User's Guide

AcuXDBC™

Version 8.1

Micro Focus
9920 Pacific Heights Blvd.
San Diego, CA 92121
858.790.1900

E-01-UG-080901-AcuXDBC-8.1

Contents

Chapter 1: Introduction

Chapter 2: AcuXDBC Architecture

Chapter 3: Preparing Your COBOL

Chapter 4: Configuration

Chapter 5: Installing AcuXDBC

Chapter 6: The System Catalog

Chapter 7: SQL Supported Commands

Chapter 8: Working with Windows and Java Applications

Chapter 9: Troubleshooting

Appendix A: Compatibility Guide

Index

1 Introduction

Key Topics

1.1 Overview

Welcome to the AcuXDBC™ data management system, designed to integrate ACUCOBOL-GT® data files into a relational database-like environment. AcuXDBC enables you to apply SQL and relational database concepts to your COBOL data files resulting in data that is accessed and managed in much the same way as many of today's popular relational database management systems.

AcuXDBC is the next generation of AcuODBC and is engineered to provide broader flexibility in the way your COBOL data is accessed and maintained. Like previous versions of AcuODBC, AcuXDBC lets you retrieve and update ACUCOBOL-GT's Vision indexed files, relative files, and sequential files from Windows-based applications including Microsoft Word, Excel, and Access. Business Intelligence tools such as Crystal Reports® Professional, and custom applications developed in ODBC (Open Database Connectivity) supported environments such as Visual Basic® are supported as well. With the enterprise edition, new functionality lets you retrieve data through Java applications that use JDBC (Java Database Connectivity) standards. Direct SQL access to your ACUCOBOL-GT data is now available in both the Windows and UNIX environments, in both single and two-tier configurations. AcuXDBC is part of the ***extend***® family of solutions.

This manual describes how to configure and use AcuXDBC to access data from ODBC and JDBC enabled applications. It also describes what SQL commands and relational database concepts users can apply to a Vision database.

Unless otherwise indicated, the references to "Windows" in this manual denote the following 32-bit versions of the Windows operating systems: Windows Vista, Windows XP, Windows NT 4.0 or later, Windows 2000, Windows 2003; and the following 64-bit versions of the Windows operating system: Windows Server 2003 and 2008 x64, Vista x64. In those instances where it is necessary to make a distinction among the individual versions of those operating systems, we refer to them by their specific version numbers ("Windows 2000," "Windows NT 4.0," etc.).

1.2 Features of AcuXDBC

AcuXDBC provides ACUCOBOL-GT data files with extended data management capabilities typically associated with relational database management systems, as well as seamless access to this data from popular ODBC and JDBC enabled applications. To accomplish this, many new AcuXDBC features are implemented, as well as enhancements to previous features of AcuODBC.

1.2.1 Relational Database Features

Tabular structure of Vision data. The fundamental concept behind relational databases is the organization of data within collections of tables consisting of columns and rows. AcuXDBC creates a Vision database by restructuring Vision files into tables. It does this by importing extended file descriptors or XFD files, which can be generated by the checker to describe the schema information to be associated with a COBOL data file.

System catalog. Special utilities included with AcuXDBC create and populate a system catalog with information obtained from your XFDs. No special restructuring of your data files is necessary in order to gain the relational database-like functionality.

SQL DDL (Data Definition Language) support. AcuXDBC supports many of the DDL commands such as CREATE and DROP, which enables the creation and deletion of new tables or views based on conditional operators and multiple fields/columns.

SQL DML (Data Manipulation Language) enhancements. In addition to DDL support, AcuXDBC includes broader and more complete support for DML commands such as UNION, UNION ALL, combined inner/outer joins, as well as an optimized ORDER BY clause.

Multi-level security. Through the use of the DDL commands GRANT and REVOKE, AcuXDBC provides data security options at both database-wide, and object-specific levels. For example, one table from a database can be read-only for certain users while another table within the same database can be read/write accessible by either the same or different users. By combining views and object level security, even column level security is available.

Enhanced performance. AcuXDBC's re-engineered drivers and SQL processor results in faster execution of many complex SQL commands on large data sets.

1.2.2 Data Access Features

ODBC driver for ODBC-compliant Windows applications. Popular Microsoft Office applications such as Access and Excel as well as Crystal Reports and Visual Basic programs utilize the ODBC standard for connecting to external data sources. AcuXDBC provides a driver that enables users of these applications to connect to your database and execute SQL commands for data retrieval and manipulation purposes.

ODBC driver for UNIX platforms (enterprise edition). AcuXDBC enables connections to an ODBC driver from a UNIX command line or script. This allows UNIX users to access the database directly, as opposed to connecting from third-party applications such as the Microsoft Office applications mentioned above. Once connected, UNIX users can execute SQL commands or scripts through the AcuXDBC command-line query tool.

JDBC driver for Java applications (enterprise edition). AcuXDBC includes a JDBC driver, which enables Java applications to connect to the database.

Command-line query tool. An SQL query tool enables you to execute SQL commands and scripts directly on your database tables without having to connect to the database from a third-party application or going through an ODBC connection. The command-line query tool is useful for such tasks as creating batch reports that require a series of complicated SQL commands to produce.

1.3 Changes in AcuXDBC

If you are a previous user of AcuODBC, you will find changes in some of the methods used to set up and administer AcuXDBC. This is due mainly to the new relational database features, which by nature require the use of SQL and other concepts inherent to relational databases. This section briefly highlights key differences between AcuODBC and AcuXDBC. Details on each difference are provided in other sections of this manual, as well as in *Appendix A*, which describes differences in set up, and configuration.

1.3.1 System Catalog and the Role of XFDs

The system catalog forms the core of your database. It comprises a set of system tables that describe all of the tables and relations contained within the database. You can have multiple system catalogs (databases), but only a single catalog may be accessed at one time. Just like Vision files, the system catalog is fully portable between machines with the same byte ordering.

AcuXDBC populates a system catalog with information it reads from your XFDs. Once this information is loaded, AcuXDBC no longer requires your XFDs. Instead, it refers to the system catalog to obtain the information needed to construct and display your Vision tables. This gives you the option of providing end users with XFDs for building their own system catalog, or you can provide end users a pre-built system catalog. The ability to supply a pre-built system catalog can lead to other benefits for your applications. For example, you can also pre-build user logins, database object permissions, and complex data handling views.

Universal Configuration

Many of the variables that were set from the AcuODBC Configuration screen and tabs are now set from within a configuration file called “acuxdbc.cfg.” by default. If you are migrating from AcuODBC, you need to create new DSNs (Data Source Names). The main advantage of using a configuration file is that DSN settings become universal, meaning you don’t have to set them individually each time you create a DSN. Refer to Appendix A, **section A.2**, which compares the location of each AcuODBC configuration variable to its corresponding location in AcuXDBC.

DSN Setup Screen

Since many of the variables moved from the DSN configuration screen into a configuration file, the DSN setup screen has been modified appropriately and is less populated. Chapter 5, **section 5.8** includes screen shots and descriptions of the AcuXDBC Setup screen.

System Security

The AcuAccess file and manager utility are no longer used with AcuXDBC. Instead, the SQL GRANT/REVOKE commands and its variants are used to manage database security. This provides more flexibility and levels of security than were previously available. Chapter 5, **section 5.4** provides information on setting database security.

Multi-company Support

AcuXDBC offers two convenient ways to manage multiple company data sets. You can use wild cards (defined in the AcuXDBC configuration file) to identify which company file to access. Alternatively, you can assign table ownership to the different companies, then issue a single query to access all the data files. Chapter 5, **section 5.5.2** provides instructions on implementing each of these methods.

