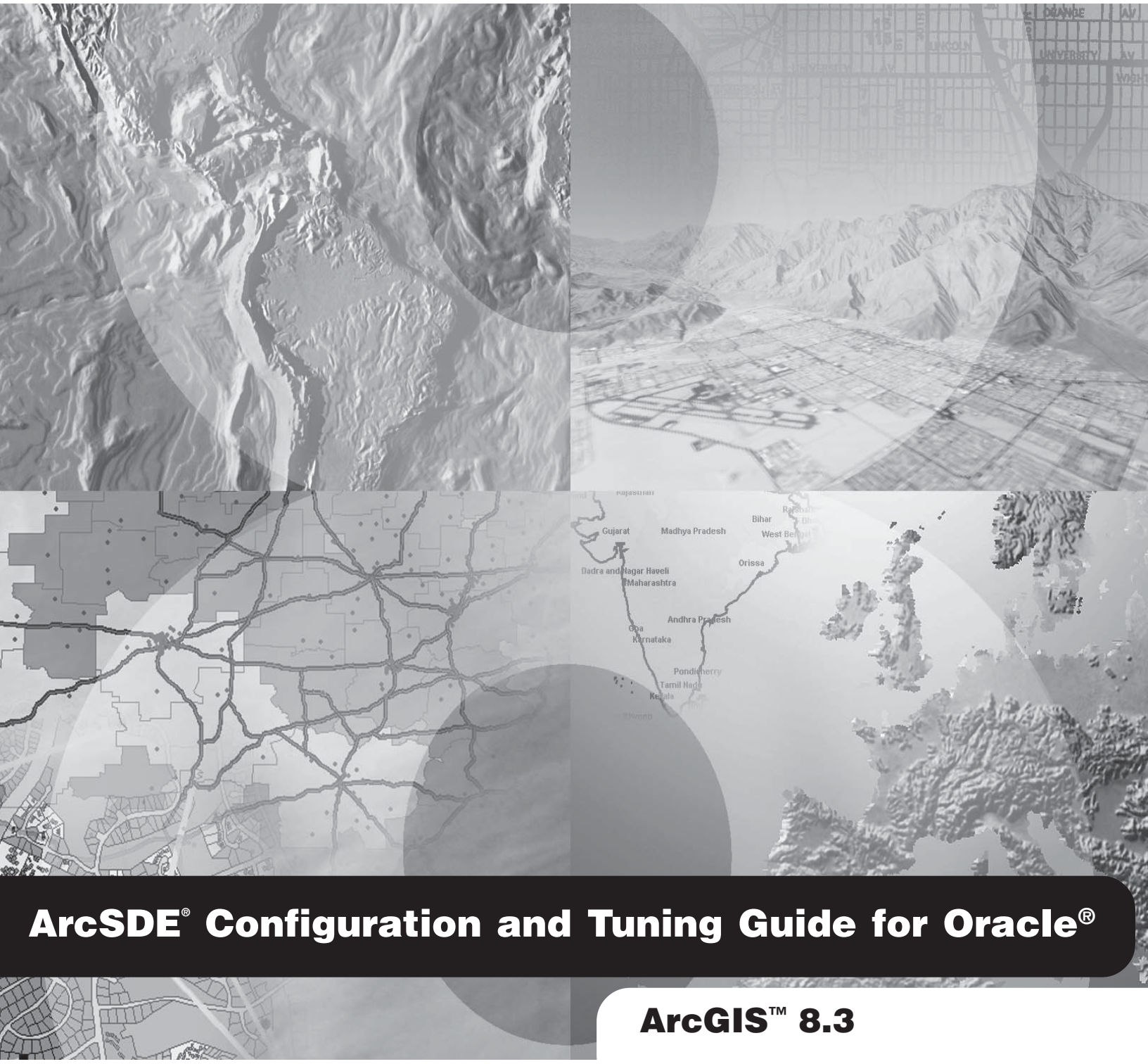


ArcSDE® Configuration and Tuning Guide for Oracle®

ArcGIS™ 8.3

Contents

CHAPTER 1

Getting started

Creating and populating a geodatabase is arguably a simple process, especially if you use ESRI's ArcCatalog™ or ArcToolbox™ to load the data. So why is there a configuration and tuning guide? Well, while database creation and data loading can be relatively simple, the resulting performance may not be acceptable. It requires some effort to build a database that performs optimally. Also, as an Oracle® user, you have some choices for storing the geometry of your spatial data.

This book provides instruction for configuring the physical storage parameters of your data in the database management system (DBMS) as well as providing information about the available options you have to store the geometry. This book also provides some important guidelines for configuring and tuning the Oracle instance itself.

Tuning and configuring the Oracle instance

Building an efficient geodatabase involves properly tuning and configuring the Oracle instance and proper arrangement and management of the database's tables and indexes. Chapter 2, 'Essential Oracle configuring and tuning', teaches you how to do just that.

Chapter 2 lists the necessary steps to create a geodatabase. You will learn how to properly

- Create an Oracle database.
- Create the tablespaces that will store your tables and indexes.
- Tune the Oracle instance that will mount and open the database.
- Manage the optimization statistics of the tables and indexes after they have been created and populated.

Arranging your data

Every table and index created in a database has a storage configuration. How you store your tables and indexes affects your database's performance.

DBTUNE storage parameters

How is the storage configuration of the tables and indexes controlled? ArcSDE™ reads storage parameters from the DBTUNE table to define physical data storage parameters of ArcSDE tables and indexes. The storage parameters are grouped into configuration keywords. You assign configuration keywords to your data objects (tables and indexes) when you create them from an ArcSDE client program.

Prior to ArcSDE 8.1, configuration keywords were stored in a dbtune.sde file maintained under the ArcSDE etc directory. The dbtune.sde file is still used by ArcSDE 8.1 as the initial source of storage parameters. When the ArcSDE 8.1 sdesetupora* command executes, the configuration parameters are read from the dbtune.sde file and written into the DBTUNE table.

It should also be noted that ArcSDE 8.1 has simplified the storage parameters. Rather than matching each Oracle storage parameter with an ArcSDE storage parameter, the ArcSDE storage parameters have evolved into configuration strings and represent the entire storage configuration for a table or index. Pre-ArcSDE 8.1 storage parameters are automatically converted to the new simpler ArcSDE 8.1 storage parameters. The ArcSDE storage parameter holds all the Oracle storage parameters of an Oracle CREATE TABLE or CREATE INDEX statement.

The sdedbtune command has been introduced at ArcSDE 8.1 to provide the ArcSDE administrator with an easy way to maintain the DBTUNE table. The sdedbtune command exports and imports the records of the DBTUNE table to a file in the ArcSDE etc directory.

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

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

Spatial data storage choices

The DBTUNE storage parameter GEOMETRY_STORAGE allows you to select from three possible spatial column storage formats.

The three possible storage formats are:

- ArcSDE compressed binary with LONG RAW. The ArcSDE geometry is stored in a ‘LONG RAW’ column in a separate feature table. A business table's spatial column is a foreign key reference to the records of the feature table. This is the default spatial storage format for ArcSDE.

- ArcSDE compressed binary with binary large object (BLOB). The schema of this storage format is the same as the previous one except for the fact that the geometry is stored in the BLOB data type.

- Oracle Spatial geometry type. Starting at Oracle8*i* this object-relational model extends the database model to include an SDO_GEOMETRY type. Under this storage format, the spatial column is an SDO_GEOMETRY data type, and no foreign key reference to another table storing a geometry column is required.

These spatial storage choices are discussed more fully in this book.

Appendix C, 'ArcSDE compressed binary', describes the ArcSDE compressed binary for both LONG RAW and BLOB.

Appendix D, 'Oracle Spatial geometry type', describes the Oracle Spatial storage format supported by ArcSDE.

Creating spatial data in an Oracle database

ArcCatalog and ArcToolbox are graphical user interfaces (GUIs) specifically designed to simplify the creation and management of a spatial database. These applications provided the easiest method for creating spatial data in an Oracle database. With these tools you can convert existing ESRI® coverages and shapefile format into ArcSDE feature classes. You can also import an existing ArcSDE export file containing the data of a business table, feature class, or raster column.

Multiversioned ArcSDE data can be edited directly with either the ArcCatalog or ArcMap™ GUI.

An alternative approach to creating spatial data in an Oracle database is to use the administration tools provided with ArcSDE.

Chapter 4, 'Managing tables, feature classes, and raster columns', describes the methods used to create and maintain spatial data in an Oracle database.

Connecting to Oracle

Starting at ArcSDE for Oracle8*i*™ there are two different ways to connect to the Oracle instance. ArcSDE clients can either connect to the ArcSDE service, or they may now connect directly to the Oracle instance.

Under the traditional ArcSDE three-tiered architecture, the ArcSDE client connects to the ArcSDE service, and the ArcSDE service spawns a dedicated *gsrvr* process that connects to the Oracle instance. The *gsrvr* process brokers the spatial data between the ArcSDE client and the Oracle instance. The ArcSDE service and the *gsrvr* processes typically reside on the Oracle host machine, while ArcSDE clients are typically on remote machines.

Under the new two-tiered architecture, ArcSDE clients can connect directly to an Oracle instance. Essentially the functionality of the *gsrvr* process has been moved into the ArcSDE client. Typically, the ArcSDE clients run on remote desktop machines, while the Oracle instance runs on a server class machine.

Note: The direct connection method can only be used starting at Oracle8*i*.

It is possible to use a combination of these two architectures to connect to an Oracle instance. With a mixed configuration, some of the ArcSDE clients may be connected directly to the Oracle instance, while others connect through an ArcSDE service.

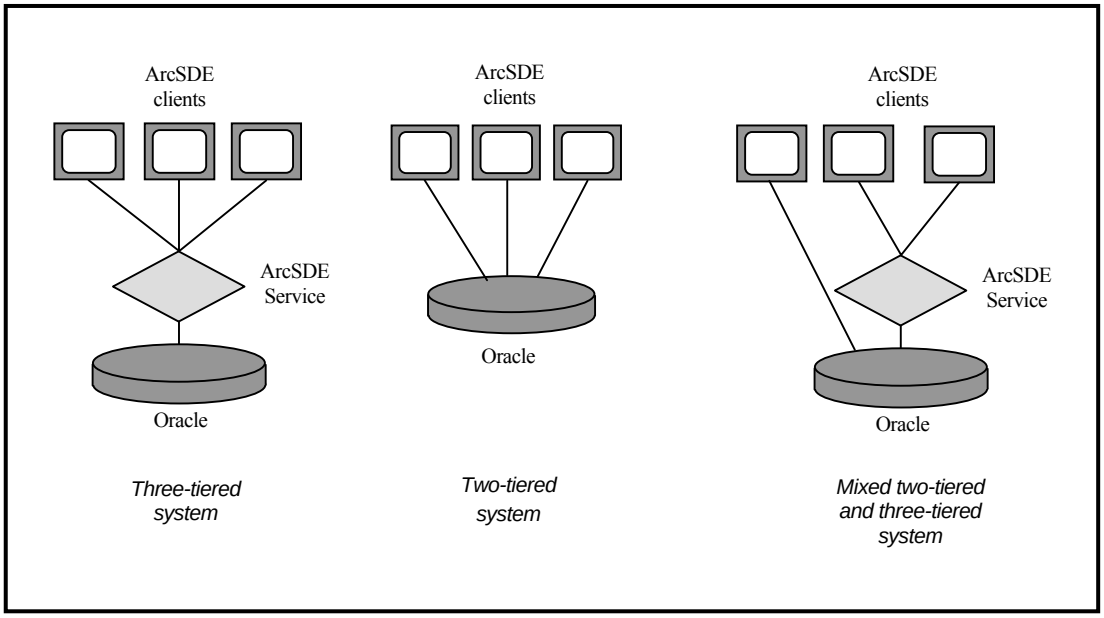


Figure 1.1 With the traditional three-tiered architecture, the ArcSDE client applications connect to the ArcSDE service, which in turn connects to the Oracle instance. Within the two-tiered architecture, the ArcSDE client applications connect directly to the Oracle instance. Under a mixed architecture,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

National language support

If you intend to support a database that does not use the Oracle default 7-bit United States ASCII English (US7ASCII) character set, you will have to take a few extra steps in creating the Oracle database. You will also need to set the national language system environment of the client applications.

Chapter 6, 'National language support', describes how to configure the Oracle database and set up the application environment.

Backup and recovery

Developing and testing a backup strategy is every bit as important as the effort put into creating it. A good backup strategy protects the database in the event of a media failure.

Chapter 7, 'Backup and recovery', lists the ArcSDE files that must be included as part of the regular Oracle backup. In addition, suggested Oracle reference materials are listed for further reading.

CHAPTER 2

Essential Oracle configuring and tuning

The performance of an ArcSDE service depends on how well you configure and tune Oracle. This chapter provides basic guidelines for tuning an Oracle database for use with an ArcSDE application server. It assumes that you have a basic understanding of the Oracle data structures, such as tablespaces, tables, and indexes, and that you are proficient with Structured Query Language (SQL). We encourage you to refer to Oracle's extensive documentation, in particular *Oracle Server Administrator's Guide, Oracle Concepts Guide,* and *Oracle Server Tuning,* for your appropriate Oracle release.

Tuning goals

A well-tuned Oracle database makes optimum use of available central processing unit (CPU) time and memory while minimizing disk input/output (I/O) contention. What constitutes *optimal* however, is a matter of judgment. Will your system be used by a large number of users making many concurrent edits to data? Or will your system be used more frequently in a readonly fashion, by just a few users who wish to access your large datasets?

Disk I/O contention provides the most challenging performance bottleneck. Other than purchasing faster disk drives and additional network cards, the solution to this problem lies in minimizing disk I/O and balancing it throughout the file system—reducing the possibility of one process waiting for another to complete its I/O request. This is often referred to as "waiting on I/O".

How much time should you spend tuning?

The appreciable difference between a well-tuned database and one that is not depends on how it is used. A database created and used by a single user does not require as much tuning as a database that is in constant use by many users. The reason is quite simple—the more people using a database, the greater the contention for its resources.

By definition, tuning is the process of sharing resources among users by configuring the components of a database to minimize contention and remove bottlenecks. The more people you have accessing your databases, the more effort is required to provide access to a finite resource.

Experienced database administrators know that each additional hour spent will often return a lesser gain in performance. Eventually, they reach a point of diminishing returns, where it is impractical to continue tuning; instead, they continue to monitor the server and address performance issues as they arise.

Planning your ArcSDE installation

Before installing Oracle and creating the database, decide where to position the software and the files of the Oracle database. The Oracle installer program will request this information.

If you have already installed Oracle and created your database files, you should still read the sections that follow. Although it involves more effort, you can move the Oracle database files after they have been created.

The physical components of an ArcSDE service and the underlying Oracle database, as they exist on any given file system, include the ArcSDE and Oracle software and all of the physical files (data files, redo log files, and control files) of the Oracle database. Each of the components is described below.

Software

The software includes both Oracle and ArcSDE. The ArcSDE software occupies approximately 32 MB of space. Oracle software can occupy different amounts of disk space depending on the version of Oracle and which components you choose to install. Please see your Oracle installation notes for further details.

Control files

The control files maintain an inventory of an Oracle database's overall physical architecture. If they are all lost or destroyed, you must recover the database from your last full backup. During the creation of the database, create at least three control files on different disk drives. If a disk containing a control file fails, the Oracle server must be shutdown. After the disk drive has been repaired, use the operating system copy command to copy another control file to the restored location. Starting at Oracle8*i*, control files are more dynamic and initially require about 4 MB of disk space and can grow to more than 10 MB depending on the activity of your database.

The initialization parameter CONTROL_FILE_RECORD_KEEP_TIME controls the size of the control files. By default, this parameter is set to 7—instructing Oracle to overwrite its reusable section every seven days. For more information on initialization parameters, refer to ‘Setting the Oracle initialization parameters’ later in this chapter.

Online redo log files

The online redo log files record the changes made to the database. Oracle requires a database to have at least three online redo log file groups present. Our analysis has found that for ArcGIS™ Desktop applications, the Oracle server performs reliably when the Oracle database has at least three online redo log file groups present.

An Oracle database that receives regular edits (inserts, updates, or deletes) has highly active online redo log files. Writes to the current online redo log file occur according to the following schedule:

- The log buffer becomes one third full.
- Any session issues a commit.
- Every three seconds if the data block buffer contains nonlogged dirty blocks.

It is important to physically separate the online redo log files from other data files that also experience high rates of I/O. Whenever possible, create the log files on their own disk drives or with other relatively static files.

Each time a log file fills up, a log file switch occurs and Oracle begins writing to the next log file. (Remember that log files occurred in mirrored *groups* of one or more identical log files. When a log switch occurs, all log files in a group are closed and all log files in the next group are opened. Thus a log file switch really refers to switching all members of a log file group to the next group.) A checkpoint must occur after each log file switch. Much happens within the database during a checkpoint, so you will want to lower the frequency of this event. You can and should force checkpoints to occur only after a log switch by setting the initialization parameters LOG_CHECKPOINT_INTERVAL and LOG_CHECKPOINT_TIMEOUT to 0.

The size and number of online redo log files in your database depend on the type of database. This section will describe three basic kinds of databases:

- A newly created database.
- An OnLine Transaction Processing (OLTP) database. A multiversioned ArcSDE database that is constantly edited while it is being queried is an example of an OLTP database.
- A read-only database, meaning a database that, once loaded, receives changes at posted intervals. An ArcIMS database is an example of a read-only database.

Be sure to monitor the log file usage during data loading activities or normal operation.

Spatial database

With the vast amount of spatial data available as coverage and shapefile format, many spatial databases are mass populated immediately following their initial creation. For this type of database, create three log file groups each containing at least one very large redo log file. If possible, place all log files on a disk drive separate from all other data files. In this situation, it is not unreasonable to create log files in excess of 1 GB.

After you have finished loading the data, connect to Oracle as the database administrator (DBA) and issue a checkpoint with ALTER SYSTEM CHECKPOINT. Then create new, smaller log files. The size and number of the log files depend on what kind of database it will become, either OLTP or read-only (see below).

When loading a database you may turn off archiving. You obtain a performance gain by eliminating the periodic copy to the archive log destination following a log switch. It is just as easy to recover the database from your load scripts and source data as it is to read the changes stored in the archive logs. Remember to turn archiving on after the database has been loaded if it is going to be an OLTP database.

OLTP database

For these types of databases, the redo log files should be large enough to delay the checkpoint as much as possible. Redo log files should also be mirrored to provide maximum protection against loss of transaction data.

If you are archiving the redo log files, create 3 to 10 redo log file groups having log files about 50 MB in size. If possible, place them on disk drives that experience very low I/O. The archive log file destination should also be placed on a separate and protected disk drive.

If you are not going to archive your log files, your total space given to redo log files should be enough to store all log entries generated between full database backups. If you create 3 redo log file groups that are 250 MB each, you will have a total of 750 MB of redo log space. If you experience a disk failure on a disk storing one of your Oracle data files, you may still be able to recover the changes made since the last backup. It is recommended that you turn archiving on following the initial creation of the database.

In either case, you should mirror the online redo log files. Place the mirror copy of the online redo log file on a physically separate disk drive from the original copy.

Note: ESRI does *not* recommend that NOARCHIVELOG be the normal operating mode of an OLTP database due to the risk of losing committed transactions in the event of media failure.

Read-only databases

Some ArcSDE databases become relatively static following their creation. Such databases receive posted intervals of changes over their lifetime. For this type of database, create three 50 MB online redo log files. Since they're used infrequently, positioning is not as critical as for the other two types of databases just described.

Tablespace data files

The tablespace represents Oracle's logical storage container. Each tablespace has assigned to it one or more physical data files.

System tablespace

The system tablespace stores Oracle's data dictionary. Each time Oracle parses a SQL statement, it checks metadata concerning data objects referenced by the statement from the data dictionary. Among other things, Oracle ensures the data objects actually exist and the user has the proper privileges.

Place the SYSTEM tablespace on a disk of moderate activity. The default size of the SYSTEM tablespace for Oracle8*i* is 175 MB.

Rollback tablespaces

The rollback tablespaces store rollback segments, which maintain the undo image needed to roll back aborted transactions. Rollback segments also provide read consistency for queries started prior to a transaction.

Determine the storage parameters of the rollback tablespace and the rollback segments by the type of transactions using them. ArcSDE has three basic categories of transactions.

Loading transactions—The initial loading of data into an ArcSDE database generally entails converting an existing storage format such as an ArcSDE coverage, shapefile, SDE export file, or file format provided by a data vendor into an ArcSDE feature class.

To maximize throughput of the load process, ArcSDE provides a commit interval, allowing you to batch inserts. The commit interval also serves to regulate transaction size. The commit interval defaults to 5,000 features and is set with the ArcSDE giomgr.defs AUTOCOMMIT parameter. Refer to *Managing ArcSDE Services* for more information on the AUTOCOMMIT parameter. When loading data this interval uses approximately 2 MB of rollback segment space. Therefore, we recommend setting the extent size of the rollback segment to 2 MB when you are loading data into your database. Create a 50 MB rollback tablespace and assign two rollback segments to that tablespace.

If more than two processes are used to load data, create an additional rollback tablespace on a separate disk drive and create two rollback segments on it. The addition of the rollback tablespace reduces contention for the rollback segment header:

```
create tablespace rbs '<path to data file>' size 50M extent management local
uniform size 2M;

create rollback segment ro1 tablespace rbs storage (minextents 10);

create rollback segment ro2 tablespace rbs storage (minextents 10);
```

Version edit transactions—The data maintenance transactions of the ArcGIS system tend to be smaller than loading transactions. We recommend that the extent size of the rollback segments assigned to these transactions be approximately 256 KB.

Create a 50 MB rollback tablespace and assign four rollback segments to that tablespace. Each rollback segment can optimally support the concurrent transactions of six users, so this configuration will support 24 concurrent users:

```
create tablespace rbs1 '<path to data file>' size 50M extent management local
uniform size 256K;

create rollback segment r01 tablespace rbs1 storage (minextents 20);

create rollback segment r02 tablespace rbs1 storage (minextents 20);

create rollback segment r03 tablespace rbs1 storage (minextents 20);

create rollback segment r04 tablespace rbs1 storage (minextents 20);
```

You may need to create additional rollback segment tablespaces depending on the number of concurrent transactions you expect the database to support. Each transaction is assigned to a rollback segment. A rollback segment may be assigned more than one transaction at a time. Oracle evenly distributes transactions among the available rollback segments.

After the database has been started and is in use for a period of time, query the V$ROLLSTAT table to determine if the transaction waits are greater than 4 percent of the transaction gets:

```
select ((sum(waits) / sum(gets)) * 100) from v$rollstat;
```

If the result of this query is larger than 4 percent, you should create additional rollback tablespaces on another disk drive and add more rollback segments to them.

Version compress transactions—Periodically, the ArcSDE administrator is required to compress the states of a multiversioned database to reduce the number of records held by the delta tables. To guarantee the consistency of the database, the transactions of the compress operation are large, requiring an equally large transaction. Therefore, if you are maintaining a multiversioned database you should create a separate rollback tablespace that is at least 300 MB in size and assign one rollback segment to it. Set the name of this rollback segment in the COMPRESS_ROLLBACK_SEGMENT storage parameter of the DEFAULTS dbtune configuration keyword. If this parameter is not set, the next available online rollback segment will be used. If the rollback segment is not large enough, the compress operation will fail since the transaction will be forced to roll back.

The extent size of the version compress rollback segment should be set to 100 MB:

```
create tablespace big_o_rb_tspace '<path to data file>'
size 301M extent management local uniform size 100M;

create rollback segment big_o_rbspace tablespace big_o_rb_tspace
storage (minextents 3);
```

ESRI recommends that you *do not* set the rollback segment optimal storage parameter. The optimal parameter causes a rollback segment to shrink whenever it exceeds the optimal threshold. Constantly shrinking the rollback segments will significantly degrade performance.

As a rule, if you find that your transactions are rolled back because your rollback segments fill up, you should reduce the size of your transactions or increase the size of the rollback tablespaces rather than set the optimal parameter. Large transactions delay recovery, increase overhead for queries that must access them for read consistency, and increase overhead for other transactions that must allocate additional extents. ArcSDE allows you to limit the size of your transactions by setting the giomgr.defs AUTOCOMMIT parameter.

Temporary tablespace

Oracle applications need temporary tablespace whenever they perform a sort that exceeds the memory allocated to the sort area. The sort area is memory allocated to the user's Program Global Area (PGA) and is controlled by the init.ora parameter SORT_AREA_SIZE. Sorts occur when indexes are created (Oracle:CREATE INDEX statement), statistics are generated (ANALYZE statement), and queries require on-the-fly sorting (SELECT statements that include table joins, ORDER BY clauses, and GROUP BY clauses).

When establishing a new database, the temporary tablespace will need to be large enough to create the indexes. Oracle requires twice as much temporary space to create the indexes as it does to store it. Therefore, determine the size of your largest index.

If you are using the ArcSDE compressed binary format to store your spatial data, the S<n>_IX1 index on the spatial index table is likely to be your largest index. Refer to

Appendix A, 'Estimating the size of your tables and indexes', for information on determining the size of your indexes.

If you are storing your spatial data as an Oracle Spatial data type, refer to Appendix A, 'Tuning Tips and Sample SQL Scripts', of the *Oracle Spatial User's Guide and Reference* for more information on sizing temporary tablespace for the construction of the Oracle Spatial data type indexes.

After the data has been loaded and the indexes created, temporary tablespace is used for data sorts. Temporary tablespace is used when sorts exceed the PGA's sort area, which is allocated according to the init.ora SORT_AREA_SIZE parameter.

The SORT_AREA_SIZE should be increased to perhaps as much as 100 MB during the construction of a database, when it is needed to create large indexes and analyze tables. During database construction, it is generally a single session that performs the work. If multiple sessions are used to construct the database, the SORT_AREA_SIZE should be scaled down to avoid memory contention.

After the database is ready for use, reduce the SORT_AREA_SIZE to the Oracle default value of 64 KB if you are storing your spatial data as compressed binary.

The temporary tablespace can be mixed with other data files of higher I/O since there is a low risk of disk I/O contention.

By default, the Oracle installation process creates the temporary tablespace stored in a data file that has logging turned on. Since this tablespace holds temporary data, for which there is no requirement to maintain an archive, drop the tablespace and re-create it with a tempfile that has logging turned off.

Always create the temporary tablespace with the syntax:

```
CREATE TEMPORARY TABLESPACE <name> TEMPFILE '<path to temporary data file>'
SIZE 300M EXTENT MANAGEMENT LOCAL UNIFORM SIZE 272K;
```

rather than the Oracle default syntax:

```
CREATE TABLESPACE <name> DATAFILE '<path to temporary data file>' SIZE 300M
TEMPORARY DEFAULT STORAGE (INITIAL 128K NEXT 128K MINEXTENTS 1 MAXEXTENTS 121
PCTINCREASE 0);
```

If you use the latter syntax, the temporary segments are logged and must be recovered in the event of a media failure. The former (and preferred syntax) is created with locally managed extents, which Oracle allocates more efficiently.

Set the extent size of the temporary tablespace to a multiple of the SORT_AREA_SIZE plus the size of an Oracle data block. For example, if you have set your SORT_AREA_SIZE to the Oracle default 64 KB, and you use the recommended 16 KB DB_BLOCK_SIZE, you should set your temporary tablespace extents to either 272 KB, 336 KB, or 400 KB. Always use an extent size that is at least four times larger than your SORT_AREA_SIZE plus a data block. Doing so reduces the allocation of temporary extents.

ArcSDE system tablespaces

The ArcSDE system tablespaces store the ArcSDE and geodatabase system tables and indexes created by the ArcSDE sdesetupora* command. The number and placement of the tablespaces depend on what you intend to use the ArcSDE database for.

The placement of these tables and their indexes is controlled by the storage parameters of the dbtune DATA_DICTIONARY configuration keyword. The DATA_DICTIONARY keyword is used exclusively for the creation of the ArcSDE and geodatabase system tables.

Multiversioned databases that support ArcGIS OLTP applications have a highly active state-tree. The state-tree maintains the states or the change history of all editing operations that have occurred on tables and feature classes registered as multiversioned. Four ArcSDE system tables—STATES, STATE_LINEAGES, MVTABLES_MODIFIED, and VERSIONS—maintain the transaction information of the versioned database's state-tree. In this type of environment these four tables and their indexes have their own DATA_DICTIONARY configuration keyword storage parameters.

In an active multiversioned database, the STATES_LINEAGE table can easily grow to between one and two million records, occupying between 26 and 52 MB of tablespace. The STATES table is much smaller, storing between 5,000 to 10,000 records, occupying between 2 to 4 MB of tablespace. The MVTABLES_MODIFIED table typically has between 50,000 and 100,000 records occupying between 1 to 2 MB of tablespace. The VERSIONS table is usually quite small with less than 100 rows occupying about 64 KB.

For most applications you can probably create a tablespace for the ArcSDE system tables and one for their indexes on different disk drives and set the DATA_DICTIONARY parameters accordingly. For highly active editing ArcGIS applications, the STATES, STATES_LINEAGE, and MVTABLES_MODIFIED tables and their indexes need to be created in separate tablespaces and positioned across the file system to minimize disk I/O contention.

If you are not using a multiversioned database, the aforementioned tables are dormant, in which case the tables can be stored with the other ArcSDE system tables and indexes.

The remainder of the ArcSDE and geodatabase system tables store information relating to schema changes. They are relatively small and have a low frequency of I/O. They should be grouped together in two separate tablespaces—one for tables and one for indexes—and positioned with other tablespaces of high activity.

To summarize, if you are creating an active multiversioned database, create a 70 MB tablespace to store ArcSDE tables. On a separate disk drive create a 30 MB tablespace for the tables indexes.

If you are not going to use a multiversioned database, reduce the extent sizes of the STATE_LINEAGES, STATES, and MVTABLES_MODIFIED tables and their indexes to 40 KB. Create two 5-MB tablespaces on separate disk drives—one for the tables and one for the indexes.

For more information about the DATA_DICTIONARY configuration keyword, see Chapter 3, ‘Configuring DBTUNE storage parameters’.

Business table and index tablespaces

The Oracle installation creates the USER tablespace as the default user tablespace. There is no requirement by Oracle that you use the USER tablespace, and you may drop it if you wish. If you are building a sizable, permanent spatial database, create tablespaces with names reflecting the data they will store.

The B_STORAGE DBTUNE storage parameter holds business table storage parameters.

The 15 MB INDX tablespace created by the Oracle installation process can be used to store indexes. However, for large spatial databases you will need to create index tablespaces whose name and size reflect the indexes they store. You may drop the Oracle-generated INDX tablespace if you do not intend to use it.

The B_INDEX_USER DBTUNE storage parameter holds the storage parameters of the business table indexes that you create.

Arranging the database components

Minimizing disk I/O contention is achieved by balancing disk I/O across the file system—positioning frequently accessed "hot" files with infrequently accessed "cold" files. Estimate the size of all the database components and determine their relative rates of access. Position the components given the amount of disk space available and the size and number of disk drives. Diagramming the disk drives and labeling them with the components help keep track of the location of each component. Have the diagram handy when you create the Oracle database.

Establish a maximum data file size

Choose the maximum size of a data file. It is a good policy to set a maximum size limit for your data files because doing so facilitates interchanging them. After the database has been in use for some time and a "normal" pattern of usage has been established, heavily accessed data files sharing the same disk drive need to be separated. They can only be interchanged with medium- or low-accessed data files of a similar size (unless you have a lot of free space on your disk drives).

A common maximum size is 512 MB, but with the advent of more advanced backup technology, larger sizes are being adopted. Some operating systems have a maximum file size of 2 GB, so check your operating system documentation for further details.

Separate tables from their indexes

Each time Oracle accesses the index to locate a row, it must then access the table to fetch the referenced row. The disk head must travel between the index and the table if they are stored on the same disk.

Whenever possible, store indexes and their corresponding tables in separate tablespaces, on physically separate disk drives. Doing this eliminates disk head travel that occurs when the data blocks of the table and its indexes are accessed at the same time.

Establish the threshold data object size

As a rule, small data objects, whether they are tables or indexes, are stored together in the same tablespace, while larger data objects are stored by themselves in their own tablespace. Decide how large a data object must be before it requires its own tablespace. Generally, the threshold data object size corresponds in part to the maximum data file size. Data objects capable of filling the maximum size data file should be stored in their own tablespace. Data

objects that approach this limit should also be considered. Since each new tablespace requires its own data file, you should strive to keep the number of tablespaces to a minimum to reduce the number of data file headers that must be updated during a database checkpoint.

Separate the tables and indexes into those that require their own tablespaces and those that will be grouped together. Never store tables and their indexes together in the same tablespace.

Store small tables and indexes by access

Base the decision of which small tables to store together in the same tablespace on expected access. Store tables of high access in one tablespace and tables of low access in another. Doing so allows you to position the data files of the high-access tablespaces with low-access data files. This same rule applies to indexes. They, too, should be divided by access.

Create tablespaces containing a single table or index according to the size of the table or index they contain. A tablespace may be large enough that it requires several data files. Therefore, if the maximum data file size allowed is 2 GB and a tablespace must store a table or index that will grow to 7 GB, the tablespace is created with a 2 GB data file. Then the tablespace is altered three times to add two 2 GB data files and one 1 GB data file.

Example:

```
create tablespace roads datafile '/gis1/oradata/roads1.dbf' size 2048M
extent management local uniform size 1024M;

alter tablespace roads add datafile '/gis2/oradata/roads2.dbf' size 2048M;

alter tablespace roads add datafile '/gis3/oradata/roads3.dbf' size 2048M;

alter tablespace roads add datafile '/gis4/oradata/roads4.dbf' size 1024M;
```

Find the sum of the sizes of the objects that you will store together to determine the size of their tablespaces. If you expect the table or index to grow in the future, be sure to allow for that as well.

Maintain a small number of extents

Keep the number of tablespace extents for tables and indexes less than 1,000 to minimize overhead associated with each additional extent. In fact, creating a table or index with MAXEXTENTS set to UNLIMITED can render a database unusable if the table or index acquires more extents than the database can manage (~10,000). As a general rule of thumb, you should try to keep the number of extents to a minimum; however, you do not need to fanatically maintain a single extent for each object.

Balance the file loads

Once you have estimated the size of the data files, determine where to position them on the file system. This section provides a list of guidelines that you may not be able to follow in its entirety, given the number and size of your disk drives. The guidelines have been listed in order of importance—from the greatest to the least.

Store the online redo log files on their own disk drive. In a database that is frequently edited, the online redo log files are the most active in terms of I/O. If you cannot position them on their own disk drive, store them with other files that experience relatively low rates of I/O.

After the indexes have been constructed, ArcSDE does not use temporary tablespace if SORT_AREA_SIZE has been set to the recommended 64 KB. Therefore, the temporary tablespace can be positioned with other data files of high activity, provided your other applications do not use it.

Keep the rollback segment data files separate from the redo log files. The rollback segments are frequently accessed when a database is edited. Try to separate these data files from other highly active data. Doing so improves the rate at which Oracle is able to process transactions.

Position the system tablespace data file with other data files that experience high I/O activity. The access to these data dictionary tables is moderately low because their data is cached in the shared pool and the buffer cache.

Position your business table and index data files according to their expected I/O. If you expect a particular data file to experience a high degree of I/O, try to position it alone on its own disk drive or with other data files of low to moderate activity.

The spatial index table of the ArcSDE compressed binary storage format is written to whenever new features are added to a feature, but the table is never read. The spatial index table's S<n>_IX1 index is a covering index; therefore, Oracle reads the values from the index and never accesses the table. Since the table is never read from, the I/O is low, so the positioning of this table is unimportant.

Creating the ArcSDE database

The database should be created after the location of the database files has been determined. At the very least, you should assign the locations for the rollback segment, temporary and system tablespace data files, and the control files and online redo log files.

Once you have mapped out the way you want the files to be arranged on disk, create the database. First, install the Oracle and ArcSDE software.

Refer to the Oracle installation guide for instructions on installing the Oracle software and the ArcSDE installation guide for instructions on installing the ArcSDE software.

The Oracle installation provides you with the opportunity to create the Oracle database following the installation of the software. If you elect to do so, select the custom database option so that you can position the Oracle files on the disks according to your layout.

Starting with Oracle8*i,* the database creation scripts are not generated automatically. Instead, you are given the choice of generating the scripts and creating the database later or creating the database immediately without creating the scripts. If you want the scripts available as a record of how the database was created, then you will have to generate the scripts and run them following the completion of the installation process.

The control files, online redo log files, and Oracle database tablespaces (system, rollback, and temporary) are created during the Oracle installation process. The USER and INDX tablespaces created by the Oracle installation process may be dropped if you do not wish to use them.

Setting the data block size

During the installation process, you will be prompted to enter the data block size. Data blocks are the Oracle atomic unit of data transfer. After a database has been created, you cannot change the size of the data blocks. It is important that it be set correctly at this time.

The minimum recommended data block size for ArcSDE applications is 8 KB; however, 16 KB have been found to deliver a higher overall level of performance for databases storing mostly linear or area features.

If you are not going to let the Oracle installer create the database, you can set the data block size with the DB_BLOCK_SIZE initialization parameter. Set the DB_BLOCK_SIZE to at least
8 KB:

```
DB_BLOCK_SIZE = 8192
```

For more information on the init.ora file, refer to *Setting the Oracle initialization parameters* in this chapter.

Creating the business and index tablespaces

Once you have completed the installation of Oracle, create the tablespaces that store your tables and indexes. You may want to write a SQL script and execute it as the SYSTEM user within SQL* Plus. Alternatively, you could use the graphical user interface of the Oracle Enterprise Manager's Storage Manager to perform this task.

Creating the Oracle SDE user space

During the installation of ArcSDE, you will create the Oracle SDE user and the SDE user's default tablespace to store the geodatabase system tables.

Updating the storage parameters of the DATA_DICTIONARY keyword

Update the storage parameters of the DATA_DICTIONARY configuration keyword in the dbtune.sde file located in the SDEHOME/etc directory on UNIX® systems. On Windows NT® systems, the install shield will prompt you for the location of this file (see the *ArcSDE for Oracle Installation Guide* for more information). Set the extent sizes and the names of the tablespaces the geodatabase system tables will be stored in. The geodatabase system tables are created by the ArcSDE sdesetupora* administration command.

Installing the optional sdesys_util package

The optional sdesys_util package, found under the SDEHOME/tools/oracle directory can be installed in the Oracle sys users schema. This package contains stored procedures that allow you to create and manage the sde user.

