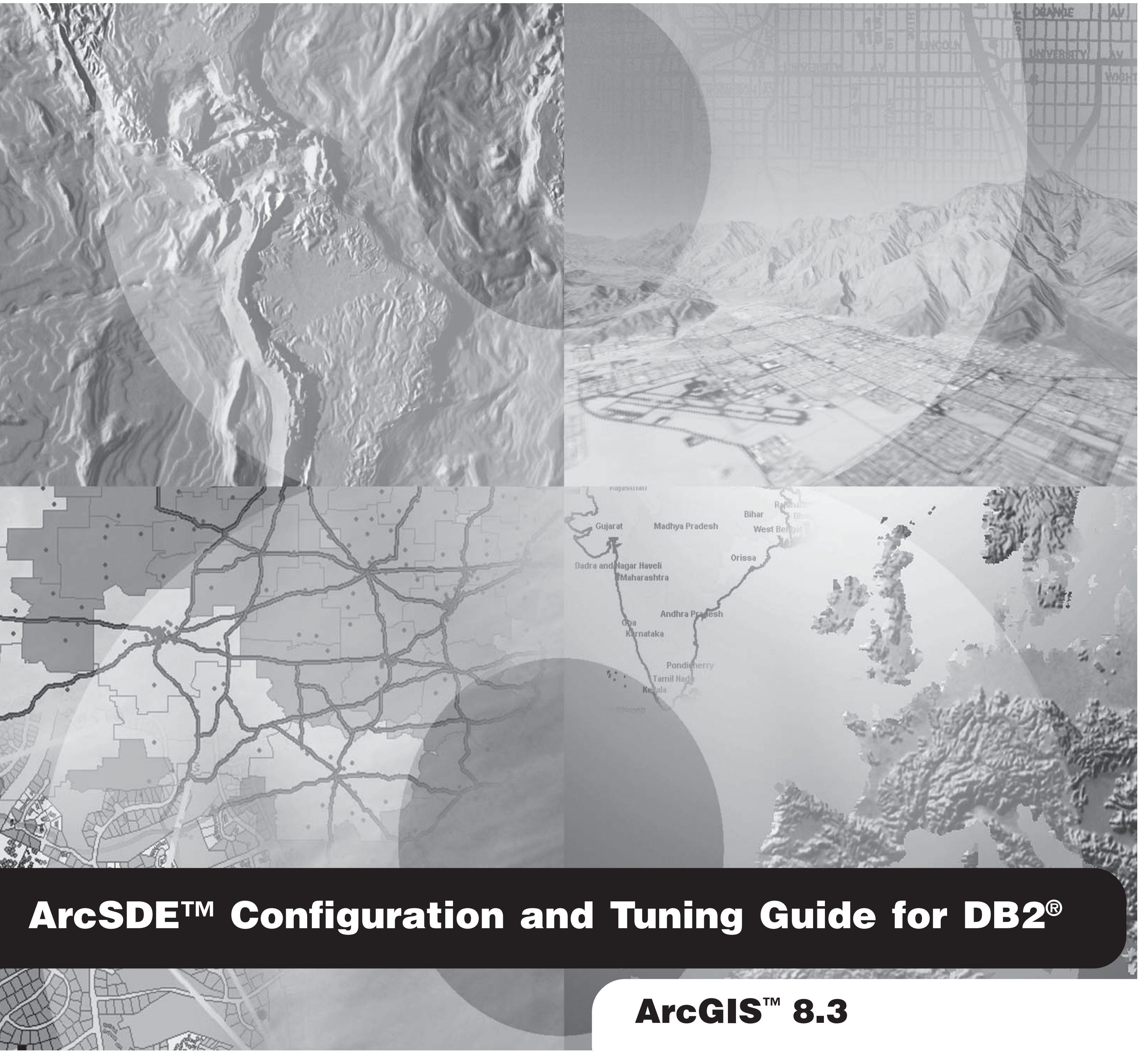


ArcSDE™ Configuration and Tuning Guide for DB2®

ArcGIS™ 8.3

Copyright © 2001–2002 ESRI
All Rights Reserved.
Printed in the United States of America.

The information contained in this document is the exclusive property of ESRI. This work is protected under United States copyright law and the copyright laws of the given countries of origin and applicable international laws, treaties, and/or conventions. No part of this work may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopying or recording, or by any information storage or retrieval system, except as expressly permitted in writing by ESRI. All requests should be sent to Attention: Contracts Manager, ESRI, 380 New York Street, Redlands, CA 92373-8100, USA.

The information contained in this document is subject to change without notice.

U. S. GOVERNMENT RESTRICTED/LIMITED RIGHTS
Any software, documentation, and/or data delivered hereunder is subject to the terms of the License Agreement. In no event shall the U.S. Government acquire greater than RESTRICTED/LIMITED RIGHTS. At a minimum, use, duplication, or disclosure by the U.S. Government is subject to restrictions as set forth in FAR §52.227-14 Alternates I, II, and III (JUN 1987); FAR §52.227-19 (JUN 1987) and/or FAR §12.211/12.212 (Commercial Technical Data/Computer Software); and DFARS §252.227-7015 (NOV 1995) (Technical Data) and/or DFARS §227.7202 (Computer Software), as applicable. Contractor/Manufacturer is ESRI, 380 New York Street, Redlands, CA 92373-8100, USA.

ESRI, ArcView, ArcSDE, SDE, MapObjects, ArcInfo, ArcCatalog, ArcMap, ArcToolbox, ArcStorm, ArcGIS, ArcInfo, and Spatial Database Engine, and www.esri.com are trademarks, registered trademarks, or service marks of ESRI in the United States, the European Community, or certain other jurisdictions.

Other companies and products mentioned herein are trademarks or registered trademarks of their respective trademark owners.

Contents

CHAPTER 1

Getting started

ESRI® ArcSDE™ for DB2® allows you to store geographic data in a DB2 database and requires, like any other application of DB2, consideration for configuring and tuning the data stored. This document explains how to use ArcSDE and its applications to create, store, and index the spatial data in a DB2 database.

Some tuning tips

Chapter 2, 'Essential configuring and tuning', provides a brief overview of proper container placement to minimize the impact of disk I/O contention. Also, the proper selection of the grid cell sizes for the construction of the spatial index is discussed.

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 8.3 as the initial source of storage parameters. When the ArcSDE 8.3 sdesetupdb2 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 8.3 has simplified the storage parameters. Rather than matching each DB2 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. The ArcSDE storage parameter holds all the DB2 storage parameters of a DB2 CREATE TABLE or CREATE INDEX statement. Pre-ArcSDE 8.1 DB2 storage parameters are ignored.

The sdedbtune command has been introduced at ArcSDE 8.2 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 8.3 installation creates the DBTUNE table. If the dbtune.sde file is absent or empty, sdesetupdb2 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.

Creating spatial data in a DB2 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 a DB2 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 ArcCatalog or ArcMap™. An alternative approach to creating spatial data in a DB2 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 a DB2 database.

CHAPTER 2

Essential configuring and tuning

The performance of an ArcSDE service depends to some extent on how well you configure and tune your DB2 instance. This chapter discusses the basic approach to database management system (DBMS) tuning and provides some instruction on tuning the DB2 Spatial Extender's spatial index.

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.

A well-tuned DB2 database makes optimum use of available CPU and memory while minimizing disk input/output (I/O) contention. Database administrators approach this task knowing 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.

Reducing disk I/O contention

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".

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 DB2 database.

Separate tables from their indexes

Each time DB2 accesses an index to locate a row, it must access the table to fetch the referenced row. The disk head travels between the index and the table if they are stored on the same disk.

Whenever possible, store indexes and tables in different tablespaces so you can store them on different disk drives, thus eliminating repetitive and costly disk head travel.

Establish the threshold table size

As a rule, store small tables together in the same tablespace and large tables by themselves in their own tablespaces. Decide how large a table must be before it requires its own tablespace. Generally, the threshold data object size corresponds in part to the maximum container size. Tables capable of filling the maximum size container should be stored in their own tablespace. Tables approaching this limit should also be considered. Follow the same policy for indexes.

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 containers of the high access tablespaces with low access containers. This same rule applies to indexes. They, too, should be divided by access.

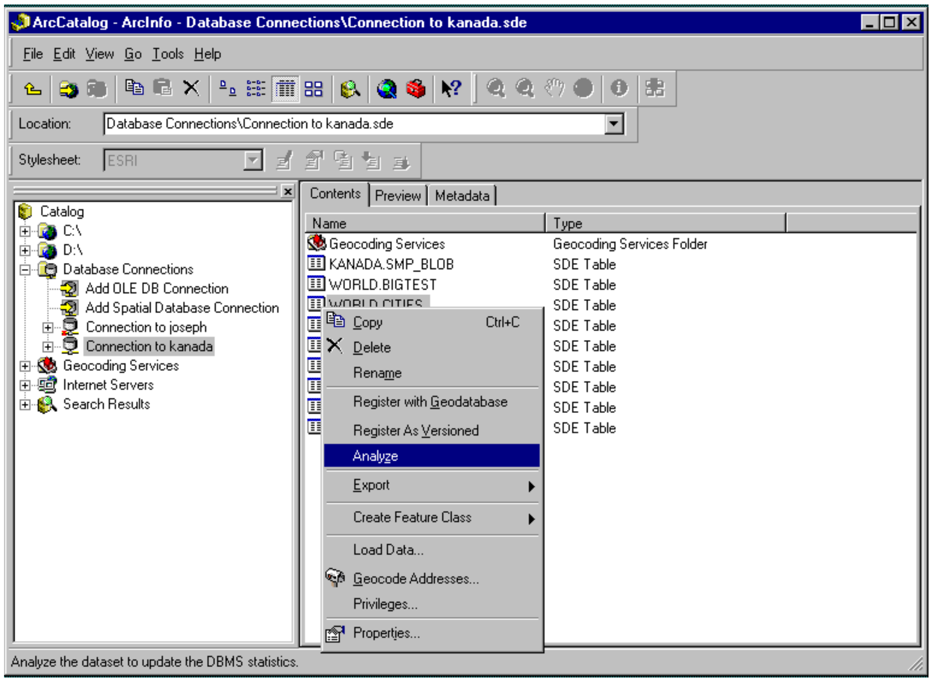
Positioning the files

Once you have estimated the size of the containers, 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.

Updating DB2 statistics

For the best performance, the statistics of the ArcSDE tables and indexes that you have stored in DB2 must be kept 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 Analyze. To update the tables and indexes within a feature class, right-click on the feature class and click Analyze.



Name	Type
Geocoding Services	Geocoding Services Folder
KANADA.SMP_BLOB	SDE Table
WORLD.BIGTEST	SDE Table
WORLD.CITIES	SDE Table
	SDE Table
	SDE Table
	SDE Table
	SDE Table
	SDE Table
	SDE Table

Analyze the dataset to update the DBMS statistics.

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 sdetable UPDATE_DBMS_STATS operation rather than individually analyzing the tables with the DB2 RUNSTATS statement because it updates the statistics for all tables of a feature class. In addition to the business table, an ArcSDE for DB2 feature class may include an adds and deletes table as well.

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 DB2 optimizer continues to choose an optimum execution plan. To save time, you can update the statistics of all the data objects within a feature dataset in ArcCatalog.

If you decide to update the statistics of all or some of the feature class tables with the DB2 RUNSTATS statement, use the following syntax:

```
RUNSTATS ON TABLE <table_name> WITH DISTRIBUTION AND DETAILED INDEXES ALL
```

For more information on the DB2 SQL RUNSTATS statement, refer to the *IBM DB2 Universal Database Command Reference*.

