index grid sizes. Optimum grid cell sizes depend on the spatial extent of all feature geometries, the variation in feature geometry spatial extent, and the types of searches to be performed on the map feature class. Below is a sample output generated by si_stats:

```
$ sdelayer -o si_stats -l victoria,parcels -u av -p mo -i sde81
ArcSDE   8.1          Wed Jan 17 22:43:09 PST 2000
Layer    Administration Utility
-----------------------------------------------------
Layer 1 Spatial Index Statistics:
Level 1,   Grid Size 200
|-----------------------------------------------------|
| Grid Records: 978341                                |
| Feature Records: 627392                             |
| Grids/Feature Ratio: 1.56                           |
| Avg. Features per Grid: 18.26                       |
| Max. Features per Grid: 166                         |
| % of Features Wholly Inside 1 Grid: 59.71           |
```

```
|-----------------------------------------------------|
|             Spatial Index Record Count By Group     |
| Grids:     <=4    >4  >10  >25  >50  >100  >250  >500 |
|---------- ------ --- ---- ---- ---- ----- ----- ----- |
| Shapes:   627392   0    0    0    0     0     0     0 |
| % Total:     100%  0%   0%   0%   0%    0%    0%    0%|
|-----------------------------------------------------|
Level 2,   Grid Size 1600 (Meters)
|-----------------------------------------------------|
| Grid Records: 70532                                 |
| Feature Records: 36434                              |
| Grids/Feature Ratio: 1.94                           |
| Avg. Features per Grid: 18.21                       |
| Max. Features per Grid: 82                          |
| % of Features Wholly Inside 1 Grid: 45.35           |
|-----------------------------------------------------|
|             Spatial Index Record Count By Group     |
| Grids:     <=4    >4  >10  >25  >50  >100  >250  >500 |
|---------- ------ --- ---- ---- ---- ----- ----- ----- |
| Shapes:   35682  752   87   17    3     0     0     0 |
| % Total:     97%   2%   0%   0%   0%    0%    0%    0%|
|-----------------------------------------------------|
```

As the output shows, for each defined spatial index level, the following values and statistics are printed:

- Grid level and cell size.
- Total spatial index records for the current grid level.
- Total geometries stored for the current grid level.
- Ratio of spatial index records per geometry.
- Geometry counts and percentages by group that indicate how geometries are grouped within the spatial index at this grid level. The column headings have the following meaning (where 'N' is the number of grid cells):

<=N Number of geometries and percentage of total geometries that fall within <= N grid cells
>N Number of geometries and percentage of total geometries that fall within > N grid cells

Notice that the '>' groupings include count values from the next group. For instance, the '>4' group count represents the number of geometries that require more than four grid records as well as more than 10, and so on.

- Average number of geometries per grid.
- Maximum number of geometries per grid. This is the maximum number of geometries indexed into a single grid.
- Percentage of geometries wholly inside one grid. This is the percentage of all geometries wholly contained by one grid record.

The output sample shows spatial index statistics for a map feature class that uses two grid levels: one that specifies a grid size of 200 meters, the other a grid size of 1,600 meters. When a geometry requires more than four spatial index records, it is automatically promoted to the next grid level, if one is defined. That is, in no case will a geometry generate more than four grid records if more than one grid level exists. If a higher grid level doesn't exist, then a geometry can have more than four grid records.

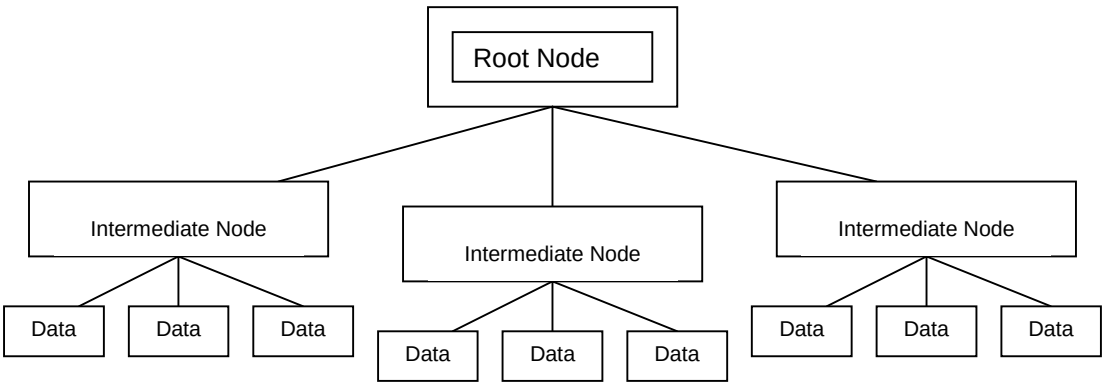
In the example above, 627,392 features are indexed through grid level 1. Because the system automatically promotes geometries that need more than four spatial index records to the next defined grid level, all 627,392 geometries for grid level 1 are indexed with four grid records or less. Grid level 2 is the last defined grid level, so geometries indexed at this level are allowed to be indexed with more than four grid records. At grid level 2,

there are a total of 36,434 geometries and 70,532 spatial index records; 35,682 geometries are indexed with four grid records or fewer, 752 geometries with more than four grid records, 87 geometries with more than 10, 17 geometries with more than 25, and three geometries with more than 50 grid records. Percentage values below each column show how the geometries are dispersed through the eight groups.

Indexes

Every data table in ArcSDE for MS SQL Server has a clustered index. The clustered key in an ArcSDE table is almost always the join key in a query like the business table's spatial column or the f table's fid column. The SQL Server query processor favors clustered indices because a clustered index orders the table data at its leaf level based on the index key. A normal index would have a pointer to the data page. Clustered indexes are b-tree structures containing pages with index rows holding a key value and a pointer to either a data page or a lower-level index page. All searches begin at the top level of the b-tree, the root node. This page is found in the sysindexes table in the root field. Within this page are pointers to intermediate pages, themselves holding ranges of values and pointers to other pages.

Figure A.2: B-tree diagram of a clustered index



Here's an example of the contents of the root node (shortened for brevity's sake). The Page Header displays 0:0 for both m_nextPage and m_prev_Page, indicating that it is a root node. Also, two data slots are displayed, each with a key value and a pointer to another page. The pages pointed to are intermediate pages, themselves containing more key values and pointers.

```
PAGE HEADER:
------------
Page @0x19A8A000
----------------
m_pageId = (1:4713)       m_headerVersion = 1       m_type = 2
m_typeFlagBits = 0x0      m_level = 0               m_flagBits = 0x0
m_objId = 1526296497      m_indexId = 1             m_prevPage = (0:0)
m_nextPage = (0:0)        pminlen = 11              m_slotCnt = 133
m_freeCnt = 5968          m_freeData = 1958         m_reservedCnt = 0
m_lsn = (69:1958:6)       m_xactReserved = 0        m_xdesId = (0:0)
m_ghostRecCnt = 0         m_tornBits = 0


DATA:
-----
```

```
Slot 0, Offset 0x60
-------------------
Record Type = INDEX_RECORD
Record Attributes =  NULL_BITMAP
19a8a060:  00000116  00126800  02000100      0000 .....h........

Slot 1, Offset 0x6e
-------------------
Record Type = INDEX_RECORD
Record Attributes =  NULL_BITMAP
19a8a06e:  00001816  00126a00  02000100      0000 .....j........
```

Since a cluster orders the storage of a table, you cannot specify different filegroups for the cluster and its table. For example, if you specify in the dbtune table that the feature table's clustered index should go on filegroup f_idx and the table should reside on f_storage, your table will be stored with the index on f_idx.

What will happen here?

DEFAULTS	F_INDEX_CLUSTER	1
DEFAULTS	F_INDEX_FID	WITH FILLFACTOR=75 ON F_IDX
DEFAULTS	F_STORAGE	ON F_STORAGE

Index maintenance

When you initially create a table with a clustered index and specify a fill factor for that index, each data page is filled to that fill percentage when you initially insert rows. Later, when you edit this table with insert, delete, and update statements, the amount of data contained in each page will tend to increase.

If a data page fills to 100 percent and that particular page incurs an update or an insert, the page splits approximately in half. Half of the page moves into a newly allocated extent not contiguous to its former location. This will fragment your tables and increase query time as the disk spindles must travel to more locations to fetch data.

You can use the fillfactor clause in the dbtune table to delay page splits by limiting the fill percentage of a data page when it is initially allocated. Any ensuing inserts or updates will add to the fill percentage of a data page. Monitor page splits with DBCC SHOWCONTIG, and when your tables become fragmented, rebuild your clustered indexes or run DBCC INDEXDEFRAG (SQL Server 2000 only) or DBCC DBREINDEX.

Rebuilding a clustered index will reorder its table and defragment it. Data pages are reset to their initial fillfactor setting. The simplest way to rebuild an index is to place an ArcSDE feature class into Load_only_io and then back to normal_io. However, this won't work on versioned data.

If your data is multiversioned, you don’t need to worry about monitoring the ArcSDE business table for extent fragmentation. Instead, your delta tables (a&d), feature tables, and spatial index will become fragmented. You can issue a DBCC DBREINDEX statement or invoke the DBCC INDEXDEFRAG command.

