
Using ArcView[®] Database Access

(Version 2.0)

Environmental Systems Research Institute, Inc.

Contents

Chapter 1 Welcome to ArcView Database Access 5

- Using the Database Access extension 6
- Tips on learning the Database Access extension 8
- Before you start the ‘Quick start tutorial’ exercises 8
- Getting on-line help 11
- Getting technical support from ESRI 12
- Visit ESRI on the Web 12

Chapter 2 Quick start tutorial for SDE 13

- Exercise 1: Make a map of U.S. states, cities, and counties 14
- Exercise 2: Find counties with many single family homes 22
- Exercise 3: Find locations for an advertising campaign 32

Chapter 3 Quick start tutorial for ODBC 41

- Exercise 1: Identifying potential market cities in the United States 42
- Exercise 2: Narrowing your search for new market cities 52

Chapter 4 Understanding the Database Access objects 61

- The common Database Access objects 62
- The SDE interface objects 64
- The ODBC interface objects 66

Chapter 5 Using Avenue to manipulate SDE objects 69

- Adding a database theme to a view 70
- Stepping through a database theme’s records 72
- Selecting features in a database theme 73
- Working with selected features 75
- Creating and working with database tables 76
- Working with selected records in Database Tables 80
- Modifying the source data 80
- Using locks and transactions 85
- Checking for errors 86
- Creating new tables and spatial columns 88

Chapter 6 Using Avenue to manipulate ODBC objects 91

Creating and working with database tables 92

Stepping through a database table's records 93

Modifying the source data 95

Using locks and transactions 100

Checking for errors 101

Working with selected records 104

Appendix A Setting up your computer 105

Setting up your computer for SDE 106

Set up your computer for ODBC 108

Appendix B Creating new tables in SDE 111

Creating new tables in the database 112

Index 115

CHAPTER 1

Welcome to ArcView Database Access

ArcView® Database Access extension lets you bring data from your relational database management system into ArcView GIS, where you can use it to solve spatial problems. This book will help you understand how you can use ArcView GIS software's main components, together with some new ones designed specifically to work with relational databases, in a way that best suits your database environment.

You don't need to be experienced with ArcView or relational databases to start using the Database Access extension. The 'Quick start tutorial for SDE' and the 'Quick start tutorial for ODBC' chapters cover the ArcView basics and help you get acquainted with relational database concepts. Even if you're an experienced user, you will find the 'Quick start' chapter essential to adapting to the Database Access environment.


The database access extension supports connections to SDE 3 and ArcSDE 8 servers. Information and instructions pertaining to SDE throughout this book can also be applied to ArcSDE 8 servers. Also, ODBC is only supported on PC platforms.

In this chapter, you will:

- Get an introduction to Database Access.
- Find out what you need to get started.

Using the Database Access extension

Your database contains many tables of data whose subjects are related to each other. You could have a table with property information, and several lookup tables describing codes in the property information table. Database Access lets you see the contents of these tables using an ArcView database table.



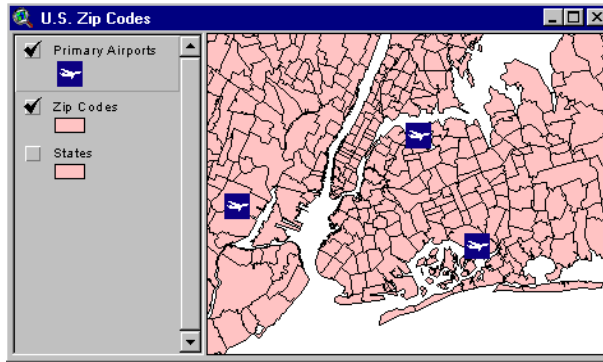
LAND PARCEL	STREET	CITY STATE	ZIP CODE
3652 SHERIDAN DR	PO BOX 290	DALLAS TX	75221-0290
3676 SHERIDAN DR	3676 SHERIDAN DR	AMHERST NY	14226-1701
3700 SHERIDAN DR	49 BEDFORD AVE	BUFFALO NY	14216-3508
3675 SHERIDAN DR	223 VISCOUNT DR	WILLIAMSVILLE NY	14221-1771
3689 SHERIDAN DR	84 ASPINWOOD PL	BUFFALO NY	14223-1516
3671 SHERIDAN DR	P O BOX 25025	GLENDALE CA	91201-5025
3651 SHERIDAN DR	3651 SHERIDAN DR	BUFFALO NY	14226-1702
3720 SHERIDAN DR	3720 SHERIDAN DR	BUFFALO NY	14226-1732
3750 SHERIDAN DR	3750 SHERIDAN DR	AMHERST NY	14226-1732
3775 SHERIDAN DR	3900 SHERIDAN DR	EGGERTSVILLE NY	14226-1730

Database tables are created using Structured Query Language (SQL) Select statements, which choose the columns and records of data you want to see from the tables in your database. You don't need to know SQL to see your data in ArcView. The Database Access extension provides a wizard that will walk you through the steps of choosing the data you want to see. When you're comfortable writing your own SQL Select statements, you'll be able to combine columns of data, summarize groups of records, and sort the records you get by more than one column.

Once you have a database table in your project, you can join it to other ArcView tables such as a shapefile's attribute table. If you're not satisfied with a database table's contents, you can change its query.

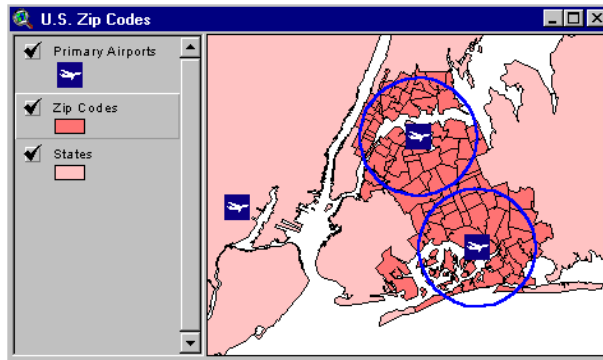
With software like Spatial Database Engine™ (SDE™) or ArcSDE 8 from Environmental Systems Research Institute, Inc. (ESRI), you can create a spatial database that has land parcel shapes in your property information table. Database Access lets you add the land parcels in your database to an ArcView map using a database theme.

Database themes are created using queries that can retrieve shapes from the database. You can build those queries using the wizard. Walking through the wizard, you could, for example, join the property information table and the owners table to a table of zip code shapes, then create a zip code theme containing data from all three tables.



When a database theme is part of a map, you can change its projection, classify land parcels by their attributes, and locate specific land parcels.

Using Avenue™ software, ArcView's programming language, you can create themes that use spatial relationships to choose which shapes appear in a theme. The example below demonstrates Avenue's ability to analyze a theme's contents, showing a five mile buffer that's been created around La Guardia and John F. Kennedy International airports. The Zip Codes theme below now includes only the zip codes intersecting the buffers.



If you wanted to notify the residents within the five mile buffer zones that changes to flight times could increase noise levels, you could generate a list of addresses with the property information contained in the Zip Codes theme.

Though you can't directly edit the contents of your database in ArcView using database themes and database tables, you can edit the database using Avenue scripts.

Relational databases let you put all of your data in one place, where it can be accessed by many people. The Database Access extension lets many people view, query, analyze, and edit that data at the same time without conflict.

Tips on learning the Database Access extension

To become familiar with the Database Access environment, we highly recommend you work through the exercises in either ‘Quick start tutorial for SDE’, Chapter 2, or ‘Quick start tutorial for ODBC’, Chapter 3. These chapters provide exercises that will quickly get you working with your SDE or ODBC data using ArcView’s interface and will guide you through the basics of ArcView functionality. If you want more information about how to use ArcView, see the *Using ArcView GIS* book.

Once you’ve learned the basics of the Database Access extension and have started using it, you can start accessing the on-line help system. The on-line help system contains reference and “how to” information. This is an excellent way to get your questions answered while you are using the Database Access extension.

If you want to find out how you can do more with your data, start by learning Avenue and then working through the other chapters in this book. You can start learning Avenue with the *Using Avenue* book and by looking at the many sample scripts provided with ArcView. To find descriptions of the samples, see the Sample scripts and extensions on-line help topic.

Before you start the ‘Quick start tutorial’ exercises

Before you can begin the ‘Quick start’ exercises in Chapters 2 or 3, you need to find out some information and make sure your machine is set up correctly.

If you’re using SDE...

Find out who the SDE database administrator is in your organization, then arrange some time to ask the following questions.

- What is the name of the SDE server I’m connecting to?

The SDE server is the machine that has your data.

- What is the name of the SDE database instance I’m connecting to?

The SDE database instance is the SDE process that lets you access a set of spatial data.

- What user name and password do I use to access the data?
- Is my machine already set up to connect to SDE?

If your machine isn't already set up and you can do the work yourself, follow the instructions located in Appendix A. You need to find out two things: the service number of the SDE database instance, and the IP address of the SDE server.

- Which tables in the database contain the data I want to access?

If you aren't familiar with the contents of the tables, ask for a data dictionary. It is important to know which tables contain what data, how those tables can be joined together, and what the column names in the tables mean.

To follow the exercises in the 'Quick start' chapter, you need to access five tables containing state, city, and county spatial data for the United States, and demographic tables for U.S. cities and counties. Make sure the database administrator has loaded them into an SDE database; they're located in the avtutor\dbaccess directory. The fields joining the spatial tables to the demographic tables should be indexed to improve performance.

- Which database contains the tables I need?

Some relational databases group associated tables together in a database (e.g., SQL Server™ and Sybase®), while others don't (e.g., Oracle® and DB2®). If the relational database you're connecting to contains databases, you need to know the name of the database containing your tables.

If you're using a SQL Server database, make sure the SDE administrator has set up the SDE default Data Source Name on the server with a default database. When you're connecting with an Avenue script, you need to know which data source name to use, and whether its default database contains your tables; if not, you need to change the current database with an Avenue request after connecting.

- Who do I contact if I can't connect to the database?

Sometimes you won't be able to connect to SDE because the SDE server isn't running or because of network problems. Make sure you can contact the people responsible for maintaining the database and the network to find out what is wrong and solve the problem.

Now you have all the information you need to start using SDE data in ArcView.

If you're using ODBC...

The database that you plan to use to complete the ODBC tutorial will determine the amount of preparation you need to make on your machine. If you are using a large relational database like Oracle, you may need to consult your database administrator. If you are using a small local database like dBASE®, you may be able to do all that is needed yourself. In either case, you should read through the following checklist.

- Is my machine already set up to connect to ODBC?

Before you connect to your database, you need to have ODBC, the proper ODBC driver and possibly the native database client software installed. You also need to configure an ODBC data source to connect to your database. For more information on these requirements, refer to the section called “Set up your computer for ODBC” in Appendix A.

- What is the name of the ODBC data source I’m connecting to?

If your machine is set up properly, you will be prompted to select an ODBC data source when you make a connection. You need to find out which data source name to select to access the data that you want to use.

If the ODBC data source that you need is not listed, your machine is not set up properly. You need to go back and check the data source configuration.

- What user name and password do I use to access the data?

Some databases do not require a user name and password to make a connection. If a user name and password are required, you will be prompted when trying to connect.

- Which database contains the tables I need?

Some relational databases group associated tables together in a database (e.g., SQL Server and Sybase), while others don’t (e.g., Oracle). If the relational database you’re connecting to contains databases, you may need to specify a database when connecting. Most drivers, however, allow you to specify a default database when configuring the data source. If this is the case, make sure that the default is set to the correct database.

If you need to connect to more than one database on your server, you need to set up a separate data source for each database. For more information on setting up data sources see Appendix A.

- Which tables in the database contain the data I want to access?

To follow the exercises in the Quick start tutorial for ODBC, you need to access five data sets which are provided with the extension. The data includes shape files with state, city, and county data for the United States, and demographic tables for U.S. cities and counties. The data is in dBASE format and located in the avtutor\dbaccess directory.

You will need to make sure that this data has been loaded into your database. If your database does not provide a way of loading the dBASE files, you can use the “Tools for ODBC sample extension” to load the tables. You can find information about this

extension by searching for “Database Access extension, sample extensions” under the index tab of the on-line help.

- Who do I contact if I can't connect to the database?

Sometimes you won't be able to connect using ODBC because the database server isn't running or because of network problems. Make sure you contact the person responsible for maintaining the network and the database to find out what is wrong and solve the problem.


Now you have all the information you need to start using data from the ODBC database in ArcView.

Getting on-line help

To find out what a button, tool, or menu choice does

- Move the cursor over it but do not select it. A short description will appear in the ArcView status bar.

To get more help about a button, tool, or menu choice

1. Click the Help button .
2. Click the button, tool, or menu choice you want to get help about.

To get help about a dialog box

- Press the F1 key on your keyboard when the dialog is displayed.

To browse the contents of ArcView's help

1. From the Help menu, choose Help Topics.
2. Click the Index tab.

To search ArcView's help for a particular word

1. From the Help menu, choose Help Topics.
2. Click the Find tab.

Note If you use the Index or Find to locate topics in the on-line help, check the topic to make sure it belongs to the Database Access extension. All help topics for the extension have the name Database Access in the title or the first line of the topic.

Getting technical support from ESRI

Please see the software registration and support card that came with ArcView, or look at the 'Obtaining technical support' section of ArcView software's on-line help.

Visit ESRI on the Web

Find out everything you want to know about ESRI® software, data, and services by visiting ESRI's Web home page at *www.esri.com*.

CHAPTER 2

Quick start tutorial for SDE

The exercises in this chapter begin with the basics of creating a map with SDE data and progress to summarizing a database theme's attributes. They also illustrate how to create database tables and use them in combination with database themes to solve spatial problems.

To follow the exercises, you need access to the SDE database containing state, city, and county spatial data for the United States, and demographic tables for U.S. cities and counties. If you haven't already spoken to your SDE administrator, do so now before you begin. The section "Before you start the 'Quick start tutorial exercises'" in Chapter 1 lists the information you need to know.

Each exercise builds on the previous one. You can work through them at your own pace. When you stop, save your work so you can pick up where you left off.

In this chapter, you'll perform these exercises with the Database Access extension:

- Exercise 1: Make a map of U.S. states, cities, and counties
- Exercise 2: Find counties with many single family homes
- Exercise 3: Find locations for an advertising campaign

Exercise 1: Make a map of U.S. states, cities, and counties

Suppose you work for a national insurance company that wants to encourage home owners to purchase a new insurance package on their houses. Your job is to find out where to focus an advertising campaign. You want to start the campaign off by advertising in urban areas in a few states. You're going to find out where to advertise by evaluating U.S. counties according to their demographic data.

The first step in solving this problem is to bring all your data together in an ArcView map. The map will include State, City, and County themes, and the County theme will include demographic data. This map will serve to illustrate trends in the County demographic data using various analysis techniques in later exercises.

In this exercise, you'll learn how to:

- Load ArcView extension software.
- Create new database themes.
- Identify a feature in a map.
- Change the color of a theme.
- Set the projection of a map.
- Set a theme's display properties.
- Save your work.


Start ArcView. Before you can start assembling a map, you need to load the Database Access extension. Loading the extension makes the required components available for connecting ArcView to an SDE database and working with SDE data.

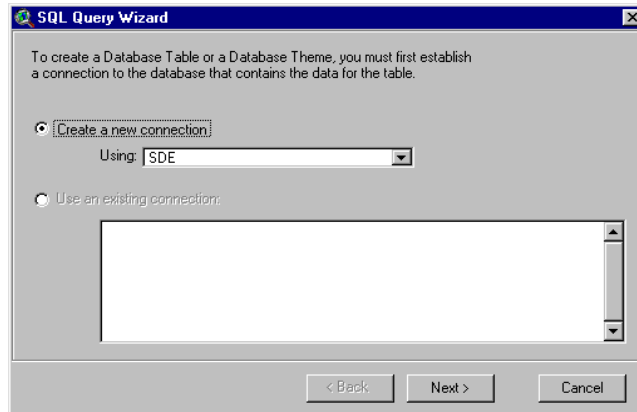
To load ArcView extensions

1. Make the Project window active.
2. From the File menu, choose Extensions.
3. Click the check box next to Database Access.
4. Click OK.

You'll create the map in two stages, adding the state and cities data together, then adding the county data with its demographics. The steps below show how to add simple database themes to your map.

To make a map and add simple database themes

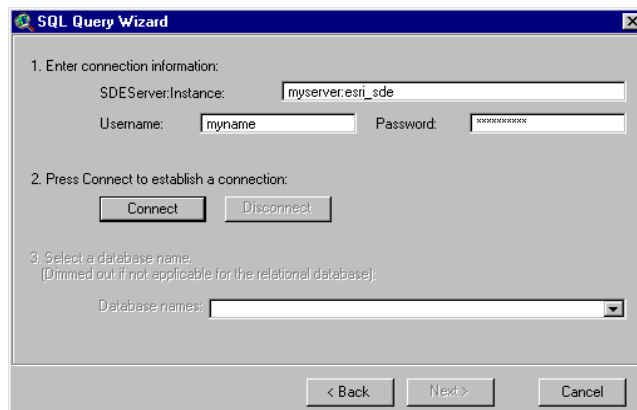
1. In the Project window, double-click on Views  to create a new map.
2. From the View menu, choose Add Database Theme. The SQL Query Wizard dialog appears. It will guide you through the steps of adding database themes to your map.
3. Click Create a new connection, then press Next.



4. First, type the appropriate information into the Server:Instance, Username, and Password text boxes. (If you don't know what to type, read "Before you start the 'Quick start tutorial exercises'" in Chapter 1.) Be careful when you type; SDE is case sensitive with these text strings. Second, click Connect. When the Connect button is dimmed out, you've successfully connected to the SDE server.

Third, if not dimmed out, choose the database containing the U.S. state, city, and county data. Some relational databases group related tables into a database; for them, a list of database names will appear in the drop-down list. You can only access one database of tables at a time with a connection.

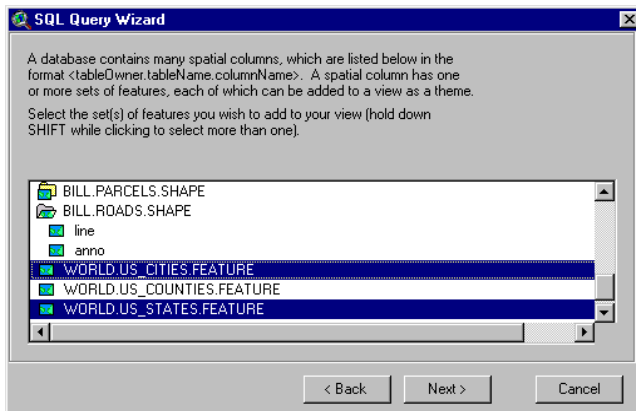
Continue by pressing Next.



- Choose the States and Cities spatial columns from the list (hold down SHIFT and click on them). The list won't show the names "States" and "Cities." The States spatial column will look something like "World.States.Shape". In this example, the user "World" owns the table "States", which has a spatial column called "Shape".

There is no way for this exercise to know what the tables will be called in your database or who will own them. There are no restrictions for naming spatial columns; they might be called "Feature" or "Geometry".

Continue by pressing Next.



- Click Create a theme(s) from the selected set(s) of features, then press Finish. Your map now contains two database themes, which are named for their spatial column.
- Draw the States and Cities themes by clicking the check box next to the theme names in the map's Table of Contents. Make sure that the Cities theme is listed at the top of the Table of Contents so that it draws on top of the States theme. You can change the order by dragging themes up or down in the Table of Contents.



- Change the names of the themes to Cities and States, respectively. First, make the theme active by clicking on its name in the map's Table of Contents, and then click

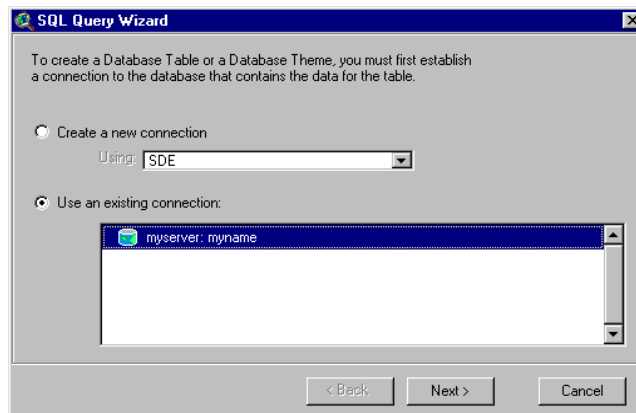
the Theme Properties button . In the Theme Name text box, type the appropriate name. Click OK.

A table in your database contains U.S. county data; it has a spatial column for county shapes and other columns with basic information. Another table contains demographic data for each county. If you created a database theme from the county table using the steps above, it wouldn't have any demographic data. To include demographic data in the theme, you need to join the county and demographic tables together using a column that they have in common (FIPS), whose values are the same in both tables.

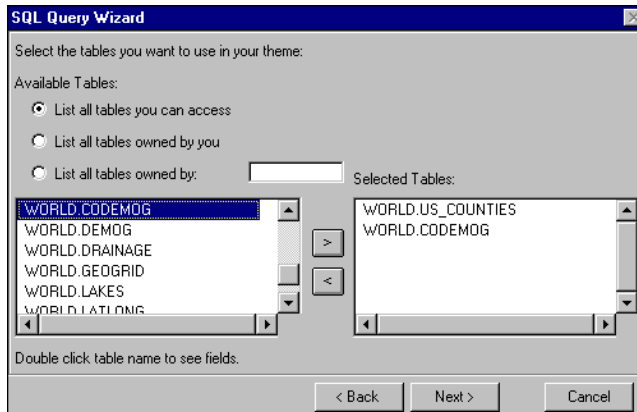
The following steps show you how to create a database theme with joined tables. They refer to the county table as Counties and the demographic table as Codemog.

To add database themes with joined tables

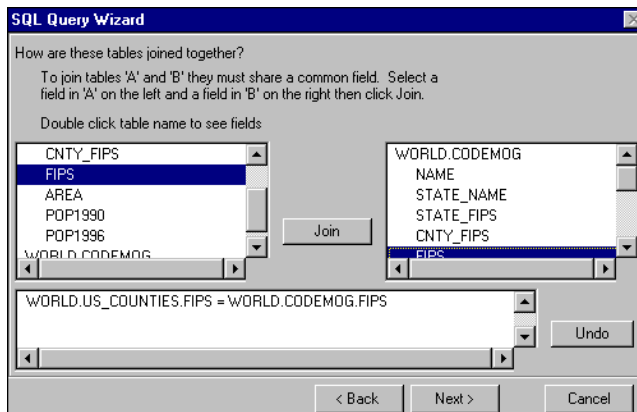
1. From the View menu, choose Add Database Theme. The SQL Query Wizard dialog appears.
2. Click Use an existing connection, then choose the connection used by the States and Cities themes. Press Next.




3. Choose the Counties spatial column from the list, then press Next.
4. Click Create a theme containing a subset of features or with joined table(s), then press Next.
5. Choose the Codemog table from the Available Tables list, and add it to the Selected Tables list by clicking the '>' button. If you can't find the Codemog table in the list, you're connected to the wrong database or you don't have permission to select data from it. Talk to your database administrator before continuing with this exercise. Press Next.



6. Double-click the Counties table in one list and the Codemog table in the other to see a list of the columns in each table. Choose the FIPS column in each list, then click Join. The join statement appears below. Press Next.




7. Add this theme to your map by stepping through the rest of the wizard without making any more selections. Your map now contains another theme, which is named for its spatial column.
8. Draw the Counties theme by clicking the check box next to its name in the map's Table of Contents. Make sure the Cities theme is listed at the top of the Table of Contents so it draws on top of the States and Counties themes.
9. Change the name of the new theme to Counties. Make the theme active, click the Theme Properties button , then type the new name into the Theme Name text box. Click OK.

