Working with the Geodatabase Using SQL

An ESRI ® Technical Paper • February 2004

This technical paper is aimed primarily at GIS managers and data administrators who are responsible for the installation, design, and day-to-day management of a geodatabase.

ESRI 380 New York St., Redlands, CA 92373-8100, USA • TEL 909-793-2853 • FAX 909-793-5953 • E-MAIL info@esri.com • WEB www.esri.com

Working with the Geodatabase Using SQL

An ESRI Technical Paper

Contents

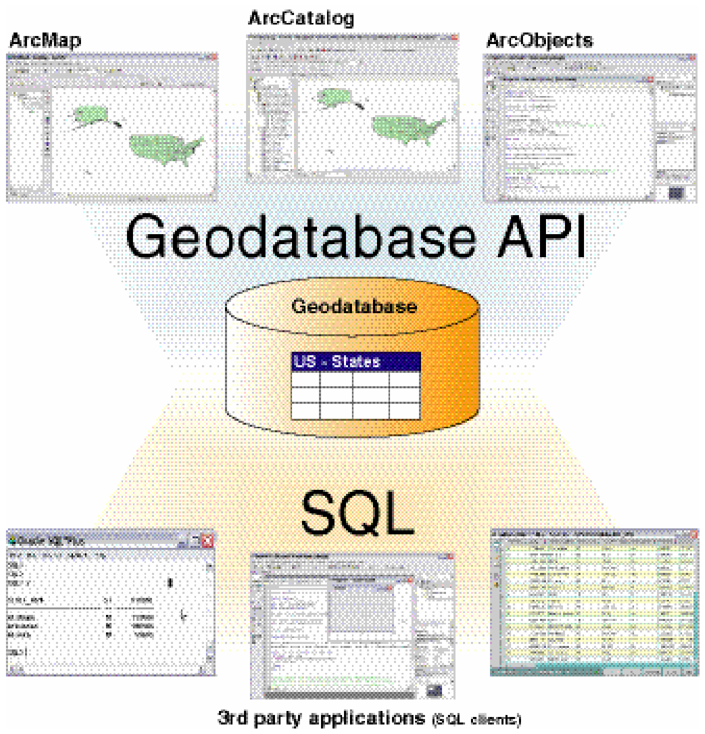
Working with the Geodatabase Using SQL

Feature classes and tables in a geodatabase can be queried and modified using SQL. This technical paper discusses SQL and the geodatabase, how to query and edit a geodatabase using SQL, multiversioned views, and version reconcile.

SQL and the geodatabase

Working with ArcGIS® client applications, end users and ArcObjects™ developers alike can interact with the geodatabase to create and manage spatial data. The geodatabase application programming interface (API) and ArcGIS application user interfaces shield end users from the internal architecture of the geodatabase—often complex geometries and their relationships with adjacent features are viewed and manipulated as intuitive real-world objects rather than the underlying table rows and columns.

This technical paper will focus on an alternative approach to working with a geodatabase that will support those who may have no specific knowledge of the geodatabase, its internal structure, and API, but who are required to work with the information stored in the underlying database management system (DBMS) and managed by the geodatabase and ArcSDE®. As geodatabases are implemented in a relational DBMS using DBMS data types and table formats, the DBMS's own Structured Query Language or SQL, a database language that supports data definition and data manipulation commands, may be used to work with the information stored in the database.

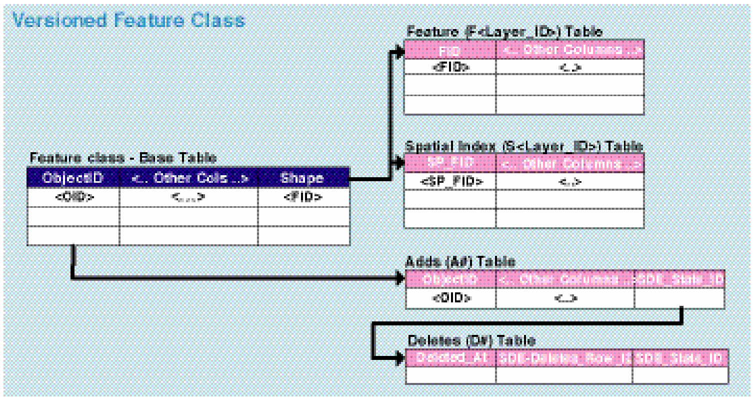


Although ArcSDE does not specifically provide an additional SQL API, there are SQL functions available to users of DBMS platforms that support spatial data types such as Oracle® Spatial, IBM® DB2® Spatial Extender, and Informix® Spatial DataBlade®.

Accessing the information in a geodatabase via SQL extends support for:

- Customizations to the ArcGIS application framework, written in a non-ArcObjects environment, such as Microsoft® ActiveX® Data Objects (ADO), that will work with data managed by the underlying database.
- Third party applications such as nongeographic database applications that also require access to the tabular data managed by the geodatabase but have no requirement to display the data or perform any spatial analysis.
- Applications developed with ESRI® products, such as ArcView® 3.x and earlier releases of MapObjects®, which do not support the geodatabase API but require access to the information in a geodatabase.

Understanding how geodatabase objects are stored in the DBMS and how the SQL works with these objects as DBMS tables is an important prerequisite to working with the geodatabase using SQL. The following section will briefly review the internal storage structure of a versioned feature class in a geodatabase that supports the ArcSDE Compressed Binary format implementation.



The base table is an existing DBMS table with a shape column, added by ArcSDE, for spatial data. The feature ID in this column provides the link between the base table with the associated ArcSDE-managed feature and spatial index tables.

The feature table stores the geometries for each feature, and the spatial index table stores references to shapes based on a simple, regular grid. The spatial index table contains an entry for each shape and grid cell combination to support spatial queries. The Adds and Deletes tables record the changes made to versioned tables in an enterprise geodatabase.

Querying and editing a geodatabase using SQL

Querying a geodatabase using SQL

The SQL 92 standard is supported by both personal (Access) and enterprise (ArcSDE) geodatabases, although each geodatabase implementation supports a different version of SQL—ArcSDE geodatabases support ANSI SQL and personal geodatabases support JET SQL. Although these two versions of SQL are very similar, there are slight differences as to what each data source will support with respect to SQL syntax. Some of the differences to be aware of are:

- In both personal and ArcSDE geodatabases, the where clause is not case sensitive with regard to field names and keywords.