Here’s an example of DBCC SHOWCONTIG run against a feature class that has incurred several edits—in this case, many updates and deletes. The output shown below reflects only fragmentation against the spatial index.

```
dbcc showcontig('hydro.s5')

DBCC SHOWCONTIG scanning 's5' table...
Table: 's5' (1622296839); index ID: 1, database ID: 7
TABLE level scan performed.
- Pages Scanned................................: 57
- Extents Scanned..............................: 11
- Extent Switches..............................: 36
- Avg. Pages per Extent........................: 5.2
- Scan Density [Best Count:Actual Count].......: 21.62% [8:37]
- Logical Scan Fragmentation ..................: 33.33%
- Extent Scan Fragmentation ...................: 27.27%
- Avg. Bytes Free per Page.....................: 3780.6
- Avg. Page Density (full).....................: 53.29%
```

For an exact explanation of these numbers, see the SQL Server Books Online under DBCC SHOWCONTIG. In this case, 36 extent switches are far too many to cover 57 pages. The best count reflects an optimal amount of extent switches, which in this case would be 8. Both extent scan fragmentation and logical scan fragmentation should be as close to 0 as possible. These measurements reflect the ordering of pages and extents in the index allocation map. Running either of these commands on this table will defragment the table.

```
dbcc indexdefrag (sde,'vtest.s5',s5_pk)
dbcc dbreindex('hydro.s5')

DBCC SHOWCONTIG scanning 's5' table...
Table: 's5' (1622296839); index ID: 1, database ID: 7
TABLE level scan performed.
- Pages Scanned................................: 41
- Extents Scanned..............................: 6
- Extent Switches..............................: 5
- Avg. Pages per Extent........................: 6.8
- Scan Density [Best Count:Actual Count].......: 100.00% [6:6]
- Logical Scan Fragmentation ..................: 4.88%
- Extent Scan Fragmentation ...................: 33.33%
- Avg. Bytes Free per Page.....................: 2096.6
- Avg. Page Density (full).....................: 74.10%
```

DBCC DBREINDEX and DBCC INDEXDEFRAG differ in that INDEXDEFRAG can be run online. It will not block queries or updates. For relatively unfragmented tables, DBCC INDEXDEFRAG should run faster than DBREINDEX. For more severely fragmented data, use DBREINDEX.

You only need to worry about business table fragmentation after running a full compress, when the delta tables have been flushed into the business table. Therefore, you should rebuild your business tables' clustered indexes after running a full compress.

NETWORK PACKET SIZE (SDEPACKETSIZE)

ArcSDE for MS SQL Server stores geometry in an image data type column, points, of the feature table. Several Geodatabase network tables use image type columns. Microsoft recommends increasing the size of the **network packet size** setting when employing image data type columns.

Network Packet Size is the size of the tabular data scheme (TDS) packets used to communicate between applications and the relational database engine. The default packet size is 4 KB and is controlled by the network packet size configuration option.

ArcSDE, by default, sets this to 8192k, double its default setting of 4096k. You can also make this setting global to your MS SQL Server by using the sp_configure statement's "network packet size" setting.

```
sp_configure 'show advanced options',1
```

```
reconfigure with override

go

sp_configure 'network packet size',8192

reconfigure with override

go
```

You can control the network packet size setting with the environment variable SDEPACKETSIZE, although we recommend you stay with the default setting.

Text in row (SQL Server 2000 Only)

SQL Server 2000 allows up to 7 KB of image, text, or ntext data to be stored directly in row on a data page. SQL Server 7 stored these data types in separate image pages, located with a 16-byte pointer in the table's data page. Text in row at SQL Server 2000 permits you to place image type data directly in the data page, and anything exceeding the setting is placed in a traditional image page. All ArcSDE geometry is stored in an image type column in the feature table. ArcSDE Raster layers and Geodatabase network tables use image data types.

This setting is enabled on tables using the command sp_tableoption. ArcSDE 8.1 does this automatically for you through the dbtune table. The SDE_dbtune table sets a default of 256 bytes for many settings.

Keyword	Parameter	Explanation
DEFAULTS	B_TEXT_IN_ROW	Business tables with BLOB data type
DEFAULTS	F_TEXT_IN_ROW	Feature tables points column
NETWORK_DEFAULTS	B_TEXT_IN_ROW	Network business table with BLOB type
NETWORK_DEFAULTS	F_TEXT_IN_ROW	Feature table for generated junctions class
NETWORK_DEFAULTS::NETWORK	B_TEXT_IN_ROW	N_ tables with image type columns

This may not be the best size for all your data. You don't want to allow too much text in row space on your data pages if you can't fill it. For example, it would be unwise to set this option to 7000 because you would create many more data pages and extents, thus lengthening the distance a query must travel to retrieve data. Likewise, underestimating this number would create more image data pages and could produce the same problem.

One way to determine an optimal table text in row setting is to estimate the 80th% size of all your image type records and store up to that size in the *_text_in_row setting. First you'll have to load your data and then run this query:

```
select top 20 percent(datalength(points)) from boris.f12 order by
datalength(points) desc
```

This query will list the top 20 percent largest data records in the "points" column of a feature class's feature table. You then pick the smallest value from its result set and use

that for a dbtune entry and reload your data, referencing that keyword. In this case, the 80th% value equaled 455, so you would make this entry in the dbtune:

Keyword	Parameter	Explanation
CARTO	F_INDEX_FID	WITH FILLFACTOR=75 ON FG_CARTO
CARTO	**F_TEXT_IN_ROW**	**455**

Now reload this data referencing this keyword to pick up the new F_TEXT_IN_ROW setting. You could also put this value in the DEFAULTS keyword, but then all data loads would use this size.

```
Shp2sde –o create –l silos,shape –f silos –a all –D carto –s bigboy –u
boris –p badinoff –k CARTO
```

You may want to check if a particular table has been enabled for text in row and what its current size is set to. Use the OBJECTPROPERTY Transact-SQL metadata function:

```
select objectproperty(object_id('boris.f12'),'tabletextinrowlimit')
```

You may not want to reload your data. In this case, you can enable a table for text in row with sp_tableoption and write a query to copy all the existing image data types into a new column:

```
/*
  FeatureTable_Text_in_Row.sql
  script to copy existing points data into new column
  with new text in row value enabled

  SQL Server 2000 Only

  Script Unsupported by ESRI
*/

sp_tableoption 'boris.f12','text in row',455
go
alter table boris.f12
add newpoints image
go
update boris.f12
set newpoints=points
go
alter table boris.f12
drop column points
go
sp_rename 'boris.f12.newpoints','points','column'
go
```

If you have a lot of existing data in SQL Server 7.0 and ArcSDE 8.0.2 and desire to move this data into ArcSDE 8.1 on SQL Server 2000, you'll want to move this data into the new text_in_row format. How you perform your upgrade will determine how your data is moved from a traditional image page into text_in_row.

Text_in_row is enabled with the sp_tableoption command, and 8.1 sets a text_in_row default (in the sde_dbtune table) for all feature tables to 255 bytes. This means that any layer or feature class creation in ArcSDE 8.1 on SQL Server 2000 will use the text_in_row option. To upgrade an existing SQL Server 7.0 database to 8.1 and push your image data into text_in_row format, you can:

- Perform a Geodatabase copy and paste between two ArcSDE servers: one running SQL Server 7.0 and the other SQL Server 2000.
- Reload all your data into ArcSDE on SQL Server 2000.

- Run the SQL Server 2000 upgrade wizard or restore your SQL Server 7.0 databases to a SQL Server 2000 instance. Once you have done this, run the FeatureTable_`Text_in_Row.sql` listed above.

TEXT_IN_ROW for rasters

A raster stored in ArcSDE is composed of four tables. Of those, the sde_blk_<n> table holds the largest image type column. You can enable text_in_row for this table with the dbtune entry BLK_TEXT_IN_ROW. For example:

RASTER	BLK_TEXT_IN_ROW	2000

Triggers

All feature class business tables have a delete update trigger and an insert trigger. The insert trigger ensures that the spatial column cannot have duplicate values in the business table, while the delete–update trigger manages activity against the spatial column in the business, feature, and spatial index tables. You can view these triggers in the SQL Server Enterprise Manager.

These triggers are automatically dropped whenever a feature class, either standalone or in a feature dataset, is multiversioned. They are re-created when a feature class is "unregistered as versioned." If you edit one of these triggers and then multiversion the feature class, the trigger will be dropped.

"I" tables and stored procedures

You'll notice that once you've loaded data or run sdesetupmssql, you will have several "i" tables and stored procedures in your databases. These stored procedures and tables are used for fetching feature IDs for feature classes. Editing these tables or stored procedures is not supported and highly discouraged.

APPENDIX B

The components of the installation

This *ArcSDE Configuration and Tuning Guide for Microsoft SQL Server* covers material after the installation of the ArcSDE software. Users should now be prepared to start the ArcSDE service or make direct client connections to the spatial data store. If the ArcSDE software has not been installed, refer to the *ArcSDE Installation Manual* for directions. This chapter describes what the installation configured on your hard disk, the services it created, what must be present for a successful ArcSDE service start, and how to start the ArcSDE service. Direct connection two-tier ArcSDE setup will not be covered in this chapter; this is covered in Chapter 5, 'Connecting to SQL Server.'

ArcSDE installation basics

The ArcSDE installation moves several files to your hard disk, sets one environment variable, modifies the PATH environment variable, edits two system files, and creates several registry entries. To successfully start the ArcSDE services, there must be:

1. SDE Service file edits:

The services.sde file should contain the name and TCP/IP port number of the ArcSDE service. This file is located in %sdehome%\etc. If the ArcSDE service is called "esri_sde" and resides on port 5151, then this file should read:

```
Esri_sde 5151/tcp
```

2. The Windows 2000 or Windows NT Services file:

This file, "services", is located in %windir%\system32\drivers\etc and must match the entry in point 1. There must be a carriage return after the last line in this file and its counterpart in %sdehome%\etc.