1.4 Product Requirements

To interface your COBOL data to ODBC-enabled applications through AcuXDBC, you must have the following:

- AcuXDBC's ODBC driver (acuxdbc.dll).
- ACUCOBOL-GT indexed or relative data files created with ACUCOBOL-GT Version 3.0 or later.
- XFD files, which are created at compile time with the "-Fx" or "-Fa" options, and are discussed in Chapter 3. If you have older XFDs, we recommend that you update to XFD Version 4 or 5 format.
- AcuXDBC Server if data will reside on a remote UNIX® or Windows NT server and processing will be remote. AcuXDBC Server may be licensed together with AcuXDBC, or separately.

To interface your COBOL data to a JDBC-enabled application through AcuXDBC, you must have the following:

- Java Runtime Environment (JRE) 1.5 or above.
- ACUCOBOL-GT indexed or relative data files created with ACUCOBOL-GT Version 3.0 or later.
- XFD files, which are created at compile time with the "-Fx" or "-Fa" options, and are discussed in **Chapter 3**. If you have older XFDs, we recommend that you update to XFD Version 6 format.

1.5 What Is ODBC/JDBC?

ODBC (Open Database Connectivity) is an SQL-based Application Programming Interface (API) created by Microsoft that is used by Windows software applications to access databases via SQL. Very similar in concept, is JDBC (Java Database Connectivity), which was created by Sun Microsystems to enable Java applications to use SQL for database access.

These APIs provide communications between an application residing on a client machine and a data source residing on the same client machine or on another server computer.

AcuXDBC includes ODBC and JDBC drivers for ACUCOBOL-GT's data files. AcuXDBC gives ODBC-enabled Windows applications (like those in Microsoft Office) access to ACUCOBOL-GT Vision indexed data files, relative data files, and fixed-length sequential data files. Likewise, it enables the same data access for Java applications.

1.6 Technical Services

You can reach Technical Services in the United States Monday through Friday from 6:00 a.m. to 5:00 p.m. Pacific time, excluding holidays. You can also raise and manage product issues online and follow the progress of the issue or post additional information directly through the website. Following is our contact information:

Phone:	+1 858.795.1902
Phone:	800.399.7220 (in the USA and Canada)
Fax:	+1 858.795.1965
E-mail:	**support@microfocus.com**
Online:	**http://supportline.microfocus.com**

For worldwide technical support information, please visit **http://supportline.microfocus.com**.

2 AcuXDBC Architecture

Key Topics

2.1 AcuXDBC Design

The AcuXDBC interface is designed to let users of Windows and Java applications read and write ACUCOBOL-GT Vision files (as well as relative and sequential files) as if they were tables in a relational database.

Since none of these file formats is relational to begin with, AcuXDBC's job is to present COBOL files in a table format that can be accessed by applications like Microsoft Excel or ColdFusion.

2.1.1 Basic Components

AcuXDBC includes a database tool (**xdbcutil**) that creates and populates a system catalog from your XFDs. A *system catalog* is where database systems store metadata that describe the structure of the database system itself, as well as the structure of specific tables. *XFDs* are extended file descriptors that you can generate at compile time along with your COBOL object files.

After you run **xdbcutil**, your COBOL data becomes like a relational database, a *Vision* database, accessible to applications that issue SQL requests for data. After setup, your XFDs are expendable, unless you use them for another ***extend*** product like an Acu4GL interface. AcuXDBC will not refer to them again.

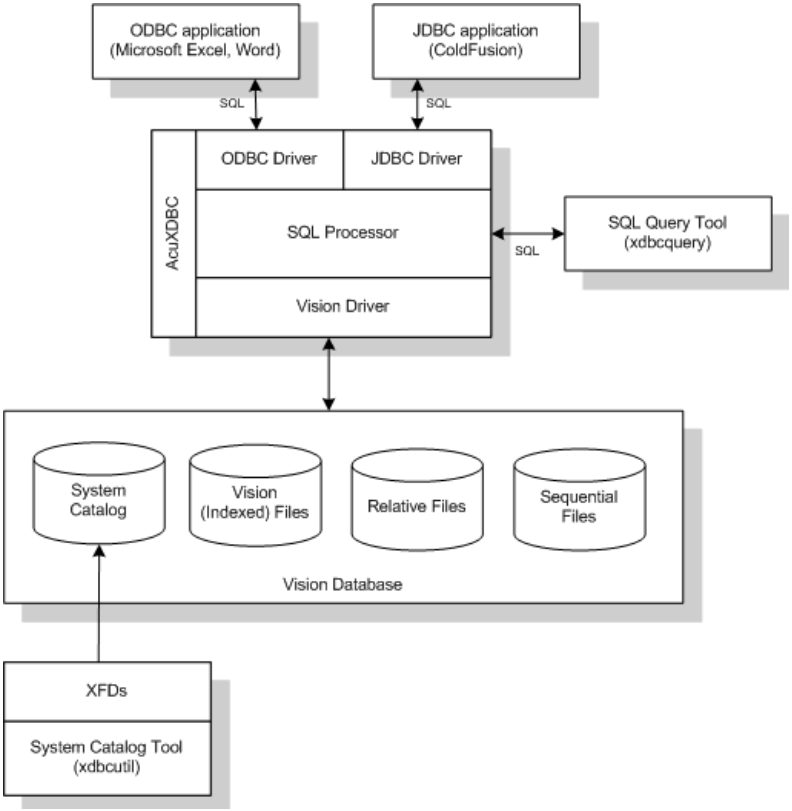
When deployed, the basic components of an AcuXDBC system include:

- An ODBC and/or JDBC driver—Receives SQL requests from ODBC- or JDBC-enabled applications and routes them to the SQL processor.
- An SQL processor—Processes SQL requests, queries the Vision database, and returns results.
- A Vision driver—Provides access to the Vision database.
- Your Vision database, comprising:
    - Vision data files
    - ACUCOBOL-GT sequential files

- ACUCOBOL-GT relative files
- A system catalog—A series of files that map your data files to database fields. Based on your XFDs, these files are themselves Vision files. The system catalog is a deployment piece and has no bearing on the COBOL developer.

With AcuXDBC, you can also use an SQL query tool such as **xdbcquery** to issue SQL requests for Vision data directly. This bypasses the ODBC/JDBC layer altogether. **xdbcquery** comes with AcuXDBC.

The following illustration depicts the basic components of AcuXDBC.



AcuXDBC transforms your COBOL data files into an SQL-like database

A Vision database is the system catalog and the COBOL files it describes (Vision, fixed sequential, and relative) that reside in one or more directories. For a given installation, you may have one or more Vision databases. Only one database may be accessed by an application at one time; however, a single file may be a component of multiple databases.

2.2 System Architecture

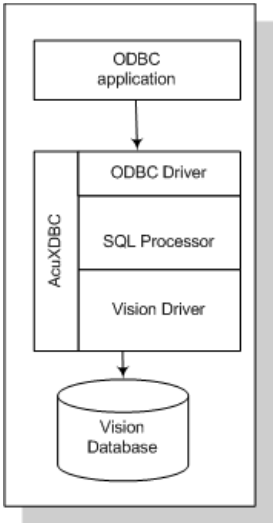
AcuXDBC can operate in one of three main configurations:

- **Local Processing (Stand-Alone)**
- **Remote Processing (AcuXDBC with AcuXDBC Server)**
- **Remote Access/Local Processing (AcuXDBC with AcuServer)**

2.2.1 Local Processing (Stand-Alone)

In a local processing configuration, both the AcuXDBC interface and the Vision database that it creates reside on the same computer. This is considered a *single-tier* architecture.

A typical local-processing installation is illustrated in the following diagram.