  “STATE_NAME LIKE …” is the same as “state_name like …”.

  However, although the Access database is not case sensitive with regard to the actual field value that is being queried, DBMS platforms that support ArcSDE are case sensitive.

  For example, querying a feature class of U.S. states with the following SQL statement:

  "Select * from US_States where State_name = 'Florida' "

will return one record in both ArcSDE and personal geodatabases (assuming the US_states feature class contains a feature for Florida). However,

"Select * from US_States where State_name = 'florida' "

will return one record in a personal geodatabase but no records in an ArcSDE geodatabase.

- The Like predicate is again slightly different for personal and ArcSDE geodatabases. In a personal geodatabase, an asterisk symbol (*) is used to match any number of characters, and a question mark (?) is used to match any single character. For example,

  "Select * from US_States where State_name like 'M*' "

  "Select * from US_States where State_name like 'Ma?ne' "

  will return all States that begin with 'M' in a personal geodatabase.
  With ArcSDE, the percent (%) and (_) symbols are used. The equivalent ArcSDE geodatabase syntax for a WhereClause would be:

  "Select * from US_States where State_name like 'M%' "

  "Select * from US_States where State_name like 'Ma_ne' "

The following two examples illustrate how to use SQL to query and modify information maintained in a geodatabase.

Attribute queries

The following Visual Basic® code uses ADO and an Oracle OLE DB provider to connect to an Oracle database. As the Oracle database also supports an enterprise geodatabase, it hosts a mixture of spatial and nonspatial data. The Visual Basic code could be used by non-ArcGIS clients who require access to some of the unversioned spatial information in the geodatabase—to perform attribute-only queries, attribute updates, report generation, and so on.

For further information on working with ADO and OLE DB providers, refer to http://www.microsoft.com/data.

```
' Open the connection to the geodatabase
Dim ADO_CON As ADODB.Connection
Dim ADO_RS As ADODB.Recordset

Set ADO_CON = New ADODB.Connection
Set ADO_RS = New ADODB.Recordset

ADO_CON.Open "Provider=OraOLEDB.Oracle.1;User
ID=<MyUserName>;Password=<MyPassword>;Data
Source=<MyDataSource>"

' Open a recordset from a feature class - include a
' Where clause to identify the required records

ADO_RS.Open "select * from US_States
where name like 'W%'" ADO_CON,
adOpenForwardOnly, adLockOptimistic

' Modify some attributes
Do Until ADO_RS.EOF
 ADO_RS.Fields.Item(5).Value = "No"
 ADO_RS.Update
 ADO_RS.MoveNext
Loop
```

Spatial queries

For DBMSs capable of storing and managing spatial data in spatial geometry data types (Informix, IBM DB2, and Oracle), additional SQL functions are provided to access these data types and work with the spatial data they contain. These DBMS platforms support spatial queries and analysis via SQL, which extends the capabilities of non-ArcGIS DBMS applications that have no specific requirement to display the results but still need to work with the spatial data.

In the following example, based on the DB2 Spatial Extender SQL implementation, a SQL select statement is used to perform a spatial join between two feature classes (tables).

The city engineer needs a list of all buildings that are within one foot of any lot line. The building_id column of the BUILDINGFOOTPRINTS table uniquely identifies each building. The lot_id column in turn identifies the lot to which each building belongs. The column MULTIPOLYGON stores the geometry of each building’s footprint.

```
create table BUILDINGFOOTPRINTS
    (building_id integer,
     lot_id      integer,
     footprint   multipolygon);
```

The LOTS table contains a lot_id column that uniquely identifies each lot and a lot multipolygon column that contains the lot line geometry.

```
create table LOTS
     (lot_id  integer,
      lot   multipolygon);
```

The query below identifies the buildings that are within one foot of their lot lines—the distance function performs a spatial join between the building footprints and the boundary of the lot multipolygons and the equijoin between the BF.lot_id and LOTS.lot_id columns ensures that only the multipolygons belonging to the same lot are compared by the distance function.

```
select BF.building_id from
   BUILDINGFOOTPRINTS BF, LOTS
   where BF.lot_id = LOTS.lot_id AND
    distance(BF.footprint,boundary(LOTS.lot)) <= 1.0;
```

Editing a geodatabase using SQL

As well as querying the geodatabase, it is also possible to edit the data in a geodatabase using SQL. However, before doing so it is important to understand the geodatabase data model that has been implemented—are there any advanced geodatabase objects present, such as geometric networks or topologies, and if yes which feature classes are involved? When working with a geodatabase using ArcGIS applications, the behavior and validation rules that the geodatabase enforces will automatically manage the relationships between those feature classes. This implicit geodatabase behavior does not occur if SQL is used to directly modify the data.

For example, when a composite relationship is established between two objects, as in the case of feature-linked annotation, any modifications made to those objects using ArcGIS applications and the geodatabase API will ensure the relationship between the features and the annotation is always maintained. This relationship, and the integrity of the data, is guaranteed by the following behavior:

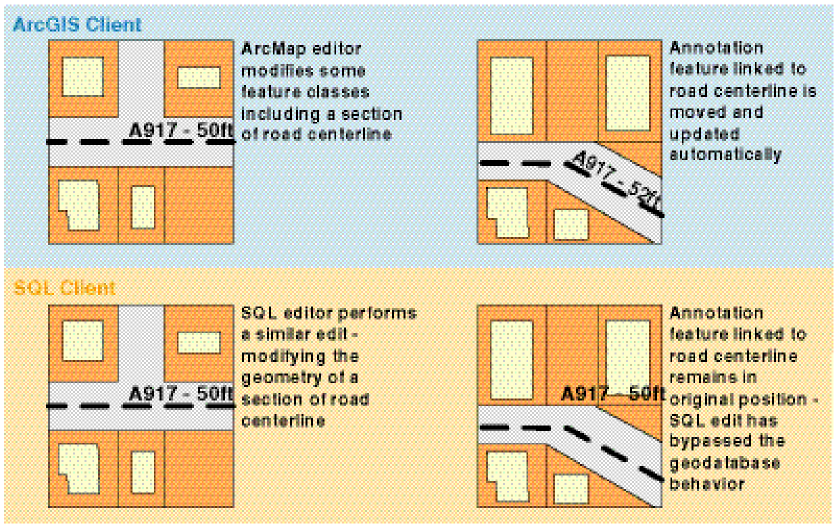
When a new feature is created, a new annotation element is automatically created.