To install the sdesys_util package, log in as the Oracle sys user and execute the package specification followed by its body:

```
connect sys/<password>

@sdesys_util.sps

Package created.

@sdesys_util.spb
```

```
Package body created.
```

You can allow a user other than sys to execute the stored procedures of the sdesys_util package by executing the sdesys_util.grant_admin_privs that grants the necessary privileges. The usage of the grant_admin_privs stored procedure is as follows:

```
grant_admin_privs (
    administrator IN varchar2(30) DEFAULT 'system'
)
```

If you do not specify any arguments, the privileges are granted to the Oracle system user:

```
connect sys/<password>
exec sdesys_util.grant_admin_privs('JOE');
```

If a user, other than sys, executes an sdesys_util stored procedure, they must qualify it with the sys users schema. For example, the system user would execute the grant_pipes_locks stored procedure as follows:

```
connect system/<password>
exec sys.sdesys_util.grant_pipes_locks;
```

Granting execute privileges on DBMS_PIPE and DBMS_LOCKS to PUBLIC

ArcSDE uses the stored procedures of the DBMS_PIPE and DBMS_LOCK Oracle built-in packages. ArcSDE calls stored procedures of the DBMS_PIPE package when it stores and transmits ArcSDE rowids. ArcSDE calls the stored procedures of the DBMS_LOCK package to add a row to the PROCESS_INFORMATION table whenever an ArcSDE session connects. Your Oracle DBA must connect to the Oracle instance as the SYS user and grant execute on these packages to PUBLIC:

```
connect sys/<password>
grant execute on dbms_pipe to public;
grant execute on dbms_lock to public;
```

Alternatively, if you have installed the sdesys_util package you can connect as the Oracle sys user and execute its sdesys_util.grant_pipes_locks stored procedure:

```
connect sys/<password>
exec sdesys_util.grant_pipes_locks;
```

Creating the Oracle sde user

Before you run the sdesetupora* command to create the geodatabase system tables on a UNIX system the sde Oracle user must be created. The Windows NT installation of ArcSDE has automated this procedure. However, you can, if you wish, create the sde Oracle user prior to installing ArcSDE.

Like any Oracle user the sde user requires a default tablespace. Review the previous discussion of the ArcSDE system tablespaces to determine how many tablespaces you will need to store your ArcSDE system data.

Use the following SQL commands to create the sde user. Substitute your own entries for the sde user's password, default tablespace and temporary tablespace:

```
connect system/<password>

create user sde
  identified by <password>
  default tablespace <sde user's default tablespace>
  temporary tablespace <temporary tablespace>;
```

Alternatively, you can use the sys user's SDESYS.UTIL_CREATE_SDE stored procedure to create the SDE Oracle user. The usage for CREATE_SDE_USER is:

```
SDESYS_UTIL.CREATE_SDE_USER(
    sdedatafile IN VARCHAR2(256) DEFAULT NULL,
    sdetabspace IN VARCHAR2(30) DEFAULT 'SDE',
    temptabspace IN VARCHAR2(30) DEFAULT 'TEMP',
    sdepasswd IN VARCHAR2(30) DEFAULT 'SDE');
```

Executing this stored procedure without any arguments will create the SDE Oracle user with a default tablespace set to SDE, a temporary tablespace set to TEMP, and the password set to SDE:

```
connect sys/<password>
exec sdesys_util.create_sde_user;
```

CREATE_SDE_USER creates the SDE user's default tablespace if you specify the location of the tablespace's data file.

If the default tablespace already exists, CREATE_SDE_USER will check it to make sure that it is big enough. If it is less than 100 MB, CREATE_SDE_USER issues a warning message telling you to either increase the size of the default tablespace or decrease the initial extents of the state, state_lineages, and mvtables DATA_DICTIONARY configuration keyword storage parameters. If you have configured these storage parameters to store these tables and their indexes in other tablespaces, then you may disregard this warning. The sdesetupora* command will fail if it cannot find enough space to store the geodatabase system tables and indexes.

Granting install privileges to the sde Oracle user

If you are creating a fresh install of ArcSDE, grant the following list of privileges to the SDE user. These privileges must be granted to the SDE user:

```
SELECT ANY TABLE
CREATE SESSION
CREATE TABLE
CREATE PROCEDURE
CREATE SEQUENCE
CREATE TRIGGER
UNLIMITED TABLESPACE
```

Alternatively, you can call the GRANT_SDE_INSTALL_PRIVS stored procedure (found in the SDESYS_UTIL package) to grant these privileges to the SDE Oracle user:

```
connect sys/password
exec sdesys_util.grant_sde_install_privs;
```

Note: ArcSDE requires the Oracle SDE user be granted SELECT ANY TABLE privileges. Granting SELECT ANY TABLE privileges to the SDE user allows it to auto-register Oracle Spatial tables. Auto registration occurs when an ArcSDE application performs a layer list and detects an Oracle Spatial table that is listed in the Oracle Spatial metadata table that is not listed in the SDE.LAYERS table. The SDE user must be able to confirm the presence of the Oracle Spatial table before auto-registering it. Since there is no guarantee that the owner of the table has granted select permissions on the table to the SDE user, the SDE user must be granted SELECT ANY TABLES privileges.

Granting upgrade privileges to the SDE Oracle user

To upgrade a previous version of an ArcSDE database, grant the following list of privileges to complete the upgrade process. Following the successful completion of the upgrade process, you can revoke these privileges and grant the previous list of privileges to the SDE Oracle user.

```
CREATE SESSION
SELECT ANY TABLE
```

```
ALTER ANY TABLE
CREATE ANY INDEX
ALTER ANY INDEX
DROP ANY INDEX
CREATE ANY SEQUENCE
CREATE ANY PROCEDURE
EXECUTE ANY PROCEDURE
DROP ANY PROCEDURE
SELECT ANY SEQUENCE
CREATE ANY VIEW
DROP ANY VIEW
CREATE ANY TRIGGER
DROP ANY SEQUENCE
ANALYZE ANY
UNLIMITED TABLESPACE
```

You can grant this list of privileges by executing the SDESYS_UTIL.GRANT_SDE_UPGRADE_PRIVS stored procedure:

```
connect sys/<password>
exec sdesys_util.grant_sde_upgrade_privs;
```

The upgrade privileges can be revoked and the install privileges granted by executing the SDESYS_UTIL.REVOKE_SDE_UPGRADE_PRIVS stored procedure:

```
connect sys/<password>
exec sdesys_util.revoke_sde_upgrade_privs;
```

Note: Previous versions of ArcSDE maintained the locks in memory. Therefore, to upgrade the ArcSDE database from a previous version to ArcSDE, the SDE user must be granted privileges to create the sequences and triggers in the schemas of users who own tables listed in the TABLE_REGISTRY, LAYERS, or RASTER_COLUMNS tables. Following the completion of a successful upgrade, the privileges can be revoked, and lesser privileges assigned for the install operation can be granted.

Creating Oracle users

The creation of an ArcSDE Oracle user differs from that of other Oracle users that you normally create. Each ArcSDE user must own two tables for storing SDE log file data—SDE_LOGFILES and SDE_LOGFILE_DATA. The SDE log file tables and their associated indexes, sequences, and triggers can be created when the ArcSDE Oracle user is created. If they are not, ArcSDE detects their absence and attempts to create them. In this case, the ArcSDE Oracle user must be able to create the tables when they connect for the first time. ArcSDE returns an SE_NO_ACCESS (-15) error message if the user does not have the required privileges to create the tables, indexes, sequences and triggers associated with the SDE log files.

The SDE_LOGFILES and SDE_LOGFILE_DATA tables and associated indexes are created according to the SDE log file storage parameters specified in the DBTUNE table. For more information about these storage parameters, see Chapter 3, ‘Configuring DBTUNE storage parameters’.

The following list of privileges is required for all ArcSDE users connecting for the first time:

```
CREATE SESSION
CREATE TABLE
CREATE PROCEDURE
CREATE SEQUENCE
CREATE TRIGGER
UNLIMITED TABLESPACE
```

Once the user has successfully connected for the first time, privileges can be revoked since the SDE_LOGFILES and SDE_LOGFILE_DATA tables now exist. The table below lists the type of users DBAs typically create and the privileges assigned to them.

Title	Description	Privileges
Viewer	The viewer is allowed to connect to an ArcSDE database. Other users grant select privileges on their tables and feature classes to the viewer or to the public role. The DBA can create a role that can be granted select privileges on data objects owned by other users. The role can be granted to the viewer.	CREATE SESSION SELECT on other user's data objects
Editor	The editor is allowed to connect to an ArcSDE database. Other users grant select and insert, update, or delete on data objects they own to the editor. The DBA may create a role that can be granted select, insert, update, and delete privileges on data objects owned by other users. The role can be granted to the editor.	CREATE SESSION SELECT, INSERT, UPDATE, or DELETE on other user's data objects
Owner	The owner is allowed to connect to an ArcSDE database and create data objects. The owner may grant privileges on their objects to other users or roles. Other users can grant select and insert, update, or delete on data objects they own to creator. The DBA may create a role that can be granted select, insert, update, and delete privileges on data objects owned by other users. The role can be granted to the owner.	CREATE SESSION CREATE TABLE CREATE PROCEDURE CREATE SEQUENCE CREATE TRIGGER UNLIMITED TABLESPACE SELECT, INSERT, UPDATE, or DELETE on other user's objects

Manually creating ArcSDE Oracle users

When you create an Oracle user, assign your users a default tablespace and a temporary tablespace.

Note: If you do not specifically assign a temporary or default tablespace to a user, Oracle uses the SYSTEM tablespace by default. Since the SYSTEM tablespace holds the Oracle system tables, a vital part of the database, it is important not to use it for anything else. Since the data sets managed by ArcSDE may be large, using the SYSTEM tablespace to store this data could completely fill the SYSTEM tablespace and cause the Oracle database to crash.

This is the Oracle user creation syntax:

```
create user <username>
  identified by <password>
  default tablespace <default tablespace>
  temporary tablespace <temporary tablespace>
  quota unlimited on <default tablespace>
  quota unlimited on <temporary tablespace>;
```

These are the basic privileges that must be granted to all ArcSDE Oracle users until they connect for the first time:

```
CREATE SESSION
CREATE TABLE
CREATE PROCEDURE
CREATE SEQUENCE
CREATE TRIGGER
UNLIMITED TABLESPACE
```

Connect as the user to create the SDE_LOGFILES and SDE_LOGFILE_DATA tables. To create a user of type VIEWER or EDITOR, revoke the following list of privileges after the SDE log file tables have been created:

```
CREATE TABLE
CREATE PROCEDURE
CREATE SEQUENCE
CREATE TRIGGER
UNLIMITED TABLESPACE
```

For the VIEWER, grant SELECT privileges on the specific data objects you wish the user to have permission to display or query. If you have a large community of viewers, create a role that can be granted SELECT access on the data objects.

To grant or revoke SELECT access to complex data objects, such as feature datasets or standalone feature classes, you should use ArcCatalog. Use the grant or revoke operation of the sdelayer administration command to grant or revoke SELECT access if you have not installed ArcGIS Desktop. The access can be granted directly to a user or to a role.

For the editor, grant SELECT privileges and editing (insert, update, or delete) privileges on the specific data objects you wish the users to have permissions to query and edit. If you have many editors, save time by creating a role and granting the select and edit privileges to it. Then grant the role to each editor.

In this example, the DBA creates the user Sam with the password TREETOP. The default tablespace in which Sam will create his tables and indexes that are not assigned a tablespace is GIS1.

Note: The ArcSDE Administrator creates DBTUNE configuration keywords that hold the tablespace and other storage parameters Oracle assigns to tables and indexes when it creates them. For more information about DBTUNE configuration keywords and how to create and use them, see Chapter 3, ‘Configuring DBTUNE storage parameters’.

The temporary tablespace used when Sam creates indexes and performs sorts is TEMP1:

```
create user sam
  identified by treetop
  default tablespace gis1
  temporary tablespace temp1
  quota unlimited on gis1
  quota unlimited on temp1;
```

To create Sam’s log file tables (SDE_LOGFILES and SDE_LOGFILE_DATA) and their associated sequence generator and trigger, the DBA grants the user Sam the following privileges:

```
CREATE SESSION
CREATE TABLE
CREATE PROCEDURE
CREATE SEQUENCE
CREATE TRIGGER
UNLIMITED TABLESPACE
```

The DBA connects to the ArcSDE database as SAM to create the log file tables in SAM's schema. The sdelayer administration command describe operation provides a simple way to do this:

```
$ sdelayer -o describe -u sam -p tree_top
```

Using the REVOKE <privilege> FROM <user> command to revoke all but the CREATE SESSION privileges since Sam is an editor and is not allowed to create a schema:

```
CREATE TABLE FROM SAM;
REVOKE CREATE PROCEDURE FROM SAM;
REVOKE CREATE SEQUENCE FROM SAM;
REVOKE CREATE TRIGGER FROM SAM;
REVOKE UNLIMITED TABLESPACE FROM SAM;
```

Create SDE_VIEWER and SDE_EDITOR roles. The DBA has determined that all users who own schema should grant select privileges on their data objects to the SDE_VIEWER role and that they should grant INSERT, UPDATE, and DELETE to the SDE_EDITOR role:

```
CREATE ROLE SDE_VIEWER;
CREATE ROLE SDE_EDITOR;
```

The DBA grants both the SDE_VIEWER and SDE_EDITOR roles to Sam:

```
GRANT SDE_VIEWER TO SAM;
GRANT SDE_EDITOR TO SAM;
```

Betty, a creator, uses the grant operation of the sdelayer command to grant select access on her roads feature class to the SDE_VIEWERS role. All viewers granted the SDE_VIEWERS role will have select access to Betty's roads feature class:

```
$ sdelayer -o grant -l roads,feature -A select -U sde_viewers -u betty -p cdn
```

Betty also grants INSERT, UPDATE, and DELETE access to the SDE_EDITORS role. Now editors granted the SDE_EDITORS role, such as SAM, will be able to make changes to Betty's roads feature class:

```
$ sdelayer -o grant -l roads,feature -A insert -U sde_editors -u betty -p cdn
$ sdelayer -o grant -l roads,feature -A update -U sde_editors -u betty -p cdn
$ sdelayer -o grant -l roads,feature -A delete -U sde_editors -u betty -p cdn
```

Installing the USER_UTIL package to create and maintain ArcSDE users

As an alternative to executing the SQL statements to create and maintain ArcSDE users, you can install the user_util package of stored procedures located in the SDEHOME/tools/oracle directory. To install the package connect as an Oracle DBA (usually the Oracle system user) and run the package specification script followed by the body script:

```
connect system/<password>
@user_util.sps

Package created

@user_util.spb

Package created
```

Using the stored procedures of the USER_UTIL package

The CREATE_USER_AND_LOGFILES stored procedure creates an Oracle user with SDE log files and grants privileges according to its user type. The usage for this stored procedure is as follows:

```
USER_UTIL.CREATE_USER_AND_LOGFILES (
    username IN VARCHAR2(30),
    password IN VARCHAR2(30),
    default_tabsp IN VARCHAR2(30) DEFAULT 'USERS',
    temp_tabsp IN VARCHAR2(30) DEFAULT 'TEMP',
    usertype IN VARCHAR2(8) DEFAULT 'OPERATOR'
)
```

The username and password must be entered. The user's default tablespace defaults to USERS tablespace, while the temporary tablespace defaults to the TEMP tablespace. The usertype argument defines the type of user that will be created and it defaults to OPERATOR.

A user created as an OPERATOR is granted CREATE SESSION privilege, is not allowed to create a schema, and does not have access to any data objects owned by other users.

A user created as an OWNER is granted privileges to create tables, and granted CREATE SCHEMA privileges. OWNER users automatically grant SELECT, INSERT, UPDATE or DELETE privileges on their own tables to other OWNER users and to OPERATOR users.

Updating the DBTUNE file

After you have created the tablespaces for your tables and indexes, update the DBTUNE file. DBTUNE files are always located under the SDEHOME/etc directory. The dbtune.sde file is the default DBTUNE file that the ArcSDE setup program uses to populate the DBTUNE table.

Create the keywords in the dbtune.sde file that will contain the Oracle configuration parameters of the feature classes and tables that you intend to create with the ArcGIS programs ArcCatalog and ArcToolbox or the many ArcSDE administration tools.

For a detailed discussion on the maintenance of the DBTUNE table, refer to Chapter 3, *Configuring DBTUNE storage parameters*. This chapter describes DBTUNE parameters that can be applied to each of the spatial storage methods supported by ArcSDE for Oracle. The supported spatial storage formats are described in Appendixes C, D, and E.

Creating tables, feature classes, and raster column

ArcInfo and ArcSDE offer several ways to create and maintain the tables, indexes, and feature classes of an ArcSDE database. Chapter 4, 'Managing tables, feature classes, and raster columns', describes in detail the possible methods for creating tables and feature classes using either ArcInfo or ArcSDE tools.

Setting the Oracle initialization parameters

Whenever you start an Oracle instance, Oracle reads its initialization parameters from the init.ora file. These parameters define the characteristics of the instance. This section describes some of the parameters that control allocation of shared memory. For a detailed discussion of the Oracle initialization parameters, refer to *Oracle Server Tuning* for your Oracle release.

The init.ora file is located under the $ORACLE_BASE/admin/<ORACLE_SID>/pfile directory or folder. Init.ora is a common name given to the initialization file of an Oracle database instance but for any given instance, the file is actually called init<oracle SID>.ora.

For example, if the Oracle SID is GIS, the init.ora file for this instance would be called initGIS.ora.

Managing Oracle's memory

Care must be taken when setting the initialization parameters that affect memory. Setting these parameters beyond the limits imposed by the physical memory resource of the host machine significantly degrades performance. This section provides a few general rules regarding configuration of System Global Area (SGA) as well as memory structures affecting the size of an Oracle user's private area, the PGA. The SGA is a block of shared memory that Oracle allocates and shares with all sessions. For more information about the SGA, refer to the *Oracle Concepts Guide* for your Oracle release.

SGA must not swap

You should not create an SGA that is larger than two-thirds the size of your server's physical RAM. Your virtual memory must be able to accommodate both the SGA and the requirements of all active processes on the server.

Avoid excessive paging

Using your operating system tools (vmstat on UNIX systems and the task manager on Windows NT), check for excessive paging. A high degree of paging can be the result of an SGA that is too large.

Configure enough virtual memory

As a rule, Oracle recommends that your swap space be at least two to four times the size of your physical RAM. The required size of the swap file UNIX or the page file on Windows NT depends on the number of active ArcSDE sessions. For every ArcSDE session, a gsrvr process and a corresponding Oracle process is started. To determine the memory usage of these processes, use the ps -elf command on UNIX systems and the Processes tab of the Windows NT Task Manager. You must deduct the size of the Oracle SGA from the Oracle user processes. The total size of the ArcSDE gsrvr processes, the ArcSDE giomgr processes, Oracle user processes, Oracle background processes, operating system processes, and any other process running on the server must be able to fit into virtual memory.

For ArcSDE client applications that connected directly to an Oracle instance, the gsrvr process does not exist. Also, if the ArcSDE service is not used because all client applications connect directly to the Oracle instance, the giomgr process will not be started either. For this reason direct connections have a smaller memory imprint on the server since the gsrvr process is absent.

Redo log buffer

The redo log buffer is a component of the Oracle SGA that holds uncommitted changes to the database. The log buffer is flushed to the current online redo log file whenever a user issues a commit and the buffer becomes one-third full or every three seconds. The size of the redo log buffer is controlled by the LOG_BUFFER parameter. Because of the rapid rate at which the log buffer is flushed, it does not need to be that large.

Oracle recommends that you set this parameter to 500 KB or 128 KB multiplied by the number of CPUs. If you have less than 4 CPUs, set this parameter to 512,000 (500 KB).

```
log_buffer = 512000
```

Otherwise, set LOG_BUFFER to 128 KB * the number of CPUs.

Setting the LOG_BUFFER to large values in the hope of processing huge loading transactions may in fact result in a performance reduction. Latch contention between transactions may occur if the log buffer is set too large.

To determine if the redo log buffer is large enough while the system is active examine the Oracle dynamic table V$SYSSTAT. Compare the values of the *redo log space requests* and the *redo entries*. Oracle recommends that you increase the size of the redo log buffers if the ratio of these values is greater than 1:5,000.

```
select name, value
from v$sysstat
where name in ( 'redo entries' , 'redo log space requests' );
```

Shared pool

The shared pool is another component of the Oracle SGA that holds both the data dictionary cache and the library cache. The data dictionary cache holds information about data objects, free space, and privileges. The library cache holds the most recently parsed SQL statements. Generally, if the shared pool is large enough to satisfy the resource requirements of the library cache, it is already large enough to hold the data dictionary cache. The size of the shared pool is controlled by the SHARED_POOL_SIZE parameter.

Set this parameter to at least 55 MB:

```
shared_pool_size = 55,000,000
```

Highly active geodatabases supporting volatile utility or parcel editing systems may require the SHARED_POOL_SIZE to be set as high as 200 MB.

Of the three SGA buffers, the shared pool is the most important. If the SGA is already as large as it can be, given the size of your physical memory, reduce the size of the buffer cache to accommodate a larger shared pool.

Buffer cache

The buffer cache is another component of the Oracle SGA that stores the most recently used data blocks. Data blocks are the Oracle atomic unit of data transfer. Oracle reads and writes data blocks to and from the database whenever the user edits or queries it. The size of the buffer cache is controlled by the DB_BLOCK_BUFFERS parameter.

For optimum performance, increase the size of the buffer cache without causing the operating system to page excessively and/or swap the SGA. Oracle recommends that the SGA not be larger than two-thirds of the physical RAM.

To estimate the size of the buffer cache, first determine how much physical RAM your server has. Multiply this number by 0.66 to determine the target size of the SGA. Deduct the SHARED_POOL_SIZE and LOG_BUFFER to return the amount of memory available to the buffer cache. Reduce this number by 10 percent to account for Oracle's internal memory usage. Finally, divide by the database block size to determine the DB_BLOCK_BUFFERS

setting. (The recommended minimum data block size for ArcSDE is at least 8 KB; however, we have found overall performance does improve with larger block sizes.)

*memory available to SGA = physical RAM * 2/3*

memory available to buffer cache

*= (memory available to SGA - (shared_pool_size + log_buffer)) * 0.9*

db_block_buffers

= memory available to buffer cache / db_block_size

Oracle recommends that the buffer cache hit ratio be at least 95 percent. In other words, for all data block requests made to the Oracle server, the data blocks were found in the buffer cache at least 95 percent of the time. Data blocks not found in the buffer cache had to be retrieved from disk less than 5 percent of the time. Examine the performance of the buffer cache after the system has been in use for some time. Compare the values of the *db block gets*, *consistent gets*, and *physical reads* in the Oracle dynamic view V$SYSSTAT. If the *physical reads* are greater than 5 percent of the sum of the *db block gets* and the *consistent gets*, you should increase the size of the buffer cache:

```
select name, value
from v$sysstat
where name in ( 'db block gets', 'consistent gets', 'physical reads');
```

Buffer cache hit ratio = 1 - (physical reads / (db block gets + consistent gets))

You should not increase the buffer cache at the expense of the shared pool.

Sort area

The sort area is a component of each session's PGA and not the SGA.

As the name implies, Oracle uses sort area to perform sorting. Larger sort areas reduce the time required to build an index. As Oracle constructs an index, it writes the index blocks to the sort area. When the sort area becomes full, its contents are transferred to the temporary tablespace. The larger the sort area, the less frequently Oracle must write intermediate results to temporary disk space.

You must be careful not to make the sort area too large because it is allocated per session to the PGA and not per instance as is the SGA. Additional users connecting to the database causes more space to be allocated to the Oracle instance because of the creation of additional PGA's

Also, Oracle may allocate sort area for each table referenced within a join. Therefore, it is theoretically possible for Oracle to allocate the maximum sort area for each table referenced in a join. Generally, the logic of the query prevents this from happening. However, you should be aware of the dynamic nature of sort area.

Sort area is controlled by the SORT_AREA_SIZE parameter.

For the construction of the indexes during the creation of the database, set the SORT_AREA_SIZE parameter to a large value since few sessions are active, and the indexes are created faster if more sort area is available. For 2 GB of RAM, setting the sort area to 200

MB is not unreasonable if you are using one session to create indexes on tables that have millions of rows of data.

Once the database is constructed you should reduce the size of the sort area to account for the increased number of sessions. ArcInfo™ applications accessing a multiversioned geodatabase generally use little sort area. A sort area of 1 MB is sufficient to prevent I/O to temporary tablespace for these types of applications.

Set SORT_AREA_SIZE to 1 MB:

```
sort_area_size = 1048576
```

You can determine if SORT_AREA_SIZE is large enough by comparing the value of *sorts (memory)* and *sorts (disk)* of the Oracle dynamic view V$SYSSTAT. If the sorts performed on *disk* are greater than 10 percent of those performed in *memory,* consider increasing the sort area.

```
select name, value
from v$sysstat
where name in ('sorts (memory)', 'sorts (disk)');
```

Prepage the SGA

Set the PRE_PAGE_SGA to true for servers that are running a single Oracle instance. Prepaging the SGA increases the amount of time required to start the Oracle server as well as the amount of time required for users to connect. However, it does reduce the number of page faults—which occur whenever Oracle must allocate another page to the SGA—while the server is active:

```
pre_page_sga = true
```

Enabling the optional Oracle startup trigger

ArcSDE includes an optional startup trigger that is run whenever the Oracle instance is started. The startup trigger cleans up any orphaned session information that remains in the ArcSDE system tables following an instance failure. The startup trigger is optional because ArcSDE invariably cleans up the orphaned metadata during its normal operation. The startup trigger merely offers a guarantee that orphaned metadata will not be present following Oracle instance startup.

To create the startup trigger run the SQL arcsde_database_startup.sql script as the Oracle SYS user. The script is located at $SDEHOME/etc/tools/oracle on UNIX systems and %SDEHOME%\etc\tools\oracle on Windows NT.

Monitoring and modifying the ArcSDE installation

Monitoring the log files

For all three types of databases, connect as the SYSTEM user and issue the following query to determine if your online redo log files are large enough and if the checkpoint frequency is occurring at a desirable interval:

```
SELECT TO_CHAR(FIRST_TIME,'dd-mon-yy hh24:mi:ss') FROM V$LOGHIST;
```

This is an example of the output:

```
TO_CHAR(FIRST_TIME)
```

```
------------------
04-nov-99 13:15:14
04-nov-99 13:21:04
04-nov-99 13:27:04
04-nov-99 13:32:36
```

The example output shows the log switches are occurring at intervals greater than five minutes, the interval at which Oracle issues the checkpoints. If the interval was less than five minutes, the DBA should consider increasing the size of the online redo log files.

Modifying the online redo log files

To change the size of the log files, you must create new log file groups of the correct size, make one of the new groups active, and drop the old log file groups. Remember, Oracle requires that you always have at least three log file groups active.

Note: ESRI recommends that you always mirror the online redo log file groups across physically separate disk drives.

Follow this procedure using the SQL statements listed below.

1. Add the log file groups with their new size:

```
ALTER DATABASE ADD LOGFILE
('<path to log file member1>','<path to log file member2>', …)
SIZE <size>;
```

2. Determine which of the existing log file groups is current:

```
SELECT GROUP#,STATUS FROM V$LOG;

GROUP#    STATUS
--------- ----------------
        1 INACTIVE
        2 CURRENT
        3 INACTIVE
```

3. Issue the correct number of manual log switches required to make a new log file group current:

```
ALTER SYSTEM SWITCH LOGFILE;
```

4. Remove the old log file groups, identifying them by their group numbers. You can get a log file's group number by querying the V$LOG table (see step 2):

```
ALTER DATABASE DROP LOGFILE GROUP <group number>;
```

Repositioning data files

After the database has been in use for a while, you can examine the reads and writes to the data files with the following query:

```
select vd.name, vs.phyrds, vs.phywrts
from v$datafile vd, v$filestat vs
where vs.file# = vd.file#;
```

If you find that some disk drives are receiving a higher percentage of the I/O than others, you can balance the I/O by repositioning the data files. The data files are repositioned by shutting down the Oracle instance, performing a full backup, and moving the files using operating system commands to copy them from one disk to another. Using SQL*Plus, the instance is started, and the database is mounted but not opened:

```
startup mount
```

Before the database can be opened, you must update the control files with the new locations of the data files using the *alter database* statement:

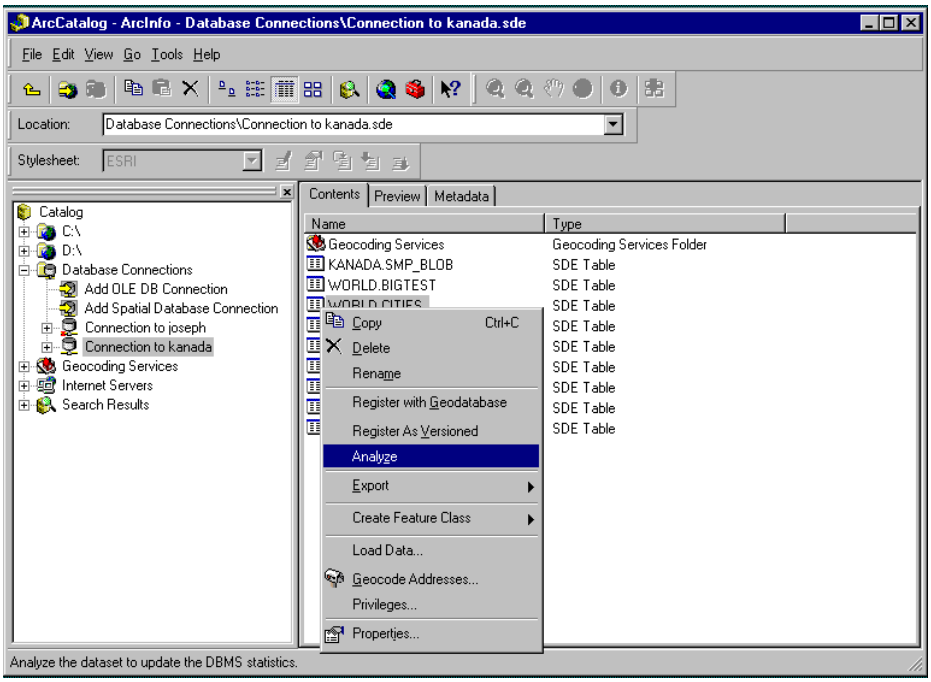
```
alter database rename datafile 'old name' to 'new name';
```

Once all of the new locations have been entered, you may open the database.

```
alter database open;
```

Updating ArcSDE compressed binary statistics

For optimal performance of feature classes created with the ArcSDE compressed binary storage format, keep the statistics up-to-date.



In ArcCatalog, to update the statistics of all of the tables and indexes within a feature dataset, right-click on the feature dataset and click on Analyze. To update the tables and indexes within a feature class, right-click on the feature and click on Analyze.

From the command line, use the UPDATE_DBMS_STATS operation of the sdetable administration command to update the statistics for all the tables and indexes of a feature class. It is better to use the UPDATE_DBMS_STATS operation rather than individually analyzing the tables with the Oracle SQL ANALYZE statement because it updates the statistics for all the tables of a feature class that require statistics. To have the UPDATE_DBMS_STATS operation update the statistics for all the required tables, do not specify the -K (schema object) option.

```
sdetable -o update_dbms_stats -t roads -m compute -u av -p mo
```

When the feature class is registered as multiversioned, the "adds" and "deletes" tables are created to hold the business table's added and deleted records. The version registration process automatically updates the statistics for all the required tables at the time it is registered.

Periodically update the statistics of dynamic tables and indexes to ensure that the Oracle cost-based optimizer continues to choose an optimum execution plan. To save time, you can analyze all of the data objects within a feature dataset in ArcCatalog.

If you do not have enough temporary tablespace to compute statistics on larger tables, use the -m option to estimate statistics. The tables will be estimated at a sample rate of 33 percent.

```
sdetable -o update_dbms_stats -t roads -m estimate -u av -p mo
```

You should consider increasing the size of your temporary tablespace to compute statistics rather than estimate them as it provides more accurate statistics for the Oracle cost-based optimizer.

The statistics of a table's indexes are automatically computed when the table is analyzed, so there is no need to analyze the indexes separately. However, if you need to do so you can use the UPDATE_DBMS_STATS -n option with the index name.

The example below illustrates how the statistics for the f2_ix1 index of the roads feature table can be updated:

```
sdetable -o update_dbms_stats -t roads -K f -n F2_IX1 -u av -p mo
```

For more information on analyzing geodatabase objects from ArcCatalog, refer to *Building a Geodatabase*.

For more information on the sdetable administration command and the UPDATE_DBMS_STATS operation, refer to the *ArcSDE Developer Help*.

Updating Oracle Spatial geometry type statistics

Update the statistics of the business table containing an Oracle Spatial column with Analyze from ArcCatalog or with the sdetable UPDATE_DBMS_STATS operation.

```
sdetable -o update_dbms_stats -t roads -m compute -u av -p mo
```

You may also update the statistics using the Oracle ANALYZE statement. Use the compute statistics option if there is enough temporary tablespace available to permit the operation. Otherwise, use the estimate statistics option with as high a sample rate as possible.

```
analyze table roads compute statistics;
```

When ArcSDE creates data using Oracle Spatial Geometry, it can optionally create the layer's Oracle Spatial index. Oracle Spatial suggests that you analyze this spatial index. If ArcSDE is asked to create the Oracle Spatial index for you, it automatically analyzes this index. If you decide to create the Oracle Spatial index on your own, you will have to use the Oracle ANALYZE statement to analyze your index after creating it.

For more information on the Oracle ANALYZE statement, refer to the *Oracle SQL Reference Manual* for your release of Oracle.

CHAPTER 3

Configuring DBTUNE storage parameters

DBTUNE storage parameter strings allow you to control how ArcSDE clients create objects within an Oracle database. They allow you to determine such things as how to allocate space to a table or index, which tablespace a table or index is created in, as well as other Oracle specific storage attributes. They also allow you to specify one of the available storage formats for the geometry of a spatial column.

This chapter discusses the mechanism by which ArcSDE manages storage parameters that you provide and applies them to specific statements submitted to Oracle for the creation of ArcSDE tables, indexes and other objects.

For detailed information how Oracle uses storage parameters to control space allocation, refer to the *Oracle Server Administrator's Guide* for your Oracle release.

The DBTUNE table and file

The DBTUNE storage parameters are maintained online 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 is given specific default values upon initial setup, but these values may be changed through use of a dbtune file. Several example versions of the dbtune file are provided with the installation media for ArcSDE.

DBTUNE parameters

Parameters define the storage configuration of simple objects, such as tables and indexes, as well as complex objects such as feature classes, network classes, and raster columns. Many different parameters may be grouped together under a single *configuration keyword*.

ArcSDE client applications and some ArcSDE administration tools reference one or more configuration keywords when creating object.

When a configuration keyword is specified by an ArcSDE application or administration tool, the parameters within the associated *parameter group* are searched and the necessary configuration strings are incorporated into the CREATE TABLE or CREATE INDEX statement submitted to the Oracle database server.

The structure of the DBTUNE file

Storage parameters in a dbtune file occur as a combination of *parameter name* and *configuration string* delimited by white space. A configuration string value may span multiple lines and must be enclosed in double quotes. For example, a valid specification for the parameter named A_INDEX_ROWID might look like this:

```
A_INDEX_ROWID   "PCTFREE 10 INITRANS 4 STORAGE (FREELISTS 4 MINEXTENTS
                1 PCTINCREASE 0)"
```

Storage parameters are grouped by *keyword*. Each parameter group is introduced by its keyword which is prefixed by two pound signs, "##". The word 'END' terminates each parameter group. Double pound signs, "##", signal the presence of a keyword but are not part of the keyword itself.

For example, a group of parameters under the configuration keyword "WILSON_DATA" may look like this:

```
##WILSON_DATA

ATTRIBUTE_BINARY        "LONGRAW"
A_INDEX_ROWID   "PCTFREE 10 INITRANS 4 STORAGE (FREELISTS 4 MINEXTENTS 1
                 PCTINCREASE 0)"
A_INDEX_SHAPE   "PCTFREE 10 INITRANS 4 STORAGE (FREELISTS 4 MINEXTENTS 1
                 PCTINCREASE 0)"
A_INDEX_STATEID "PCTFREE 10 INITRANS 4 STORAGE (FREELISTS 4 MINEXTENTS 1
                 PCTINCREASE 0)"
B_STORAGE       "PCTFREE 10 PCTUSED 90 INITRANS 1 STORAGE (FREELISTS 4
                 MINEXTENTS 1 PCTINCREASE 0)"
END
```

In special circumstances ArcSDE references a *compound keyword* when creating database objects. The compound keyword allows ArcSDE to create related database objects having very different object creation parameters to accommodate difference performance needs. A compound keyword consists of a configuration keyword plus a suffix delimited by a double colon, "`::`". For example:

```
##ELECTRIC::DESC

parameter name   "configuration string"
etc…

END
```

Comments within the dbtune file are indicated by a single pound sign, "#". Default versions of the dbtune file provided in the general software release contain lines that are commented out. Such lines are used as placeholders for certain storage parameters, such as tablespace name, and may be restored by removing the comment character and editing the line.

Any number of parameter groups may be specified in a dbtune file. However, certain groups and certain parameter names within groups are expected to exist and will be created in the DBTUNE table if they do not exist in the dbtune file.

The structure of the DBTUNE table

Parameters from a dbtune file are loaded into the DBTUNE table using the **sdedbtune** utility. The DBTUNE table has the following definition:

```
Name             Null?     Datatype
keyword          not null  varchar2(32)
parameter_name   not null  varchar2(32)
config_string    null      varchar2(2048)
```

The keyword field stores the configuration keyword for the group in which each parameter is found. For a single keyword, there may be many different parameter_name values, each one associated with a config_string value.

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 in the %SDEHOME%/etc directory under Windows or the $SDEHOME/etc under Unix. If the DBTUNE table already exists, the ArcSDE setup phase will not alter its contents.

Managing the DBTUNE table

Through the use of the **sdedbtune** utility you can initialize or alter the contents of the DBTUNE table. This utility guarantees that the DBTUNE table maintains a certain default set of keywords, parameters and parameter values.

In addition to the default keywords and parameters, you may add to the DBTUNE file keywords and configuration values of your choosing.

Note: ESRI does *not* recommend using SQL to directly alter the contents of the DBTUNE table. Doing so would bypass certain protections written into the **sdedbtune** utility, possibly leading to reduced performance.

Initializing the DBTUNE table

The dbtune.sde file that is provided with the install media contains default values, which are used to initialize the DBTUNE table.

On Unix systems, you can modify the dbtune.sde file prior to running the **sdesetupora** command. On Windows NT systems the setup phase is part of the install so you will have to edit the file and use the **sdedbtune** import operation to customize the DBTUNE table.

If the dbtune.sde file is missing when the **sdesetupora** command is executed, or if specific parameters are missing because dbtune.sde file has been altered, the ArcSDE software will enter *software default* values into the dbtune table.

Customizing the DBTUNE file

Prior to creating ArcSDE objects, you should customize the dbtune.sde file by specifying the tablespace names for storage parameters. In the default dbtune.sde file, the tablespace entries in the dbtune.sde file have been commented out with the "#" character.

To customize the dbtune.sde file, remove the comment character preceding each tablespace specification and enter the names of the tablespaces where you wish to store your ArcSDE tables and indexes. Be careful not to remove the double quotation marks that surround the configuration strings.

Follow the procedure provided in the following example for updating the Oracle tablespace parameter in the dbtune.sde file.