AcuXDBC in a stand-alone configuration

Note that AcuXDBC can operate on both Windows and UNIX machines.

2.2.2 Remote Processing (AcuXDBC with AcuXDBC Server)

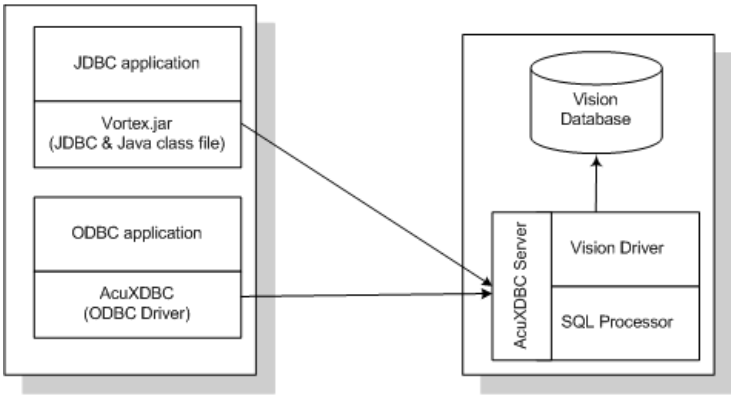
In a two-tier configuration, the client computer is running the ODBC/JDBC application or the command-line query tool. The data to be processed resides on a remote UNIX or Windows server. With AcuXDBC Server (**xdbcsrvr**) running on the remote machine, your application is able to take advantage of server-side processing of the SQL statements.

Remote processing can mean more efficient use of your system resources. Queries often process large amounts of data to get results. Remote processing decreases the amount of data that must be transferred between the client and the server. If files reside on the same server where processing occurs, network traffic is reduced. Only the final result of the processing is

returned to the client machine. In addition, this structure encourages centralized management of your data, cutting down on duplication and ensuring that the information returned to the application is up-to-date.

For remote processing on a Windows server, **xdbcsrvr** resides as a *service* on the remote machine, waiting for calls for SQL processing. On UNIX machines, **xdbcsrvr** resides as a *daemon*. AcuXDBC Server is a single-instance server: each time **xdbcsrvr** gets a request, it spawns a new copy to deal with the request and the original goes back to listening for additional requests. You may run **xdbcsrvr** continuously, or start and stop it.

The following illustration depicts the architecture for remote SQL processing.



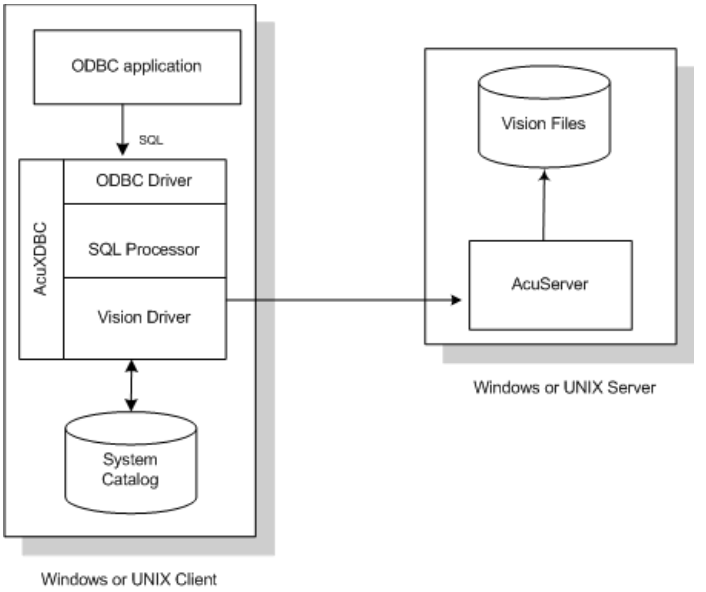
AcuXDBC with AcuXDBC Server in a remote processing configuration

2.2.3 Remote Access/Local Processing (AcuXDBC with AcuServer)

For those whose configurations already use AcuServer, AcuServer can be used to gain access to files that reside on a remote computer; however, with AcuServer, these files are transferred to the client and processed locally. Due to the performance impact of network communication, AcuServer should be used with AcuXDBC only when a small amount of data is being accessed.

AcuServer is available for most UNIX and Windows NT (Intel) machines. It can serve COBOL applications running on UNIX and Windows TCP/IP based networks. You must have a TCP/IP networking client installed on the client machine and a licensed copy of AcuServer installed on the server. For more information about AcuServer, see the AcuServer documentation.

Remote file access architecture is depicted as follows:



AcuXDBC with AcuServer in a remote access/local processing configuration

Notice that in this AcuServer configuration, the Vision database is split up. That is, the system catalog is local on the client, but the Vision data files themselves are on the remote server. The Vision data files can also be a mixture of local and remote.

2.3 Security

AcuXDBC provides a variety of security measures, some at the **network** level, and some at the **database** level.

2.3.1 Network Security Layer

AcuXDBC Server offers one form of network security: User ID and password encryption

Use the "-k" option to the **xdbcsrvr** command to encrypt the database user ID/password that goes across the network. Use it to specify the masking key value used to perform the encryption, then use the KEY_CONNECT variable in the "net.ini" file to specify the same value on each client machine. (See Chapter 4, **section 4.3**, for more information on the "net.ini" file.)

See Chapter 5, **section 5.7**, for more information on starting and stopping AcuXDBC Server.

2.3.2 Database Security Layer

AcuXDBC offers a variety of ways to secure your database. If you do not adopt any of these methods, your database is open to everyone.

- You can hide individual data fields from within your COBOL application by using the HIDDEN directive. This will instruct the compiler to flag this COBOL field as hidden when the XFD file is generated. When the XFD is subsequently loaded into the database, this field will not appear as a column in any description of the table. This is useful for hiding data like passwords, telephone numbers, and financial information—whatever information you do not want users to see. (See Chapter 3, **section 3.3.8**, for more information.)

- You can tag field/columns of data as read-only using the READ-ONLY directive. Users can then view the data but not update it. (See Chapter 3, **section 3.3.11**, for more information.)

- You can use the READ_ONLY configuration variable to establish the read/write permission for all the files belonging to the database. (See Chapter 4, **section 4.2.23**, for more information.)

- You can grant database-level and table/file-level privileges to individual users or to PUBLIC using the SQL statement GRANT. This updates the system tables known as **GENESIS_USERS** and GENESIS_AUTH. When AcuXDBC receives the connection string, it compares the

username and password supplied through the DSN (or user-entered) with the GENESIS_USERS table. If the GENESIS_USERS table doesn't exist, AcuXDBC allows the connection. If the table does exist, AcuXDBC checks to make sure the user and password match a record.

See Chapter 5, **section 5.4** and **section 5.6**, for general information on granting permissions and Chapter 7, **section 7.5.10** and **section 7.5.11**, for details on the GRANT statement.

- You can create views of your data using the SQL statement CREATE VIEW to show certain fields/columns of data in your database but not others. (See Chapter 7, **section 7.5.4**, for more information.)

  A potential situation where this can be useful is the following:

  An application has a customer record containing all of the information on a customer. The application developer would like to be able to provide the ability for clerical workers to update some of the customer information, while masking any private information.

  A good way to do this would be to make one user the owner of the full table. The application developer could then create a "VIEW" of the table that contained only the non-private information. If the view were created as owned by "PUBLIC," all users would be able to access this information while maintaining the confidentiality of the remainder of the information.