- If a feature is moved, the annotation for that feature automatically moves with it.
- If a feature attribute on which the annotation text is based is changed, the annotation text changes automatically.
- If a feature is deleted, the annotation is also deleted, again automatically.

However, for performance reasons there may be occasions when modifying the data in a geodatabase is best done using SQL, instead of the geodatabase API, as in the case of bulk attribute updates or data loading operations. If the target geodatabase table attributes play no part in geodatabase behavioral constraints or data validation rules, then updating geodatabase tables using SQL can be an efficient approach to modifying the data. However, as any edit operation undertaken using SQL will bypass these geodatabase constraints and data integrity rules and will not respect the relationships between feature classes, the potential impact of such an operation should be considered carefully prior to making any changes. It is important to ensure that modifying certain table attributes will not affect other objects in the geodatabase.

As an example, if SQL was used to modify a feature or row attribute from which text was derived for feature-linked annotation, the related annotation would not be alerted by internal object messaging to update automatically—the annotation would continue to reflect the pre-update text value resulting in the annotation and feature becoming out of sync. Similar behavior would result if the geometry of a feature was modified via SQL—

this would bypass internal geodatabase triggers to automatically update related annotation or topologically related features, leaving objects such as networks and topologies in an inconsistent and possibly corrupt state.


ArcGIS Client
A917 - 50ft
ArcMap editor modifies some feature classes including a section of road centerline
A917 - 52ft
Annotation feature linked to road centerline is moved and updated automatically
SQL Client
A917 - 50ft
SQL editor performs a similar edit - modifying the geometry of a section of road centerline
A917 - 50ft
Annotation feature linked to road centerline remains in original position - SQL edit has bypassed the geodatabase behavior

For these reasons, it is recommended that the use of SQL to modify geometries be confined to modifying simple geodatabase objects, such as standalone feature classes. This type of operation is only supported for Oracle Spatial and the SQL implementations supported on the DB2 and Informix DBMS platforms. It is not supported against the ArcSDE binary schema implementation.

The following guidelines apply to the use of SQL for modifying information in a geodatabase:

- Avoid updating records using SQL after the data has been versioned, unless SQL is used in conjunction with an ArcSDE multiversioned view. Multiversioned views will be discussed in greater detail later in this paper. This applies to ArcSDE geodatabases only—personal geodatabases do not support versioning.
- Always remember to issue a commit or rollback statement after the SQL statement has been executed in the database to ensure the changes are either made permanent or undone as required. Failing to do so can cause problems for the DBMS. For example, a database compress operation would wait for uncommitted DBMS DML statements to be committed before executing successfully.
- When updating unversioned data using SQL, avoid modifying any attributes that, through geodatabase behavior, affect other objects in the database—as in the case of feature-linked annotation and relationship classes.

- Avoid using SQL to modify the geometries of feature class that participate in any advanced geodatabase objects or functionality such as geometric networks, topologies, and relationships. Modifications to feature classes participating in networks, topologies, or relationship classes should be restricted to ArcGIS desktop applications and the geodatabase API. Third party SQL-based applications can read the data, but any modifications they make to the data will not be reflected in the network, topology, or relationship.
  In particular, never update the geodatabase specific *Enabled* or *AncillaryRole* field for a network feature class using SQL. When this field is updated through ArcGIS desktop applications and the geodatabase API, any updates automatically trigger changes to the internal geometric network topology tables. Editing these fields via SQL will bypass this behavior and leave the network in an inconsistent state.
- Never update the Row_ID (ObjectID) field with SQL—this field is allocated and managed by the geodatabase and should not be altered.

Versioned tables and multiversion views

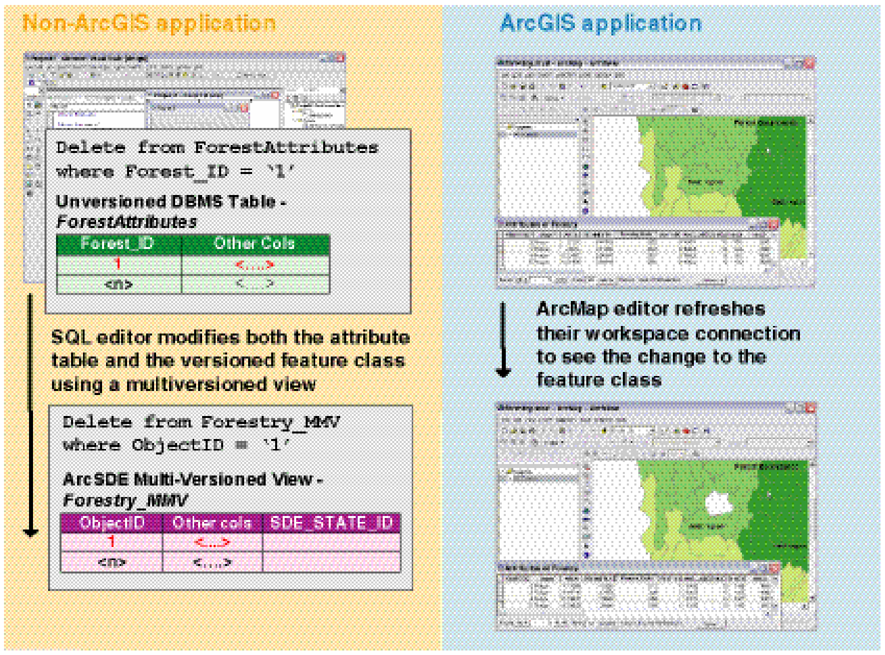
Querying a geodatabase using multiversioned views

Simultaneous multiuser read–write access to the geodatabase is supported through versioning, with the changes made to versioned datasets being recorded in delta tables. Client applications that do not support versioning and the geodatabase have no specific knowledge of these delta tables. Without an appreciation of the internal architecture of a versioned geodatabase or a mechanism to interrogate these delta tables, only the information in the base table can be queried.

To support client applications that must use SQL, access to spatial and nonspatial versioned tables in a geodatabase is provided with ArcSDE through multiversioned views—a combination of DBMS views, stored procedures, and triggers on the underlying versioned table schema (the business, feature, spatial index, and delta tables). A multiversioned view presents a view of a versioned table for a particular version in the geodatabase—all the records from the base table are selected and unioned with records from the delta tables that have been modified in the current version state lineage. Although the internal implementations of these multiversioned views vary across database platforms, the same general data access model is provided for Oracle, Microsoft's SQL Server™, IBM DB2, and Informix databases.