The DEFAULTS configuration keyword in the dbtune.sde file contains the B_STORAGE storage parameter with the Oracle tablespace parameter commented out.

```
##DEFAULTS

GEOMETRY_STORAGE     "SDEBINARY"
ATTRIBUTE_BINARY     "LONGRAW"
B_STORAGE            "PCTFREE 10 PCTUSED 90 INITRANS 4
#                     TABLESPACE   <default business table tablespace name>
                      STORAGE (FREELISTS 4 MINEXTENTS 1 PCTINCREASE 0)"
```

Edit the dbtune.sde file, remove the "#" comment character, and enter the name of the tablespace you want to store business tables in by default.

```
##DEFAULTS

GEOMETRY_STORAGE     "SDEBINARY"
ATTRIBUTE_BINARY     "LONGRAW"
B_STORAGE            "PCTFREE 10 PCTUSED 90 INITRANS 4
                      TABLESPACE   ROADS
                      STORAGE (FREELISTS 4 MINEXTENTS 1 PCTINCREASE 0)"
```

When the setup program loads your customized dbtune, configuration keywords are written into the DBTUNE table.

Editing the DBTUNE table

If you need to change the contents of the DBTUNE table after it is loaded, you should use the **sdedbtune** utility and follow these steps.

1. Export the DBTUNE table to a text file using the **sdedbtune –o export** command.

2. Edit the resulting file with a UNIX file-based editor, such as "vi", or a Windows NT file-based editor such as "notepad".

3. Import the edited file to the DBTUNE table using the **sdedbtune –o import** command.

In the following example, the DBTUNE table is exported to the dbtune.out file, and the file is edited with the UNIX "vi" file-based editor.

```
$ sdedbtune -o export -f dbtune.out -u sde -p fredericton

ArcSDE   8.1               Wed Oct  4 22:32:44 PDT 2000
Attribute        Administration Utility
----------------------------------------------------

        Successfully exported to file SDEHOME\etc\dbtune.out

$ vi dbtune.out

$ sdedbtune -o import -f dbtune.out -u sde -p fredericton -N

ArcSDE   8.1               Wed Oct  4 22:32:44 PDT 2000
Attribute        Administration Utility
```

```
-------------------------------------------------

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

The sdedbtune administration tool always exports the file to the $SDEHOME/etc directory. You cannot relocate the file to another directory. By not allowing the relocation of the file, the sdedbtune command ensures the dbtune parameters remain under the ownership of the ArcSDE administrator.

Adding keywords to the DBTUNE table

You may add parameter groups to the DBTUNE table for any special purpose. For instance, you may wish to create certain feature classes in a newly created tablespace that is segregated from the rest of the data.

To add keywords, follow the instructions above for editing the DBTUNE table. When you edit the export file, it is often a good idea to create a new parameter group as a cut and paste copy of an existing parameter group in order to avoid introducing syntax errors. You may then edit the configuration keyword and any of the strings to desired new values before saving the dbtune file and importing it back into the DBTUNE table.

Using the DBTUNE table

At its most basic level, the DBTUNE table provides configuration strings that are appended to a CREATE TABLE or CREATE INDEX statement in SQL. The configuration strings specify storage parameters that must be considered valid by the Oracle server.

Selecting the configuration string

The choice of configuration strings by an ArcSDE application depends upon the operation being performed and the type of object it is being performed on, as well as the configuration keyword. For example, if the type of operation is CREATE TABLE and the type of table being created is a business table, the parameter_name of B_STORAGE will be used to determine the configuration string.

The ArcSDE application then searches the DBTUNE table for a configuration string whose configuration keyword matches the given configuration keyword and whose parameter_name matches the chosen name.

When running the **sdetable** and certain other ArcSDE administration commands, you may specify your own configuration keyword. When running ArcSDE applications, the configuration keyword is specified to the ArcSDE server automatically.

If the application cannot find the requested configuration string within the specified parameter group, it searches the DEFAULT parameter group. If the requested configuration string cannot located within the DEFAULT parameter group, the ArcSDE use no configuration string and the CREATE TABLE or CREATE INDEX statement picks up the defaults according to the rules of the Oracle server.

Table parameters

Table parameters define the storage configuration of an Oracle table. The configuration string associated with the parameter is appended to the CREATE TABLE statement prior to its execution by ArcSDE.

Valid entries for an ArcSDE table include any parameter allowable to the right of the column list in the CREATE TABLE statement, especially including the TABLESPACE and STORAGE clauses.

For example, if you define the B_STORAGE parameter in this manner:

```
B_STORAGE "tablespace roads_tabsp storage (initial 10M next 1M minextents 1
maxextents 100 pctincrease 0 freelists 4) initrans 4 pctfree 10 pctused 90"
```

ArcSDE would execute the following Oracle CREATE TABLE statement:

```
CREATE TABLE roads (road_id integer, name varchar2(32), surface_code integer)
tablespace roads_tabsp storage (initial 10M next 1M minextents 1 maxextents
100 pctincrease 0 freelists 4) initrans 4 pctfree 10 pctused 90;
```

Index parameters

Index parameters define the storage configuration of an Oracle index. The index parameter is appended to an Oracle CREATE INDEX statement prior to its execution by ArcSDE.

Valid entries in an ArcSDE index parameter include any parameter allowable by the Oracle server to the right of the column list of the CREATE INDEX statement, especially including the TABLESPACE and STORAGE clauses.

For example, if you specify the B_INDEX_USER parameter in this manner:

```
B_INDEX_USER "tablespace roads_idx_tabsp storage(initial 1M next 512K
minextents 1 maxextents 100 pctincrease 0 freelists 4) initrans 4 pctfree 10"
```

ArcSDE would assemble a CREATE INDEX statement like this:

```
CREATE INDEX roads_idx on roads (road_id)
tablespace roads_idx_tabsp storage (initial 1M next 512K minextents 1
maxextents 100 pctincrease 0 freelists 4) initrans 4 pctfree 10;
```

Defining the storage parameters

Configuration keywords may include any combination of three basic types of storage parameters: meta parameters, table parameters, and index parameters.

Meta parameters define the way certain types of data will be stored, the environment of a configuration keyword, or a comment that describes the configuration keyword. Table and index parameters establish the storage characteristics of, respectively, tables and indexes.

The business table storage parameter

A business table is any Oracle table created by an ArcSDE client, the sdetable administration command, or the ArcSDE C API SE_table_create function.

Use the DBTUNE table's B_STORAGE storage parameter to define the storage configuration of a business table.

The business table index storage parameters

Three index storage parameters exist to support the creation of business table indexes.

The B_INDEX_USER storage 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 storage parameter holds the storage configuration of the index that ArcSDE creates on a register table's object ID column, commonly referred to as the ROWID.

Note: ArcSDE registers all tables that it creates. Tables not created by ArcSDE can also be registered with the **sdetable –o alter_reg** command or with ArcCatalog. The SDE.TABLE_REGISTRY system table maintains a list of the currently registered tables.

The B_INDEX_SHAPE storage parameter holds the storage configuration of the spatial column index that 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 storage parameters

Registering a business table 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 rows are modified.

ArcSDE creates two tables—the adds table and the deletes table—for each table that is registered as multiversioned.

The A_STORAGE storage parameter maintains the storage configuration of the adds table. Four other storage 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 storage parameter holds the storage configuration of the index that 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, which the adds table is associated with.

The A_INDEX_STATEID storage parameter holds the storage configuration of the index that 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, which the adds table is associated with.

The A_INDEX_SHAPE storage parameter holds the storage configuration of the index that 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 storage parameter holds the storage configuration of user-defined indexes that ArcSDE creates on the adds table. The user-defined indexes on the business tables are duplicated on the adds table.

The D_STORAGE storage parameter holds the storage configuration of the deletes table. Two other storage parameters hold the storage configuration of the indexes that 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 storage parameter holds the storage configuration of the D<n>_IDX1 index that ArcSDE creates on the deletes table's SDE_STATE_ID and SDE_DELETES_ROW_ID columns.

The D_INDEX_DELETED_AT storage parameter holds the storage configuration of the D<n>_IDX2 index that ArcSDE creates on the deletes table's SDE_DELETED_AT column.

Note: If a configuration 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 storage parameters of the configuration keyword the business table was created with.

Feature class storage parameters

A feature class created with an ArcSDE compressed binary storage (LONG RAW or BLOB datatype) format adds two tables to the Oracle 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_* storage parameters of the business table.

The F_STORAGE storage parameter holds the Oracle CREATE TABLE storage configuration string of the feature table. The feature table is created as F_<n>, where <n> is the layer ID of the table's feature class as found in the SDE.LAYERS table.

The F_INDEX_FID storage parameter holds the Oracle CREATE INDEX storage configuration string of the feature tables spatial column index. The spatial column is created as F_<n>_IX1, where <n> is the layer ID of the index's feature class as found in the SDE.LAYERS table.

The F_INDEX_AREA storage parameter holds the Oracle CREATE INDEX storage configuration of the feature tables area column index. The spatial column is created as F_<n>_AREA_IX2, where <n> is the layer ID of the index's feature class as found in the SDE.LAYERS table.

The F_INDEX_LEN storage parameter holds the Oracle CREATE INDEX storage configuration of the feature table’s length column index. The spatial column is created as F_<n>_LEN_IX3, where <n> is the layer ID of the index's feature class as found in the SDE.LAYERS table.

The S_STORAGE storage parameter holds the Oracle CREATE TABLE storage configuration of the spatial index table. The spatial index table is created as S_<n>, where

<n> is the layer ID of the spatial index table's feature class as found in the SDE.LAYERS table.

The S_INDEX_ALL storage parameter holds the Oracle CREATE INDEX storage configuration of the spatial table first index. The spatial index table is created as S_<n>_IX1, where <n> is the layer ID of the index's feature as class found in the SDE.LAYERS table.

The S_INDEX_SP_FID storage parameter holds the Oracle CREATE INDEX storage configuration of the spatial table second index. The spatial index table is created as S_<n>_IX2, where <n> is the layer ID of the index's feature as class found in the SDE.LAYERS table.

Raster table storage 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.

The RAS_STORAGE storage parameter holds the Oracle CREATE TABLE storage configuration of the RAS table.

The RAS_INDEX_ID storage parameter holds the Oracle CREATE TABLE storage configuration of the RAS table index.

The BND_STORAGE storage parameter holds the Oracle CREATE TABLE storage configuration of the BND table index.

The BND_INDEX_ COMPOSITE storage parameter holds the Oracle CREATE INDEX storage configuration of the BND table’s composite column index.

The BND_INDEX_ID storage parameter holds the Oracle CREATE INDEX storage configuration of the BND table’s rid column index.

The AUX_STORAGE storage parameter holds the Oracle CREATE TABLE storage configuration of the AUX table.

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

The BLK_STORAGE storage parameter holds the Oracle CREATE TABLE storage configuration of the BLK table.

The BLK_INDEX_COMPOSITE storage parameter holds the Oracle CREATE TABLE storage configuration of the BLK table's index.

Arranging storage parameters by keyword

Storage parameters of the DBTUNE table are grouped by keyword. The following keywords are present by default in the DBTUNE table.

- DEFAULTS
- DATA_DICTIONARY

- IMS_METADATARELATIONSHIPS
- IMS_METADATA
- IMS_METADATATAGS
- IMS_METADATATHUMBNAILS
- IMS_METADATAUSERS
- IMS_METADATAVALUES
- IMS_METADATAWORDINDEX
- IMS_METADATAWORD

DEFAULTS keyword

Each DBTUNE table has a fully populated DEFAULTS configuration keyword.

The DEFAULTS configuration 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 configuration keyword is used. If you do not include a storage parameter in a configuration keyword that you have defined, ArcSDE uses the storage parameter from the DEFAULTS configuration keyword.

The DEFAULTS configuration keyword relieves you of the need to define all the storage parameters for each of your configuration keywords. The storage parameters of the DEFAULTS configuration 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 configuration keyword storage parameter in the dbtune.sde file, it automatically adds the storage parameter. If you import a DBTUNE file with the sdedbtune command, the command automatically adds default storage parameters that are missing. ArcSDE will detect the presence of the following list of storage parameters and insert the storage parameter and the default configuration string.

```
##DEFAULTS

ATTRIBUTE_BINARY        "LONGRAW"
A_INDEX_ROWID   "PCTFREE 10 INITRANS 4 STORAGE (FREELISTS 4 MINEXTENTS 1
                 PCTINCREASE 0)"
A_INDEX_SHAPE   "PCTFREE 10 INITRANS 4 STORAGE (FREELISTS 4 MINEXTENTS 1
                 PCTINCREASE 0)"
A_INDEX_STATEID "PCTFREE 10 INITRANS 4 STORAGE (FREELISTS 4 MINEXTENTS 1
                 PCTINCREASE 0)"
A_INDEX_USER    "PCTFREE 10 INITRANS 4 STORAGE (FREELISTS 4 MINEXTENTS 1
                 PCTINCREASE 0)"
A_STORAGE       "PCTFREE 10 PCTUSED 90 INITRANS 4 STORAGE (FREELISTS 4
                 MINEXTENTS 1 PCTINCREASE 0)"
B_INDEX_ROWID   "PCTFREE 10 INITRANS 2 STORAGE (FREELISTS 4 MINEXTENTS 1
                 PCTINCREASE 0)"
B_INDEX_SHAPE   "PCTFREE 10 INITRANS 2 STORAGE (FREELISTS 4 MINEXTENTS 1
                 PCTINCREASE 0)"
B_INDEX_USER    "PCTFREE 10 INITRANS 2 STORAGE (FREELISTS 4 MINEXTENTS 1
                 PCTINCREASE 0)"
B_STORAGE       "PCTFREE 10 PCTUSED 90 INITRANS 1 STORAGE (FREELISTS 4
                 MINEXTENTS 1 PCTINCREASE 0)"
D_INDEX_DELETED_AT      "PCTFREE 10 INITRANS 4 STORAGE (FREELISTS 4
                 MINEXTENTS 1 PCTINCREASE 0)"
D_INDEX_STATE_ROWID     "PCTFREE 10 INITRANS 4 STORAGE (FREELISTS 4
                 MINEXTENTS 1 PCTINCREASE 0)"
D_STORAGE       "PCTFREE 10 PCTUSED 90 INITRANS 4 STORAGE (FREELISTS 4
                 MINEXTENTS 1 PCTINCREASE 0)"
```

```
F_INDEX_AREA     "PCTFREE 10 INITRANS 4 STORAGE (FREELISTS 4 MINEXTENTS 1
                  PCTINCREASE 0)"
F_INDEX_FID      "PCTFREE 10 INITRANS 4 STORAGE (FREELISTS 4 MINEXTENTS 1
                  PCTINCREASE 0)"
F_INDEX_LEN      "PCTFREE 10 INITRANS 4 STORAGE (FREELISTS 4 MINEXTENTS 1
                  PCTINCREASE 0)"
F_STORAGE        "PCTFREE 30 PCTUSED 70 INITRANS 4 STORAGE (FREELISTS 4
                  MINEXTENTS 1 PCTINCREASE 0)"
GEOMETRY_STORAGE   "SDEBINARY"
S_INDEX_ALL      "PCTFREE 10 INITRANS 4 STORAGE (FREELISTS 4 MINEXTENTS 1
                  PCTINCREASE 0)"
S_INDEX_SP_FID   "PCTFREE 10 INITRANS 4 STORAGE (FREELISTS 4 MINEXTENTS 1
                  PCTINCREASE 0)"
S_STORAGE        "PCTFREE 10 PCTUSED 90 INITRANS 4 STORAGE (FREELISTS 4
                  MINEXTENTS 1 PCTINCREASE 0)"
UI_TEXT          ""

END
```

Setting the system table DATA_DICTIONARY configuration keyword

During the execution of the **sdesetupora** administration tool, the ArcSDE and geodatabase system tables and indexes are created with the storage parameters of the DATA_DICTIONARY configuration keyword. You may customize the configuration keyword in the *dbtune.sde* file prior to running the **sdesetupora*** tool. In this way you can change default storage parameters of the DATA_DICTIONARY configuration 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—VERSION, STATES, STATE_LINEAGES, and MVTABLES_MODIFIED—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 very active. Separating these objects into their own tablespace allows you to position their data files with data files that experience low I/O activity and thus minimize disk I/O contention.

If the dbtune.sde file does not contain the DATA_DICTIONARY configuration keyword, or if any of the required parameters are missing from the configuration keyword, the following records will be inserted into the DATA_DICTIONARY when the table is created. (Note that the DBTUNE file entries are provided here for readability.)

```
##DATA_DICTIONARY

B_INDEX_ROWID               "PCTFREE 10 INITRANS 4 STORAGE (FREELISTS 4
                             INITIAL 40K NEXT 40K MINEXTENTS 1 MAXEXTENTS 200
                             PCTINCREASE 0)"

B_INDEX_USER                "PCTFREE 10 INITRANS 4 STORAGE (FREELISTS 4
                             INITIAL 40K NEXT 40K MINEXTENTS 1 MAXEXTENTS 200
                             PCTINCREASE 0)"

B_STORAGE                   "PCTFREE 10 PCTUSED 90 INITRANS 4
                             STORAGE (FREELISTS 4 INITIAL 40K NEXT 40K
                             MINEXTENTS 1 MAXEXTENTS 200 PCTINCREASE 0)"

STATE_LINEAGES_TABLE        "PCTFREE 10 INITRANS 4 STORAGE (FREELISTS 4
                             INITIAL 40M NEXT 10M MINEXTENTS 1 MAXEXTENTS

200)"

STATES_TABLE                "PCTFREE 10 PCTUSED 90 INITRANS 4
                             STORAGE (FREELISTS 4 INITIAL 8M NEXT 1M
                             MINEXTENTS 1 MAXEXTENTS 200)"
```

```
STATES_INDEX                "PCTFREE 10 INITRANS 4 STORAGE (FREELISTS 4
                             INITIAL 1M NEXT 128K MINEXTENTS 1 MAXEXTENTS
200)"

MVTABLES_MODIFIED_TABLE     "PCTFREE 10 PCTUSED 90 INITRANS 4
                             STORAGE (FREELISTS 4 INITIAL 2M NEXT 1M
                             MINEXTENTS 1 MAXEXTENTS 200)"

MVTABLES_MODIFIED_INDEX     "PCTFREE 10 INITRANS 4 STORAGE (FREELISTS 4
                             INITIAL 2M NEXT 1M MINEXTENTS 1 MAXEXTENTS 200)"

VERSIONS_TABLE              "PCTFREE 10 PCTUSED 90 INITRANS 4
                             STORAGE (FREELISTS 4 INITIAL 256K
                             NEXT 128K MINEXTENTS 1 MAXEXTENTS 200)"

VERSIONS_INDEX              "PCTFREE 10 INITRANS 4 STORAGE (FREELISTS 4
                             INITIAL 128K NEXT 128K MINEXTENTS 1
                             MAXEXTENTS 200)"

END
```

The TOPOLOGY keyword

The TOPOLOGY keyword controls the storage of topology tables, which are named POINTERRORS, LINEERRORS, POLYERRORS and DIRTYAREAS. An SDE instance must have a valid topology keyword in the dbtune table, or topology will not be built.

The DIRTYAREAS table maintains information on areas within a layer that have been changed. Because it tracks versions, data will be inserted or updated but not deleted during normal use. The DIRTYAREAS table will reduce in size only when database versions get compressed.

Because the DIRTYAREAS table is much more active than the remaining topology tables, the TOPOLOGY keyword may be compound. You may specify the DIRTYAREAS suffix to list configuration string to be used to create the topology tables.

For Oracle, the default values for TOPOLOGY and TOPOLOGY::DIRTYAREAS are

```
##TOPOLOGY_DEFAULTS
A_INDEX_ROWID       "PCTFREE 10 INITRANS 4 STORAGE (FREELISTS 4 PCTINCREASE 0)
                    NOLOGGING"
A_INDEX_SHAPE       "PCTFREE 10 INITRANS 4 STORAGE (FREELISTS 4 MINEXTENTS 1
                    PCTINCREASE 0) NOLOGGING"
A_INDEX_STATEID     "PCTFREE 10 INITRANS 4 STORAGE (FREELISTS 4 PCTINCREASE 0)
                    NOLOGGING"
A_INDEX_USER        "PCTFREE 10 INITRANS 4 STORAGE (FREELISTS 4 PCTINCREASE 0)
                    NOLOGGING"
A_STORAGE           "PCTFREE 10 PCTUSED 90 INITRANS 4 STORAGE (FREELISTS 4
                    PCTINCREASE 0)"
B_INDEX_ROWID       "PCTFREE 10 INITRANS 4 STORAGE (FREELISTS 4 PCTINCREASE 0)
                    NOLOGGING"
B_INDEX_SHAPE       "PCTFREE 10 INITRANS 4 STORAGE (FREELISTS 4 MINEXTENTS 1
                    PCTINCREASE 0) NOLOGGING"
B_INDEX_USER        "PCTFREE 10 INITRANS 4 STORAGE (FREELISTS 4 PCTINCREASE 0)
                    NOLOGGING"
B_STORAGE           "PCTFREE 10 PCTUSED 90 INITRANS 4 STORAGE (FREELISTS 4
                    PCTINCREASE 0)"
D_INDEX_DELETED_AT  "PCTFREE 10 INITRANS 4 STORAGE (FREELISTS 4 PCTINCREASE 0)
                    NOLOGGING"
D_INDEX_STATE_ROWID "PCTFREE 10 INITRANS 4 STORAGE (FREELISTS 4 PCTINCREASE 0)
                    NOLOGGING"
D_STORAGE           "PCTFREE 10 PCTUSED 90 INITRANS 4 STORAGE (FREELISTS 4
                    PCTINCREASE 0)"
F_INDEX_AREA        "PCTFREE 10 INITRANS 4 STORAGE (FREELISTS 4 MINEXTENTS 1
                    PCTINCREASE 0) NOLOGGING"
F_INDEX_FID         "PCTFREE 10 INITRANS 4 STORAGE (FREELISTS 4 MINEXTENTS 1
                    PCTINCREASE 0) NOLOGGING"
F_INDEX_LEN         "PCTFREE 10 INITRANS 4 STORAGE (FREELISTS 4 MINEXTENTS 1
                    PCTINCREASE 0) NOLOGGING"
F_STORAGE           "PCTFREE 10 PCTUSED 90 INITRANS 4 STORAGE (FREELISTS 4
```

```
                    MINEXTENTS 1 PCTINCREASE 0)"
S_INDEX_ALL         "PCTFREE 10 INITRANS 4 STORAGE (FREELISTS 4 MINEXTENTS 1
                    PCTINCREASE 0) NOLOGGING"
S_INDEX_SP_FID      "PCTFREE 10 INITRANS 4 STORAGE (FREELISTS 4 MINEXTENTS 1
                    PCTINCREASE 0) NOLOGGING"
S_STORAGE           "PCTFREE 10 PCTUSED 90 INITRANS 4 STORAGE (FREELISTS 4
                    MINEXTENTS 1 PCTINCREASE 0)"
UI_TOPOLOGY_TEXT    "The topology default configuration"
END

##TOPOLOGY_DEFAULTS::DIRTYAREAS
A_INDEX_ROWID       "PCTFREE 10 INITRANS 4 STORAGE (FREELISTS 4 PCTINCREASE 0)
                    NOLOGGING"
A_INDEX_SHAPE       "PCTFREE 10 INITRANS 4 STORAGE (FREELISTS 4 MINEXTENTS 1
                    PCTINCREASE 0) NOLOGGING"
A_INDEX_STATEID     "PCTFREE 10 INITRANS 4 STORAGE (FREELISTS 4 PCTINCREASE 0)
                    NOLOGGING"
A_INDEX_USER        "PCTFREE 10 INITRANS 4 STORAGE (FREELISTS 4 PCTINCREASE 0)
                    NOLOGGING"
A_STORAGE           "PCTFREE 10 PCTUSED 90 INITRANS 4 STORAGE (FREELISTS 4
                    PCTINCREASE 0)"
B_INDEX_ROWID       "PCTFREE 10 INITRANS 4 STORAGE (FREELISTS 4 PCTINCREASE 0)
                    NOLOGGING"
B_INDEX_SHAPE       "PCTFREE 10 INITRANS 4 STORAGE (FREELISTS 4 MINEXTENTS 1
                    PCTINCREASE 0) NOLOGGING"
B_INDEX_USER        "PCTFREE 10 INITRANS 4 STORAGE (FREELISTS 4 PCTINCREASE 0)
                    NOLOGGING"
B_STORAGE           "PCTFREE 10 PCTUSED 90 INITRANS 4 STORAGE (FREELISTS 4
                    PCTINCREASE 0)"
D_INDEX_DELETED_AT  "PCTFREE 10 INITRANS 4 STORAGE (FREELISTS 4 PCTINCREASE 0)
                    NOLOGGING"
D_INDEX_STATE_ROWID "PCTFREE 10 INITRANS 4 STORAGE (FREELISTS 4 PCTINCREASE 0)
                    NOLOGGING"
D_STORAGE           "PCTFREE 10 PCTUSED 90 INITRANS 4 STORAGE (FREELISTS 4
                    PCTINCREASE 0)"
F_INDEX_AREA        "PCTFREE 10 INITRANS 4 STORAGE (FREELISTS 4 MINEXTENTS 1
                    PCTINCREASE 0) NOLOGGING"
F_INDEX_FID         "PCTFREE 10 INITRANS 4 STORAGE (FREELISTS 4 MINEXTENTS 1
                    PCTINCREASE 0) NOLOGGING"
F_INDEX_LEN         "PCTFREE 10 INITRANS 4 STORAGE (FREELISTS 4 MINEXTENTS 1
                    PCTINCREASE 0) NOLOGGING"
F_STORAGE           "PCTFREE 10 PCTUSED 90 INITRANS 4 STORAGE (FREELISTS 4
                    MINEXTENTS 1 PCTINCREASE 0)"
S_INDEX_ALL         "PCTFREE 10 INITRANS 4 STORAGE (FREELISTS 4 MINEXTENTS 1
                    PCTINCREASE 0) NOLOGGING"
S_INDEX_SP_FID      "PCTFREE 10 INITRANS 4 STORAGE (FREELISTS 4 MINEXTENTS 1
                    PCTINCREASE 0) NOLOGGING"
S_STORAGE           "PCTFREE 10 PCTUSED 90 INITRANS 4 STORAGE (FREELISTS 4
                    MINEXTENTS 1 PCTINCREASE 0)"
END
```

The IMS METADATA keyword

The IMS METADATA keyword controls the storage of the IMS Metadata tables. This keyword is a standard part of the dbtune table. If the IMS_METADATA storage parameters are not present in the dbtune file when it is imported into the DBTUNE table, ArcSDE supplies software defaults.

The software defaults have the same settings as the parameters listed in the dbtune.sde table that is shipped with ArcSDE. The Oracle parameter settings such as the *initial* and *next* should be sufficient. However, you will need to edit the tablespace names.

For more information about installing IMS Metadata and the associated tables and indexes refer to ArcIMS Metadata Server documentation.

The IMS_METADATA storage parameters control the storage of the ims_metadata feature class. Four indexes are created on the ims_metadata business table. ArcSDE creates the following default IMS_METADATA keyword in the DBTUNE table if the keyword is missing from the dbtune file when it is imported.

The IMS metadata keywords are as follows:

```
##IMS_METADATA

B_STORAGE           "PCTFREE 10 PCTUSED 90 INITRANS 4
                     STORAGE (FREELISTS 4 MINEXTENTS 1 PCTINCREASE 0
                     INITIAL 100M )"

B_INDEX_ROWID       "PCTFREE 10 INITRANS 4
                     STORAGE (FREELISTS 4 INITIAL 6M
                     MINEXTENTS 1 MAXEXTENTS 200 PCTINCREASE 0) NOLOGGING"

B_INDEX_SHAPE       "PCTFREE 10 INITRANS 4
                     STORAGE (FREELISTS 4 MINEXTENTS 1 PCTINCREASE 0
                     INITIAL 6M) NOLOGGING"

B_INDEX_USER        "PCTFREE 10 INITRANS 4
                     STORAGE (FREELISTS 4 MINEXTENTS 1 PCTINCREASE 0
                    INITIAL 6M) NOLOGGING"

F_STORAGE           "PCTFREE 30 PCTUSED 70 INITRANS 4
                     STORAGE (FREELISTS 4 MINEXTENTS 1 PCTINCREASE 0
                     INITIAL 40M)"

F_INDEX_FID         "PCTFREE 10 INITRANS 4
                     STORAGE (FREELISTS 4 MINEXTENTS 1 PCTINCREASE 0
                     INITIAL 3M) NOLOGGING"

F_INDEX_AREA        "PCTFREE 10 INITRANS 4
                     STORAGE (FREELISTS 4 MINEXTENTS 1 PCTINCREASE 0
                     INITIAL 3M) NOLOGGING"

F_INDEX_LEN         "PCTFREE 10 INITRANS 4
                     STORAGE (FREELISTS 4 MINEXTENTS 1 PCTINCREASE 0
                     INITIAL 3M) NOLOGGING"

S_STORAGE           "PCTFREE 10 PCTUSED 90 INITRANS 4
                     STORAGE (FREELISTS 4 MINEXTENTS 1 PCTINCREASE 0
                     INITIAL 16M)"

S_INDEX_ALL         "PCTFREE 10 INITRANS 4
                     STORAGE (FREELISTS 4 MINEXTENTS 1 PCTINCREASE 0
                     INITIAL 16M) NOLOGGING"

S_INDEX_SP_FID      "PCTFREE 10 INITRANS 4
                     STORAGE (FREELISTS 4 MINEXTENTS 1 PCTINCREASE 0
                     INITIAL 3M) NOLOGGING"

COMMENT             "The IMS metatdata feature class"

UI_TEXT             ""

END
```

The IMS_METADATARELATIONSHIPS keyword controls the storage of the ims_metadatarelationships business table. Three indexes are created on 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_STORAGE           "PCTFREE 10 PCTUSED 90 INITRANS 4
                     STORAGE (FREELISTS 4 MINEXTENTS 1 PCTINCREASE 0
                     INITIAL 256K)"

B_INDEX_ROWID       "PCTFREE 10 INITRANS 4
                     STORAGE (FREELISTS 4 MINEXTENTS 1 PCTINCREASE 0
                     INITIAL 64K) NOLOGGING"

B_INDEX_USER        "PCTFREE 10 INITRANS 4
                     STORAGE (FREELISTS 4 MINEXTENTS 1 PCTINCREASE 0
                     INITIAL 64K) NOLOGGING"

END
```

The IMS_METADATATAGS keyword controls the storage of the ims_metadatatags business table. Two indexes are created on the ims_metadatatags business table. ArcSDE creates the following default IMS_METADATATAGS keyword in the DBTUNE table if the keyword is missing from the dbtune file when it is imported.

```
##IMS_METADATATAGS

B_STORAGE           "PCTFREE 10 PCTUSED 90 INITRANS 4
                     STORAGE (FREELISTS 4 MINEXTENTS 1 PCTINCREASE 0
                     INITIAL 64K)"

B_INDEX_ROWID       "PCTFREE 10 INITRANS 4
                     STORAGE (FREELISTS 4 MINEXTENTS 1 PCTINCREASE 0
                     INITIAL 64K) NOLOGGING"

B_INDEX_USER        "PCTFREE 10 INITRANS 4
                     STORAGE (FREELISTS 4 MINEXTENTS 1 PCTINCREASE 0
                     INITIAL 64K) NOLOGGING"

END
```

The IMS_METADATATHUMBNAILS keyword controls the storage of the ims_metadatathumbnails business table. One index is created on 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_STORAGE           "PCTFREE 10 PCTUSED 90 INITRANS 4
                     STORAGE (FREELISTS 4 MINEXTENTS 1 PCTINCREASE 0
                     INITIAL 4M)"

B_INDEX_USER        "PCTFREE 10 INITRANS 4
                     STORAGE (FREELISTS 4 MINEXTENTS 1 PCTINCREASE 0
                     INITIAL 256K) NOLOGGING"

END
```

The IMS_METADATAUSERS keyword controls storage of the ims_metadatausers business table. One index is created on the ims_metadatausers business table. ArcSDE creates the following default IMS_METADATAUSERS keyword in the DBTUNE table if the keyword is missing from the dbtune file when it is imported.

```
##IMS_METADATAUSERS

B_STORAGE           "PCTFREE 10 PCTUSED 90 INITRANS 4
                     STORAGE (FREELISTS 4 MINEXTENTS 1 PCTINCREASE 0
                     INITIAL 7M)"

B_INDEX_ROWID       "PCTFREE 10 INITRANS 4
                     STORAGE (FREELISTS 4 MINEXTENTS 1 PCTINCREASE 0
                     INITIAL 512K) NOLOGGING"

B_INDEX_USER        "PCTFREE 10 INITRANS 4
                     STORAGE (FREELISTS 4 MINEXTENTS 1 PCTINCREASE 0
                     INITIAL 512K) NOLOGGING"

END
```

The IMS_METADATAVALUES keyword controls the storage of the ims_metadatavalues business table. Two indexes are created on 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_STORAGE           "PCTFREE 10 PCTUSED 90 INITRANS 4
                     STORAGE (FREELISTS 4 MINEXTENTS 1 PCTINCREASE 0
                     INITIAL 7M)"

B_INDEX_USER        "PCTFREE 10 INITRANS 4
```

```
                    STORAGE (FREELISTS 4 MINEXTENTS 1 PCTINCREASE 0
                    INITIAL 512K) NOLOGGING"

END
```

The IMS_METADATAWORDINDEX keyword controls the storage of the ims_metadatawordindex business table. Three indexes are created on 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_STORAGE           "PCTFREE 10 PCTUSED 90 INITRANS 4
                     STORAGE (FREELISTS 4 MINEXTENTS 1 PCTINCREASE 0
                     INITIAL 10M)"

B_INDEX_USER        "PCTFREE 10 INITRANS 4
                     STORAGE (FREELISTS 4 MINEXTENTS 1 PCTINCREASE 0
                     INITIAL 512K) NOLOGGING"

END
```

The IMS_METADATAWORDS keyword controls the storage of the ims_metadatawords business table. One index is created on 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_STORAGE           "PCTFREE 10 PCTUSED 90 INITRANS 4
                     STORAGE (FREELISTS 4 MINEXTENTS 1 PCTINCREASE 0
                     INITIAL 7M)"

B_INDEX_ROWID       "PCTFREE 10 INITRANS 4
                     STORAGE (FREELISTS 4 MINEXTENTS 1 PCTINCREASE 0
                     INITIAL 512K) NOLOGGING"

B_INDEX_USER        "PCTFREE 10 INITRANS 4
                     STORAGE (FREELISTS 4 MINEXTENTS 1 PCTINCREASE 0
                     INITIAL 512K) NOLOGGING"

END
```

Setting the storage format—the GEOMETRY_STORAGE parameter

ArcSDE for Oracle provides three spatial data storage formats. The GEOMETRY_STORAGE parameter indicates which geometry storage method is to be used. The GEOMETRY_STORAGE parameter has the following values:

- ArcSDE compressed binary stored in LONG RAW data type. This is the default spatial storage method of ArcSDE for Oracle.

  Set the GEOMETRY_STORAGE parameter to SDEBINARY if you wish to store your spatial data in this format. If the GEOMETRY_STORAGE parameter is not set, the SDEBINARY format is assumed.

- ArcSDE compressed binary stored as a BLOB data type. This data type can be replicated through Oracle Advanced Replication.

  Set the GEOMETRY_STORAGE parameter to SDELOB if you wish to store your spatial data in this format. If you wish to make this format the default, set the

GEOMETRY_STORAGE parameter to SDELOB in the DEFAULTS configuration keyword.

- Oracle Spatial geometry type. Available starting at Oracle8*i,* this object-relational model extends the database model to include an SDO_GEOMETRY type in the Oracle DBMS.

  Set the GEOMETRY_STORAGE parameter to SDO_GEOMETRY if you wish to store your spatial data in this format. If you wish to make this format the default, set the GEOMETRY_STORAGE parameter to SDO_GEOMETRY in the DEFAULTS configuration keyword.

The default value for GEOMETRY_STORAGE is SDEBINARY.

If all of the feature classes in your database use the same geometry storage method, set the GEOMETRY_STORAGE parameter once in the DEFAULTS configuration keyword. To change the default GEOMETRY_STORAGE from SDEBINARY to SDO_GEOMETRY, the following change is made:

```
## DEFAULTS
GEOMETRY_STORAGE    SDO_GEOMETRY
<other parameters>
END
```

For convenience, three predefined configuration keywords are provided in the dbtune.proto file to allow the use of each of the supported geometry storage methods. These configurations are defined as follows:

```
## SDEBINARY
GEOMETRY_STORAGE    SDEBINARY
END
##SDELOB
GEOMETRY_STORAGE    SDELOB
END
## SDO_GEOMETRY
GEOMETRY_STORAGE    SDO_GEOMETRY
END
```

Setting the COMPRESS_ROLLBACK_SEGMENT storage parameter

Periodically compressing the versioned database's state-tree is a required maintenance procedure. The state-tree can be compressed with either the compress operation of the **sdeversion** administration command or the SE_state_compress_tree() C API function.

The transactions of the compress operation tend to be large; we recommend that you create a separate large rollback segment to contain their changes. The COMPRESS_ROLLBACK_SEGMENT storage parameter stores the name of a rollback segment that you have created for this purpose. Add the COMPRESS_ROLLBACK_SEGMENT storage parameter to the DEFAULTS configuration keyword.

For more information on creating the rollback segment used for compressing the state-tree, see Chapter 2, ‘Essential Oracle configuring and tuning’.

Changing the ATTRIBUTE_BINARY storage format

ArcSDE defines attribute columns used to store binary data as LONG RAW or as BLOB. The default is LONG RAW.

If the storage parameter is not set in the DEFAULTS configuration keyword when a DBTUNE file is imported by the **sdedbtune** administration tool, ArcSDE inserts the ATTRIBUTE_BINARY storage parameter under the DEFAULTS configuration keyword with a configuration string set to LONGRAW.

Changing the appearance of DBTUNE configuration keywords in the ArcInfo user interface

Starting in ArcSDE 8.1, ArcSDE introduces two new storage parameters that will support the ArcInfo user interfaces UI_TEXT and UI_NETWORK_TEXT. ArcSDE administrators can add one of these storage parameters to each configuration keyword to communicate to the ArcInfo schema builders the intended use of the configuration keyword. The configuration string of these storage parameters will appear in ArcInfo interface DBTUNE configuration keyword scrolling lists.

The UI_TEXT storage parameter should be added to configuration keywords that will be used to build tables, feature classes, and indexes.

The UI_NETWORK_TEXT storage parameter should be added to parent network configuration keywords.

Adding a comment to a configuration keyword

The COMMENT storage parameter allows you to add informative text that describes such things as a configuration keyword's intended use, the last time it was changed, or who created it.

LOG FILE configuration keywords

Log files 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.

You may create a configuration 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 configuration keyword. If this configuration keyword is not found, ArcSDE will use the storage parameters of the LOGFILE_DEFAULTS configuration keyword to create the SDE_LOGFILES and SDE_LOGFILE_DATA tables.