3 Preparing Your COBOL

Key Topics

3.1 Mapping COBOL Data Items and Database Fields

AcuXDBC relies upon eXtended File Descriptors (XFDs), to map your COBOL data items to database columns. XFDs are automatically generated when you compile your program with the "-Fx" or "-Fa" compiler option. If the default mapping described in this chapter is sufficient, there is nothing you need to do to prepare your COBOL for AcuXDBC except create the XFDs themselves. (See **section 3.2**.)

If you want to add more detail to your database tables, you can add *directives* to your program before generating the XFDs. Directives are optional comments that you can place into an FD in your COBOL source code to control how XFDs are built. (See **section 3.3**)

These concepts are described more fully in this chapter.

3.2 The Role of XFDs

For AcuXDBC to map your Vision data file to an SQL table format, descriptive information must be written to an XFD file for each file. An XFD typically has the same name as the COBOL data file, but its extension is ".xfd". When you distribute your AcuXDBC application, you do not need to include the XFD with the executables and data files. The XFDs are used only one time to populate your system catalog.

This section describes what information is included in the XFD and how it is mapped to a system catalog format that is easily recognized by your ODBC or JDBC application. **Section 3.3** describes how you can take control of the XFDs using *directives*. Since this material is relevant during compilation, it applies only to COBOL developers.

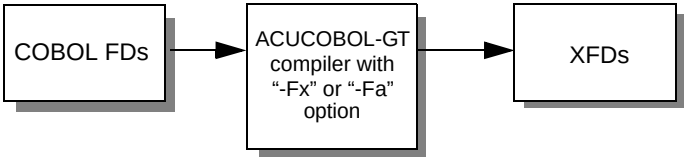
3.2.1 Creating XFD Files

To create an XFD file, specify the "-Fx" or "-Fa" option at compile time. For example:

```
ccbl32 –fa animals.cbl
```

When you specify the "-Fx" option, an ".xfd" file is created for each indexed data file specified in the compiled program. Use the "-Fa" option to generate XFDs for all indexed, relative, and sequential files. Specify "-Fae" or "-Fxe" to generate the XFDs in XML format. XFDs are stored in your source code directory unless you use "-Fo" to specify another location.

Once you compile with one of these options, each COBOL file has a corresponding ".xfd" file. The "-Fx" and "-Fa" options create ".xfd" files without changing anything in the object code.



Creating XFD files at compile time offers two significant advantages:

- Any changes made to the file definitions are automatically included in the XFDs when the program is recompiled.
- The effects of all compile-time options, COPY REPLACING, and source-code control lines are reflected correctly in the XFDs.

The XFDs contain information about the structure of the COBOL files, the names of the fields, their format, and so on. However, this information is only a subset of the information available in most Windows and Java applications. To map your COBOL fields to tables that are more meaningful and useful to the Windows/Java application, you may need to add additional information to your XFD. To add information to your XFDs, or to change the default names of the existing fields, you must use *directives*, or optional comments, in the FD section of your COBOL program. These directives are explained in **Section 3.3**.

Note: If you create your XFDs with a data storage option, (for example, "ccbl 32 -Dci -Fa animals.cbl"), you must specify the "-s" switch to **xdbcutil** when loading your system catalog with your XFDs. See Chapter 5, **section 5.3.1** for more information.

3.2.2 How XFDs Are Formed

XFDs enable the AcuXDBC interface to populate tables in a system catalog that map COBOL records in the indexed, relative, or binary sequential file to "rows" and "columns" of data that can be accessed by common SQL commands.

In the database table, each *column* contains the values for one *data item*. The column names are essentially the field names. The table that is built is *based on the largest record in the COBOL file*, and contains the fields from that record, plus any key fields (key fields are those that are named in KEY IS phrases of SELECT statements in the FILE CONTROL section). This ensures that data from all COBOL records fits within the table, and simplifies the storage and retrieval process. If you were to examine the database columns, only the fields from the largest record, and the key fields, would appear.

Note: If the field named in the KEY IS phrase is a group item, it does not become a column in the XFD table. Instead, the elementary items subordinate to the named group item each become a column. You can force a group item to be a column by using the USE GROUP directive, described in **section 3.3.13**.

With multiple record formats (level 01), not all COBOL fields are represented in the database by name, but all fields *are* storable and retrievable. The XFD maps fields from all records of a file to the corresponding locations in the "master" (largest) record of the file, and thus to the Vision database table. Only the fields included in the XFD will be available to ODBC or JDBC applications. Because AcuXDBC has access to the table holding information from the XFD, it "knows" where the data from a given COBOL record fits in the database tables. This activity is invisible to the ODBC or JDBC application.

To include multiple record definitions in the XFD, you can use the WHEN directive (**section 3.3.15**). Alternatively, you can create views of your data using the SQL command, CREATE VIEW (**section 7.5.4**).

Caution: *Each field or group of fields in your COBOL FD or XFD field must correspond to an SQL column name.* To ensure that this is the case, you may need to add field names to the XFD using the NAME directive.

To help you determine whether any fields need to be added, the next section describes which fields are automatically included and excluded.

3.2.3 Defaults Used in XFD Files

There are several elements of COBOL file descriptors that require special handling when XFDs are built. These include multiple record definitions, REDEFINES, FILLER, and OCCURS. This section describes how ACUCOBOL-GT handles each of these situations.

As described earlier, in many instances you can override the default behavior described below by placing *directives* in the FDs of your COBOL code. Directives are described in **section 3.3**. For example, the WHEN directive allows you to use multiple definitions for a single set of data by specifying when each definition should be used.

Like most data sources, AcuXDBC does not support the notion of multiple definitions for the same column. As the following paragraphs explain, whenever a COBOL program gives more than one definition for the same data, the compiler makes a choice about which definition to use in the XFD. Then it disregards the rest.

KEY IS phrase

Fields named in KEY IS phrases of SELECT statements are included as column names in the XFD. Other fields that occupy the same areas as the key fields (by either explicit or implicit redefinition) are not included by name, but are mapped to the key field column names by the XFD.

Remember, if the field named in the KEY IS phrase is a group item, it will not be included in the XFD unless a USE GROUP directive is used (see **section 3.3.13**.).

REDEFINES clause

Fields contained in a redefining item occupy the same positions as the fields being redefined. The compiler needs to select only one of the field definitions to use. The default rule that it follows is to use the fields in the item being redefined as column names; fields that appear subordinate to a REDEFINES clause are mapped to column names in the Vision database using the XFD.

Multiple record definitions

This same rule extends to multiple record definitions. In COBOL, multiple record definitions are redefinitions of the entire record area. This leads to the same complication that is encountered with REDEFINES: multiple definitions for the same data. So the compiler needs to select one definition to use.

Because the multiple record types can be different sizes, the compiler must use the largest one, so that it can cover all of the fields adequately. Thus, the compiler's rule is to use the fields in the largest record defined for the file. If more than one record is of the largest size, the compiler uses the first one.

Group items

Note that group items are, by default, never included in an XFD for the same reason that REDEFINES are excluded: they result in multiple names for the same data items, and they are shown in the XFD file as a comment. You can, however, choose to combine grouped fields into one data item by specifying the USE GROUP directive, described in **section 3.3.13**.

FILLER data items

In a COBOL FD, FILLER data items are place holders. FILLER items are not uniquely named and thus cannot be uniquely referenced. For this reason, they are not placed into the XFD. The XFD maintains the correct mapping of the other fields, and no COBOL record positional information is lost.

If your ODBC or JDBC application needs to refer to information contained in a FILLER data item, you may need to include it. In such a case, you could include it under a USE GROUP directive or give it a name of its own with the NAME directive, described in **section 3.3.9**.

OCCURS clauses

An OCCURS clause requires special handling, because the ACUCOBOL-GT compiler must assign a unique name to each database column. The compiler accomplishes this by appending sequential index numbers to the item named in the OCCURS.