3. Windows Registry:

The ArcSDE Instance should be listed in the windows registry under these keys:

HKEY_LOCAL_MACHINE\Software\ESRI\ArcInfo\ArcSDE\8.0\ArcSDE for SQL Server

HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services

HKEY_LOCAL_MACHINE\SYSTEM\ControlSet001\Services

HKEY_LOCAL_MACHINE\SYSTEM\ControlSet002\Services

The entries in these keys are vitally important. They must be correct for a successful ArcSDE service start. These entries are discussed below in the section entitled "Manually deleting and creating an ArcSDE service."

4. The environment variable %SDEHOME% must point to the installation directory of the ArcSDE software. There can only be one path, and it must contain the ArcSDE bin, etc, lib, tools, and codepage folders.

%SDEHOME%=d:\ArcSDE81

5. There must be an 'sde' database and an 'sde' user. The sde user can either be SQL Server or Windows NT/Windows 2000 authenticated. Either way, this user must have at least create table and create procedure within the 'sde' database.

6. There must be a valid license. Check in the sdelic.log file in %sdehome%\etc. If the service is properly licensed, there will be an entry similar to:

```
Initializing connection to license server (@ZAPHOD)
```

7. The sde and gdb metadata tables must be present in the database, and they must be owned by the sde user. There should be at least 20 'SDE_*' tables (not counting the sde_logfile* tables) and 26 'GDB_*' tables present. There should also be at least 66 'SDE_*' stored procedures in the sde database. The install creates the metadata tables by running the sdesetupmssql.exe executable. See Chapter 5, 'Connecting to SQL Server', for instructions on running this command.

If the seven points above are correct and the ArcSDE service will not start, add the line SET SDEVERBOSE=TRUE in the dbinit.sde file in %sdehome%\etc and try again to start the ArcSDE service. If it fails to start, open the sde.errlog in %sdehome%\etc. There will appear an informative written error message here.

How to start the ArcSDE service

After verification that the above procedure has been correctly performed, it is now possible to start the ArcSDE service to accept incoming connection requests from client programs. If a direct connection to ArcSDE without using the three-tier architecture is desired, refer to Chapter 5, 'Connecting to SQL Server.'

To start the ArcSDE service, there must be a Windows NT or Windows 2000 named 'service.' Use one of the three tools described below to start this service.

Use the "services" utility in the control panel or services ui. Open the services form in the control panel, locate the ArcSDE service name configured during installation, select it, and press the Start button.

Use the "net" command from a command prompt. Open a command prompt and type: net start <service name>

```
c:\ArcSDE81\bin\net start esri_sde
```

Use the sdemon tool located in %SDEHOME%\bin. Open a command prompt and type: sdemon -o start -i <service name> -p <sde dba pass>

```
c:\ArcSDE81\bin\sdemon -o start -i esri_sde -p big.bob
```

How to stop the ArcSDE service

Once the ArcSDE service is successfully started, it can be stopped by employing one of the following methods:

Use the services utility in either the Windows NT control panel or the Windows 2000 services management console under System Administration. Locate the ArcSDE service, select it, and click the stop control.

Use the "net" command from a command prompt. Open a command prompt and type "net stop <service_name>".

```
c:\ArcSDE81\bin\net stop esri_sde
```

Use the sdemon command located in %SDEHOME%\bin. Open a command prompt and type: sdemon -o shutdown -i <service name> -p <sde admin password>

```
c:\ArcSDE81\bin\sdemon -o shutdown -i esri_sde -p
big.bob
```

Manually deleting and creating an ArcSDE service

The ArcSDE installation creates an ArcSDE service. The installation prompted the user for a service name, TCP/IP port number, license manager, and the sde dba password.

However, once the ArcSDE binaries have been installed on disk, it is possible to create a new service manually using the 'sdeservice' tool in %SDEHOME%\bin.

Multiple ArcSDE services are required if there are multiple instances of Microsoft SQL Server running simultaneously on a single server. Microsoft SQL Server 2000 allows multiple installations of the database software on a single server. Similarly, there can be one MS SQL Server 6.5 instance or 7.0 instance running in conjunction with the MS SQL Server 2000 instance. This can be valuable in upgrading a SQL Server 7.0 instance to SQL Server 2000. This tool can also be run as a substitute for uninstalling/reinstalling the ArcSDE software. If it is believed there is a corrupted instance, delete the ArcSDE service by running sdeservice -o delete, then re-create it with sdeservice -o create.

To remove an ArcSDE service:

```
sdeservice -o delete -i esri_sde -p big.bob
```

Running the above command will remove the service entry from the Windows Registry in these keys:

- **\\HKEY_LOCAL_MACHINE\SOFTWARE\ESRI\ArcInfo\ArcSDE\8.0\ArcSDE for SQLServer**
- **\\HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\<service name>**
- **\\HKEY_LOCAL_MACHINE\SYSTEM\ControlSet001**
- **\\HKEY_LOCAL_MACHINE\SYSTEM\ControlSet002**

The minimum required arguments to create a new service are service name (-i esri_sde), license server name (-l zaphod), and sde dba password (-p big.bob).

```
Sdeservice -o create -p big.bob -l zaphod -i esri_sde
```

The above command will re-create all the pertinent registry keys. NOTE: This command's usage has a [-u <service_user>] [-P <service_user_password>] switch. For example:

```
sdeservice -o create -p <DB_Admin_password>
-l <license server name>
[-q] [-H <sde_directory>]
[-d
<ORACLE,SID|ORACLE8I,SID|SQLSERVER|DB2,DB2INSTANCE|INFOR
MIX>]
[-i <service>]
[-u <service_user>] [-P <service_user_password>]
[-s <OLE DB datasource (default: local hostname)>]
```

Use of the -u and -P arguments is not necessary. Use of these switches specifies a domain user and password, not a SQL Server user or password. The ArcSDE install program does not set these switches. Care should be exercised in the use of these switches. The -p argument specifies the sde user's password.

The "DATASOURCE" of an ArcSDE service

The sdeservice command has a DATASOURCE argument with the -s switch. The command sdesetupmssql -o upgrade also has a DATASOURCE argument. A datasource represents a named SQL Server instance. At SQL Server 7.0, the datasource is generally the host name of the server hosting the database.

For example, to create an ArcSDE service specifying a datasource, use the -s switch followed by the name of the database server host. In this case, the datasource is "vogon", meaning the name of the server hosting SQL Server 7 is "vogon." The license manager resides on the server "zaphod."

```
Sdeservice -o create -p big.bob -l zaphod -i esri_sde -s
vogon
```

Using DATASOURCE to separate ArcSDE and the SQL Server host

Although this would probably result in detrimental performance, it is possible to install the ArcSDE software onto a server other than the SQL Server. During the install or when invoking the sdeservice command, simply specify the datasource in the -s switch. For example, it is desired to point to the ArcSDE datasource for testing

purposes at another SQL Server. The ArcSDE application software is located on the server “vogon”, but its datasource points to another SQL Server, “marvin.”

```
sdeservice -o create -p big.bob -l zaphod -i esri_test -
s marvin
```

When starting this new service, the data store holding ArcSDE tables will actually be located on another machine, marvin.

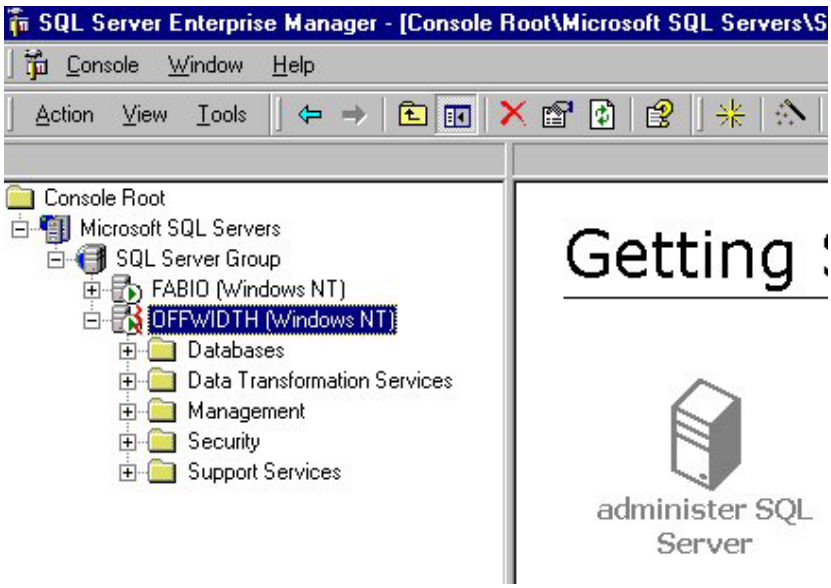
Using DATASOURCE with SQL Server 2000

During the installation of SQL Server 2000, the user is prompted to enter an instance name. This instance name becomes the datasource parameter for ArcSDE 8.1. Also, to run SQL Server 7 and SQL Server 2000 on the same machine, it is necessary to provide different instance names so that two different ArcSDE instances can be created for them. For example, if SQL Server 2000 is installed on server “Vogon,” the instance name on Vogon for SQL Server 2000 becomes “Vogon\SQL2k.” To create two ArcSDE services on Vogon for SQL Server 7 and SQL Server 2000, you would use:

```
sdeservice -o create -i esri_sql -p big.bob -l zaphod -s
vogon /*SQL Server 7*/
sdeservice -o create -i esri_sql -p bob_2k -l zaphod -s
vogon\sql2k /*SQL Server 2000*/
```

Their datasource will be listed in the SQL Server Enterprise Manager under the “SQL Server Group” branch. In the example below, the highlighted name “OFFWIDTH” would be the datasource name for the server.