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 roads_ix index of the roads table can be updated.

```
sdetable -o update_dbms_stats -t roads -n roads_ix -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 ArcSDE Developer Help.

Tuning the spatial index

Applications querying the two-dimensional geographic data contained in a spatial column require an index strategy that will quickly identify all geometries lying within a given extent. For this reason the Spatial Extender provides the three-tiered grid spatial index.

The two-dimensional spatial index differs from the traditional hierarchical Btree index provided by DB2. To better understand the difference, first review how a Btree index is structured and used.

The top level of a Btree index called the root node contains one key for each node at the next level. The value of each of these keys is the largest existing key value for the corresponding node at the next level. Depending on the number of values in the base table, several intermediate nodes may be needed to bridge the root node with the leaf nodes, which hold the actual base table row IDs.

The DB2 database manager searches a Btree index starting at the root node, working its way through the intermediate nodes until it reaches the leaf node with the row ID of the base table.

The Btree index may not be applied to a spatial column because the two-dimensional characteristic of the spatial column requires the structure of a spatial index. For the same reason, you may not apply a spatial index to a nonspatial column, and a spatial index may not be applied to a composite column of any kind.

The spatial index's CREATE INDEX syntax includes the additional *USING* clause, which directs DB2 to use the Spatial Extender's spatial index rather than a Btree index. The full syntax is as follows:

```
create index <index_name> on <table> (<spatial column>)
using db2gse.spatial_index (<grid level 1>, [grid level 2] , [grid level 3])
```

The addition of the *USING* clause distinguishes the spatial index from the Btree index. The db2gse owner of the Spatial Extender functionality must qualify the spatial_index index extension name as this statement does not follow the current function path.

Because of the simple nature of the data a Btree was designed to index, the database designer merely directs DB2 to create the index on one or more table columns. Spatial data being complex requires the designer to understand its relative size distribution. The designer must determine the optimum size and number of the spatial index's grid levels.

The grid levels (<grid level 1>, [grid level 2], [grid level 3]) are entered by increasing cell size. Thus the second level must have a larger cell size than the first and the third a larger cell size than the second. The first grid level is mandatory, but you may disable the second and third with a zero value (0).

How the Spatial Extender generates a spatial index

The DB2 Spatial Extender constructs a spatial index as follows:

1. The Spatial Extender intersects each geometry's envelope with the grid starting with the first level.

2. If less than four intersections occur with the first grid level, the Spatial Extender enters the geometry ID and the intersecting grid cell IDs in the spatial index and continues with the next geometry.

3. If the Spatial Extender detects more than four intersections, it intersects the geometry with the second grid level. If you have not enabled the second grid level, the Spatial Extender enters the geometry ID and grid cell IDs in the spatial index and continues with the next geometry.

4. If less than four intersections occur with the second grid level, the Spatial Extender enters the geometry ID and the intersecting grid cell IDs in the spatial index and continues with the next geometry.

5. If the Spatial Extender detects more than four intersections, it intersects the geometry with the third grid level. If you have not enabled the third grid level, the Spatial Extender enters the geometry ID and grid cell IDs in the spatial index and continues with the next geometry.

6. The Spatial Extender enters the geometry ID and the intersecting grid cell IDs of the third level in the spatial index and continues with the next geometry.

The Spatial Extender does not actually create polygon grid structure of any sort. The Spatial Extender manifests each grid level parametrically by defining the origin as the x,y offsets of the column's spatial reference system extending into positive coordinate space. Using a parametric grid the Spatial Extender generates the intersections mathematically.

How the Spatial Extender uses the spatial index

The Spatial Extender uses a spatial index to improve the performance of a spatial query. Consider the box query—the most basic and probably most popular spatial query. The box query returns geometries of a spatial column that intersect a user-defined box. If a spatial index does not exist, the Spatial Extender must compare all of a spatial column's geometries with the box.

Using the spatial index, the Spatial Extender identifies index grid entries that intersect the box. Since the spatial index is ordered on grid, the Spatial Extender quickly obtains a list of candidate geometries. The process just described is referred to as the *first pass*.

A *second pass* disqualifies candidate geometries whose envelope does not intersect the box.

A *third pass* compares the actual coordinates of the candidate geometry with the box to determine whether or not the geometry intersects the box. This last complex process of comparison operates on a subset of the table rows, significantly reduced by the first two passes.

All spatial queries perform the three passes with the exception of the SE_EnvelopesIntersect function. It performs only the first two passes and was designed for display operations that use display driver clipping routines and that don't require the granularity of the third pass.

Selecting the optimum grid cell sizes

Selecting the grid cell size is complicated by the fact that envelopes of irregularly shaped geometries do not fit neatly within a grid cell. Because of this irregularity, some geometry envelopes intersect several grids, while others fit inside a single grid cell. On the flip side, grid cells may intersect several geometry envelopes depending on the spatial distribution of the data.

A spatial index performs well when you enable the correct number of levels and their grid cell sizes to 'fit' the data. To simplify this discussion, first consider a spatial column containing geometry whose size is uniform. In this case, it is not necessary to create a multileveled spatial index since a single grid level will suffice. Create a spatial index with a single grid level whose grid cell size is 1.5 times the size of the average geometry envelope.

While testing your application, you may find that it performs better with a larger grid cell size because each grid cell references more geometries, enabling the *first pass* to discard nonqualifying geometries faster. However, if you continue to increase the grid cell size, performance deteriorates as the number of geometries filtered by the *second pass* increases.

Selecting the number of levels

Few spatial columns contain geometry of the same relative size. However, geometries of most spatial columns can be grouped into size intervals. For instance, consider a spatial column of county parcels containing a vast number of small parcels clustered in the urban areas surrounded by a few large rural parcels. These situations are very common and require the use of a multilevel spatial index. To select the grid cell sizes of each level, determine the intervals of geometry envelope sizes. Create a spatial index with grid level cell sizes slightly larger than each interval. Test the index by performing queries against the spatial column using your application. Try adjusting the grid sizes up or down slightly to determine if an appreciable improvement in performance can be obtained.

CHAPTER 3

Configuring DBTUNE storage parameters

DBTUNE storage parameters allow you to control how ArcSDE clients create objects within a DB2 database. They determine such things as which tablespace a table or index is created in. The storage parameters define the size of the data objects they store as well as other DB2-specific storage attributes.

The DBTUNE table

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

The DBTUNE table has the following definition:

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

The keyword field stores the keywords. Within each keyword, there are a number of storage parameters, and the names of these are stored in the parameter_name field. Each storage parameter has a configuration string associated with it, and this is stored in the config_string field.

After creating the DBTUNE table, the setup phase of the ArcSDE 8.3 installation populates the table with the contents of the dbtune.sde file, which it expects to find under the etc directory of the SDEHOME directory.

If the DBTUNE table already exists, the ArcSDE setup phase will not alter its contents should you decide to run it again.

Arranging storage parameters by keyword

Storage parameters of the DBTUNE table are grouped by keyword. When the contents of the DBTUNE table are exported to a file, the keywords are prefixed by two pound signs “##”. The ‘END’ clause terminates each keyword.

Keywords define the storage configuration of simple objects such as tables and indexes and complex objects such as feature classes, network classes, and raster columns. ESRI client applications and some ArcSDE administration tools assign DBTUNE keywords to these objects. The pound signs ‘##’ are not included when the keywords are assigned.

DEFAULTS keyword

Each DBTUNE table has a fully populated DEFAULTS keyword. The DEFAULTS keyword can be selected whenever you create a table, index, feature class, or raster column. If you do not select a keyword for one of these objects, the DEFAULTS keyword is used. If you do not include a storage parameter in a keyword you have defined, ArcSDE substitutes the storage parameter from the DEFAULTS keyword.

The DEFAULTS keyword relieves you of the need to define all the storage parameters for each of your keywords. The storage parameters of the DEFAULTS keyword should be populated with values that represent the average storage configuration of your data.

During installation, if the ArcSDE software detects a missing DEFAULTS keyword 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
A_INDEX_ROWID              ""
A_INDEX_STATEID            ""
A_INDEX_USER               ""
#A_STORAGE                 "IN <TABLESPACE> INDEX IN <TABLESPACE> LONG IN
<TABLESPACE>"
B_INDEX_ROWID              ""
B_INDEX_USER               ""
#B_STORAGE                 "IN <TABLESPACE> INDEX IN <TABLESPACE> LONG IN
<TABLESPACE>"
B_RUNSTATS                 "YES"
#BLK_STORAGE               "IN <TABLESPACE> INDEX IN <TABLESPACE> LONG IN
<TABLESPACE>"
#AUX_STORAGE               "IN <TABLESPACE> INDEX IN <TABLESPACE>"
#BND_STORAGE               "IN <TABLESPACE> INDEX IN <TABLESPACE>"
#RAS_STORAGE               "IN <TABLESPACE> INDEX IN <TABLESPACE>"
D_INDEX_DELETED_AT         ""
D_INDEX_STATE_ROWID        ""
#D_STORAGE        "IN <TABLESPACE> INDEX IN <TABLESPACE> LONG IN
<TABLESPACE>"
LOB_OPTION        "LOGGED NOT COMPACT"
LOB_SIZE     "1M"
UI_TEXT           ""
END
```

Setting the default BLOB size

DB2 requires a size on BLOB column creation.

If a BLOB column is to be created, and it has a size of greater than 2 GB, this size will be ignored and the default LOB_SIZE parameter of 1 MB will be used. This will allow the DBA to carefully craft their database parameters.

Setting the B_RUNSTATS parameter

At ArcSDE 8.1.2 a new parameter called B_RUNSTATS was added to the dbtune.sde file.

"YES" will be the default if no B_RUNSTATS parameter is present in the DEFAULTS keyword of the dbtune.sde file. B_RUNSTATS only applies to the business table. This parameter will be used at the end of a data load, after all the records are inserted, and the layer is being readied to put into normal_io mode. The last part of switching to the normal_io mode will be the checking of B_RUNSTATS. If B_RUNSTATS is equal to "YES" or "yes", then a full runstats will be performed on the table automatically. If it is set to anything else, then a runstats will not happen. The vast majority of users will want to have the full runstats done on the table. For those who wish to do something special with it for some reason, like only do indexes, they can set B_RUNSTATS to "NO" and then perform a manual RUNSTATS command with any options they would like.

Setting the system table DATA_DICTIONARY keyword

During the execution of sdesetupdb2 the ArcSDE and geodatabase system tables and indexes are created with the storage parameters of the DATA_DICTIONARY keyword. You may customize the keyword in the dbtune.sde file prior to running the sdesetupdb2 tool. In this way you can change default storage parameters of the DATA_DICTIONARY keyword.

Edits to all of the geodatabase system tables and most of the ArcSDE system tables occur when schema change occurs. As such, edits to these system tables and indexes usually happen during the initial creation of an ArcGIS™ database with infrequent modifications occurring whenever a new schema object is added.