For example, if the following were part of a file's description:

```
03  employee-table occurs 20 times.
    05  employee-number     pic 9(3)
```

these column names would be created in the Vision database table, that is, the "table" of Vision data created when AcuXDBC performs its translation:

```
employee_number_1
employee_number_2
.
.
.
employee_number_10
employee_number_11
.
.
.
employee_number_20
```

You can use the SUBTABLE directive to modify this behavior, resulting in the compiler storing just the base name along with name of the subtable specified by the directive. See **Section 3.3.12** for information on this directive.

Note that the hyphens in the COBOL code are translated to underscores in database field names, and the index number is preceded by an underscore.

Identical field names

In COBOL you distinguish fields with identical names by qualification. For example, there are two fields named MONTH in the following code, but they can be qualified by their group items. Thus, you would reference MONTH OF LAST_VISIT and MONTH OF LAST_PAYMENT in your program:

```
10  last_visit.
    15  month               pic 99.
    15  day                 pic 99.
    15  year                pic 99.
10  last_payment.
    15  month               pic 99.
    15  day                 pic 99.
    15  year                pic 99.
```

However, database systems consider duplicate names an error. Thus, if more than one field in a particular file has the same name, you receive a compile warning, and an ".xfd" file is not generated.

The solution to this situation is to add a NAME directive that associates an alternate name with one or both of the conflicting fields. (See **section 3.3.9** for more information.)

3.2.4 Examples of Default Mapping

The following section includes examples of how XFDs are formed.

If your program has one file with the three records shown below, the underlined fields are included in the XFD by default (this example assumes that **ar-codes-key** is named in a KEY IS phrase). Some fields do not appear in the XFD, but the XFD maps them to the "master" field names. The interface thus eliminates redundancies and gives you optimum performance.

Note: In the following example, the ship-weight and ship-instruct fields would be invisible to any ODBC, JDBC, or SQL application, unless you use a directive to make the fields visible.

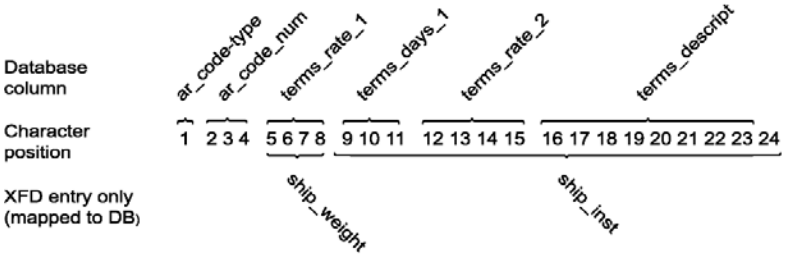
```
01  ar-codes-record.
    03  ar-codes-key.
        05  ar-code-type      pic x.
        05  ar-code-num       pic 999.
```

These fields are included because they are the key.

```
01  ship-code-record.
    03  filler              pic x(4).
    03  ship-weight         pic s999v9.
    03  ship-instruct       pic x(15).
01  terms-code-record.
    03  filler              pic x(4).
    03  terms-rate-1        pic s9v999.
    03  terms-days-1        pic 9(3).
    03  terms-rate-2        pic s9v999.
    03  terms-descript      pic x(15).
```

These fields are included because they comprise the largest record.

The diagram below shows how AcuXDBC identifies database columns for some of the fields in the COBOL record, while other fields are mapped to those columns by the XFD; this means that all the fields are accessible to the ODBC/JDBC program.



Note: If your application reads a record that should be a ship-code-record, the data that is displayed is unreadable. If you try to modify that record, you may receive an error message.

3.2.5 Summary of XFD Fields

Fields defined with an OCCURS clause are assigned unique sequential names. Fields without names are disregarded. When multiple fields occupy the same area, the compiler chooses only one of them unless you have a WHEN directive to distinguish them. To choose among multiple fields occupying the same area, the compiler:

- preserves fields mentioned in KEY IS phrases.
- discards group items unless USE GROUP is specified.
- discards REDEFINES.
- uses the largest record if there are multiple record definitions.

3.2.6 Naming the XFD File

The compiler must give a name to each XFD file that is built. The compiler attempts to build the name from your COBOL code, but in some instances the name in the code is nonspecific, and you must provide a name.

Each ".xfd" name is built from a *starting name* that is derived (if possible) from the SELECT statement in your COBOL code. The following paragraphs explain how that occurs.

ASSIGN name is a variable

If the SELECT for the file has a variable ASSIGN name (ASSIGN TO *filename*), you must specify a starting name for the ".xfd" file via a FILE directive in your code.

ASSIGN name is a constant

If the SELECT for the file has a constant ASSIGN name (such as ASSIGN TO "COMPFILE"), that name is used as the starting name for the ".xfd" file.

ASSIGN name is generic

If the ASSIGN phrase refers to a generic device (such as ASSIGN TO "DISK"), the compiler uses the SELECT name as the starting name.

Forming the final XFD name

The final name is formed from the starting name as follows:

1. The compiler removes any extensions from the starting name.

2. It constructs a “universal” base name by stripping out directory information that fits any of the formats used by the operating systems that run ACUCOBOL-GT.
3. It converts the base name to lower case.
4. It appends the extension “.xfd” to the base name.

Examples of XFD names

COBOL code	File name
ASSIGN TO "usr/ar/customer.dat"	customer.xfd
SELECT TESTFILE, ASSIGN TO DISK	testfile.xfd
ASSIGN TO "-D SYS$LIB:HELP"	help.xfd

3.2.7 How AcuXDBC Locates XFD Files

You provide the path to your XFDs on the command line when first populating your system catalog using **xdbcutil** (see **Chapter 5, section 5.3, “Creating a System Catalog and Views.”**) or addfile (see **Chapter 5, section 5.5, “Loading the System Catalog with Your XFDs.”**). Once your XFDs are used by **xdbcutil**, AcuXDBC no longer needs them. Unlike older versions of AcuODBC, AcuXDBC does not require you to distribute your XFDs to end users. Your XFDs are completely expendable unless they are being used by another product, such as Acu4GL.

3.3 Using Directives

XFDs may be built directly from your source code with no directives if the compiler’s default mapping rules are sufficient for your situation. If you would like to override the default mapping behavior, or map a field to a different name, you can add *directives* to your COBOL code.

Directives are optional comments that you can place into an FD in your COBOL source code to control how XFDs are built. By controlling how XFDs are built, you can guide the way AcuXDBC will map the content of your Vision files' columns in the tables that comprise your database.

Among other things, directives enable you to

- specify a column name to be used in place of a COBOL field name.
- map elementary items of a group item together into a single column.
- cause the fields from a specific record in a file to appear in the database table (rather than just the fields from the largest record).
- give a name to the XFD file itself.

Directives are always placed within a COBOL FD. They do *not* affect Procedure Division I/O statements, and they do *not* change your COBOL data in any way.

Note: *If you are going to access Vision from an ODBC- or JDBC-enabled application, each field in the application must correspond to a data item in your COBOL FD or XFD.* To ensure that this is the case, you may need to use the NAME directive to add fields to the XFD.

3.3.1 Sample Files and Examples

In addition to describing the directives you can include in your COBOL file's FD section, this section provides sample code and examples based on the sample files, "animals.cbl" and "file_dir.cbl" that come with AcuXDBC. (These files are located with the other files that make up the AcuXDBC installation.) These files were developed with ACUCOBOL-GT on a Windows 2000 system. The files were compiled with the following command:

```
ccbl32 –fa animals.cbl
```

After successful compilation, the Vision file was created with the following command:

```
wrun32 animals
```

The tables resulting from these files are displayed in Microsoft Access 2000. See **Chapter 8** for information on bringing external data into the database.