Figure B.1: The highlighted entry "OFFWIDTH" represents a datasource.



To create an instance with OFFWIDTH as the datasource, the service creation command would look like this:

```
sdeservice -o create -i esri_sql -p big.bob -l fabio -s
offwidth
```

Upgrading SDE 3.0.2 or ArcSDE 8.0.2 to ArcSDE 8.1

If either SDE 3.0.2 or ArcSDE 8.0.2 is installed, the ArcSDE 8.1 installation will attempt to upgrade it to ArcSDE 8.1. **NOTE:** An upgrade requires the use of a sysadmin account.

When the install detects an ArcSDE 8.0.2 instance on the server, it makes an internal connection to SQL Server as a sysadmin fixed server role user, creates new stored procedures for each ArcSDE table in the database, then grants execute on those stored procedures to all users who have permissions to the data. New tables at ArcSDE 8.1 are also created. All data owners are granted create procedure as well. The upgrade process changes the sde user's permissions in the sde database to create table, view, stored procedure, and function (SQL Server 2000 only).

When the install detects an SDE 3.0.2 or 3.0.2.1 instance, it restructures the SDE "F" tables by moving all data stored in the varbinary columns into the points image column. Then it upgrades all layers and metadata to the ArcSDE 8.1 format. Once

the layer migration is complete, ArcSDE rebuilds all indexes on its data. Upgrading from SDE 3.0.2 to ArcSDE 8.1 will probably increase the size of your database.

It is also possible to manually upgrade an 8.0.2 or 3.0.2/3.0.2.1 database to ArcSDE 8.1 with the sdesetupmssql.exe command in %SDEHOME%\bin. The upgrade option requires the command to be run as a sysadmin user.

To upgrade an ArcSDE 8.0.2 instance to ArcSDE 8.1

Install ArcSDE 8.1. The installer will detect the existence of ArcSDE 8.0.2 and attempt to upgrade an 8.0.2 schema.

To upgrade, the installer runs the command "sdesetupmssql –o upgrade." This command can also be run manually. The command must be run by a sysadmin account.

After the command finishes, check the upgrade.log file in %SDEHOME%\etc. If the file does not end with a report of success, rerun the command manually until it does. **NOTE: If you have a customized dbtune.sde file from ArcSDE 8.0.2, this file must be resident in the %SDEHOME%\etc folder to successfully run sdesetupmssql –o upgrade.** The installer does present you with the option of copying an existing dbtune.sde into the new installation.

To upgrade an SDE 3.0.2 or SDE 3.0.2.1 instance to ArcSDE 8.1

This example presumes that you will upgrade an SDE 3.x instance running on SQL Server 7.0 to ArcSDE 8.1.

Shutdown your SDE 3.x instance.

Install MDAC 2.5. Microsoft Data Access Components (MDAC) 2.5 is available through Microsoft.

Install SQL Server Service Pack 3. This service pack is available through Microsoft.

Install ArcSDE 8.1. The installer will attempt to remove the version 3.x software. The postinstallation will attempt to upgrade the version 3.x instance to version 8.1. The upgrade must be run as a sysadmin account.

The upgrade will take some time and depends upon how much data you have and how large your tables are. If the upgrade returns an error, check the upgrade.log file in %SDEHOME%\etc, correct the file, and start the command (sdesetupmssql –o upgrade) again. This command is located in %SDEHOME%\bin. The –o upgrade option must be run as a sysadmin account. Rerun the command manually until it does. **NOTE: If you have a customized dbtune.sde file from ArcSDE 8.0.2, this file must be resident in the %SDEHOME%\etc folder to successfully run sdesetupmssql –o upgrade.** The installer does present you with the option of copying an existing dbtune.sde into the new installation.

Upgrading an ArcSDE 8.1 instance on SQL Server 7 to SQL Server 2000

It is recommended that users first allow the installation to upgrade any existing SDE 3.0.2.x or ArcSDE 8.0.2 instances to ArcSDE 8.1 before migrating data into SQL Server 2000. ArcSDE 8.1 leverages the SQL Server 2000 Text in Row option for each table with an image data type. Since all F tables and many network tables use image types, it is beneficial to use this setting. See the SQL Server 2000 Books Online for

information regarding sp_tableoption's 'text in row' setting. The option, as it relates to ArcSDE 8.1, is discussed in Appendix A.

To upgrade ArcSDE from SQL Server 7 to SQL Server 2000

There are two basic options:

Allow the ArcSDE install to upgrade the ArcSDE 8.0.2 or SDE 3.x instance to version 8.1 or do this manually with the sdesetupmssql tool in %sdehome%\bin. To perform an upgrade, the user must initiate this command as a member of the sysadmin fixed server role. This can be either the 'sa' user or an Nt/Win2k user.

If you have the licensing available, create an instance of ArcSDE 8.1 on SQL Server 2000. Next, if ArcGIS is installed, use the Geodatabase copy and paste tool to copy the data from the version 8.1 instance on SQL Server 7 to the version 8.1 instance on SQL Server 2000. Using Geodatabase copy and paste will preserve relationship classes, geometric networks, and other advanced Geodatabase models. This will require starting two instances of ArcSDE. The image data types (all ArcSDE geometry) will be created with a default text in row setting of 255. The new dbtune table sets this automatically.

If ArcGIS is not installed or you don't have the licensing to maintain two instances of ArcSDE, use SQL Server tools such as backup and restore, sp_detach_db and sp_attach_db, DTS, or the upgrade wizard. However, the image data will not be upgraded to the text in row format. Appendix A shows a t-sql example of moving image type data into the SQL Server 2000 text_in_row format.

When upgrading from SQL Server 7 to SQL Server 2000 use the sp_dbcmptlevel stored procedure to set certain database behaviors to be compatible with the specified earlier version of SQL Server. A server upgraded from SQL Server 7 to SQL Server 2000 will have all its databases compatibility level set to level 80. If a server is upgraded from SQL Server 6.0 or 6.5 will have its databases set to the SQL Server version which the database was last used, either 60 or 65. All databases in SQL Server 2000 must have their compatibility level set to 80. To verify that all databases are set to compatibility level 80, execute the following procedure:

EXEC sp_dbcmptlevel 'database name'

To change to a different compatibility mode, run the following command:

EXEC sp_dbcmptlevel 'database name' compatibility-mode

The sp_dbcmptlevel command is covered in greater detail in SQL Server Books Online.

In addition the sdesetupmssql –o upgrade command must be run on upgraded instances of SQL Server 7 to SQL Server 2000. To perform an upgrade, the user must initiate this command as a member of the sysadmin fixed server role. This can be either the 'sa' user or an Nt/Win2k user The syntax of this command is:

-o upgrade [-H <sde_directory>] [-u <DB_Admin_user>] [-p <DB_Admin_password>]

[-s datasource] [-N] [-q]

Upgrade examples

NOTE: These are simple examples of what you can do to upgrade your instance. There are many ways to do this, the most simple of which, and the only supported method, is to let the installer upgrade an 8.0.2 instance to 8.1 or manually run the sdesetupmssql –o upgrade command.

Example one: ArcSDE 8.0.2 on MS SQL Server 7.0 to ArcSDE 8.1 on SQL Server 2000 (different servers).

Install SQL Server 2000 and ArcSDE 8.1 onto the new server.

Create the same databases and users as presently configured on the SQL Server 7.0, ArcSDE 8.0.2 instance.

Start an ArcSDE 8.1 service on the SQL Server 2000 server.

Use the copy and paste tool from ArcCatalog to move the data from the 8.0.2 instance on SQL Server 7.0 to the ArcSDE 8.1 instance on SQL Server 2000.

Note: You must have enough ArcSDEServer licenses to do this.

After the copy and paste has finished, backup the database.

Example two: ArcSDE 8.0.2 on MS SQL Server 7.0 to ArcSDE 8.1 on SQL Server 2000 (same server)

Install ArcSDE 8.1 onto the server. The installer upgrades the version 8.0.2 instance to ArcSDE 8.1. The installer will execute the command "sdesetupmssql –o upgrade" as a sysadmin user. You must supply this username and password during the postinstallation portion of the install. This command is located in the %SDEHOME%\bin folder of the ArcSDE 8.1 installation.

Start the ArcSDE 8.1 service to verify that the upgrade succeeded. If the service will not start, check the "upgrade.log" file in %SDEHOME%\etc for information. Correct any errors listed in this log and manually re-execute the sdesetupmssql –o upgrade command as a sysadmin user. Repeat this process until the upgrade succeeds.