Four of the ArcSDE system tables—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 keyword, or if any of the required parameters are missing from the keyword, the following records will be inserted into the DATA_DICTIONARY when the table is created. (Note that the DBTUNE file entries are provided here for readability.)

```
##DATA_DICTIONARY
B_INDEX_ROWID           ""
B_INDEX_USER            ""
```

```
#B_STORAGE            "IN <TABLESPACE> INDEX IN <TABLESPACE> LONG IN
<TABLESPACE>"
#STATES_TABLE         "IN <TABLESPACE> INDEX IN <TABLESPACE>"
STATES_INDEX          ""
#STATE_LINEAGES_TABLE        "IN <TABLESPACE> INDEX IN <TABLESPACE>"
#VERSIONS_TABLE              "IN <TABLESPACE> INDEX IN <TABLESPACE>"
VERSIONS_INDEX               ""
#MVTABLES_MODIFIED_TABLE  "IN <TABLESPACE> INDEX IN <TABLESPACE>"
MVTABLES_MODIFIED_INDEX   ""
END
```

THE SURVEY MULTIBINARY KEYWORD

This keyword is used to support 2 BLOB columns on the SDB_<n>_Surveys table. However, it is mainly meant for Oracle since it cannot have multiple LONG RAW
Columns in the same business table.

```
##SURVEY_MULTI_BINARY
UI_TEXT ""
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 DB2, the default values for TOPOLOGY and TOPOLOGY::DIRTYAREAS are

```
##TOPOLOGY_DEFAULTS
UI_TOPOLOGY_TEXT "The topology default configuration"
A_INDEX_ROWID   ""
A_INDEX_SHAPE   ""
A_INDEX_STATEID ""
A_INDEX_USER    ""
#A_STORAGE  "IN <TABLESPACE> INDEX IN <TABLESPACE>"
B_INDEX_ROWID   ""
B_INDEX_SHAPE   ""
B_INDEX_USER    ""
#B_STORAGE  "IN <TABLESPACE> INDEX IN <TABLESPACE>"
D_INDEX_DELETED_AT  ""
D_INDEX_STATE_ROWID ""
#D_STORAGE  "IN <TABLESPACE> INDEX IN <TABLESPACE>"
END

##TOPOLOGY_DEFAULTS::DIRTYAREAS
A_INDEX_ROWID   ""
```

```
A_INDEX_SHAPE   ""
A_INDEX_STATEID ""
A_INDEX_USER    ""
#A_STORAGE  "IN <TABLESPACE> INDEX IN <TABLESPACE>"
B_INDEX_ROWID   ""
B_INDEX_SHAPE   ""
B_INDEX_USER    ""
#B_STORAGE  "IN <TABLESPACE> INDEX IN <TABLESPACE>"
D_INDEX_DELETED_AT  ""
D_INDEX_STATE_ROWID ""
#D_STORAGE  "IN <TABLESPACE> INDEX IN <TABLESPACE>”
END
```

The IMS METADATA keywords

The IMS METADATA keywords control the storage of the IMS Metadata tables. These keywords are a standard part of the dbtune table. If the keywords are not present in the dbtune file when it is imported into the DBTUNE table, ArcSDE applies software defaults. The software defaults have the same settings as the keyword parameters listed in the dbtune.sde table that is shipped with ArcSDE. You will need to edit the storage parameters tablespace names. As always try to separate the tables and indexes into different tablespaces.

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

The IMS metadata keywords are as follows:

The IMS_METADATA keyword controls 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.

```
##IMS_METADATA
B_INDEX_ROWID   ""
B_INDEX_SHAPE    ""
B_INDEX_USER     ""
#B_STORAGE        "IN <TABLESPACE> INDEX IN <TABLESPACE> LONG IN <TABLESPACE>"
LOB_OPTION        "LOGGED NOT COMPACT"
LOB_SIZE          "1M"
COMMENT           "The IMS metadata 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_INDEX_ROWID   ""
B_INDEX_USER    ""
#B_STORAGE       "IN <TABLESPACE> INDEX IN <TABLESPACE>"
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_INDEX_ROWID   ""
B_INDEX_USER    ""
#B_STORAGE      "IN <TABLESPACE> INDEX IN <TABLESPACE>"
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_INDEX_USER    ""
#B_STORAGE      "IN <TABLESPACE> INDEX IN <TABLESPACE> LONG IN <TABLESPACE>"
LOB_OPTION     "LOGGED NOT COMPACT"
LOB_SIZE       "1M"
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_INDEX_ROWID   ""
B_INDEX_USER    ""
#B_STORAGE      "IN <TABLESPACE> INDEX IN <TABLESPACE>"
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_INDEX_USER    ""
#B_STORAGE      "IN <TABLESPACE> INDEX IN <TABLESPACE>"
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_INDEX_USER    ""
#B_STORAGE      "IN <TABLESPACE> INDEX IN <TABLESPACE>"
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_INDEX_ROWID   ""
B_INDEX_USER    ""
#B_STORAGE       "IN <TABLESPACE> INDEX IN <TABLESPACE>"
END
```

Changing the appearance of DBTUNE keywords in the ArcInfo user interface

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

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

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

Adding a comment to a keyword

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

LOGFILE keywords

Logfiles are used by ArcSDE to maintain temporary and persistent sets of selected records. Whenever a user connects to ArcSDE for the first time, the SDE_LOGFILES and SDE_LOGFILE_DATA tables and indexes are created.

You may create a keyword for each user that begins with the LOGFILE_<username>. For example, if the user's name is STANLEY, ArcSDE will search the DBTUNE table for the LOGFILE_STANLEY keyword. If this keyword is not found, ArcSDE will use the storage parameters of the LOGFILE_DEFAULTS keyword to create the SDE_LOGFILES and SDE_LOGFILE_DATA tables.

ArcSDE always creates the DBTUNE table with a LOGFILE_DEFAULTS keyword. If you do not specify this 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 keyword storage parameters, ArcSDE supplies the rest. Therefore, the LOGFILE_DEFAULTS keyword is always fully populated.

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

Creating a logfile keyword for each user allows you to position their sde logfiles onto separate devices by specifying the tablespace the logfile 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 that make use of sde logfiles, it may help performance by spreading the logfiles across the file system.

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

```
##LOGFILE_DEFAULTS
LD_INDEX_DATA_ID          ""
LD_INDEX_ROWID            ""
#LD_STORAGE               "IN <TABLESPACE> INDEX IN <TABLESPACE>"
#LF_STORAGE               "IN <TABLESPACE> INDEX IN <TABLESPACE>"
UI_TEXT                   "LOGFILES"
END
```

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

Meta parameters define the way certain types of data will be stored, the environment of a keyword, or a comment that describes the keyword.

Table parameters

Table parameters define the storage configuration of a DB2 table. The table parameter is appended to a DB2 CREATE TABLE statement during its creation by ArcSDE. Valid entries for an ArcSDE table parameter include the parameters to the right of the SQL CREATE TABLE statement's columns list.

For example, a business table created with the following DB2 CREATE TABLE statement:

```
CREATE TABLE roads (road_id integer, name varchar2(32), surface_code integer)
in SDEDB2 index in SDEINDEX long in SDELOBS
```

would be entered into a DBTUNE file B_STORAGE table parameter with the following configuration string.

```
B_STORAGE "IN SDEDB2 INDEX IN SDEINDEX LONG IN SDELOBS"
```

Index parameters define the storage configuration of a DB2 index. The index parameter is appended to a DB2 CREATE INDEX statement during its creation by ArcSDE. Valid entries in an ArcSDE index parameter include all parameters to the right of the SQL CREATE INDEX statement's column list.

The business table storage parameter

A business table is any DB2 table created by an ArcSDE client, the sdetable administration command, or the ArcSDE C application programming interface (API) SE_table_create function.

Use the DBTUNE table's B_STORAGE 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 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 alter_reg operation of the sdetable command or with ArcCatalog. The SDE.TABLE_REGISTRY system table maintains a list of the currently registered tables.

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 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_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 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 that the business table was created with.

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 supporting indexes.

The RAS_STORAGE storage parameter holds the DB2 CREATE TABLE storage configuration of the RAS table.

The BND_STORAGE storage parameter holds the DB2 CREATE TABLE storage configuration of the BND table index.

The AUX_STORAGE storage parameter holds the DB2 CREATE TABLE configuration of the AUX table.

The BLK_STORAGE storage parameter holds the DB2 CREATE TABLE storage configuration of the BLK table.

```
##RASTER
AUX_STORAGE "IN <TABLESPACE> INDEX IN <TABLESPACE>"
BLK_STORAGE "IN <TABLESPACE> INDEX IN <TABLESPACE> LONG IN <TABLESPACE>"
BND_STORAGE "IN <TABLESPACE> INDEX IN <TABLESPACE>"
RAS_STORAGE "IN <TABLESPACE> INDEX IN <TABLESPACE>"
UI_TEXT     ""
END
```

Network class composite keywords

The composite keyword is a unique type of keyword designed to accommodate the tables of the ArcGIS network class. The network table's size variation requires a keyword that provides configuration 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 keyword is subdivided into elements. A network composite keyword has three elements: the parent element defines the general characteristic of the keyword and the junctions feature class, the description element defines the configuration of the DESCRIPTIONS table and its indexes, and the network element defines the configuration of the remaining network tables and their indexes.

The parent element does not have a suffix, and its keyword looks like any other keyword. The description element is demarcated by the addition of the ::DESC suffix to the parent element's keyword, and the network element is demarcated by addition of the ::NETWORK suffix to the parent element's keyword.

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

```
##ELECTRIC

COMMENT       This keyword is dedicated to the electrical geometric network class

UI_NETWORK_TEXT      "The electrical geometrical network class keyword"

B_STORAGE            "IN SDEDB2 INDEX IN SDEINDEX LONG IN SDELOBS"

B_INDEX_ROWID        "PCTFREE 15 DISALLOW REVERSE SCANS"

B_INDEX_USER         "PCTFREE 10 MINPCTUSED 25 DISALLOW REVERSE SCANS"

A_STORAGE            "IN SDEDB2 INDEX IN SDEINDEX LONG IN SDELOBS"

A_INDEX_ROWID        "PCTFREE 15 DISALLOW REVERSE SCANS"

A_INDEX_USER         "PCTFREE 10 MINPCTUSED 25 DISALLOW REVERSE SCANS"

A_INDEX_STATEID      ""

D_STORAGE            "IN SDEDB2 INDEX IN SDEINDEX LONG IN SDELOBS"

D_INDEX_DELETED_AT  ""

D_INDEX_STATE_ROWID  ""

END


##ELECTRIC::DESC

B_STORAGE            "IN SDEDB2 INDEX IN SDEINDEX LONG IN SDELOBS"

B_INDEX_ROWID        "PCTFREE 15 DISALLOW REVERSE SCANS"

B_INDEX_USER         "PCTFREE 10 MINPCTUSED 25 DISALLOW REVERSE SCANS"
```

```
A_STORAGE             "IN SDEDB2 INDEX IN SDEINDEX LONG IN SDELOBS"

A_INDEX_ROWID         "PCTFREE 15 DISALLOW REVERSE SCANS"

A_INDEX_USER          "PCTFREE 10 MINPCTUSED 25 DISALLOW REVERSE SCANS"

A_INDEX_STATEID       ""

D_STORAGE             "IN SDEDB2 INDEX IN SDEINDEX LONG IN SDELOBS"

D_INDEX_DELETED_AT  ""

D_INDEX_STATE_ROWID ""

END

##ELECTRIC::NETWORK

B_STORAGE             "IN SDEDB2 INDEX IN SDEINDEX LONG IN SDELOBS"

B_INDEX_ROWID         "PCTFREE 15 DISALLOW REVERSE SCANS"

B_INDEX_USER          "PCTFREE 10 MINPCTUSED 25 DISALLOW REVERSE SCANS"

A_STORAGE             "IN SDEDB2 INDEX IN SDEINDEX LONG IN SDELOBS"

A_INDEX_ROWID         "PCTFREE 15 DISALLOW REVERSE SCANS"

A_INDEX_USER          "PCTFREE 10 MINPCTUSED 25 DISALLOW REVERSE SCANS"

A_INDEX_STATEID       ""

D_STORAGE             "IN SDEDB2 INDEX IN SDEINDEX LONG IN SDELOBS"

D_INDEX_DELETED_AT  ""

D_INDEX_STATE_ROWID ""

END
```

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

```
DB2> 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              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 keyword, and the remaining smaller network tables are created with the ELECTRIC::NETWORK keyword.