The only way to see data from a relational database in ArcView is to build a query that selects the data you want. Database themes are themes that display the shapes contained in a spatial column for all the rows that a query retrieves from the database. A database


theme's attributes are values in the nonspatial columns retrieved by the query. A database theme can't include data stored in system files on your computer.

When you added the Counties theme to the map you created a query that joined two tables together. You also created a query when you added the States and Cities themes to the map, but you didn't add anything extra to it. The SQL Query Wizard builds the query for you based on your choices as you step through the dialog.

The queries for the States and Cities themes retrieve all rows and columns in the States and Cities tables, respectively. The query for the Counties theme selects all columns in both the Counties and Codemog tables, and all rows where the FIPS values in the Counties table equal the FIPS values in the Codemog table. For example, if the Codemog table had demographics for the State of New Hampshire only, the theme would only display New Hampshire counties.

You can see the attributes of any feature in a map using the Identify tool . Identifying a feature in the States or Cities theme will display attribute values from all columns in the States and Cities tables, respectively. Identifying a feature in the Counties theme will display attribute values from all columns in both the Counties and Codemog tables.



To identify a feature in a map

1. Make the theme whose features you want to identify active by clicking on its name in the map's Table of Contents.
2. Click the Identify tool .
3. Click on one of the features you want to identify. The feature you click on flashes in the map, and its attributes appear in the Identify Results dialog.

Now that the map contains the required data, start customizing the way it looks. You can change the colors of the themes, the map's projection, and the themes' display properties.

The theme's colors were randomly assigned when the themes were created. To change their colors, use the Legend Editor.

To change the color of a theme

1. Make the States, Cities, and Counties themes active by holding down the SHIFT key while clicking on their names in the map's Table of Contents.
2. Click the Legend Editor button .
3. Choose a theme's name from the Theme drop-down list.
4. Double-click the symbol to display the Symbol Window.
5. In the Symbol window, click the Color button  to display the Color Palette.


6. In the Color Palette, click the color you want to use for the theme.
7. Click Apply in the Legend Editor.
8. Repeat steps 3–7 for the other themes in the Theme drop-down list.
9. Close the Legend Editor and the Symbol Window.

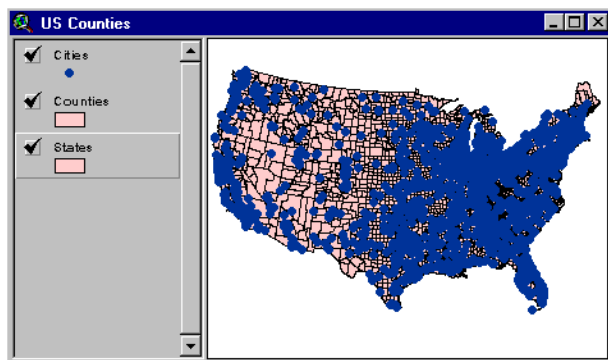
When all the themes in a map have shapes whose coordinates are in decimal degrees, you can project the map. The projection fits a map's shapes to a view of the earth's surface. If you leave the Projection set to None (the default) your map fits the longitude/latitude coordinates to a rectangle instead of the surface of the earth.

Spatial columns might contain shapes which are already projected. You can add themes from these spatial columns to a map, but you won't be able to change their projection with ArcView. All themes in a map must have the same projection to see them at the same time. The United States themes in your map have shapes with decimal degrees coordinates. For this map, use the Albers Equal-Area (Conterminous U.S.) projection.

To set the projection for a map


1. From the View menu, choose Properties.
2. Click the Projection button.
3. From the Category drop-down list, choose Projections of the United States.
4. From the Type drop-down list, choose Albers Equal-Area (Conterminous U.S.).
5. Click OK in the Projection Properties dialog.
6. Click OK in the View Properties dialog.



The Scale field in the tool bar shows the current scale of your map, which changes automatically if you zoom in or out. To display the map at a particular scale, type the scale into the Scale box, then press Return. For this tutorial, zoom into the continental states with the Zoom In tool .



Your map should always display a level of detail appropriate to its scale. When your map displays the entire United States, states are the appropriate level of detail. When you zoom in, counties are more appropriate. You control the scale a theme draws at by setting its display property using the Theme Properties dialog. A theme's display scale is particularly important if it selects a large number of features from the database.

To set a theme's display property

1. Make the Counties theme active.
2. Click the Theme Properties button .
3. In the Theme Properties dialog, click Display. The display property panel appears.
4. Type "25000000" into the Maximum Scale field. Now you'll only see states when your map's scale is larger than 1:25,000,000 (e.g., 1:1,000,000).
5. Click OK.

When your map redraws, the Counties shapes won't appear. With the display property set, your map is now easier to explore. Try using the Zoom In tool  to draw a box around the State of Florida. Then, you can go back to your view of the continental states by clicking the Zoom to Previous Extent button . Change your map's scale to 1:30,000,000 by typing "30000000" into the Scale box then pressing Return. You'll get the right amount of detail for your current scale while you're exploring your map.

In the next exercise, you'll continue working with this project, so you can continue with Exercise 2 now. If you want to exit ArcView now and continue with the next exercise later, remember to save your work.

To save your work

1. From the Window menu, choose Untitled to make the Project window active.
2. From the File menu, choose Save Project.
3. In the dialog that appears, specify a name (e.g., Sdetutorial) and location for the new project file, then click OK. ArcView will automatically add the .apr extension to the name you gave for the new project file.

When you save your project, ArcView doesn't store the theme's values in the project; it stores the query used to get the values from the database. If you close your project now and open it again later, you will be prompted to log in. When you do, ArcView will reestablish its connection to the database and retrieve the shapes and values for the themes in your map.

Exercise 2: Find counties with many single family homes

Now that you have a map you can start to analyze the data in it. In your advertising campaign you want to focus on counties with a large number of single family homes. The Units_1det column in the Codemog table contains the number of single, detached houses for each county, which is the nearest attribute for the number of single family homes. You want to determine the minimum Units_1det value for the counties that will be part of the advertising campaign.

You'll find out about the values in the Units_1det column by looking at them, getting statistics, then classifying the Counties theme. After determining the minimum Units_1det value for the campaign, you will redefine the Counties theme to represent only the counties that fall above that value, the candidates for the advertising campaign.

In this exercise, you'll learn how to:


- See a database theme's attributes in a database table.
- Get statistics for a column of values in a database table.
- Create a new database table.
- Classify a theme's features by their attributes.
- Change a database table's definition query.
- Change the shapes and attributes a database theme represents.

If ArcView isn't already running, start it. Open the project you created in Exercise 1 by choosing Open Project from the File menu. Log in to the database connections stored in the project by typing your user name and password into the Login dialog.

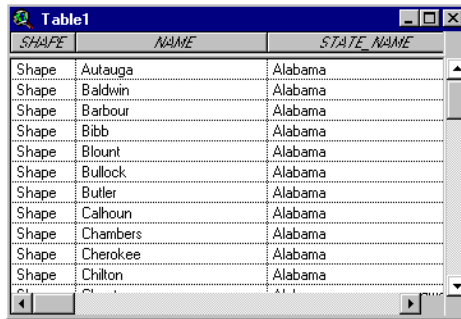
A database theme's connection tells ArcView where to find its values on the network. When you log in, the relational database verifies your user name and the permissions you have in the database. Different users may not be able to access the same set of tables. Similarly, you may be able to read from tables in a database but unable to edit the data in those tables.

Before deciding on the minimum Units_1det value for the campaign, you need to get some information about the Units_1det values. It would help to look at a sample of the values to get a general idea of their magnitude and range. You can create a database table to see a database theme's attributes.

To see a database theme's attributes in a database table

1. Make your map, the U.S. Counties window, active by choosing U.S. Counties from the Window menu. If U.S. Counties isn't listed, choose U.S. Counties in the Project window's Views list, then click Open.
2. Make the Counties theme active.
3. Click the Open Theme Table button .


A database table appears containing the attribute values retrieved by your database theme's query.



<i>SHAPE</i>	<i>NAME</i>	<i>STATE_NAME</i>
Shape	Autauga	Alabama
Shape	Baldwin	Alabama
Shape	Barbour	Alabama
Shape	Bibb	Alabama
Shape	Blount	Alabama
Shape	Bullock	Alabama
Shape	Butler	Alabama
Shape	Calhoun	Alabama
Shape	Chambers	Alabama
Shape	Cherokee	Alabama
Shape	Chilton	Alabama

There are two types of database tables, forward only scrolling database tables and keyset database tables. A forward only scrolling database table allows you to scroll forward all the way through the table but limits how far backwards you can scroll. The number of records that you can access scrolling backwards is defined by the cache size. You can set the cache size using database table preferences, which is explained later in the chapter. A keyset database table supports selections and allows you to scroll forwards and backwards through the entire table.

Database tables that are created from database themes will be keyset tables except when the theme has a one-to-many join or if the theme has more records than the maximum keyset size. The maximum keyset size is described in more detail below.

To support selections and scrolling, keyset database tables require a unique string or integer column in one of the tables in the query. For tables defined from SDE database themes, this column is automatically set to SDE's unique id column. You can tell that the table created in this example is a keyset table because the select tool  is enabled. When a keyset database table is created, a link is established where a database theme selection is applied to the database table and a database table selection is applied to the associated database theme.

Maximum Keypset size

The maximum keyset size is the maximum number of records allowed for a keyset database table. Database tables with more records than the maximum keyset size will be converted to forward only scrolling tables.

Since keyset database tables provide more functionality, they are usually slower to generate and use more memory. It may be inefficient to use keyset database tables for very large datasets. You can use the maximum keyset size to prevent database tables with more than a certain number of records from being created as keyset tables.




The default maximum keyset size is 50,000 records. You can adjust this setting as follows:

1. Make the Project window active.
2. From the Project menu choose Database Table Preferences.
3. In the Database Table Preferences dialog, set the maximum number of rows in a keyset table to the desired value.
4. Click OK.

The spatial column in the database table contains the word Shape for each record you can see. This column does not contain the actual shape value displayed in the map. The purpose for displaying the word Shape is to indicate the record has a shape value.


Repairing a database theme's connection

If you type your password incorrectly or if the SDE server isn't running, ArcView won't be able to reestablish the database connection. ArcView will still open your project, but you won't see any shapes in the map. Until you've done some project repair, ArcView will display an error message for each database theme visible in the map that doesn't have an active connection. After opening the project, you can reestablish the database connections yourself using the Theme Properties dialog.

1. Make your map, the U.S. Counties window, active by choosing U.S. Counties from the Window menu. If U.S. Counties isn't listed, choose U.S. Counties in the Project window's Views list, then click Open.
2. Make the States theme active.
3. Click the Theme Properties button . In the Theme Properties dialog, choose the Definition properties.
4. Click Create a New Connection.
5. Click the Connect button . A connection dialog appears.
6. Type the appropriate information into the Server:Instance, Username, and Password text boxes, then click Connect. Choose the database containing the tables used in this project (if appropriate).
7. Click OK in the Connection dialog.
8. Click OK in the Theme Properties dialog.
9. Make the Cities theme active.
10. Click the Theme Properties button .
11. Click Use Existing Connection, then choose the connection you created above from this list.
12. Click OK.
13. Repeat steps 9–12 for the Counties theme.

A sample script called SDEODBCREST.AVE has been provided to repair a database table's connection. You can find information about this script by opening the online help, clicking the index tab, and typing 'Database Access Extension, Sample Scripts'. Then, Double-click Sample Scripts, choose 'SDEODBC Restoring disconnected Database tables' and click Display.

If the values in the relational database have changed since you created the table, you can see those changes by choosing Refresh from the Table menu. ArcView queries the database again and updates the values in your table.

Scroll right until you can see the Units_1det column in the table. To see all of a record's attributes at the same time, click on it with the Identify tool . They appear in the Identify window, which you can enlarge. Scroll down in the database table to review some of the values in the Units_1det field. Most values should be under 20,000.

You can get statistics about the values in this field for all the records selected by the table's query by following the steps below.

To get statistics for a column of values in a database table

1. Make the Units_1det column active by clicking on its name at the top of the table.
2. From the Field menu, choose Statistics.
3. A message box appears containing statistics for the values in the Units_1det column. There are a total of 3,140 counties in the United States. The maximum number of single, detached houses in a county is 1,538,036, while the minimum number is 22; the mean is 19,230.4.
4. Click OK.
5. Close the table.

How many records have a Units_1det value that falls below the mean? You can find out by creating a new database table to calculate that information.

To create a new database table

1. If the Project window isn't active, make it active by choosing Sdetutorial.apr (the name you saved this project file under) from the Window menu.
2. From the Project menu, choose Add Database Table. The Add Database Table dialog appears. Choose SDE from the Database Access drop down list. Click Use an existing connection, then choose your connection from the drop-down list.
3. Double-click the Counties table in the Tables list (e.g., World.Counties); it appears in the From box.
4. Double-click the Codemog table in the Tables list (e.g., World.Codemog); it appears in the From box.
5. Place the cursor in the Select box, then type, "Count(*) as Num_Counties, Count(*)/3140*100 as Pct_Counties".

6. Place the cursor in the Where box, then type, “World.Counties.Fips = World.Codemog.Fips and World.Codemog.Units_1det < 19230.4”. Instead of typing the column names, you can place them by double-clicking the Columns list.
7. From the Unique Column drop down list, choose <none>.
8. In the Table Name box, give a name for this database table (e.g., County Data).
9. Click Query.
10. Close the Add Database Table dialog.

A new database table is created when you click Query in the Add Database Table dialog. This table shows that 2,496 of 3,140 records, or 79 percent, have a Units_1det value that falls below the mean.

Since the Unique Column was set to <none>, a forward only scrolling database table is created instead of a keyset database table. You can tell that the table is forward only scrolling because the select tool is disabled.

The query created above is called a Select statement. It’s a kind of SQL statement that chooses the data you want to see from the database. A Select statement consists of three parts, or clauses: Select, From, and Where. The Select clause chooses the columns you want to see, From identifies the tables containing the columns, and Where specifies how the tables are joined together and which records you want to see. The queries used to create database themes are similar to SQL Select statements.

More about queries

Earlier you retrieved statistics for the values in the Units_1det column by choosing Statistics from the Field menu. This method won’t always work, depending on how you specify a column of values in the Select statement. However, you can write a new query to create a database table containing the statistics you want. The following statement retrieves the number of values in the Units_1det column, and the maximum, minimum, and mean values:

```
SELECT Count(World.Codemog.Units_1det),  
MAX(World.Codemog.Units_1det), MIN(World.Codemog.Units_1det),  
AVG(World.Codemog.Units_1det) FROM World.Counties, World.Codemog  
WHERE World.Counties.Fips = World.Codemog.Fips
```

You can add other statistical operators to this Select statement to retrieve a complete set of statistical values.

The Select clause in the statement specifies calculations used by the database to create the values that appear in the table. The Count(*) operator counts the records selected by the where clause, the records below the mean Units_1det value. The Select clause also specifies aliases for the calculations (Num_counties and Pct_counties), which become the column names in the database table. If aliases weren't provided, the columns would be named with the calculations.

If you don't have a lot of database experience and you're going to do a lot of work with database themes, you should take the time to learn SQL. The on-line Help topic, "Selecting database theme features by their attributes," will help you get started. You can find this topic by opening the Help, clicking the Index tab, then typing the topic's name into the field at the top of the dialog.


The syntax of the queries in this book and in the Help topic are correct for Oracle databases only. Each relational database requires its queries to be a little bit different. For example, some databases are case-sensitive for table and column names, while others aren't. If you don't know the characteristics and syntax rules of your database, talk to your database administrator and get documentation for the relational database.

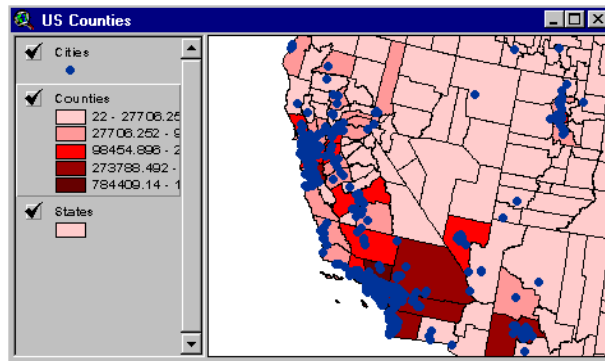
Symbolizing your data can be more than changing its color. You can classify the features in a theme according to their attributes. For your advertising campaign you want to find the counties with the largest number of single, detached houses; you can find them by classifying the Counties theme using the Units_1det column. Use a Natural Breaks type of classification, which divides the values into groups according to their natural distribution.

To classify a theme's features by their attributes

1. Make the U.S. Counties window active by clicking on its title bar.
2. Zoom into a scale where you can see the Counties theme.
3. Double-click on the Counties name in the map's Table of Contents to display the Legend Editor.
4. Choose Graduated Color from the Legend Type drop-down list.
5. Choose the Units_1det column from the Classification Field drop-down list. The counties are automatically classified into five classes using an Equal Interval classification. The classes are created by dividing the range between the minimum and maximum by five.
6. Click Classify.
7. Choose Natural Breaks from the Type drop-down list.
8. Click OK. The values represented by each class have changed.

9. You can change the color of each class by double-clicking on its symbol and assigning a new color from the Symbol Window, or you can change all the colors in the legend at once by choosing a ramp from the Color Ramps drop-down list below.
10. Click Apply. The theme redraws with the new legend.
11. Close the Legend Editor.

Looking at the classified Counties theme in your map you can see that most counties fall into the class with the lowest range of single, detached houses. To see a good cross-section of classes, zoom in to the Southwestern states, particularly California and Nevada, with the Zoom In tool .



The upper limit of the lowest class is 27,706.252. The mean value for the Units_1det column falls well inside the lowest class of values. Since the Natural Breaks analysis has identified this legend class as an associated group of values, it would be helpful to know how many counties are in this class. You can calculate that number by modifying your database table's query. Change the number in the where clause to 27,700.



To change a database table's definition query

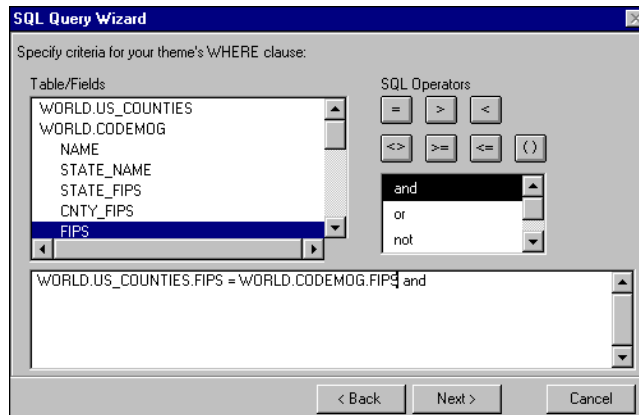
1. Make the County Data table active by clicking on its window.
2. From the Table menu, choose Properties. The Database Table Property dialog appears, containing the SQL Select statement that was sent to the relational database to create the database table.
3. In the SQL Statement box, delete the number 19230.4, then type in the number 27700.
4. Click OK.

The contents of the database table changes to reflect the new SQL query. The table now shows that 2,703, or 86 percent, of the counties fall within the lowest class in the database theme's legend.

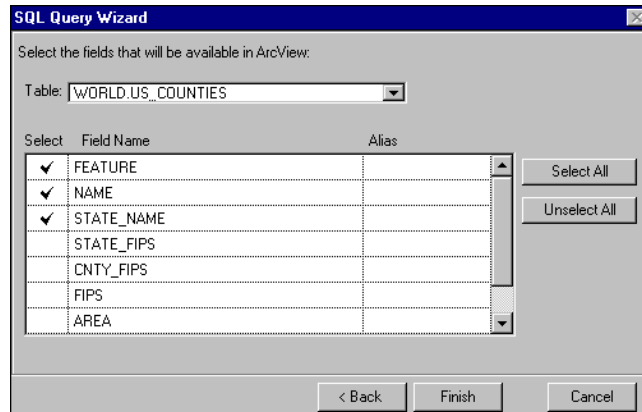
The candidate counties for the advertising campaign are those falling outside the lowest legend class; that is, counties whose Units_1det value is greater than 27,700. Redefine the Counties theme's query to represent these counties by changing its where clause.

To change the shapes and attributes a database theme represents


1. Close the database table, then make the U.S. Counties window active by clicking on its title bar.
2. Make the Counties theme active, then click the Theme Properties button . The Theme Properties dialog appears. Make sure the Definition property is selected.
3. Click the Query Builder button  in the Theme Properties dialog. The SQL Query Wizard appears.
4. Click Next. The where clause panel contains the theme's complete where clause, which includes the join statement.
5. Double-click 'and' in the SQL operators list to link the attribute criteria to the join statement in the where clause.
6. Double-click the Codemog table's name to see a list of its columns.



7. Double-click the name of the Units_1det column. It appears in the where clause below.
8. Click the Greater Than button to add the > operator to the where clause.
9. Type in "27700". The where clause now contains a join statement and attribute criteria. Click Next.
10. Click Unselect All, then click in the Select column next to the Feature, Name, and State_Name columns. You can provide aliases for these attributes that you'll see when working with this theme by typing into the Alias column.



11. From the Table drop-down list, choose the Codemog table.
12. Click Unselect All, then click in the Select column next to the Units_1det field.
13. Click Finish.
14. Click OK in the Theme Properties dialog.

The Counties theme redraws, showing only the counties satisfying the new query. If you identify one of the counties with the Identify tool , you will see its values for the Name, State_Name, and Units_1det attributes selected above.

In the next exercise, you'll continue working with this project, so you can begin Exercise 3 now. If you want to exit ArcView now and continue with the next exercise later, remember to save your work.

Exercise 3: Find locations for an advertising campaign


In the previous exercise, you chose the candidate counties for your advertising campaign. However, you only want to advertise in urban areas, and at first you only want to advertise in three states. Based on the results of the campaign in these states you can modify the campaign as it grows.

First you need to find out where the urban centers in the United States are by finding the cities with a population of at least 100,000. Then you can find out which counties are within 50 miles of those cities. The states you start the campaign in will be the three states with the highest number of single, detached houses.


In this exercise, you'll learn how to:



- Change a database theme's attributes and the tables they come from.
- Find features in a map by their attributes.
- Find features in a map by their spatial relationship with other features.
- Export features from a database theme to a shapefile.
- Create a summary database table.
- Sort the records in a database table.
- Join attributes in a database table to features in a shapefile.

If ArcView isn't already running, start it. Open the project from Exercise 1 by choosing Open Project from the File menu. Log in to the database connections stored in your project by typing your user name and password into the Login dialog.

To locate the urban centers in the United States, find out which cities have a population of at least 100,000. If you identify a city in the Cities theme using the Identify tool , you won't see a population attribute. The cities demographic data is stored in a different table in your database. You need to join the city and demographic tables together, then add the Pop1990 column to the theme's attributes. The common field between these tables is State_City. The steps below refer to the cities demographic table as Cidemog.

To change a database theme's attributes

1. Make the Cities theme active.
2. Click the open theme table button . Move and/or resize the database table and the view so they do not overlap and are both visible.
3. Click on the View to make it the active document.

4. Click on the Theme Properties button . The Theme Properties dialog appears. Make sure the Definition property is selected.
5. Click the Query Builder button  in the Theme Properties dialog.
6. Choose the Cidemog table from the Available Tables list, then add it to the Selected Tables list by clicking the '>' button. Click Next.
7. Double-click the Cities table's name to see a list of its columns.
8. Double-click the name of the State_City column. It appears in the where clause below.
9. Click the Equal To button to add the = operator to the where clause.
10. Double-click the Cities table's name to hide its list of columns, then double-click the Cidemog table's name to see a list of its columns.
11. Double-click the name of the State_City column. It appears in the where clause below. Click Next.
12. From the Table drop-down list, choose the Cidemog table.
13. Click Unselect All, then click in the Select column next to the Pop1990 column. Click Finish.
14. Click OK in the Theme Properties dialog.