Install SQL Server 2000 onto the server. Shutdown the ArcSDE 8.0.2 service. Use a SQL Server tool to move the 7.0 databases to the 2000 server. You can use backup and restore, sp_detach_db and sp_attach_db, DTS, or the SQL Server 2000 Copy Database Wizard to do this. See the SQL Server 2000 Books Online for this information and Chapter 8, 'Backup and recovery', of this guide for an example of using backup and restore to move databases. If you restore a backup of your SQL Server 7.0 databases to a SQL Server 2000 instance, you must manually execute the sp_dbcmptlevel stored procedure to set your database's compatibility level to '80' (SQL Server 2000). You must do this for all restored databases. For example:

```
sp_dbcmptlevel sde,'80'
```

Modify your existing ArcSDE 8.1 service to point to the SQL Server 2000 datasource. Use the sdeservice –o modify tool to change the –r DATASOURCE argument of your service. You can also delete and re-create your service. See the section above, Manually deleting and creating an ArcSDE service, for information on how to do this.

Start your ArcSDE 8.1 service.

Example three: ArcSDE 8.0.2 on MS SQL Server 7.0 to ArcSDE 8.1 on SQL Server 2000 (same server, maintaining both the version 8.0.2 instance and version 8.1 instance). This upgrade scenario is more involved. In order to maintain both the older service and the version 8.1 service on the same machine, you'll need to do the following:

Install SQL Server 2000 onto the server. Installing SQL Server 2000 onto the same machine as SQL Server 7.0 is possible with SQL Server 2000's support of multiple instances. See the section in this guide on multi-instance support for more information.

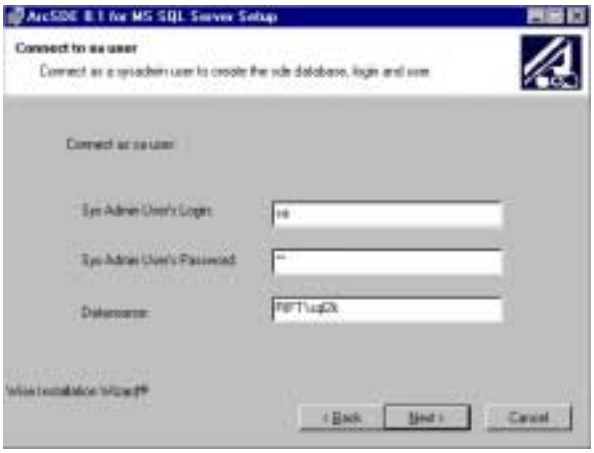
Before installing ArcSDE 8.1, shutdown ArcSDE 8.0.2. Make a copy of your older SDEHOME (all the directories and files located beneath SDEHOME). You will need to copy it back after the 8.1 installation is finished.

Install ArcSDE 8.1. It will remove your ArcSDE 8.0.2 installation, delete your ArcSDE 8.0.2 service, and remove your port entry in the %windir%\system32\drivers\etc\services file. Do not drop the sde database. SDEHOME will be set to the 8.1 install directory, and your path will include the 8.1 install directory\bin folder.

Start the ArcSDE 8.1 postinstallation. **Click Next>** in the *Welcome Screen.*

Click Next> in the *Define User Environment Screen* to create the sde login, and sde database and to add the login to the database as a user. If you precreated the sde login and database and have granted the login access to the sde database, **press Skip** in this screen and **click Next** in *Setup Configuration Files Screen* and go to step 8.

In the next form, input a sysadmin user and password and your SQL Server 2000 datasource. The wizard will proceed through the steps of creating an sde login and database. The following screenshot displays a SQL Server 2000 datasource on a machine called "Rift."



Click Next> through the wizard until you get to the *Repository Setup* form.

The *Repository Setup* form will create the sde metadata if it does not exist or upgrade a version 3.0.2.x sde or version 8.0.2 ArcSDE instance if it is present. **Click Next>. Click Skip** if your version 8.1 metadata already exists and proceed to step 10.

Now connect as a sysadmin user and input your SQL Server 2000 datasource. Again, see the above screenshot for how this should look. **NOTE: If you input your SQL Server 7.0 Datasource here, you will upgrade your version 8.0.2 instance to version 8.1** Finish the version 8.1 Postinstallation process of creating and starting a service.

Open a DOS prompt and change directories into the ArcSDE 8.0.2 %SDEHOME%\bin folder. Re-create your ArcSDE 8.0.2 service with the sdeservice –o create command. Make sure you get the datasource argument correct; your datasource should be the name of the SQL Server 7.0 server. **NOTE: You cannot point to two instances of ArcSDE at the same datasource.** Don't forget to use the –H flag of the sdeservice –o create command to register the correct SDEHOME for version 8.0.2.

```
C:\ArcSDE802\bin\sdeservice –o create –p <sde user password in ss7> -l
<license server> -H c:\ArcSDE802 –i esri_sql802 –s <server name>
```

Open the services file in %WINDIR%\system32\drivers\etc and readd your ArcSDE 802 port. For example, you'll need to readd this line:

```
esri_sql802             5151/tcp
```

Before starting both services, make sure your services are correct by running sdeservice –o list within the bin folder of each SDEHOME, for example, for ArcSDE 8.0.2:

```
C:\ArcSDE802\bin>sdeservice -o list

SDE service Information
------------------------------------------------------------------------
Service Name:                   ArcSde Service(esri_sql802)
Service SDEHOME:                C:\ArcSDE802
Service License Server:         @fabio
Service Datasource:             rift
Service Status:                 SERVICE_STOPPED
```

For ArcSDE 8.1:

```
C:\sde81\bin>sdeservice -o list

SDE service Information
------------------------------------------------------------------------
Service Name:                   ArcSde Service(sde81)
Service SDEHOME:                C:\sde81
Service License Server:         @fabio
Service Datasource:             rift\sql2k
Service Status:                 SERVICE_STOPPED
```

Note that in this example, each service has its own SDEHOME. Also note that each service has its own datasource argument. The ArcSDE 8.0.2 service points to the **`rift`** datasource, while the ArcSDE 8.1 service points to the **`rift\sql2k`** SQL Server 2000 datasource.

Now start one or both instances. NOTE: You'll have to have the proper licensing to do this. You can run both instances to test your data under the 8.1 configuration. You can move data from the 8.0.2 instance by copying and pasting with the ArcCatalog or exporting with sdeexport, sde2shp, etc.

Example four: ArcSDE 8.0.2 or SDE 3.0.2.x on SQL Server 7.0 to ArcSDE 8.1 on SQL Server 2000 (same machine)

Make backups of all your sde and spatial databases using the SQL Server 7.0 backup tool.

Shutdown your ArcSDE 8.0.2 or SDE 3.0.2.x instance. **Uninstall the SDE software. If you have customized dbtune.sde, you must save a copy of this file!**

Install SQL Server 2000. Create an identical set of logins on the SQL Server 2000 instance as exist on the SQL Server 7.0 instance.

Do one of the following three steps:

Restore all the SQL Server 7.0 databases to the SQL Server 2000 instance. Run sp_dbcmptlevel for each database you've restored:

```
sp_dbcmptlevel sde,'80'
```

or

Run sp_change_users_login 'update_one','<user>','<user>' for each user, in each, database. A detailed example of this is given in Chapter 8, 'Backup and recovery'.

Run sp_detach_db on the sde and any other spatial database in the SQL Server 7.0 instance. In the SQL Server 2000 instance, run sp_attach_db to reattach them to the newer server. Again, you must run sp_change_users_login.

or

Use the SQL Server 2000 Copy Database Wizard to copy the SQL Server 7.0 databases to the SQL Server 2000 instance.

Install ArcSDE 8.1. Run the postinstallation utility and let it upgrade your 8.0.2 databases from SQL Server 7.0 to ArcSDE 8.1 in SQL Server 2000. **NOTE**: If the upgrade utility throws an error, note the error, check the upgrade.log file in %SDEHOME %\etc, correct the listed error, and manually restart sdesetupmssql –o upgrade. Sdesetupmssql –o upgrade must be run as a sysadmin user. For example:

```
sdesetupmssql –o upgrade –u sa –p sa.password –s rift\sql2k –H
c:\sde81\sqlexe
```

You can run this command as many times as needed until the upgrade completes successfully. **If you customized your dbtune.sde file from 8.0.2, you must run the upgrade process, either from the postinstall or using the sdesetupmssql –o upgrade command, with the customized dbtune.sde file in %sdehome%\etc**. The installer will prompt you to use a different dbtune.sde file during the postinstallation.

7.0 Start your sde service: `Net start esri_sde`

Troubleshooting the ArcSDE service startup

Once you have the ArcSDE software on disk and Microsoft SQL Server installed, you should not have to remove either of them. Follow the process listed below, and you should be able to start your ArcSDE service once you're done.

1. Check for the existence of an 'sde' user and 'sde' database: Verify that the sde database exists and that the sde login is a user in this database by logging in as 'sde.' **Make note of the sde user's password**. The sde user must have at least create table, create view, and create procedure in the sde database. This can be verified by right-clicking the database in the Enterprise Manager and selecting Properties. Then click the Permissions tab and check what permissions the 'sde' user has in the sde database. Verify that the sde and gdb metadata tables exist by examining the sde database. There should be approximately 20 SDE_* tables, 26 GDB_* tables, and 66 SDE_* stored procedures. If these tables and stored procedures do not exist, they must be created by running the sdesetupmssql.exe command located in %SDEHOME%\bin. See Chapter 5, 'Connecting to SQL Server,' for instructions on running this command.