The NETWORK_DEFAULTS keyword

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

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

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

```
##NETWORK_DEFAULTS
A_INDEX_ROWID           ""
A_INDEX_STATEID         ""
A_INDEX_USER            ""
#A_STORAGE              "IN <TABLESPACE> INDEX IN <TABLESPACE>"
B_INDEX_ROWID           ""
B_INDEX_USER            ""
#B_STORAGE              "IN <TABLESPACE> INDEX IN <TABLESPACE>"
COMMENT                         "The base system initialization parameters for
NETWORK_DEFAULTS"
D_INDEX_DELETED_AT      ""
D_INDEX_STATE_ROWID     ""
#D_STORAGE              "IN <TABLESPACE> INDEX IN <TABLESPACE>"
UI_NETWORK_TEXT         "The network default configuration"
END
```

```
##NETWORK_DEFAULTS::DESC
A_INDEX_ROWID          ""
A_INDEX_STATEID        ""
A_INDEX_USER           ""
#A_STORAGE             "IN <TABLESPACE> INDEX IN <TABLESPACE>"
B_INDEX_ROWID          ""
B_INDEX_USER           ""
#B_STORAGE             "IN <TABLESPACE> INDEX IN <TABLESPACE>"
D_INDEX_DELETED_AT     ""
D_INDEX_STATE_ROWID    ""
#D_STORAGE             "IN <TABLESPACE> INDEX IN <TABLESPACE>"
END

##NETWORK_DEFAULTS::NETWORK
A_INDEX_ROWID          ""
A_INDEX_STATEID        ""
A_INDEX_USER           ""
#A_STORAGE             "IN <TABLESPACE> INDEX IN <TABLESPACE>"
B_INDEX_ROWID          ""
B_INDEX_USER           ""
#B_STORAGE             "IN <TABLESPACE> INDEX IN <TABLESPACE>"
D_INDEX_DELETED_AT     ""
D_INDEX_STATE_ROWID    ""
#D_STORAGE             "IN <TABLESPACE> INDEX IN <TABLESPACE>"
END
```

DB2 default parameters

By default, DB2 stores tables and indexes in the user's default tablespace using the tablespace's default storage parameters. This tablespace is called USERSPACE1 in DB2.

Editing the DBTUNE table

To edit the storage parameters, the sdedbtune administration command allows you to export the DBTUNE table to a file located in the $SDEHOME/etc directory on UNIX® servers and in the %SDEHOME%\etc folder on Windows servers. It is an ArcSDE configuration file that contains DB2 table and index creation parameters. These parameters allow the ArcSDE service to communicate to the DB2 server such things as where a tablespace, table, or index will be created, as well as other parameters that can be set on either the CREATE TABLE or CREATE INDEX statement. The file can be edited with a UNIX file-based editor such as "vi" or a Windows NT® file-based editor such as "notepad". After updating the file, you can repopulate the DBTUNE table using the import operation of the sdedbtune command.

In the following example the DBTUNE table is exported to the dbtune.out file.

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

ArcSDE   8.3          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   8.3              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 in the etc directory of the ArcSDE home directory. You cannot relocate the file to another directory with a qualifying pathname. By not allowing the relocation of the file, the sdedbtune command ensures they remain under the ownership of the ArcSDE administrator.

The complete list of ArcSDE 8.3 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
B_STORAGE	<string>	Business table	DB2 defaults
B_INDEX_ROWID	<string>	Business table object ID column index	DB2 defaults
B_INDEX_USER	<string>	Business table user index(s)	DB2 defaults
A_STORAGE	<string>	Adds table	DB2 defaults
A_INDEX_ROWID	<string>	Adds table object ID column index	DB2 defaults
A_INDEX_STATEID	<string>	Adds table sde_state_id column index	DB2 defaults
A_INDEX_USER	<string>	Adds table index	DB2 defaults
D_STORAGE	<string>	Deletes table	DB2 defaults
D_INDEX_ STATE_ROWID	<string>	Deletes table sde_states_id and sde_deletes_row_id column index	DB2 defaults
D_INDEX_DELETED_AT	<string>	Deletes table sde_deleted_at column index	DB2 defaults
LOB_SIZE	<string>	Size of BLOB column	1 MB

Parameter Name	Value	Parameter Description	Default Value
LOB_OPTION	<string>	Storage configuration properties of the BLOB column	DB2 defaults
LF_STORAGE	<string>	Sde_logfiles table	DB2 defaults
LF_INDEXES	<string>	Sde_logfile table column indexes	DB2 defaults
LD_STORAGE	<string>	Sde_logfile_data table	DB2 defaults
LD_INDEX_DATA_ID	<string>	Sde_logfile_data table	DB2 defaults
LD_INDEX_ROWID	<string>	Sde_logfile_data table sde_row_id column index	DB2 defaults
RAS_STORAGE	<string>	Raster RAS table	DB2 defaults
BND_STORAGE	<string>	Raster BND table	DB2 defaults
AUX_STORAGE	<string>	Raster AUX table	DB2 defaults
BLK_STORAGE	<string>	Raster BLK table	DB2 defaults
UI_TEXT	<string>	User interface name of the configuration keyword	DB2 defaults
UI_NETWORK_TEXT	<string>	User interface name of the network configuration keyword	DB2 defaults
COMMENT	<string>	Comments	none

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 DB2 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 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.

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® GIS 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 graphical 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 to:

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 contain up to 1012 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 the user ‘beetle’.

The same table could be created with a SQL CREATE TABLE statement using the DB2 SQL interface.

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

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 system.

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

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 I/O 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. ArcMap requires data to be multiversioned to edit it. For further information on versioning data, refer to the *Building a Geodatabase* book.

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

```
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 group.

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 groups 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

The shp2sde command converts shapefiles into ArcSDE feature classes. The spatial column definition is read directly from the shapefile. You can use the shpinfo command to display the shapefile column definitions. In this example the -k option references the DBTUNE configuration keyword ROADS. The ROADS keyword contains storage parameters for storing the tables and indexes of the roads feature class.

```
shp2sde -o create -f rdshp -l roads,shape -k ROADS -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 DB2 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 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 understand how ArcSDE stores rasters in DB2, refer to Appendix A, '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 DB2 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 ArcSDE Developer Help for more information.

Exporting data

As with importing data, there are 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 DB2 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 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 ArcGIS Desktop documentation on ArcCatalog, ArcToolbox, and the geodatabase for a full discussion of these tools.

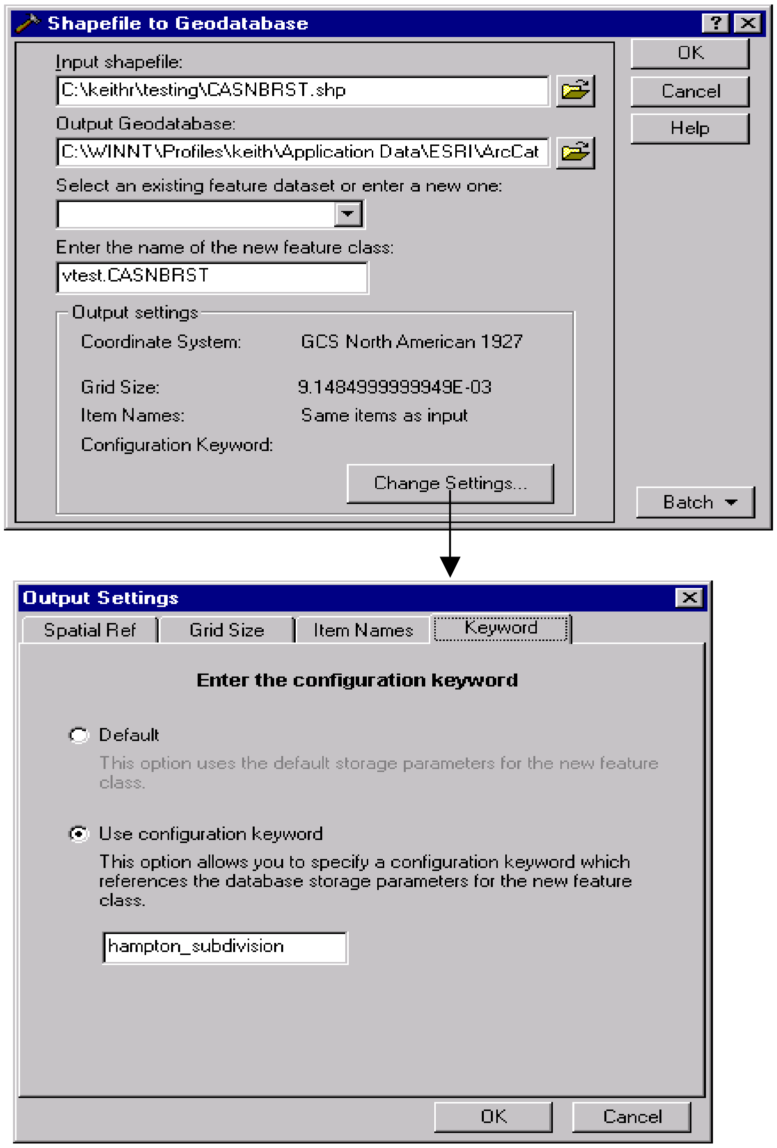
Loading data

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.

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.

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



Input shapefile: C:\keithr\testing\CASNBRST.shp

Output Geodatabase: C:\WINNT\Profiles\keith\Application Data\ESRI\ArcCat

Select an existing feature dataset or enter a new one:

Enter the name of the new feature class: vtest.CASNBRST

Output settings

Coordinate System:	GCS North American 1927
Grid Size:	9.1484999999949E-03
Item Names:	Same items as input
Configuration Keyword:	

Enter the configuration keyword

( ) Default

This option uses the default storage parameters for the new feature class.

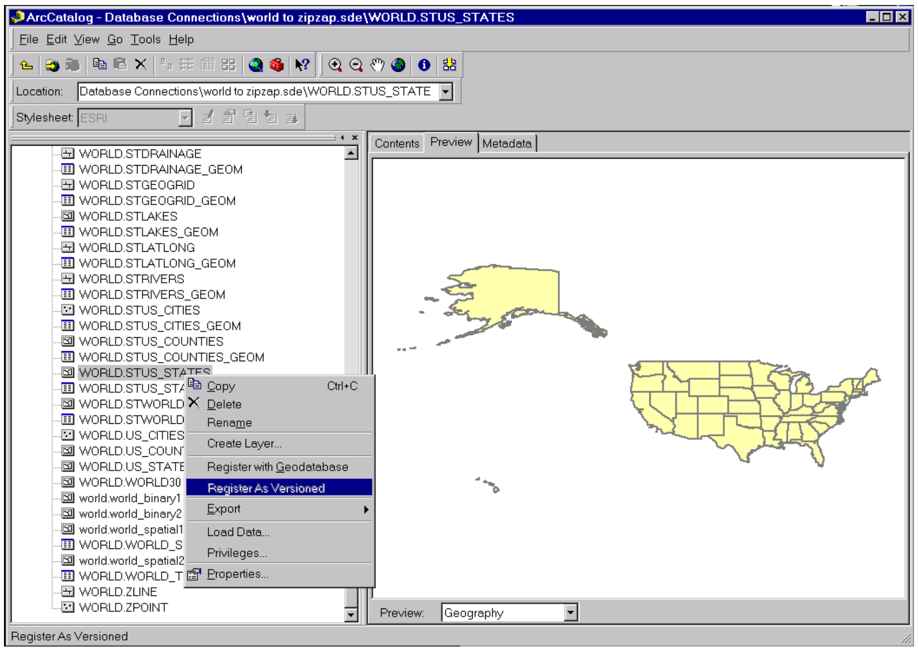
(•) Use configuration keyword

This option allows you to specify a configuration keyword which references the database storage parameters for the new feature class.

hampton_subdivision

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.