The following code is the FD for the source “animals.cbl” file. Note that this is a highly simplified example intended only to demonstrate the use of directives. You would not insert data into a table with this design.

```
FILE SECTION.
FD  jr-file.
01  jr-record.
    03  animal-info.
        05  patient-id                 pic x(5).
        05  atype                      pic x.
        05  ctype redefines atype      pic x.
        05  dtype redefines atype      pic x.
        05  otype redefines atype      pic x.
    03  owner-info.
        05  phone                      pic x(8).
        05  owner                      pic x(30).
    03  financial.
        05  acct_no.
            10  year                   pic x(2).
            10  seq_no                 pic x(4).
        05  last_visit.
            10  yyyy                   pic 9(4).
            10  mm                     pic 9(2).
            10  dd                     pic 9(2).
        05  fee                        pic s9(5)v99.
        05  date_paid                  pic 9(8).
```

With each directive, a new line is added to the code as shown in subsequent sections of this chapter. Examples in each section will show the code and also the resulting database table, where appropriate. If you are working on a different system, you may use different commands to compile “animals.cbl” and to create the Vision file. If you are using an application other than Microsoft Access 2000, the resulting tables may look different.

The following table was imported into Microsoft Access 2000 based on the FD above. You may want to note the changes as directives are added to the FD.

PATIENT	ATYPE	PHONE	OWNER	YEAR	SEQ_NO	YYYY	MM	DD	FEE	DATE_PAI
00001	C	555-0123	Robert Jones	85	5678	2002	5	21	60	2002060
00002	C	555-7777	Hartt Knox	96	1370	2002	4	21	105.55	2002051
00018	D	555-0904	Robert Michaels	1	3174	2002	6	8	13.25	2002060
00054	C	555-8832	Sidney Leonard	0	7143	2002	2	11	133.45	2002021
00102	D	555-0145	Ellen Smith	94	9012	2002	6	17	200	2002072
00160	D	555-6219	Neil Braziel	91	5247	1998	7	6	55.25	1998070
00328	D	555-9921	Marcia Adams	2	4319	2002	3	14	28.7	2002031
00378	C	555-1011	Alix Parker	88	6002	1999	10	22	80	1999111
00480	T	555-1303	Buddy Green	97	5430	2002	1	29	98.25	2002022
00503	B	555-0167	Sam Tucker	95	9876	2002	5	8	90	2002050
00801	R	555-0198	Bob Hernandez	99	1234	2002	5	23	25.75	2002052
00802	R	555-0198	Bob Hernandez	0	2234	2002	2	7	17.5	2002020

The file “file_dir.cbl” is used only when describing the FILE directive. The sample source code is shown in **section 3.3.7, “FILE Directive.”**

3.3.2 Directive Syntax

Place each directive on a line by itself, immediately before the COBOL line to which it pertains.

Introduce each directive with a “$” in the Indicator Area (column 7 in standard ANSI source format), followed immediately by the letters “XFD”, and then the directive itself. There must be no space between the $ and the XFD. Spaces are permitted elsewhere on the line as separators. For example, the NAME directive looks like this:

```
$XFD  NAME = COLOR
```

An alternate ANSI-compliant way to introduce a directive is with an asterisk (*****) in the Indicator Area. In this case, you begin the directive with the letters “XFD” and enclose the entire directive in double parentheses. For example:

```
*(( XFD  NAME = COLOR ))
```

There must be no space between the asterisk and the double left parentheses. Spaces are permitted elsewhere on the line as separators.

You may use either form of the directive syntax (or both) in your applications.

Two or more directives that pertain to the same line of COBOL code may be combined on one comment line. The directives should be separated by a space or a comma. For example, to specify both USE GROUP and NUMERIC at the same time, you would add this line:

```
$XFD USE GROUP, NUMERIC
```

or

```
*(( XFD USE GROUP, NUMERIC ))
```

The following sections describe each of the directives that AcuXDBC supports, in alphabetical order. These include the:

- ALPHA Directive
- BINARY Directive
- **COMMENT Directive**
- **DATE Directive**
- **FILE Directive**
- **HIDDEN Directive**
- **NAME Directive**
- **NUMERIC Directive**
- **READ-ONLY Directive**
- **USE GROUP Directive**
- VarLength Directive
- **WHEN Directive**
- **XSL Directive**

3.3.3 ALPHA Directive

The ALPHA directive allows you to treat a data item as alphanumeric text in the database, when it is declared as numeric in the COBOL program.

Syntax

```
$XFD ALPHA
```

or

```
*(( XFD ALPHA ))
```

This is especially useful when you have numeric keys in which you occasionally store non-numeric data, such as LOW-VALUES or special codes. In this situation, treating the field as alphanumeric allows you to move any kind of data to it.

Example

Suppose you have specified KEY IS code-key. Then assume the following record definition:

```
01  code-record.
    03  code-key.
        05  code-num      pic 9(5).
```

In the database, group items are disregarded, so CODE-NUM is the actual key field. Suppose you needed to move a non-numeric value to the key:

```
MOVE "C0531" TO CODE-KEY.
WRITE CODE-RECORD.
```

In this case the results are not well-defined because a non-numeric value has been moved into a numeric field. The database might very well reject the record.

One way to solve this problem is to use the ALPHA directive. This causes the corresponding database field to accept alphanumeric data:

```
 01  code-record.
     03  code-key.
$XFD ALPHA
         05  code-num      pic 9(5).
```

As an alternative, you could specify the **USE GROUP Directive** on the line before *code-key.* The USE GROUP directive implies that the field is alphanumeric.

3.3.4 BINARY Directive

The BINARY directive is used to specify that the data in the field could be alphanumeric data of any classification. Absolutely any data is allowed.

The BINARY directive may not be used in combination with the **VAR_LENGTH Directive** directive.

The method of storing fields declared as binary is database-specific. For example, with Informix databases, binary data is stored in char fields with an extra leading character. This character always contains a space. Oracle uses the data type raw for the field.

Syntax

```
$XFD BINARY
```

or

```
*(( XFD BINARY ))
```

Example

You might use this directive when you need to store a key that contains LOW-VALUES; COBOL allows a numeric field to contain LOW or HIGH values, but these are invalid for a numeric field in the RDBMS:

```
 01  code-record.
     03  code-key.
         05  code-indic          pic x.
*(( XFD BINARY ))
         05  code-num            pic 9(5).
         05  code-suffix         pic x(3).
             .
             .
             .
 move low-values to code-num.
```

3.3.5 COMMENT Directive

The COMMENT directive allows you to include comments in an XFD file. Because the information is embedded in a comment, it does not interfere with processing by AcuXDBC. Each comment entered with this directive appears in the XFD file with the symbol "#" in column 1. This information will only appear in the XFD file. It will not be imported into the system catalog by XDBCUtil.

Syntax

```
$XFD COMMENT text
```

or

```
*(( XFD COMMENT text ))
```

For example, the following FD includes a comment:

```
       FILE SECTION.
       FD  qa-file.
      $XFD COMMENT  This sample file demonstrates directives.
       01  qa-record.
           03  animal-info.
               05  patient-id                  pic x(5).
               05  atype                       pic x.
               05  ctype redefines atype       pic x.
               05  dtype redefines atype       pic x.
               05  otype redefines atype       pic x.
           03  owner-info.
               05  phone                       pic x(8).
               05  owner                       pic x(30).
           03  financial.
               05  acct_no.
                   10  year                    pic x(2).
                   10  seq_no                  pic x(4).
               05  last_visit.
                   10  yyyy                    pic 9(4).
                   10  mm                      pic 9(2).
                   10  dd                      pic 9(2).
               05  fee                         pic s9(5)v99.
               05  date_paid                   pic 9(8).
```

The following line appears in the "animals.xfd" file:

```
<!--  This sample file demonstrates directives. -->
```

3.3.6 DATE Directive

The DATE directive creates a map between the ODBC/JDBC application date fields and COBOL numeric fields. Because there's no COBOL syntax that identifies a field as a date, you should add this directive to differentiate dates from other numbers. This way, when a user of a Windows or Java application requests date information, AcuXDBC can respond properly.