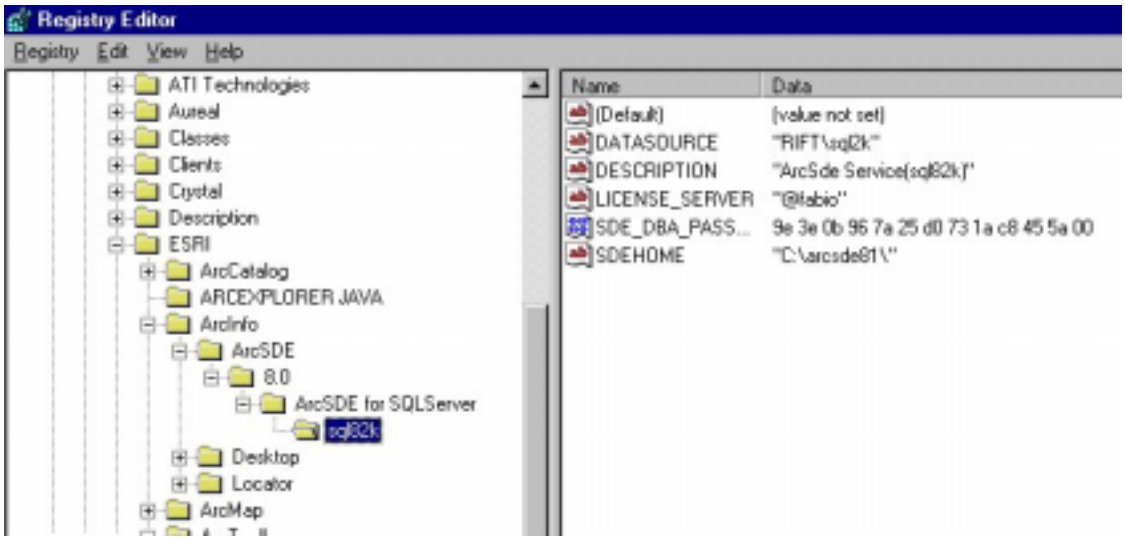
2. Check that the ArcSDE service is listed in both %SDEHOME%\etc\services.sde and %WINDIR%\system32\drivers\etc\services. The same service should be present in both. If this service is the last entry in these files, put a carriage return after it (blank space). The entries should look like the following:

```
Esri_sde 5151/tcp
```

3. The ArcSDE service must be correctly entered in the Windows Registry. The most critical locations in the registry are:

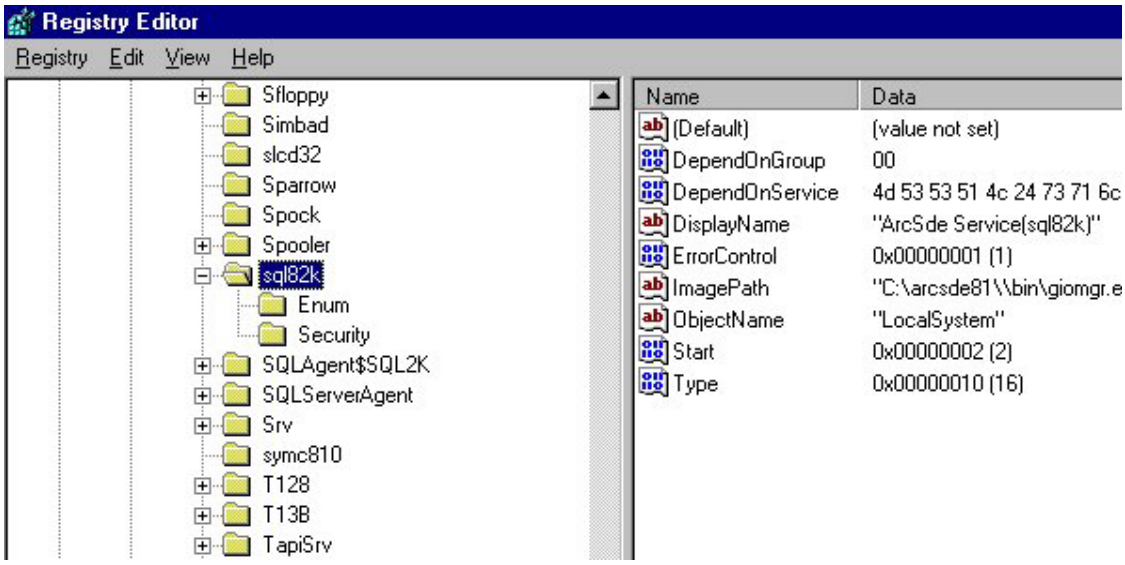
HKEY_LOCAL_MACHINE\Software\ESRI\ArcInfo\ArcSDE\8.0\ArcSDE for SQL Server—In this key, there will be the name of your service, its registered datasource, its license server, DBA_password, and SDEHOME. See Figure B.1. This service must be located under the “ArcSDE for SQL Server” key. Its DATASOURCE must point to your named SQL Server. A valid license_server should be listed with the @<name> convention. SDEHOME should point to an actual location on disk.

Figure B.2: A Service Entry in the HKEY_LOCAL_MACHINE\Software key



HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services—In this key, there will be a Windows representation of the actual service. See Figure B.2. The DependOnService represents the -d flag of the sdeservice -o create command and identifies a dependency between the ArcSDE service and another Windows NT service, most likely the MSSQLSERVER process. The ImagePath gives the path to the giomgr.exe executable, the io manager.

Figure B.3: A service entry in the HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet key



Instead of editing the registry directly, delete the service with sdeservice -o delete. Since this is a DOS executable, be aware of the path from which it is run. Run it from %SDEHOME%\bin to ensure that it picks up the correct libraries. See the above section, 'Manually deleting and creating an ArcSDE service', for more information. Confirm that the password and data source arguments are entered correctly.

4. Confirm that there is a valid license manager process running on your network. The license manager may reside locally on the server, or it may be accessed on the network. You identify the name of the license host when installing or with the -l argument to the sdeservice -o create command. Verify your services' license server by typing sdeservice -o list from a DOS prompt:

```
C:\snoopyng>sdeservice -o list -i esri_sql
SDE service Information
-------------------------------------------------------
-----
Service Name: ArcSde Service(esri_sql)
Service SDEHOME: C:\arcsde81\
Service License Server: @fabio
Service Status: SERVICE_STOPPED
```

In this example, the service "esri_sql" has its license service running on fabio. Fabio is identified with the "@" symbol. Sometimes, you'll correctly setup the license server but for some reason not be able to access a license. In this case, ArcSDE appears to hang. You can tell if this has happened on Windows NT by examining the Windows NT Service Control panel's start and stop buttons. If both are grayed out, your service cannot access a valid license. You will also see an 'unable to initialize

connection to license server error' message in the sdelic.log file in %SDEHOME%\etc.

5. In some rare instances, there will be an incorrect installation of MDAC, or it can become corrupted. MDAC is the Microsoft Data Access Components and is used to access the data stores in SQL Server through OLE DB. ArcSDE 8.1 supports MDAC 2.6 and 2.5. To verify which MDAC is installed, check the version of msado15.dll in program files\common files\system\ado or by downloading the component checker from Microsoft. File version 2.5.4403 and up are supported.

Now attempt to start the service. If it still cannot start, add the line "set SDEVERBOSE=TRUE" in the dbinit.sde file in %sdehome%\etc. The word 'set' must be in lower case. Now try again to start the service. Then open the sde.errlog file in %sdehome%\etc. There will be an informative error message written here.

APPENDIX C

Storing raster data

A raster is a rectangular array of equally spaced cells, which taken as a whole represent thematic, spectral, or picture data. Raster data can represent everything from qualities of land surface such as elevation or vegetation to satellite images, scanned maps, and photographs.

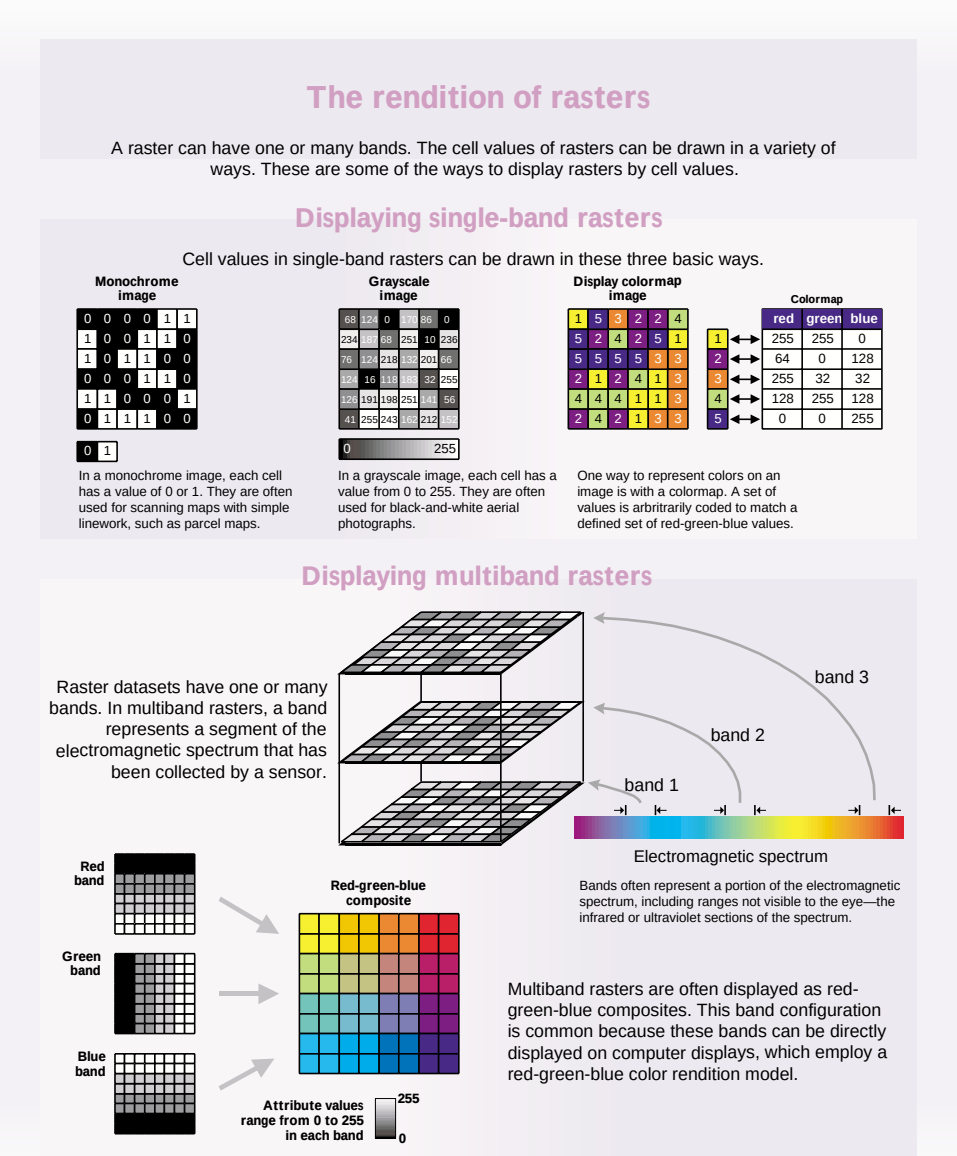
You are probably familiar with raster formats such as tagged image file format (TIFF), Joint Photographic Experts Group (JPEG), and Graphics Interchange Format (GIF) that your Internet browser renders. These rasters are composed of one or more bands. Each band is segmented into a grid of square pixels. Each pixel is assigned a value that reflects the information it represents at a particular position.