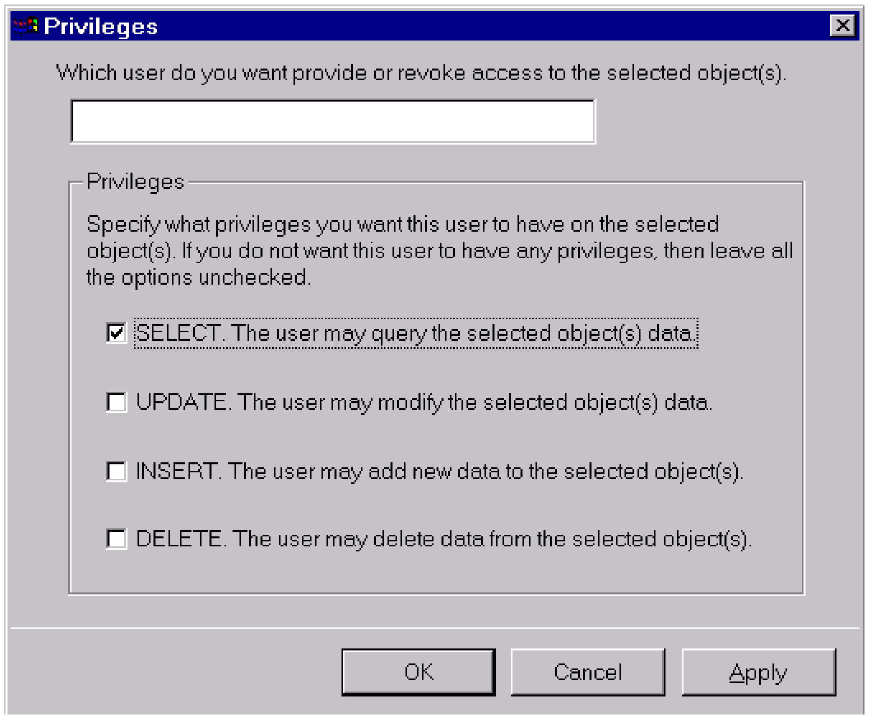
Location: Database Connections\world to zipzap.sde\WORLD.STUS_STATE

Preview: Geography

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.


Privileges

Which user do you want provide or revoke access to the selected object(s).

Privileges

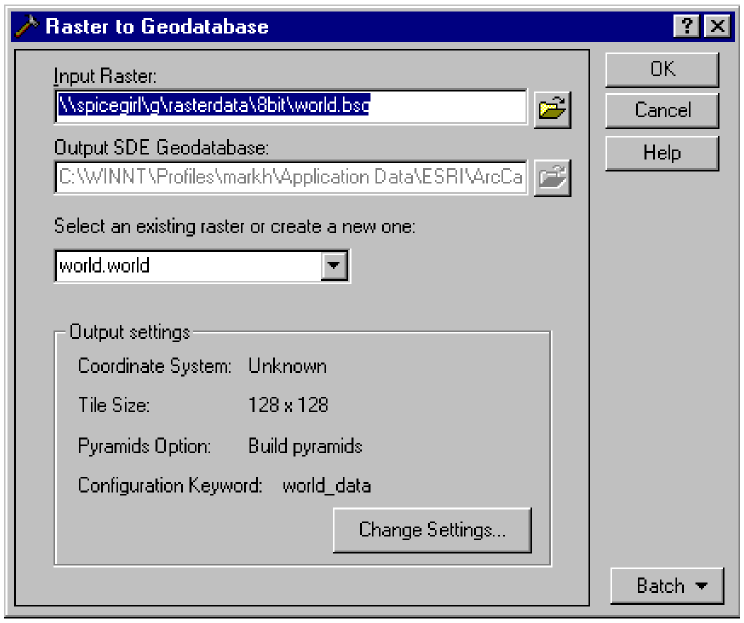
Specify what privileges you want this user to have on the selected object(s). If you do not want this user to have any privileges, then leave all the options unchecked.

- [✓] SELECT. The user may query 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).
- [ ] DELETE. The user may delete data from the selected object(s).

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 DB2 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

APPENDIX A

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.

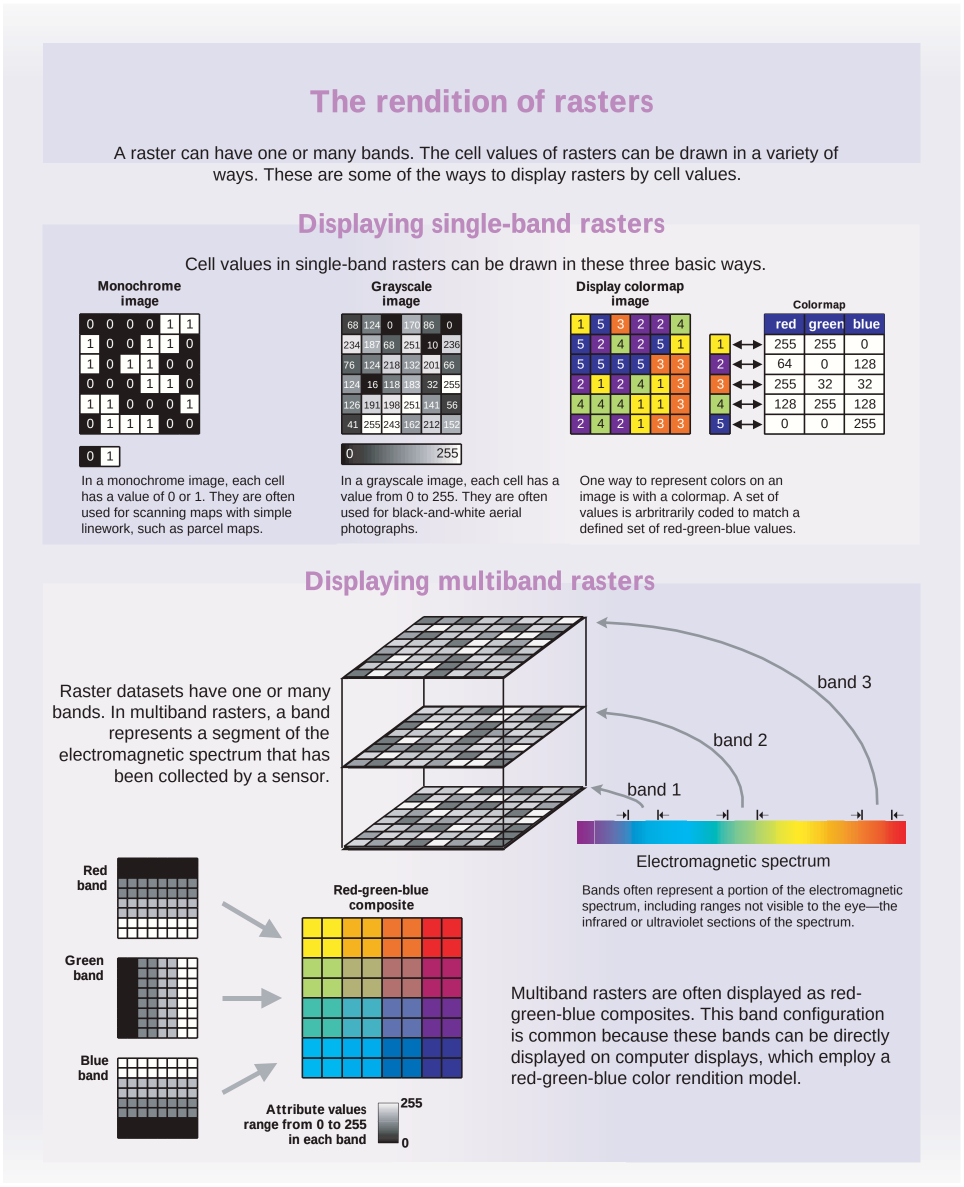
You are probably familiar with raster formats, such as tagged image file format (TIFF), Joint Photographic Experts Group (JPEG), and Graphics Interchange Format (GIF), that your Internet browser renders. These rasters are composed of one or more bands. Each band is segmented into a grid of square pixels. Each pixel is assigned a value that reflects the information it represents at a particular position.

For an expanded discussion of the type of raster data supported by ESRI products, review Chapter 9, ‘Cell-based modeling with rasters’, in *Modeling Our World*.

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 rendition of rasters

A raster can have one or many bands. The cell values of rasters can be drawn in a variety of ways. These are some of the ways to display rasters by cell values.

Displaying single-band rasters

Cell values in single-band rasters can be drawn in these three basic ways.

Monochrome image

0	0	0	0	1	1
1	0	0	1	1	0
1	0	1	1	0	0
0	0	0	1	1	0
1	1	0	0	0	1
0	1	1	1	0	0

In a monochrome image, each cell has a value of 0 or 1. They are often used for scanning maps with simple linework, such as parcel maps.

Grayscale image

68	124	0	170	86	0
234	187	68	251	10	236
76	124	218	132	201	66
124	16	118	183	32	255
126	191	198	251	141	56
41	255	243	162	212	152

In a grayscale image, each cell has a value from 0 to 255. They are often used for black-and-white aerial photographs.

Display colormap image

1	5	3	2	2	4
5	2	4	2	5	1
5	5	5	5	3	3
2	1	2	4	1	3
4	4	4	1	1	3
2	4	2	1	3	3

Colormap

	red	green	blue
1	255	255	0
2	64	0	128
3	255	32	32
4	128	255	128
5	0	0	255

One way to represent colors on an image is with a colormap. A set of values is arbritrarily coded to match a defined set of red-green-blue values.

Displaying multiband rasters

Raster datasets have one or many bands. In multiband rasters, a band represents a segment of the electromagnetic spectrum that has been collected by a sensor.

Bands often represent a portion of the electromagnetic spectrum, including ranges not visible to the eye—the infrared or ultraviolet sections of the spectrum.

Multiband rasters are often displayed as red-green-blue composites. This band configuration is common because these bands can be directly displayed on computer displays, which employ a red-green-blue color rendition model.

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 the compression method, if one is specified, determine the storage size of each raster block.

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 height of the pyramid is determined by the number of levels specified by the application. ArcToolbox and ArcCatalog calculate the pyramid for you, so there is no need to define the number of levels.

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.

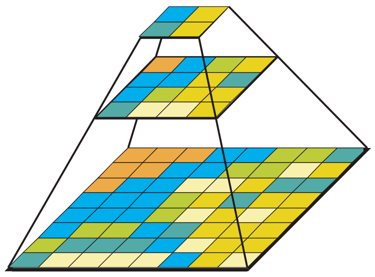


Figure A.1 When you build a pyramid, more rasters are created by progressively downsampling the previous level by a factor of two until the apex is reached. As the application zooms out and the raster cells grow smaller than the resolution threshold, ArcSDE selects a higher level of the pyramid. The 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 DB2'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 rasterband_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.

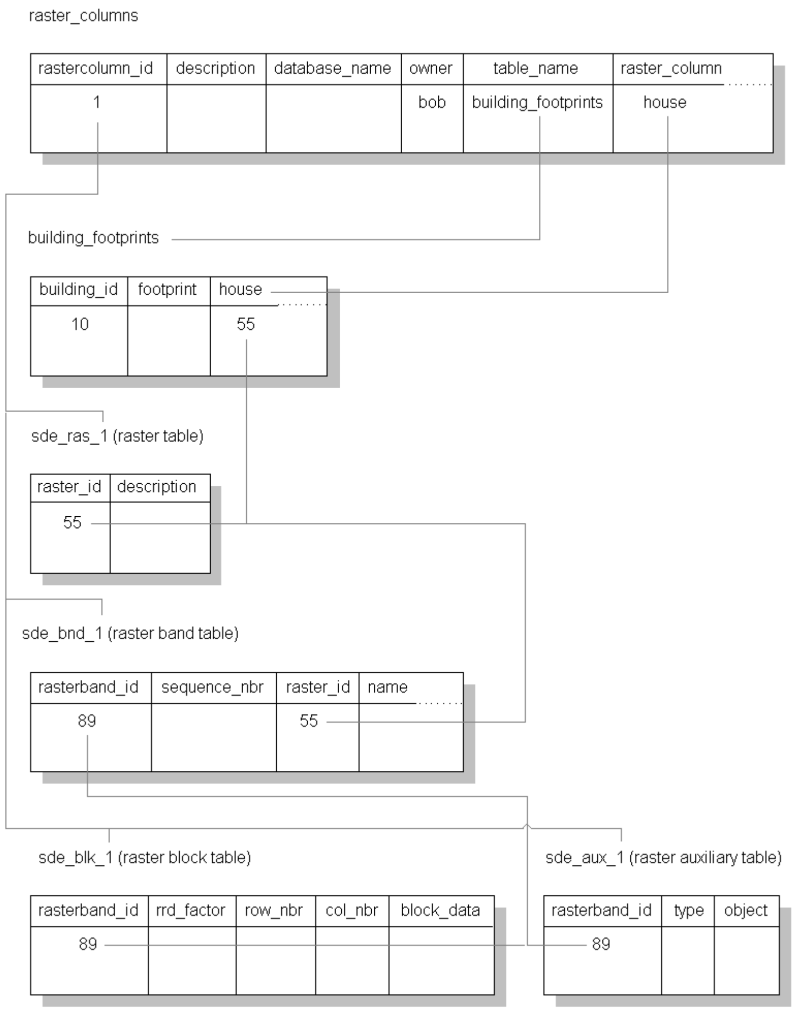


Figure A.2 When ArcSDE adds a raster column to a table, it records that column in the sde user's raster_columns table. The rastercolumn_id table is used in the creation of the table names of the raster, raster band, raster auxiliary, and raster blocks table.