The Cities theme redraws and the Cities database table is refreshed. The population attribute that was added to the theme's definition is also added to the database table.


In the steps above, you defined a new join statement because you added a table to the theme's query. If you remove a table, you also need to remove the corresponding join statement from the where clause.

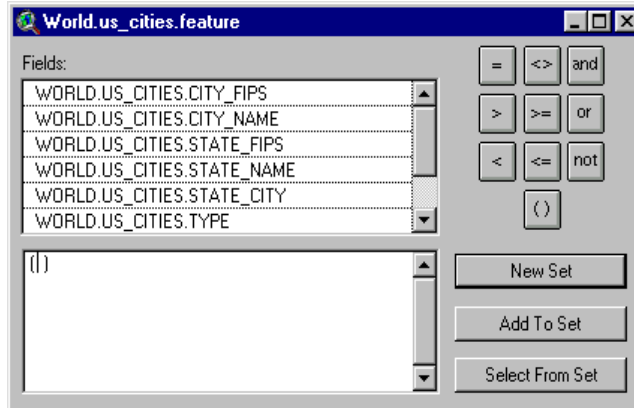
The type of join used by the Counties and Cities themes is called an inner join, where the rows returned have the same value in both tables for the common field. There are other ways to join tables together. Suppose the Codemog table had data for the State of New Hampshire only. An outer join would retrieve a row for every US county, but only the New Hampshire counties would have values in fields from the Codemog table.

To define outer joins for Oracle databases, the syntax is "World.Counties.Fips = World.Codemog.Fips(+)". To define an outer join statement when creating a database theme, skip the Join panel by clicking Next, then build it in the where clause panel.



Now find out which cities have a population over 100,000. Using the SQL Query Builder, specify attribute criteria to select those cities. The attribute selection criteria are appended to the theme's query to create a new selection query. ArcView finds the records satisfying the Selection query, then highlights their shapes by drawing them in yellow on the map.

To find features in a map by their attributes

1. Make sure the U.S. Counties window active.
2. Click the Query Builder button  in the button bar for Views. The SQL Query Builder dialog appears.



3. Double-click the Cidemog.Pop1990 column. It appears in the where clause below.
4. Click the Greater Than button to enter the > operator into the where clause.
5. Type in “100000”.
6. Click New Set. ArcView’s status bar shows that 200 cities have been selected.
7. Close the Query Builder dialog.

Spatial queries can be used to select features in a database theme as well. Using the Selection tool , select features by clicking or by dragging a rectangle over them, or draw graphic shapes on the map and then select the features intersecting those shapes by clicking the Select by Graphics button . The spatial selection criteria are appended to the theme’s query to create the selection query.

Since the Cities database theme is linked to the Cities database table, the cities with a population greater than 100,000 are selected in both the theme and the table.

You can find features in one theme that are spatially related to the selected features in another theme. This is called theme-on-theme selection. To find the US counties near urban centers you’ll select the counties within 50 miles of the selected cities.

To find features in a map by their spatial relationship with other features

1. Make the Counties theme active.
2. From the Theme menu, choose Select By Theme. The Select By Theme dialog appears.

3. From the second drop-down list, choose the Cities theme.
4. From the first drop-down list, choose Are Within Distance Of.
5. In the Selection distance box, type 50.
6. Click New Set.

ArcView's status bar shows that 377 counties have been selected. This is the final list of counties to advertise in.

When you save a project, ArcView doesn't record which features in a database theme are selected because if the database is dynamic, today's queries can produce different results tomorrow. To preserve the selected counties, you need to export them to a shapefile.

To export features from a database theme to a new shapefile

1. From the Theme menu, Choose Convert to Shapefile.
2. In the dialog that appears, specify the name (e.g., mycounties) and location of the new shapefile that will be created. Click OK.


Optimizing your queries

Tables in a relational database can contain millions of records. Even simple queries against large tables can take time. Try to optimize queries to get the best performance possible.

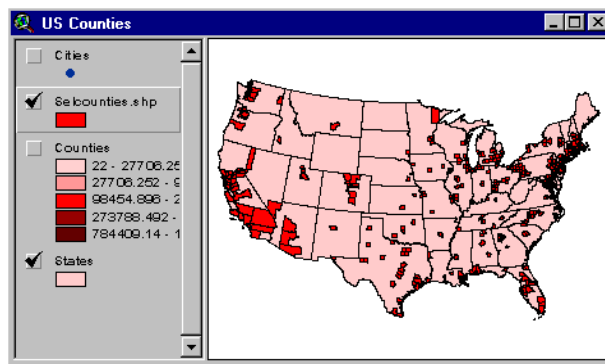
Index the columns used in join statements and attribute criteria to improve performance (you can do this with Avenue scripts). Choose the SQL operators for attribute criteria carefully. Some operators are faster; some don't use indexes even if they exist. In a join statement, placing tables on the left or right of the equal sign can impact performance. These are a few considerations. Carefully read the database's documentation to learn more about optimizing attribute queries.

SDE lets you use spatial criteria for queries as well. With Avenue scripts, you can create spatial criteria to select the features a database theme represents. To optimize queries with both attribute and spatial criteria, evaluate first the criteria that eliminates the most records. If the map contains few features selected by attribute criteria that are spread across a large area, evaluate the attribute criteria first. If your map covers a small area, evaluate the spatial criteria first.

Change the order for evaluating criteria by choosing Attributes first or Spatial first in the Theme Properties dialog. Optimized queries (the default) are almost always Spatial first; they evaluate attribute criteria first only when they have spatial criteria with a negative test, for example, "select features not inside the buffer".

3. Click Yes to add this new shapefile to the view as a theme.
4. Turn off the Counties theme in the view.
5. Draw the new theme by clicking the check box next to its name in the Table of Contents. Make sure the Cities theme is at the top of the Table of Contents so it draws on top of the other themes.
6. Clear the selected cities by making the Cities theme active, then clicking the Clear Selected Features button .

If you identify a county in the shapefile theme using the Identify tool, you'll see that only the attributes that were part of the database theme are now part of the shapefile.



There are several other reasons you might export a database theme's features to a shapefile. You might create a shapefile to use your SDE data with other ArcView extension software such as ArcView Network Analyst, ArcView Spatial Analyst, or ArcView 3D Analyst™. You might also export data to use it locally when you don't have access to an SDE server (e.g., when you are using a portable computer) or to release an SDE client license for someone else in your organization to use.

The final step, in this exercise, is to find out in which three states you'll begin the advertising campaign. Find the states with the highest number of single, detached houses by creating a database table that summarizes the Units_1det values for each state.

To create a summary database table

1. Make the Project window active by choosing Sdetutorial.apr (the name you saved this project file under) from the Window menu.
2. From the Project menu, choose Database Table Preferences.
3. Change the Fetch count to 40.
4. Change Cache size 200.

5. Click OK.
6. From the Project menu, choose Add Database Table. The Add Database Table dialog appears.
7. Choose SDE from the Database Access drop-down list.
8. Double-click World.Counties in the Tables list. The Counties table appears in the From box.
9. Double-click World.Codemog in the Tables list. The Codemog table appears in the From box.
10. Place the cursor in the Select box, then type “World.Counties.State_Name, SUM(World.Codemog.Units_1det) as Total_1det”.
11. Place the cursor in the Where box, then type “World.Counties.Fips = World.Codemog.Fips Group By World.Counties.State_Name”.
12. From the Unique Column drop down list, choose <none>.
13. Set the Table Name to States Summary.
14. Click Query.
15. Close the Add Database Table dialog.

A new forward only scrolling database table appears containing the total number of single, detached houses by state for the first 40 rows.

The first 40 rows are retrieved because you set the fetch count, which is the number of rows retrieved from the database at once, to 40. If you scroll forward past the first 40 records, you will see that the table size grows to 51 records. This is because a second fetch was required to view the rows below the first 40. The cache size was set to 200, because this is the minimum cache size and it is large enough to hold the entire table in memory.

By default, the Fetch count is 100 rows and the Cache size is 1000 rows. With the default settings, any table with less than 1000 rows is held entirely in memory. If the table is forward only scrolling and has more than 1000 records, you may not be able to scroll back to the beginning of the table. For example, if you scroll forward to record 1100 and then scroll backwards, you will only be able to get back to record 100 since only 1000 records can be cached. You can get back to the beginning of the table by choosing refresh from the table menu.

The Fetch count must be greater than or equal to 20 and less than or equal to 200. The cache size can be set to either 0 or greater than or equal to 200. If the cache size is set to 0, all records will be cached in memory.

Although not as obvious, the Cache size and Fetch count also affect keyset database tables.


You must consider the amount of memory on your machine and the number of tables you plan to create when setting the cache size. The larger you can set the cache size, the better the scrolling performance since more of the table is held in memory at once.

The query used to create the States Summary table uses a group by clause with the State_Name column. This means that each row in our States Summary table can be uniquely identified by state name. We can use the State_Name column as a Unique Column and convert the table to a keyset table.

To convert to a keyset database table

1. Make the States Summary table active.
2. From the Table menu choose Properties. The Database Table Property dialog appears.
3. From the Unique Column Table drop-down list, choose World.Counties.
4. From the Unique Column drop-down list, choose STATE_NAME.
5. Click OK.

The select tool and the select buttons in the table GUI are now enabled and ToolBar displays the total number of records returned from the query (51).

To make the table easier to analyze, you can sort the rows in the table based on the Total_1det column. To do this, add the OrderBy clause to the Table's query. Sorting this way ensures that the rows will always appear in sort order, even if the table is refreshed. If you use the sort descending button , the sort order is temporary and will be lost when the table is refreshed.

Working with database tables

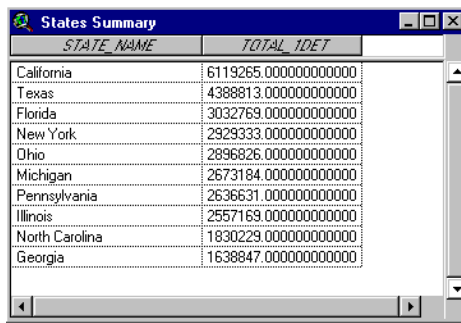
Some table functions work a little differently on database tables than they do on normal tables.

- The Find function is usually case sensitive on database tables. It will be case sensitive if searches involving data values are case sensitive in your database.
- If you get statistics about a column in the table, they're calculated for all the records selected by the table's query, not just the current selection.
- A chart cannot be created for a database table directly. You must first export it to a normal table.
- When you refresh a database table, the selection is lost.

To sort the records in a database table

1. From the Table menu, choose Properties.
2. Click in the SQL Statement box, placing the cursor at the end of the SQL statement.
3. Type in a space, then type, “Order By Total_1det desc”.
4. Click OK.



The values in your database table change to reflect the new SQL Select statement.



STATE_NAME	TOTAL_1DET
California	6119265.000000000000
Texas	4388813.000000000000
Florida	3032769.000000000000
New York	2929333.000000000000
Ohio	2896826.000000000000
Michigan	2673184.000000000000
Pennsylvania	2636631.000000000000
Illinois	2557169.000000000000
North Carolina	1830229.000000000000
Georgia	1638847.000000000000

You'll start advertising in California, Texas, and Florida. There are two ways to find the counties in those states. You can redefine the counties shapefile to contain only counties whose state names are California, Texas, or Florida. Alternatively, you can join this database table to the shapefile's attribute table, then redefine the theme to contain only the counties with a Total_1det value greater than 3,000,000.

To join attributes in a database table to features in a shapefile

1. Make the U.S. Counties window active.
2. Make the Mycounties theme active.
3. Click the Open Theme Table button . The theme's attribute table, called Attributes of Mycounties, appears.
4. Make the States Summary table active, then make its State_Name field active.
5. Make the Attributes of Mycounties table active, then make its State_Name field active.
6. Click the Join button .

Now the local shapefile contains values from the database table in addition to its own values. Those values aren't saved with the shapefile; they're stored in a temporary file separate from the database table. ArcView will retrieve values from the database and re-create the join every time you open this project. For information on redefining a shapefile theme, see the *Using ArcView GIS* book.

The examples you've completed illustrate how to use database themes, database tables, and shapefiles together to solve spatial problems. Database themes allow you to work with an extremely large number of features efficiently, while shapefiles are appropriate for working with a small number of shapes that you've found using database themes. Database tables let you calculate and summarize values and get statistics about your data; database themes don't. Each component provides a unique set of capabilities, and therefore you often need to use all components together to generate a solution.

There are several kinds of problems you can't solve using ArcView's interface, but you can using Avenue scripts. To learn how, you first need to understand which objects you need to use and how they work. The following chapters explain the Database Access objects and will get you started writing Avenue scripts for your SDE data. However, if you've never used Avenue before, you should review the *Using Avenue* book before moving on to Chapter 4.

CHAPTER 3

Quick start tutorial for ODBC

The exercises in this chapter begin with the basics of adding a database table, joining database tables, and changing the query for your database table using ODBC.

To follow the exercises, you need access to the ODBC database containing city data for the United States, and demographic tables for U.S. cities. You should review the section “Before you start the ‘Quick start tutorial exercises’” in Chapter 1 before you begin.

Each exercise builds on the previous one. You can work through them at your own pace. When you stop, save your work so you can pick up where you left off.

In this chapter, you’ll perform exercises with the Database Access extension to identify potential markets for new home developments:

- Exercise 1: Identifying potential market cities in the United States
- Exercise 2: Narrowing your search for new market cities

Exercise 1: Identifying potential market cities in the United States

Suppose you work for a national builder of residential homes. It is your job to identify potential markets for new homes. Initially, you want to come up with a sizable list that will allow you to apply other criteria, such as cities with a population greater than 100,000, to narrow down the search. To start off, you are going to identify larger market cities. These cities represent the potential markets that you want to focus on.

You have an external database of demographic information, but you want to use this information within ArcView. As you narrow the field of potential new cities and locations within those cities, you will want the spatial analysis capabilities of ArcView.

In this exercise, you'll learn how to:

- Load ArcView extension software.
- Add a forward only scrolling database table using the SQL Query Wizard.
- Convert a forward only scrolling database table to a keyset database table.
- Use selections on a keyset database table.
- Create a new keyset database table.
- Save your work.

Start ArcView. Before you can add your database table, you need to load the Database Access extension. Loading the extension makes the required components available for connecting ArcView to an ODBC database and working with ODBC data.

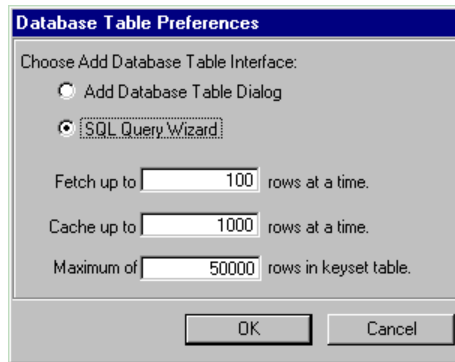
To load ArcView extensions

1. Make the Project window active.
2. From the File menu, choose Extensions.
3. Click the check box next to Database Access.
4. Click OK.

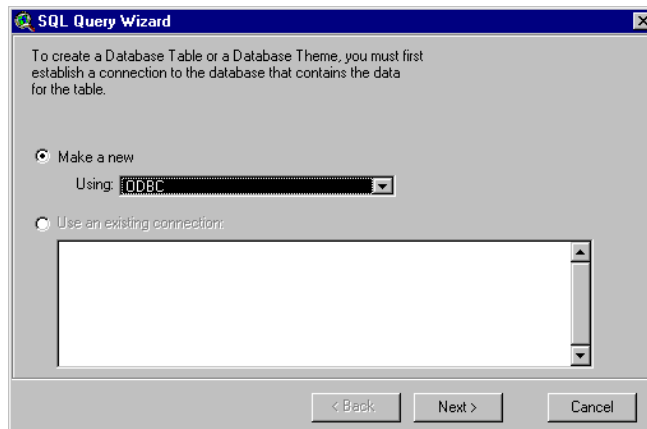
To add a forward only scrolling database table using the SQL Query Wizard

1. Make the Project window active.
2. From the Project menu choose Database Table Preferences.

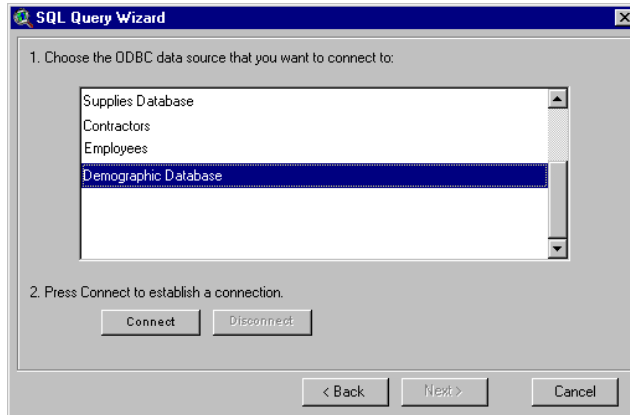
3. Choose the SQL Query Wizard option on the Database Table Preferences dialog and click OK. When you choose this option you will use the SQL Query Wizard dialogs to add your database tables.



4. From the Project menu, choose Add Database Table. The SQL Query Wizard dialog appears. It will guide you through the steps of adding an ODBC database table to your project.
5. Click the Make a new connection option, and then from the drop-down list choose ODBC. Click Next.



6. For this exercise you want to connect with the database that contains U.S. State, City and County data and the City and County demographic data. Once you choose the appropriate ODBC data source, click the Connect button. If your database requires it, a dialog appears for login information. Type your Username and Password in the appropriate text boxes, then click OK. Be careful when you type; ODBC is case sensitive with these text strings. When the Connect button is dimmed, you've successfully connected to the data source with ODBC. Click Next.



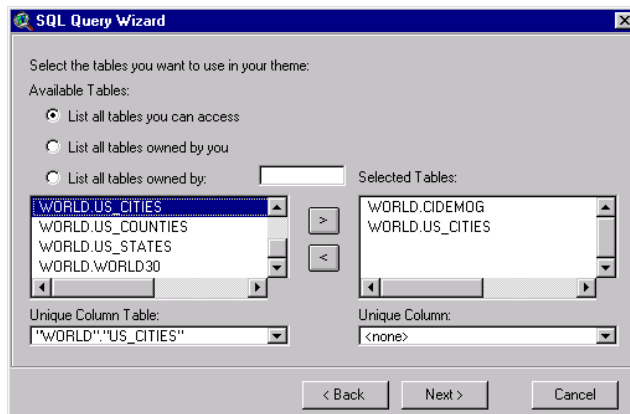
7. Choose the Cidemog table from the Available Tables list and click the '>' button to send it to the Selected Tables column. Locate the Cities table in the same list and click the '>' button to send it to the Selected Tables list.

The Tables list may not show the names “cidemog” or “cities”. The names may appear as something like “World.US_Cities” and “World.Cidemog”. In this example, the word World appears because the World database user owns these tables. The Cities table is named US_Cities to distinguish it from other city tables. The database and the naming convention that you use will determine how the tables appear in the list.

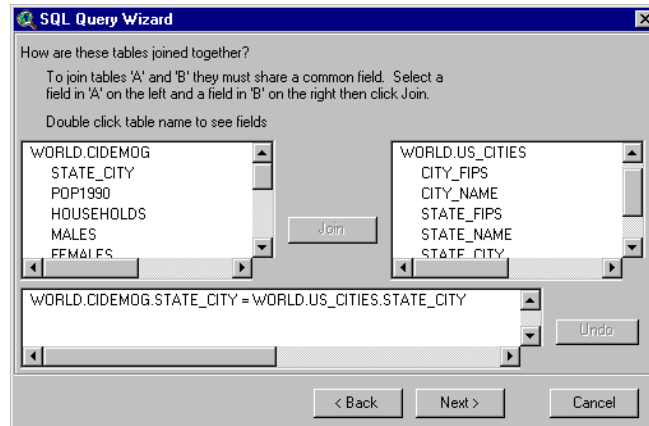
When tables are added to the Selected Tables list, they are also added to the Unique Column Table list.

8. Choose the Cities table from the Unique Column Table list. This fills the Unique Column list with the integer and character fields in the World.US_Cities table. From the Unique Column list, choose <none>. Setting the Unique Column to <none> means that a forward only scrolling database table will be created.

Click Next.



9. Double-click the Cities table in one list and the Cidemog table in the other to see a list of columns in each table. Highlight State_City in each list, then click Join. The Join statement appears below. Click Next.



10. Click Next on the following screen.
11. Click Finish on the last screen. Your new database table is created.

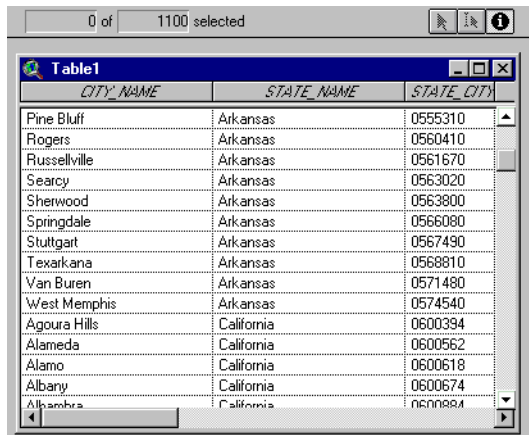
ODBC allows you to connect to a database (like Oracle or Sybase) and access the data stored in the database. The data can be queried through the Database Access extension and joined to ArcView tables. Instead of storing the data locally in your project, only the query used to retrieve the data is stored. This query retrieves the data from the database when the project is reopened. If the data has been updated since the project was created, the updated table will appear in your project.

When you stepped through the SQL Query Wizard, you generated an SQL query that was executed against the database. The query that you created joined two tables together. All columns from the US_Cities and Cidemog tables and all rows where the State_city values in the Cidemog table equal the State_city values in the US_Cities table were returned. For example, if the Cidemog table had demographics for Los Angeles only, the database table would only display data for Los Angeles.

When the data is returned it is displayed in the database table. There are two types: forward only scrolling database tables or keyset database tables. A forward only scrolling database table allows you to scroll forward all the way through the table but limits how far backwards you can scroll. The table that you have just created is a forward only scrolling database table.

Viewing data in a forward only scrolling database table

1. Make the table active. Notice how the ToolBar indicates that there are only 100 records in the table.
2. Scroll vertically to the bottom of the database table. When you get to the bottom of the first 100 records, another 100 records are added and the ToolBar indicates that there are 200 records in the table.
3. Continue scrolling forward till the record count reaches 1,100. As you scroll, the table continues to grow and the scroll position indicator decreases in size.
4. Scroll vertically backwards as far as the table allows. When you reach record 100, the table will stop scrolling and not let you access records above this location. By default, you can scroll backwards up to 1,000 records from your current position in the table.



CITY_NAME	STATE_NAME	STATE_CITY
Pine Bluff	Arkansas	0555310
Rogers	Arkansas	0560410
Russellville	Arkansas	0561670
Searcy	Arkansas	0563020
Sherwood	Arkansas	0563800
Springdale	Arkansas	0566080
Stuttgart	Arkansas	0567490
Texarkana	Arkansas	0568810
Van Buren	Arkansas	0571480
West Memphis	Arkansas	0574540
Agoura Hills	California	0600394
Alameda	California	0600562
Alamo	California	0600618
Albany	California	0600674
Alhambra	California	0600884

5. Scroll forwards till there are no more records added to the table. The ToolBar indicates that there are 3,149 records in the table. This is the total number of records returned by your query.
6. From the Table menu click Refresh. This repositions you back at the top of the database table and fills it with the first 100 records returned by the query. Refreshing the table at any point will reset the table to its original position.