ArcSDE always creates the DBTUNE table with a LOGFILE_DEFAULTS configuration keyword. If you do not specify this configuration keyword in a DBTUNE file imported by the sdedbtune command, ArcSDE will populate the DBTUNE table with default LOGFILE_DEFAULTS storage parameters. Further, if the DBTUNE file lacks some of the LOGFILE_DEFAULTS configuration keyword storage parameters, ArcSDE supplies the rest. Therefore, the LOGFILE_DEFAULTS configuration keyword is always fully populated.

If a user-specific configuration keyword exists, but some of the storage parameters are not present, the storage parameters of the LOGFILE_DEFAULTS configuration keyword are used.

Creating a log file configuration keyword for each user allows you to position their sde log files onto separate devices by specifying the tablespace the log file tables and indexes are created in. Most installations of ArcSDE will function well using the LOGFILE_DEFAULTS storage parameters supplied with the installed dbtune.sde file. However, for applications making use of sde log files, such as ArcInfo, ArcEditor™, and ArcView®, it may help performance by spreading the log files across the file system. Typically log files are updated whenever a selection set exceeds 100 records.

If the imported DBTUNE file does not contain a LOGFILE_DEFAULTS configuration keyword, or if any of the log file storage parameters are missing, ArcSDE will insert the following records:

```
##LOGFILE_DEFAULTS

LD_INDEX_DATA_ID  "PCTFREE 10 INITRANS 2 STORAGE (FREELISTS 4 MINEXTENTS 1
                  PCTINCREASE 0) NOLOGGING"
LD_INDEX_ROWID    "PCTFREE 10 INITRANS 2 STORAGE (FREELISTS 4 MINEXTENTS 1
                  PCTINCREASE 0) NOLOGGING"
LD_STORAGE        "PCTFREE 10 PCTUSED 90 INITRANS 1 STORAGE (FREELISTS 4
                  MINEXTENTS 1 PCTINCREASE 0)"
LF_INDEXES        "PCTFREE 10 INITRANS 2 STORAGE (FREELISTS 4 MINEXTENTS 1
                  PCTINCREASE 0) NOLOGGING"
LF_STORAGE        "PCTFREE 10 PCTUSED 90 INITRANS 1 STORAGE (FREELISTS 4
                  MINEXTENTS 1 PCTINCREASE 0)"
UI_TEXT ""

END
```

The LD_STORAGE and LF_STORAGE parameters that control the storage of the SDE_LOGFILE_DATA and SDE_LOGFILES tables. By default these tables are created with Oracle logging turned on (the Oracle NOLOGGING parameter is absent from the configuration string of these parameters). If you are not using a customized application that stores persistent log files, you should add NOLOGGING to the LD_STORAGE and LF_STORAGE parameters. ESRI applications accessing ArcSDE data use temporary log files.

Network class composite configuration keywords

The composite keyword is a unique type of configuration keyword designed to accommodate the tables of the ArcGIS network class. The network table's size variation requires a configuration keyword that provides configuration storage 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 the size of the network tables, the network composite configuration keyword is subdivided into elements. A network composite configuration keyword has three elements: the parent element defines the general characteristic of the configuration 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 configuration keyword looks like any other configuration keyword. The description element is demarcated by the addition of the ::DESC suffix to the parent element's configuration keyword, and the network element is demarcated by the addition of the ::NETWORK suffix to the parent element's configuration keyword.

For example, if the parent element configuration keyword is ELECTRIC, the network composite configuration keyword would appear in a DBTUNE file as follows:

```
##ELECTRIC

COMMENT  This configuration keyword is dedicated to the electrical geometric
network class

UI_NETWORK_TEXT "The electrical geometrical network class configuration
keyword"

B_STORAGE     "TABLESPACE BUSINESS INITRANS 4 PCTFREE 10 PCTUSED 90 STORAGE
(INITIAL 500M NEXT 100M MINEXTENTS 1 MAXEXTENTS 200 PCTINCREASE 0)"

B_INDEX_ROWID "TABLESPACE BUSINESS_INDEX INITRANS 4 PCTFREE 10 STORAGE
(INITIAL 100M NEXT 50M MINEXTENTS 1 MAXEXTENTS 200 PCTINCREASE 0) NOLOGGING"

B_INDEX_SHAPE "TABLESPACE BUSINESS_INDEX INITRANS 4 PCTFREE 10 STORAGE
(INITIAL 100M NEXT 50M MINEXTENTS 1 MAXEXTENTS 200 PCTINCREASE 0) NOLOGGING"

B_INDEX_USER  "TABLESPACE BUSINESS_INDEX INITRANS 4 PCTFREE 10 STORAGE
(INITIAL 100M NEXT 50M MINEXTENTS 1 MAXEXTENTS 200 PCTINCREASE 0) NOLOGGING"

F_STORAGE     "TABLESPACE FEATURE INITRANS 4 PCTFREE 30 PCTUSED 70 STORAGE
(INITIAL 500M NEXT 100M MINEXTENTS 1 MAXEXTENTS 200 PCTINCREASE 0)"

F_INDEX_FID   "TABLESPACE FEATURE_INDEX INITRANS 4 PCTFREE 10 STORAGE
(INITIAL 100M NEXT 50M MINEXTENTS 1 MAXEXTENTS 200 PCTINCREASE 0) NOLOGGING"

F_INDEX_LEN   "TABLESPACE BUSINESS INITRANS 4 PCTFREE 10 STORAGE (INITIAL
100M NEXT 50M MINEXTENTS 1 MAXEXTENTS 200 PCTINCREASE 0) NOLOGGING"

F_INDEX_AREA  "TABLESPACE BUSINESS INITRANS 4 PCTFREE 10 STORAGE (INITIAL
100M NEXT 50M MINEXTENTS 1 MAXEXTENTS 200 PCTINCREASE 0) NOLOGGING"

S_STORAGE      "TABLESPACE BUSINESS INITRANS 4 PCTFREE 10 PCTUSED 90 STORAGE
(INITIAL 200M NEXT 20M MINEXTENTS 1 MAXEXTENTS 200 PCTINCREASE 0)"

S_INDEX_ALL    "TABLESPACE BUSINESS INITRANS 4 PCTFREE 10 STORAGE (INITIAL
200M NEXT 20M MINEXTENTS 1 MAXEXTENTS 200 PCTINCREASE 0) NOLOGGING"

S_INDEX_SP_FID  "TABLESPACE BUSINESS INITRANS 4 PCTFREE 10 STORAGE (INITIAL
50M NEXT 10M MINEXTENTS 1 MAXEXTENTS 200 PCTINCREASE 0) NOLOGGING"

A_STORAGE       "TABLESPACE BUSINESS INITRANS 4 PCTFREE 10 PCTUSED 90 STORAGE
(INITIAL 100M NEXT 50M MINEXTENTS 1 MAXEXTENTS 200 PCTINCREASE 0)"

A_INDEX_ROWID    "TABLESPACE BUSINESS INITRANS 4 PCTFREE 10 STORAGE (INITIAL
20M NEXT 10M MINEXTENTS 1 MAXEXTENTS 200 PCTINCREASE 0) NOLOGGING"

A_INDEX_SHAPE    "TABLESPACE BUSINESS INITRANS 4 PCTFREE 10 STORAGE (INITIAL
20M NEXT 10M MINEXTENTS 1 MAXEXTENTS 200 PCTINCREASE 0) NOLOGGING"

A_INDEX_USER     "TABLESPACE BUSINESS INITRANS 4 PCTFREE 10 STORAGE (INITIAL
20M NEXT 10M MINEXTENTS 1 MAXEXTENTS 200 PCTINCREASE 0) NOLOGGING"

A_INDEX_STATEID  "TABLESPACE BUSINESS INITRANS 4 PCTFREE 10 STORAGE (INITIAL
20M NEXT 10M MINEXTENTS 1 MAXEXTENTS 200 PCTINCREASE 0) NOLOGGING"

D_STORAGE        "TABLESPACE BUSINESS INITRANS 4 PCTFREE 10 PCTUSED 90
STORAGE (INITIAL 20M NEXT 10M MINEXTENTS 1 MAXEXTENTS 200 PCTINCREASE 0)"

D_INDEX_DELETED_AT   "TABLESPACE BUSINESS INITRANS 4 PCTFREE 10 STORAGE
(INITIAL 20M NEXT 10M MINEXTENTS 1 MAXEXTENTS 200 PCTINCREASE 0) NOLOGGING"

D_INDEX_STATE_ROWID   "TABLESPACE BUSINESS INITRANS 4 PCTFREE 10 STORAGE
(INITIAL 20M NEXT 10M MINEXTENTS 1 MAXEXTENTS 200 PCTINCREASE 0) NOLOGGING"

END

##ELECTRIC::DESC

B_STORAGE        "TABLESPACE BUSINESS INITRANS 4 PCTFREE 10 PCTUSED 90
STORAGE (INITIAL 500M NEXT 100M MINEXTENTS 1 MAXEXTENTS 200 PCTINCREASE 0)"

B_INDEX_ROWID    "TABLESPACE BUSINESS INITRANS 4 PCTFREE 10 STORAGE (INITIAL
100M NEXT 50M MINEXTENTS 1 MAXEXTENTS 200 PCTINCREASE 0) NOLOGGING"

B_INDEX_USER     "TABLESPACE BUSINESS INITRANS 4 PCTFREE 10 STORAGE (INITIAL
100M NEXT 50M MINEXTENTS 1 MAXEXTENTS 200 PCTINCREASE 0) NOLOGGING"
```

```
A_STORAGE         "TABLESPACE BUSINESS INITRANS 4 PCTFREE 10 PCTUSED 90
STORAGE (INITIAL 500M NEXT 100M MINEXTENTS 1 MAXEXTENTS 200 PCTINCREASE 0)"

A_INDEX_ROWID     "TABLESPACE BUSINESS INITRANS 4 PCTFREE 10 STORAGE (INITIAL
20M NEXT 10M MINEXTENTS 1 MAXEXTENTS 200 PCTINCREASE 0) NOLOGGING"

A_INDEX_SHAPE     "TABLESPACE BUSINESS INITRANS 4 PCTFREE 10 STORAGE (INITIAL
20M NEXT 10M MINEXTENTS 1 MAXEXTENTS 200 PCTINCREASE 0) NOLOGGING"

A_INDEX_USER      "TABLESPACE BUSINESS INITRANS 4 PCTFREE 10 STORAGE (INITIAL
20M NEXT 10M MINEXTENTS 1 MAXEXTENTS 200 PCTINCREASE 0) NOLOGGING"

A_INDEX_STATEID   "TABLESPACE BUSINESS INITRANS 4 PCTFREE 10 STORAGE (INITIAL
20M NEXT 10M MINEXTENTS 1 MAXEXTENTS 200 PCTINCREASE 0) NOLOGGING"

D_STORAGE          "TABLESPACE BUSINESS INITRANS 4 PCTFREE 10 PCTUSED 90
STORAGE (INITIAL 20M NEXT 10M MINEXTENTS 1 MAXEXTENTS 200 PCTINCREASE 0)"

D_INDEX_DELETED_AT    "TABLESPACE BUSINESS INITRANS 4 PCTFREE 10 STORAGE
(INITIAL 20M NEXT 10M MINEXTENTS 1 MAXEXTENTS 200 PCTINCREASE 0) NOLOGGING"

D_INDEX_STATE_ROWID   "TABLESPACE BUSINESS INITRANS 4 PCTFREE 10 STORAGE
(INITIAL 20M NEXT 10M MINEXTENTS 1 MAXEXTENTS 200 PCTINCREASE 0) NOLOGGING"

END

##ELECTRIC::NETWORK

B_STORAGE         "TABLESPACE BUSINESS INITRANS 4 PCTFREE 10 PCTUSED 90
STORAGE (INITIAL 1M NEXT 1M MINEXTENTS 1 MAXEXTENTS 200 PCTINCREASE 0)"

B_INDEX_ROWID     "TABLESPACE BUSINESS INITRANS 4 PCTFREE 10 STORAGE (INITIAL
1M NEXT 1M MINEXTENTS 1 MAXEXTENTS 200 PCTINCREASE 0) NOLOGGING"

B_INDEX_USER      "TABLESPACE BUSINESS INITRANS 4 PCTFREE 10 STORAGE (INITIAL
1M NEXT 1M MINEXTENTS 1 MAXEXTENTS 200 PCTINCREASE 0) NOLOGGING"

A_STORAGE         "TABLESPACE BUSINESS INITRANS 4 PCTFREE 10 PCTUSED 90
STORAGE (INITIAL 1M NEXT 1M MINEXTENTS 1 MAXEXTENTS 200 PCTINCREASE 0)"

A_INDEX_ROWID     "TABLESPACE BUSINESS INITRANS 4 PCTFREE 10 STORAGE (INITIAL
1M NEXT 1M MINEXTENTS 1 MAXEXTENTS 200 PCTINCREASE 0) NOLOGGING"

A_INDEX_SHAPE     "TABLESPACE BUSINESS INITRANS 4 PCTFREE 10 STORAGE (INITIAL
1M NEXT 1M MINEXTENTS 1 MAXEXTENTS 200 PCTINCREASE 0) NOLOGGING"

A_INDEX_USER      "TABLESPACE BUSINESS INITRANS 4 PCTFREE 10 STORAGE (INITIAL
1M NEXT 1M MINEXTENTS 1 MAXEXTENTS 200 PCTINCREASE 0) NOLOGGING"

A_INDEX_STATEID   "TABLESPACE BUSINESS INITRANS 4 PCTFREE 10 STORAGE (INITIAL
1M NEXT 1M MINEXTENTS 1 MAXEXTENTS 200 PCTINCREASE 0) NOLOGGING"

D_STORAGE          "TABLESPACE BUSINESS INITRANS 4 PCTFREE 10 PCTUSED 90
STORAGE (INITIAL 1M NEXT 1M MINEXTENTS 1 MAXEXTENTS 200 PCTINCREASE 0)"

D_INDEX_DELETED_AT    "TABLESPACE BUSINESS INITRANS 4 PCTFREE 10 STORAGE
(INITIAL 1M NEXT 1M MINEXTENTS 1 MAXEXTENTS 200 PCTINCREASE 0) NOLOGGING"

D_INDEX_STATE_ROWID   "TABLESPACE BUSINESS INITRANS 4 PCTFREE 10 STORAGE
(INITIAL 1M NEXT 1M MINEXTENTS 1 MAXEXTENTS 200 PCTINCREASE 0) NOLOGGING"

END
```

Following the import of the DBTUNE file, these records would be inserted into the DBTUNE table.

```
SQL> select keyword, parameter_name from DBTUNE;

KEYWORD            PARAMETER_NAME
-----------------  ----------------
ELECTRIC           COMMENT
ELECTRIC           UI_NETWORK_TEXT
ELECTRIC           B_STORAGE
ELECTRIC           B_INDEX_ROWID
ELECTRIC           B_INDEX_SHAPE
ELECTRIC           B_INDEX_USER
ELECTRIC           F_STORAGE
ELECTRIC           F_INDEX_FID
```

```
ELECTRIC            F_INDEX_LEN
ELECTRIC            F_INDEX_AREA
ELECTRIC            S_STORAGE
ELECTRIC            S_INDEX_ALL
ELECTRIC            S_INDEX_SP_FID
ELECTRIC            A_STORAGE
ELECTRIC            A_INDEX_ROWID
ELECTRIC            A_INDEX_SHAPE
ELECTRIC            A_INDEX_USER
ELECTRIC            A_INDEX_STATEID
ELECTRIC            D_STORAGE
ELECTRIC            D_INDEX_DELETED_AT
ELECTRIC            D_INDEX_STATE_ROWID
ELECTRIC::DESC      B_STORAGE
ELECTRIC::DESC      B_INDEX_ROWID
ELECTRIC::DESC      B_INDEX_USER
ELECTRIC::DESC      A_STORAGE
ELECTRIC::DESC      A_INDEX_ROWID
ELECTRIC::DESC      A_INDEX_STATEID
ELECTRIC::DESC      A_INDEX_USER
ELECTRIC::DESC      D_STORAGE
ELECTRIC::DESC      D_INDEX_DELETE_AT
ELECTRIC::DESC      D_INDEX_STATE_ROWID
ELECTRIC::NETWORK   B_STORAGE
ELECTRIC::NETWORK   B_INDEX_ROWID
ELECTRIC::NETWORK   B_INDEX_USER
ELECTRIC::NETWORK   A_STORAGE
ELECTRIC::NETWORK   A_INDEX_ROWID
ELECTRIC::NETWORK   A_INDEX_STATEID
ELECTRIC::NETWORK   A_INDEX_USER
ELECTRIC::NETWORK   D_STORAGE
ELECTRIC::NETWORK   D_INDEX_DELETE_AT
ELECTRIC::NETWORK   D_INDEX_STATE_ROWID
```

The network junctions feature class is created with the ELECTRIC configuration keyword storage parameters, the network descriptions table is created with the storage parameters of the ELECTRIC::DESC configuration keyword, and the remaining smaller network tables are created with the ELECTRIC::NETWORK configuration keyword.

The NETWORK_DEFAULTS configuration keyword

The NETWORK_DEFAULTS configuration keyword contains the default storage parameters for the ArcGIS network class. If the user does not select a network class composite configuration keyword from the ArcCatalog interface, the ArcGIS network is created with the storage parameters within the NETWORK_DEFAULTS configuration keyword.

Whenever a network class composite configuration keyword is selected, its storage parameters are used to create the feature class, table, and indexes of the network class. If a network composite configuration keyword is missing any storage parameters, ArcGIS substitutes the storage parameters of the DEFAULTS configuration keyword rather than the NETWORK_DEFAULTS configuration keyword. The storage parameters of the NETWORK_DEFAULTS configuration keyword are used when a network composite configuration keyword has not been specified.

If a NETWORK_DEFAULTS configuration keyword is not present in a DBTUNE file imported into the DBTUNE table, the following NETWORK_DEFAULTS configuration keyword is created.

```
##NETWORK_DEFAULTS

A_INDEX_ROWID   "PCTFREE 10 INITRANS 4 STORAGE (FREELISTS 4 PCTINCREASE 0)
                 NOLOGGING"
A_INDEX_SHAPE   "PCTFREE 10 INITRANS 4 STORAGE (FREELISTS 4 PCTINCREASE 0)
                 NOLOGGING"
A_INDEX_STATEID "PCTFREE 10 INITRANS 4 STORAGE (FREELISTS 4 PCTINCREASE 0)
```

```
                NOLOGGING"
A_INDEX_USER    "PCTFREE 10 INITRANS 4 STORAGE (FREELISTS 4 PCTINCREASE 0)
                NOLOGGING"
A_STORAGE       "PCTFREE 10 PCTUSED 90 INITRANS 4 STORAGE (FREELISTS 4
                PCTINCREASE 0)"
B_INDEX_ROWID   "PCTFREE 10 INITRANS 4 STORAGE (FREELISTS 4 PCTINCREASE 0)
                NOLOGGING"
B_INDEX_SHAPE   "PCTFREE 10 INITRANS 4 STORAGE (FREELISTS 4 PCTINCREASE 0)
                NOLOGGING"
B_INDEX_USER    "PCTFREE 10 INITRANS 4 STORAGE (FREELISTS 4 PCTINCREASE 0)
                NOLOGGING"
B_STORAGE       "PCTFREE 10 PCTUSED 90 INITRANS 4 STORAGE (FREELISTS 4
                PCTINCREASE 0)"
COMMENT "The base system initialization parameters for NETWORK_DEFAULTS"
D_INDEX_DELETED_AT  "PCTFREE 10 INITRANS 4 STORAGE (FREELISTS 4 PCTINCREASE
0)
                    NOLOGGING"
D_INDEX_STATE_ROWID "PCTFREE 10 INITRANS 4 STORAGE (FREELISTS 4 PCTINCREASE
0)
                    NOLOGGING"
D_STORAGE       "PCTFREE 10 PCTUSED 90 INITRANS 4 STORAGE (FREELISTS
                PCTINCREASE 0)"
F_INDEX_AREA    "PCTFREE 10 INITRANS 4 STORAGE (FREELISTS 4 MINEXTENTS 1
                PCTINCREASE 0) NOLOGGING"
F_INDEX_FID     "PCTFREE 10 INITRANS 4 STORAGE (FREELISTS 4 MINEXTENTS 1
                PCTINCREASE 0) NOLOGGING"
F_INDEX_LEN     "PCTFREE 10 INITRANS 4 STORAGE (FREELISTS 4 MINEXTENTS 1
                PCTINCREASE 0) NOLOGGING"
F_STORAGE       "PCTFREE 30 PCTUSED 70 INITRANS 4 STORAGE (FREELISTS 4
                MINEXTENTS 1 PCTINCREASE 0)"
S_INDEX_ALL     "PCTFREE 10 INITRANS 4 STORAGE (FREELISTS 4 MINEXTENTS 1
                PCTINCREASE 0) NOLOGGING"
S_INDEX_SP_FID  "PCTFREE 10 INITRANS 4 STORAGE (FREELISTS 4 MINEXTENTS 1
                PCTINCREASE 0) NOLOGGING"
S_STORAGE       "PCTFREE 10 PCTUSED 90 INITRANS 4 STORAGE (FREELISTS 4
                MINEXTENTS 1 PCTINCREASE 0)"
UI_NETWORK_TEXT "The network default configuration"

END

##NETWORK_DEFAULTS::DESC

A_INDEX_ROWID   "PCTFREE 10 INITRANS 4 STORAGE (FREELISTS 4 PCTINCREASE 0)
                NOLOGGING"
A_INDEX_SHAPE   "PCTFREE 10 INITRANS 4 STORAGE (FREELISTS 4 PCTINCREASE 0)
                NOLOGGING"
A_INDEX_STATEID "PCTFREE 10 INITRANS 4 STORAGE (FREELISTS 4 PCTINCREASE 0)
                NOLOGGING"
A_INDEX_USER    "PCTFREE 10 INITRANS 4 STORAGE (FREELISTS 4 PCTINCREASE 0)
                NOLOGGING"
A_STORAGE       "PCTFREE 10 PCTUSED 90 INITRANS 4 STORAGE (FREELISTS 4
                PCTINCREASE 0)"
B_INDEX_ROWID   "PCTFREE 10 INITRANS 4 STORAGE (FREELISTS 4 PCTINCREASE 0)
                NOLOGGING"
B_INDEX_SHAPE   "PCTFREE 10 INITRANS 4 STORAGE (FREELISTS 4 PCTINCREASE 0)
                NOLOGGING"
B_INDEX_USER    "PCTFREE 10 INITRANS 4 STORAGE (FREELISTS 4 PCTINCREASE 0)
                NOLOGGING"
B_STORAGE       "PCTFREE 10 PCTUSED 90 INITRANS 4 STORAGE (FREELISTS 4
                PCTINCREASE 0)"
D_INDEX_DELETED_AT  "PCTFREE 10 INITRANS 4 STORAGE (FREELISTS 4 PCTINCREASE
0)
                    NOLOGGING"
D_INDEX_STATE_ROWID "PCTFREE 10 INITRANS 4 STORAGE (FREELISTS 4 PCTINCREASE
0)
                    NOLOGGING"
D_STORAGE       "PCTFREE 10 PCTUSED 90 INITRANS 4 STORAGE (FREELISTS 4
                PCTINCREASE 0)"

END

##NETWORK_DEFAULTS::NETWORK

A_INDEX_ROWID   "PCTFREE 10 INITRANS 4 STORAGE (FREELISTS 4 PCTINCREASE 0)
                NOLOGGING"
A_INDEX_SHAPE   "PCTFREE 10 INITRANS 4 STORAGE (FREELISTS 4 PCTINCREASE 0)
                NOLOGGING"
A_INDEX_STATEID "PCTFREE 10 INITRANS 4 STORAGE (FREELISTS 4 PCTINCREASE 0)
```

```
                 NOLOGGING"
A_INDEX_USER     "PCTFREE 10 INITRANS 4 STORAGE (FREELISTS 4 PCTINCREASE 0)
                 NOLOGGING"
A_STORAGE        "PCTFREE 10 PCTUSED 90 INITRANS 4 STORAGE (FREELISTS 4
                 PCTINCREASE 0)"
B_INDEX_ROWID    "PCTFREE 10 INITRANS 4 STORAGE (FREELISTS 4 PCTINCREASE 0)
                 NOLOGGING"
B_INDEX_SHAPE    "PCTFREE 10 INITRANS 4 STORAGE (FREELISTS 4 PCTINCREASE 0)
                 NOLOGGING"
B_INDEX_USER     "PCTFREE 10 INITRANS 4 STORAGE (FREELISTS 4 PCTINCREASE 0)
                 NOLOGGING"
B_STORAGE        "PCTFREE 10 PCTUSED 90 INITRANS 4 STORAGE (FREELISTS 4
                 PCTINCREASE 0)"
D_INDEX_DELETED_AT  "PCTFREE 10 INITRANS 4 STORAGE (FREELISTS 4 PCTINCREASE
0)
                    NOLOGGING"
D_INDEX_STATE_ROWID "PCTFREE 10 INITRANS 4 STORAGE (FREELISTS 4 PCTINCREASE
0)
                    NOLOGGING"
D_STORAGE        "PCTFREE 10 PCTUSED 90 INITRANS 4 STORAGE (FREELISTS 4
                 PCTINCREASE 0)"

END
```

ArcSDE storage parameters for Oracle Spatial

Several storage parameters exist to determine the storage of a feature class created as an Oracle Spatial SDO_GEOMETRY type.

The SDO_SRID identifies the Oracle Spatial coordinate reference description. Consult the *Oracle Spatial Users Guide and Reference* for a list of supported coordinate references.

ArcSDE provides storage parameters to define fixed values for the Oracle Spatial dimension information. If you do not define these storage parameters, Oracle Spatial calculates the dimensions from the data. Oracle Spatial allows you to define up to four dimensions. There are three basic parameters that are repeated for each dimension, delineated by the number of the dimension that is appended to the storage parameter's name.

SDO_DIMNAME_<n> stores the dimension’s name.

SDO_LB_<n> stores the dimension’s lower boundary.

SDO_UB_<n> stores the dimension’s upper boundary.

SDO_TOLERANCE_<n> stores the dimension’s precision.

Oracle Spatial provides three different spatial index methods: RTREE, FIXED, and HYBRID. The default is RTREE. The SDO_INDEX storage parameter lets you set the spatial index method to one of the methods offered by Oracle Spatial.

The SDO_INDEX_SHAPE storage parameter stores the spatial index’s storage parameters. The parameters are a quoted string of a list of *parameter = value* elements that are passed to the PARAMETERS clause of the Oracle user-defined CREATE INDEX statement.

The SDO_COMMIT_INTERVAL, SDO_LEVEL, SDO_NUMTILES, SDO_MAXLEVEL, and SDO_ORDCNT storage parameters all affect the configuration of the spatial index. These are actually Oracle Spatial parameters that have been documented in the *Oracle Spatial Users Guide and Reference.* For more information on how to set these parameters, see that document. You should be aware that in some cases Oracle will calculate the values

for you based on the values of the spatial data being indexed. It is recommended that you use the Oracle defaults before setting these values yourself.

The ArcSDE SDO_VERIFY storage parameter determines whether the geometry data fetched from Oracle Spatial feature class should be examined and, if necessary, corrected.

Listed below is an ArcSDE DBTUNE configuration with storage parameters set to store a feature class created with an Oracle Spatial column.

```
##ORACLE_SPATIAL_FC

GEOMETRY_STORAGE            SDO_GEOMETRY

B_STORAGE                   "TABLESPACE BUSINESS PCTFREE 10 PCTUSED 90
                            INITRANS 4 STORAGE(FREELISTS 4 INITIAL 512000 NEXT
                            512000 MINEXTENTS 1 MAXEXTENTS 100 PCTINCREASE 0)"
B_INDEX_USER                "PCTFREE 10 INITRANS 4 STORAGE (FREELISTS 4
                            INITIAL 512000 NEXT 512000 MINEXTENTS 1 MAXEXTENTS
                            100 PCTINCREASE 0) NOLOGGING"
B_INDEX_ROWID               "PCTFREE 10 INITRANS 4 STORAGE (FREELISTS 4
                            INITIAL 512000 NEXT 512000 MINEXTENTS 1 MAXEXTENTS
                            100 PCTINCREASE 0) NOLOGGING"

A_STORAGE                   "PCTFREE 10 PCTUSED 90 INITRANS 4 STORAGE
                            (FREELISTS 4 INITIAL 512000 NEXT 512000 MINEXTENTS
                            1 MAXEXTENTS 100 PCTINCREASE 0)"
A_INDEX_USER                "PCTFREE 10 PCTUSED 90 INITRANS 4 STORAGE
                            (FREELISTS 4 INITIAL 512000 NEXT 512000 MINEXTENTS
                            1 MAXEXTENTS 100 PCTINCREASE 0)"
A_INDEX_ROWID               "PCTFREE 10 PCTUSED 90 INITRANS 4 STORAGE
                            (FREELISTS 4 INITIAL 512000 NEXT 512000 MINEXTENTS
                            1 MAXEXTENTS 100 PCTINCREASE 0)"
A_INDEX_STATEID             "PCTFREE 10 PCTUSED 90 INITRANS 4 STORAGE
                            (FREELISTS 4 INITIAL 512000 NEXT 512000 MINEXTENTS
                            1 MAXEXTENTS 100 PCTINCREASE 0)"

D_STORAGE                   "PCTFREE 10 PCTUSED 90 INITRANS 4 STORAGE
                            (FREELISTS 4 INITIAL 512000 NEXT 512000 MINEXTENTS
                            1 MAXEXTENTS 100 PCTINCREASE 0)"
D_INDEX_DELETED_AT          "PCTFREE 10 PCTUSED 90 INITRANS 4 STORAGE
                            (FREELISTS 4 INITIAL 512000 NEXT 512000 MINEXTENTS
                            1 MAXEXTENTS 100 PCTINCREASE 0)"
D_INDEX_STATE_ROWID         "PCTFREE 10 PCTUSED 90 INITRANS 4 STORAGE
                            (FREELISTS 4 INITIAL 512000 NEXT 512000 MINEXTENTS
                            1 MAXEXTENTS 100 PCTINCREASE 0)"

SDO_COMMIT_INTERVAL         100
SDO_INDEX_SHAPE             "tablespace = spatial_index,initial = 409600, next
                            = 409600, minextents = 1, maxextents = 1000,
                            pctincrease = 0, btree_initial = 12384, btree_next
                            = 4096, btree_pctincrease = 0"
SDO_LAYER_GTYPE             "POINT"
SDO_LEVEL                   30
SDO_ORDCNT                  2
SDO_VERIFY                  TRUE

SDO_DIMNAME_1               LONGITUDE1
SDO_LB_1                    -160.0
SDO_UB_1                     20.0
SDO_TOLERANCE_1             0.000050

SDO_DIMNAME_2               LATITUDE2
SDO_LB_2                    -68.0
SDO_UB_2                     60.0
SDO_TOLERANCE_2             0.000050

SDO_DIMNAME_3               ZVals
SDO_LB_3                    0.0
SDO_UB_3                    15000.0
SDO_TOLERANCE_3             0.000050

SDO_DIMNAME_4               MVals
SDO_LB_4                    0.0
SDO_UB_4                    5000.0
SDO_TOLERANCE_4             0.000050
```

```
END
```

Oracle default parameters

By default, Oracle stores tables and indexes in the user's default tablespace using the tablespace's default storage parameters. Determine a user's default tablespace by querying the DEFAULT_TABLESPACE field of the USER_USERS system table when connected as that user. As the Oracle DBA, query the DEFAULT_TABLESPACE field of the DBA_USERS table using a where clause to specify the user.

```
SQL> connect <user>/<password>
SQL> select default_tablespace from user_users;
```

or

```
SQL> connect system/<password>
SQL> select default_tablespace from dba_users where username = <user>;
```

Obtain a list of tablespace default storage parameters for a tablespace by querying USER_TABLESPACES.

```
SQL> connect <user>/<password>
SQL> select * from user_tablespaces where tablespace_name = <tablespace>;
```

Converting previous versions of SDE storage parameters into the DBTUNE table

For older versions of the Spatial Database Engine™ (SDE®), the DBTUNE storage parameters were maintained in the dbtune.sde file. The storage parameters of these previous versions were mapped directly to each Oracle storage parameters. For example, the ArcSDE 8.0.2 F_TBLSP storage parameter holds the name of the feature table's tablespace.

Beginning at ArcSDE 8.1, storage parameters hold entire configuration strings of the table or index they represent. For example, the F_STORAGE storage parameter holds the configuration string of the feature table. Any legal Oracle storage parameter listed to the right of the columns clause of the Oracle CREATE TABLE statement can be listed in the F_STORAGE storage parameter. Therefore, the F_STORAGE storage parameter incorporates all of the ArcSDE 8.0.2 feature table storage parameters.

If you are upgrading to ArcSDE 8.1 or higher from a version of ArcSDE prior to ArcSDE 8.1, the conversion of the dbtune files storage parameters occurs automatically when the sdesetupora* utility reads the storage parameters from the previous version's dbtune.sde file. The import operation of the sdedbtune command will convert a DBTUNE file into current ArcSDE storage parameters before it writes them to the DBTUNE table. To see the results you can either use SQL*Plus to list the storage parameters of the DBTUNE table or write the storage parameters of the DBTUNE table to another file using the export operation of the sdedbtune command.

The following table lists the conversion of ArcSDE 8.0.2 storage parameters to ArcSDE 8.1 storage parameters.

The ArcSDE 8.0.2 business table and index parameter prefix is "A_". The ArcSDE 8.1 business table and index parameter prefix is "B_". The ArcSDE 8.0.2 business table storage parameters are converted to the single ArcSDE 8.1 B_STORAGE storage parameter. The B_STORAGE parameter holds the entire business table's configuration string.

ArcSDE 8.0.2 storage parameters

```
A_TBLSP       ROADS

A_INIT        10M
A_NEXT        5M
A_MINX        1
A_MAXX        200
A_PCTI        0
A_ITRANS      5
A_MAXTRS      255
A_PCTFREE     10
A_PCTUSD      90
```

ArcSDE 8.1 storage parameters

```
B_STORAGE   "TABLESPACE ROADS
              STORAGE (FREELISTS 4
                       INITIAL 10M
                       NEXT 5M
                       MINEXTENTS 1
                       MAXEXTENTS 200
                      PCTINCREASE 0)
            INITRANS 5
            MAXTRANS 255
            PCTFREE 10
            PCTUSED 90"
```

The ArcSDE 8.0.2 business table index storage parameters are converted to ArcSDE 8.1 storage parameter configuration strings. The example below illustrates how the ArcSDE 8.0.2 storage parameters are converted into the ArcSDE 8.1 spatial column index storage parameter B_INDEX_SHAPE. The other ArcSDE 8.1 business table index storage parameters B_INDEX_ROWID and B_INDEX_USER are also constructed this way.

ArcSDE 8.0.2 storage parameters

```
INDEX_TABLESPACE  ROADS_IX

A_IX1_INIT        10M
A_IX1_NEXT        5M
A_MINX            1
A_MAXX            200
A_PCTI            0
A_ITRANS          5
A_MAXTRS          255
A_PCTFREE         10
```

ArcSDE 8.1 storage parameters

```
B_INDEX_SHAPE   "TABLESPACE ROADS_IX
                  STORAGE (FREELISTS 4
                           INITIAL 10M
                           NEXT 5M
                           MINEXTENTS 1
                           MAXEXTENTS 200
                           PCTINCREASE 0)
                  INITRANS 5
                  MAXTRANS 255
                  PCTFREE 10"
```

The ArcSDE 8.0.2 feature table storage parameters are converted to the ArcSDE 8.1 F_STORAGE storage parameter. The F_STORAGE parameter holds the entire feature table's configuration string.

ArcSDE 8.0.2 storage parameters

```
F_TBLSP      ROADS_F

F_INIT       10M
F_NEXT       5M
F_MINX       1
F_MAXX       200
F_PCTI       0
F_ITRANS     5
F_MAXTRS     255
F_PCTFREE    10
F_PCTUSD     90
```

ArcSDE 8.1 storage parameters

```
F_STORAGE  "TABLESPACE ROADS_F
            STORAGE (FREELISTS 4
                     INITIAL 10M
                     NEXT 5M
                     MINEXTENTS 1
                     MAXEXTENTS 200
                    PCTINCREASE 0)
           INITRANS 5
           MAXTRANS 255
           PCTFREE 10
           PCTUSED 90"
```

The ArcSDE 8.0.2 feature table index storage parameters are converted to ArcSDE 8.1 storage parameter configuration strings. The example below illustrates how the ArcSDE 8.0.2 storage parameters are converted into the ArcSDE 8.1fid column index storage parameter F_INDEX_FID. The other ArcSDE 8.1 feature table index storage parameters F_INDEX_AREA and F_INDEX_LEN are also constructed this way.

ArcSDE 8.0.2 storage parameters

```
INDEX_TABLESPACE  ROADS_F_IX

F_IX1_INIT       10M
F_IX1_NEXT       5M
F_MINX           1
F_MAXX           200
F_PCTI           0
F_ITRANS         5
F_MAXTRS         255
F_PCTFREE        10
```

ArcSDE 8.1 storage parameters

```
F_INDEX_FID      "TABLESPACE ROADS_F_IX
                  STORAGE (FREELISTS 4
                           INITIAL 10M
                           NEXT 5M
                           MINEXTENTS 1
                           MAXEXTENTS 200
                           PCTINCREASE 0)
                  INITRANS 5
                  MAXTRANS 255
                  PCTFREE 10"
```

The ArcSDE 8.0.2 spatial index table storage parameters are converted to the ArcSDE 8.1 S_STORAGE storage parameter. The S_STORAGE parameter holds the entire spatial index table's configuration string.

ArcSDE 8.0.2 storage parameters

```
S_TBLSP       ROADS_S

S_INIT        10M
S_NEXT        5M
S_MINX        1
S_MAXX        200
S_PCTI        0
S_ITRANS      5
S_MAXTRS      255
S_PCTFREE     10
S_PCTUSD      90
```

ArcSDE 8.1 storage parameters

```
S_STORAGE  "TABLESPACE ROADS_S
            STORAGE (FREELISTS 4
                     INITIAL 10M
                     NEXT 5M
                     MINEXTENTS 1
                     MAXEXTENTS 200
                    PCTINCREASE 0)
           INITRANS 5
           MAXTRANS 255
           PCTFREE 10
           PCTUSED 90"
```

The ArcSDE 8.0.2 spatial index table storage parameters are converted to ArcSDE 8.1 storage parameter configuration strings. The example below illustrates how the ArcSDE 8.0.2 index storage parameters are converted into the ArcSDE 8.1 storage parameter S_INDEX_ALL. The S_INDEX_SP_FID storage parameter is converted the same way except ArcSDE 8.0.2 storage parameters S_IX2_INIT and S_IX2_NEXT are used.