For an expanded discussion of the type of raster data supported by ESRI products, review Chapter 9, 'Cell-based modeling with rasters', in *Modeling Our World*.

ArcSDE stores raster datasets similar to the way it stores compressed binary feature classes (see Appendix A, 'ArcSDE compressed binary'). A raster column is added to a business table, and each cell of the raster column contains a reference to a raster stored in a separate raster table. Therefore, each row of a business table references an entire raster.

ArcSDE stores the raster bands in the raster band's table. ArcSDE joins the raster band's table to the raster table on the raster_id column. The raster band table's raster_id column is a foreign key reference to the raster table's raster_id primary key.

ArcSDE automatically stores any existing image metadata such as image statistics, colormaps, or bit-masks in the raster auxiliary table. The rasterband_id column of the raster auxiliary table is a foreign key reference to the primary key of the raster band's table. ArcSDE joins the two tables on this primary/foreign key reference when accessing a raster band's metadata.



The raster block's table stores the pixels of each raster band. ArcSDE tiles the pixels into blocks according to a user-defined dimension. ArcSDE does not have a default dimension, however, applications that store raster data in ArcSDE do. ArcToolbox and ArcCatalog, for example, use default raster block dimensions of 128-by-128 pixels per block. The dimensions of the raster block along with the compression method, if one is specified, determine the storage size of each raster block. You should select raster block dimensions that, combined

with the compression method, allow each row of the raster block table to fit within SQL Server.

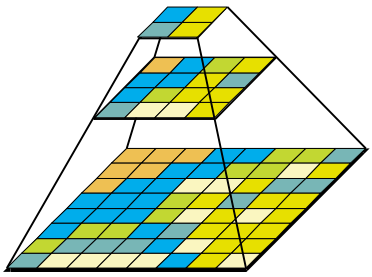
The raster block's table contains the rasterband_id column, which is a foreign key reference to the raster band table's rasterband_id primary key. ArcSDE joins these tables together on the primary/foreign key reference when accessing the blocks of the raster bands.

ArcSDE populates the raster blocks table according to a declining resolution pyramid. The height of the pyramid is determined by the number of levels, specified by application. The application such as ArcToolbox or ArcCatalog may allow you to define the levels, or it may request that ArcSDE calculate them, or it may offer both possible choices.

The pyramid begins at the base, or level 0, which contains the original pixels of the image. The pyramid proceeds toward the apex by coalescing four pixels from the previous level into a single pixel at the current level. This process continues until less than four pixels remain or until ArcSDE exhausts the defined number of levels.

The apex of the pyramid is reached when the uppermost level has less than four pixels. The addition of the pyramid levels increases the number of raster blocks by one third. However, since it is possible for the user to specify the number of levels, the apex of the pyramid may not be obtainable, and therefore the size increase of the raster blocks table is restricted to the number of resolution levels added.

Figure C.1: When you build a pyramid more rasters are created by progressively downsampling the previous level by a factor of two until the apex. As the application zooms out and the raster cells grow smaller than the resolution threshold, ArcSDE selects a higher level of the pyramid. The purpose of the pyramid is to optimize display performance.



The pyramid allows ArcSDE to provide the application with a constant resolution of pixel data regardless of the rendering window's scale. Data of a large raster transfers quicker to the client when a pyramid exists since ArcSDE can transfer fewer cells of a reduced resolution.

Raster schema

When you import a raster into an ArcSDE database, ArcSDE adds a raster column to the business table of your choice. You may name the raster column whatever you like, so long as

it conforms to SQL Server's column naming convention. ArcSDE restricts one raster column per business table.

The raster column is a foreign key reference to the raster_id column of the raster table created during the addition of the raster column. Also joined to the raster table's raster_id primary key, the raster band table stores the bands of the image. The raster auxiliary table, joined one-to-one to the raster bands table by rasterband_id, stores the metadata of each raster band. The rasterband-id also joins the raster band's table to the raster blocks table in a many-to-one relationship. The raster blocks table rows store blocks of pixels, determined by the dimensions of the block.

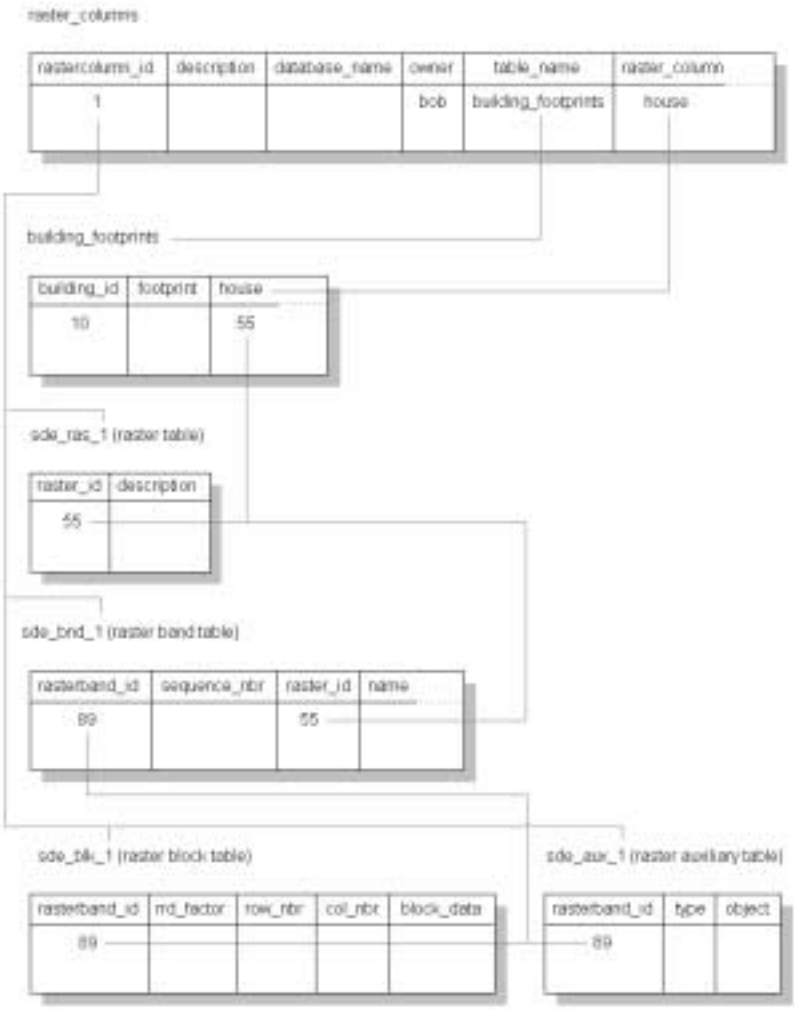


Figure C.2: When ArcSDE adds a raster column to a table, it records that column in the sde user's raster_columns table. The rastercolumn_id is used in the creation of the table names of the raster, raster bands, raster auxiliary, and raster blocks table.

The sections that follow describe the schema of the tables associated with the storage of raster data. Refer to Figure C.2 for an illustration of these tables and the manner in which they are associated with one another.

RASTER_COLUMNS table

When you add a raster column to a business table, ArcSDE adds a record to the RASTER_COLUMNS system table maintained in the sde user's schema. ArcSDE also creates four tables to store the raster images and metadata associated with each one.