Cache size and fetch count

The cache size determines the maximum number of records that will be held in memory by each database table you create. The fetch count is the maximum number of records retrieved from the database at once. The default cache size is 1,000 records, and the default fetch count is 100 records. This is why it is possible to scroll backwards up to 1,000 records in the table and why the table grows 100 records at a time when scrolling forwards. The cache size and fetch count also affect keyset database tables, which is discussed below.

The cache size and fetch count can be adjusted in Database Table Preferences. You must consider the amount of memory on your machine and the number of tables you plan to create when setting the cache size. The larger you set the cache size, the better the scrolling performance, since more of the table is held in memory at once. To demonstrate this, you can adjust the cache size and fetch count and re-create the database table as follows:

1. Make the Project window active.
2. From the Project menu choose Database Table Preferences.
3. In the Database Table dialog, set the cache size to 0 and the fetch count to 200. A cache size of 0 means that all records returned by the query will be cached. The maximum fetch count is 200.
4. Repeat steps 4–10 from the ‘To add a forward only scrolling database table using the SQL Query Wizard’ section above.

When the table is created, scroll to the bottom. Once this is done, all of the records returned from the query have been fetched and stored in memory. You can now scroll all the way forwards and backwards in the table, and there are no pauses as you scroll. Before creating another database table, you may want to go back into Database Table Preferences and reset the cache size to decrease the amount of memory used for the new tables.

A keyset database table allows you to select records and scroll back and forth through the entire table. In order to create a keyset table, one of the tables in your query must contain either an integer or a character column with a unique value for each record returned by the query. You can tell that you have successfully created a keyset table if the select tool and the select buttons in the table GUI are enabled.

The State_City column in the US_Cities table is a unique character column and can be used to make the table a keyset table. You should also rename the database table to something more meaningful, like “City Demographic Data”. Using table properties, both of these tasks can be performed at once, as you’ll see in the next section of this exercise.

To rename and convert to a keyset database table

1. Make the table active.
2. From the Table menu choose Properties. The Database Table Property dialog appears.
3. In the table name textbox, set the name of the table to “City Demographic Data”.
4. From the Unique Column Table drop-down list, choose the Cities table.
5. From the Unique Column drop-down list, choose State_City. Choosing State_City as the unique column will transform the table to a keyset table. Click OK.

The select tool and the select buttons in the table GUI are now enabled and the database table is named “City Demographic Data”. The ToolBar displays the total number of records returned from the query (3,149) since the table is now a keyset table and can access all records.

Maximum keyset size



The maximum keyset size is the maximum number of records allowed for a keyset database table. Database tables with more records than the maximum keyset size will be converted to forward only scrolling tables.

Since keyset database tables provide more functionality, they are usually slower to generate and use more memory. It may be inefficient to use keyset database tables for very large datasets. Using Database Table Preferences, you can change the maximum keyset size to prevent database tables with more than a certain number of records from being created as keyset tables.

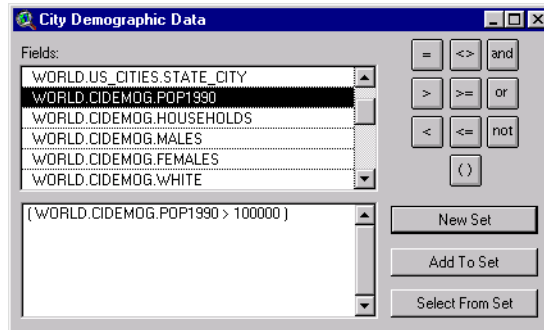
The default maximum keyset size is 50,000 records.


Since your company is only interested in larger market cities that have a total population greater than 100,000, you can begin to eliminate some of the cities listed in the table. Before applying this condition to a new database table, you can see the effect this will have on the data by making selections with the query builder.

Selecting records in a database table

1. Click the POP1990 column on the City Demographic Data table to make it active.
2. Click the Sort ascending button . This sorts the records in ascending order based on population.
3. Click the query builder button . The Query builder is displayed.

4. Double-click the POP1990 field (e.g., World.Cidemog.POP1990), then click the ‘>’ button and type in “100000”. Click New Set to select the cities that have a population greater than 100,000. Dismiss the query builder by clicking the ‘X’ button.



5. Click the Promote Button . This brings the selected records to the top of the table.

All of the cities in the U.S. with a population greater than 100,000 are now displayed in yellow at the top of the City Demographic Data table. The records are also ordered from smallest to largest population. You can see that applying this condition will narrow the dataset to 200 cities.

Next, you will create a new keyset database table of just the records that satisfy the query.

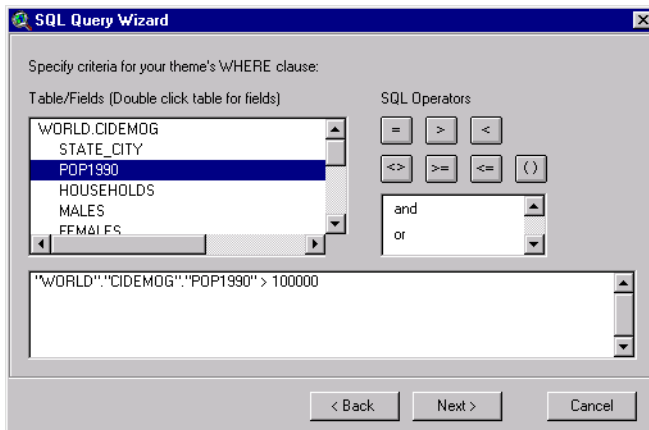
Working with database tables

Some table functions work a little differently on database tables than they do on normal tables.

- The Find function is usually case sensitive on database tables. It will be case sensitive if searches involving data values are case sensitive in your database.
- If you get statistics about a column in the table, they’re calculated for all the records selected by the table’s query, not just the current selection.
- A chart cannot be created for a database table directly. You must first export it to a normal table.
- When you refresh a database table, the selection is lost.

Create a new keyset database table

1. Make the Project window active.
2. From the Project menu choose Add Database Table. The SQL Query Wizard will appear.
3. Choose the Use an existing connection option and select the connection you used to create the City Demographic Data table. Click Next.
4. Choose the Cidemog table (e.g., World.Cidemog) and the cities table (e.g., World.US_Cities) from the list (hold down the Shift key as you click the tables) and send both to the Selected Tables column. To make a keyset table, choose the cities table from the Unique Column Table list and State_City from the Unique Column list. Click Next.
5. Double-click the Cidemog table in the left table list and the cities table in the right table list. Select the State_City field in each list and click Join. This will join the two tables based on the State_City field. Click Next.
6. Double-click the Cidemog table to list the fields, then double-click the 'POP1990' field. Click '>=' and type in 100000, where 100,000 is the smallest population that may appear in the table. Click Next.



7. On the next dialog you can choose not to display some of the columns of data. At this time you don't need to make any changes here. Click Finish. Your new table of data appears.
8. From the Table menu choose Properties and rename the table to "Larger Market Cities".

The query that you generated this time is the same as the query for the City Demographic table, except here you have specified that only cities with a population of greater than 100,000 are to be returned. This new table represents the larger market cities in the

United States as defined by your company. You now have a preliminary list that you will use to narrow down the search for new sites.

To save your work

1. Make the Project window active.
2. From the File menu choose Save Project.
3. In the dialog that appears, specify a name (e.g., “ODBCtutorial”) and location for the new project file, then click OK. ArcView will automatically add the .apr extension to the name you gave for the new project file.
4. Either close your project or continue on to the next exercise.

If you close your project now and open it again later, you will be prompted to log in if your database requires a user name and password. After you log in, ArcView will reestablish its connection to the database and retrieve the values for the tables.

Exercise 2: Narrowing your search for new market cities

Next, you are going to format a new query for cities to research as potential markets for new homes. You will continue to use the Larger Market Cities query as your base and add other criteria to narrow down the search.

In this exercise, you'll learn how to:

- Get statistics for a column of values in a database table.
- Add database tables from the Add Database Table dialog.
- Change a database table's definition query.
- Save your project.

If ArcView isn't already running, start it. Open the project you created in Exercise 1 by choosing Open Project from the File menu. If prompted, log in to the database connections stored in the project by typing your user name and password in the login dialog.

Repairing a database table's connection

If there was a problem connecting (e.g., incorrect password), ArcView won't reestablish your database connection. The project will still open, but you won't be able to open the database tables.

You can reestablish a database tables connection yourself using a sample script called SDEODBCREST.AVE. To find information about this script, open the on-line help. Click the index tab and type in 'Database Access Extension, Sample Scripts'. Then, double-click Sample Scripts, choose 'SDEODBC restoring disconnected database tables' and click Display.

Now you can apply more criteria to narrow your search for potential markets. The Larger Market Cities table contains many columns of demographic data for the Larger Market Cities in the United States. The Renter_occ field in this table contains the number of housing units occupied by renters in each city. The information in this field is of interest since renters represent potential home buyers. You can calculate statistics in this field to find more information.

Performing statistics within a table

1. Click the field labeled 'renter_occ' in the Larger Market Cities table to make it active.
2. From the Field menu choose Statistics.

Note If your database does not support the SQL STDDEV function, you will get an error message. If you see the message, click OK and all of the statistics in the next dialog will be valid except for Standard deviation.

3. The Statistics dialog box will appear displaying some statistics about the number of renter-occupied dwellings. As you can see, the count is 200, the maximum is 2012023, the minimum is 3976, and the mean is 63685 when rounded. Click OK.
4. Close the Larger Market Cities table.

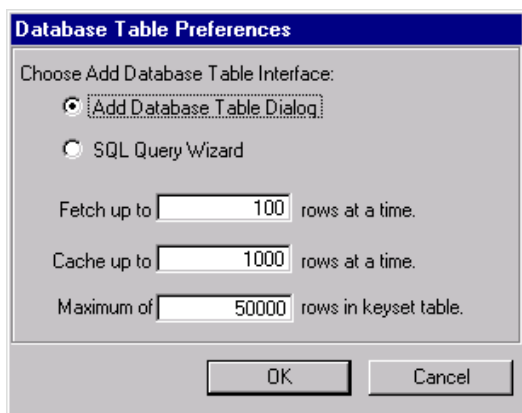
After reviewing this information, you decide you want to narrow the dataset to include only larger market cities with an above average number of renter-occupied dwellings. To apply this condition, you will create a new database table using the statistics information.

In the previous exercise you added a database table with the SQL Query Wizard. Next, you are going to add a database table using the Add Database Table dialog. The Add Database Table dialog provides all the functionality of the SQL Query Wizard in one dialog. This exercise will guide you through using the add database dialog, however, you should be familiar with SQL to use the Add Database Table dialog effectively.

The tables will only contain the records where Renter_occ is greater than 3,976.

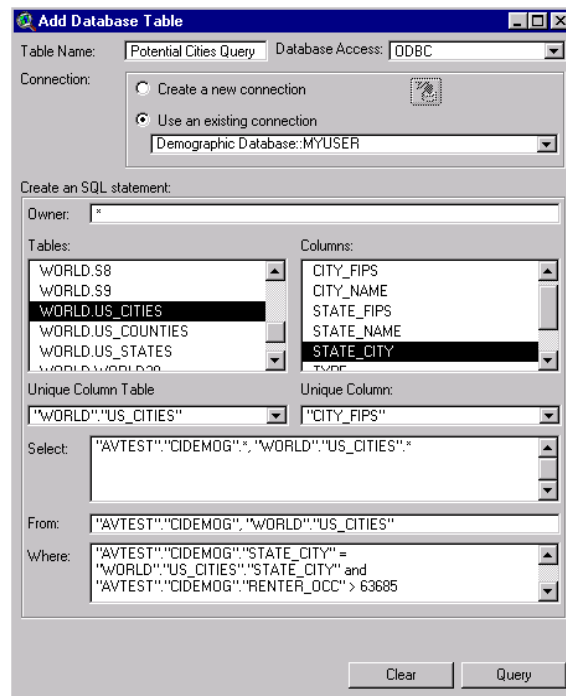
To add a database table using the Add Database Table dialog

1. Make the Project window active.
2. From the Project menu choose Database Table Preferences.
3. Choose the Add Database Table Dialog option and click OK.



4. Choose Add Database Table from the Project menu. The Add Database Table dialog appears.
5. Choose the Use an existing ODBC connection option.
6. Set the table name to “Potential Cities Query”.
7. Double-click the Cidemog table (e.g., World.Cidemog) in the Tables list to place it in the From box.
8. Double-click the Cities table (e.g., World.US_Cities) in the Tables list to place it in the From box.
9. Click inside the Where box to place the cursor there.
10. Click the Cities table in the Tables list to get a list of columns. Then double-click the State_City column in the columns list to place it in the Where box.
11. Type “=” in the Where box after World.US_Cities.State_City.

12. Click the Cidemog table in the Tables list to get a list of columns. Next, double-click the State_City column in the Columns list to place it at the end of the Where clause.
13. Type “and” at the end of the Where clause.
14. Double-click the Renter_occ column in the Columns list to add it to the Where clause, then type “> 63685” at the end of the Where clause.
15. Click inside the Select box to place the cursor there.
16. Click the Cidemog table to see a list of 15 columns. Double-click <All Columns> from the Columns list to add it to the Select Box. This indicates that all columns in the Cidemog table will be in the database table. Keyset database tables require that all columns be specified this way instead of just using ‘*’.
17. Click the Cities table to see a list of its columns. Double-click the <All Columns> option to add it to the Select box. This indicates that all the columns in the Cities table will also be in the database table.
18. Choose the Cities table in the Unique Column Table drop-down list. Then choose State_City in the Unique Column drop-down list. Selecting State_City as the Unique Column will make the database table a keyset table.



19. Click Query. Your Potential Cities Query database table appears.
20. Close the Add Database Table dialog.

The Potential Cities Query table contains all the large market cities in the United States with an above average number of renter-occupied dwellings.

The query created above is called a Select statement. It's a kind of SQL statement, where you choose the data you want from the database. A Select statement consists of three parts, or clauses: Select, From, and Where. The Select clause chooses the columns you want to see, From identifies the tables containing the columns, and Where specifies how the tables are joined together and which records you want to see.

More about queries

Earlier, you retrieved statistics for the values in the Renter_occ column by choosing Statistics from the Field menu. This method may not always work, depending on how you specify a column of values in the Select statement. However, you can use the Add Database Table dialog and write a new query to create a database table containing the statistical information you need. The following statement retrieves the number of values in the Renter_occ column, and the maximum, minimum, and mean values:

```
SELECT Count(World.Cidemog.Renter_occ) as COUNT,  
MAX(World.Cidemog.Renter_occ) as MAX,  
MIN(World.Cidemog.Renter_occ) as MIN,  
AVG(World.Cidemog.Renter_occ) as AVG  
FROM World.US_Cities, World.Cidemog  
WHERE World.US_Cities.State_City = World.Cidemog.State_City
```

You can add other statistical operators to this Select statement to retrieve a complete set of statistical values.

The Add Database Table dialog lets you type in the select, from and where text boxes to allow you to create such tables. In this case, the Select clause in the statement specifies calculations used by the database to create the values that appear in the table. Since neither the Cities table nor the Cidemog table have a column that holds a unique value for a record of these calculated values, a forward only scrolling database table is created.

If you don't have a lot of database experience and you're going to do a lot of work with database tables, you should take the time to learn SQL. The online Help topic, "Tips for

creating SQL Where clauses,” will help you get started. You can find this topic by opening the Help, clicking the Index tab, then typing the topic’s name into the field at the top of the dialog.

Optimizing your queries

Tables in your relational database can contain millions of records. Even simple queries against large tables can take time. Try to optimize your queries to get the best performance possible.

Index the columns used in join statements and attribute criteria in the Where clause to improve your query’s performance; you can do it with a script. Choose the SQL operators for the attribute criteria carefully. Some SQL operators are faster than others; some don’t use indexes even if they exist. In a join statement, placing a table on the left or right side of the equals sign can impact the query’s performance. These are a few considerations. You should carefully read the documentation for your relational database to find out how to optimize your attribute queries.

The syntax of the queries in this book and in the Help topics are correct for Oracle databases only. Each relational database requires its queries to be a little bit different. For example, some databases are case sensitive for table and column names, while others aren’t. If you don’t know the characteristics and syntax rules of your database, talk to your database administrator and get documentation for the relational database.

You have now narrowed the search to 47 cities. To make it easier to analyze the remaining markets, you can modify the SQL statement for the Potential Cities Query table so that the records are always sorted in a relevant order.

To change a database table’s definition query

1. Make the Potential Cities Query database table active.
2. From the Table menu, choose Properties. The Database Table Property dialog appears, containing the SQL Select statement that was sent to the relational database to create the database table.
3. Add “Order by State_Name, Renter_occ” in the SQL Statement box, after the existing statement.
4. Replace “World.Cidemog.*, World.US_Cities .*” with “Cities.State_Name, Cities.City_Name, Cidemog.Renter_occ, Cidemog.Owner_occ, Cidemog.Median_Val, Cidemog.Hse_Units, Cidemog.Vacant”. You can enter the column names by typing them out.

Database Table Property

Table Name: Potential Cities Query Database Access: ODBC

Connection:

Create a new connection

Use an existing connection

Demographic Database::MYUSER

Create an SQL statement:

Owner: *

Tables:	Columns:
WORLD.US_CITIES	<All Columns>
WORLD.US_COUNTIES	CITY_FIPS
WORLD.US_STATES	CITY_NAME
WORLD.WORLD3D	STATE_FIPS
WORLD.ZLINE	STATE_NAME
WORLD.ZPOINT	STATE_CITY

Unique Column Table: "WORLD"."US_CITIES"

Unique Column: "STATE_CITY"

SQL Statement:

```
SELECT STATE_NAME, CITY_NAME, RENTER_OCC, OWNER_OCC,
MEDIAN_VAL, HSE_UNITS, VACANT FROM "WORLD"."CIDEMOG",
"WORLD"."US_CITIES" WHERE "WORLD"."CIDEMOG"."STATE_CITY" =
"WORLD"."US_CITIES"."STATE_CITY" and
"WORLD"."CIDEMOG"."RENTER_OCC" > 63685 ORDER BY STATE_NAME,
RENTER_OCC
```

Comments:

OK Cancel

- Click OK. The contents of the database table changes to reflect the new SQL query.

The table now shows the remaining 47 cities in order, by state name and number of renter-occupied dwellings. Also, the irrelevant columns have been removed, leaving mostly those columns that hold information specific to housing.

Quote characters in SQL statements

When building or modifying SQL statements, quote characters are put around table and column names that are added, using the table and column lists. These characters are required by some ODBC drivers in order to make the SQL statement easier to parse. The character that is used is driver specific. For example, the Oracle driver that is being used in the exercises requires a “ character, while the Access 97 driver requires a ` character.

For most drivers, the quotes are not required unless the table or column name contains characters that have a special meaning. For example, if a table in Access has a space in its name, you must quote the table name or there will be an error when you use the table in a query. To be safe, database access quotes all column and table names that are added using the table and column lists.

When you modified the SQL statement of the Potential Cities Query table, you did not need to use quote characters because the column names did not contain special characters.

This is the end of the tutorial. You’ve made a good start on a list of cities to start building projects. You could use this table as a base as you come up with more conditions to narrow your search.

To save your work

1. Close all of your open tables.
2. With the Project window active, from the File menu, choose Save Project.

Using Avenue

There are several other types of operations that you can perform using Avenue scripts. To learn how, you first need to understand which objects you need to use and how they work. The following chapters explain the Database Access objects and will get you started writing Avenue scripts for your ODBC data. However, if you’ve never used Avenue before, you should review the *Using Avenue* book before moving on to Chapter 4.

CHAPTER 4

Understanding the Database Access objects

Avenue scripts can change the way you work with relational database data in ArcView. With Avenue you can assign spatial criteria to database themes and load new attribute tables into the database.

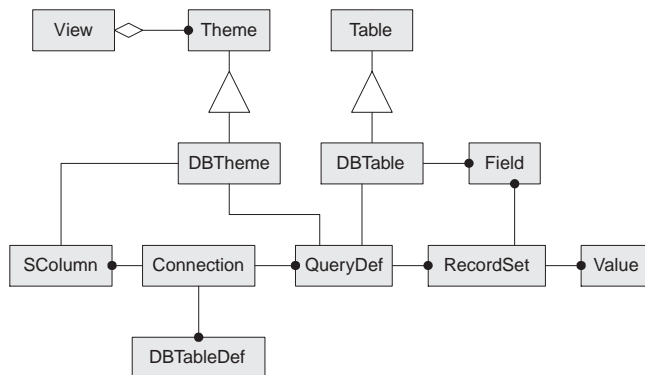
To write Avenue scripts, you need to know which objects together make database themes and database tables work. The extension has a set of common objects that define the general methods for working with databases and a set of objects that define specific methods for each database interface (e.g., SDE).

If you've never used Avenue before, you should review the *Using Avenue* book before reading this chapter.

Read this chapter to learn about:

- The common Database Access objects.
- The SDE interface objects.
- The ODBC interface objects.

The common Database Access objects



Database Access has a set of common objects defining general methods for working with relational database data: DBTheme, DBTable, Connection, QueryDef, RecordSet, DBTableDef, and SColumn. The object model diagram above shows how they relate to each other and to ArcView's core objects: View, Theme, Table, and Field. Each object is associated with a class of requests that it understands.

To access data, ArcView needs an active connection (Connection) to the relational database. An ArcView project can have many connections to different databases. From a connection you can create a query definition (QueryDef) to choose the rows and columns you want to use in ArcView.

Database themes (DBTheme) and database tables (DBTable) are different user interfaces for a query definition. A database table shows you the nonspatial values in the rows retrieved by a query. A database theme in a map (View) shows you the value in its spatial column (SColumn) for the rows retrieved by a query. Database tables have a Field for each column retrieved by the query; database themes do not. ArcView accesses a DBTheme's nonspatial columns, which are attributes of the spatial features, using internal queries.

From a QueryDef, you can create a RecordSet using Avenue to directly access the values in each row. A RecordSet has a Field object for each column retrieved by the query, and one value object of the appropriate data type for each Field: a Shape, a String, a Number, a Date, or a FileName (for BLOB fields).

Immediately after creating a RecordSet, using the GetRow request will return a list of null values. Use the Next request to retrieve the first row selected by the query from the database; the first row becomes the current record. After using Next, the RecordSet's value objects automatically contain the current record's actual values.

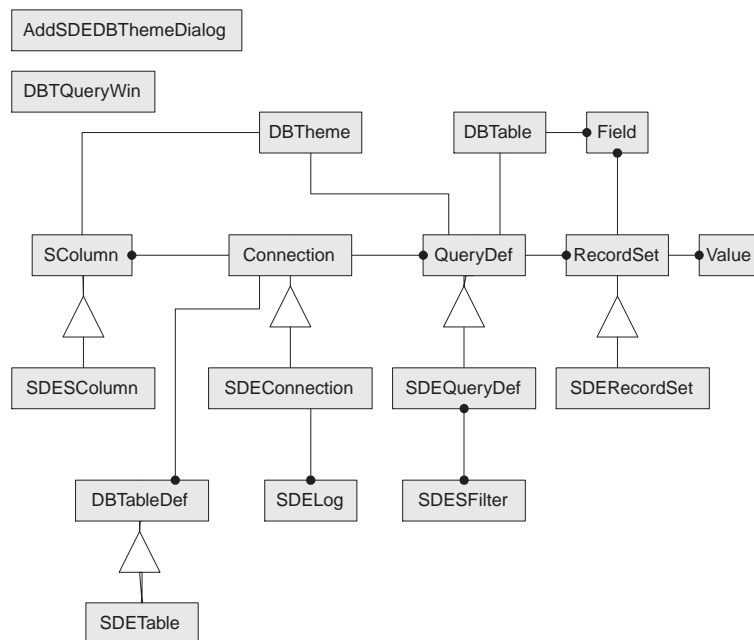
When you're finished with the current record, use the Next request again. ArcView discards the first row, then retrieves the second row from the database; the second row is now the current record, and the value objects automatically contain its values. To go back to the first record, you need to close this RecordSet and open another.