Syntax

```
$XFD DATE[=date-format-string optional]
```

or

```
*(( XFD DATE[=date-format-string optional] ))
```

The DATE directive implies the NUMERIC directive.

The *date-format-string* is a description of the desired date format, composed of characters from the following list:

Y	year (2 or 4-digit)
M	month (01 - 12)
D	day of month (01 - 31)
J	Julian day (00000000 - 99999999)
E	day of year (001 - 366)
H	hour (00 - 23)
N	minute
S	second
T	hundredth of a second

If *date-format-string* contains only date characters (Y, M, D, E or J), the string is treated as a DATE variable. If *date-format-string* contains only time characters (H, N, S), it is treated as a TIME variable. If *date-format-string* contains any combination of date and time characters, or if it contains fractional seconds (T), it is treated as a TIMESTAMP variable.

Note: Some applications have difficulty processing a *date-format-string* that contains fractional seconds (for example: HHNNSSTT). If you are using such an application, modify strings containing fractions to be purely numeric.

Each character in a *date-format-string* can be considered a place holder that represents the type of information stored at that location. The characters also determine how many digits will be used for each type of data. Any other characters cause the *date-format-string* to be invalid. Invalid formats are automatically treated as numeric data.

For example, although you would typically represent the month with two digits, if you specify MMM as part of your date format, the resulting date uses three digits for the month, left-zero-filling the value. If the month is given as M, the resulting date uses a single digit, and truncates on the left.

If you don't specify a *date-format*-string, the default is YYMMDD if the field has six digits, or YYYYMMDD if the field has eight digits.

It is sometimes desirable to have incomplete date types, for example to have YYYYMM to simply store the year and month. AcuXDBC defaults the month and day to 1, so that incomplete types will be valid. Note that AcuXDBC can handle incomplete date types, but if you have a complete type and invalid date data, AcuXDBC cannot handle the date.

Julian dates

Because the definition of Julian day varies, the DATE directive offers a great deal of flexibility for representing Julian dates. Many source books define the Julian day as the day of the year, with January 1st being 001, January 2nd being 002, and so forth. If you want to use this definition for Julian day, simply use EEE (day of year) in your date formats.

Other reference books define the Julian day as the number of days since some specific "base date" in the past. This definition is represented in the DATE directive with the letter J (for example, a six-digit date field would be preceded with the directive "$XFD DATE=JJJJJJ"). The default "base date" for this form of Julian date is January 1, 1900.

You can define your own base date for Julian date calculations by setting the JULIAN_BASE_DATE configuration variable in your AcuXDBC configuration file. See Chapter 4, **section 4.2.10** for details.

Using group items

You may place the DATE directive in front of a group item, so long as you also use the USE GROUP directive. For more information, see **section 3.3.13**.

Examples

The source code for the "animals" table contains a group item for the date of the animal's last visit, and an elementary item for the date of the last payment:

```
05  last_visit.
    10  yyyy                    pic 9(4).
    10  mm                      pic 9(2).
    10  dd                      pic 9(2).
05  fee                         pic s9(5)v99.
05  date_paid                   pic 9(8).
```

The date portions of a database table based on source code with this FD look similar to this. All of these date-related fields are of type NUMBER in the (Access) database.

YYYY	MM	DD	FEE	DATE_PAID
2002	5	21	60	20020602
2002	4	21	105.55	20020517
2002	6	8	13.25	20020608
2002	6	17	200	20020721
2002	5	8	90	20020508
2002	2	11	133.45	20020215
1998	7	6	55.25	19980706
2002	1	29	98.25	20020220
2002	3	14	28.7	20020314
1999	10	22	80	19991115
2002	5	23	25.75	20020523
2002	2	7	17.5	20020207

The examples that follow build on this source code.

Example 1 – Elementary data item, DATE directive

The “date_paid” field is defined in the sample file as an elementary item of type NUMBER. Inserting the DATE directive before this line maps the type to DATE/TIME (in Access) and changes the format of the date in the table, as shown below.

```
 FILE SECTION.
 FD  qa-file.
$XFD COMMENT  This sample file demonstrates directives.
 01  qa-record.
     03  animal-info.
         05  patient-id                 pic x(5).
         05  atype                      pic x.
         05  ctype redefines atype      pic x.
         05  dtype redefines atype      pic x.
         05  otype redefines atype      pic x.
     03  owner-info.
         05  phone                      pic x(8).
         05  owner                      pic x(30).
     03  financial.
         05  acct_no.
             10  year                   pic x(2).
             10  seq_no                 pic x(4).
         05  last_visit.
             10  yyyy                   pic 9(4).
             10  mm                     pic 9(2).
             10  dd                     pic 9(2).
         05  fee                        pic s9(5)v99.
$XFD DATE=YYYYMMDD
         05  date_paid                  pic 9(8).
```

The resulting entries in a database table look similar to the following.

YYYY	MM	DD	FEE	DATE_PAID
2002	5	21	60	6/2/2002
2002	4	21	105.55	5/17/2002
2002	6	8	13.25	6/8/2002
2002	6	17	200	7/21/2002
2002	5	8	90	5/8/2002
2002	2	11	133.45	2/15/2002
1998	7	6	55.25	7/6/1998
2002	1	29	98.25	2/20/2002
2002	3	14	28.7	3/14/2002
1999	10	22	80	11/15/1999
2002	5	23	25.75	5/23/2002
2002	2	7	17.5	2/7/2002

Example 2 – Group data item, DATE and USE GROUP directives

If your date information is defined as a group item, you must use both the DATE and USE GROUP directives to map your COBOL numeric data items to SQL date fields. Insert the directives on the line preceding the group item.

```
 FILE SECTION.
 FD  qa-file.
$XFD COMMENT  This sample file demonstrates directives.
 01  qa-record.
     03  animal-info.
         05  patient-id                pic x(5).
         05  atype                     pic x.
         05  ctype redefines atype     pic x.
         05  dtype redefines atype     pic x.
         05  otype redefines atype     pic x.
     03  owner-info.
         05  phone                     pic x(8).
         05  owner                     pic x(30).
     03  financial.
         05  acct_no.
             10  year                  pic x(2).
             10  seq_no                pic x(4).
*(( XFD DATE=YYYYMMDD, USE GROUP ))
         05  last_visit.
             10  yyyy                  pic 9(4).
             10  mm                    pic 9(2).
             10  dd                    pic 9(2).
         05  fee                       pic s9(5)v99.
```

```
$XFD DATE=YYYYMMDD
         05  date_paid                    pic 9(8).
```

The resulting table now has a column with the name of the group item ("last_visit"), which is defined as type DATE/TIME (in Access).