ArcSDE 8.0.2 storage parameters

```
INDEX_TABLESPACE ROADS_S_IX

S_IX1_INIT        10M
S_IX1_NEXT        5M
S_MINX            1
S_MAXX            200
S_PCTI            0
S_ITRANS          5
S_MAXTRS          255
S_PCTFREE         10
```

ArcSDE 8.1 storage parameters

```
S_INDEX_ALL       "TABLESPACE ROADS_S_IX
                    STORAGE (FREELISTS 4
                             INITIAL 10M
                             NEXT 5M
                             MINEXTENTS 1
                             MAXEXTENTS 200
                             PCTINCREASE 0)
                    INITRANS 5
                    MAXTRANS 255
                    PCTFREE 10"
```

The complete list of ArcSDE 8.1 storage parameters

Parameter Name	Value	Parameter Description	Default Value
STATES_LINEAGES_TABLE	<string>	State_lineages table	B_STORAGE
STATES_TABLE	<string>	States table	B_STORAGE
STATES_INDEX	<string>	States indexes	B_INDEX_USER
MVTABLES_MODIFIED_TABLE	<string>	Mvtables_modified table	B_STORAGE
MVTABLES_MODIFIED_INDEX	<string>	Mvtables_modified index	B_INDEX_USER
VERSIONS_TABLE	<string>	Versions table	B_STORAGE
VERSIONS_INDEX	<string>	Version index	B_INDEX_USER
COMPRESS_ROLLBACK_SEGMENT	<string>	Version compression rollback segment	Oracle defaults
B_STORAGE	<string>	Business table	Oracle defaults
B_INDEX_ROWID	<string>	Business table object ID column index	Oracle defaults
B_INDEX_SHAPE	<string>	Business table spatial column index	Oracle defaults
B_INDEX_USER	<string>	Business table user index(s)	Oracle defaults
F_STORAGE	<string>	Feature table	Oracle defaults
F_INDEX_FID	<string>	Feature table fid column index	Oracle defaults
F_INDEX_AREA	<string>	Feature table area column index	Oracle defaults
F_INDEX_LEN	<string>	Feature table length column index	Oracle defaults
S_STORAGE	<string>	Spatial index table	Oracle defaults
S_INDEX_ALL	<string>	Spatial index table first index	Oracle defaults
S_INDEX_SP_FID	<string>	Spatial index table second index	Oracle defaults
A_STORAGE	<string>	Adds table	Oracle defaults

Parameter Name	Value	Parameter Description	Default Value
A_INDEX_ROWID	<string>	Adds table object ID column index	Oracle defaults
A_INDEX_SHAPE	<string>	Adds table spatial column index	Oracle defaults
A_INDEX_STATEID	<string>	Adds table sde_state_id column index	Oracle defaults
A_INDEX_USER	<string>	Adds table index	Oracle defaults
D_STORAGE	<string>	Deletes table	Oracle defaults
D_INDEX_ STATE_ROWID	<string>	Deletes table sde_states_id and sde_deletes_row_id column index	Oracle defaults
D_INDEX_DELETED_AT	<string>	Deletes table sde_deleted_at column index	Oracle defaults
LF_STORAGE	<string>	Sde log files table	Oracle defaults
LF_INDEXES	<string>	Sde log file table column indexes	Oracle defaults
LD_STORAGE	<string>	Sde log file data table	Oracle defaults
LD_INDEX_DATA_ID	<string>	Sde log file data table	Oracle defaults
LD_INDEX_ROWID	<string>	SDE log file data table SDE rowid column index	Oracle defaults
RAS_STORAGE	<string>	Raster RAS table	Oracle defaults
RAS_INDEX_ID	<string>	Raster RAS table RID index	Oracle defaults
BND_STORAGE	<string>	Raster BND table	Oracle defaults
BND_INDEX_COMPOSITE	<string>	Raster BND table composite column index	Oracle defaults
BND_INDEX_ID	<string>	Raster BND table RID column index	Oracle defaults
AUX_STORAGE	<string>	Raster AUX table	Oracle defaults
AUX_INDEX_COMPOSITE	<string>	Raster AUX table composite column index	Oracle defaults
BLK_STORAGE	<string>	Raster BLK table	Oracle defaults

Parameter Name	Value	Parameter Description	Default Value
BLK_INDEX_COMPOSITE	<string>	Raster BLK table composite column index	Oracle defaults
ATTRIBUTE_BINARY	<string>	Set this storage parameter to longraw or blob	longraw
GEOMETRY_STORAGE	<string>	Set this storage parameter to sdebinary, sdelob, or sdo_geometry	SDEBINARY
SDO_COMMIT_INTERVAL	integer	Specifies the Oracle Spatial Geometry Types index commit interval	Use Oracle Default
SDO_DIMNAME_1	String	The name of the first dimension for Oracle Spatial Geometry Types only	X
SDO_DIMNAME_2	String	The name of the second dimension for Oracle Spatial Geometry Types only	Y
SDO_DIMNAME_3	string	The name of the third dimension for Oracle Spatial Geometry Types only	Z
SDO_DIMNAME_4	string	The name of the fourth dimension for Oracle Spatial Geometry Types only	M
SDO_INDEX	string	The Oracle Spatial Geometry Types index type (FIXED, HYBRID, RTREE)	Oracle Defaults
SDO_INDEX_SHAPE	string	The Oracle Spatial Geometry types spatial index storage parameters	Oracle Default
SDO_LAYER_GTYPE	POINT	Specifies that the parameter "LAYER_GTYPE = POINT" should be used when creating the spatial index	Set ONLY for single part POINT data when using quadtree indexes
SDO_LB_1	real number	Lower dimension boundary for Oracle Spatial Geometry Type	Based on extent of data loaded
SDO_LB_2	real number	Lower dimension boundary for Oracle Spatial Geometry Type	Based on extent of data loaded

Parameter Name	Value	Parameter Description	Default Value
SDO_LB_3	real number	Lower dimension boundary for Oracle Spatial Geometry Types only	Based on extent of data loaded
SDO_LB_4	real number	Lower dimension boundary for Oracle Spatial Geometry Types only	Based on extent of data loaded
SDO_LEVEL	integer	Specifies the desired fixed-size tiling level for Oracle Spatial Geometry Types only	Computed by SDO_TUNE.ESTIMATE_TILING_LEVEL
SDO_MAXLEVEL	integer	Specifies the maximum tiling level for Oracle Spatial Geometry Types only	Computed by SDO_TUNE.ESTIMATE_TILING_LEVEL
SDO_NUMTILES	integer	Specifies the number of variable-sized tiles to be used in tessellating an object for Oracle Spatial Geometry Type	Computed by SDO_TUNE.ESTIMATE_TILING_LEVEL
SDO_ORDCNT	integer	The number of ordinates in a row (for Oracle Spatial Schema Only)	16 (2 for POINT data)
SDO_SRID	integer	Oracle Spatial coordinate reference identifier assigned to the SDO_GEOMETRY column	NULL
SDO_TOLERANCE_1	real number	The precision of the dimension for Oracle Spatial Geometry Type	.0005
SDO_TOLERANCE_2	real number	The precision of the dimension for Oracle Spatial Geometry Type Schema	.0005
SDO_TOLERANCE_3	real number	The precision of the dimension for Oracle Spatial Geometry Types only	.0005
SDO_TOLERANCE_4	real number	The precision of the dimension for Oracle Spatial Geometry Types only	.0005
SDO_UB_1	real number	Upper dimension boundary for Oracle Spatial Geometry Type	Based on extent of data loaded

Parameter Name	Value	Parameter Description	Default Value
SDO_UB_2	real number	Upper dimension boundary for Oracle Spatial Geometry Type	Based on extent of data loaded
SDO_UB_3	real number	Upper dimension boundary for Oracle Spatial Geometry Types only	Based on extent of data loaded
SDO_UB_4	real number	Upper dimension boundary for Oracle Spatial Geometry Types only	Based on extent of data loaded
SDO_VERIFY	boolean	If TRUE, ArcSDE will verify and correct as necessary all geometries as they are accessed from the Oracle Spatial Geometry Type	FALSE

CHAPTER 4

Managing tables, feature classes, and raster columns

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 class creation and loading process.

Data creation

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

1. ArcSDE administration commands located in the bin directory of SDEHOME:

   - sdelayer—creates and manages feature classes.
   - sdetable—creates and manages tables.
   - sdeimport—takes an existing sdeexport file and loads the data into a feature class.
   - shp2sde—loads an ESRI shapefile into a feature class.
   - cov2sde—loads a coverage, Map LIBRARIAN layer, or an ArcStorm™ layer into a feature class.
   - tbl2sde—loads an attribute-only dBASE® or INFO™ file into a table.
   - sdegroup—a specialty feature class creation command that combines the features of an existing feature class into single multipart features and stores them in a new feature class for background display. The generated feature class is 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.
   - sderaster—creates, inserts, modifies, imports and manages raster data stored in an ArcSDE database.

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

Other applications include:

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

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

4. ArcView 3.2—use the Database Access extension.

5. MapObjects®—custom Component Object Model (COM) applications can be built to create and populate databases.

6. ArcSDE CAD Client extension—for AutoCAD® and MicroStation® users.

7. Other third party applications built with either the 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 Desktop. For details on how to use ArcCatalog or ArcToolbox (another desktop data loading tool), please refer to the ArcGIS books:

- *Using ArcCatalog*
- *Using ArcToolbox*
- *Building a Geodatabase*

Creating and populating a feature class

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

1. Create the business table.
2. Record the business table and the spatial column in the ArcSDE LAYERS and GEOMETRY_COLUMNS system tables, thus adding a new feature class to the database.
3. Switch the feature class to load_only_io mode (optional step to improve bulk data loading performance. It is OK to leave 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 privileges 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 (requiring considerably 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 the ArcSDE sdetable command. The sdetable command allows you to include a dbtune configuration keyword containing 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 Oracle SQL*Plus interface.

```
create table roads
  (road_id    integer,
   name       varchar(32),
   shape      integer)
   tablespace beetle_data
   storage (initial 16K next 8K);
```

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 record the spatial column in the ArcSDE LAYERS and GEOMETRY_COLUMNS system tables and thus add a new feature class to the database.

Adding a feature class

After creating a business table, you must add an entry for the spatial column in the ArcSDE LAYERS system tables before the ArcSDE server can reference it. Use the sdelayer command with the '-o add' operation to add the new feature class.

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 spatial column are combined to form a unique feature class reference. To understand the purpose of 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 ArcSDE compressed binary format, the feature and spatial index tables are created. 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 record the feature class in 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 absence 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.

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. Don't use the init operation on feature classes that already contain data unless you want to remove the existing data. 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) option.

```
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
```

Versioning your data

Optionally, you may enable your feature class as multiversioned. Versioning is a process that allows multiple representations of your data to exist without requiring duplication or copies of the data. 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.

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

Granting privileges 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 privileges to other users. The owner can use the sdelayer command to grant privileges. 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 initial privilege.

Creating and loading feature classes from existing data

We have reviewed the "from scratch" method of creating a schema and then loading it. 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.

Each of the ArcSDE administration commands, shp2sde, cov2sde, and sdeimport, includes a '-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

shp2sde 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. As part of the create operation, you can specify which spatial storage format you wish to adopt for the data storage by including a '–k' option that references to a configuration keyword containing a GEOMETRY_STORAGE parameter.

In this example, Oracle Spatial geometry type is selected as the storage format by including the configuration keyword 'GEOMETRY_TYPE'. The keyword GEOMETRY_TYPE, defined in the DBTUNE table, also sets a number of additional parameters for this storage type, which will be used to create the Oracle Spatial index. See Appendix D, 'Oracle Spatial Geometry Type', for additional information on Oracle Spatial and specifics about how it can be configured.

```
shp2sde -o create -f rdshp -l roads,shape -k GEOMETRY_TYPE -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 grant privileges on the feature class to other users.

Appending data to an existing feature class

A common requirement for data management is to be able to append data to existing feature classes. 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 been defined, the metadata exists and is not altered by the append operation.

In the shp2sde example below, a previously created 'roads' feature class appends features from a shapefile, 'rdshp2'. All existing features, loaded from the 'rdshp' shapefile, remain intact, and ArcSDE updates the feature class 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 sdetable update_dbms_stats operation updates the table and index statistics required by the Oracle cost-based optimizer. Without the statistics the optimizer may not be able to select the best execution plan when you query the table. For more information on updating statistics, see Chapter 2, 'Essential Oracle configuring and tuning'.

Creating and populating raster columns

Raster columns are created from 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.*

To estimate the size of your raster data, refer to Appendix A, 'Estimating the size of your tables and indexes'.

To understand how ArcSDE stores rasters in Oracle, refer to Appendix B, 'Storing raster data'.

Creating views

There are times when a DBMS view is required in your database schema. ArcSDE provides the sdetable create_view operation to accommodate this need. The view creation is much like any other Oracle view creation. If you want to create a view using a layer and you want the resulting view to appear as a feature class to client applications, include the feature class's spatial column in the view definition. As with the other ArcSDE commands, see the ArcSDE developer help for more information.

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 DBMSs

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 a DBMS table

Schema modification

There will be occasions when it is necessary to modify the schema of some tables. You may need to add or remove columns from a table. The ArcSDE command to do this is sdetable with the –o alter option. 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.

Using the ArcGIS Desktop 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 ArcGIS Desktop, you may have to use ArcCatalog to create feature datasets and feature classes within those feature datasets to use specific ArcGIS Desktop functionality. For that reason, we provide a glimpse of how to use

ArcToolbox and ArcCatalog to load data. Please refer to the ArcInfo documentation on ArcCatalog, ArcToolbox, and the geodatabase for a full discussion of these tools.

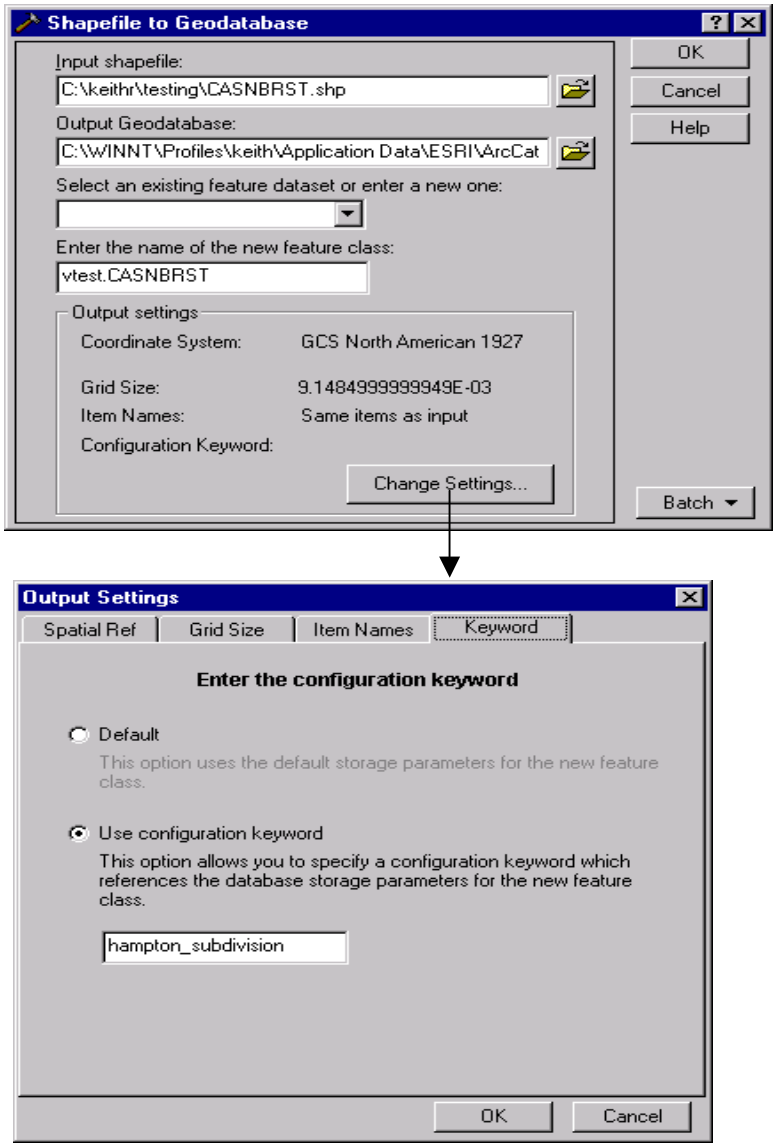
Loading data

You can convert ESRI shapefiles, coverages, Map LIBRARIAN layers, and ArcStorm layers into geodatabase feature classes with the 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 with the GEOMETRY_STORAGE storage parameter, the user can choose one of the three Oracle spatial storage methods supported by ArcSDE.

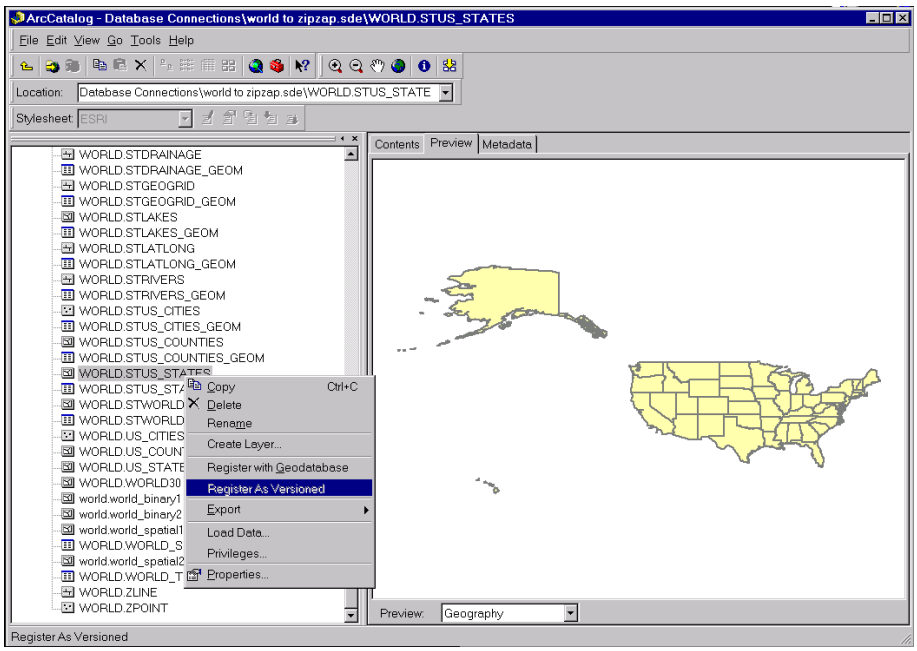
In the ArcToolbox Shapefile to Geodatabase wizard, you can see that a configuration keyword has been specified for the loading of the hampton_streets shapefile into the geodatabase. Since the geodatabase is maintained by an ArcSDE service operating on an Oracle database, you can store the resulting feature class using the Oracle Spatial geometry type format. This keyword has set the GEOMETRY_STORAGE parameter to SDO_GEOMETRY, indicating to ArcSDE that the resulting feature class should be stored as an Oracle Spatial geometry type.

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



Versioning your data

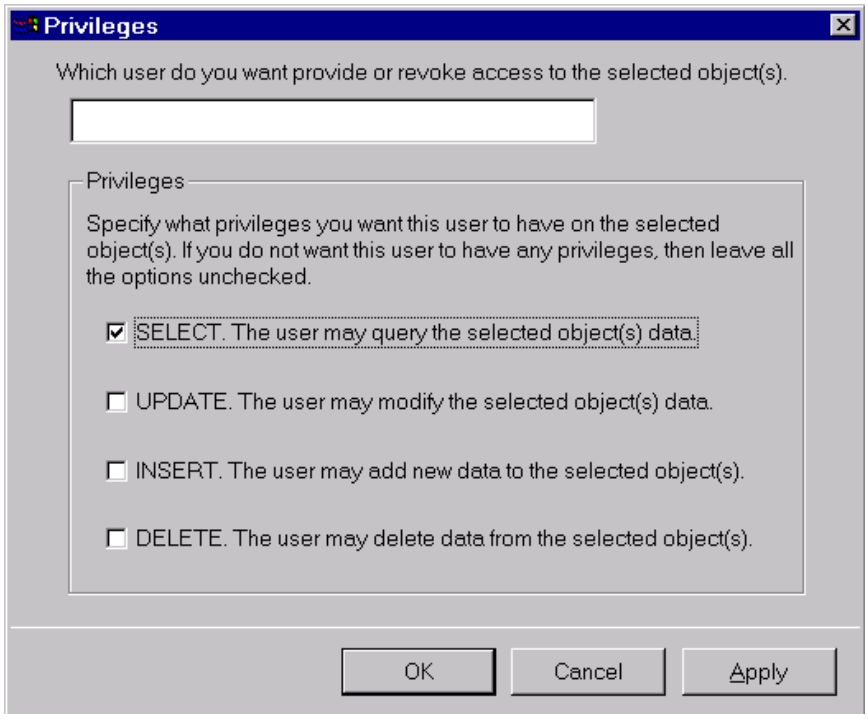
ArcCatalog also provides a means for registering data as multiversioned. Simply right-click the feature class to be registered as multiversioned and select the Register As Versioned context menu item.



A feature class is registered as multiversioned from within ArcCatalog.

Granting privileges

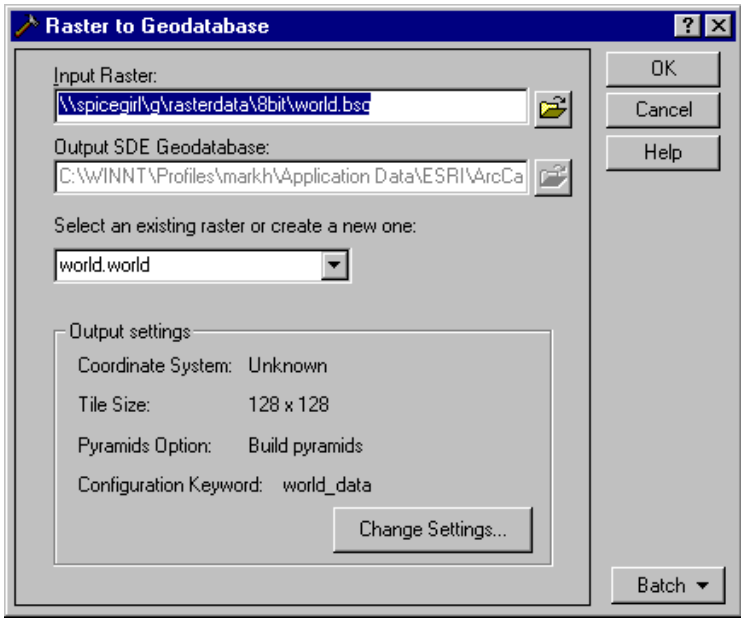
Using ArcCatalog, right-click on the data object class and click on the Privileges context menu. From the Privileges context menu assign privileges specifying the username and the privilege you wish to grant to or revoke from a particular user.



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.

Creating a raster column 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 Oracle database.


Raster to Geodatabase
Input Raster:
\\spicegirl\g\rasterdata\8bit\world.bsq
Output SDE Geodatabase:
C:\WINNT\Profiles\markh\Application Data\ESRI\ArcCa
Select an existing raster or create a new one:
world.world
Output settings
Coordinate System: Unknown
Tile Size: 128 x 128
Pyramids Option: Build pyramids
Configuration Keyword: world_data
Change Settings...
OK
Cancel
Help
Batch

Registering a business table

When ArcSDE registers a business table it performs a number of tasks depending on the type of registration that was requested. The duration of the registration process is dependent on the type of registration, whether the business table has a spatial column, and the table's number of rows.

Registering a table as NONE or USER maintained

Tables registered as NONE, are registered without a row id column.

Tables registered as USER, are registered with a row id column whose values you must maintain.

If the registration type is NONE or USER, ArcSDE merely adds a record to the SDE.TABLE_REGISTRY that references the business table. For tables registered as type USER the name of the row id is also added to the SDE.TABLE_REGISTRY entry.

Registration of these two registration types happens rather quickly.

Registering a table as SDE maintained

Tables registered as type SDE, must have a row id column that uniquely identifies the rows of the table.

NOTE: Tables registered by the geodatabase must be registered to ArcSDE as SDE maintained. If the geodatabase determined that the table has already been registered by ArcSDE as SDE maintained, the geodatabase uses the SDE maintained row id. In this case the geodatabase registration process is relatively inexpensive.

If a table was registered with a USER maintained row id, the geodatabase alters its row id registration to be SDE maintained.

By default the geodatabase adds a column called objectid to the table and registers it as SDE maintained. If the objectid column already exists, and is not currently registered as SDE maintained, the geodatabase will add a new column to the table called objectid_1.

Creating a new SDE maintained row ID column

If the row id column does not exist when the table is registered, ArcSDE adds a column of type INTEGER, with a NOT NULL constraint. If the table contains rows, ArcSDE populates the column with unique ascending values starting at your specified minimum id value. The minimum id value defaults to 1 if left unspecified. It then creates a unique index on the column called R<registration_id>_SDE_ROWID_UK, where registration_id is the registration identifier ArcSDE assigns the table when it was registered.

ArcSDE creates a sequence generator called R<registration_id> and uses it to generate the next value of the row id column whenever a value is added to the column.

Adding a column to an Oracle table that already contains rows can result in row migration, when the free space of the Oracle data block (as defined by PCTFREE) is consumed by the newly added column values.

Therefore, if at all possible, it is better to include the row id column as part of the table's original definition or add it before data is inserted into the table. If, however, you must add the row id column to a table that contains data, consider exporting the table afterwards, truncating the rows, and importing the data back into the table. Doing so eliminates the excessive disk I/O generated by fetching the migrated rows.

Using an existing column

If the row id column already exists, ArcSDE confirms that the column was defined as an integer. If it is not, the registration fails.

Next, ArcSDE confirms that the column has a unique index. If the column was defined with a non-unique index, ArcSDE drops the index.

In the event that the column does not have a unique index, ArcSDE attempts to create a unique index on the column. If the index creation fails because the column contains non-unique values, ArcSDE repopulates the column with ascending values beginning at 1 and then creates the unique index. ArcSDE names the unique index R<registration_id>_SDE_ROWID_UK.

Next, ArcSDE verifies that the column has been defined as NOT NULL.

If the column was defined as NULL, ArcSDE attempts to redefine it as NOT NULL. If this action fails, ArcSDE repopulates the column and defines it as NOT NULL.

Repopulating the column either because it contained null values, or because it contained non-unique values is an expensive process, especially if the table contains more than a 100,000 records.

Therefore if at all possible you should not rely on ArcSDE to perform this operation. You instead define the row id column as not null when the table is created and create your own unique index on it. At the very least, you should insure that the column is populated with unique integer values.

Analyzing the tables

Tables registered as type SDE must have statistics.

ArcSDE queries the LAST_ANALYZED value from the USER_TABLES view for the business table to determine if it has has been analyzed. If the value is NULL, ArcSDE analyzes the table with compute statistics.

Note: This is a very expensive operation for tables with more than a million rows. Therefore, to speed up the registration process, you should analyze large tables before you register them. For tables containing more than about 100000 rows, sampling from 1 to 10 percent of the rows, should be adequate. The larger the table, the smaller the sampling percentage can safely be.

If the business table contains a spatial column that is stored as binary, (see Appendix C for more information), the feature table and spatial index table must also have statistics. ArcSDE also queries the LAST_ANALYZED value of the USER_TABLES view for these tables.

If the value is not null, these tables and their indexes are analyzed.

Again, if they are very large (more than 1 million rows), you should analyze them yourself using a percentage of the total number of records.

Use the Oracle ANALYZE statement to analyze the tables.

```
ANALYZE TABLE <table_name>
ESTIMATE STATISTICS SAMPLE <percent_value> PERCENT;
```

If the statistics exist, but are stale, (many changes have occurred since the last time the statistics were created), you should reanalyze the tables. In this case ARCSDE will not analyze the tables since it will detect that the LAST_ANALYZE value is not null.The ArcSDE SQL statements have been written to take advantage of the Oracle cost based optimization.

Under cost based optimization Oracle uses the statistics of the tables it is querying to select the most optimal execution plan.

Note: The registration of an SDE maintained row id column is the only time the ArcSDE automatically analyzes the business, adds and deletes tables and their indexes as well as the feature table and the spatial index table if they exist.

Registering a table as multiversioned

To perform versioned edits on a business table, the table must be registered as multiversioned. When tables are registered as multiversioned the associated adds and deletes table are created and analyzed. These tables as their name implies, store the records of the business that are added and deleted. They are named A<registration_id> and D<registration_id>.

CHAPTER 5

Connecting to Oracle

Beginning with ArcSDE 8.1, ArcSDE client applications can connect to either an ArcSDE service or directly to an Oracle instance (at least Oracle 8.1). The direct connection capability, added for 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 libraries can connect directly to an Oracle instance.

Locating the ArcSDE Components

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 Oracle instance. Upon receiving results from the Oracle instance, 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 Oracle instances. Spatial requests are converted to SQL statements by the ArcSDE client applications instead of the ArcSDE service.

Setting up an 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 (sdesetupora*) 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.

Setting up a direction connection

Since a direct connection to an Oracle 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 (sdesetupora*), which creates the necessary ArcSDE system tables in the sde users schema. Also, to obtain a read–write connection to the Oracle 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.

Oracle Net8 Listener

To connect directly to an Oracle instance, you must configure the Oracle Net8 listener process, on the database host, to listen for Oracle client connections. You must also install the Oracle client on your client machine. The sections that follow provide details on how to set up the client and server machines to perform a direct connection to an Oracle 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 Oracle instance. The only difference occurs during the entry of the connection information.

Creating the Net8 listener service

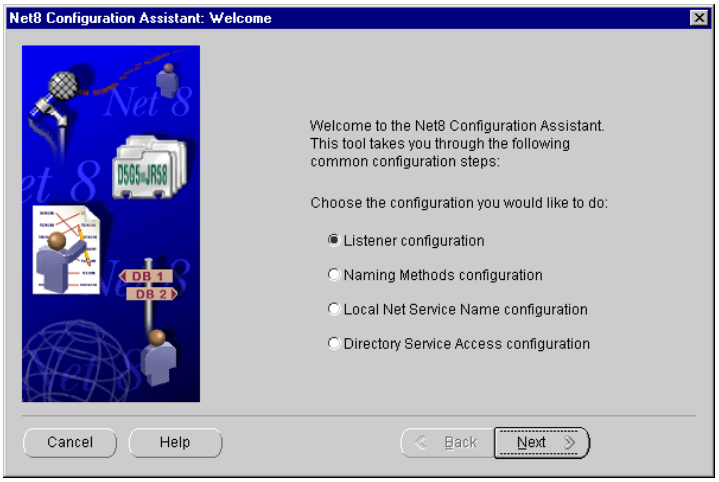
To establish a connection to an Oracle instance that is running on a remote computer, you must configure and start the Oracle Net8 listener process. Oracle allows you to configure the Net8 listener to listen on just about any network protocol that might exist in today's diverse networking environments. The discussion and the examples provided in this document are limited to the widely used TCP/IP network protocol.

The examples below are taken from an Oracle8*i* installation. Please refer to your Oracle installation documentation for your specific Oracle release.

Oracle offers a Java-based GUI on both the Windows NT and the UNIX server that simplifies the task of configuring the Oracle listener process. Provided is a simple walk-through of the Oracle Net8 Configuration Assistant. For a more in-depth discussion of the Net8 Configuration Assistant and other Oracle utilities, please refer to the Oracle online documentation.

On Windows NT launch the Net 8 Configuration Assistant by clicking Programs>Oracle> Ora*HOME*>Net8 Configuration Assistant. On UNIX systems launch the Net8 Configuration Assistant by running $ORACLE_HOME/bin/netca.

After the ‘Welcome’ panel appears choose the Listener configuration radio button and click Next.



The Net8 Configuration Assistant edits the listener.ora file normally located under the %ORACLE_HOME%\network\admin folder on Windows NT and the $ORACLE_HOME/network/admin directory on UNIX systems.

The Net8 Configuration Assistant will create a listener in the listener.ora file that looks something like this:

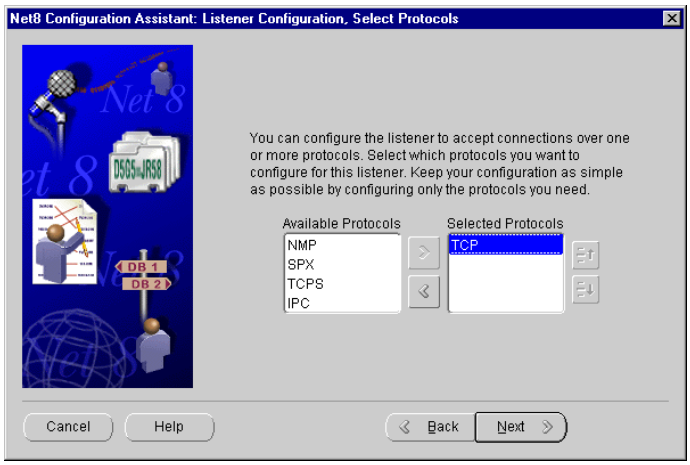
```
# LISTENER.ORA Network Configuration File:
D:\Oracle\Ora81\network\admin\listener.ora
# Generated by Oracle configuration tools.
LISTENER =
  ( DESCRIPTION_LIST =
    (  DESCRIPTION =
      ( ADDRESS_LIST =
        ( ADDRESS = ( PROTOCOL = TCP )( HOST = bruno )( PORT = 1521 ))
      )
    )
  )

SID_LIST_LISTENER =
  ( SID_LIST =
    ( SID_DESC =
      ( GLOBAL_DBNAME = bruno.esri.com )
      ( ORACLE_HOME = D:\oracle\ora81 )
      ( SID_NAME = bruno )
    )
  )
```

Note: For this particular listener.ora file, the GLOBAL_DBNAME parameter was manually entered after the Net8 Configuration Assistant added the listener service. You may also need to manually enter this parameter to the listener.ora file, depending on your network configuration.

If you receive an ORA 12514 error when you try to connect to the listener service, manually enter the GLOBAL_DBNAME parameter.

On the 'Listener Configuration, Listener Name' panel, enter the listener name. Unless you are an experienced Oracle DBA, use the LISTENER default value. Click Next and continue on to the 'Listener Configuration, Select Protocols' panel.

Select the TCP protocol from the Available Protocols list and click the right arrow button to move the TCP protocol into the Selected Protocols list. Click Next to configure the TCP/IP network protocol adapter from the Listener Configuration, TCP/IP Protocol.



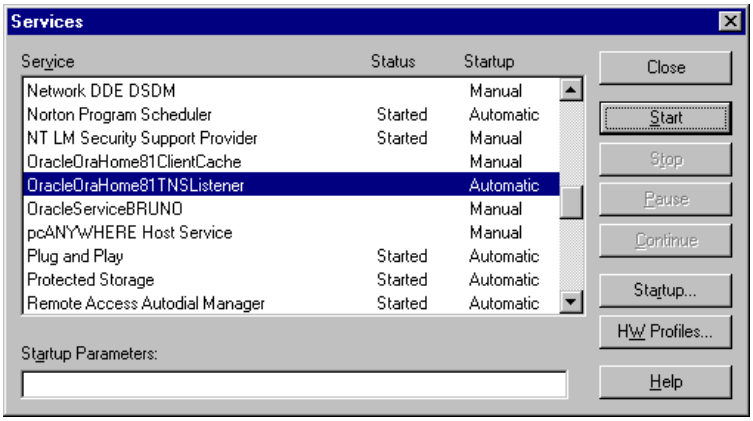
Choose the default ‘Use standard port number of 1521’ radio button. If you know that another software product is already using this port number, enter a different one.

Click Next to continue on to the ‘Listener Configuration, More Listeners?’ panel. You should not need to configure another listener, so choose the "No" radio button and click Next.

Click Next on the ‘Listener Configuration Done’ panel, which returns to the ‘Welcome’ panel.
Click Finish.

Starting the Net8 listener service

Before you can connect to the listener service, you must start it. To start the listener service on Windows NT, click the Start>Settings>Control panel and double-click the Services icon in the Control Panel window. After the Services dialog box appears, scroll down until you find the Oracle Net8 listener service and click the Start button.



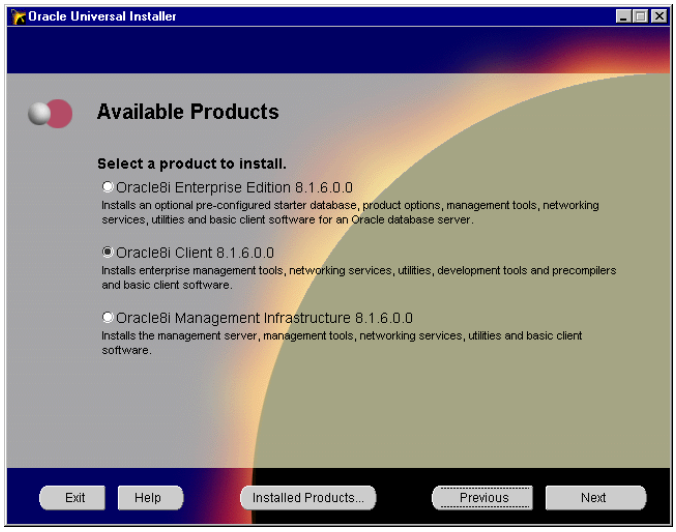
To start the Net8 listener process on a UNIX platform as the Oracle user, issue the **lsnrctl start** command at the operating system prompt.

```
$ lsnrctl start

LSNRCTL for Solaris: Version 8.1.7.0.0 - Production on 12-JAN-2001 16:37:15

(c) Copyright 1998 Oracle Corporation.  All rights reserved.

Starting /ultra1/app/ora817/product/8.1.7/bin/tnslsnr: please wait...