RecordSets can also be opened from a DBTableDef, which represents a table in the relational database. These RecordSets will retrieve values from all columns in the table. A database interface may let you create new tables in the relational database with its subclass of DBTableDef.

ArcView communicates with relational databases through a database interface, for example, SDE's C language interface. Each interface has subclasses of the common classes discussed above, which are tailored to the functionality supported by the interface. Additional classes specific to the interface may be required. Similarly, if the interface doesn't support spatial data, it won't have a subclass for SColumn, and you won't be able to create DBThemes.

For example, the QueryDef class defines methods for querying databases, like creating standard SQL statements with the SetSQL request. RecordSets can be created from SQL Select statements. The SDEQueryDef class inherits this method of using SQL statements, but SQL Select statements won't retrieve shapes from an SDE database. To retrieve shapes and create DBThemes, the SDEQueryDef must perform SDE specific operations.

The SDE interface objects



The objects for the SDE database interface include the subclasses **SDEConnection**, **SDEQueryDef**, **SDERecordSet**, and **SDESColumn**, and three additional classes, **SDEFilter**, **SDETable**, and **SDELog**. The new classes are required for supporting database theme functionality and additional functionality available from the SDE interface.

Creating an **SDEConnection** requires an SDE server name, an SDE database instance name, a data reference name, and your user name and password for accessing the data. Provide the server and instance names together in the format `<server>:<instance>`. You only need to provide the server name if the instance has the default name, `esri_sde`, or if you set the `SDEINSTANCE` environment variable with the instance name.

If the relational database doesn't contain databases of associated tables (e.g., Oracle or DB2), provide an empty string, `""`, for the data reference name. For relational databases like SYBASE, which have databases of tables, the data reference name is the database name whose tables you want to access. For SQL Server, the data reference name is a Data Source Name. The SDE administrator sets up data source names on the server. Each data source has a default database, whose tables you can access after connecting.

For SYBASE and SQL Server, you can also connect to SDE with an empty string for the data reference name; before creating objects, assign a database name to the new

connection with the SetDatabase request. A connection provides access to one database of tables only; that is, you can only set a connection's database once (when you first create the connection or when you use SetDatabase).

An SDEQueryDef that retrieves shapes is similar to an SQL Select statement. The SetSelectColumns request defines the columns you want to retrieve, the SetFromTables request defines the tables containing those columns, and the SetWhereClause request defines how the tables are joined together and also defines attribute criteria to select the rows you want to work with. These spatial query requests can't be used in combination with the SetSQL request.

When you create or modify a database theme from the user interface, ArcView creates a spatial query with the SetSelectColumns, SetFromTables, and SetWhereClause requests. When you create or modify a database table, ArcView creates a nonspatial query with the SetSQL request. When creating a DBTable from a DBTheme by clicking the OpenThemeTable button, ArcView copies the theme's spatial query. When this is done, a special relationship is established between the DBTable and the DBTheme where changing the selection on one will change the selection on the other. To maintain this relationship, table properties does not allow the DBTable's query to be changed and you can create only one DBTable per DBTheme. If you change the DBTable's query with Avenue, the relationship is maintained, but features in the DBTheme may no longer correspond to features in the DBTable.

Using the SetSFilters request, you can assign spatial filters (SDEFilter) to a query that defines spatial criteria to select the rows you want to work with. A spatial filter compares all the shapes in a spatial column (SDESColumn) to a filter shape. The query selects the rows whose shapes have the desired relationship with the filter shape.

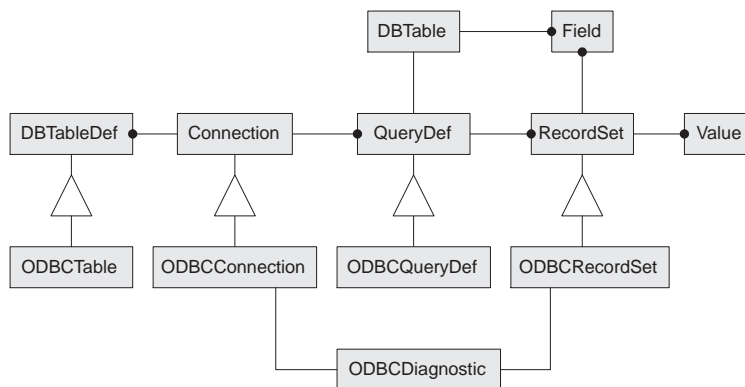
There are three distinct ways of creating a RecordSet using the SDE interface: by sending the OpenRecordSet request to a spatial SDEQueryDef, a nonspatial SDEQueryDef, and an SDETable. The distinction is important because you can only add, edit, or delete rows if the SDERecordSet wasn't created from a nonspatial query and if its columns come from only one table. You can find out if you can edit an SDERecordSet with the CanUpdate request.

You can create queries with the SetSelectColumns, SetFromTables, and SetWhereClause requests that select attribute values only. These are spatial queries because of their structure; it doesn't matter that they don't select shapes. You can edit attribute data in the database with SDERecordSets created from queries like this.

An SDETable can represent any table in the database you're connected to. SDETables let you add a spatial column to, and index nonspatial columns in, the source table. SDERecordSets created from an SDETable represent all columns in the table; a Where clause defines which rows are represented. You can retrieve and modify the spatial and nonspatial data in the table with these SDERecordSets.

An SDELog represents a logfile, which is a file on the SDE server that records a list of shapes in a spatial column. You can use an SDELog to record which shapes satisfy a query by setting the query's destination to be an SDELog object. Similarly, if you have an SDELog, you can retrieve the rows containing its shapes by setting the query's source to the SDELog. You change a spatial query's source and destination using the SetSrcDest request. ArcView uses logfiles to make selections work for database themes.

The ODBC interface objects



The objects for the ODBC database interface include the subclasses ODBCConnection, ODBCQueryDef, ODBCRecordSet, ODBCTable, and an additional class ODBCDiagnostic. The new class is provided to support error reporting from ODBC.

An ODBCConnection defines the communication layer between ArcView and the database. In order to use the ODBC database interface you must be using a Windows version of ArcView and have ODBC installed with the appropriate database drivers.

If you use the user interface to make a connection, you must first create an ODBC data source using the ODBC Data Source Administrator. An ODBC data source includes the information needed by the driver to connect to your database.

If you use Avenue to connect to the database, you can choose to provide all of the needed connection information with the request or to use an already existing data source. There are three connection requests available, each allowing you to provide the connection information in a different way.

The Make request takes a data source name, the name of the database, the user name and the password. Depending on the database, the database name, user name and password arguments may not be applicable. For SYBASE and SQL Server, data is organized in databases so the database name argument is relevant. There is no corresponding concept

in an Oracle database so the argument is of not relevant. When an argument is not applicable or is not required, provide an empty string "".

The `MakeWithConnStr` request takes a formatted ODBC connection string. You can format the string to use either an existing machine data source, an existing file data source or provide all of the connection information without a data source. Refer to you ODBC driver documentation to find out how to format this string.

When the `MakeWithDialog` request is used, an ODBC connection dialog appears. This dialog is created by the driver and allows you to enter all the information needed to connect.

An `ODBCQueryDef` contains an SQL statement that can be executed on the database. You can use SQL statements that do not return data, like create table or create index, or you can use a select statement to query tables in the database. `ODBCQueryDef`'s defined with select statements can be used to create database tables. The SQL syntax may vary slightly depending on the driver and the database.

You can assign a SQL statement to an `ODBCQueryDef` using the `SetSelectColumns`, `SetFromTables` and `SetWhereClause` requests or the `SetSQL` request. You cannot use the `SetSQL` request in combination to the other three requests.

An `ODBCTable` can be created to represent any single table in the database. You can retrieve the data in the table by creating an `ODBCRecordSet`. With an `ODBCTable`, you can provide a where clause to restrict the rows that are returned by the `ODBCRecordSet`.

An `ODBCRecordSet` can be created by using an `OpenRecordSet` request on either an `ODBCQueryDef` or an `ODBCTable`. An `ODBCRecordSet` is used to read data or add, edit and delete rows from a table in the database. You can determine if you have editing privileges by using the `ODBCRecordSet`'s `CanUpdate` request. In order for an `ODBCQueryDef`'s `RecordSet` to be editable, it must also have been created using the `SetSelectColumns`, `SetFromTables` and `SetWhereClause` requests and must have only one table in the `FromTables` list. An `ODBCTable`'s `RecordSet` will be editable as long as you have the proper permissions.

The `ODBCDiagnostic` class is used to retrieve the ODBC error and warning messages for a given transaction. An `ODBCDiagnostic` object can be created for either an `ODBCConnection` or an `ODBCRecordSet`. ODBC typically returns several messages for a transaction, even if it is successful. The `ODBCDiagnosticRecord` messages contain a SQL status message, the ODBC SQL state and native RDBMS Error Codes. The most severe error is always at the top of the list of messages. SQL status, SQL state and RDBMS error codes vary from database vendor to database vendor so you need to consult the RDBMS and ODBC driver documentation to interpret the information.

CHAPTER 5

Using Avenue to manipulate SDE objects

You can write Avenue scripts to customize the way you create database themes and database tables; to analyze the data they contain; and to add, edit, or delete the source data in the database. With Avenue, you have greater control over the Database Access objects and therefore the results. This chapter assumes that you have already worked through Chapters 2 and 4 in this book and that you know how to create and use Avenue scripts.

In this chapter, you'll learn how to use Avenue scripts to:

- Add a database theme to a view.
- Step through a database theme's records.
- Select features in a database theme.
- Work with selected features.
- Create and work with database tables.
- Work with selected records in database tables.
- Modify the source data.
- Use locks and transactions.
- Check for errors.
- Create new tables and spatial columns.

Adding a database theme to a view

When you create a database theme using the SQL Query Wizard, you create a connection to the SDE database, choose a spatial column, then define a query to choose the shapes and attributes the theme will represent. To create a database theme using an Avenue script, you go through the same steps. The following script creates a database theme and adds it to a view:

```
theView = av.GetActiveDoc

'make a new connection to your database
theCon = SDEConnection.Make("jazz","","","myUser","myPassword")

'make a query definition
theQuery = SDEQueryDef.Make(theCon)
theQuery.SetSelectColumns({"Shape","Name"})
theQuery.SetFromTables({"Countries"})

'make a spatial column
theSCol = SDESColumn.Make(theCon,"Countries","Shape")

'put the pieces together to make a DBTheme
theDBTheme = DBTheme.Make(theQuery, theSCol, polygon)

'add the theme to the view
theDBTheme.SetName("Countries")
theDBTheme.SetVisible(true)
theView.AddTheme(theDBTheme)
```

To create an SDEConnection you provide the name of the SDE server and database instance, a data reference name (an empty string for Oracle and DB2, a database name for Sybase, or a data source name for SQL Server), your user name, and your password. In the above example, the computer with the SDE server is called jazz. The SDE database instance containing the table you want to use, Countries, has the default name, esri_sde.

Next, the script creates an SDEQueryDef that selects values in the Shape and Name columns from all the rows in the Countries table. Then, it creates an SDESColumn to tell ArcView which column selected by the query contains the shapes that the database theme will draw in the view. Finally, it creates a polygon DBTheme to draw the polygon shapes in the spatial column, called Shape.

You can also use the SQL QueryWizard to change the database theme's query. The following script shows how a theme's query can be modified with an Avenue script.

```
theView = av.GetActiveDoc

'get the theme's query definition
theDBTheme = theView.FindTheme("Countries")
theQuery = theDBTheme.GetQueryDef
```

```

‘change the query definition
theQuery.SetSelectColumns({"Countries.Shape", "Demog.Proj_pop2000", "Economy.Gnp"})
theQuery.SetFromTables({"Countries", "Demog", "Economy"})
theQuery.SetWhereClause("Countries.Name = Demog.Country and
    Countries.Name = Economy.Country and Demog.Grw_rate > 1")

‘redraw the view
theDBTheme.InvalidDate(false)
theView.GetDisplay.Flush

```

To change a theme’s query you need to get the DBTheme object from the view, then get its SDEQueryDef object. Using the SetSelectColumns, SetFromTables, and SetWhereClause requests you change the query’s properties and therefore the values the theme represents. The script above selects the attributes Proj_pop2000 from the Demog table and Gnp from the Economy table for the Countries theme. The SetWhereClause request joins the three tables together and redefines the theme to include only the countries whose population growth rate is greater than one.

A database theme’s query can also use spatial criteria to define which rows it represents. The following script creates a new SDEQueryDef using the theme’s existing connection, then assigns it to the theme. The new query has the Countries theme’s original attributes and spatial criteria.

```

theView = av.GetActiveDoc

‘get the theme’s connection
theDBTheme = theView.FindTheme("Countries")
theCon = theDBTheme.GetQueryDef.GetConnection
theSCol = theDBTheme.GetSColumn

‘make a new query definition
newQuery = SDEQueryDef.Make(theCon)
newQuery.SetSelectColumns({"Shape", "Name"})
newQuery.SetFromTables({"Countries"})

‘assign spatial constraints
aCircle = theView.GetGraphics.Get(0).GetShape
inside = #SDESPATIAL_SEARCH_ISWITHIN
shapesInCircle = SDEFilter.Make(theSCol, aCircle, inside)
poly = theView.GetGraphics.Get(1).GetShape
intersect = #SDESPATIAL_SEARCH_INTERSECTAREA
shapesNotInPoly = SDEFilter.Make(theSCol, poly, intersect)
shapesNotInPoly.SetTest(false)
newQuery.SetSFilters({shapesInCircle, shapesNotInPoly})

‘assign the new query to the database theme
theDBTheme.SetQueryDef(newQuery)

```

You create spatial criteria by creating an SDEFilter object. This script assumes the view contains two graphic shapes: a circle overlapping a polygon. You create one spatial filter that chooses the rows whose shapes fall inside the circle and another that chooses the rows whose shapes don’t intersect the polygon. The resulting theme contains the countries falling inside the part of the circle graphic that doesn’t overlap the polygon.

By default, a spatial filter will choose the shapes that respond positively to the spatial test. To find the shapes that don't intersect the polygon, or respond negatively to the INTERSECTAREA spatial test, you need to use the SetTest request to change the shapesNotInPoly spatial filter's test property. Assign the spatial filters to the SDEQueryDef with the SetSFilters request.

Stepping through a database theme's records

When ArcView draws a database theme, it opens a RecordSet that only retrieves the shape value and any attribute values required to classify the theme. For each row in the RecordSet, ArcView retrieves the values then draws the shape in the view with the appropriate symbol. You can work with a database theme's records individually with a script that opens a RecordSet from the theme's query, then loops through each record in the set with the following method:

```
theDBTheme = av.GetActiveDoc.GetThemes.Get(0)
theRecordSet = theDBTheme.GetQueryDef.OpenRecordSet
while (theRecordSet.Next)
  MsgBox.Info("Now do something...", "Got a record")
end
theRecordSet.Close
```

When you open a RecordSet, the current record is positioned above the first record. The first time through the loop the Next request positions the first record as the current record. Each subsequent pass through the while loop advances the current record to the next record in the set. You must close the RecordSet when the while loop finishes because SDE limits the number of open RecordSets you can have to eight; the SDE administrator can change that number.

You can perform tasks using the current record's shape and attribute values. The example below creates a list of all the countries in a database theme.

```
theView = av.getActiveDoc

'get the theme's query definition
theDBTheme = theView.FindTheme("Countries")
theQuery = theDBTheme.GetQueryDef

'open a record set and start looping
nameList = {}
theRSet = theQuery.OpenRecordSet
nameField = theRSet.FindField("Name")
theName = theRSet.GetValue(nameField)
while (theRSet.Next)
  nameList.Add(theName.Clone)
end
theRSet.Close

'show the results
theString = ""
```



```

for each n in nameList
  theString = theString+n+NL
end
msgbox.Report(theString, "All Countries in the Theme")

```

The RecordSet has a set of Field objects representing the columns selected by the query, and one value object of the appropriate data type for each field. GetRow returns a list containing the RecordSet's Value objects; each object's value represents one of the current record's values. GetValue returns the value object for one Field. You only need to get the RecordSet's value objects once. Each time through the loop, when the Next request changes the current record, the value of each Value object will automatically change.

ArcView recycles the value objects for each record in the RecordSet because if your query retrieved a large number of rows, and if new objects were created for each record's values, your computer could run out of memory. The script above clones the name in each row before adding it to the list to keep names from each selected record. Otherwise, the list would contain many string objects all pointing to the name value in the current record; each element in the name list would have the same value.

Field objects in ArcView have many properties, but changing them has no impact on the RecordSet's contents, which are selected by its QueryDef. The Fields merely let you access the values for a specific column. Similarly, you can't change a database theme by manipulating field objects in a RecordSet opened from the theme's query. The RecordSet has no direct relationship to the theme. Closing the RecordSet deletes its field and value objects.

Selecting features in a database theme

After a database theme is created, you can query its values using the DBTheme SelectBy requests. The following script selects a county from a Counties theme then all the highways in that county from a Highways theme, and reports how many highways were selected.

```

theView = av.GetActiveDoc
cntyTheme = theView.FindTheme("Counties")
hwyTheme = theView.FindTheme("Highways")
cntyTheme.SelectByAttributes("Name = 'Los Angeles'", #VTAB_SELTYPE_NEW)
hwyTheme.SelectByTheme(cntyTheme, #FTAB_RELTYPE_INTERSECTS, 0,
  #VTAB_SELTYPE_NEW)
numHwys = hwyTheme.GetSelection.Count
msgbox.Info(numHwys.AsString, "Number of highways in L.A. County")

```

The shapes found with a SelectBy request are recorded in an SDE logfile. In the script above, the GetSelection request returns the SDELog for the theme's current selection.

To find the shapes in a theme where “Name = ‘Los Angeles’”, ArcView creates a new query combining the theme’s query and the attribute criteria from the `SelectByAttributes` request. You can write a script that produces the same results using an `SDEQueryDef` and an `SDELog`. The following script shows you how. It copies the theme’s query so the modifications don’t alter the theme’s contents. The results of the modified query are sent to a new logfile, which becomes the theme’s current selection.

```
'copy the theme's query
newQuery = cntyTheme.GetQueryDef.Clone

'make a new logfile
tableName = cntyTheme.GetSColumn.GetTableName
theCon = newQuery.GetConnection
newName = SDELog.MakeLogName(theCon)
newLog = SDELog.Make(theCon, newName, tableName, #SDELOG_MODE_CREATE)

'prepare the new query
theWhere = newQuery.GetWhereClause
if (theWhere <> "") then theWhere = theWhere++"and " end
newQuery.SetWhereClause(theWhere+"Name = 'Los Angeles'")
newQuery.SetSrcDest(nil, newLog, false)

'populate the new log file
theRecordSet = newQuery.OpenRecordSet
theRecordSet.Close
newLog.Close

'make the new log file the current selection
cntyTheme.SetSelection(newLog)
```

When creating a new logfile, the `MakeLogName` request guarantees it will have a unique name, which avoids naming conflicts with other users. The logfile can record any shapes in one spatial column; because SDE only allows one spatial column per table, you only need to specify which table in the database the shapes will come from.

A query’s results are sent to ArcView so you can draw a database theme or loop through the rows in a `RecordSet`. This script changes the destination of the query’s results from a `nil` (sends the results to ArcView) to the new `SDELog` with the `SetSrcDest` request. Now, for each row selected by the query, SDE records its shape in the logfile.

If the third parameter in the `SetSrcDest` request is true, SDE sends each row to both ArcView and the logfile; otherwise, SDE sends all rows directly to the logfile and sends nothing to ArcView. Because the third parameter is false, this script opens a `RecordSet` to populate the logfile, then closes it; you can’t loop through this `RecordSet`.

There are other `SelectBy` requests that take rectangles, points, or a list of shapes as spatial criteria. Commonly, the shapes for these requests are derived from the view, either from its graphics or from user interaction with it (dragging a rectangle or clicking a point). You could write a script similar to this one that creates a logfile from a combination of the theme’s query and spatial filters from the shapes in the view.

If you create spatial filters using shapes derived from a view, and if the view is projected, you need to unproject the shapes so they are in decimal degrees like the source data. You can ensure the unprojected shape is correct by adding it to the view as a graphic shape.

Working with selected features

When shapes are selected in a database theme, ArcView redraws them in yellow in the view. To do that ArcView needs to access the contents of the theme's selection logfile. You can write a script that accesses and works with the shapes in a database theme's selection logfile. The following script creates a buffer around selected highways.

```
theView = av.GetActiveDoc
hwyTheme = theView.FindTheme("Highways")
theSName = hwyTheme.GetSColumn.GetSColumnName

'change the query to retrieve selected records only
theLog = hwyTheme.GetSelection
theQuery = hwyTheme.GetQueryDef.Clone
theQuery.SetSrcDest(theLog, nil, false)
theRSet = theQuery.OpenRecordSet
theSField = theRSet.FindField(theSName)

'get the buffer for the first shape in the record set
theRSet.Next
theShape = theRSet.GetValue(theSField)
totalBuffer = theShape.ReturnBuffered(50)

'buffer rest of shapes and union with totalbuffer
while(theRSet.Next)
  theShape = theRSet.GetValue(theSField)
  buffer = theShape.ReturnBuffered(50)
  totalBuffer = totalBuffer.ReturnUnioned(buffer)
end
theRSet.Close

'show the buffer in the view as a graphic
theView.GetGraphics.Add(GraphicShape.Make(totalBuffer))
```

The distance you use to buffer the shapes must have the same units as the view's map units. If the shapes are stored in decimal degrees, and the view isn't projected, the distance must also be in decimal degrees. You can convert distances from meters to decimal degrees using the `Units.Convert` request.

Queries retrieve rows and columns from the tables joined by the query's Where clause. This script changes the source of the query from a nil (get data directly from the tables) to the theme's selection SDELog with the `SetSrcDest` request. The modified query retrieves the same columns, but it only retrieves rows whose shapes are recorded in the SDELog. Because the destination in the `SetSrcDest` request is nil, the query's results are returned to ArcView. The third parameter is irrelevant if the destination is nil.

The previous example shows how to use the SetSrcDest request with either the source or destination set. The script fragment below shows how you can create a query that has both a source and destination.

```
theQuery.SetSrcDest(theLog, newLog, true)
theRSet = theQuery.OpenRecordset
while (theRSet.Next)
    'buffer the shape and add graphic to the view
end
theRSet.Close
newLog.Close
```

A script like this would find all the rows with shapes in theLog that also satisfy theQuery. Each time you retrieve a row with the Next request, the row's shape will be recorded in newLog. You can buffer the shape in the current row before moving to the next one.

New logfiles created in a script or by ArcView for a selection are stored temporarily on the SDE server. They'll be deleted when the project is closed. ArcView uses temporary logfiles and doesn't save a database theme's selection in a project because logfiles aren't synchronized with the database. The shapes recorded can be deleted or changed so they no longer satisfy the query used to create the logfile.

If your database isn't going to change, you can make a logfile permanent using the ChangeType request. If your database is dynamic, you should consider saving the query rather than its results. If you save a query that has a source SDELog, make sure the logfile is permanent or is created from a script before the query is sent to the database.

Using SDELogs

You can create queries that join shapes to attributes in a one-to-many relationship. If you retrieve the contents of a logfile with a query like this, you need to be careful how you interpret the results. After joining tables together, many rows can contain the same shape. A logfile records the shape in a record, not the record itself. The query reading the logfile will retrieve the many records containing a shape. To retrieve one of those records, the query must include attribute criteria to select that record.