Multiversioned views provide a solution to some of the interoperability issues that arise when supporting multivendor or legacy systems. In land use management, for example, it is not untypical for one department to manage and maintain the attribute data in a central DBMS, while another department, tasked with the collection and maintenance of the associated spatial data, implements a versioned geodatabase in the same DBMS. The two departments rely on the native APIs provided by the DBMS and ArcGIS to manage their respective data holdings, with DBMS forms-based applications developed to query and update the attribute data and ArcGIS display/edit/print applications to work with the spatial data.

To preserve the consistency and currency of the data, there will inevitably be times when changes made to one data source must be reflected in the other—for example, if a row is deleted in the attribute table, the corresponding spatial record must also be deleted. This can be achieved using a multiversioned view—a SQL editor can make the necessary change to the versioned feature class via a multiversioned view. When the ArcGIS application refreshes their view of the version, the feature class will reflect the recent change made by the non-ArcGIS application using the multiversioned view.



```
Delete from ForestAttributes
where Forest_ID = '1'
```

Unversioned DBMS Table - ***ForestAttributes***

Forest_ID	Other Cols
1	<....>
<n>	<....>

SQL editor modifies both the attribute table and the versioned feature class using a multiversioned view

```
Delete from Forestry_MMV
where ObjectID = '1'
```

ArcSDE Multi-Versioned View - ***Forestry_MMV***

ObjectID	Other cols	SDE_STATE_ID
1	<...>	
<n>	<....>	

ArcMap editor refreshes their workspace connection to see the change to the feature class

These multiversioned views are created in one of two ways:

1. Using the ArcSDE SDETABLE administration command:

```
sdetable -o create_mv_view -T
world_mv_view -t world -u av -p mo
```

This creates a multiversioned view 'world_mv_view' from the feature class 'world'.

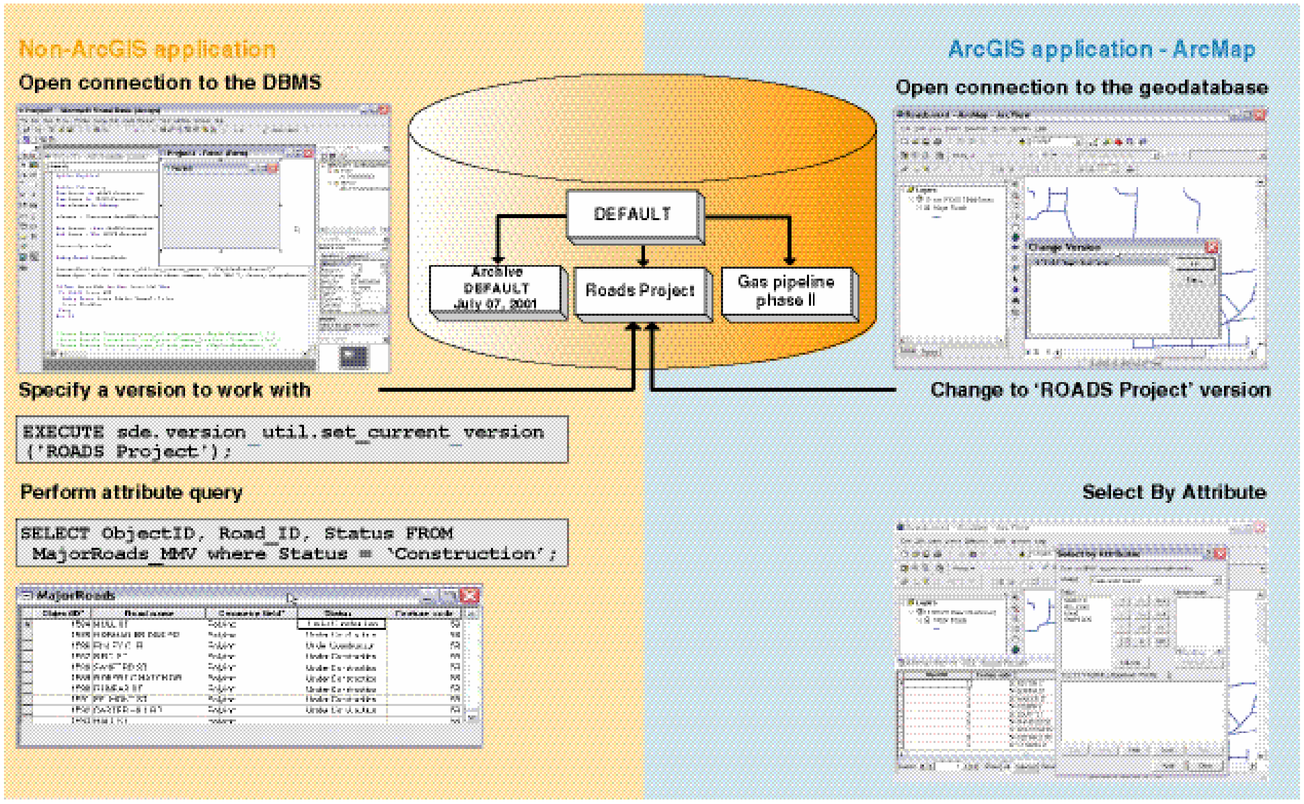
2. Through the SDE C or Java API.

   For further information on using the SDETABLE command and the C and Java APIs and sample code for working with multiversioned views, refer to the ArcSDE Developer Help, http://arcsdeonline.esri.com/.

Once the multiversioned view has been created, the client application will work with the view as it would work with any other DBMS view. When an application connects to the DBMS, access to specific versions is supported by stored procedures created when ArcSDE is installed in the DBMS. The required version is set by the calling application using the ArcSDE stored procedure, *set_current_version ('*<required version>*').* For the complete command syntax for each supported DBMS platform database, refer to current ArcSDE documentation at http://arcsdeonline.esri.com.

This procedure validates the supplied version name and sets the corresponding database state internally; if no version name is supplied, the DEFAULT version is used. This procedure may also be called again to change to other versions as required, and it is called each time the workspace is refreshed to return the current state of the versioned table to the calling application.

With the connection to the required version established via the multiversioned view, SQL queries can be executed against the database. The view guarantees that all the rows, from the base and delta tables, that represent a feature or features in a particular version of the geodatabase will be returned transparently to the end user.