TNSLSNR for Solaris: Version 8.1.7.0.0 - Production
System parameter file is
/ultra1/app/ora817/product/8.1.7/network/admin/listener.ora
Log messages written to
/ultra1/app/ora817/product/8.1.7/network/log/listener.log
Trace information written to
/ultra1/app/ora817/product/8.1.7/network/trace/listener.trc
Listening on: (DESCRIPTION=(ADDRESS=(PROTOCOL=ipc)(KEY=ultra)))
Listening on: (DESCRIPTION=(ADDRESS=(PROTOCOL=tcp)(HOST=ultra)(PORT=1521)))

Connecting to (DESCRIPTION=(ADDRESS=(PROTOCOL=IPC)(KEY=ultra)))
STATUS of the LISTENER
------------------------
Alias                     LISTENER
Version                   TNSLSNR for Solaris: Version 8.1.7.0.0 - Production
Start Date                12-JAN-2001 16:37:15
Uptime                    0 days 0 hr. 0 min. 0 sec
Trace Level               admin
Security                  OFF
SNMP                      OFF
Listener Parameter File
/ultra1/app/ora817/product/8.1.7/network/admin/listener.ora
Listener Log File
/ultra1/app/ora817/product/8.1.7/network/log/listener.log
Listener Trace File
/ultra1/app/ora817/product/8.1.7/network/trace/listener.trc
Services Summary...
  ora817               has 1 service handler(s)
The command completed successfully
```

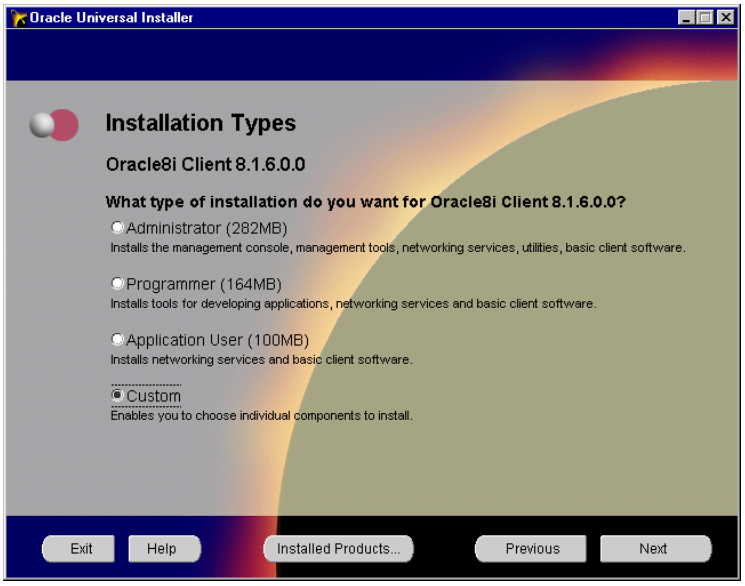
Net8 Client installation and configuration

To make a direct connection to an Oracle instance, Oracle Client must be installed on your client machine. Oracle Client is shipped with the Oracle Enterprise Edition. Oracle Client includes the installation of the Oracle Net8 software, which is necessary to connect to an Oracle server.

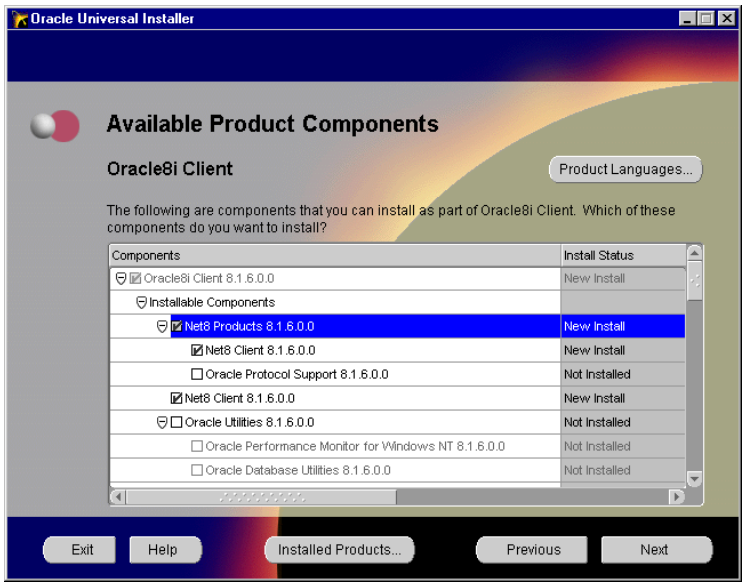
After you start the Oracle Universal installer and complete the preliminary panels, choose the Oracle Client radio button from the Available Products panel and click Next.



From the ‘Installation Types’ panel select the Custom radio button and click Next.

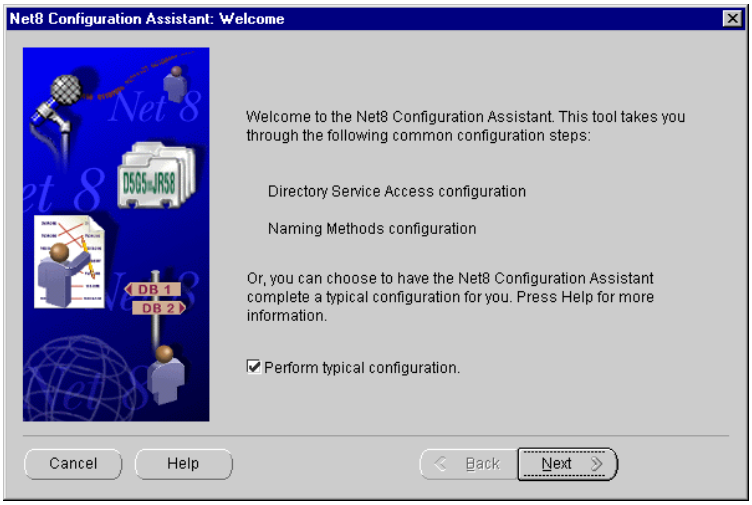


From the ‘Available Product Components, Oracle 8*i* Client’ panel you need to check at least the Net8 Products and Net8 Client boxes to enable a direction connection to an Oracle instance. Click Next to proceed to the Installation Summary panel and click Next again to install the Net8 products of the Oracle Client. Immediately following the installation of the software, the Net8 Configuration Manager appears, which allows you to configure the net service name needed to connect to a remote Oracle database.

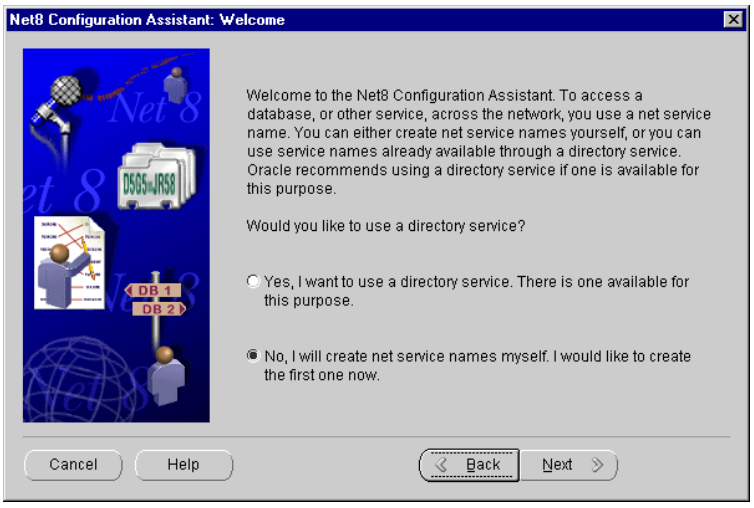


When the 'Net8 Configuration Assistant Welcome' panel appears, check 'Perform typical configuration' and click Next to add the net service name. Alternatively, you can click Cancel to defer the configuration of the net service name. Clicking Cancel returns control to the Oracle Universal Installer installation summary panel.

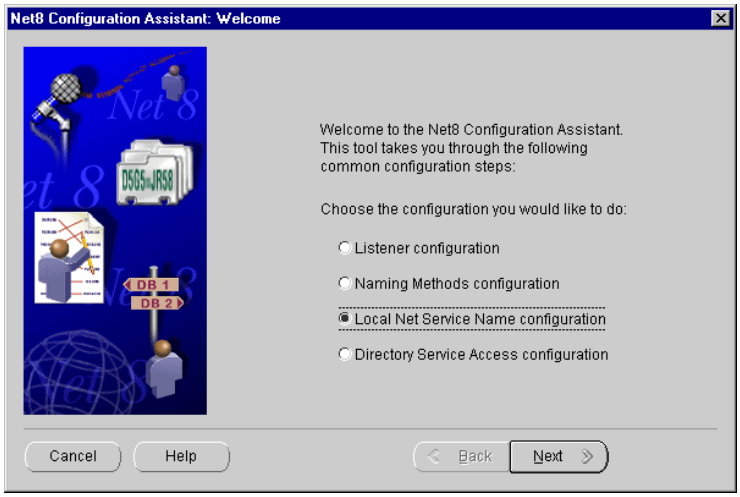
On Windows NT you can launch the Net 8 Configuration Assistant by clicking Programs>Oracle> Ora*HOME*>Net8 Configuration Assistant. On UNIX systems you can launch the Net8 Configuration Assistant by running $ORACLE_HOME/bin/netca.



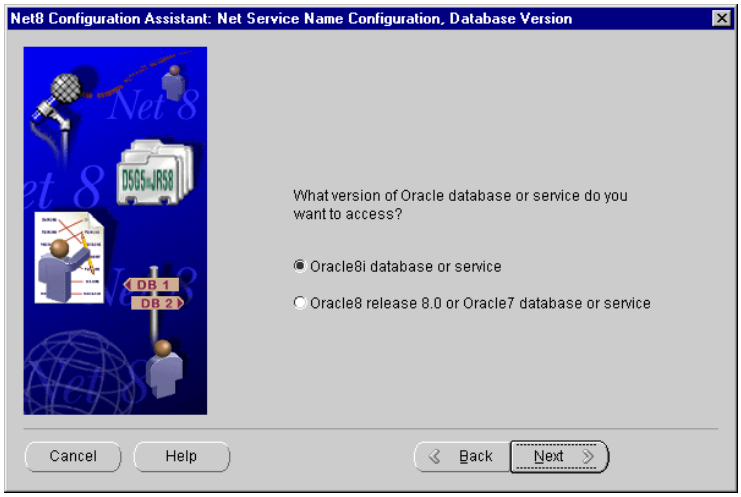
On the 'Net8 Configuration Assistant Welcome' panel that follows, click the 'No, I will create net service names myself. I would like to create the first one now.' radio button.



From the 'Net8 Configuration Assistant: Welcome' panel, choose the 'Local Net Service Name configuration' radio button and click Next to proceed to the 'Net Service Name Configuration, Database Version' panel.

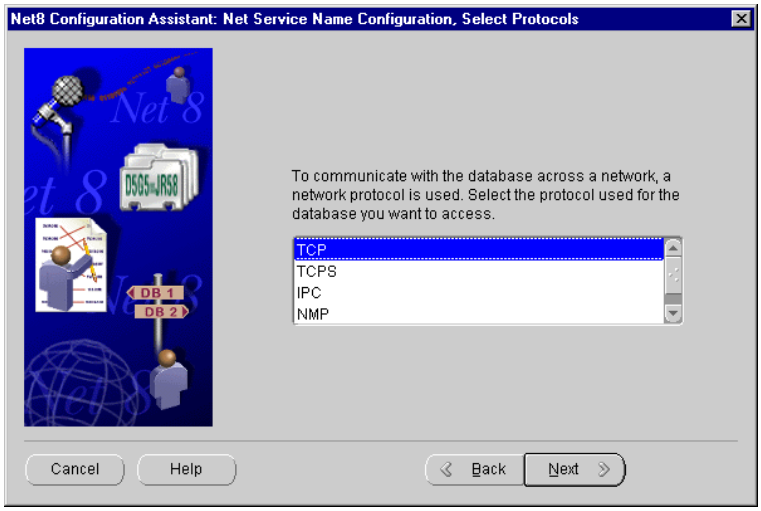


Choose the 'Oracle database or service' radio button. Click Next and continue to the 'Net Service Name Configuration, Service Name' panel.

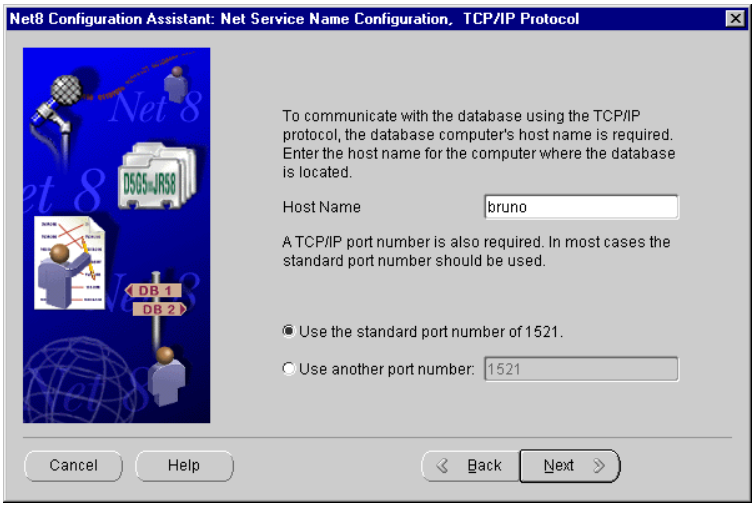


Although the service name can be anything you want it to be, for some Oracle applications such as Advance Replication it is required to be the global database name. By convention, you should use the global database name as it is easier to keep track of. The global database name is formed by concatenating the DB_NAME (database name) and the DB_DOMAIN (database domain) together and separating them by a period. In this example the database name, *bruno*, and the database domain, *esri.com*, concatenate to form the global database name *bruno.esri.com*. Click Next to continue to the Net Service Name Configuration, Select Protocols panel.

In the previous section we created a listener that listens on the TCP/IP protocol. Select the TCP protocol and click Next to proceed to the 'Net Service Name Configuration, TCP/IP Protocol' panel.



Enter the host name of the computer that the database is running on. The host name should be listed in your NIS hosts file. You can also enter the computer's IP address, although you should set up and use the host name since it is easier to recognize. Choose the 'Use the standard port number of 1521.' radio button unless you were forced to use a different port number when you configured the listener. Click Next to continue to the Net Service Name Configuration, Test.



You should test the connection of the service name that you have just created. You can return later to test the connection, but if you are ready to test it now, choose the 'Yes, perform a test' radio button and click Next. The 'Net Service Name Configuration, Connecting' panel will appear with the results of the connection test. If the Details dialog box does not contain 'Connecting…Test successful.', there is a problem with the connection. The first thing to look at is the username and password used to make the connection. By default, the Net8 Configuration Assistant defaults to scott as the username and tiger as the password. If necessary, click Change Login to use a different username and password. When you click OK on the 'Change Login' panel, Net8 Configuration Assistant will attempt the connection again. If you are still unable to connect to the listener, contact Oracle Technical Support for help.

Following a successful connect test, click Next to proceed to the 'Net Service Name Configuration Done' panel. Click Next on this panel to return to the 'Welcome' panel. Click Finish. The tnsnames.ora file has been updated and can now be referenced by an Oracle client software product to make a connection to an Oracle database.

The entry made to the tnsnames.ora file by the Net* Configuration Assistant should look something like this:

```
BRUNO.ESRI.COM =
  ( DESCRIPTION =
    ( ADDRESS_LIST =
      ( ADDRESS = ( PROTOCOL = TCP )( HOST = bruno )(PORT = 1521 ))
    )
    ( CONNECT_DATA =
      ( SERVICE_NAME = bruno.esri.com )
    )
  )
```

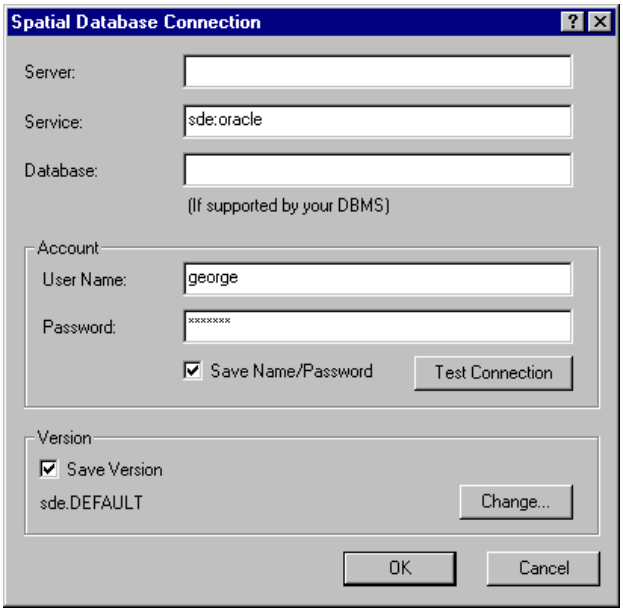
Connecting to Oracle from an ArcSDE client application

ArcSDE applications have the option of connecting to Oracle through an ArcSDE service. With this option, the ArcSDE *giomgr* service establishes a *gsrvr* process to manage the transfer of data between the database server and the client application. There will be one *gsrvr* process for each database connection established by any ArcSDE application.

ArcSDE applications that have linked in the client libraries of the ArcSDE for Oracle C API are also capable of connecting to Oracle instances without using the ArcSDE *giomgr* service. The function of the *gsrvr* process is incorporated directly into the application itself. This is called *direct connect.*

ESRI applications software use the same graphical user interface for getting database connection parameters regardless of the type of database you are connecting to, or the method of connecting.

In ArcCatalog, the SDE Connections dialog box, shown below, appears after double-clicking on the Database Connections in the Contents pane, followed by double-clicking on Add SDE Connection.



The contents of the Server, Service, User Name, and Password fields depend upon whether you use a direct connection or connect through the ArcSDE server. For Oracle connections, regardless of method, you must leave the Database field blank. (The field is present in the dialog box because other database engines may require it.)

Whether connecting to the Oracle database through an ArcSDE service or connecting directly, you must:

- install the Oracle Client software on the local machine.

- configure the Oracle Client tnsnames.ora file with the net service name containing the connection information to the remote Oracle.

- set up certain environment variables or registry variables in the ArcSDE environment.

The previous sections in this chapter provide specific information on this installation and configuration of Oracle. Refer to the following sections for specific information on setting up the ArcSDE environment for various configurations of ArcSDE service and Oracle database server.

Configuring the ArcSDE /etc directory

As an option, files can be created in the /etc directory of ArcSDE client application to capture error messages and establish a default Oracle connection environment. An ArcSDE client application will search for the etc directory under the SDEHOME environment variable.

SDEHOME is determined by an ArcSDE client application using the following search criteria:

1. If the SDEHOME environment variable is set in the system environment, the etc directory, if one exists, must be located in the directory defined by the value of the SDEHOME environment variable.

2. If the SDEHOME environment variable is not set and the ARCHOME environment variable is, the etc directory, if one exists, must be located in the directory defined by the value of the ARCHOME environment variable.

3. Finally, if neither the SDEHOME nor the ARCHOME environment variables is set, the etc directory must be located in the directory defined by the registry key value of HKEY_LOCAL_MACHINE / SOFTWARE / ESRI / ArcInfo / Desktop / 8.0 / SourceDir.

If the etc directory exists, upon initial connection to the Oracle instance, the ArcSDE application creates the sde.errlog file within the etc directory. All subsequent error messages will be written to this file. Therefore, if you cannot establish a connection to the Oracle instance, look in the sde.errlog for possible further information as to what the problem may be.

Locating the dbinit.sde file (Unix and Windows)

You may optionally use the dbinit.sde file to set environment variables for a specific client machine. For example, including this line within the dbinit.sde file:

```
ORACLE_SID = TESTDB01
```

will override any existing setting of the ORACLE_SID variable once an ArcSDE client application starts.

To locate the dbinit.sde file on Unix systems, the following directories are searched in order:

1. $SDEHOME/etc

2. $ARCHOME/etc

Resolving the Oracle server

If the Oracle server is running on the same host as the client application, and you do not wish to create a tnsnames entry for it, then you must specify the name of the instance using the ORACLE_SID environment variable.

If the Oracle server is running on a different host from the client application, you must create a tnsnames entry describing the host and Oracle instance. With the tnsnames entry established, you may

1. (Windows) set the LOCAL system variable to refer to the tnsnames entry and log in using the "username/password" syntax;

2. (Unix) set the TWO_TASK environment variable to refer to the tnsnames entry and log in using the "username/password" syntax;

3. Use the syntax "username/password@<tnsentry>" when logging in.

Resolving system or environment variables

System variables for Windows are resolved in the following priority order:

1. dbinit.sde is checked to see if it sets either LOCAL or ORACLE_SID

2. If not set within dbinit, the LOCAL system variable is checked

3. ORACLE_SID environment variable

4. If a service is used, then the following registry key is checked:

HKEY_LOCAL_MACHINE / SOFTWARE / ESR I / ArcInfo / ArcSDE / 8.0 / ArcSDE for Oracle / ORACLE_SID

Choosing a service vs. a direct connection

To connect to the ArcSDE server using a service, you must first create the service with the **sdeservice** command and then start the service with the **sdemon** command. Refer to the *Managing ArcSDE Services* manual for more information on how to do this.

After the service is started you specify the host on which the service is running in the Server field and specify the service name running on that host in the Service field of the login screen. If the service is running on the same host as the client application, you may leave the Server field blank.

Examples of a direct connection

In the examples that follow, we assume that the ORACLE_SID is "GEODATA", the username is "surveyor" and the password is "theodolite". Note that the password is not displayed in the dialog box, but is echoed as a sequence of asterisks. If the example uses a tnsnames entry, we assume that it will be "GEODATA", the same as the ORACLE_SID. (Making the tnsnames entry the same as the ORACLE_SID is common practice.)

Example 1 – Local connection

To make a direct connection to an Oracle instance running on your local machine, you would do the following:

1. Set the ORACLE_SID environment variable to name the Oracle instance on the local machine that you want to connect to.

2. Fill in the database login parameters as follows:

Server	
Service	sde:oracle
Database	
User Name	surveyor
Password	theodolite

Since neither TWO_TASK nor LOCAL are set, the ArcSDE service attempts to connect to an Oracle instance running on the local machine only.

Example 2 – Remote connection using LOCAL or TWO_TASK

To make a direct connection to an Oracle instance running on another host you would do the following

1. Create a tnsnames entry, for example GEODATA

2. Set TWO_TASK (Unix) or LOCAL (Windows) equal to "GEODATA"

3. Fill in the database login parameters as follows:

Server	
Service	sde:oracle
Database	
User Name	surveyor
Password	theodolite

The ArcSDE server uses the information in the LOCAL or TWO_TASK variable to reference the tnsnames entries. From the tns entry, the system knows which host to connect to.

Example 3 – Remote connection without using LOCAL or TWO_TASK

To make a direct connection to an Oracle instance running on another host without using LOCAL or TWO_TASK, you would do the following:

1. Create a tnsnames entry, for example GEODATA

2. Fill in the database login parameters as follows:

Server	
Service	sde:oracle
Database	
User Name	surveyor
Password	theodolite@GEODATA

The tnsnames entry is explicitly given this time, which is necessary since it is not specified by the TWO_TASK or LOCAL environment variables.

Example of connecting through an ArcSDE service

In the examples that follow, we assume that the ORACLE_SID is "GEODATA", the Server is "geoserver", the Service is "esri_sde", the username is "surveyor" and the password is "theodolite". If the example uses a tnsnames entry, we assume that it will be the same as the ORACLE_SID, that is, it will be "GEODATA".

Example – Connecting to a service running on a different server

To connect to Oracle through a service running on another host you would do the following:

1. Create a tnsnames entry, for example GEODATA on the remote host

2. Use **sdeservice** to create the esri_sde service on the geodata host and **sdemon** to start it.

3. Fill in the database login parameters as follows:

Server	geoserver
Service	esri_sde
Database	
User Name	surveyor
Password	theodolite@GEODATA

The client application uses the Server value to establish a connection to the ArcSDE service running on that server. From that host it uses the GEODATA tns entry to create a connection to the database. Note that in this configuration, the client application, the ArcSDE service and the Oracle database server can be running on three separate machines.

Troubleshooting direct connection problems

Some of the more common problems that occur when you try to establish a direct connection to Oracle are listed below.

Oracle server was not started

Symptom: You try to connect to the Oracle instance, and you receive:

```
ORA 01034.
```

Cause: The Oracle instance has not been started.

Solution: Start the Oracle instance.

Oracle listener was not started

Symptom: You try to connect to the Oracle instance, and you receive:

```
ORA-12541: TNS:no listener.
```

Cause: The Oracle listener has not been started.

Solution: Start the Oracle listener.

Oracle username or password are incorrect

Symptom: You try to connect to an Oracle instance, and you receive:

```
ORA-01017: invalid username/password; logon denied error
```

Cause: Either the username or the password is incorrect.

Solution: Connect the username or password and try again. If you cannot remember the password, connect to Oracle as the DBA and change the password with the ALTER USER command. If you cannot remember the username, connect to Oracle as the DBA and list the available usernames from the DBA_USERS table.

Net service name does not exist

Symptom: You try to connect to an Oracle instance, and you receive:

```
ORA-12154: TNS:could not resolve service name
```

Cause: Oracle could not find the net service name in the tnsnames.ora file.

Solution: Check the spelling of the net service name and reenter if misspelled. If you are sure you entered the net service name correctly, examine the net service name in the tnsnames.ora file to make sure that it was entered correctly.

Net service name cannot be resolved in listener

Symptom: You try to connect to an Oracle instance, and you receive:

```
ORA-12514: TNS:listener could not resolve SERVICE_NAME given in
connect descriptor
```

Cause: Although the net service name was found in the tnsnames.ora file, and connection information was able to locate the listener service running on the remote host, Oracle was not able to locate the Oracle instance.

Solution: A common cause of this problem is the absence of the GLOBAL_DBNAME parameter in the listener.ora file. Examine the listener.ora file. If the GLOBAL_DBNAME parameter is missing, add it and restart the listener again. Retry the connection.

The Oracle instance is not at least an Oracle 8.1 instance

Symptom: You try to connect to an unsupported instance, and you receive:

```
ERROR in clearing lock and process tables.
Error:  -51
DBMS error code: 6550
ORA-06550: line 1, column 219:
PLS-00201: identifier 'SDE.PINFO_UTIL' must be declared
ORA-06550: line 1, column 219:
PL/SQL: Statement ignored

Couldn't initialize shared memory, error = -51.
Cannot Attach to Shared Memory -1
```

Cause: The Oracle instance is not at least Oracle 8.1, or the sdesetupora* program was not run. Make sure the sdesetupora* program has been run. The sdesetupora* program will only run on at valid, supported Oracle instance.

Solution: Check your Oracle version by querying the dynamic view V$VERSION. If the Oracle version is not at least Oracle 8.1, you cannot establish a direct connection.

CHAPTER 6

National language support

Storing data in an ArcSDE Oracle database using character sets other than US7ASCII requires some extra configuration on both the client and the server. This section provides guidelines for configuring both the Oracle database and the ArcSDE client environment to enable the use of character sets other than US7ASCII.

Oracle database character sets

If you are using UNIX, by default an Oracle database is created with a US7ASCII character set. On Windows NT the default character set is WE8ISO8859P1. The character set is selected when the database is created with the CREATE DATABASE statement and cannot be changed afterwards. To change a character set the database must be re-created and the data reloaded. Consult the *Oracle National Language Support Guide* for your Oracle release to determine the character set that is right for your data.

Setting the NLS_LANG variable on the client

Once the ArcSDE Oracle database has been created with the proper character set, data can be loaded into it using a variety of applications such as ArcGIS Desktop and the ArcSDE administration tools shp2sde and cov2sde. To properly convert and preserve the characters, you must set the Oracle NLS_LANG variable in the client applications system environment.

For instance, using ArcToolbox installed on Windows NT to convert a coverage containing German attribute data into an Oracle database on a UNIX server created with the Western European character set WE8ISO8859P1, you would set the NLS_LANG to GERMAN_GERMANY.WE8ISO8859P1. To set the NLS_LANG variable for ArcToolbox, click Start, Settings, and Control Panel. Double-click on the System icon and select the Environment tab after the System menu appears. Click on the System Variables scrolling list and enter NLS_LANG in the Variable: input line and GERMAN_GERMANY.WE8ISO8859P1 in the Value: input line. Click Set and then OK.

Setting the NLS_LANG variable for Windows NT clients

Be careful setting the NLS_LANG on the Windows NT platform because there are actually two different codepage environments on this platform. Windows applications such as ArcGIS Desktop run in the Windows American National Standards Institute (ANSI) codepage environment, while ArcSDE administration tools and C and Java API applications

invoked from the MS–DOS Command Prompt run in the original equipment manufacturer (OEM) codepage environment. Some languages require two different NLS_LANG settings for the language character set component for each of these codepage environments.

For instance, in the above example the NLS_LANG variable would be set to GERMAN_GERMANY.WE8PC850 if the data was loaded from the MS–DOS Command Prompt using the ArcSDE administration tool shp2sde. If you use both Windows applications and MS–DOS applications together, then you should set the NLS_LANG variable for MS–DOS applications when you open the MS–DOS Command Prompt using the MS–DOS SET command.

SET NLS_LANG = GERMAN_GERMANY.WE8PC850

To determine if your language requires a separate language character set, consult the *Oracle National Language Support Guide* for your Oracle release.

For more information on ArcInfo national language support, refer to the National Language Support section of the Systems Administration chapter found on your ArcDoc™ CD.

CHAPTER 7

Backup and recovery

Data safety is ensured by implementing, testing, and verifying an appropriate database backup and recovery procedure. Oracle provides a number of choices for backing up a database including exporting data, selective backup of tablespaces, or copying database files.

This chapter presents a summary of the Oracle concepts that are essential for understanding backup and recovery, plus specific guidelines that may be applied to your ArcSDE installation.

For details, refer to the *Oracle Backup and Recovery Guide* for your Oracle release.

Recording Database Changes

Oracle manages changes to its contents and structure in such a manner as to guarantee recovery of the database to the last committed transaction after any single point of failure. This guarantee is at the heart of Oracle's architecture and this statement says a great deal.

To the last committed transaction means that, once control is returned to the user after executing a COMMIT statement, Oracle guarantees that the committed data has been written to disk in some form and is recoverable.

Single point of failure means that any single file or process can fail without losing the contents of any committed transaction. If a data file is lost or corrupted, the contents of the redo logs guarantee that the data is recoverable. If a control file breaks, the other control files ensure that the information remains safe. A process may be killed, but committed data is never lost.

Such a guarantee requires a high level of coordination between database processes and database files. Files "know about" each other and may not be deleted or changed without properly coordinating through the Oracle instance.

This section briefly describes the role of the various Oracle database files in backup and recovery to achieve this guarantee.

Control files

The physical structure of the database—the arrangement of physical files on disk—is recorded in the *control file*. The control file is so important for the purpose of database consistency that Oracle maintains multiple identical copies of it.

If the database loses one of its control files, the database will halt. You will have to do a SHUTDOWN ABORT on the database instance prior to recovering the database.

Oracle recommends that any database instance should use *at least three* identical control files, and further recommends that each control file be placed on a physically separate disk drive under a separate disk controller. While a single hardware failure may cause the loss of one control file, this strategy ensures the safety of the remaining control files.

The locations and names of the control files are controlled by the init.ora parameter:

CONTROL_FILES

Note: If you add or drop a data file or an online redo log file, or alter file locations, the contents of the control file change as this information is recorded. Since the backup version of the control file no longer matches the current database structure, you should take an immediate backup of the new control file.

Redo log files

Every change that is made to an Oracle database—such as creating or altering schema objects or inserting or manipulating data—is recorded in the *online redo log file*. This happens automatically in addition to the normal action of writing the changes to the appropriate *data file* or control file.

Changes to the database are recorded in the order that they occur. If, for example, you create a table to store GIS data, then insert some data into that table, the create table action will be recorded first, and the insert data action will be recorded second.

Recording the order of the actions is very important and individual steps may not be neglected. In this example, if the table creation step is skipped or performed at the wrong time, the data insertion step will fail.

Eventually, the database change information recorded in a redo log file accumulates to some preset maximum. Oracle then performs a *log switch*, closing the current online redo log file and opening the next one.

Oracle uses the online redo log files in a circular fashion, reusing the first log file after it closes the last log file. If three online log files are specified, for example, Oracle will use the log files in order: 1, 2, 3, 1, 2, 3, 1, … This means that earlier information stored in any of the online redo log files will be lost when the file is used again.

Archiving redo log files

Oracle provides for the online redo log files to be copied to an offline location immediately after a log switch. This is called *archiving* the redo log files.

When an Oracle database is run in ARCHIVELOG mode, redo log information is accumulated indefinitely by copying the online redo log file to an offline location with a unique name based on the current *log sequence number*.

For example, as Oracle switches through online redo log files in order (1, 2, 3, 1, 2,…) the following archived redo log files (reflecting log sequence numbers 15, 16, 17, 18, 19,…) might be created:

```
LOG0015.ARC
LOG0016.ARC
LOG0017.ARC
LOG0018.ARC
LOG0019.ARC
```

etc.

In ARCHIVELOG mode, the total amount of redo log information is limited only by the total file space available at the archive destination. When an Oracle database is run in NOARCHIVELOG mode, the online redo log files are not copied, so that the amount of usable redo log information is limited to the contents of the current online redo log files.

Which archive mode should you use?

Often databases are created and initially loaded under the NOARCHIVELOG mode because, should a media failure occur, the same effort is required to restore the database regardless of whether the database was reloaded using the archived redo log files or the original source data. In addition, a huge amount of log file data is created, and maintaining this data is unnecessary since the original source data is readily available.

After the initial data load is complete and the database goes into production, the changes become difficult to re-create. Switch to ARCHIVELOG mode to ensure that point-in-time recovery of the database is available in the event of a media failure.

Operate the database in NOARCHIVELOG mode and make frequent backups *only* if you understand the risk of data loss and can afford to re-enter the changes made since the last backup.

Note: Because of the risk of losing committed transactions in the event of media failure, ESRI does *not* recommend using NOARCHIVELOG mode for normal operation of an ArcSDE database.

Switching from NOARCHIVELOG to ARCHIVELOG

An Oracle database created in NOARCHIVELOG mode can be switched to ARCHIVELOG mode. To make this change, shut down the database using the NORMAL, IMMEDIATE or TRANSACTIONAL priority. (Do not use ABORT.):

```
SVRMGR> connect internal
Connected.
SVRMGR> shutdown normal
Database closed.
Database dismounted.
Oracle instance shut down.
```

Before switching to ARCHIVELOG mode, Oracle recommends that you make a full backup of the database.

To the init.ora file, add the LOG_ARCHIVE_START and LOG_ARCHIVE_DEST parameters.

```
LOG_ARCHIVE_START = TRUE
LOG_ARCHIVE_DEST = <pathname_to_archives>/arch%t_%s.dbf
```

Setting LOG_ARCHIVE_START to true causes the ARCH process to archive the online redo log files automatically. If you do not set this parameter or set it to false, you will have to manually archive the log files. For information on how to perform a manual archive, consult the *Oracle Server Administrator's Guide* for your Oracle release.

After you have backed up the Oracle database, switch the database to ARCHIVELOG mode:

```
$ svrmgrl

Oracle Server Manager Release 3.1.6.0.0 - Production

(c) Copyright 1997, Oracle Corporation. All Rights Reserved.