Creating and working with database tables

There are two ways to create a database table from the user interface. You can create a database table from scratch by choosing Add Database Table from the project menu, or you can create one containing a database theme's attributes by clicking the OpenThemeTable button.

When you create a database table from scratch you build an SQL Select statement. The script below creates a new connection and a query representing an SQL Select statement, then defines a database table.

```
theCon =
SDEConnection.Make("bigSleep:esri_sde3","business","bogart","myPassword")
theQuery = QueryDef.Make(theCon)
theQuery.SetSQL("Select customers.* From customers")
theQuery.SetKeyColumn("")
theQuery.SetKeyTable("")
theDBTable = DBTable.Make(theQuery)
theDBTable.SetName("Customers")
theDBTable.GetWin.Open
```

This script connects to a relational database that has databases of tables. The database containing the customers table is called business. The select statement chooses all the columns in the customers table using customers.*. The SetKeyColumn and SetKeyTable requests are used to specify a column from a table in the SQL Select statement to be used for creating a keyset. To create a forward only scrolling database table, the KeyTable and KeyColumn are set to null strings as shown above.

From the user interface you can change a database table's query using the Database Table Properties dialog. To see only the customers in the southwest region you would type "Where region = 'SW'" after the existing query, then click OK. The script below makes the same change to an existing database table.

```
theDBTable = av.FindDoc("Customers")
theQuery = theDBTable.GetQueryDef
theQuery.SetSQL("Select customers.* From customers Where region = 'SW'")
theDBTable.GetVTab.Refresh
```

If you want to select records in this table, you have to convert it to a keyset database table. From the interface you would do this using the Database Table Properties dialog. The script below shows how to accomplish the same task with Avenue. Once you run this script, the selection tools in the user interface will become enabled when the Customers table is active.

```
theDBTable = av.FindDoc("Customers")
theQuery = theDBTable.GetQueryDef
theQuery.SetKeyTable("Customers")
theQuery.SetKeyColumn("Cust_ID")
theDBTable.GetVTab.Refresh
```

In the previous scripts, we used the notation customers.* to select all columns in the customers table. This was used because keyset database tables require that you specify all columns using the tablename.* notation. If you are using forward only scrolling tables, the * notation will also work.

```
theDBTable = av.FindDoc("Customers")
theQuery = theDBTable.GetQueryDef
theQuery.SetSQL("Select * From customers Where region = 'SW'")
theDBTable.Refresh
```

Suppose the customers table contains a spatial column with points representing the customers' location and you have a database theme displaying those points. To create a database table to see the theme's attributes you need to use a script like this:

```
theView = av.FindDoc("Customer Locations")
theTheme = theView.FindTheme("Customers")
theQuery = theTheme.GetClonedQueryDef
theDBTable = DBTable.Make(theQuery)
theDBTable.GetWin.Open
```

In this script, the `GetClonedQueryDef` request creates a copy of the theme's querydef which is linked to the original querydef. A table created from the copied querydef will be a keyset database table, using a unique column maintained by SDE as the keycolumn. A selection made on the theme will also appear in the database table since the querydefs are linked. If you change the table's querydef, the link is still available, but a selection on a database theme may not appear in the database table.

Selecting columns from the database

While spatial queries are like SQL, there are several SQL standards that aren't supported with them. For example, you can't use an asterisk "*" with the `SetSelectColumns` request to choose all columns in the selected tables. Similarly, you can't assign aliases to columns and tables in the `SetSelectColumns` and `SetFromTables` requests, then use those aliases elsewhere in the statement. You can't retrieve shape values when you use `GroupBy` or `OrderBy` operators in the statement.

When you provide an alias for a column in a query, ArcView names the column with the alias you provide, but it doesn't keep its real name or the calculation. In a database table, you'll see the column of values, but you won't be able to get statistics for this column from the user interface because ArcView doesn't know how to get the original data in the column. In a database theme, you won't be able to retrieve values from the aliased column with `Identify`, and you won't be able to use the column in the legend editor.

If you define a column like `TotalSales*100` and export to a dBASE file or a shapefile, the column's name will not be accurately represented. In the database table a field representing the column above will have a field name and alias of `TotalSales*100` as expected. Field names in dBASE can't include characters like * so the corresponding column will be written to the dBASE file or the shapefile with the name `Total_100`. Also, since dBASE allows a maximum of 10 characters in a field name, any columns in the database table with more than 10 characters will be truncated when exported.

The query in the script above was taken from the database theme and therefore is a spatial query created with the SetSelectColumns, SetFromTables, and SetWhereClause requests. You can't use the SetSQL request in combination with any of the spatial query requests.

The following script shows how to see the attributes of just the features in the customer theme that appear in the view.

```
theView = av.FindDoc("Customer Locations")
theTheme = theView.FindTheme("Customers")
theScol = theTheme.GetSColumn
theQuery = theTheme.GetClonedQueryDef
theRect = theView.GetDisplay.ReturnVisExtent
thesFilt = SDEFilter.Make(theScol, theRect,
#SDESPATIAL_SEARCH_INTERSECTAREA)
theQuery.SetSFilters({thesFilt})
theDBTable = DBTable.Make(theQuery)
theDBTable.GetWin.Open
```

This script uses the SetSFilters request to limit the records that appear in the database table to just the ones that have features within the view's extent. If the layer is large, this significantly decreases the amount of time that it takes to retrieve the records from the database. Since the table's querydef is a copy, setting a spatial filter on the table does not effect the theme. When you pan and zoom, you must delete the database table and then re-run the script to display the appropriate attributes. Refer to the "Open Database Table of visible features" sample script in the on-line help to find out how to use this with your applications. To find the sample scripts, start the on-line help and look under the extensions book on the contents page for Database Access. The Database Access book lists the samples in a section called sample scripts and extensions.

If a database table's query selects a spatial column, the table won't contain shape values whether it has a spatial query or not. If you created the table from a database theme, ArcView puts the word shape in the column to indicate the row has a shape value. When you create a database table from scratch in the user interface or from an Avenue script, the column will contain the shapes' internal SDE identification number.

You can loop through the records selected by a database table's query the same way you loop through a database theme's records. The script below totals the sales to customers in the southwest region.

```
theDBTable = av.FindDoc("Customers")
theQuery = theDBTable.GetQueryDef
theRSet = theQuery.OpenRecordSet
theField = theRSet.FindField("Sales")
total = 0
while (theRSet.Next)
    total = total + theRSet.GetValue(theField)
end
theRSet.Close
msgbox.info(total.AsString, "Total Sales in the Southwest Region")
```

Instead of calculating the total sales using a RecordSet, you could create a new query with an SQL statement where the database calculates the total for you. The script below shows how.

```
theDBTable = av.FindDoc("Customers")
theCon = theDBTable.GetQueryDef.GetConnection
theQuery = SDEQueryDef.Make(theCon)
theQuery.SetSQL("Select SUM(Sales) From customers Where region = 'SW'")
theRSet = theQuery.OpenRecordSet
theRSet.Next
total = theRSet.GetRow.Get(0)
theRSet.Close
msgbox.info(total.AsString, "Total Sales in the Southwest Region")
```

The SUM function totals the values in the Sales column for each record selected by the Where clause. There are many SQL functions and operators that let you summarize the values in a table. The GroupBy operator lets you summarize the values in one column for each unique value in another column.

Working with selected records in Database Tables

When you create a keyset database table, you can select records in the table just like with regular tables. What is different, however, is the way that you access the selected set with Avenue. With a regular table, the values in the selection bitmap correspond to the records in the VTab. In a database table, the values in the bitmap refer to the rows in the table. To get the corresponding record in the database, you must use the ConvertRowtoRecord request. This request returns the row's keyset value, which can be used as a unique identifier in the database.

The sample script titled "Deleting selected records in a database table" shows how to access the records in a database based on the selected rows in a database table.

Modifying the source data

To modify the source data you need to have the appropriate permissions. Insert permission is needed to add records to a table, update permission to change records in a table, and delete permission to remove records from a table. The database administrator grants permissions for each user; therefore, your connection determines whether or not you can modify the database.

If you create a query using an SQL Select statement, you can loop through the selected records with a RecordSet, but you can't use RecordSet requests to modify the source data. The RecordSet.CanUpdate request will return false. Instead, you need to create another query that has an SQL Insert, Update, or Delete statement. The script below shows how.


```

theQuery = SDEQueryDef.Make(theCon)
theQuery.SetSQL("Update Streets Set Name = 'Highway 50' Where Name =
'Albion Road'")
theQuery.Execute

```

This script changes a street's name from Albion Road to Highway 50. Use the Execute request to send the database an SQL statement that doesn't retrieve records.

With a spatial query you can edit the source data using RecordSet requests if the query selects columns from one table only. If the query joins two tables together, the CanUpdate request will return false. The following sections illustrate how to modify the source data using RecordSet requests.

Adding records to the database

Suppose you have a table containing highway information. There is a proposed construction plan to add a new highway, number 55. The new highway shapes have been outlined in an ArcView shapefile.

To load shapes into the database you need to use the AddNew, Update, and EndUpdate requests. The AddNew request prepares the RecordSet for creating new rows in the database. The Update request sends a value for each of the RecordSet's fields to the database and places them in a new record in the table. Use the EndUpdate request after all new rows have been sent to the database; it completes the AddNew request and ensures that SDE has sent all new records to the relational database.

```

'get field objects from the new highway shapefile's FTab
newHwyTheme = theView.FindTheme("New Highways")
newHwyFTab = newHwyTheme.GetFTab
shapeField = newHwyFTab.FindField("Shape")
keyField = newHwyFTab.FindField("Key")

'define the query
theQuery = QueryDef.Make(theCon)
theQuery.SetSelectColumns({"Shape","Key","Name"})
theQuery.SetFromTables({"Highways"})

'prepare RecordSet for adding records, loop through shapefile's records
theRSet = theQuery.OpenRecordSet
if (theRSet.CanUpdate) then
  theRSet.AddNew
  for each rec in newHwyFTab
    'set the new record's values, then send it to the database
    theShape = newHwyFTab.ReturnValue(shapeField, rec)
    theKey = newHwyFTab.ReturnValue(keyField, rec)
    theRSet.SetRow({theShape, theKey, "Highway 55"})
    theRSet.Update
  end
  theRSet.EndUpdate
end
theRSet.Close

```

The view contains a theme representing the new highway's shapefile. This script gets that theme and its FTab, then loops through the FTab's records. The SetRow request assigns values to the value objects for each field in the RecordSet. The order of values in the SetRow request must be the same as the columns in the SetSelectColumns request. If the RecordSet has additional fields that aren't in the FTab, use null objects for the appropriate data types as placeholders in the list of values.

Instead of the SetRow request, you can assign values to the value objects with the SetValue request by changing the script as follows.

```
if (theRSet.CanUpdate) then
  nField = theRSet.FindField("Name")
  sField = theRSet.FindField("Shape")
  kField = theRSet.FindField("Key")
  theRSet.AddNew
  theRSet.SetValue(nField, "Highway 55")
  for each rec in newHwyFTab
    theShape = newHwyFTab.ReturnValue(shapeField, rec)
    theRSet.SetValue(sField, theShape)
    theKey = newHwyFTab.ReturnValue(keyField, rec)
    theRSet.SetValue(kField, theKey)
  theRSet.Update
end
theRSet.EndUpdate
end
```

This version of the script sets the RecordSet's name field to the value Highway 55 once. Each time through the "for each" loop the script changes the value in the Shape and Key columns. Update creates a new record containing the current value of the Shape and Key columns and the constant Name value. If the Highways table has other columns not represented in the RecordSet, no values would be stored in the database in those columns.

Editing records in the database

Suppose that later you want to edit the contents of the Highways table to add the description "Proposed" for the new Highway 55 shapes.

To change records in the database you need to use the Edit, Update, and EndUpdate requests. The Edit request prepares the RecordSet to change the records identified by its Where clause. The Update request replaces values in each identified record with the current value for each field in the RecordSet. Use the EndUpdate request after all changes have been made; it completes the editing process and ensures that SDE has sent all changes to the database.

There are two ways to edit a table's contents. You can loop through a table's records to check for errors in the data, or you might know in advance that you want to change specific records in the table. The following script shows how to edit the current record

while looping through a RecordSet. From an SDETable representing the Highways table, the script opens a RecordSet that will have fields for all columns in the table.

```
'open the record set and get the construction field
theTable = SDETable.Make(theCon, "Highways")
theRSet = theTable.OpenRecordSet("")
if (theRSet.CanUpdate) then
  theFields = theRSet.GetFields
  nameIndex = theFields.Find(theRSet.FindField("Name"))
  keyIndex = theFields.Find(theRSet.FindField("Key"))
  descField = theRSet.FindField("Description")
  while (theRSet.Next)
    'get current record's values, edit if it's a Highway 55 record
    theRow = theRSet.GetRow
    hwyName = theRow.Get(nameIndex)
    if (hwyName = "Highway 55") then
      'identify which highway 55 record you're editing with the key
      keyValue = theRow.Get(keyIndex)
      theRSet.Edit("Key = '"+keyValue+"'")      ' keyValue is a String
      theRSet.SetValue(descField, "Proposed")
      theRSet.Update
    end
  end
  theRSet.EndUpdate
end
theRSet.Close
```

This script edits the current record if it has a Highway 55 shape. To edit the current record, you need to construct a Where clause that uniquely identifies it. The Key value used here has a unique value for each record in the table.

Here the RecordSet has fields for all columns in the table. You only want to change one column's value, but you need to read values in the Name and Key columns as well to edit the appropriate records. GetRow retrieves the current record's values. SetValue changes Description field's original value to the new value, "Proposed".

The Update request changes the value in the database for each column represented in the RecordSet. However, only the value in the Description column will be different afterwards. If any of the current record's values are set to null values, the Update request will replace the original values in the database with null values. It is possible to replace the shape in the spatial column with a null value as well.

This script only requires that the Name, Key, and Description columns to be represented in the RecordSet. To avoid writing values to the database that don't need to be changed, create a query with the SetSelectColumns, SetFromTables, and SetWhereClause requests that chooses the Name, Key, and Description columns only.

The following script illustrates how to edit records without looping through a RecordSet. Use this method if you know in advance that you want to change the Description value

for all the records named Highway 55. Because you want to make the same change for each record, you can change them all in one step.

```
'define the query
theQuery = QueryDef.Make(theCon)
theQuery.SetSelectColumns({"Description"})
theQuery.SetFromTables({"Highways"})

'open the record set and change the description field for
'all Highway 55 records in the table
theRSet = theQuery.OpenRecordSet
if (theRSet.CanUpdate) then
    descField = theRSet.FindField("Description")
    theRSet.Edit("Name = 'Highway 55'")
    theRSet.SetValue(descField, "Proposed")
    theRSet.Update
    theRSet.EndUpdate
end
theRSet.Close
```

This script changes the Description value for all records in the Highways table whose shapes are named Highway 55. Because you don't need to read any values from the database to edit those records, you don't need to use the Next request.

It is essential that the QueryDef select only the columns whose values will be changed. If this query selected additional columns from the table but didn't set their values, the current record's Value objects would all contain null values except for the Description value. That's because this script doesn't use the Next request. On using Update, the original values for all columns in the RecordSet are replaced; the additional columns would no longer have any values in the records selected by the Edit request.

The above example assumes that the only Highway 55 records in the Highways table are those associated with the proposed highway. Suppose the Highways table contains roads for California and Arizona, that California has a Highway 55, Arizona doesn't, and the proposed Highway 55 is in Arizona. Both example scripts in this section would have to be modified to change only the Highway 55 records in Arizona.

The first script would have to read values from the State column as well to ensure the current record's Highway 55 shape was in Arizona. Its Where clause for the Edit request is still correct because the Key value is unique for each record. The second script's Where clause for the Edit request would have to change to choose the Highway 55 records from Arizona only; for example, "Name = 'Highway 55' and State = 'Arizona'". The Edit request's Where clause needs to include the State = 'Arizona' criterion even if the query's Where clause selected highways from Arizona only.

Deleting records in the database

Suppose Arizona's proposed Highway 55 project is cancelled. The following script shows how you can delete records associated with this project from the Highways table.

```
theQuery = QueryDef.Make(theCon)
theQuery.SetSelectColumns({"Description"})
theQuery.SetFromTables({"Highways"})
theRSet = theQuery.OpenRecordSet
if (theRSet.CanUpdate) then
    theRSet.Delete("Name = 'Highway 55' and State = 'Arizona'")
end
theRSet.Close
```

The Delete request removes the records identified by its Where clause from the database. You don't use the Update or EndUpdate requests with the Delete request.

Using locks and transactions

When you modify the source data in a database, it's important to use locks and transactions. Locks prevent two people from editing the same records at the same time. Transactions let you control when and if your changes become permanent in the database. Using locks and transactions together lets you maintain the integrity of a dynamic database in a multiuser environment.

You can lock records in tables with a spatial column using SDESColumn requests. To lock records in other tables, you need to use methods provided by the relational database. Use the Execute request to send the appropriate SQL statement to the database. The script below provides an example of how to use locks and transactions when you add a new record to the database.

```
theDBTheme = theView.FindTheme("Highways")
theSColumn = theDBTheme.GetSColumn
theCon = theDBTheme.GetQuerydef.GetConnection
theArea = theView.GetGraphics.Get(0).GetShape

'lock the area under the rectangle, then start the transaction
locked = theSColumn.LockArea(theArea, true)
if (not locked) then exit end
theCon.BeginTrans

'add a record to the database
aShape = PolyLine.Make({{-54@48, -57@49}})
theKey = 9
theRSet = theDBTheme.GetQueryDef.Clone.OpenRecordSet
theRSet.AddNew
theRSet.SetRow({aShape,theKey,"Highway 55"})
success = theRSet.Update
theRSet.EndUpdate
theRSet.Close
```

```
'close the transaction and unlock
if (success) then
  theCon.CommitTrans
else
  theCon.Rollback
end
theSColumn.FreeLock
```

This script attempts to lock the records in the area where the new shape will be added; if successful, the script starts a transaction. If the Update request is successful, the CommitTrans request makes your changes permanent by committing the transaction; otherwise, the Rollback request discards your changes. Other users can't retrieve the new record with their queries until your changes have been committed to the database.

LockArea lets you lock records whose shapes intersect a rectangle. The request's second parameter determines whether or not you can modify the locked records. If true, you can change them; if false, no one can. Use locks where no one can modify the data when the database is dynamic and you need to calculate values accurately. The FreeLock request removes your locks from a spatial column.

When adding many records to the database, use transactions that commit records at a regular interval. For example, you can commit new records to the database after every 50th Update request. If you can't load a record, when you roll back the transaction you'll only have to reload a maximum of 50 records. To set up this kind of a transaction, use the SetAutoCommitFreq request before starting the transaction. Use the CommitTrans request after loading all records to close the transaction.

Checking for errors

The examples above have included little error checking, focusing instead on illustrating functionality. However, it's essential to check for errors in your scripts.

When you create a new connection, you must check the Connection object's status before continuing with your script to make sure the database connection was successful. Check a connection like this:

```
theCon = SDEConnection.Make("bird:esri_sde30", "", "vince", "myPassword")
if (theCon.HasError) then
  msgbox.Error(theCon.GetErrorMsg, "Error Message")
  exit
end
```

If the connection wasn't successful, you can't continue executing the script. However, you can use the erroneous SDEConnection object to find out from SDE why your connection failed.

The `GetErrorCode` request returns an enumeration that briefly indicates the problem. The `GetErrorMsg` request returns a sentence rather than an enumeration. If the error is detected by the relational database rather than by SDE, the error code indicates that it's a database I/O (input/output) error; use the `GetDBErrorCode` and `GetDBErrorMsg` requests to locate the source of the problem.

After successfully creating a connection, you shouldn't have to check its error status again. Errors that occur when querying or changing data shouldn't impact your connection. However, it's possible for a serious database I/O error to confuse communication and prevent you from continuing with your work. Check the connection object if you receive an error like this.

Most requests on Database Access objects return a value indicating whether or not they were successful. If the remainder of your script is dependent on the outcome of a request, it's essential to check the outcome before your script continues. The transaction example checks the Boolean value returned by the `LockArea` request and stops the script if it wasn't successful. You can learn more about the problem by getting the error code from the connection before exiting.

Some requests return objects other than Boolean values; the `OpenRecordSet` request returns a `RecordSet` object. If a query is syntactically incorrect, the database can't understand it, and `OpenRecordSet` returns a nil object instead of a `RecordSet`. All of the scripts in this chapter working with `RecordSets` should include the following lines, but they didn't because they were focusing on functionality rather than error checking.

```
theRSet = theQuery.OpenRecordSet
if (theRSet = nil) then
  msgbox.Error(theCon.GetErrorCode.AsString, "Error Code")
  exit
end
```

There is no way of checking a query's validity in ArcView before you open a `RecordSet`. If you create a `DBTable` from an invalid `SDEQueryDef` you don't get a nil object. Instead, you need to check the `DBTable`'s error status before using it.

```
theDBTable = DBTable.Make(anSDEQueryDef)
if (theDBTable.HasError) then
  msgbox.Error(theDBTable.GetErrorMsg, "Error Message")
  exit
end
```

If the `DBTable` has an error, remove it from the project. If you create a `DBTheme` from an invalid `SDEQueryDef`, you'll see an error message in the user interface instead of retrieved data.

Always check the outcome of a request, and retrieve the error code from SDE if it failed. `GetErrorCode` returns the error generated by the last failed request; a successful request doesn't generate an error code indicating its success.

Creating new tables and spatial columns

You can create new tables and spatial columns in an SDE database using `SDETable`, `SDESColumn`, and `SDEConnection` requests. The following script creates a table that will hold attributes for oil wells.

```
theCon = SDEConnection.Make("blues", "", "Coltraine", "aPassword")
if (theCon.HasError) then
  msgbox.Error(theCon.GetErrorCode.AsString, "Can't Connect")
  exit
end
theCon.SetDatabase("aDatabaseName")
field1 = Field.Make("Key", #FIELD_CHAR, 15, 0)
field2 = Field.Make("Code", #FIELD_CHAR, 10, 0)
newTable = SDETable.MakeNew("wells", {field1, field2})
newTable.SetCreationKeyword("wells")
ok = theCon.CreateTable(newTable)
if (ok) then
  myTable = SDETable.Make(theCon, "wells")
else
  msgbox.Error(theCon.GetErrorCode.AsString, "Can't Create Table")
end
```

To create a new table in the database, you create an `SDETable` representing the table's definition, then add it to the database using the `SDEConnection.CreateTable` request.

The SDE keyword `wells`, assigned to the new table with the `SetCreationKeyword` request, refers to a set of parameters in SDE's `DBTune` file; these parameters can control how much space is allocated when the table is created and what disk space it resides on. If you don't assign an SDE keyword, SDE uses the default parameters to create your table. Talk to your SDE administrator about setting up keywords that will maintain efficiency in the database.

After adding the new table to the database, your script might try to add records or a spatial column to it. You won't be able to use the `newTable` object in the script above with the `OpenRecordSet` request or the `CreateSColumn` request because it isn't associated with an `SDEConnection`. You need to create another `SDETable`, `myTable`, with the `Make` request, and `myTable` with the `OpenRecordSet` request.