Non-ArcGIS application
Open connection to the DBMS
ArcGIS application - ArcMap
Open connection to the geodatabase
DEFAULT
Archive DEFAULT July 07, 2001
Roads Project
Gas pipeline phase II
Specify a version to work with
Change to 'ROADS Project' version
Perform attribute query
Select By Attribute
MajorRoads

```
EXECUTE sde.version_util.set_current_version
('ROADS Project');
```

```
SELECT ObjectID, Road_ID, Status FROM
 MajorRoads_MMV where Status = 'Construction';
```

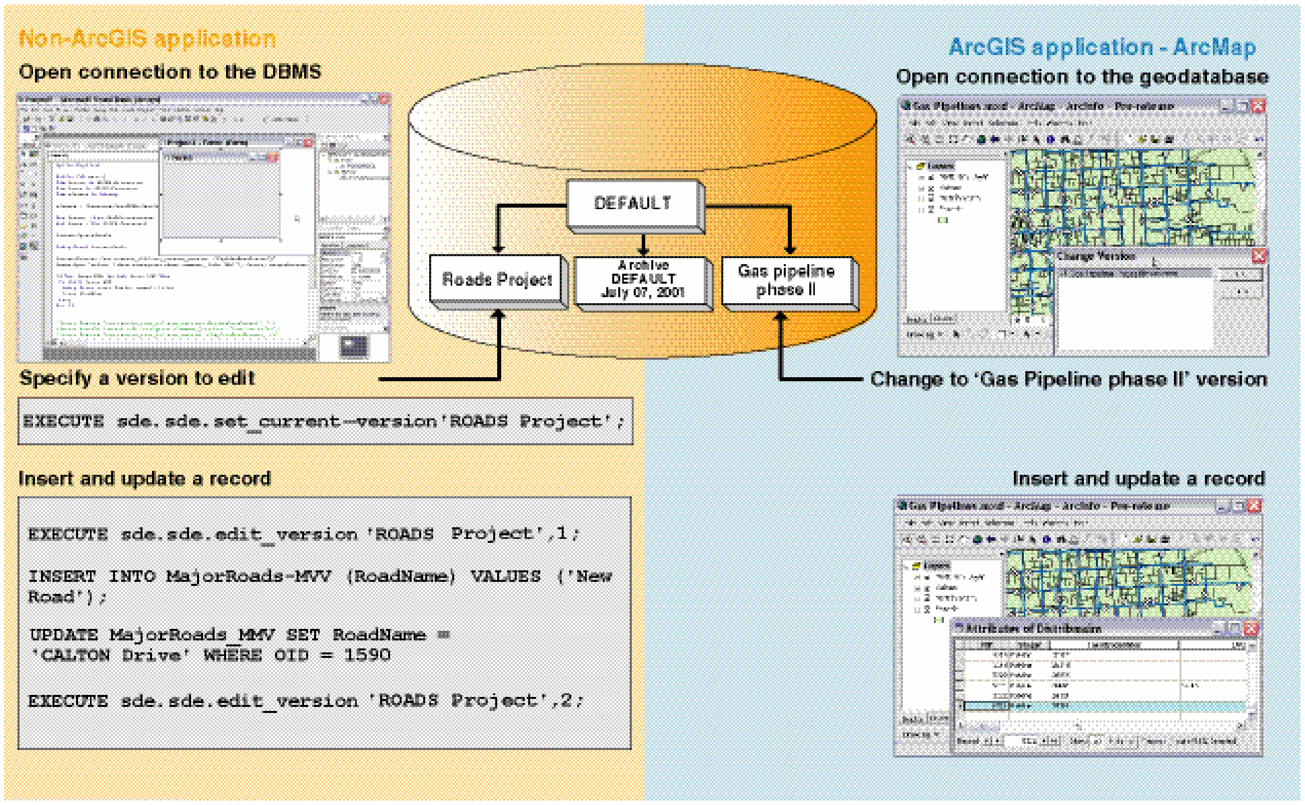
Editing a geodatabase using multiversioned views

Editing versioned feature classes or tables is also supported using multiversioned views—the multiversioned view ensures that all the changes are logged in the appropriate delta tables, Row_Ids for any new records are assigned automatically, and associated indexes are updated.

Note: Read–Write multiversioned views are supported on ArcSDE for Oracle, SQL Server, and DB2 platforms. ArcSDE for Informix supports read-only multiversioned views.

Before editing can occur, the target version is again identified with the *set_current_version* procedure and opened for editing with another SQL stored procedure, *edit_version(*'<required version>*'*,<edit action>*)*. The state currently referenced by the version is opened and closed for editing by setting the appropriate edit action parameter—1 = open, 2 = closed.

In the illustration below, a multiversioned view is used by a non-ArcGIS application to add a new record and update an existing record in a versioned feature class. The ArcSDE/DBMS platform in this case is Microsoft SQL Server. An equivalent operation in ArcMap is included for comparison.

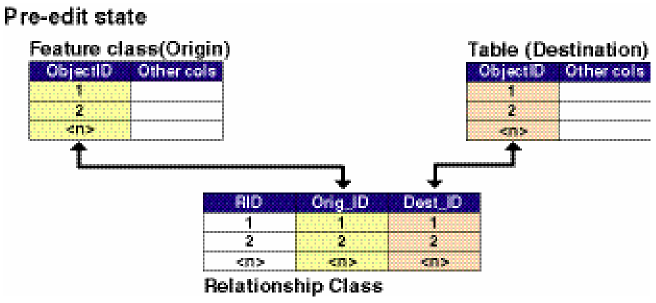

Non-ArcGIS application
Open connection to the DBMS
DEFAULT
Roads Project
Archive DEFAULT July 07, 2001
Gas pipeline phase II
ArcGIS application - ArcMap
Open connection to the geodatabase
Specify a version to edit
Change to 'Gas Pipeline phase II' version
Insert and update a record
Insert and update a record

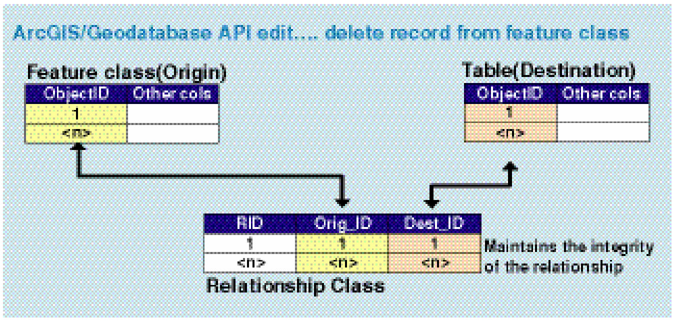
```
EXECUTE sde.sde.set_current-version 'ROADS Project';
```