Oracle8i Enterprise Edition Release 8.1.6.0.0 - Production
With the Partitioning and Java options
PL/SQL Release 8.1.6.0.0 - Production

SVRMGR> startup mount
Total System Global Area          245317008 bytes
Fixed Size                            64912 bytes
Variable Size                     103677952 bytes
Database Buffers                  131072000 bytes
Redo Buffers                       10502144 bytes
Database mounted.
SVRMGR> alter database archivelog;
Statement Processes
SVRMGR> alter database open;
Statement processed.
SVRMGR>
```

Note: After the database successfully reopens in ARCHIVELOG mode, you should shut it down again and make another full backup. If you don't, your last full backup will have recorded that the database was in NOARCHIVELOG mode, and you will not be able to apply the online redo log files following the restoration of the backup.

Sizing the redo log files

If you have created small online redo log files, you should consider creating new, larger redo log files.

Every time that Oracle switches the log file, it first performs a *checkpoint* in which the contents of changed blocks within the database buffer cache are written to disk. Changes to the database are always suspended while checkpoint processing is taking place.

In a large multiuser system where many changes are being made, the checkpoint process may involve practically the entire database buffer cache and should, therefore, be put off for as long as possible. Where the number of database changes is small, checkpoint processing will be insignificant.

If the database is configured with only three large online redo log files without mirroring—the default configuration—you should create more log file groups and mirror them before you switch the database to ARCHIVELOG mode.

Additional log file groups helps smooth out the archiving process by enabling the archive process to finish copying a log file to the archive destination before the database needs to make the log file current again. If the database must make a log file current before the archive process has finished copying it, the database will wait until the archive is finished before continuing processing.

Refer to Chapter 2, *Essential Oracle configuring and tuning* for guidance on configuring the online redo logs.

For details on commands used to manage redo log files, refer to the *Administrator's Guide* for your Oracle release.

Database backup

To back up your ArcSDE Oracle database, you must copy these files to an offline location:

- Control files
- Data files
- giomgr.defs, dbinit.sde, and services.sde files
- Archived redo log files

You should make at least three copies of the control files with each backup because of their importance in ensuring database consistency. Because control files are comparatively small, there is negligible cost in doing do.

Remember that the redo log files are essential for bringing the data files from an earlier state to a later state. Between any two points in time, the redo logs must be found in an *unbroken* sequence if database recovery is to succeed.

ESRI recommends that you maintain *at least two* copies of all archived redo logs for as far back in time as may be reasonably necessary for database recovery. The two copies should be stored on physically distinct media—separate disk drives, for example, or a disk drive and a tape drive.

If you intend to purge the archived redo log files from their location on disk, be sure that you have a *second* backup copy of each archived redo log file before purging.

This strategy of multiple backups of the archived redo log files helps guard against multiple media failure, which is not as rare as it might seem. Some tape drives, for example, fail to detect bit errors until you attempt to restore a file, when it may be too late.

You only need to make a single copy of each data file with each backup as long as you carefully maintain multiple copies of archived redo logs.

Note: The online redo log files are *not* necessary for backup. If the current online redo log file fails, committed information still exists in memory, which Oracle writes to the data files when a checkpoint is issued. Oracle issues a checkpoint automatically when you shutdown a database instance with NORMAL, IMMEDIATE or TRANSACTIONAL priority. Before shutting down a database with ABORT priority you should force a checkpoint with the ALTER SYSTEM … CHECKPOINT command if at all possible.

Hot backup

Backing up an Oracle database while the database instance is running is called a *hot backup*. If you plan to take a hot backup, you must operate your database in ARCHIVELOG mode.

Enter the ALTER TABLESPACE … BEGIN BACKUP command prior to the backup of *each* tablespace, which tells Oracle that a hot backup is being taken. If this command is not issued, the hot backup will appear to succeed but it may be useless for restoring the database. To finish the hot backup, for each tablespace enter the ALTER TABLESPACE … END BACKUP command.

Changes to data are recorded and held in the rollback segment until they are no longer needed by any outstanding transaction. Taking a hot backup prevents the release of rollback segment data until the ALTER TABLESPACE … END BACKUP command is issued.

Therefore, the rollback segment must be large enough to accommodate changes made during the hot backup. If the rollback segment runs out of space, a transaction will fail with an ORA-1555 error:

```
ORA-1555: snapshot too old (rollback segment too small)
```

While the hot backup will succeed despite this error, changes made to the database may need to be reentered.

You may avoid this error by taking a hot backup during times of low database activity or by ensuring that your rollback segments are large enough to accommodate data changes made during backup.

You do not need to shutdown the ArcSDE server process (giomgr) prior to making a hot backup.

For details on hot backup refer to the *Oracle Backup and Recovery Guide* for your Oracle release.

Cold backup

Backing up an Oracle database while the database instance is shutdown is called a *cold backup*. Managing a cold backup is simpler than a hot backup and less prone to error.

If you run the database in NOARCHIVELOG mode, a cold backup is your only option. Running the database in ARCHIVELOG mode will enable you to use a cold backup to recover a database to the latest committed transaction.

Database export

You can use Oracle's export utility to supplement a full backup. If changes are made to a known set of data objects between full backups, you can export the objects.

However, the export utility should only be used on data objects that are not changing during the export and on all data objects that are closely related. For example, if you use export to back up a business table, you should also capture the related spatial index, feature and delta tables in the same backup.

ESRI does *not* recommend that you use the export utility as your only backup method.

You can also back up the entire Oracle database with the Oracle export utility and then make cumulative and incremental backups.

For more information on the export, utility refer to the *Oracle Utilities* manual for your Oracle release.

Frequency of backups

Base the frequency of your backups on the rate at which the data in your database is changing. The more changes that occur, the more frequently backups should occur.

If your Oracle database is operating under ARCHIVELOG mode, you have several variations that you may add to your backup strategy in addition to the periodic full backup. Databases operating under the NOARCHIVELOG mode are restricted to full backups with the possible addition of Oracle export files.

For more information on different Oracle database backup strategies, refer to the *Oracle Backup and Recovery Guide* for your Oracle release.

Regardless of which archiving mode you are using, you should make regular full backups of the Oracle database. A full backup should include the Oracle database and, if you have started an ArcSDE service, the giomgr.defs, dbinit.sde, and services.sde files.

Scripting the backup

Once you have decided upon your backup strategy, the commands that are entered through the system prompt or through SQL*Plus may often be scripted.

ESRI recommends using a script for backup purposes whenever possible.

Executing the backup commands from a Unix shell script, or a Windows batch file will help ensure that the database is being backed up consistently and that all intended steps are being followed.

Be sure to update the script whenever the physical structure of the database changes, for instance whenever you add a data file or move a redo log file.

For examples of backup scripts refer to the *Oracle Backup and Recovery Guide* for your Oracle release.

Database recovery

To recover a database after any failure, Oracle takes these steps:

1. Reads the init.ora file to determine the names and locations of the control files.
2. Reads the control files to verify their consistency with each other and to determine the physical file structure of the database.
3. Opens each data file mentioned in the control file to determine whether that data file is current and reflects the latest committed change or is in need of recovery.
4. Opens each redo log file *in sequence* and applies the information found there to each data file, as necessary, to bring each data file to the state where it contains all its committed transactions.

It is possible to recover a database by combining current data files with older data files restored from backup, since Oracle selectively applies changes from the redo log files to all files that are not current. However, this strategy succeeds only if an *unbroken* sequence of redo log files is available to bring the older data files up to date.

Database failure and recovery modes

If the database has lost a control file, the database is recovered by replacing the lost control file with a copy of a current control file.

If the database has lost one or more data files, the database is recovered by, first, replacing the lost data file(s) with backup copies and then using the redo logs (online or archived) to make the restored copies current. If the backup copies are restored to different locations from the original files they are intended to replace, you must use the ALTER DATABASE RENAME FILE command to tell the Oracle instance where the restored files are to be found.

If the database has lost the current online redo log, the database instance will halt when it attempts to commit more transactions. No data will have been lost, except that the latest transaction will not be committed and may need to be reentered when the database comes back up. However, the current online redo log file will have to be replaced and a backup of the database should immediately be taken.

If the database has lost any archived redo logs, the database instance will continue to function, as it will have no knowledge of the loss. However, the ability to recover the database in the event of a second media failure or file loss may be compromised. A fresh backup should be taken if the archived redo logs are lost.

For detailed information on the recovery of an Oracle database, refer to the *Oracle Backup and Recovery Guide* for your Oracle release.

Note: ESRI *strongly* recommends that you test your backup procedure by building a functioning database instance from backup files *before* an emergency arises. If you have just loaded your database, you should make a full backup and then recover the database from the backed up files to ensure the recovery process will work when you need it.

CHAPTER 8

Replication

An ArcGIS geodatabase can be modified to work with Oracle Advanced Replication. It is possible to create a virtual geodatabase by linking a remote read-only geodatabase to a master geodatabase site. The tables within the virtual geodatabase are replicated at each site, and changes made to the tables on the master site are propagated across the link to the read-only site. Thus, users viewing at either site see the changes after they have been propagated.

An ArcGIS geodatabase can be modified to work with Oracle read-only snapshot replication. This section discusses the setup and configuration of an ArcGIS geodatabase for read-only snapshot replication.

For an in depth discussion of Oracle replication, refer to the *Oracle Replication* and *Oracle Replication Management API Reference* for your Oracle release.

Setting up advanced replication

The replication of a geodatabase requires the installation of the Oracle Advanced Replication option and configuration of the replication environment on both sites.

Installing Oracle Advanced Replication

Using replication requires the installation of the Oracle Enterprise Server with the Advanced Replication option at both sites. For more information on the installation of Oracle, refer to the *Oracle Installation Guide* for your Oracle release.

Replication Manager

You should include the Replication Manager as part of your installation of the Oracle Enterprise Manager. The Replication Manager complements the scripts and stored procedures provided by ESRI, allowing you to monitor the status of the distributed Oracle database and the replicated tables at either site.

Adding Advanced Replication init.ora parameters

You should add the following parameters to the Oracle instance init.ora files:

```
job_queue_processes = 31
job_queue_interval = 15
```

```
distributed_transactions = 10
open_links = 4
```

Establishing NET8 communications between databases

Before you can create an Oracle database link, you must configure the NET8 connections between the databases. Database A must be able to connect to database B, and database B must be able to connect to database A.

If you did not configure the NET8 listener service with a TCP/IP protocol when you installed the Oracle Enterprise Server, use the NET8 Assistant to add and start the listener service. Perform this task at both sites. In this case you are configuring the server component of NET8 by updating the ORACLE_HOME/network/admin/listener.ora file.

Use the Net8 Assistant at each site to create a net service name alias that contains the connection parameters to the listener service of the remote site. In this case you are configuring the client component of the NET8 by updating the tnsnames.ora file. In a distributed database each remote database is both a client and a server of the other.

Converting LONG RAW data to LOB data

Oracle does not replicate the LONG RAW data type—the ArcSDE default storage for geometry. However, Oracle replicates the BLOB data type—an alternate ArcSDE storage for geometry. Therefore, to replicate data that has been stored in LONG RAW columns, you must convert the data to BLOB. The simplest way to convert an entire geodatabase that contains LONG RAW columns to one that contains only BLOB columns is to perform an ArcCatalog copy/paste into a newly created geodatabase.

Follow these steps:

1. Create a new Oracle database.

2. Create the tablespaces of geodatatabase including the sde tablespace.

3. Create the sde user and grant the privileges to it according to instructions provided in Chapter 2, ‘Essential configuring and tuning for Oracle’.

4. Edit the SDEHOME/etc/dbtune.sde file. In the DEFAULTS configuration keyword set the GEOMETRY_STORAGE parameter to SDELOB and the ATTRIBUTE_BINARY parameter to BLOB. Search the remainder of the dbtune.sde file and change all occurrences of the GEOMETRY_STORAGE parameter to SDELOB and the ATTRIBUTE_BINARY parameter to BLOB.

5. Run the sdesetupora* install operation. The sde metadata tables that are normally created with a LONG RAW data type are created instead with BLOB data type. To confirm this connect to Oracle as the sde user with SQL*Plus and describe the GDB_OBJECTCLASSES table. The EXTPROPS column should have been created with a blob data type.

```
$ sqlplus sde/sde

SQL*Plus: Release 8.1.7.0.0 - Production on Wed Jun 27 15:43:45 2001

(c) Copyright 2000 Oracle Corporation.  All rights reserved.
```

```
Connected to:
Oracle8i Enterprise Edition Release 8.1.7.0.0 - Production
JServer Release 8.1.7.0.0 - Production

SQL> describe gdb_objectclasses
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------
-
 ID                                        NOT NULL NUMBER(38)
 DATABASENAME                                       VARCHAR2(32)
 OWNER                                     NOT NULL VARCHAR2(32)
 NAME                                      NOT NULL VARCHAR2(160)
 ALIASNAME                                          VARCHAR2(160)
 MODELNAME                                          VARCHAR2(160)
 CLSID                                     NOT NULL VARCHAR2(38)
 EXTCLSID                                           VARCHAR2(38)
 EXTPROPS                                           BLOB
 SUBTYPEFIELD                                       VARCHAR2(32)
 DATASETID                                          NUMBER(38)

SQL>
```

6. Start ArcCatalog and connect to the geodatabase containing the LONG RAW data types and the new geodatabase that will contain data with BLOB data types. Right-click on an object from the 'LONG RAW' geodatabase and select Copy. Right-click the 'BLOB' geodatabase and select Paste, and the object is copied into the BLOB geodatabase. Do this for each object until all the objects are copied into the new geodatabase. To confirm that the data is being created with BLOB data types, connect to Oracle as a data user and describe one of its feature tables and examine the POINTS column.

```
$ sqlplus gisuser1/gisuser1

SQL*Plus: Release 8.1.7.0.0 - Production on Wed Jun 27 16:06:23 2001

 (c) Copyright 2000 Oracle Corporation.  All rights reserved.

Connected to:
Oracle8i Enterprise Edition Release 8.1.7.0.0 - Production
JServer Release 8.1.7.0.0 - Production

SQL> describe f1
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------
 FID                                       NOT NULL NUMBER(38)
 NUMOFPTS                                  NOT NULL NUMBER(38)
 ENTITY                                    NOT NULL NUMBER(38)
 EMINX                                     NOT NULL NUMBER(64)
 EMINY                                     NOT NULL NUMBER(64)
 EMAXX                                     NOT NULL NUMBER(64)
 EMAXY                                     NOT NULL NUMBER(64)
 EMINZ                                              NUMBER(64)
 EMAXZ                                              NUMBER(64)
 MIN_MEASURE                                        NUMBER(64)
 MAX_MEASURE                                        NUMBER(64)
 AREA                                      NOT NULL NUMBER(64)
 LEN                                       NOT NULL NUMBER(64)
 POINTS                                             BLOB

SQL>
```

Granting privileges to remote schema users

Oracle requires remote schema users to have privileges granted directly to them. Oracle will not honor privileges granted to remote users through a role. Therefore, if you normally grant privileges using roles such as CONNECT and RESOURCE to your schema owners, you will have to instead grant the privileges assigned by these roles to the schema users. If you fail to do so, Oracle will not be able to successfully execute the CREATE SNAPSHOT statement. For the correct method of assigning user privileges refer to Chapter 2.

Oracle 8*i* users require patch 1721870

If you are using Oracle 8*i* servers you must upgrade to Oracle 8.1.7 and install patch 1721870 which can be downloaded from Oracle. Without this patch you will not be able to create more than 256 snapshots at any given master site.

Setting up read-only snapshot replication

Read-only snapshot replication allows you to link a snapshot site to a master site and receive from the master site. Users connected to the read-only snapshot site cannot change the tables that have been replicated. Only users connected to the master site are able to edit the replicated tables. All of the tables listed in the SDE.TABLE_REGISTRY and the SDE.LAYERS tables are replicated. Only the tables owned by the SDE user that need to store the changes made to the geodatabase are replicated. The remainder of the sde user's tables store the changes of a user session and cannot be replicated because users edit these tables at both the master site and the read-only snapshot site.

Duplicating the tables of a geodatabase

As an option, Oracle will copy the rows of a table from the master definition site to the snapshot site when the snapshots are created at the snapshot site. Copying the rows of the large tables of a GIS database can require a lot of network I/O. Therefore, the stored procedures rep_util_snapshot.create_sde_repobjects and rep_util_snapshot.create_user_repobjects create the snapshots with the 'on prebuilt table without reduced precision' clause. When the snapshots are created Oracle compares the definitions of the tables to ensure that they are the same.

Duplicating the data using the Oracle export/import utilities is usually more efficient than having the large tables of your GIS database traverse the network when the snapshots are created. To duplicate an Oracle database in its entirety, use the database export option. Otherwise, export the schema of each user that owns data to replicated. If you choose to duplicate schemas, rather than the entire Oracle database, you must duplicate the schema of the sde user that owns the geodatabase metadata tables.

Creating the replication environment

In the snapshot_replication folder the rossetup.sql SQL script installs and executes the stored procedures of the reputil_master and reputil_snapshot packages. The stored procedures of these packages are installed and executed as the sys user, so you need to be the database administrator to run the rossetup.sql script.

Updating rossetup.sql

Before you run the rossetup.sql script you must update the variables that define your distributed database environment.

user0

A data user is an Oracle user that owns registered tables in the geodatabase you are about to replicate. If you have more than one data user, you will have to define and assign values to additional variables such as user1 and user2.

user_password0

The data user's password is needed to connect as the data user to create the snapshot logs at the master site.

sys_password

The sys password is needed to connect as the sys user. The password for the sys user is assumed to be the same at both the master site and snapshot site.

The sys user creates and owns the rep_util_master package at the master site and the rep_util_snapshot package at the snapshot site. The packages contain stored procedures and functions designed for the setup of read-only snapshot replication. The sys user grants execute privileges on the packages to the public.

The sys user creates the t_constraints and t_constraint_list types used by the rep_util_master.create_sde_pkeys stored procedure that creates primary keys on SDE tables.

system_password

The password of the system user is needed to connect as the system user. The system user creates the replication administration user at both the master site and snapshot site. The password is assumed to be the same at both sites.

sde_password

The password of the sde user is needed to connect as the sde user. The sde user creates primary keys and snapshot logs on its tables.

master_site

The Oracle NET8 network service name used by users at the snapshot site to connect to the master site. Review 'Establishing NET8 communications between databases' above to establish and test your network service names.

snapshot_site

The Oracle NET8 network service name used by users at the master site to connect to the snapshot site. Review 'Establishing NET8 communications between databases' above to establish and test your network service names.

default_tablespace

The default tablespace that will be used for the creation of the replication administration users. The default tablespace is assumed to be the same at both the master site and snapshot site.

temporary_tablespace

The temporary tablespace that will be used for the creation of the replication administration users. The temporary tablespace is assumed to be the same at both the master site and snapshot site.

masdef_site_gdb_name

The master site's global database name that can be found with the query:

```
select * from global_name
```

using SQL*Plus while connected to the master site's database.

snapshot_site_gdb_name

The snapshot site's database name that can be found with the query:

```
select * from global_name
```

using SQL*Plus while connected to the snapshot site's database.

group_name

The replication group is created to hold the replicated objects of the geodatabase. The objects include the metadata tables of the SDE user with the exception of the tables that support log files and user session data and all the business tables, delta tables, and feature class tables of the users that own schema within the geodatabase.

Running rossetup.sql

Once you have updated the variables with your replication environment, connect to the sys user with SQL*Plus and run rossetup.sql.

```
$ sqlplus sys/manager

SQL*Plus: Release 8.1.7.0.0 - Production on Mon Jun 25 10:26:46 2001
 (c) Copyright 2000 Oracle Corporation.  All rights reserved.

Connected to:
Oracle8i Enterprise Edition Release 8.1.7.0.0 - Production
JServer Release 8.1.7.0.0 - Production

SQL> @rossetup
```

The rossetup.sql script performs the following tasks:

1. Connects as the sys user at the master site.
2. Creates the t_constraints and t_constraint_list objects for use by the rep_util_master.create_sde_pkeys stored procedure that creates the primary keys on SDE tables that lack them. Read-only replication requires that all replicated tables have primary keys.
3. Creates the rep_util_master package of stored procedures and functions that the rossetup.sql script uses to set up read-only snapshot replication at the master site.
4. Connects as the sde user at the master site.
5. Creates the primary keys on the sde tables that need them.
6. Connects as the data user.
7. Creates the primary keys on the data user's tables that need them.
8. Connects as the system user.

9. Creates the replication administration users.
10. Grants privileges to the replication administration users.
11. Registers the replication administration repadmin as the receiver at the master site.
12. Creates the public database link to the snapshot site.
13. Connects as the repadmin user at the master site.
14. Creates a private database link to the master site.
15. Schedules the purge job.
16. Schedules the push job to the snapshot site.
17. Creates the replication group.
18. Adds the sde tables as replication objects to the replication group.
19. Connects as the sde user at the master site.
20. Creates the snapshot logs on the sde tables that have been replicated.
21. Connects as the repadmin user at the master site.
22. Adds the users' tables to the replication group.
23. Connects as the data user.
24. Creates the snapshot logs on the data user's table.
25. Connects as the sys user at the snapshot site.
26. Creates the remote_repobjects public synonym.
27. Creates the rep_util_snapshot package, which contains stored procedures and functions that the rossetup.sql SQL script uses to set up replication at the snapshot site.
28. Connects as the system user at the snapshot site.
29. Create the replication administration users at the snapshot site.
30. Creates the public database link to the master site.
31. Connects as the snapadmin user at the snapshot site.
32. Creates the database link to the master site.
33. Schedules the purge job.
34. Schedules the push job to the master site.

35. Connects as the propagator user at the snapshot site.
36. Creates the private database link that connects as the repadmin user at the master site.
37. Connects as the sde user at the snapshot site.
38. Creates the private database link that connects as the proxy_refresher to the master site.
39. Connects as the sys user at the snapshot site.
40. Re-creates the remote_repobjects synonym of the dba_repobjects table at the master site.
41. Connects as the snapadmin user at the snapshot site.
42. Creates the snapshot replication group.
43. Adds the sde tables to the snapshot replication group.
44. Connects as the data user to the snapshot site.
45. Creates the private database link that connects as the proxy_refresher to the master site.
46. Connects as the snapadmin user to the snapshot site.
47. Adds the data user's tables to the snapshot replication group.
48. Connects as the repadmin user at the master site.
49. Resumes replication.

Once replication resumes, you may edit the geodatabase while connected to either of the master sites. The master site periodically refreshes the snapshot site with the committed changes.

Editing a replicated database with ArcMap

Edits made to versioned feature classes, tables, and geometric networks are not visible to the distributed site until the edits have been saved. The saved edits are visible at the snapshot site after it has been refreshed. You can refresh the version to see changes made at a remote site. To refresh the current version of a geodatabase, first enable the Versioning toolbar. Click Tools and Customize and check Versioning. From the Versioning toolbar click the Refresh button.

Using the Oracle replication manager

The Oracle Replication Manager is a graphical user interface tool that is installed with the Oracle Enterprise Manager. It is very useful for monitoring and altering your replication environment. It is particularly useful for monitoring the progress of the rossetup.sql script during snapshot replication setup.

It is possible to use the replication manager to add additional tables to the replication group.

If you decide to generate replication for a business table that has been registered as multiversioned, you must replicate the adds and deletes tables. Query SDE.TABLE_REGISTRY to obtain REGISTRATION_ID, which forms the table names of the adds and deletes tables.

```
SELECT REGISTRATION_ID
FROM SDE.TABLE_REGISTRY
WHERE TABLE_NAME = 'ROADS'
AND OWNER = 'DAN';
```

If the query for the registration ID returns 30, the adds table is A30, and the deletes table is D30.

If the business table contains a spatial column, you must replicate the feature table and the spatial index table. To determine the names of these tables, query the SDE.LAYERS table for the layer ID of the business table's spatial column.

```
SELECT LAYER_ID
FROM LAYERS
WHERE TABLE_NAME = 'ROADS'
AND OWNER ='DAN'
```

If the query for the layer ID returns 10, the feature table is F10, and the spatial index table is S10.

APPENDIX A

Estimating the size of your tables and indexes

The formulas provided in this appendix provide approximations of the actual sizes of the Oracle tables and indexes created by ArcSDE.

The business table

The following calculation of business table size is based on the calculations provided in *Oracle Administrator's Guide* for your Oracle release.

1. Determine the data block size. Do this by connecting to the database as the DBA and querying the db_block_size parameter stored in the v$parameter file.

```
$ sqlplus system/<password>

SQL> select value from v$parameter where name = 'db_block_size';
```

2. Remove the header from the data block space.

```
space after headers(sph) = db_block_size - 86 - ((INITRANS -1) * 24)
```

3. Calculate the available space by removing the percent free from the space after headers.

```
available space = ceil(sph * (1 - PCTFREE / 100)) - 4
```

4. Reduce the available space further by 5 percent.

```
available space = (available space * 0.95)
```

5. Calculate the row size. You may use the formulas described in Step 3 of Appendix A in the *Oracle Administrator's Guide* to calculate the row size. However, if you can create the table and load about 100 sample records into it, you can obtain the most accurate estimate of row size by analyzing the table and then querying user_tables for the table's avg_row_len.

```
$ sqlplus <username>/<password>

SQL> create table <table_name> . . .
SQL> insert into table <table_name> . . .
SQL> analyze table <table_name> compute statistics;
SQL> select avg_row_len from user_tables where table_name = <table_name>;
```

6. Compute the number of rows that can be stored by each data block.

```
rows per data block = floor(available space / row size)
```

7. Compute the number of data blocks required to store the table. You will need to estimate the number of rows you expect the table to contain.

```
total data blocks = total rows / rows per data block
```

8. Compute tablespace required to store the table in bytes.

```
tablespace size = data block size * total data blocks
```

The feature table

Use the steps listed in the business table size calculation to calculate the size of the feature table. If you cannot load a sample feature class, use the following formula to calculate the row size:

row size = feature header + (average points per geometry) * 0.586 * coordinate factor

The feature header varies depending on the type of geometry the feature class stored and whether z-values and/or measures are stored. Table A.1 lists the feature header sizes for the various combinations.

	x,y	x,y,z	x,y,m	x,y,z,m
points	85	103	121	139
lines	93	111	129	147
polygons	101	119	137	155

Table A.1 Feature headers vary according to the geometry type of the feature class and whether z-values and/or measures are stored along with each x,y coordinate pair. The geometry type is listed by row, while the coordinate type is listed by column.

The coordinate factor varies depending on whether it will store z-values and/or measures. Table A.2 lists the possible coordinate factors.

Coordinate type	Coordinate factor
x,y	8
x,y,z	12
x,y,m	12
x,y,z,m	16

Table A.2 Coordinate factors vary depending on the type of coordinates stored.

If the features store annotation, the row size will be larger. Calculating the space required to store annotation can be rather complex with the annotation text, leader line, placement, and metadata. Most annotation will occupy between 100 and 300 bytes of space per feature. Therefore, increase the row size by 200 bytes if your features include annotation.

The spatial index table

You can use the steps listed in calculating the business table to calculate the size of the spatial index table except for the row size. The row size for the spatial index table is always 43 bytes.

However, the size of the spatial index table can be fairly accurately estimated given the number of records in a feature class's business table.

spatial index table size = number of business table records * 1.3

This formula assumes that the spatial index table contains an average of 1.3 grid cells for each feature.

Use steps 6, 7, and 8, listed in the business table, to calculate the amount of space required to store the spatial index table.

The version delta tables

When you register a business table as multiversioned, three tables are created to maintain the deltas: the adds table, the deletes table, and the equivalency table. Depending on how the versions are managed, these tables may remain relatively small, or they may become quite large. The adds table receives one record for each record that is added to a business table version. The deletes table receives a record each time a record is deleted from a business table version. The equivalency table receives records each time versions are merged. The function of the equivalency table is rather complex, but its main role is to maintain the integrity of versioned records that have been copied between versions. As a result, it can grow exponentially depending on the number of merges that occur.

The most difficult part of estimating space for the delta tables is trying to predict how many rows they will contain. Partly it will depend on how many versions the table is involved in. In other words, how many records are added or deleted throughout the life of a version will affect the size of the adds and deletes tables. The number of times the version of one work order is copied to another will affect the size of the equivalency table.

The size of the delta tables also depends on how often records are removed. These tables shrink only when the states preceding the level 0 version are compressed. This occurs only after a version branching directly off the root of the version tree completes and is removed from the system. The compression of states that follows will cause the changes of the states between the level 0 version and the next version following the one removed to be written to the business table and deleted from the delta tables.

For most applications the delta tables should not be expected to grow beyond 10,000 records. However, there are applications where several versions of the database must be maintained over a long period. In these cases as many as 30,000 records per delta table can be expected.

The adds table

The table definition of the adds table is identical to that of the business table with the addition of the SDE_STATE_ID column. Therefore, to compute the space required to store the adds table, use the calculations for the business table and increase the row size of the adds table by 4 bytes to account for the addition of the SDE_STATES_ID column.

The record count for the adds table will be an estimate of the number of records added to the versioned business table after it has been registered as multiversioned. Records are removed from the adds table only when a version next to the 0 version is removed and its states are compressed.

The deletes table

The record count depends on the number of records deleted from the business table after it was registered as multiversioned. Records are removed from the deletes table only when a version next to the 0 version is removed and its states are compressed.

Calculate the size of the deletes table with the following formula:

deletes table size = 5.4 * number of rows.

For more information on versioning, refer to *Building a Geodatabase*.

The network tables

There are 10 standard network tables created whenever you create a geodatabase network class. Optionally, the application designer may also create weights for the network, in which case two tables are created for each weight.

The size of the network tables depends on the number of rows in the base tables. A network class is basically the logical combination of point and linestring feature classes. In network terms, linestring features are referred to as junctions.

The first step in determining the size of the network tables is to establish the total number of edges and junctions.

The total number of edges is calculated by adding up all the linestring features in the network.

A junction must exist at the endpoint of each linestring. If a point feature does not exist at the endpoint of a linestring, a logical junction is created within the network tables. For endpoints of two or more linestrings that share the same coordinate position, only one junction exists. So some of the endpoints will share the same coordinate position, and some will not. Some will have a point feature at the endpoint, and some will not. As a rough guide assume that half of your linestring endpoints share the same coordinate position and that the total number of junctions required by the network is equal to the number of edges.

Once you have the junction and edge totals, you can determine the size of the network tables using the following formulas.

Description table (N_<n>_DESC)

The number of rows in the description table is equal to the sum of the junctions and edges.

The size of the description table is calculated as

size in bytes = number of rows * 7.4

Edge description table (N_<n>_EDESC)

The number of rows in the edge description table is calculated as

number of rows = number of edges / 1638

The size of the edge description table is calculated as

size in bytes = number of rows * 6250

Edge status table (N_<n>_ESTATUS)

The number of rows in the edge status table is calculated as the

number of rows = number of edges / 65536

size in bytes = number of rows * 10000

Edge topology table (N_<n>_ETOPO)

The number of rows in the edge topology table is calculated as the

number of rows = number of edges / 4096

The size of the edge topology table is calculated as

size in bytes = number of rows * 6336

Flow direction table (N_<n>_FLODIR)

The number of rows in the flow direction table is calculated as

number of rows = number of edges / 65536

The size of the flow direction table is calculated as

size in bytes = number of rows * 40960

Junction description table (N_<n>_JDESC)

The number of rows in the junction description table is calculated as

number of rows = number of junctions / 1638

The size of the junction description table is calculated as

size in bytes = number of rows * 24986

Junction status table (N_<n>_JSTATUS)

The number of rows in the junction status table is calculated as

number of rows = number of junctions / 65536

The size of the junction status table is calculated as

size in bytes = number of rows * 40960

Junction topology table (N_<n>_JTOPO)

The number of rows in the junction topology table is calculated as

number of rows = number of junctions / 2048

The size of the junction topology table is calculated as

size in bytes = number of rows * 24865

Junction overflow topology table (N_<n>_JTOPO2)

The number of rows in the junction overflow topology table is calculated as

number of rows = number of junctions / 20480

The size of the junction overflow topology table is calculated as

size in bytes = number of rows * 29257

Property table (N_<n>_PROP)

The number of rows in the network properties table is always 10.

number of rows = 10

The size of the network properties table is approximately 250 bytes.

size in bytes = 250

The network properties table will always occupy one data block.

Edge weight table (N_<n>_E<w>)

The number of rows in an edge weight table is calculated as

number of rows = number of edges / 2048

The size of the junction overflow topology table is calculated as

size in bytes = number of rows * 24865

Junction weight table (N_<n>_J<w>)

The number of rows in the junction overflow topology table is calculated as

number of rows = number of junctions / 2048

The size of the junction overflow topology table is calculated as

size in bytes = number of rows * 24865

The raster data tables

Four raster data tables are created whenever a user adds a raster column to a business table. A row is added to the sde raster_columns system table for each raster column created in the database. Each new raster column is given a unique ID called a rastercolumn_id. The rastercolumn_id is used in the naming of the raster data tables.

Raster data schema consists of four tables and their associated indexes. The four tables and their associated indexes are:

1. The raster table (SDE_RAS_<raster_column_id>) stores one record for each raster. The number of raster table rows is always less than or equal to the number of business table rows. If each row of the business table has an image stored in the raster tables, then the number of rows is equal. If some of the business table rows do not have an image associated with them, then the number for raster tables is less.

The raster table has a single unique index on the raster_id column (SDE_RAS_<raster_column_id>_UK).

2. The raster bands table (SDE_BND_<raster_column_id>) stores one record for each raster band. The one-to-many relationship between the raster table and the raster bands table is maintained by the primary/foreign key raster_id column. The number of rows in the raster bands table can be determined by multiplying the number of image bands by the number of raster table rows.

   The raster bands table has two unique indexes, one on the rasterband_id column (SDE_BND_<raster_column_id>_UK1) and the other a composite of the raster_id and sequence_nbr columns (SDE_BND_<raster_column_id>_UK2).

3. The data stored in the raster auxiliary table (SDE_AUX_<raster_column_id>) is optional. Therefore, a zero-to-many relationship exists between the raster bands table and the raster auxiliary table. The tables are joined on the primary/foreign key relationship of the rasterband_id column. The raster auxiliary table stores one record of metadata about each band. For example, if the image has a color map associated with it, one record will be added to the auxiliary table for each record in the raster band table. If you elect to create statistics for the raster, one record will be added for each record in the raster bands table. If you are not sure how much metadata will be stored for each raster, you don't need to be overly concerned since the size of this table is very small.

   The raster auxiliary table has one unique index, a composite of the type, and rasterband_id columns (SDE_AUX_<raster_column_id>_UK).

4. The raster blocks table (SDE_BLK_<raster_column_id>) stores the raster band pixels as blocks defined by the block's dimensions. A one-to-many relationship exists between the raster bands table and the raster blocks table. The number of rows in the raster blocks table can be estimated by first determining the pixels required to complete a block, dividing this number into the total number of pixels within a band, and multiplying the result by the number of bands in the raster. If you have elected to store a resolution pyramid, multiply the result by 1.33 to account for the resolution pyramid blocks.

   The raster blocks table has one unique index—a composite of the rasterband_id, rrd_factor, row_nbr, and col_nbr columns (SDE_BLK_<raster_column_id>_UK).

Raster table (SDE_RAS_<raster_column_id>)

On the average, 2,400 raster table rows fit in a 16 KB Oracle data block, given an Oracle data block percent free of 10 percent and initial transactions of 4. To roughly estimate the number of data blocks required to store the rows of the raster table, divide the total number of rows you expect the table to have by 2,400.

Raster bands table (SDE_BND_<raster_column_id>)

On the average, 190 raster band table rows fit within a 16 KB Oracle data block, given an Oracle data block percent free of 10 percent and initial transactions of 4. To roughly estimate the number of data blocks required to store the rows of the raster bands table, divide the total number of rows you expect the table to have by 190.

Raster auxiliary table (SDE_AUX_<raster_column_id>)

On the average, six raster auxiliary table rows fit within a 16 KB Oracle data block, given an Oracle data block percent free of 10 percent and initial transactions of 4. To roughly estimate the number of data blocks required to store the rows of the raster auxiliary table, divide the total number of rows you expect the table to have by 6.

Raster blocks table (SDE_BLK_<raster_column_id>)

The size of the raster blocks table varies depending on whether you have built a resolution pyramid and whether an image compression method has been applied. The addition of the pyramid will increase the number of rows in the raster blocks table from the base level by a factor of one third.

The pixel depth affects the row size of the raster block. Single bit pixels require less storage space than do 4-bit pixels, which in turn require less space than 8-bit pixels, and so on.

The pixel dimension of the raster blocks also affects the row size of the raster blocks. Larger dimensions require more space. For example, the storage space of a 64 x 64 pixel raster block requires a quarter of the space of a 128 x 128 raster block.

Compression reduces the total size of the stored image in a manner similar to the algorithmic compression of ASCII files. The software interrogates rows of pixels searching for equal valued groups and storing a single value and the pixel count, rather than a string of equal values. Compression reduces the row size of the raster blocks.

Compression results vary from image to image; however, Table A.3 provides a general guideline for the number of Oracle 16 KB data blocks required to store raster block table rows, given the image's pixel depth. To compute the total number of 16 KB Oracle data blocks required, multiply the expected number of rows in the raster block's table by the appropriate Oracle data blocks per row factor in Table A.3. If you created the database with an Oracle data block size of 8, multiply the result by 2.

Table A.1 Number of 16 KB Oracle data blocks required to store a row of the raster blocks table for a given pixel depth.

	Pixel Depth			
Compression	**1-bit**	**8-bit**	**16-bit**	**32-bit**
None	0.17	1.5	2.6	4.5
LV77	0.01	0.55	0.78	1.08

The indexes

All of the indexes, with the exception of the spatial index table's S<n>_IX1, index single-integer columns. The spatial index table's S<n>_IX1 indexes all of the columns of the table. Since the definition of the indexes is fixed, the size of the indexes can be based on the number of rows they index. The space for all indexes, except the spatial index table's S<n>_IX index, can be calculated using the following formula:

adjusted size = 500 rows per data block * (1 - PCTFREE)

total data blocks = table rows / adjusted size

tablespace size = total data blocks * data block size

For the S<n>_IX1 index the formula is

adjusted size = 150 rows per data block * (1 - PCTFREE)

total data blocks = table rows / adjusted size

tablespace size = total data blocks * data block size

APPENDIX B

Storing raster data

A raster is a rectangular array of equally spaced cells that, 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.

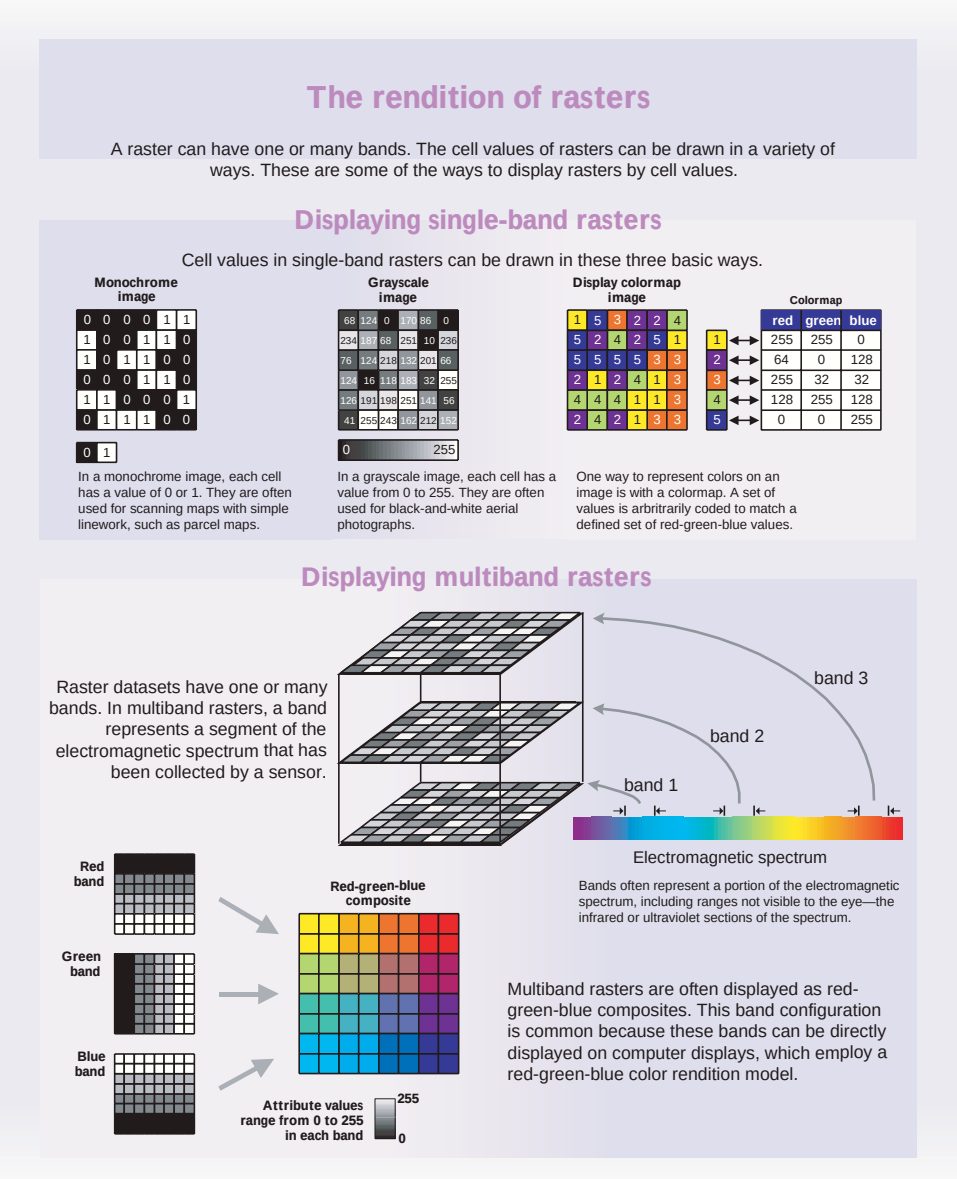
You are probably familiar with raster formats such as tagged image file format (TIFF), Joint Photographic Experts Group (JPEG), and Graphics Interchange Format (GIF) that your Internet browser renders. These rasters are composed ofone or more bands. Each band is segmented into a grid of square pixels. 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 C, '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 table. ArcSDE joins the raster band 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, color maps, or bitmasks, 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 table. ArcSDE joins the two tables on this primary/foreign key reference when accessing a raster band's metadata.



The raster blocks 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 any specified compression method, 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 an Oracle data block. For Oracle databases, storing raster data should be created with a 16 KB Oracle data block size. See Appendix A, 'Estimating the size of your tables and indexes', for more information on estimating the size of your raster tables and indexes.

Using a compression method, such as lossless lz77, almost always results in an improvement in performance. The savings in disk space and network I/O generally offset the additional CPU cycles required for the application to decompress the image.

The raster blocks 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 band.

ArcSDE populates the raster blocks table according to a declining resolution pyramid. The number of levels specified by the application determines the height of the pyramid. 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 additional levels of the pyramid increase the number of raster block table rows by one third. However, since it is possible for the user to specify the number of levels, the true apex of the pyramid may not be obtained, limiting the number of records added to the raster blocks table.

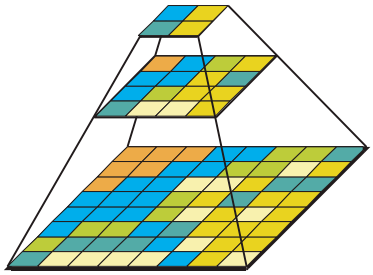


Figure B.1 When you build a pyramid, more rasters are created by progressively downsampling the previous level by a factor of two until the apex is reached. As the application zooms out and the raster cells grow smaller than the resolution threshold, ArcSDE selects a higher level of the pyramid. The 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 Oracle'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 band table by rasterband_id, stores the metadata of each raster band. The rasterbands_id also joins the raster band 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.

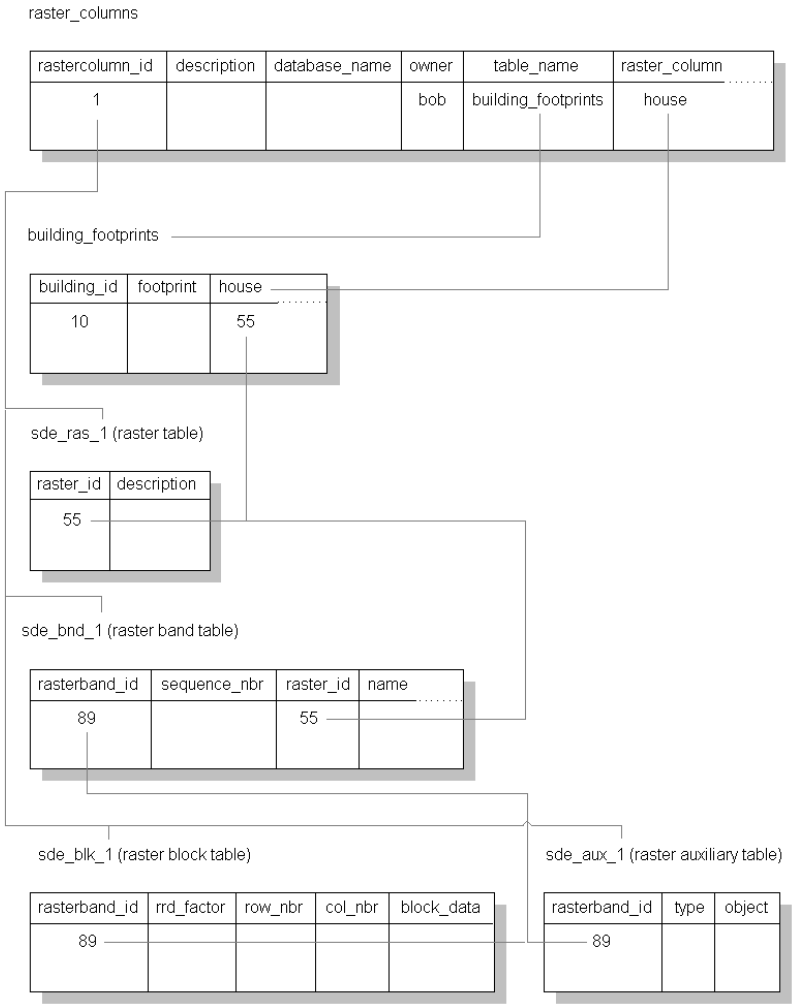


Figure B.2 When ArcSDE adds a raster column to a table, it records that column in the sde user's raster_columns table. The rastercolumn_id table is used in the creation of the table names of the raster, raster band, raster auxiliary, and raster blocks table.

The sections that follow describe the schema of the tables associated with the storage of raster data. Refer to Figure B.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	NUMBER(38)	NOT NULL
description	VARCHAR2(65)	NULL

database_name	VARCHAR2(32)	NULL
owner	VARCHAR2(32)	NOT NULL
table_name	VARCHAR2(160)	NOT NULL
raster_column	VARCHAR2(32)	NOT NULL
cdate	NUMBER(38)	NOT NULL
config_keyword	VARCHAR2(32)	NULL
minimum_id	NUMBER(38)	NULL
base_rastercolumn_id	NUMBER(38)	NOT NULL
rastercolumn_mask	NUMBER(38)	NOT NULL
srid	NUMBER(38)	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)—Field is always NULL for Oracle.
- owner (SE_STRING_TYPE)—The owner of the 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 Oracle 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.
- srid (SE_INTEGER_TYPE)—The spatial reference ID (srid) is a foreign key reference to the SPATIAL_REFERENCES table. For images that can be georeferenced, the srid references 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	NUMBER(38)	NOT NULL
building_footprint	NUMBER(38)	NOT NULL
house_picture	NUMBER(38)	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 Oracle database, stores a record for each image stored in a raster column. The 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	NUMBER(38)	NOT NULL
raster_flags	NUMBER(38)	NULL
description	VARCHAR2(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 a 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	NUMBER(38)	NOT NULL
sequence_nbr	NUMBER(38)	NOT NULL
raster_id	NUMBER(38)	NOT NULL
name	VARCHAR2(65)	NULL
band_flags	NUMBER(38)	NOT NULL
band_width	NUMBER(38)	NOT NULL
band_height	NUMBER(38)	NOT NULL
band_types	NUMBER(38)	NOT NULL
block_width	NUMBER(38)	NOT NULL
block_height	NUMBER(38)	NOT NULL
block_origin_x	NUMBER(64)	NOT NULL
block_origin_y	NUMBER(64)	NOT NULL
eminx	NUMBER(64)	NOT NULL
eminy	NUMBER(64)	NOT NULL
emaxx	NUMBER(64)	NOT NULL
emaxy	NUMBER(64)	NOT NULL
cdate	NUMBER(38)	NOT NULL
mdate	NUMBER(38)	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 the 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 bottommost 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 an Oracle 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	NUMBER(38)	NOT NULL
rrd_factor	NUMBER(38)	NOT NULL
row_nbr	NUMBER(38)	NOT NULL
col_nbr	NUMBER(38)	NOT NULL
block_data	LONG RAW or BLOB	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 band's 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 bitmasks 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	NUMBER(38)	NOT NULL
type	NUMBER(38)	NOT NULL

```
object          LONG RAW or BLOB      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.