You can spatially enable the `wells` table so it can store well points in addition to well attributes. Any table in an SDE database can have one spatial column. The script below defines the new spatial column, then adds it to the `wells` table.

```
newSCol = SDESColumn.MakeNew("Geometry", {#SDESCOLUMN_TYPE_POINT})
newSCol.SetZ(true)
newSCol.SetNumShapes(5000000)
newSCol.SetAveragePoints(1)
newSCol.SetCreationKeyword("wells")
newSCol.SetCoordRefByRect(Rect.Make(-180@(-90), 360@180))
```



```
newSCol.SetGridSizes({20,0,0})
ok = myTable.CreateSColumn(newSCol)
if (ok) then
  mySCol = SDESColumn.Make(theCon, "wells", "Geometry")
else
  msgbox.Error(theCon.GetErrorCode.AsString, "Can't Create SColumn")
  exit
end
```

Like creating a new table, you define the properties of a new spatial column using an `SDESColumn` created with the `MakeNew` request; it isn't associated with a connection. The `SDETable.CreateSColumn` request adds a column to the wells table in the database for spatial data. To work with this spatial column in the rest of the script you create an `SDESColumn` with the `Make` request.

The `SDESColumn.MakeNew` request defines the spatial column's name and the kind of features it can contain. A spatial column doesn't have to be called `Shape` or `Feature`; it can be called anything appropriate that is a valid field name.

Using the `SetMeasured` request on a new spatial column lets you store two-dimensional measured shapes in it. A measured shape can have a measure value at each vertex; measures are useful for identifying road segments that are under construction, for example. Using the `SetZ` request on a new spatial column lets you store three-dimensional shapes in it; three-dimensional shapes can have measure values. A database theme created from a measured or three-dimensional spatial column will draw the same as themes from regular two-dimensional spatial columns.

When retrieving shapes with a `RecordSet` from measured or three-dimensional spatial columns, you get measured or three-dimensional `Shape` objects. For example, the shape retrieved from a regular two-dimensional point spatial column will be a `Point` object. The shape retrieved from a measured point spatial column will be a `PointM` object; the shape retrieved from a three-dimensional point spatial column will be a `PointZ` object. The `-M` and `-Z` shape classes have new requests for manipulating measured and three-dimensional shapes with Avenue.

When you create a new spatial column, SDE creates internal tables to maintain the spatial data; the `SetNumShapes` and `SetAveragePoints` requests are required for SDE to calculate how much disk space is required for its tables. You can specify a creation keyword to control other details with SDE's `DBTune` file.

The `SetCoordRefByRect` request defines a geographic boundary; all shapes in the spatial column must fall completely inside this rectangle. Shapes with vertices on or outside of the rectangle will be clipped or rejected by SDE. `ArcView` automatically enlarges your rectangle by one percent to try to avoid these data storage problems. The rectangle's coordinates must be in the same units as the shape's coordinates; this spatial column will store decimal degrees shapes.

SDE locates shapes in a spatial column using a spatial index grid. A spatial column can have up to three grids, which are nested in size. The SetGridSizes request defines their sizes. This script defines one index grid that uses twenty degree squares to locate shapes. The grid size must be appropriate for the area covered by the coordinate reference rectangle, the dispersion of shapes across it, and the size of the shapes.

CHAPTER 6

Using Avenue to manipulate ODBC objects

You can write Avenue scripts to customize the way you create database tables using ODBC; to analyze the data they contain; and to add, edit, or delete the source data in the database. With Avenue, you have greater control over the Database Access objects and therefore the results. This chapter assumes that you have already worked through Chapter 3 and read Chapter 4 in this book and that you know how to create and use Avenue scripts.

In this chapter, you'll learn how to use Avenue scripts to:

- Create and work with database tables.
- Step through a database table's records.
- Modify the source data.
- Use locks and transactions.
- Check for errors.
- Work with selected records.

Creating and working with database tables

You can create an ODBC database table from the user interface by choosing Add Database Table from the project menu. When you create a database table you build an SQL Select statement to define the contents of the table. The script below creates a new ODBC connection, defines an SQL Select statement, and then creates a database table.

```
theCon = ODBCConnection.Make("esri_odbc","business","Jdoe","myPsswd")
theQuery = QueryDef.Make(theCon)
theQuery.SetSQL("Select customers.* From Customers")
theQuery.SetKeyColumn("")
theQuery.SetKeyTable("")
theDBTable = DBTable.Make(theQuery)
theDBTable.SetName("Customers")
theDBTable.GetWin.Open
```

This script connects to a relational database that has databases of tables. The database containing the Customers table is called business. The select statement chooses all the columns in the Customers table using the asterisk "*" character. The SetKeyColumn and SetKeyTable requests are used to specify a column from a table in the SQL Select statement to be used for creating a keyset. To create a forward only scrolling database table, the KeyTable and KeyColumn are set to null strings as shown above.

From the user interface you can change a database table's query using the Database Table Properties dialog. To see only the customers in the southwest region you would type "Where region = 'SW'" after the existing query, then click OK. The script below makes the same change to an existing database table.

```
theDBTable = av.FindDoc("Customers")
theQuery = theDBTable.GetQueryDef
theQuery.SetSQL("Select customers.* From Customers Where region = 'SW'")
theDBTable.GetVTab.Refresh
```

If you want to select records in this table, you have to convert it to a keyset database table. From the interface you would do this using the Database Table Properties dialog. The script below shows how to accomplish the same task with Avenue. Once you run this script, the selection tools in the user interface will become enabled when the Customers table is active.

```
theDBTable = av.FindDoc("Customers")
theQuery = theDBTable.GetQueryDef
theQuery.SetKeyTable("Customers")
theQuery.SetKeyColumn("Cust_ID")
theDBTable.GetVTab.Refresh
```

In the previous scripts, we used the notation customers.* to select all columns in the Customers table. This was used because keyset database tables require that you specify all columns using the tablename.* notation. If you are using forward only scrolling tables, the * notation will also work.

Stepping through a database table's records

You can step through each record in a database table using an `ODBCRecordSet`. When you open a `RecordSet`, the current record is positioned above the first record. The first time through the loop the `Next` request positions the first record as the current record. Each subsequent pass through the while loop advances the current record to the next record in the set. You should close the `RecordSet` when you are finished accessing records so the memory can be freed for other use.

The following script shows how to read data from an `ODBCRecordSet` to create a list of customer names.

```
theDBTable = av.FindDoc("Customers")
theQuery = theDBTable.GetQueryDef
theRSet = theQuery.OpenRecordSet
theField = theRSet.FindField("customer")
theCustomers = {}
while (theRSet.Next)
    custName = theRSet.GetValue(theField).Clone
    theCustomers.Add(custName)
end
theRSet.Close
msgbox.listAsString(theCustomers, "Customer List", "Southwest
Region")
```

The `RecordSet` has a set of `Field` objects representing the columns selected by the query, and one value object of the appropriate data type for each field. `GetValue` returns a string containing the `RecordSet`'s Value object for the customer field; each object's value represents one of the current record's values. `GetValue` returns the value object for one `Field`. You only need to get the `RecordSet`'s value objects once. Each time through the loop, when the `Next` request changes the current record, the value of each Value object will automatically change.

`ArcView` recycles the value objects for each record in the `RecordSet` because if your query retrieved a large number of rows, and if new objects were created for each record's values, your computer could run out of memory. The script above clones the customer name before adding it to the list to keep names from each selected record. Otherwise, the list would contain many string objects all pointing to the name value in the current record; each element in the name list would have the same value.

`Field` objects in `ArcView` have many properties, but changing them has no impact on the `RecordSet`'s contents, which are selected by its `QueryDef`. The `Fields` merely let you access the values for a specific column. Closing the `RecordSet` deletes its field and value objects.

The script below shows how to use a recordset to total the sales to customers in the southwest region.

```

theDBTable = av.FindDoc("Customers")
theQuery = theDBTable.GetQueryDef
theRSet = theQuery.OpenRecordSet
theField = theRSet.FindField("Sales")
total = 0
while (theRSet.Next)
    total = total + theRSet.GetValue(theField)
end
theRSet.Close
msgbox.info(total.AsString, "Total Sales in the Southwest Region")

```

Instead of calculating the total sales using a RecordSet, you could create a new query with an SQL statement where the database calculates the total for you. The script below shows how.

```

theDBTable = av.FindDoc("Customers")
theCon = theDBTable.GetQueryDef.GetConnection
theQuery = ODBCQueryDef.Make(theCon)
theQuery.SetSQL("Select SUM(Sales) From customers Where region = 'SW'")
theRSet = theQuery.OpenRecordSet
theRSet.Next
total = theRSet.GetRow.Get(0)
theRSet.Close
msgbox.info(total.AsString, "Total Sales in the Southwest Region")

```

Selecting columns from the database

While queries that use SetSelectColumns, SetFromTables and SetWhereClause are like SQL, there are several SQL standards that aren't supported with them. For example, you can't use an asterisk "*" with the SetSelectColumns request to choose all columns in the selected tables. Similarly, you can't assign aliases to columns and tables in the SetSelectColumns and SetFromTables requests, then use those aliases elsewhere in the statement. You must use these queries, if you plan on editing the data in the database.

When you provide an alias for a column in a query, ArcView names the column with the alias you provide, but it doesn't keep its real name or the calculation. In a database table, you'll see the column of values, but you won't be able to get statistics for this column from the user interface because ArcView doesn't know how to get the original data in the column.

If you define a column like Total*100 and export the database table to a dBASE file, the column's name will not be accurately represented. In the database table, a field representing the column above will have a field name and alias of Total*100 as expected. Field names in dBASE can't include characters like * so the corresponding column will be written to the dBASE file with the name Total_100. Also, since dBASE allows a maximum of 10 characters in a field name, any columns in the database table with more than 10 characters will be truncated when exported.

In this script, a new ODBCQuerydef object was created from an existing connection, and an SQL statement which uses the SUM function is applied to it. The SUM function totals the values in the Sales column for each record selected by the Where clause. There are many SQL functions and operators that let you summarize the values in a table. The GroupBy operator lets you summarize the values in one column for each unique value in another column.

Modifying the source data

To modify the source data you need to have the appropriate permissions. Insert permission is needed to add records to a table, update permission to change records in a table, and delete permission to remove records from a table. The database administrator grants permissions for each user; therefore, your connection determines whether or not you can modify the database.

With a query you can edit the source data using RecordSet requests if the query selects columns from one table only. If the query joins two tables together, the CanUpdate request will return false. If you use a querydef object, you must use the SetSelectColumns, SetFromTables, and SetWhereClause requests to specify the table, fields, and rows that you wish to edit. A QueryDef object created by the Add Database Table dialog uses the SetSQL request to define the query instead of SetSelectColumns, SetFromTables, and SetWhereClause. For this reason, recordsets created from these querydefs can't be used to edit data. The following sections illustrate how to modify the source data using RecordSet requests.

Adding records to the database

Suppose you have a table containing highway information. There is a proposed construction plan to add a new highway, number 55.

To load new data into the database you need to use the AddNew, Update, and EndUpdate requests. The AddNew request prepares the RecordSet for creating new rows in the database. The Update request sends a value for each of the RecordSet's fields to the database and places them in a new record in the table. Use the EndUpdate request after all new rows have been sent to the database; it completes the AddNew request and ensures that ODBC has sent all new records to the database.

```
'define the query
theQuery = QueryDef.Make(theCon)
theQuery.SetSelectColumns({"KeyCol", "Name"})
theQuery.SetFromTables({"Highways"})
'prepare RecordSet for adding records
theRSet = theQuery.OpenRecordSet
if (theRSet.CanUpdate) then
    theRSet.AddNew
```

```
theRSet.SetRow({101, "Highway 55"})
theRSet.Update
theRSet.SetRow({102, "Albion Road"})
theRSet.Update
theRSet.EndUpdate
end
theRSet.Close
```

The SetRow request assigns values to the value objects for each field in the RecordSet. The order of values in the SetRow request must be the same as the columns in the SetSelectColumns request. If the RecordSet had other fields, you would use null objects for the appropriate data types as placeholders in the list of values. Instead of the SetRow request, you can assign values to the value objects with the SetValue request by changing the script as follows.

```
if (theRSet.CanUpdate) then
  nField = theRSet.FindField("Name")
  kField = theRSet.FindField("KeyCol")
  theRSet.AddNew
  theRSet.SetValue(nField, "Highway 55")
  for each rec in 103..106
    theKey = rec
    theRSet.SetValue(kField, theKey)
    theRSet.Update
  end
theRSet.EndUpdate
end
```

This version of the script sets the RecordSet's name field to the value Highway 55 once. Each time through the "for each" loop the script changes the value in the KeyCol column. The Update request creates a new record containing the current value of the KeyCol column and the constant Name value. If the Highways table has other fields, null values would be stored in the database for them.

Editing records in the database

Suppose that later you want to edit the contents of the Highways table to add the description "Proposed" for the new Highway 55 records.

To change records in the database you need to use the Edit, Update, and EndUpdate requests. The Edit request prepares the RecordSet to change the records identified by its Where clause. The Update request replaces values in each identified record with the current value for each field in the RecordSet. Use the EndUpdate request after all changes have been made; it completes the editing process and ensures that ODBC has sent all changes to the database.

There are two ways to edit a table's contents. You can loop through a table's records to check for errors in the data, or you can know in advance that you want to change specific records in the table. The following script shows how to edit the current record

while looping through a RecordSet. From an ODBCTable representing the Highways table, the script opens a RecordSet that will have fields for all columns in the table.

```
'open the recordset to update the Description field
theTable = ODBCTable.Make(theCon, "Highways")
theRSet = theTable.OpenRecordSet("")
if (theRSet.CanUpdate) then
    theFields = theRSet.GetFields
    nameIndex = theFields.Find(theRSet.FindField("Name"))
    keyIndex = theFields.Find(theRSet.FindField("KeyCol"))
    descField = theRSet.FindField("Description")
    theRow = theRSet.GetRow
    while (theRSet.Next)
        'get current record's values, edit if it's a Highway 55 record
        hwyName = theRow.Get(nameIndex)
        if (hwyName = "Highway 55") then
            'identify which highway 55 record you're editing with the key
            keyValue = theRow.Get(keyIndex)
            theRSet.Edit("KeyCol = "+keyValue.asstring)
            theRSet.SetValue(descField, "Proposed")
            theRSet.Update
        end
    end
    theRSet.EndUpdate
end
theRSet.Close
```

This script will edit the current record if it has a Highway 55 record. To edit the current record, the script needs to construct a Where clause that uniquely identifies it. This script uses the Key value in the Where clause because it's unique for each record in the table.

In this script, the RecordSet has fields for all columns in the table. You only want to change the value in one column, but you need to read values in other columns to find out which records you need to edit. The Next request sets the value object for each field in the list to the current record's original values. The SetValue request changes the value object for the Description field to the new value, Proposed. The Update request changes the values in all columns, but only the value in the Description column will be different afterwards.

This script only requires that the Name, Key, and Description columns be represented in the RecordSet. To avoid writing values to the database that don't need to be changed, create a query with the SetSelectColumns, SetFromTables, and SetWhereClause requests that chooses the Name, Key, and Description columns only.

The following script illustrates how you can edit records without looping through a RecordSet. You know in advance that you want to change the Description value for all the records named Highway 55. Because you want to make the same change for each of the Highway 55 records, you can change them all in one step.

```
'define the query
```

```
theQuery = QueryDef.Make(theCon)
theQuery.SetSelectColumns({"Description"})
theQuery.SetFromTables({"Highways"})
'open the record set and change the description field for
'all Highway 55 records in the table
theRSet = theQuery.OpenRecordSet
if (theRSet.CanUpdate) then
    descField = theRSet.FindField("Description")
    theRSet.Edit("Name = 'Highway 55'")
    theRSet.SetValue(descField, "Proposed")
    theRSet.Update
    theRSet.EndUpdate
end
theRSet.Close
```

This script changes the Description value for all records in the Highways table whose name is Highway 55. Because you don't need to read any values from the database to edit those records, you don't need to use the Next request.

It is essential that the QueryDef select only the columns whose values will be changed. If this query selected additional columns from the table but didn't set their values, the current record's Value objects would all contain null values except for the Description value. That's because this script doesn't use the Next request. When executing Update, the original values for all columns in the RecordSet are replaced; the additional columns would no longer have any values in the records selected by the Edit request.

The above examples assume that the only Highway 55 records in the Highways table are those associated with the proposed highway. Suppose the Highways table contains roads for California and Arizona, that California has a Highway 55, Arizona doesn't, and the proposed Highway 55 is in Arizona. Both example scripts in this section would have to be modified to change only the Highway 55 records in Arizona.

The first script would have to read values from the State column as well to ensure the current record's Highway 55 was in Arizona. Its Where clause for the Edit request is still correct because the Key value is unique for each record. The second script's Where clause for the Edit request would have to change to choose the Highway 55 records from Arizona only; for example, "Name = 'Highway 55' and State = 'Arizona'". The Edit request's Where clause needs to include the State = 'Arizona' criterion even if the query's Where clause selected highways from Arizona only.

If you create a query using an SQL Select statement (SetSQL request), you can loop through the selected records with a RecordSet, but you can't use RecordSet requests to modify the source data. The RecordSet.CanUpdate request will return false. Instead, you need to create another query that has an SQL Insert, Update, or Delete statement. The script below shows how.

```
theQuery = ODBCQueryDef.Make(theCon)
theQuery.SetSQL("Update Highways Set Name = 'Highway 50' Where Name =
'Albion Road'")
```

```
theQuery.Execute
```

This script changes a street's name from Albion Road to Highway 50. Use the Execute request to send the database an SQL statement that doesn't retrieve records. The QueryDef class can also be used to create tables or indexes by using the appropriate SQL statement in the SetSQL request and then using the Execute request to execute the SQL statement.

Deleting records in the database

Suppose Arizona's proposed Highway 55 project is cancelled. The following script shows how you can delete records associated with this project from the Highways table.

```
theQuery = QueryDef.Make(theCon)
theQuery.SetSelectColumns({"Description"})
theQuery.SetFromTables({"Highways"})
theRSet = theQuery.OpenRecordSet
if (theRSet.CanUpdate) then
    theRSet.Delete("Name = 'Highway 55' and State = 'Arizona'")
end
theRSet.Close
```

The Delete request removes the records identified by its Where clause from the database. You don't use the Update or EndUpdate request with the Delete request.

Creating new tables in the database

There are no requests provided with the ODBC classes to create new tables in the database. This is because there is no standard way of creating a table in a database using ODBC. To create a new table, you must use the SetSQL and Execute requests to execute a database-specific command. The script below creates a new table in an Oracle database that will contain information about secondary roads.

```
'define the query
theQuery = QueryDef.Make(theCon)
theQuery.SetSQL("Create table secondary (keycol number(5), name
varchar2(30), description varchar2(50), state varchar2(30))")
ok = theQuery.Execute
if (ok.Not) then
    MsgBox.Error(theCon.GetErrorMsg,"Error creating secondary table")
end
```

This method can also be used to perform other types of database operations for which a request has not been provided. Dropping tables, truncating tables, and creating indexes are some examples.

Using locks and transactions

When you modify the source data in a database, it's important to use transactions and locks. Transactions let you control when and if your changes become permanent in the database. Locks keep other users from updating the data that you are currently editing in the database. Using both transactions and locks lets you maintain the integrity of a dynamic database in a multiuser environment.

The ODBCConnection class provides a number of requests for setting up a transaction. There are no ODBC-specific requests for creating locks since some ODBC databases handle locking in a very different way from others. To create a lock, you must execute a database-specific locking statement on the database using the Execute request.

The following script shows how to use transactions and locks while looping through a RecordSet to perform updates.

```
'open the recordset to update the Description field
theCon = ODBCConnection.Make("Highway DSN", "", "jdoe", "mypasswd")
theTable = ODBCTable.Make(theCon, "Highways")
theRSet = theTable.OpenRecordSet("")
if (theRSet.CanUpdate) then
    theCon.BeginTrans

    'try to lock the table
    lockStr = "lock table Highways in exclusive mode nowait"
    aQDef = ODBCQueryDef.Make(theCon)
    aQDef.SetSQL(lockStr)
    isLocked = aQDef.Execute

    'If not successful, exit
    if (isLocked.Not) then
        MsgBox.Error("Unable to lock highways table", "Update Highways")
        theCon.Rollback
        theRSet.Close
        Return 0
    end

    theFields = theRSet.GetFields
    nameIndex = theFields.Find(theRSet.FindField("Name"))
    keyIndex = theFields.Find(theRSet.FindField("KeyCol"))
    descField = theRSet.FindField("Description")
    theRow = theRSet.GetRow
    while (theRSet.Next)
        'get current record's values, edit if it's a Highway 55 record
        hwyName = theRow.Get(nameIndex)
        if (hwyName = "Highway 55") then
```

```

        'identify which highway 55 record you're editing with the
key
        keyValue = theRow.Get(keyIndex)
        theRSet.Edit("KeyCol = "+keyValue.asstring)
        theRSet.SetValue(descField, "Proposed")
        success = theRSet.Update
        if (success.not) then
            MsgBox.Error("Unable to update the record", "Update
Highways")
            theCon.Rollback
            theRSet.Close
            Return 0
        end
    end
end
theRSet.EndUpdate
theCon.CommitTrans
end
theRSet.Close

```

The script performs the same function as the first script in the Editing Records in the Database section above. The difference is that this script includes transaction requests and executes the Oracle Lock Table statement. By using transactions, you can roll back all of the updates if you encounter an error. The lock table statement, as used above, prevents conflicts by ensuring that you are the only one allowed to update the table in the database.

The BeginTrans request starts the transaction. The Rollback request undoes any inserts, updates, or deletes that have occurred since the BeginTrans request and ends the transaction. The CommitTrans request makes all pending updates, inserts, and deletes permanent, and also ends the transaction. You must always begin a Transaction with the BeginTrans request and end it with either the CommitTrans or Rollback requests.

While locked, no one else can edit the table even if they attempt to edit it in another application such as SQL*Plus®. With Oracle, the Commit or Rollback statements free the table lock. Some ODBC drivers may not support transactions and locks. Make sure that you check the driver and database documentation to confirm that these operations are valid.

Checking for errors

The examples above have included little error checking, focusing instead on illustrating functionality. However, it's essential to check for errors in your scripts.

When you create a new connection, you must check the Connection object's status before continuing with your script to make sure the database connection was successful. Check a connection like this:

```
theCon = ODBCConnection.Make("Highway DSN", "", "jdoe", "mypasswd")
if (theCon.HasError) then
    msgbox.Error(theCon.GetErrorMsg, "Error Message")
    Return 0
end
```

If the connection wasn't successful, you can't continue executing the script. However, you can use the erroneous ODBCConnection object to find out from ODBC why your connection failed.

The GetErrorCode request returns the native database error code number that was generated as a result of problem. The GetErrorMsg request returns more detailed information that includes where the message was generated (i.e., ODBC driver, database) and a description of the problem.

After successfully creating a connection, you shouldn't have to check its error status again. Errors that occur when querying or changing data shouldn't impact your connection. However, it's possible for a serious database I/O error to confuse communication and prevent you from continuing with your work.

Most requests on Database Access objects return a value indicating whether or not they were successful. If the remainder of your script is dependent on the outcome of a request, it's essential to check the outcome before your script continues. The transaction example checks the Boolean value returned by the Update request and rolls back the transactions if it wasn't successful. You can learn more about the problem by getting the error code from the connection before exiting.

Some requests return objects other than Boolean values; the OpenRecordSet request returns a RecordSet object. If a query is syntactically incorrect, the database can't understand it, and OpenRecordSet returns a nil object instead of a RecordSet. All of the scripts in this chapter working with RecordSets should include the following lines, but they didn't because they were focusing on functionality rather than error checking.

```
theRSet = theQuery.OpenRecordSet
if (theRSet = nil) then
    msgbox.Error(theCon.GetErrorCode.AsString, "Error Code")
    Return 0
end
```

There is no way of checking a query's validity in ArcView before you open a RecordSet. If you create a DBTable from an invalid ODBCQueryDef, you don't get a nil object. Instead, you need to check the DBTable's error status before using it.

```
theDBTable = DBTable.Make(anODBCQueryDef)
if (theDBTable.HasError) then
    msgbox.Error(theDBTable.GetErrorMsg, "Error Message")
    Return 0
end
```

Always check the outcome of a request and retrieve the error code from ODBC if it failed. There is no need to check when the request is successful because the error code and error message are only updated when there is a failure.