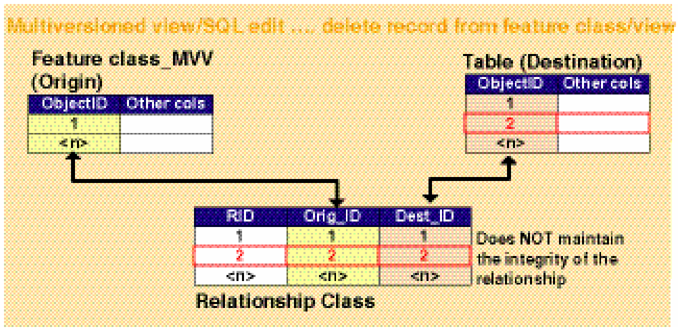
```
EXECUTE sde.sde.edit_version 'ROADS  Project',1;

INSERT INTO MajorRoads-MVV (RoadName) VALUES ('New
Road');

UPDATE MajorRoads_MMV SET RoadName =
'CALTON Drive' WHERE OID = 1590

EXECUTE sde.sde.edit_version 'ROADS Project',2;
```

As with editing unversioned geodatabase tables using SQL, editing versioned tables using multiversioned views does not trigger internal geodatabase behavior for related tables. Care should be taken when updating any attribute fields that participate in a relationship with other geodatabase objects. Multiversioned views should not be used for modifying the geometries of feature classes that participate in networks, topologies, and other objects with specific geodatabase behavior.

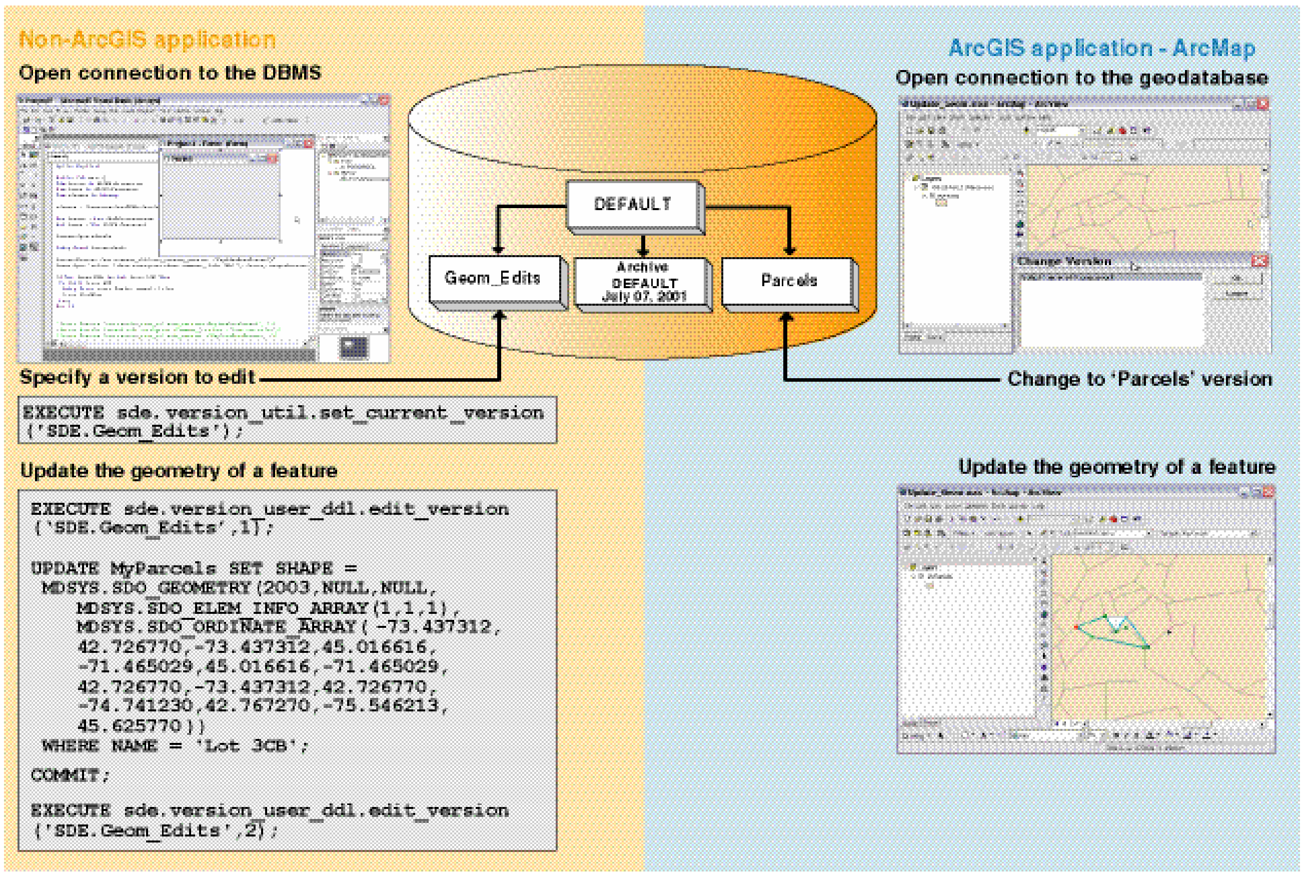
The following example illustrates this behavior with respect to a feature class, an attribute table, and a relationship class that defines the composite relationship between the two. A multiversioned view is created on the feature class, and this view is subsequently edited using SQL. If a feature were deleted via the view, the corresponding row in a relationship table and the related attribute table would not be deleted.






Multiversioned view/SQL edit .... delete record from feature class/view
Feature class_MVV (Origin)
ObjectID
Other cols
1
<n>
Table (Destination)
ObjectID
Other cols
1
2
<n>
RID
Orig_ID
Dest_ID
1 1 1
2 2 2
<n> <n> <n>
Does NOT maintain the integrity of the relationship
Relationship Class

Multiversioned views and SQL could also be used to modify some spatial data. In the next example, one of the geometries of a versioned feature class is modified—in this case, the ArcSDE/DBMS platform is Oracle and the geometry is stored in Oracle Spatial format. Again, an equivalent operation in ArcMap is included for comparison.