The sections that follow describe the schema of the tables associated with the storage of raster data. Refer to Figure A.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	INTEGER	NOT NULL
description	VARCHAR(65)	NULL

database_name	VARCHAR(32)	NULL
owner	VARCHAR(32)	NOT NULL
table_name	VARCHAR(128)	NOT NULL
raster_column	VARCHAR(128)	NOT NULL
cdate	INTEGER	NOT NULL
config_keyword	VARCHAR(32)	NULL
minimum_id	INTEGER	NULL
base_rastercolumn_id	INTEGER	NOT NULL
rastercolumn_mask	INTEGER	NOT NULL
srid	INTEGER	NULL

Raster columns table

- rastercolumn_id (SE_INTEGER_TYPE)—The table's primary key.
- description (SE_STRING_TYPE)—The description of the raster table.
- database_name (SE_STRING_TYPE)—The DB2 database name.
- owner (SE_STRING_TYPE)—The schema 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 DB2 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, it establishes the 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 DB2GSE.GSE_SPATIAL_REF 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 users 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	INTEGER	NOT NULL
building_footprint	INTEGER	NOT NULL
house_picture	INTEGER	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 DB2 database, stores a record for each image stored in a raster column. The raster_column_id column 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	INTEGER	NOT NULL
raster_flags	INTEGER	NULL
description	VARCHAR(65)	NULL

Raster description table schema (SDE_RAS_<raster_column_id>)

- raster_id (SE_INTEGER_TYPE)—the primary key of the raster table and unique sequential identifier of each image stored in the raster table
- raster_flags (SE_INTEGER_TYPE)—a bitmap set according to the characteristics of 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	INTEGER	NOT NULL
sequence_nbr	INTEGER	NOT NULL
raster_id	INTEGER	NOT NULL
name	VARCHAR(65)	NULL
band_flags	INTEGER	NOT NULL
band_width	INTEGER	NOT NULL
band_height	INTEGER	NOT NULL
band_types	INTEGER	NOT NULL
block_width	INTEGER	NOT NULL
block_height	INTEGER	NOT NULL
block_origin_x	DOUBLE	NOT NULL
block_origin_y	DOUBLE	NOT NULL
eminx	DOUBLE	NOT NULL
eminy	DOUBLE	NOT NULL
emaxx	DOUBLE	NOT NULL
emaxy	DOUBLE	NOT NULL
cdate	INTEGER	NOT NULL
mdate	INTEGER	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_DOUBLE_TYPE)—The leftmost pixel.
- block_origin_y (SE_DOUBLE_TYPE)—The bottom-most pixel.

If the image has a map extent, the optional eminx, eminy, emaxx, and emaxy will hold the coordinates of the extent.

- eminx (SE_DOUBLE_TYPE)—the band's minimum x-coordinate.
- eminy (SE_DOUBLE_TYPE)—the band's minimum y-coordinate.
- emaxx (SE_DOUBLE_TYPE)—the band's maximum x-coordinate.
- emaxy (SE_DOUBLE_TYPE)—the band's maximum y-coordinate
- cdate (SE_INTEGER_TYPE)—the creation date
- mdate (SE_INTEGER_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 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	INTEGER	NOT NULL
rrd_factor	INTEGER	NOT NULL
row_nbr	INTEGER	NOT NULL
col_nbr	INTEGER	NOT NULL
block_data	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	INTEGER	NOT NULL
type	INTEGER	NOT NULL
object	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.

APPENDIX B

DB2 Spatial Extender geometry types

ArcSDE for DB2 stores its spatial data in the DB2 Spatial Extender® data types. Therefore, before ArcSDE can store spatial data in a DB2 database, the Spatial Extender must be installed and the database must be spatially enabled. This document describes the ArcSDE/DB2 Spatial Extender interface and provides a brief overview of the spatial data types and functions available after the database has been spatially enabled with the DB2 Spatial Extender. For more information about the DB2 Spatial Extender, see the IBM *DB2 Spatial Extender User's Guide and Reference.*

The DB2 Spatial Extender embeds a GIS into your DB2 database. The DB2 Spatial Extender module implements the Open GIS Consortium, Inc. (OpenGIS®, or OGC) SQL 3 specification of user-defined types (UDTs), columns capable of storing spatial data such as the location of a landmark, a street, or a parcel of land.

The GIS of the past was spatially centric and focused on gathering spatial data and attaching nonspatial 'attribute' data to it. The Spatial Extender module integrates spatial and nonspatial data, providing a seamless point of access through the DB2 Structured Query Language (SQL) interface.

In addition to new data types, the DB2 Spatial Extender provides new capabilities such as spatial joins. Application programmers typically join tables by comparing two or more columns to determine whether their values are equal, not equal, greater than, and so on. The DB2 Spatial Extender includes functions capable of comparing the values of spatial columns to determine if they intersect, overlap, and so forth. These two-dimensional functions can join tables based on their spatial relationship and answer questions such as "Is this school within five miles of a hazardous waste site?" Internally, the DB2 Spatial Extender ST_Overlaps function evaluates this question as, "Does this polygon (the building footprint of a school) overlap this circular polygon (the five-mile radius of a hazardous waste site)?" An application programmer can join a table storing sensitive sites, such as schools, playgrounds, and hospitals, to another table containing the locations of hazardous sites and return a list of sensitive areas at risk.

How the DB2 Spatial Extender works

Once the DB2 Spatial Extender is installed, you can create spatially enabled tables that include spatial columns. Geographic features can be inserted into the spatial columns. The DB2 Spatial Extender converts spatial data into its storage format from one of three external formats:

- Well-known text (WKT) representation
- Well-known binary (WKB) representation
- ESRI shape representation

ArcSDE uses the ESRI shape representation.

Accessing the spatially enabled tables through the ArcSDE server allows you to write applications using the existing tools offered by the GIS software or create applications using the Spatial Database Engine™ (SDE®) C API. An experienced Open Database Connectivity (ODBC) programmer can also make calls to the DB2 Spatial Extender spatial functions. The majority of this document is devoted to discussing and applying these spatial functions.

After integrating spatial data into your database, you can include Spatial Extender functions in your SQL statements, comparing the values of spatial columns, transforming the values into other spatial data, and describing the properties of the data.

Adding records to the DB2GSE.GSE_SPATIAL_REF table

The spatial reference system identifies the coordinate transformation matrix for each geometry. Geometry is the term adopted by the Open GIS Consortium to refer to two-dimensional spatial data. All spatial reference systems known to the database are stored in the DB2GSE.GSE_SPATIAL_REF table.

NAME	DATA TYPE	NULL?
srid	integer	NOT NULL
sr_name	varchar(64)	NOT NULL
scid	integer	NOT NULL
falsex	double	NOT NULL
falsey	double	NOT NULL
xyunits	double	NOT NULL
falsez	double	NOT NULL
zunits	double	NOT NULL
falsem	double	NOT NULL
munits	double	NOT NULL

DB2GSE.GSE_SPATIAL_REF table schema

The DB2GSE.GSE_SPATIAL_REF table stores a record for each spatial reference in the database.

The datatype for each column is defined below.

- srid (SE_INTEGER_TYPE)—Contains the unique ID that identifies each SRID in the database.
- sr_name (SE_STRING_TYPE)—The name of the spatial reference system.
- scid (SE_INTEGER_TYPE)—The ID of the spatial reference system's coordinate reference system. This is a foreign key to the DB2GSE.COORD_REF_SYS table's primary key. The DB2GSE.COORD_REF_SYS table is populated when the database is spatially enabled.
- falsex (SE_DOUBLE_TYPE)—The x-value offset or the minimum allowable X-ordinate value.
- falsey (SE_DOUBLE_TYPE)—The y-value offset or the minimum allowable Y-ordinate value.
- xyunits (SE_DOUBLE_TYPE)—The XY coordinate system units or spatial reference system's XY coordinate precision. Coordinates whose precision exceeds this value are truncated when they are stored.
- falsez (SE_DOUBLE_TYPE)—The z-value offset or the minimum allowable Z-ordinate value.

- zunits (SE_DOUBLE_TYPE)—The z-coordinate system units or spatial reference system's z-coordinate precision. Coordinates whose precision exceeds this value are truncated when they are stored.
- falsem (SE_DOUBLE_TYPE)—The m-value offset or the minimum allowable M-ordinate value.
- munits (SE_DOUBLE_TYPE)—The m-coordinate system units or spatial reference system's m-coordinate precision. Coordinates whose precision exceeds this value are truncated when they are stored.

Internal functions use the parameters of a spatial reference system to translate and scale each floating point coordinate of the geometry into 32-bit positive integers prior to storage. Upon retrieval, the coordinates are restored to their external floating point format.

The floating point coordinates are converted to integers by subtracting the falsex and falsey values, which translates to the false origin, scales by multiplying by the xyunits, adds a half unit, and truncates the remainder.

The optional z-coordinates and measures are dealt with similarly, except that they are translated with falsez and falsem and scaled with zunits and munits, respectively.

The spatial reference identifier, the primary key, contains a unique number for each spatial reference system.

The spatial reference system is assigned to a geometry during its construction. The spatial reference system must exist in the spatial reference table. All geometries in a column must have the same spatial reference system.

Whenever ArcSDE creates a feature class it searches the DB2GSE.GSE_SPATIAL_REF table in an attempt to locate a matching spatial reference system. If one is found the SRID is assigned to the feature class; otherwise, ArcSDE adds a new spatial reference system to the DB2GSE.GSE_SPATIAL_REF table and assigns it to the feature class.

The ArcSDE administration tools shp2sde columns and cov2sde columns provide an option for you to enter a predefined SRID when you use them to create a new feature class. In this example, the roads coverage is converted to the roads feature class with a SRID of 10. The coordinates of the coverage feature must fit within the extent of the spatial reference system. Each feature found to lie outside the spatial reference system's extent is rejected.

```
cov2sde -o create -l roads,feature -f roads -R 10 -g 100,0,0 -u world -p world
```

Creating feature classes in a DB2 database

A DB2 spatial table can include one or more spatial columns, although ArcSDE restricts a feature class to a single spatial column. Spatial columns are defined with one of the DB2 Spatial Extender's UDTs. A spatial column can only accept data of the type required by the spatial column. For example, an ST_Polygon column rejects integers, characters, and even other types of nonpolygon geometry.

When ArcSDE creates a DB2 table with a spatial column, it also creates an SE_ROW_ID integer column. The SE_ROW_ID column is required by ArcSDE client applications to keep track of selection sets; more specifically it is used in ArcSDE log files.

ArcSDE adds a record to the DB2GSE.GEOMETRY_COLUMNS table whenever it creates a feature class in a DB2 database. Applications using the DB2 Spatial Extender are responsible for inserting a record into the DB2GSE.GEOMETRY_COLUMNS table each time they add a spatial column to the database.