It is possible for ODBC to return more than one set of error messages and codes. The `GetErrorMsg` request and the `GetErrorCode` request only return the most significant of these. The `GetDBDiag` request returns an object that contains all of the error and warning messages for the last action. The following script shows how to access all of the messages returned by ODBC.

```
theCon = ODBCConnection.Make("Highway DSN", "", "jdoe", "mypasswd")
if (theCon.HasError) then
  theDiag=theCon.GetDBDiag
  msgbox.info(theDiag.ReturnReturnCode.asstring, "")
  msgbox.listasstring(theDiag.GetAllDiagMessages, "", "")
  Return 0
end
```

The request `GetDBDiag` returns an `ODBCDiagnostic` object which contains the error codes, error numbers and error messages for the last action taken by the `ODBCConnection`. The `ReturnReturnCode` request returns the ODBC return code for the last action. The `GetAllDiagMessages` request returns the error messages in a list.

By default, errors returned by ODBC are automatically displayed. In your application, you may want to control when error messages are displayed rather than having ArcView display them automatically. To do this, you can use the `SetErrorLogStatus` request.

```
theCon = ODBCConnection.Make("Highway DSN", "", "jdoe", "mypasswd")
theCon.SetErrorLogStatus (#ODBCERRORLOGGING_NONE)
theCon.SetWarningsLogStatus (#WARNINGSLOGGING_NONE)
```

The `#ODBCERRORLOGGING_NONE` enumeration indicates that errors are not to be displayed automatically. In your application you can use the error retrieval requests to get and display the messages yourself. When your application finishes, you may want to execute the request again with the `#ODBCERRORLOGGING_DISPLAY` enumeration to make errors display automatically.

You can also control how warning messages are displayed. To do this, use the `SetWarningsLogStatus` request with warning enumerations just as you use the `SetErrorLogStatus` request with error enumerations. By default, warnings returned by ODBC are not displayed.

Working with selected records

When you create a keyset database table, you can select records in the table just like with regular tables. What is different, however, is the way that you access the selected set with Avenue. With a regular table, the values in the selection bitmap correspond to the records in the VTab. In a database table, the values in the bitmap refer to the rows in the table. To get the corresponding record in the database, you must use the `ConvertRowtoRecord` request. This request returns the row's keyset value, which can be used as a unique identifier in the database.

The sample script titled “Deleting selected records in a database table” shows how to access the records in a database based on the selected rows in a database table. To find the sample script, go to the contents page of the on-line help and look for the Database Access book under extensions. Under the Database Access book, look for sample scripts and extensions under which you will find a sample scripts book with all of the extension's sample scripts. The script titled “Deleting selected records in a database table” shows an example of how to access the records in the database.

APPENDIX A

Setting up your computer

Before using database data in ArcView, you need to establish a connection between ArcView and the relational database. Each database interface connects to databases differently. Some database interfaces require additional software to be installed on your computer, while others don't. Before ArcView can connect to the database, make sure your computer has been set up correctly.

Find out from your database administrator whether or not your computer has been set up already. If you can set up your computer yourself, refer to the appropriate section in this appendix to find out what information and software you need, then follow the steps for setting up your computer. If you are unable to set up your system, speak to your database administrator.

In this chapter, you'll find out how to:

- Set up your computer for SDE.
- Set up your computer for ODBC.

Setting up your computer for SDE

To set up your computer for SDE, you don't need to install any additional software; SDE only requires that the SDE server and your computer communicate on a TCP/IP network. However, it is necessary to edit your computer's Hosts and Services files for both PC and UNIX® operating systems; the changes are the same for both.

You can edit the Hosts and Services files in any text editor. Before you begin, you need to find out two pieces of information from your SDE database administrator: the service number for the SDE database instance, and the IP address for the SDE server computer on the network. ArcView shouldn't be open while you're editing these files.

File locations

- Windows NT®: <winnt root>\system32\drivers\etc
- Windows 95®: <win95 root>
- UNIX: /etc

Hosts file

For each SDE server that you connect to, the Hosts file must contain its computer name and its IP address. The host is the SDE server computer, and its IP address tells your computer how to find it on the network. Add a line to the Hosts file for each server with this format:

```
<host name>      <IP address>
```

For example, if the SDE server is called jazz, the entry would look something like this:

```
jazz              192.23.75.252
```

PC note: If your PC doesn't already have a Hosts file, you need to create one. A sample Hosts file is often provided, called Hosts.sam. When saving the new Hosts file, some text editors will automatically attach a .txt extension to it; rename the file to eliminate the .txt extension.

After editing the Hosts file, make sure you can 'ping' the SDE server. You can ping a machine by typing "ping <host name>" at your prompt. On a PC computer, if you don't have a Ping utility, open a DOS window to get a prompt. If you don't get a reply from the server, then you likely spelled its name wrong or typed the IP address incorrectly in the Hosts file.

Services file

An SDE server can have more than one database instance; each instance on a server must have a unique name and service number. For each SDE database instance that you want to connect to, this file must contain its instance name and service number. The service number is like a communication channel. Add a line to the Services file for each instance with this format:

```
<instance name> <service number>/tcp
```

For example, SDE's default instance name is `esri_sde`, and the default service number is 5150:

```
esri_sde          5150/tcp
```

The service number must be unique in the Services file; if it appears on more than one line, you'll have trouble connecting to the database.

PC note: Notepad is known to sometimes store hidden characters in text files. If you have trouble connecting, check this file in the DOS editor and remove any incorrect characters.

Yellow pages

Networks often have 'yellow pages' Hosts and Services files. Depending on how your network is set up, your machine may use the yellow pages files in addition to the Hosts and Services files on your computer.

To illustrate how the yellow pages files work, suppose you can't connect to SDE unless you edit the local Services file, but you can connect without editing the local Hosts file. This happens because your machine knows to check the yellow pages Hosts file, which has an entry for the server you're connecting to; however, the SDE database instance wasn't in the yellow pages Services file, so you had to edit the local Services file.

You shouldn't rely on the yellow pages files. Always edit your local Hosts and Services files to contain the correct information. Yellow pages files are often incomplete. There are usually different yellow pages files for different subnetworks in a large network environment; an SDE server may appear in one yellow pages Hosts file but not in another. To change the yellow pages files you need to contact your system administrator.

For HP users only

HP® users have an extra step. These computers use a configuration file to determine the order in which they search for local or network Services files; they use whichever Services file is found first. If you've made changes to the local Services file, you need to make sure that file will be used; this may require editing the configuration file.

The default configuration file for HPs is /usr/newconfig/etc/nsswitch.conf. The line in question is:

```
Services: nis    files
```

where “nis” stands for “network information services”. This configuration line tells the computer to look for the network’s Services file first. Most networks have Services files; therefore, with this configuration, the changes you’ve made to the local Services file will be ignored. “files” refers to the local system files.

Before looking for the default configuration file, HPs look for a custom configuration file, /etc/nsswitch.conf. If this file exists, it will be used instead of the default configuration file. If this file doesn’t exist, and if you want the local Services file to be used, copy /usr/newconfig/etc/nsswitch.conf to /etc/nsswitch.conf.

In the /etc/nsswitch.conf file, change the order of the Services entry so it appears like this:

```
Services: files nis
```

Now your computer will use the local Services file, which contains the SDE instance information required for connecting to an SDE database.

Contact your system administrator if you need help setting up your computer.

Set up your computer for ODBC

In order to set up a machine to connect to a database using ODBC, you need to have the proper software installed and configured for your database. The following describes the software that is needed, where to find it, and how it needs to be configured.

ODBC

You need to have ODBC version 3.0 or greater installed on your system. If ODBC is installed, you should find the ODBC Data Source Administrator in the Control Panel with an icon labeled ODBC. If you do not have ODBC installed, you can download it from the Microsoft web page. ArcView will also automatically install ODBC version 3.510.3711.0.

You can find the version of ODBC installed on your machine by searching for the file odbc32.dll and checking its version.

ODBC drivers

You also need to have the proper ODBC drivers installed. To see if there are drivers installed on your system, go to the Control Panel, bring up the ODBC Data Source Administrator and examine the list of drivers. The installed drivers are listed under the ODBC drivers tab. Database Access requires ODBC version 3.0, but you can use ODBC version 2.x drivers since ODBC is backwards compatible.

If you don't have the driver that you need, you can find drivers on the Internet from vendors like Microsoft and Intersolv[®], or your DBMS vendor. If you are running Windows 95, make sure that you are using the 32 bit ODBC administrator and have installed 32 bit ODBC drivers.

Client software

Certain relational databases require client software to be installed:

- Oracle requires SQL*Net[®].
- Sybase requires Open Client.
- Informix[®] requires Informix-Net.
- Some non-Microsoft SQL Server[™] drivers require the SQL Server client.

If you are not sure whether you need to install client software, check the driver documentation.

Data sources

Unless you are using Avenue, you will need to configure a data source for the driver that you are planning to use. A data source is used by ODBC to store information that is needed by a driver to connect to a database. All data sources are required to have a name, which is referred to as a DSN, or data source name. The rest of the information depends on the database.

To configure a data source, go to the Control Panel and bring up the ODBC Data Source Administrator. The administrator allows you to create data sources that are associated with your user or the local machine or a file. When you add a new data source, the driver provides an interface to collect the needed information. Refer to the driver documentation and ODBC Data Source Administrator's online help for information on configuring your particular data source.

The data sources are distinguished by their data source name, which is always unique. When you start ArcView and load the Database Access extension, you can connect to your database by referencing the ODBC data source using the DSN. You can also configure new data sources after starting ArcView.

APPENDIX B

Creating new tables in SDE

You can create new tables in an SDE database with Avenue scripts. You define the name and data type of each column in the new table by creating a Field object.

You can create many types of fields in ArcView, but you can't store all of them in an SDE database, and some fields might produce different columns in the database than you would expect, either in the column type or in its size and number of decimal places. SDE places limits on columns so the tables you create will store exactly the same data, no matter which relational database management system you're using with SDE.

In this chapter, you'll learn about:

- SDE's column rules.
- How each ArcView field type will be represented in the database.

Creating new tables in the database

To create a table in your database, you create a new SDETable object in an Avenue script using the MakeNew request. This new SDETable object represents the table's definition in the database; you need to provide the name and data type of each column in the database, and a unique name for the table. A column's data type includes the kind of column it is (e.g., Number or Date), the column's size, and the number of decimal places allowed, where appropriate.

To define a column in ArcView, you create a Field object; you specify a field enumeration to define the type of data the column will hold (e.g., #FIELD_DATE). The Field's width defines the column's size, and the Field's precision defines the number of decimal places the column can have. To create the new SDETable, you provide a unique name and a list of the Fields that together define the table you want to create. To add this table to the database, you use the SDEConnection.CreateTable request.

ArcView's field enumeration includes several types that aren't appropriate for creating a new table in a relational database. For example, to create a spatial column for point shapes, you don't add a shapepoint field to the field list; you need to write an Avenue script that creates a new SDESColumn object. If the table's definition includes a Field representing an unsupported type, the CreateTable request will fail.

With some number Fields, you might not get the expected results when the table is created in the database. SDE doesn't define as many number column types as ArcView does; more than one field type will create the same column definition in the database. When you create number Fields, you can specify any width and precision; each SDE number column has rules about the allowed size and number of decimals, and a default definition. When a Field doesn't follow those rules, its definition will be changed, the default definition will be used instead, or the Create Table request will fail.

SDE column rules ensure you can store the same values in a table with the same definition in any SDE database, regardless of which relational database you're using. The table below illustrates how ArcView fields relate to SDE columns and describes SDE's rules for them. The default definition for number columns is provided for Oracle databases only, but they are similar for other relational databases. The actual column definitions will vary depending on which relational database management system you're using.

ArcView Field Enumerations	SDE Column Types	Notes
#FIELD_BYTE	SE_SMALLINT_TYPE	See #FIELD_SHORT
#FIELD_CHAR	SE_STRING_TYPE	See #FIELD_VCHAR
#FIELD_DATE	SE_DATE_TYPE	Creates a Date field.
#FIELD_DECIMAL	SE_DOUBLE_TYPE	See #FIELD_DOUBLE
#FIELD_DOUBLE	SE_DOUBLE_TYPE	<ul style="list-style-type: none"> • Size must be ≥ 8; if size is < 8, use default instead: Number(15,4). • Decimals must be > 0; if decimals = 0, size increases by 4 and decimals = 4.
#FIELD_FLOAT	SE_FLOAT_TYPE	<ul style="list-style-type: none"> • Size must be ≥ 1 and ≤ 6; if size is > 6, use default instead: Number(6,2). • Decimals must be > 0; if decimals = 0, size increases by 2 and decimals = 2.
#FIELD_ISODATE	SE_DATE_TYPE	See #FIELD_DATE
#FIELD_ISODATETIME	SE_DATE_TYPE	See #FIELD_DATE
#FIELD_ISOTIME	SE_DATE_TYPE	See #FIELD_DATE
#FIELD_LOGICAL	SE_STRING_TYPE	See #FIELD_VCHAR
#FIELD_LONG	SE_INTEGER_TYPE	<ul style="list-style-type: none"> • Size must be ≥ 5 and ≤ 9; if size is < 5 or > 9, use default instead: Number(9). • Decimals must be = 0; if decimals > 0, CreateTable fails.
#FIELD_MONEY	SE_STRING_TYPE	See #FIELD_VCHAR
#FIELD_SHAPELINE	N/A	N/A
#FIELD_SHAPEMULTIPOINT	N/A	N/A
#FIELD_SHAPEPOINT	N/A	N/A
#FIELD_SHAPEPOLY	N/A	N/A
#FIELD_SHORT	SE_SMALLINT_TYPE	<ul style="list-style-type: none"> • Size must be ≥ 1 and ≤ 4; if size is > 4, use default instead: Number(4). • Decimals must be = 0; if decimals > 0, CreateTable fails.
#FIELD_UNSUPPORTED	N/A	N/A
#FIELD_BLOB	SE_BLOB_TYPE	Creates a Long Raw column; can store any data (e.g., ASCII text files, ArcView ODB files).
#FIELD_VCHAR	SE_STRING_TYPE	Creates Varchar2 column with size provided.

Index

A

- Add Database Table dialog
 - ODBC 54
 - SDE 26, 37
- Avenue 7, 8

C

- cache size
 - ODBC 47
 - SDE 37–38
- charts
 - ODBC 49
 - SDE 38
- columns
 - ODBC
 - aliases 94
 - indexing 57
 - SDE
 - aliases 28, 30, 78
 - indexing 35
- common objects 62
- Connection (class) 62
- connections
 - ODBC
 - errors connecting 101–102
 - existing 50, 54–56, 95
 - new 43, 66–67, 92, 100
 - repairing 53
 - SDE
 - errors connecting 86
 - existing 17, 25, 26, 71
 - new 15, 25, 64, 70, 77, 88
 - repairing 25

D

- data source name
 - ODBC 10, 66, 109–110
 - SDE 9, 64–65
- data source setup, ODBC 109
- Database Access
 - integrating with other extensions 36
 - loading the extension 14, 42–43

- object model 62
- database administrator
 - ODBC 9, 95
 - SDE 8, 80
- database instance 8, 15, 64
- database interfaces
 - defined 63
 - ODBC 66–67
 - SDE 64–65
- database name
 - ODBC 10, 66–67
 - SDE 9, 15, 64–65
- Database Table Preferences dialog
 - ODBC 47, 48, 54–56
 - SDE 24, 36
- database tables
 - defined 6, 62
 - ODBC
 - cache size 47
 - changing contents 57–58
 - creating 92
 - errors creating 102
 - fetch count 47
 - forward only scrolling vs. Keyset 92
 - getting statistics 56, 95
 - performance 57
 - renaming 48
 - repairing connections 53
 - stepping through records 93
 - summarizing records 95
 - SDE
 - cache size 37–38
 - changing contents 29
 - creating 76–80
 - errors creating 87
 - fetch count 37–38
 - forward only scrolling vs. Keyset 77
 - getting statistics 27, 80
 - joining to local tables 39
 - linked with dbthemes 23
 - performance 35
 - refreshing 26
 - repairing connections 25
 - spatial columns in 24, 79
 - stepping through records 79
 - summarizing records 36–37, 80
- database tables, forward only scrolling
 - ODBC
 - creating 42–43, 92
 - defined 45
 - viewing data 46–47
 - SDE
 - creating 26–27, 36
 - defined 23
 - viewing data 37
- database tables, keyset
 - ODBC
 - converting to 48
 - creating 50, 54–56, 92
 - defined 47
 - maximum keyset size 48
 - selecting records 48–49, 104
 - SDE
 - converting to 38
 - creating 23
 - defined 23
 - maximum keyset size 24
 - selecting records 77, 80
- database themes
 - attribute criteria 30, 35
 - attributes 19, 22–23, 23, 32–33, 62
 - changing contents 30–31, 32–33, 70–71
 - changing their names 16
 - connection 22
 - creating 15–17, 17–18, 70
 - defined 18, 62
 - display property 21
 - drawing 16, 70, 72
 - exporting features 35, 35–36, 36
 - joining tables 17–18, 32–33, 33
 - linking to dbtables 23
 - performance 35
 - projecting 20
 - repairing connections 25
 - selecting features 33–34, 66, 73, 74, 78
 - spatial criteria 35, 71
 - stepping through records 72
 - symbolizing 19–20, 28–29
 - using with other extensions 36
- databases
 - ODBC
 - of tables 10
 - SDE
 - of tables 9
- DBTable (class) 62

DBTables. *See* database tables

DBTheme (class) 62

DBThemes. *See* database themes

distances

specifying 75

drivers, ODBC 109

E

editing the database 7

ODBC

adding new tables 99

deleting records 99

inserting records 95–96

locking records 100–101

permissions for 95

updating records 96–99, 98–99

using transactions 100

with SQL statements 98–

99, 99, 100

SDE

adding new spatial columns 88–90

adding new tables 88, 112–113

deleting records 85

inserting records 81–82

locking records 85–86

permissions for 80

updating records 80–81, 82–84

using transactions 85

with SQL statements 80–81

error checking

ODBC

connecting to ODBC 101–102

creating database tables 102

failed requests 102

opening record sets 102

SDE

connecting to SDE 86–87

creating database tables 87

failed requests 87

opening record sets 87

F

fetch count

ODBC 47

SDE 37–38

Field (class) 62

field names

ODBC 94

SDE 78

Fields

ODBC

changing properties 93

getting statistics 53, 56, 95

summarizing 95

SDE

changing properties 73

getting statistics 26, 27, 80

summarizing 36–37, 80

Forward only scrolling dbtables

ODBC. *See* database tables: ODBC

creating 42–43, 92

defined 45

viewing data 46–47

SDE 77. *See also* database tables:

SDE

creating 26–27, 36

defined 23

viewing data 37

H

help

getting technical support 12

on-line 11

topics for this extension 11

J

joining tables

inner joins 19, 33, 45

outer joins 33

K

KeyColumn

ODBC 92

SDE 77

Keyset dbtables

ODBC. *See* database tables: ODBC

converting to 48

creating 50, 54–56, 92

defined 47

maximum keyset size 48

selecting records 48, 104

SDE. *See* database tables: SDE

converting to 38

creating 23

defined 23

maximum keyset size 24

selecting records 77, 80

KeyTable

ODBC 92

SDE 77

L

local files 19, 40

locking records

ODBC

in tables 100–101

SDE

in attribute tables 85

in tables with shapes 85–86

logfiles 66

creating 74

for selections 73

querying 76

temporary 76

M

maps. *See also* Views

projections 20–21

maximum keyset size

ODBC 48

SDE 24

measured shapes 89

O

ODBC

connecting to 43, 66–

67, 92, 100, 101–102

drivers 109

getting errors from 102

installation 108

object model for 66

required client software 109

setting up your computer for 108–109

ODBCConnection (class) 66

ODBCConnections. *See* connections

ODBCDiagnostic (Class) 67

ODBCDiagnostics 103

ODBCQueryDef (class) 67

ODBCQueryDefs. *See* queries

ODBCRecordSet (class) 67

ODBCRecordSets. *See* RecordSets

ODBCTable (class) 67
 ODBCTables
 creating 97
 opening a record set 97

P

projects
 ODBC
 logging in 52
 opening 52
 repairing 53–54
 saving 51, 59
 SDE
 and selections 35, 76
 logging in 22
 opening 22
 repairing 25
 saving 21

Q

queries 62. *See also* SQL Query
 Builder dialog
 ODBC
 creating 45, 94, 95
 getting results 93
 performance 57
 SQL 92, 94, 98, 99, 100
 syntax 57, 102
 table and column quote characters 59
 SDE
 changing 70–71
 copying 65, 74, 78
 creating 18–19, 27, 70, 71, 77, 78
 destination 66, 74, 76
 getting results 72
 performance 35
 saving 35, 76
 source 66, 75, 76
 spatial vs. nonspatial 63
 SQL 77, 80
 syntax 28, 87
 SQL 63
 query order 35
 QueryDef (class) 62
 QueryDefs. *See* queries

R

RecordSet (class) 62
 RecordSets
 how they work 62–63
 ODBC
 deleting records 99
 editing the database with 95
 errors when opening 102
 how they work 93
 inserting records 95–96
 opening 67, 93, 97
 stepping through records 93
 updating records 96–99
 SDE
 deleting records 85
 editing the database with 80, 81
 errors when opening 87
 how they work 73
 inserting records 81–82
 opening 65, 72, 83
 sending records to a logfile 74, 76
 stepping through records 72, 79
 updating records 82–84
 relational database 6, 7
 ODBC
 characteristics 57
 getting errors from 102
 permissions 95
 query performance 57
 SQL syntax 57, 102
 SDE
 characteristics 28
 getting errors from 87
 permissions 22, 80
 query performance 35
 SQL syntax 28, 87

S

SColumn (class) 62
 SDE 6
 connecting to
 15, 64, 70, 77, 86, 88
 database instance 8, 15, 64
 database interface 63
 getting errors from 87
 object model for 64
 server 8, 15, 64

 setting up your computer for 106–107
 SDEConnection (class) 64
 SDEConnections. *See* connections
 SDELog (class) 66
 SDELogs. *See* logfiles
 SDEQueryDef (class) 65
 SDEQueryDefs. *See* queries
 SDERecordSet (class) 65
 SDERecordSets. *See* RecordSets
 SDESColumn (class) 65
 SDESColumns. *See* spatial columns
 SDESFilter (class) 65
 SDESFilters
 creating 71
 SDETable (class) 65
 SDETables
 adding a new spatial column 88–90
 creating 83
 making a new table 88, 112–113
 opening a record set 65, 83
 spatial columns
 and database themes 62
 creating 70
 making new 88–90
 Spatial Database Engine 6
 SQL
 ODBC
 learning 56–57
 statements 56, 92, 94, 98–99, 99, 100
 syntax 57, 102
 table and column quote characters 59
 SDE
 learning 28
 statements 27, 77, 80, 80–81
 syntax 28, 87
 statements 63
 SQL Query Builder dialog
 ODBC 48–49
 SDE 33–34
 SQL Query Wizard
 ODBC 43, 50
 SDE 15–17, 17–19, 30
 Structured Query Language 6

T

three-dimensional shapes 89

transactions

 ODBC

 using 100–101

 SDE

 using 85–86

U

Unique Column. *See also* KeyColumn

 ODBC 44, 48, 55, 56

 SDE 23, 27, 38

Unique Column Table. *See also*

 KeyTable

 ODBC 44, 48, 55

 SDE 23, 38

V

View (class) 62

Views

 adding database themes 15–17, 17–18

 changing scale 20, 21

 creating 15

 projecting 20–21