Creating a raster catalog

An image catalog allows you to group many images by simply listing them in a table. ArcGIS clients like ArcCatalog or ArcMap display the images as a group by reading the entries in the image catalog table. The table must contain the five columns: IMAGE, XMIN, YMIN, XMAX, and YMAX.

The IMAGE column contains the fully qualified name of the image, while the remaining four columns describe the image's extent. 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
TOPO1	322169.2	4094888	411902.3	4151957
TOPO2	323452.5	4150579	412586.2	4207658
TOPO3	233002.1	4096472	323537.9	4154628
TOPO4	234912.9	4151847	324666.3	4210094
TOPO5	676712.7	4090675	770814.6	4154551
TOPO6	675530.7	4146368	769195.7	4210173
TOPO7	324599.5	4206098	413235.3	4263136
TOPO8	325340.9	4261470	413692.2	4318609
TOPO9	236598.8	4207336	325702	4265705
TOPO10	239302.9	4262352	327095.4	4321168
TOPO11	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.

APPENDIX C

ArcSDE compressed binary

ArcSDE uses a compressed binary format to store geometry in either an Oracle binary LONG RAW or BLOB data type. ArcSDE stores compressed binary spatial data in the POINTS columns of the feature table. Compressing the geometry offers efficient storage and retrieval of spatial data by reducing the size of the geometry. Compressed binary stored in the LONG RAW data type is the default storage format for ArcSDE feature classes.

Compressed binary

ArcSDE verifies the geometry, compresses it, and sends it to the Oracle instance, where it is inserted into a feature table in compressed binary format. 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 schema

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	NULL?
fid	NUMBER(38)	NOT NULL
numofpts	NUMBER(38)	NOT NULL
entity	NUMBER(38)	NOT NULL
eminx	NUMBER(64)	NOT NULL
eminy	NUMBER(64)	NOT NULL
emaxx	NUMBER(64)	NOT NULL
emaxy	NUMBER(64)	NOT NULL
eminz	NUMBER(64)	NULL
emaxz	NUMBER(64)	NULL
min_measure	NUMBER(64)	NULL
max_measure	NUMBER(64)	NULL

area	NUMBER(64)	NOT NULL
len	NUMBER(64)	NOT NULL
points	LONG RAW or BLOB	NULL
anno_text	VARCHAR2(256)	NULL

Feature table schema

The feature table stores the geometry, annotation, and CAD in the POINTS column. The POINTS column may be defined as either LONG RAW or BLOB depending on the setting of the GEOMETRY_STORAGE DBTUNE storage parameter. Set the GEOMETRY_STORAGE DBTUNE storage parameter to SDEBINARY if you want to store the compressed binary spatial data in a column defined as LONG RAW; otherwise, set the storage parameter to SDELOB if you want to store the compressed binary spatial data in a column defined as BLOB.

For an expanded discussion of the GEOMETRY_STORAGE DBTUNE storage parameter, see Chapter 3, 'Configuring DBTUNE storage parameters'.

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 spatial column (e.g., point, linestring)
- numofpts (SE_INTEGER_TYPE)—the number of points defining the geometry
- eminx, eminy, emaxx, emaxy (SE_FLOAT_TYPE)—the envelope of the geometry
- eminz (SE_FLOAT_TYPE)—the minimum z-value in the geometry
- emaxz (SE_FLOAT_TYPE)—the maximum z-value in the geometry
- min_measure (SE_FLOAT_TYPE)—the minimum measure value in the geometry
- max_measure (SE_FLOAT_TYPE)—the maximum measure value in the geometry
- area (SE_FLOAT_TYPE)—the area of the geometry
- len (SE_FLOAT_TYPE)—the length or perimeter of the geometry
- points (SE_SHAPE_TYPE)—contains the byte stream of point coordinates that define the geometry
- anno_text (SE_STRING_TYPE)—contains the feature annotation string

NAME	DATA TYPE	NULL?
sp_fid	NUMBER(38)	NOT NULL
gx	NUMBER(38)	NOT NULL
gy	NUMBER(38)	NOT NULL
eminx	NUMBER(64)	NOT NULL
eminy	NUMBER(64)	NOT NULL
emaxx	NUMBER(64)	NOT NULL
emaxy	NUMBER(64)	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—contains the unique ID that joins the feature table to the business table
- gx/gy—defines the feature's extent in grid cells
- eminx/eminy/emaxx/emaxy—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 class, like 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 least 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 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 completes. 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.

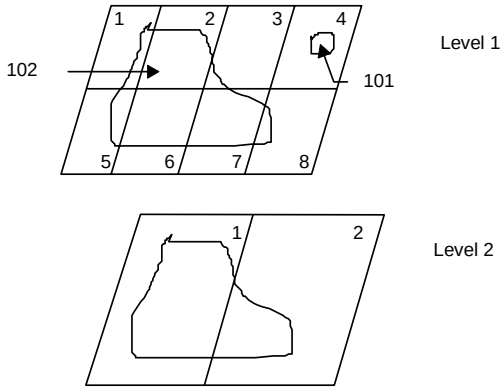


Figure C.1 Shape 101 is indexed on grid level 1, while shape 102 is indexed on grid level 2, where it is in only two grid cells.

Spatial queries and the spatial index

Spatial queries, such as finding all the lakes within a state boundary, use the spatial index. The 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's 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 to perform 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's where clause to further refine the 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 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 the spatial index table while maintaining the proficiency of the index because the majority of the features can be referenced by less than one or two grid cells.

- 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 index statistics 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. A geometry will not generate more than four records if a higher grid level is available.

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. There are 35,682 geometries indexed with four grid records or fewer, 752 geometries indexed with more than four grid records, 87 geometries indexed with more than 10, 17 geometries with more than 25, and three geometries with more than 50. Percentage values below each column show how the geometries are dispersed through the eight groups.

Creating tables with compressed binary schema

The geometry storage format for ArcSDE feature classes defaults to LONG RAW compressed binary. If you always store your geometry in this format, you do not need to adjust the geometry storage format.

If you wish to have a mix of geometry types in your schema, add configuration keywords to the DBTUNE table with the desired geometry storage format and assign the configuration keywords to the feature classes.

The DBTUNE table storage parameter GEOMETRY_STORAGE defines the geometry storage format of a feature class. The GEOMETRY_STORAGE value for the default, LONG RAW compressed binary, is SDEBINARY. In the following example, a dbtune file has a PARCELS configuration keyword that contains the value SDEBINARY.

```
##PARCELS
GEOMETRY_STORAGE            SDEBINARY
<other parameters>
END
```

Additional storage parameters precisely define the storage configuration of the parcel's feature class.

```
##PARCELS

GEOMETRY_STORAGE    "SDEBINARY"
B_STORAGE           "TABLESPACE btabsp STORAGE (FREELISTS 4 INITIAL 500M
                    NEXT 100M MINEXTENTS 1 MAXEXTENTS 50 PCTINCREASE 0)"
B_INDEX_ROWID       "TABLESPACE bindex_tsp STORAGE (FREELISTS 4 INITIAL 100M
                    NEXT 25M MINEXTENTS 1 MAXEXTENTS 50 PCTINCREASE 0)"
B_INDEX_SHAPE       "TABLESPACE bindex_tsp STORAGE (FREELISTS 4 INITIAL 100M
                    NEXT 25M MINEXTENTS 1 MAXEXTENTS 50 PCTINCREASE 0)"
A_STORAGE           "TABLESPACE atabsp STORAGE (FREELISTS 4 INITIAL 100M
                    NEXT 25M MINEXTENTS 1 MAXEXTENTS 50 PCTINCREASE 0)"
A_INDEX_ROWID       "TABLESPACE aindex_tsp STORAGE (FREELISTS 4 INITIAL 50M
                    NEXT 10M MINEXTENTS 1 MAXEXTENTS 50 PCTINCREASE 0)"
A_INDEX_SHAPE       "TABLESPACE aindex_tsp STORAGE (FREELISTS 4 INITIAL 50M
                    NEXT 10M MINEXTENTS 1 MAXEXTENTS 50 PCTINCREASE 0)"
A_INDEX_STATEID     "TABLESPACE aindex_tsp STORAGE (FREELISTS 4 INITIAL 50M
                    NEXT 10M MINEXTENTS 1 MAXEXTENTS 50 PCTINCREASE 0)"
D_STORAGE           "TABLESPACE atabsp STORAGE (FREELISTS 4 INITIAL 50M
                    NEXT 10M MINEXTENTS 1 MAXEXTENTS 50 PCTINCREASE 0)"
D_INDEX_STATE_ROWID "TABLESPACE dindex_tsp STORAGE (FREELISTS 4 INITIAL 50M
                    NEXT 10M MINEXTENTS 1 MAXEXTENTS 50 PCTINCREASE 0)"
D_INDEX_DELETED_AT  "TABLESPACE dindex_tsp STORAGE (FREELISTS 4 INITIAL 50M
                    NEXT 10M MINEXTENTS 1 MAXEXTENTS 50 PCTINCREASE 0)"
F_STORAGE           "TABLESPACE ftabsp STORAGE (FREELISTS 4 INITIAL 500M
                    NEXT 100M MINEXTENTS 1 MAXEXTENTS 50 PCTINCREASE 0)"
F_INDEX_FID         "TABLESPACE findex_tsp STORAGE (FREELISTS 4 INITIAL 50M
                    NEXT 10M MINEXTENTS 1 MAXEXTENTS 50 PCTINCREASE 0)"
F_INDEX_AREA        "TABLESPACE findex_tsp STORAGE (FREELISTS 4 INITIAL 50M
                    NEXT 10M MINEXTENTS 1 MAXEXTENTS 50 PCTINCREASE 0)"
F_INDEX_LEN         "TABLESPACE findex_tsp STORAGE (FREELISTS 4 INITIAL 50M
                    NEXT 10M MINEXTENTS 1 MAXEXTENTS 50 PCTINCREASE 0)"
S_STORAGE           "TABLESPACE stabsp STORAGE (FREELISTS 4 INITIAL 300M
                    NEXT 50M MINEXTENTS 1 MAXEXTENTS 50 PCTINCREASE 0)"
S_INDEX_ALL         "TABLESPACE sindex_tsp STORAGE (FREELISTS 4 INITIAL 100M
                    NEXT 50M MINEXTENTS 1 MAXEXTENTS 50 PCTINCREASE 0)"
S_INDEX_SP_FID      "TABLESPACE sindex_tsp STORAGE (FREELISTS 4 INITIAL 100M
                    NEXT 50M MINEXTENTS 1 MAXEXTENTS 50 PCTINCREASE 0)"
END
```

In this example, the storage schema is compressed binary; it defines the storage and location for the business table, adds table, deletes table, feature table, and spatial index table. Chapter 3, 'Configuring DBTUNE storage parameters', describes these storage parameters as well as many others.

Tuning LOB storage

Large Object (LOB) data types are used by Oracle to store large, unstructured datasets. ArcSDE uses this data type for large datasets that are processed by specialized ArcSDE algorithms.

LOB data may be stored *in-line*, meaning that LOB and non-LOB data from the same row in a table are stored in the same Oracle data block. Alternately, LOB data may be stored *out-of-line* meaning that the LOB data is stored separately from the rest of the row data in a special area set aside for it by the database. Oracle stores LOB data in-line or out-of-line depending upon the size of the LOB and the storage parameters associated with creation of the table holding the LOB data.

If the total size of a LOB datum plus the storage locator is less than 4000 bytes, and the ENABLE STORAGE IN ROW clause is specified, the LOB datum is stored in the same block as the rest of the row fields. If possible to achieve, this configuration will result in less I/O to read a single LOB datum

If the total size of a LOB datum plus the storage locator exceed 4000 bytes, or the DISABLE STORAGE IN ROW clause is specified, Oracle stores the LOB datum out-of-line. This configuration will result in two I/Os to fetch a single LOB datum—the first to fetch row data plus the LOB locator and a second to fetch LOB itself.

To create a new row with a LOB datum, Oracle effectively performs an INSERT of the new row storing the row data and the LOB locator followed by an UPDATE of the same row to add the actual LOB data. If several rows having LOB data are inserted in a single SQL statement, *all* INSERTs are performed before any UPDATE.

This has important consequences on the storage parameters chosen for a table holding LOB data. If, for example, you insert 9 rows with LOB data and Oracle INSERTs them into a single block, then performs the UPDATE, it may happen that none of the LOB data fit into the original block and that all must chain or migrate to another block. Setting a high value for PCTFREE will reduce this undesirable effect.

To calculate an appropriate value for PCTFREE, first estimate the average size of the LOB data and the average size of the non-LOB data. If datasize is the amount of space available in a block:

```
datasize = blocksize – blockheadersize
```

Then

```
num_rows = TRUNC ( datasize / (aveLOB + aveNonLOB) )
```

will be the maximum number of rows expected to fit completely into a block. Set PCTFREE to a value that will limit the amount of non-LOB data to be inserted, leaving space for LOB data, as follows

```
PCTFREE = 1 – (num_rows * aveNonLOB / datasize )
```

The F_STORAGE configuration string within the DBTUNE table controls the allocation of space for feature tables. By default, PCTFREE is set to 30 in the F_STORAGE configuration strings. A larger value of PCTFREE is necessary to reduce the problem of row chaining which may occur in the storage of BLOB column data.

Referential integrity

Maintaining the referential integrity between the business and feature table is important. You should not edit the records of either the feature table or the spatial index table. Several indexes and constraints have been added to the business, feature, and spatial index table to ensure referential integrity is maintained. However, these indexes and constraints are removed when the feature class is converted to the load-only I/O mode, a state that allows for rapid insertion of data into the feature class.

When the feature class is placed back into normal I/O mode, the state that allows users to query the feature class through an ArcSDE client, the indexes are created, and the constraints are enabled. The conversion to normal I/O mode will fail if the unique indexes cannot be built on the business table's spatial column or the feature table's FID column. It will also fail if a value exists in the business table's spatial column that is not in the feature table's FID column. In this case a reference to the offending business table record is loaded into the SDE_EXCEPTIONS table.

APPENDIX D

Oracle Spatial geometry type

ArcSDE supports Oracle Spatial's Object Relational Model as a method to store spatial data. Oracle Spatial's Object Relational Model was introduced in Oracle8*i* as an alternative to the Oracle Spatial normalized schema. The Oracle Spatial Object Relational Model uses object-relational types and methods to define, index, and perform spatial analysis on spatial data. For additional information on Oracle Spatial, please see the *Oracle Spatial Users Guide and Reference* for your Oracle release.

What is Oracle Spatial?

Oracle Spatial is a product that extends the Oracle database with the addition of spatial data management functions. Oracle Spatial provides a SQL geometry type, spatial metadata schema, spatial indexing methods, spatial functions, and implementation rules.

The Oracle Spatial product is available in Oracle Enterprise Edition and must be licensed separately. Oracle Spatial is not available for Oracle Workgroup or Personal Oracle products.

Geometry type

The Oracle Spatial geometry type is named SDO_GEOMETRY and is implemented using Oracle's extensible object-relational type system. The SDO_GEOMETRY type stores information about a geometry value including its shape type, spatial reference ID, interpolation instructions, and coordinate values.

The SDO_GEOMETRY type supports single and multipart point, line, and area geometry. Geometries may be defined as having linear interpolation between coordinates as defined by the OpenGIS Simple Feature Specification. Geometries may also be constructed from circular curves or a combination of both interpolation methods.

Applications are responsible for properly inserting, updating, and fetching the contents of the SDO_GEOMETRY type using Oracle's object-relational SQL interface. Applications are also responsible for ensuring that the contents of the SDO_GEOMETRY type adhere to the rules defined in the Oracle Spatial documentation.

Consult your *Oracle Spatial Users Guide and Reference* for your Oracle release for information on the definition and use of the SDO_GEOMETRY type.

Metadata schema

The Oracle Spatial metadata schema provides storage for information about the geometry data. It is the responsibility of an application to record in the metadata schema each SDO_GEOMETRY column it creates or modifies. Applications can record such information as the SDO_GEOMETRY schema, spatial reference ID, coordinate dimension, and spatial indexing instructions.

Consult your *Oracle Spatial Users Guide and Reference* for information on the Oracle Spatial metadata schema.

Spatial index

Oracle Spatial provides spatial indexing methods for the SDO_GEOMETRY type. Spatial indexes provide fast access to records based on the location of geometry values.

Oracle Spatial provides three different spatial indexing methods: Fixed Quadtree, Hybrid Quadtree, and Rtree indexes. Oracle Spatial provides the Oracle Spatial index advisor utility to assist in defining spatial index parameters.

Consult your *Oracle Spatial Users Guide and Reference* for information on supported Spatial index types.

Spatial functions

Oracle Spatial extends SQL with search functions that provide primary and secondary spatial filtering. Including the SDO_FILTER function in a SQL query performs a primary spatial search that uses the spatial index of tables with SDO_GEOMETRY columns. Spatial predicates, such as SDO_RELATE and SDO_CONTAINS, return secondary relationships between pairs of SDO_GEOMETRY.

Oracle Spatial provides spatial transformation functions that change the form of an SDO_GEOMETRY value. For example, the SDO_BUFFER function computes the coordinates of a new SDO_GEOMETRY object as a distance surrounding the original geometry. Other spatial transformation functions include SDO_DIFFERENCE and SDO_INTERSECTION.

Consult your *Oracle Spatial Users Guide and Reference* for information on supported spatial search functions and transformations.

Coordinate reference

Oracle Spatial provides a number of coordinate reference systems using predefined spatial reference ID (SRID) values. The predefined SRID value is stored in the SDO_GEOMETRY object. The SDO_TRANSFORM function uses the spatial reference ID to establish coordinate reference transformations.

Consult your *Oracle Spatial Users Guide and Reference* for information on supported coordinate references.

How does ArcSDE use Oracle Spatial?

ArcSDE supports Oracle Spatial for managing geometry in an Oracle Enterprise Edition database.

Making Oracle Spatial your default geometry schema

When you install ArcSDE, the SDE compressed binary geometry schema is the default. The SDE compressed binary schema is supported on Oracle Workgroup and Enterprise servers. The default settings for ArcSDE are defined in the DBTUNE table—one of these settings controls the GEOMETRY_STORAGE method. As installed, the GEOMETRY_STORAGE setting looks like this:

```
##DEFAULTS
GEOMETRY_STORAGE              SDEBINARY
<other parameters>
END
```

Changing the default ArcSDE geometry storage to use Oracle Spatial is easy. Simply changing the GEOMETRY_STORAGE setting from SDEBINARY to SDO_GEOMETRY makes Oracle Spatial the ArcSDE default geometry storage schema:

```
##DEFAULTS
GEOMETRY_STORAGE              SDO_GEOMETRY
<other parameters>
END
```

After the default GEOMETRY_STORAGE setting has been changed to SDO_GEOMETRY, ArcSDE, by default, creates feature classes with SDO_GEOMETRY columns.

ArcSDE for Oracle supports a number of different geometry storage schemas—these different schemas can all exist in the same database. While there can only be one default geometry schema, individual tables can be created using different geometry schemas. See Chapter 3, 'Configuring DBTUNE storage parameters', for instructions on using DBTUNE to define different geometry schemas.

SDO_GEOMETRY columns

ArcSDE creates a feature class by adding a geometry column to the specified business table. When the GEOMETRY_STORAGE storage parameter is set to SDO_GEOMETRY, ArcSDE adds an SDO_GEOMETRY column to the business table. In this example, a business table named 'COUNTRIES' has name and population properties—after adding a geometry column, it also has an SDO_GEOMETRY column named 'BORDERS'.

NAME	DATA TYPE	NULL?
NAME	VARCHAR2(32)	
POPULATION	NUMBER(11)	
BORDERS	MDSYS.SDO_GEOMETRY	
OBJECTID	NUMBER(38)	NOT NULL

'COUNTRIES' business table schema

A geometry column can be added to the business table using ArcCatalog, the sdelayer administration utility, or the ArcSDE C and Java API.

When ArcSDE adds the SDO_GEOMETRY column, an additional OBJECTID column is also defined. ArcSDE uses the OBJECTID for log file queries, single row operations, and multiversioned database operations. An existing column can be used for the OBJECTID as

long as it is indexed and declared as NUMBER(38) with a NOT NULL and UNIQUE constraint.

Spatial index

When an SDO_GEOMETRY column is added to a business table, a spatial index on that geometry column is also defined. By default, ArcSDE creates an RTREE index on an SDO_GEOMETRY column. ArcSDE includes support for the creation of Oracle Spatial Fixed and Hybrid indexes. ArcSDE can also create feature classes with no spatial index; however, spatial constraints cannot be supported until a spatial index is constructed.

The spatial index can be defined using ArcCatalog, the sdelayer administration utility, DBTUNE configurations, or the ArcSDE C and Java API.

ArcSDE automatically drops and re-creates Oracle Spatial indexes that ArcSDE has created whenever the feature class is switched between LOAD_ONLY_IO and NORMAL_IO mode.

Spatial indexes defined by the Oracle Spatial Index Advisor application are not dropped when ArcSDE switches the feature class to LOAD_ONLY_IO mode.

Coordinate dimension

You can create ArcSDE geometry as 2D, 2D with measures, 3D, and 3D with measures. ArcSDE defines the Oracle Spatial dimension information (DIMINFO) as:

- The X ordinate is the first dimension.
- The Y ordinate is the second dimension.
- The Z ordinate is the third dimension if the feature class is defined as having elevations.
- The M ordinate is the last dimension if the feature class is defined as having measures.

The dimension extents for X range, Y range, Z range, and M range can be calculated from the actual data or based on fixed values in the DBTUNE table. See Chapter 3, 'Configuring DBTUNE storage parameters', for DBTUNE storage parameters to establish predefined Oracle Spatial dimension properties and to change the name of any dimension.

Coordinate reference

Oracle Spatial coordinate references are predefined and cannot be modified. To set an SDO_GEOMETRY column's Spatial Reference ID (SRID), identify the appropriate Oracle Spatial coordinate reference description and set the feature class's SDO_SRID DBTUNE storage parameter to that value. If the SDO_SRID storage parameter is not set, the SRID of each SDO_GEOMETRY value is set to NULL, and so is the corresponding SRID of the metadata record in the USER_SDO_GEOM_METADATA view.

ArcSDE does not require the Oracle Spatial coordinate reference ID to be set. ArcSDE maintains the coordinate reference information for each feature class in its own SPATIAL_REFERENCES table independent of Oracle Spatial.

Consult your *Oracle Spatial Users Guide and Reference* for information on supported coordinate references.

Oracle Spatial metadata

When ArcSDE adds an SDO_GEOMETRY column to a business table, it also adds the required Oracle Spatial metadata record to the USER_SDO_GEOM_METADATA view. This metadata includes the table name, SDO_GEOMETRY column name, spatial reference ID, and coordinate dimension information.

SDO_GEOMETRY values

ArcSDE populates the SDO_GEOMETRY value from the SE_SHAPE object. The ArcSDE SE_SHAPE object can contain simple and complex geometry that may include elevations, measures, CAD data, annotation, and surface patches. The SDO_GEOMETRY value supports a subset of these geometric properties.

When generating the SDO_GEOMETRY coordinate information, follow these rules:

- Define the SDO_GTYPE parameter based on the Oracle 8.1.6 release.
- The SDO_SRID is NULL if no Oracle Spatial coordinate reference is defined.
- Coordinate values are written in the appropriate coordinate reference system.
- Coordinates are written as X, Y, <Z>, and <M>, where the elevation and measure ordinates are only defined if present in the source SE_SHAPE object.
- If elevations and/or measures are present in the source SE_SHAPE object, all coordinates are stored with an elevation and/or measure ordinate.
- Elevations and measures may be NAN if specific coordinates in the geometry contain undefined elevation or measure values.
- Measures are not restricted to ascending or descending order.
- Measures may be present on any geometry type, not just linestrings.
- Circular curves are written to the SDO_GEOMETRY type; other nonlinear interpolated shapes (cubic spline, bezier, etc.) are converted to linestrings.
- All ArcSDE-generated SDO_GEOMETRY values are generated from SE_SHAPE objects that have passed the ArcSDE rigorous geometry validation.
- Single point layers use the SDO_POINT property in the SDO_GEOMETRY object.
- ArcSDE does not support heterogeneous geometry collection in the SDO_GEOMETRY object.
- ArcSDE does not encode SDO_ETYPE 0 elements in the SDO_GEOMETRY object. SDO_ETYPE 0 elements are application specific. See ‘Interoperability considerations’ below for more information.

CAD and annotation properties

The SDO_GEOMETRY type cannot store all of the computer-aided design (CAD) or annotation data that ArcSDE supports. To store CAD or annotation data, ArcSDE adds a BLOB column called SE_CAD to the business table whenever a feature class defined with CAD or annotation properties is created with an SDO_GEOMETRY spatial column.

NAME	DATA TYPE	NULL?
NAME	VARCHAR2(32)	
POPULATION	NUMBER(11)	
BORDERS	MDSYS.SDO_GEOMETRY	
SE_CAD	BLOB	
OBJECTID	NUMBER(38)	NOT NULL

'COUNTRIES' business table schema with the addition of CAD data

Whenever ArcSDE detects that the data source has CAD data, ArcSDE writes a simple geometric representation of the CAD data into the SDO_GEOMETRY value and then writes the unmodified CAD data into the SE_CAD value. If the data source does not have CAD data, ArcSDE sets the SE_CAD value to NULL.

The SE_CAD property contains data from numerous ArcGIS components:

- AutoCAD or MicroStation data from ArcSDE CAD Client
- Parametric objects such as cubic splines and bezier curves from ArcMap
- Surface patches from ArcMap Spatial Analyst
- Annotation from ArcInfo Workstation

Spatial queries

ArcSDE uses the Oracle Spatial SDO_FILTER function to perform the primary spatial query. ArcSDE performs secondary filtering of the SDO_GEOMETRY based on the spatial relationship requested by the application.

Applications may also include Oracle Spatial primary and secondary filter functions in the SQL where clause supplied to ArcSDE. Using spatial filters in the where clause, applications can distribute the spatial query between the database server, the ArcSDE application server, and the application itself.

How does ArcSDE use existing Oracle Spatial tables?

Tables with SDO_GEOMETRY columns can be created by other applications. ArcSDE has been designed to use tables containing SDO_GEOMETRY columns that were created by other applications.

Automatic discovery of tables with SDO_GEOMETRY columns

Whenever an ArcSDE client lists the feature classes stored in the database, ArcSDE automatically searches the Oracle Spatial metadata views for new tables with

SDO_GEOMETRY columns. When a new table is discovered, it is registered with ArcSDE and made available to applications.

ArcSDE uses the first record in a newly discovered table to establish the ArcSDE geometry type. If the table contains multiple geometry types, the sdelayer administration utility can be used to alter the geometry type definition.

ArcSDE searches for a column in the table to use as an OBJECTID column. To qualify, the column must be defined as NUMBER (38), NOT NULL, and UNIQUE constraints. If such a column is found, it is recorded in the ArcSDE table registry along with the table. If an OBJECTID column is not found, the table is registered, but operations requiring an OBJECTID are unavailable.

For SDO_GEOMETRY columns that have an Oracle Spatial coordinate reference ID (SRID), ArcSDE stores the information in the ArcSDE SPATIAL_REFERENCES table. ArcSDE sets its spatial reference AUTH_NAME field to ORACLE and the AUTH_SRID field to the SRID value. ArcSDE tests the coordinate reference description and, if it is valid, sets the SRTEXT field to the Oracle Spatial coordinate reference description.

To disable automatic discovery of tables with SDO_GEOMETRY columns (often referred to as auto-registration), set the environment variable SDEDISABLEAUTOREG to 1. To disable automatic discovery for all users of an ArcSDE (three-tiered) service, set this variable in the file dbinit.sde in the etc directory under the SDEHOME environment variable. To prevent automatic discovery by client applications that use direct connect, set this environment variable in the client's environment before connecting.

Accessing SDO_GEOMETRY values

ArcSDE does not support heterogeneous geometry collections in an SDO_GEOMETRY object.

ArcSDE automatically validates SDO_GEOMETRY values as they are fetched from the database. This validation ensures that geometric objects are properly formed and adhere to feature class constraints. Geometry values that do not pass validation are skipped.

Interoperability considerations

There is a common misconception that applications can interoperate simply because they support the same underlying geometry type. The geometry type is only one aspect of the interoperability picture—a common understanding of rules, constraints, schema, and implementation is also required.

Multiple SDO_GEOMETRY columns in a table

ArcSDE and ArcGIS do not support multiple geometry columns in a table. Tables with multiple SDO_GEOMETRY columns should be accessed through views that contain only one SDO_GEOMETRY column.

Use the sdetable create_view operation to create views of business tables.

Single geometry type in an SDO_GEOMETRY column

While ArcSDE supports multiple geometry types in a geometry column, many applications do not. For example, ArcGIS requires that a geometry column be restricted to a single geometry type.

Oracle Spatial does not enforce geometry type constraints on an SDO_GEOMETRY column. While one application may want to enforce a single geometry type constraint on an SDO_GEOMETRY column, another application could insert different geometry types. You can create an insert–update trigger that checks the SDO_GEOMETRY GTYPE property to enforce geometry types.

Geometry validation

Oracle Spatial does not automatically enforce geometry validation on the insert or update of an SDO_GEOMETRY value. You can create an insert–update trigger to fire the SDO_VALIDATE function to enforce validation of SDO_GEOMETRY types.

To reduce the occurrences of problems from illegal or poorly formed geometry values, ArcSDE automatically validates any SDO_GEOMETRY value from a table that was created by other applications.

SDO_ETYPE 0 elements in an SDO_GEOMETRY object

Oracle Spatial allows applications to insert application-specific data into an SDO_GEOMETRY object. Applications do this by embedding their data using an SDO_ETYPE value of 0. The nature of the application specific data is known only to the application that generated the SDO_GEOMETRY object. Such application-specific data cannot be reliably supported in an interoperable environment. Applications reading SDO_GEOMETRY objects do not know how to interpret SDO_ETYPE 0 data. Applications updating SDO_GEOMETRY objects do not know how to edit or preserve the SDO_ETYPE 0 data.

When reading SDO_GEOMETRY objects containing SDO_ETYPE 0 elements, ArcSDE will ignore the SDO_ETYPE 0 data and will return the geometry component to the application.

When updating SDO_GEOMETRY objects containing SDO_ETYPE 0 elements, ArcSDE will not preserve the SDO_ETYPE 0 data. Applications that need to ensure that SDO_ETYPE 0 data is preserved should make sure that users do not have UPDATE access to the table.

Coordinate reference

ArcSDE requires that all SDO_GEOMETRY values in a column be in the same coordinate reference system. If the coordinate reference is undefined, the SRID value should be NULL as defined in the *Oracle Spatial Users Guide and Reference.*

Oracle Spatial does not automatically enforce spatial reference ID validation on insert or update of an SDO_GEOMETRY value. You can create an insert–update trigger to fire the SDO_VALIDATE function to enforce validation of SDO_GEOMETRY types.

Identification of elevations and measures

Oracle Spatial and ArcSDE assume that the first and second dimensions of the SDO_GEOMETRY are X and Y.

If an SDO_GEOMETRY object has three dimensions, ArcSDE assumes the third dimension is a measure if the dimension name starts with an "M"; otherwise, the third dimension is assumed to be an elevation. You can control how ArcSDE interprets the third dimension by setting the feature class's entity flag to have either elevations or measures.

In this example, sdelayer sets the entity type of a feature class to store linestrings and elevations.

```
sdelayer -o add -e l3
```

In this example, sdelayer sets the entity type of a feature class to store linestrings and measures.

```
sdelayer -o add -e lM
```

If the SDO_GEOMETRY object has four dimensions, measures are expected to be the last ordinate.

Measures and linear reference

Oracle Spatial provides functions for linear reference calculations using measure values. The Oracle Spatial linear reference functions require that all measure values in an SDO_GEOMETRY object be monotonically ascending or descending without NAN values. Oracle Spatial linear reference also restricts measure values to linestrings.

ArcSDE allows measures and linear reference calculations on all geometric types, with support for arbitrarily ordered measure values and NAN values.

If you use Oracle Spatial linear reference functions, design your application and database to operate within the Oracle Spatial linear reference constraints.

Object identification

ArcSDE provides limited support for Oracle Spatial tables without an OBJECTID column—no logfile operations, no single row operations, and no versioned database support. An OBJECTID column can easily be added using the sdetable administration command or the ArcCatalog application.

Many applications, such as ArcGIS Desktop, require an OBJECTID column in a table. Each application has its own requirements and limitations. ArcGIS Desktop can draw and query feature classes created with SDO_GEOMETRY columns but without an OBJECTID column; however, ArcGIS Desktop can't select, edit, or add these feature classes to a feature dataset.

Multiversioned database

ArcGIS Desktop uses a multiversioned database implemented through ArcSDE for all editing operations. The multiversioned database provides long transaction support for multiple simultaneous design alternatives.

Multiversioned views are available for SQL access to the multiversioned database. See the *ArcSDE Developers Guide* for more information.

Networks and topology feature classes

ArcGIS Desktop can create and maintain networks and integrated topological feature classes from simple feature classes that use the SDO_GEOMETRY type. ArcGIS Desktop manages the relationships and maintains the topological integrity of the data—modifications to the underlying features through ArcGIS Desktop are reflected in these integrated networks.

Modification of Oracle Spatial feature classes participating in these networks should be restricted to ArcGIS Desktop applications. Other applications can freely read the data, but any modifications they make are not reflected in the networks.

Relationships and constraints

ArcGIS Desktop enforces relationships and constraints across lots of different data sources. Feature classes containing an SDO_GEOMETRY type may be included in relationships and may have constraints defined on them.

Modification of Oracle Spatial feature classes participating in relationships and constraints should be restricted to ArcGIS Desktop applications. Other applications can freely read the data, but their edits are not properly handled.

Oracle Spatial software patches

Keep the Oracle Spatial software current by applying the patches as they become available.

ArcSDE Oracle Spatial Support

Please visit the ESRI On-Line Support Center (http://support.esri.com) for help with Oracle Spatial issues.

Index