NAME	DATA TYPE	NULL?
layer_catalog	varchar(30)	NOT NULL
layer_schema	varchar(30)	NOT NULL
layer_table	varchar(128)	NOT NULL
layer_column	varchar(128)	NOT NULL
geometry_type	integer	NOT NULL
srid	integer	NOT NULL

Geometry_columns table schema

The DB2GSE.GEOMETRY_COLUMNS table stores a record for each geometry column in the database.

The datatype for each column is defined below.

- layer_catalog (SE_STRING_TYPE)—The database in which the geometry column's table is stored.
- layer_schema (SE_STRING_TYPE)—The owner of the geometry column's table.
- layer_table (SE_STRING_TYPE)—The geometry column's table name.
- layer_column (SE_STRING_TYPE)—The name of the geometry column.
- geometry_type (SE_INTEGER_TYPE)—The geometry type code. ArcSDE inserts the following values into this field:

Geometry Type Code	Geometry Type
0	ST_Geometry
1	ST_Point
3	ST_LineString
5	ST_Polygon
7	ST_MultiPoint
9	ST_MultiLineString
11	ST_MultiPolygon

- srid (SE_INTEGER_TYPE)—The geometry column's spatial reference system. This is a foreign key to the SRID column of the DB2GSE.GSE_SPATIAL_REF table.

Creating a spatial index

Spatial columns contain two-dimensional geographic data, and applications querying those columns require an index strategy that will quickly identify all geometries that lie within a given extent. For this reason DB2 Spatial Extender provides support for the creation of a three-level grid spatial index.

From the DB2 command line create a spatial index on a spatial column with the Spatial Extender stored procedure DB2GSE.GSE_ENABLE_IDX documented in *IBM DB2 Spatial Extender User's Guide and Reference.*

Note also that ArcCatalog and ArcSDE administration tools, sdelayer, shp2sde, and cov2sde, provide support for creating the spatial index.

See Chapter 2, 'Essential configuring and tuning', for a discussion on selecting the spatial index's grid cell sizes.

Updating statistics

The DB2 optimizer may not use the spatial index unless the statistics on the table are up-to-date. If the spatial index is created after the data has been loaded, the statistics are up-to-date and the optimizer will use the index. However, if the index is created, and data is loaded afterwards, the optimizer will not use the spatial index because the statistics will be out of date. To update the statistics use the update statistics DB2 SQL statement.

```
RUNSTATS ON TABLE <table_name> WITH DISTRIBUTION AND DETAILED INDEXES ALL;
```

When updating statistics for ArcSDE feature classes, you should use the tools provided by either ArcCatalog or the update_dbms_stats operation of the ArcSDE administration tool sdetable. For more information on using these tools to update statistics, see Chapter 2, 'Essential configuring and tuning'.

Spatial Extender data types

The *Oxford American Dictionary* defines the noun 'geometry' as "the branch of mathematics dealing with the properties of and relations of lines, angles, surfaces, and solids." On August 11, 1997, the OGC, in its publication of *OpenGIS Features for ODBC (SQL) Implementation Specification*, coined another definition for the noun geometry. The word was selected to define the geometric features that, for the past millennium or more, cartographers have used to map the world. Typically, points represent an object at a single location, linestrings represent a linear characteristic, and polygons represent a spatial extent. A very abstract definition of the Open GIS noun geometry might be "a point or aggregate of points symbolizing a feature on the ground". This definition, however, fails to describe the rich set of properties and functionality associated with geometry.

To understand geometry in this context, it is easier to describe it as it has been implemented within the DB2 Spatial Extender a UDT, and like all UDTs in an object relational system, it has a unique set of properties and methods.

ST_Geometry columns as a data type allow you to define columns that store spatial data. The ST_Geometry data type itself is an abstract noninstantiable superclass, the subclasses of which are instantiable. An instantiated data type is one that can be defined as a table column and have values of its type inserted into it. A column can be defined as type ST_Geometry, but ST_Geometry values cannot be inserted into it since they cannot be instantiated. Only the subclass values can be inserted into this column because only they can be instantiated. Therefore, the ST_Geometry data type can accept and store any of its subclasses, while its subclass data types can only accept their own values.

Throughout the remainder of this document the term geometry or geometries collectively refers to the superclass ST_Geometry data type and all of its subclass data types. Whenever it is necessary to specify the geometry superclass directly, it will be referred to as the ST_Geometry superclass or the ST_Geometry data type.

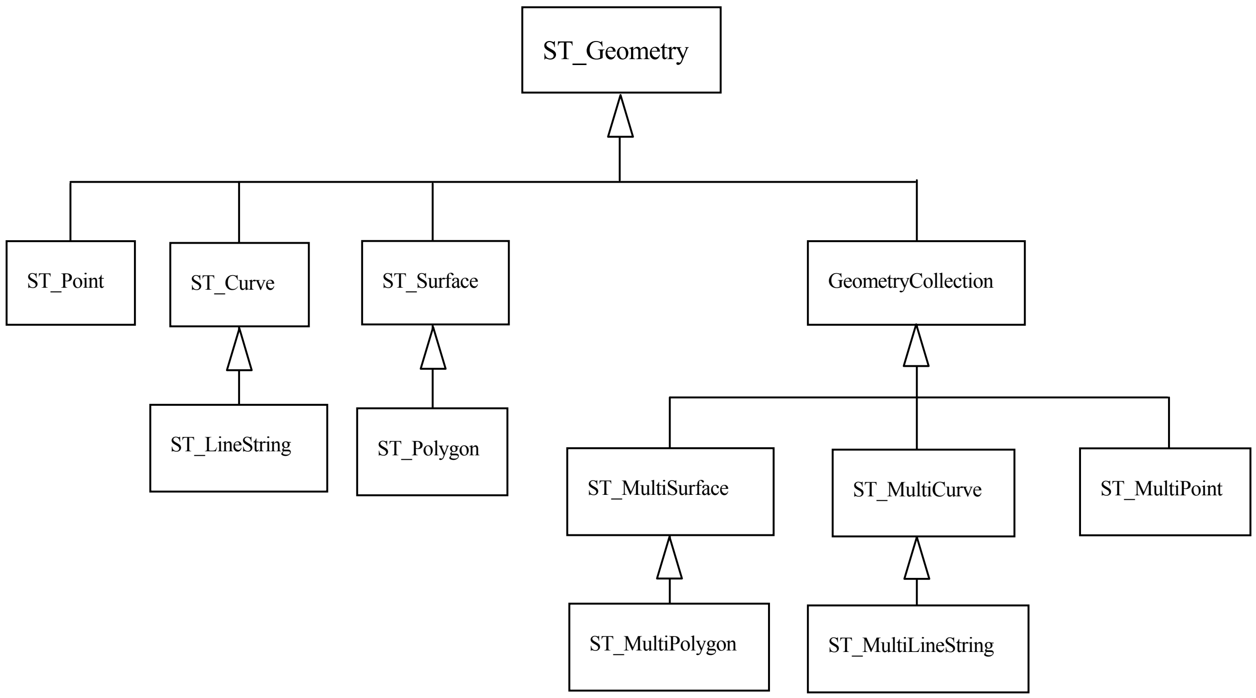


Figure B.1 The hierarchy of the ST_Geometry datatype is divided into the subtypes ST_Point, ST_Curve and ST_Surface simple types and the geometry collections ST_MultiSurface, ST_MultiCurve, and ST_MultiPoint. ST_LineString is the subtype of ST_Curve. ST_Polygon is the subtype of ST_Surface. ST_MultiPolygon is the subtype of ST_MultiSurface. ST_MultiLineString is the subtype of ST_MultiCurve.

Geometry properties

Each subclass inherits the properties of the ST_Geometry superclass but also has properties of its own. Functions that operate on the ST_Geometry data type will accept any of the subclass data types. However, some functions have been defined at the subclass level and will only accept certain subclasses' data types.

Interior, boundary, exterior

All geometries occupy a position in space defined by their interior, boundary, and exterior. The exterior of a geometry is all space not occupied by the geometry. The boundary of a geometry serves as the interface between its interior and exterior. The interior is the space occupied by the geometry. The subclass inherits the interior and exterior properties directly; however, the boundary property differs for each.

The ST_Boundary Spatial Extender function takes an ST_Geometry and returns an ST_Geometry that represents the source ST_Geometry's boundary.

Simple or nonsimple

Some subclasses of ST_Geometry (ST_LineStrings, ST_MultiPoints, and ST_MultiLineStrings) are either simple or nonsimple. They are simple if they obey all topological rules imposed on the subclass and nonsimple if they "bend" a few. An ST_LineString is simple if it does not intersect its interior. An ST_MultiPoint is simple if none of its elements occupy the same coordinate space. An ST_MultiLineString is simple if none of its element's interiors are intersected by its own interior.

The Spatial Extender ST_IsSimple predicate function takes an ST_Geometry and returns 1 (TRUE) if the ST_Geometry is simple and 0 (FALSE) otherwise.

Empty or not empty

A geometry is empty if it does not have any points. An empty geometry has a NULL envelope, boundary, interior, and exterior. An empty geometry is always simple and can have z-coordinates or measures. Empty linestrings and multilinestrings have a 0 length. Empty polygons and multipolygons have a 0 area.

The Spatial Extender ST_IsEmpty predicate function takes an ST_Geometry and returns 1 (TRUE) if the ST_Geometry is empty and 0 (FALSE) otherwise.

Number of points

A geometry can have zero or more points. A geometry is considered empty if it has zero points. The point subclass is the only geometry that is restricted to zero or one point; all other subclasses can have zero or more.

Envelope

The envelope of a geometry is the bounding geometry formed by the minimum and maximum (x,y) coordinates. The envelopes of most geometries form a boundary rectangle; however, the envelope of a point is the point since its minimum and maximum coordinates are the same, and the envelope of a horizontal or vertical linestring is a linestring represented by the boundary (the endpoints) of the source linestring.

The Spatial Extender ST_Envelope function takes an ST_Geometry and returns an ST_Geometry that represents the source ST_Geometry's envelope.

Dimension

A geometry can have a dimension of 0, 1, or 2.

The dimensions are

0—has neither length nor area

1—has a length

2—contains area

The point and multipoint subclasses have a dimension of 0. Points represent zero-dimensional features that can be modeled with a single coordinate, while multipoints represent data that must be modeled with a cluster of unconnected coordinates.

The subclasses linestring and multilinestring have a dimension of 1. They store road segments, branching river systems, and any other features that are linear in nature.

Polygon and multipolygon subclasses have a dimension of 2. Forest stands, parcels, water bodies, and other features whose perimeter encloses a definable area can be rendered by either the polygon or multipolygon data type.

Dimension is important not only as a property of the subclass but also in playing a part in determining the spatial relationship of two features. The dimension of the resulting feature or features determines whether or not the operation was successful. The dimension of the features is examined to determine how they should be compared.

The Spatial Extender ST_Dimension function takes an ST_Geometry and returns its dimension as an integer.

Z-coordinates

Some geometries have an associated altitude or depth. Each of the points that form the geometry of a feature can include an optional z-coordinate that represents an altitude or depth normal to the earth's surface.

The Spatial Extender Is3D predicate function takes an ST_Geometry and returns 1 (TRUE) if the function has z-coordinates and 0 (FALSE) otherwise.

Measures

Measures are values assigned to each coordinate. The value represents anything that can be stored as a double-precision number.

The Spatial Extender IsMeasured predicate function takes a geometry and returns 1 (TRUE) if it contains measures and 0 (FALSE) otherwise.

Spatial reference system

The spatial reference system identifies the coordinate transformation matrix for each geometry.

The Spatial Extender ST_SRID function takes an ST_Geometry and returns its spatial reference identifier as an integer.

Instantiable subclasses

The ST_Geometry data type is not instantiable but instead must store its instantiable subclasses. The subclasses are divided into two categories: the base geometry subclasses and the homogeneous collection subclasses. The base geometries include ST_Point, ST_LineString, and ST_Polygon, while the homogeneous collections include ST_MultiPoint, ST_MultiLineString, and ST_MultiPolygon. As the names imply, the homogeneous collections are collections of base geometries. In addition to sharing base geometry properties, homogeneous collections have some of their own properties as well.