Non-ArcGIS application
Open connection to the DBMS
Specify a version to edit
Update the geometry of a feature
DEFAULT
Geom_Edits
Archive DEFAULT July 07, 2001
Parcels
ArcGIS application - ArcMap
Open connection to the geodatabase
Change Version
Change to 'Parcels' version
Update the geometry of a feature

```
EXECUTE sde.version_util.set_current_version
('SDE.Geom_Edits');
```

```
EXECUTE sde.version_user_ddl.edit_version
('SDE.Geom_Edits',1);

UPDATE MyParcels SET SHAPE =
 MDSYS.SDO_GEOMETRY(2003,NULL,NULL,
    MDSYS.SDO_ELEM_INFO_ARRAY(1,1,1),
    MDSYS.SDO_ORDINATE_ARRAY( -73.437312,
    42.726770,-73.437312,45.016616,
    -71.465029,45.016616,-71.465029,
    42.726770,-73.437312,42.726770,
    -74.741230,42.767270,-75.546213,
    45.625770))
 WHERE NAME = 'Lot 3CB';

COMMIT;

EXECUTE sde.version_user_ddl.edit_version
('SDE.Geom_Edits',2);
```

Multiversioned views and version reconcile

Unlike editing versioned data in an ArcMap edit session, although each multiversioned view edit will create a new database state, no internal version reconcile is undertaken. Consequently, it is important to be aware of the possibility of other editors working in ArcMap™, editing the same version at the same time. For this reason, it is strongly recommended that multiversioned views and the ArcSDE stored procedures are *not* used to edit the DEFAULT version, or any other version, that will be subject to other edit and reconcile operations at the same time.

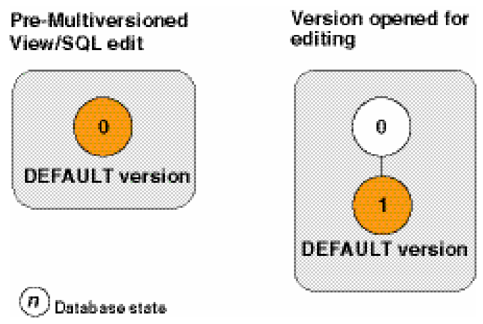
The following examples illustrate some of the problems that could be encountered. In all cases, the target geodatabase is an ArcSDE/SQL Server geodatabase.

Scenario #1

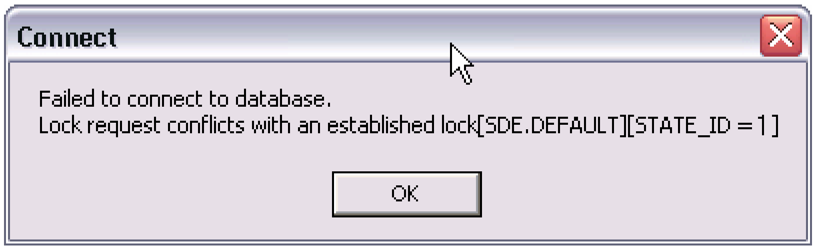
1. An editor working with a multiversioned view and SQL starts editing the DEFAULT version (owned by the SDE user):

```
execute sde.sde.edit_version
('SDE.DEFAULT',1);
```

   This action creates a new child state based on the state currently referenced by the DEFAULT version (state 0). The DEFAULT version is then relabeled to reference the new state.



Any ArcGIS client application currently connected to DEFAULT will now reference state one when they refresh their workspace connection. Any ArcGIS client application that tries to connect to the DEFAULT version now will receive the following message:

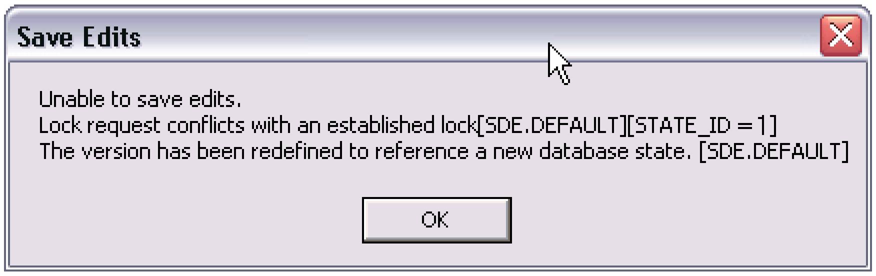


Failed to connect to database.
Lock request conflicts with an established lock[SDE.DEFAULT][STATE_ID = 1]

The multiversioned view edit has acquired an exclusive lock on the state currently referenced by DEFAULT—this prevents other users from connecting to DEFAULT.

2. At this point, an ArcMap editor starts another edit session on the DEFAULT version, which is currently referencing state one. Starting a new edit session does not create a new database state—it is any subsequent edit operations that create the new states. In this respect the data access model supported by multiversioned views differs from the ArcGIS application data access model.

   The ArcMap editor attempts to add a new feature but when completing the edit operation, the following message is displayed:

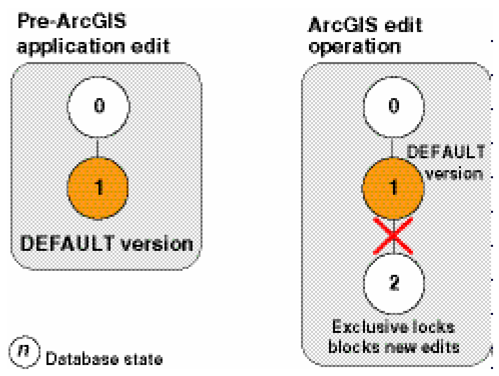


Unable to save edits.
Lock request conflicts with an established lock[SDE.DEFAULT][STATE_ID = 1]
The version has been redefined to reference a new database state. [SDE.DEFAULT]

To create a new edit state from state one, the application has attempted to acquire a shared state lock on state one—this operation fails because of the existing exclusive state lock. Only when this exclusive state lock has been released, by issuing the following command:

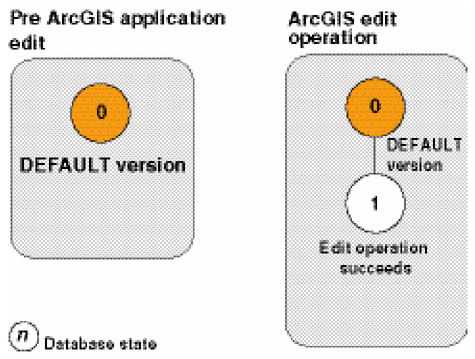
```
execute sde.sde.edit_version
('SDE.DEFAULT',2);
```

will the ArcMap editor be able to continue.