NAME	DATA TYPE	NULL?
rastercolumn_id	INT(4)	NOT NULL
description	VARCHAR(65)	NULL
database_name	VARCHAR(32)	NULL
owner	VARCHAR(32)	NOT NULL
table_name	VARCHAR(160)	NOT NULL
raster_column	VARCHAR(32)	NOT NULL
cdate	INT(4)	NOT NULL
config_keyword	VARCHAR(32)	NULL
minimum_id	INT(4)	NULL
base_rastercolumn_id	INT(4)	NOT NULL
rastercolumn_mask	INT(4)	NOT NULL
srid	INT(4)	NULL

Raster columns table

- rastercolumn_id (SE_INTEGER_TYPE)—the table's primary key.
- description (SE_STRING_TYPE)—The description of the raster table.
- database_name (SE_STRING_TYPE)—The name of the database.
- owner (SE_STRING_TYPE)—The owner of raster column's business table.
- table_name (SE_STRING_TYPE)—The business table name.
- raster_column (SE_STRING_TYPE)—The raster column name.
- cdate (SE_INTEGER_TYPE)—The date the raster column was added to the business table.
- config_keyword (SE_STRING_TYPE)—The DBTUNE configuration keyword whose storage parameters determine how the tables and indexes of the raster are stored in the SQL Server database. For more information on DBTUNE configuration keywords and their storage parameters, review Chapter 3, 'Configuring DBTUNE storage parameters.'
- minimum_id (SE_INTEGER_TYPE)—Defined during the creation of the raster, establishes value of the raster table's raster_id column.
- base_rastercolumn_id (SE_INTEGER_TYPE)—If a view of the business table is created that includes the raster column, an entry is added to the RASTER_COLUMNS table. The raster column entry of the view will have its own

rastercolumn_id. The base_rastercolumn_id will be the rastercolumn_id of the business table used to create the view. This base_rastercolumn_id maintains referential integrity to the business table. It ensures that actions performed on the business table raster column are reflected in the view. For example, if the business table's raster column is dropped, it will also be dropped from the view (essentially removing the view's raster column entry from the RASTER_COLUMNS table).

- rastercolumn_mask (SE_INTEGER_TYPE)—Currently not used, maintained for future use.
- rid (SE_INTEGER_TYPE)—The spatial reference ID (srid) is a foreign key reference to the SPATIAL_REFERENCES table. For images that can be geo-referenced, the SRID establishes the X and Y offset translation factor and the scale factor for storage of the image coordinates into the 32-bit integer ArcSDE coordinate storage system. It also stores the coordinate reference system the image was created under.

Business table

In the example that follows, the fictitious BUILD_FOOTPRINTS business table contains the raster column house_image. This is a foreign key reference to the raster table created in the user's schema. In this case the raster table contains a record for each raster of a house. It should be noted that images of houses cannot be georeferenced. Therefore, the SRID column of the RASTER_COLUMN record for this raster is NULL.

NAME	DATA TYPE	NULL?
Building_id	INT(4)	NOT NULL
building_footprint	INT(4)	NOT NULL
house_picture	INT(4)	NOT NULL

BUILDING_FOOTPRINTS business table with house image raster column

- building_id (SE_INTEGER_TYPE)—the table's primary key
- building_footprints (SE_INTEGER_TYPE)—a spatial column and foreign key reference to a feature table containing the building footprints
- house_image (SE_INTEGER_TYPE)—a raster column and foreign key reference to a raster table containing the images of the houses located on each building footprint

Raster table (SDE_RAS_<rastercolumn_id>)

The raster table, created as SDE_RAS_<raster_column_id> in the SQL Server, stores a record for each image stored in a raster column. Raster_column_id is assigned by ArcSDE whenever a raster column is created in the database. A record for each raster column in the database is stored in the ArcSDE RASTER_COLUMNS system table maintained in the sde user's schema.

NAME	DATA TYPE	NULL?
Raster_id	INT(4)	NOT NULL
raster_flags	INT(4)	NULL
description	VARCHAR(65)	NULL

Raster description table schema (SDE_RAS_<raster_column_id)

- raster_id (SE_INTEGER_TYPE)—The primary key of the raster table and unique sequential identifier of each image stored in the raster table
- raster_flags (SE_INTEGER_TYPE)—A bitmap set according to the characteristics of stored image
- description (SE_STRING_TYPE)—A text description of the image (not implemented at ArcSDE 8.1)

Raster band table (SDE_BND_<rastercolumn_id>)

Each image referenced in a raster may be subdivided into one or more raster bands. The raster band table, created as SDE_BND_<rastercolumn_id>, stores the raster bands of each image stored in the raster table. The raster_id column of the raster band table is a foreign key reference to the raster table's raster_id primary key. The rasterband_id column is the raster band table's primary key. Each raster band in the table is uniquely identified by the sequential rasterband_id.

NAME	DATA TYPE	NULL?
rasterband_id	INT(4)	NOT NULL
sequence_nbr	INT(4)	NOT NULL
raster_id	INT(4)	NOT NULL
name	VARCHAR(65)	NULL
band_flags	INT(4)	NOT NULL
band_width	INT(4)	NOT NULL
band_height	INT(4)	NOT NULL
band_types	INT(4)	NOT NULL
block_width	INT(4)	NOT NULL
block_height	INT(4)	NOT NULL
block_origin_x	FLOAT(8)	NOT NULL
block_origin_y	FLOAT(8)	NOT NULL
eminx	FLOAT(8)	NOT NULL
eminy	FLOAT(8)	NOT NULL
emaxx	FLOAT(8)	NOT NULL
emaxy	FLOAT(8)	NOT NULL
cdate	INT(4)	NOT NULL
mdate	INT(4)	NOT NULL

Raster band table schema

- rasterband_id (SE_INTEGER_TYPE)—The primary key of the raster band table that uniquely identifies each raster band.
- sequence_nbr (SE_INTEGER_TYPE)—An optional sequential number that can be combined with the raster_id as a composite key as a second way to uniquely identify the raster band.
- raster_id (SE_INTEGER_TYPE)—The foreign key reference to the raster table's primary key. Uniquely identifies the raster band when combined with sequence_nbr as a composite key.
- name (SE_STRING_TYPE)—The name of the raster band.
- band_flags (SE_INTEGER_TYPE)—A bitmap set according to the characteristics of the raster band.
- band_width (SE_INTEGER_TYPE)—The pixel width of the band.
- band_height (SE_INTEGER_TYPE)—The pixel height of the band.
- band_types (SE_INTEGER_TYPE)—A bitmap band compression data.
- block_width (SE_INTEGER_TYPE)—The pixel width of the band's tiles.

- block_height (SE_INTEGER_TYPE)—The pixel height of the band's tiles.
- block_origin_x (SE_FLOAT_TYPE)—The leftmost pixel.
- block_origin_y (SE_FLOAT_TYPE)—The bottom-most pixel.

If the image has a map extent, the optional eminx, eminy, emaxx, and emaxy will hold the coordinates of the extent.

- eminx (SE_FLOAT_TYPE)—The band's minimum x-coordinate.
- eminy (SE_FLOAT_TYPE)—The band's minimum y-coordinate.
- emaxx (SE_FLOAT_TYPE)—The band's maximum x-coordinate.
- emaxy (SE_FLOAT_TYPE)—The band's maximum y-coordinate.
- cdate (SE_FLOAT_TYPE)—The creation date.
- mdate (SE_FLOAT_TYPE)—The last modification date.

Raster blocks table (SDE_BLK_<rastercolumn_id>)

Created as SDE_BLK_<rastercolumn_id>, the raster blocks table stores the actual pixel data of the raster images. ArcSDE evenly tiles the bands into blocks of pixels. Tiling the raster band data enables efficient storage and retrieval of the raster data. The raster blocks can be configured so that the records of the raster block table fit with a SQL Server data block, avoiding the adverse effects of data block chaining.

The rasterband_id column of the raster block table is a foreign key reference to the raster band table's primary key. A composite unique key is formed by combining the rasterband_id, rrd_factor, row_nbr, and col_nbr columns.

NAME	DATA TYPE	NULL?
rasterband_id	INT(4)	NOT NULL
rrd_factor	INT(4)	NOT NULL
row_nbr	INT(4)	NOT NULL
col_nbr	INT(4)	NOT NULL
block_data	image	NOT NULL

Raster block table schema

- rasterband_id (SE_INTEGER_TYPE)—The foreign key reference to the raster band table's primary key.
- rrd_factor (SE_INTEGER_TYPE)—The reduced resolution dataset factor determines the position of the raster band block within the resolution pyramid. The resolution pyramid begins at 0 for the highest resolution and increases until the raster's bands lowest resolution level has been reached.
- row_nbr (SE_INTEGER_TYPE)—The block's row number.

- col_nbr (SE_INTEGER_TYPE)—The block's column number.
- block_data (SE_BLOB_TYPE)—The block's tile of pixel data.

Raster band auxiliary table (SDE_AUX_<rastercolumn_id>)

The raster band auxiliary table, created as SDE_AUX_<rastercolumn_id>, stores optional raster metadata such as the image color map, image statistics, and bit masks used for image overlay and mosaicking. The rasterband_id column is a foreign key reference to the primary key of the raster band table.

NAME	DATA TYPE	NULL?
rasterband_id	INT(4)	NOT NULL
type	INT(4)	NOT NULL
object	image	NOT NULL

Raster auxiliary table schema

- rasterband_id (SE_INTEGER_TYPE)—The foreign key reference to the raster band table's primary key
- type (SE_INTEGER_TYPE)—A bitmap set according to the characteristics of the data stored in the object column
- object (SE_BLOB_TYPE)—May contain the image color map, image statistics, etc.