The Spatial Extender ST_GeometryType function takes an ST_Geometry and returns the instantiable subclass in the form of a character string. The Spatial Extender ST_NumGeometries function takes a homogeneous collection and returns the number of base geometry elements it contains. The Spatial Extender ST_GeometryN function takes a homogeneous collection and an index and returns the nth base geometry.

ST_Point

An ST_Point is a zero-dimensional geometry that occupies a single location in coordinate space. An ST_Point has a single x,y coordinate value. An ST_Point is always simple and has a NULL boundary. It is used to define features such as oil wells, landmarks, and elevations.

Spatial Extender functions that operate solely on the ST_Point data type include ST_X, ST_Y, Z, and M.

The ST_X function returns a point data type's x coordinate value as a double-precision number.

The ST_Y function returns a point data type's y coordinate value as a double-precision number.

The Z function returns a point data type's z coordinate value as a double-precision number.

The M function returns a point data type's m coordinate value as a double-precision number.

ST_LineString

An ST_LineString is a one-dimensional object stored as a sequence of points defining a linear interpolated path. The ST_LineString is simple if it does not intersect its interior. The endpoints (the boundary) of a closed ST_LineString occupy the same point in space. An ST_LineString is a ring if it is both closed and simple. As well as the other properties inherited from the superclass ST_Geometry, ST_LineStrings have length. ST_LineStrings are often used to define linear features such as roads, rivers, and power lines.

The endpoints normally form the boundary of an ST_LineString unless the ST_LineString is closed, in which case the boundary is NULL. The interior of an ST_LineString is the connected path that lies between the endpoints, unless it is closed, in which case the interior is continuous.

Spatial Extender functions that operate on ST_LineStrings include ST_StartPoint, ST_EndPoint, ST_PointN, ST_Length, ST_NumPoints, ST_IsRing, and ST_IsClosed.

The ST_StartPoint function takes an ST_LineString and returns its first point.

The ST_EndPoint function takes an ST_LineString and returns its last point.

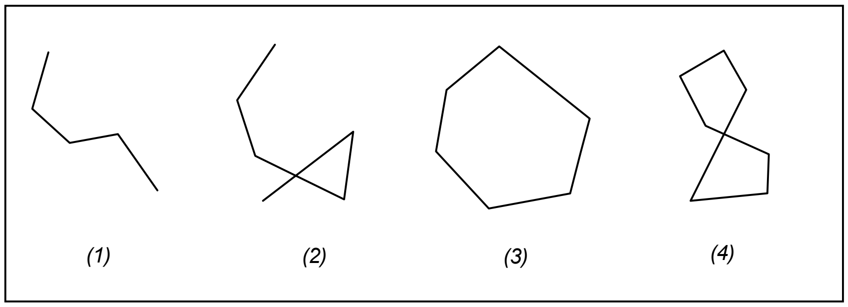
The ST_PointN function takes an ST_LineString and an index to an nth point and returns that point.

The ST_Length function takes an ST_LineString and returns its length as a double-precision number.

The ST_NumPoints function takes an ST_LineString and returns the number of points in its sequence as an integer.

The ST_IsRing predicate function takes an ST_LineString and returns 1 (TRUE) if the ST_LineString is a ring and 0 (FALSE) otherwise.

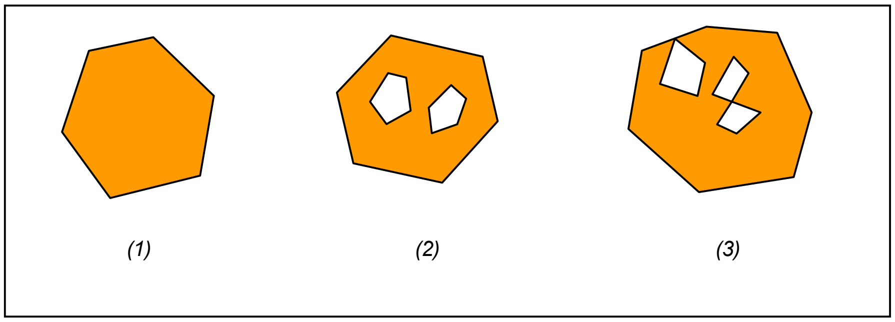
The ST_IsClosed predicate function takes an ST_LineString and returns 1 (TRUE) if the ST_LineString is closed and 0 (FALSE) otherwise.



Examples of ST_LineString objects: (1) a simple nonclosed ST_LineString, (2) a nonsimple nonclosed ST_LineString, (3) a closed simple ST_LineString and therefore is a ring, and (4) a closed nonsimple ST_LineString and is not a ring.

ST_Polygon

An ST_Polygon is a two-dimensional surface stored as a sequence of points defining its exterior bounding ring and 0 or more interior rings. ST_Polygon, by definition, is always simple. Most often ST_Polygon defines parcels of land, water bodies, and other features having spatial extent.



Examples of ST_Polygon objects: (1) an ST_Polygon whose boundary is defined by an exterior ring; (2) an ST_Polygon whose boundary is defined by an exterior ring and two interior rings, and the area inside the interior rings is part of the ST_Polygon's exterior; and (3) a legal ST_Polygon because the rings intersect at a single tangent point.

The exterior and any interior rings define the boundary of an ST_Polygon, and the space enclosed between the rings defines the ST_Polygon's interior. The rings of an ST_Polygon can intersect at a tangent point but never cross. In addition to the other properties inherited from the superclass ST_Geometry, ST_Polygon has area.

Spatial Extender functions that operate on ST_Polygon include ST_Area, ST_ExteriorRing, ST_NumInteriorRing, ST_InteriorRingN, ST_Centroid, and ST_PointOnSurface.

The ST_Area function takes an ST_Polygon and returns its area as a double-precision number.

The ST_ExteriorRing function takes an ST_Polygon and returns its exterior ring as an ST_LineString.

The ST_NumInteriorRing takes an ST_Polygon and returns the number of interior rings that it contains.

The ST_InteriorRingN function takes an ST_Polygon and an index and returns the nth interior ring as an ST_LineString.

The ST_Centroid function takes an ST_Polygon and returns an ST_Point that is the center of the ST_Polygon's envelope.

The ST_PointOnSurface function takes an ST_Polygon and returns an ST_Point that is guaranteed to be on the surface of the ST_Polygon.

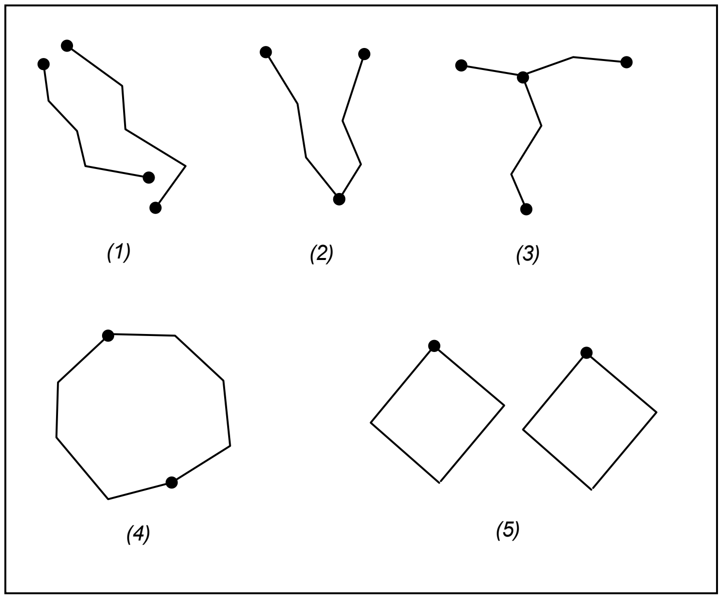
ST_MultiPoint

An ST_MultiPoint is a collection of ST_Points and, just like its elements, it has a dimension of 0. An ST_MultiPoint is simple if none of its elements occupy the same coordinate space. The boundary of an ST_MultiPoint is NULL. ST_MultiPoints define aerial broadcast patterns and incidents of a disease outbreak.

ST_MultiLineString

An ST_MultiLineString is an collection of ST_LineStrings. ST_MultiLineStrings are simple if they only intersect at the endpoints of the ST_LineString elements. ST_MultiLineStrings are nonsimple if the interiors of the ST_LineString elements intersect.

The boundary of an ST_MultiLineString is the nonintersected endpoints of the ST_LineString elements. The ST_MultiLineString is closed if all its ST_LineString elements are closed. The boundary of an ST_MultiLineString is NULL if all the endpoints of all the elements are intersected. In addition to the other properties inherited from the superclass ST_Geometry, ST_MultiLineStrings have length. ST_MultiLineStrings are used to define streams or road networks.



Examples of ST_MultiLineStrings: (1) a simple ST_MultiLineString whose boundary is the four endpoints of its two ST_LineString elements; (2) a simple ST_MultiLineString because only the endpoints of the ST_LineString elements intersect. The boundary is two nonintersected endpoints; (3) a nonsimple ST_MultiLineString because the interior of one of its ST_LineString elements is intersected. The boundary of this ST_MultiLineString is the three nonintersected endpoints; (4) a simple nonclosed ST_MultiLineString. It is not closed because its element ST_LineStrings are not closed. It is simple because none of the interiors of any of the element ST_LineStrings intersect; (5) a simple closed ST_MultiLineString. It is closed because all its elements are closed. It is simple because none of its elements intersect at the interiors.

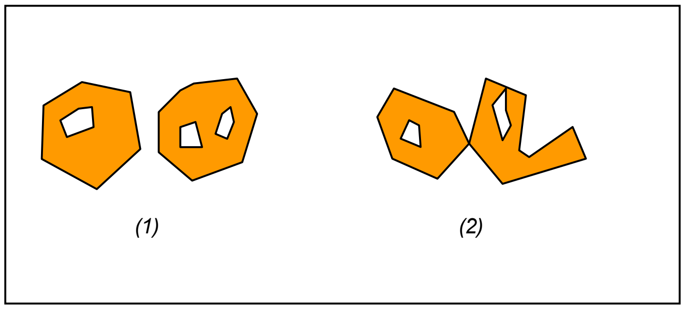
Spatial Extender functions that operate on ST_MultiLineStrings include ST_Length and ST_IsClosed.

The ST_Length function takes an ST_MultiLineString and returns the cumulative length of all its ST_LineString elements as a double-precision number.

The ST_IsClosed predicate function takes an ST_MultiLineString and returns 1 (TRUE) if the ST_MultiLineString is closed and 0 (FALSE) otherwise.

ST_MultiPolygon

The boundary of an ST_MultiPolygon is the cumulative length of its elements' exterior and interior rings. The interior of an ST_MultiPolygon is defined as the cumulative interiors of its element ST_Polygons. The boundary of an ST_MultiPolygon's elements can only intersect at a tangent point. In addition to the other properties inherited from the superclass ST_Geometry, ST_MultiPolygons have area. ST_MultiPolygons define features such as a forest stratum or a noncontiguous parcel of land such as a Pacific island chain.



Examples of ST_MultiPolygon: (1) an ST_MultiPolygon with two ST_Polygon elements. The boundary is defined by the two exterior rings and the three interior rings; and (2) an ST_MultiPolygon with two ST_Polygon elements. The boundary is defined by the two exterior rings and the two interior rings. The two ST_Polygon elements intersect at a tangent point.

Spatial Extender functions that operate on ST_MultiPolygons include ST_Area, ST_Centroid, and ST_PointOnSurface.

The ST_Area function takes an ST_MultiPolygon and returns the cumulative ST_Area of its ST_Polygon elements as a double-precision number.

The ST_Centroid function takes an ST_MultiPolygon and returns an ST_Point that is the center of an ST_MultiPolygon's envelope.

The ST_PointOnSurface function takes an ST_MultiPolygon and returns an ST_Point that is guaranteed to be normal to the surface of one of its ST_Polygon elements.