Scenario #2

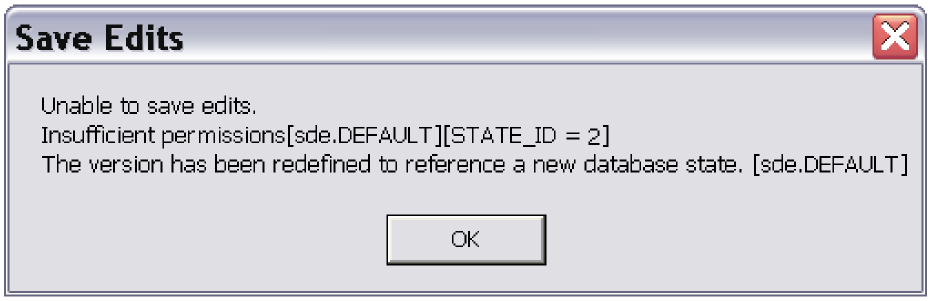
1. An ArcMap editor starts an edit session on the DEFAULT version and adds a new feature. This edit operation successfully creates a new child state from the state currently referenced by the DEFAULT version.



2. The SQL editor opens the same version for editing and adds a new feature but does *not* close the version after making the change.

```
 Execute sde.sde.set_current_version
'SDE.DEFAULT';
 Execute sde.sde.edit_version
'SDE.DEFAULT', 1;
INSERT into sde.gdb.myfeatclass_mvv
(MyField) values ('Some text');
Commit;
```

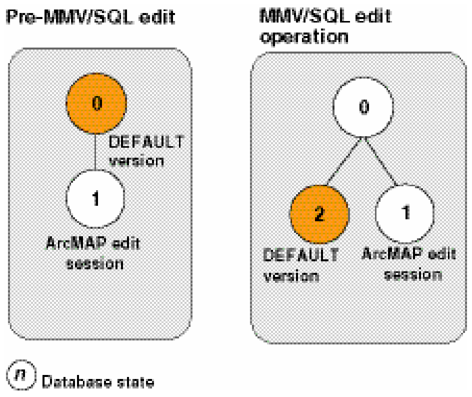
3. When the ArcMap editor then tries to save their changes and reconcile their temporary internal version with the DEFAULT version, the following message is displayed:



Save Edits

Unable to save edits.
Insufficient permissions[sde.DEFAULT][STATE_ID = 2]
The version has been redefined to reference a new database state. [sde.DEFAULT]

The edit session started by the SQL edit has moved the DEFAULT version to a new, and currently open, state—state two.



The ArcMap editor is unable to save their changes and complete the automatic reconcile with the DEFAULT version. Any changes made during the course of their edit session are now lost and must be repeated.

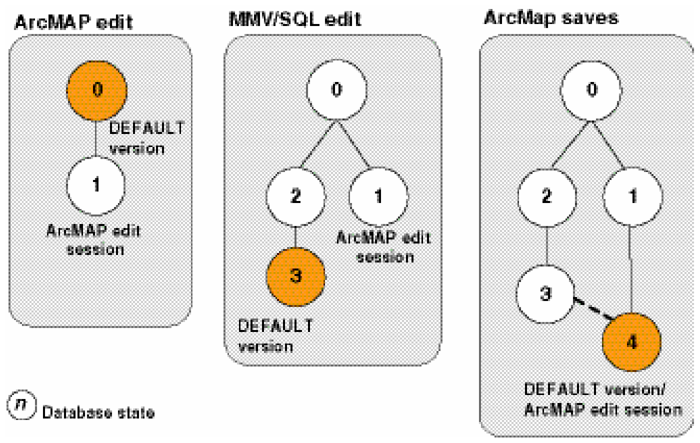
Scenario #3

1. An ArcMap editor starts an edit session on the DEFAULT version and adds a new feature.

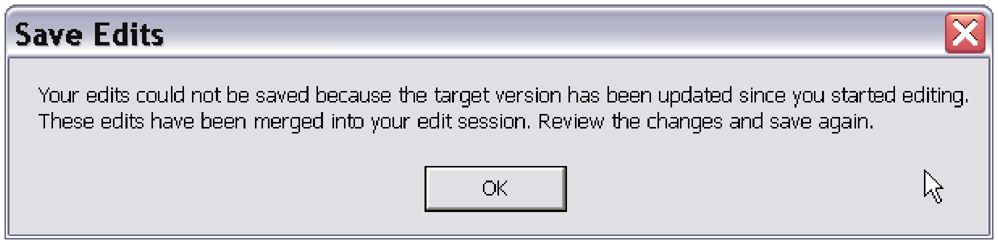
2. The SQL editor opens the same version for editing, adds a new feature, and this time closes the state once all the changes have been made.

```
Execute sde.sde.set_current_version
('SDE.DEFAULT');
Execute sde.sde.edit_version
('SDE.DEFAULT', 1);
INSERT into sde.gdb.MyFeatClass_MVV
(MyField) values ('Some text');
```

```
Commit;
Execute sde.sde.edit_version
('SDE.DEFAULT', 2);
```



3. The ArcMap editor then tries to save their changes and reconcile their temporary version with the DEFAULT version. The following message is displayed:



Save Edits

Your edits could not be saved because the target version has been updated since you started editing. These edits have been merged into your edit session. Review the changes and save again.

The SQL edit has moved the DEFAULT version to a new, and now closed, state. The ArcMap editor is advised that the target of the internal reconcile has changed since they began their edit session, and they must save again to acknowledge the changes. When the editor saves a second time, the changes are successfully applied.

The easiest way to avoid any contention between the different data access models is to create a new version for those editors working with multiversioned views and SQL. If this is not possible and the DEFAULT version has to be modified, other editors also working with the DEFAULT version need to be aware that they will be unable to modify that version until the SQL session has executed the following procedure:

execute sde.sde.edit_version 'SDE.DEFAULT', 2;

This procedure closes the current state referenced by DEFAULT and makes it available again to other editors.

SQL and multiversioned views provide an alternative development option for those wishing to work with the information managed in a geodatabase but have no requirement to work with advanced geodatabase objects, such as networks and topologies, or display the data. Through multiversioned views, non-ArcGIS client applications have access to the same versioned data management framework as ArcGIS applications and can both query and update individual named versions in the geodatabase.