YEAR	SEQ_NO	LAST_VISIT	FEE	DATE_PAID
85	5678	5/21/2002	60	6/2/2002
96	1370	4/21/2002	105.55	5/17/2002
1	3174	6/8/2002	13.25	6/8/2002
94	9012	6/17/2002	200	7/21/2002
95	9876	5/8/2002	90	5/8/2002
0	7143	2/11/2002	133.45	2/15/2002
91	5247	7/6/1998	55.25	7/6/1998
97	5430	1/29/2002	98.25	2/20/2002
2	4319	3/14/2002	28.7	3/14/2002
88	6002	10/22/1999	80	11/15/1999
99	1234	5/23/2002	25.75	5/23/2002
0	2234	2/7/2002	17.5	2/7/2002

FY and RY date formats

The FY and RY date format characters have very precise requirements and are used to handle a specific case where eight-digit dates are expressed in six characters.

Instead of YYYY or YY, you can specify FY to mean that the first character of the year specifies the decade instead of the "tens" year. The decade can be a character between space (" ") and "I" (inclusive). For the characters "0" through "9" to be treated as decades in the 20th century, the decade characters have the following meaning:

	00	**10**	**20**	**30**	**40**	**50**	**60**	**70**	**80**	**90**
1700					spc	!	"	#	$	%
1800	&	'	(	)	*	+	,	-	.	/
1900	0	1	2	3	4	5	6	7	8	9
2000	:	;	<	=	>	?	@	A	B	C
2010	D	E	F	G	H	I				

This means that a date of ?70210 is converted to 20570210, or February 10, 2057. The range of valid dates is 00101 (Jan 1, 1740) through I91231 (Dec 31, 2159).

Instead of YYYY or YY, you can also specify RY to mean that the first character of the year specifies the decade and that the entire date is to be 9s complement (to be able to sort dates in reverse order). Note that the entire date and time is treated as a 9s complement number in this case. The decade characters have the following meaning:

	00	**10**	**20**	**30**	**40**	**50**	**60**	**70**	**80**	**90**
1700					I	H	G	F	E	D
1800	C	B	A	@	?	>	=	<	;	:
1900	9	8	7	6	5	4	3	2	1	0
2000	/	.	-	,	+	*	)	(	'	&
2010	%	$	#	“	!	spc				

The date 20570210 is now specified with *29789. The range of valid dates is I99898 (Jan 1, 1740) through 08768 (Dec 31, 2159), the same valid range as when using F instead of R.

While using F and R before month, day, hour, or any other format specifier does not generate a compile error for the XFD, the results are undefined at runtime.

Note that if you use these format specifiers with Acu4GL, the actual date is written to the database, not the encoded date. That means that the R specifier is not very useful in this scenario (you won’t be able to read forward through a file in reverse date order). This format specifier is more useful with AcuXDBC where the data is in Vision format.

3.3.7 FILE Directive

The FILE directive supplies a *starting name* from which the XFD file name is formed. This directive is required only when the file name in the COBOL code is nonspecific. For example, you would use this directive when the

SELECT for the file has a variable ASSIGN name (ASSIGN TO *variable_name*). In this situation, the interface cannot form a file name automatically, and you must provide a name.

You can encounter this situation when you are using AcuXDBC's file alias feature. For example, the description section of your COBOL file may reference multiple data files with the same format. Each of these data files represents a distinct table in the database, but one XFD file describes them all. See Chapter 5, **section 5.5.1** "File Aliases" for more information.

A starting name is a short file name that serves as the basis for the XFD name. See **section 3.3.9, "NAME Directive,"** for additional information.

Syntax

```
$XFD FILE=name
```

or

```
*(( XFD FILE=name ))
```

This directive must appear on the line immediately preceding the file's FD.

Example

The sample file "file_dir.cbl" contains code that demonstrates the FILE directive. Suppose your SELECT statement has a variable ASSIGN name such as the one shown here:

```
SELECT work-file
     ASSIGN to pet-file.
```

You must add the FILE directive as shown here:

```
       FILE-CONTROL.
           SELECT work-file
              ASSIGN TO pet-file
              ORGANIZATION IS INDEXED
              ACCESS IS DYNAMIC
              RECORD KEY IS type-id
              FILE STATUS IS qa-file-status.

       DATA DIVISION.
       FILE SECTION.
```

```
$XFD FILE=patients
 FD  work-file.
 01  pet-record.
     05  type-id.
         10  atype            pic x.
         10  ano              pic 99.
     05  owner                pic x(30).
     05  breed                pic x(25).
     05  gender               pic x.
 WORKING-STORAGE SECTION.
 01  pet-file                 pic x(8).
 01  qa-file-status           pic xx.
```

Note that after compilation, the data directory now contains the files “file_dir.acu” and “patients.xfd”. Because the FILE directive assigns the name of the XFD file, this is different than in previous examples where both the “.acu” file and the “.xfd” file took the same base name as the “.cbl” file.

Refer to Chapter 5, **section 5.5.1** for information on file alias and how to load XFDs to different tables and databases using the system catalog utility program (**XDBCUtil**).

3.3.8 HIDDEN Directive

The HIDDEN directive allows you to hide specific data items from end users, while providing normal access to other data items. When you place this directive immediately before a data item in an FD, that item is hidden from end users. Field names affected by the HIDDEN directive are literally not placed into the system catalog. The user can neither read, modify, nor even know that this field exists.

Note: An alternative is to create a VIEW of the table that does not include the field. To do so, use the CREATE VIEW command, described in Chapter 7, **section 7.5.4**.

The HIDDEN directive applies only to the elementary data item it precedes. Subsequent data items revert to normal read access. Note that you cannot apply the HIDDEN directive to group items and you cannot hide the key.

If you use the INSERT command on a table that has hidden fields, spaces (hex value 20) are put into the hidden fields, whether they are numeric or alphanumeric.

Syntax

```
$XFD HIDDEN
```

or

```
*(( XFD HIDDEN ))
```

In the following FD, “fee” is hidden from users in the resulting table.

```
       FILE SECTION.
       FD  jr-file.
      $XFD COMMENT  This sample file demonstrates directives.
       01  jr-record.
           03  animal-info.
               05  patient-id                  pic x(5).
               05  atype                       pic x.
               05  ctype redefines atype       pic x.
               05  dtype redefines atype       pic x.
               05  otype redefines atype       pic x.
           03  owner-info.
               05  phone                       pic x(8).
               05  owner                       pic x(30).
           03  financial.
               05  acct_no.
                   10  year                    pic x(2).
                   10  seq_no                  pic x(4).
      *(( XFD DATE=YYYYMMDD, USE GROUP ))
               05  last_visit.
                   10  yyyy                    pic 9(4).
                   10  mm                      pic 9(2).
                   10  dd                      pic 9(2).
      $XFD HIDDEN
               05  fee                         pic s9(5)v99.
      $XFD DATE=YYYYMMDD
               05  date_paid                   pic 9(8).
```

Note that in the resulting table, the “fee” column no longer appears:

OWNER	YEAR	SEQ_NO	LAST_VISIT	DATE_PAID
Robert Jones	85	5678	5/21/2002	6/2/2002
Hartt Knox	96	1370	4/21/2002	5/17/2002
Robert Michaels	1	3174	6/8/2002	6/8/2002
Ellen Smith	94	9012	6/17/2002	7/21/2002
Sam Tucker	95	9876	5/8/2002	5/8/2002
Sidney Leonard	0	7143	2/11/2002	2/15/2002
Neil Braziel	91	5247	7/6/1998	7/6/1998
Buddy Green	97	5430	1/29/2002	2/20/2002
Marcia Adams	2	4319	3/14/2002	3/14/2002
Alix Parker	88	6002	10/22/1999	11/15/1999
Bob Hernandez	99	1234	5/23/2002	5/23/2002
Bob Hernandez	0	2234	2/7/2002	2/7/2002

This directive is useful for hiding data like passwords, telephone numbers, and financial information—whatever information you do not want users to see.

You cannot include a HIDDEN field in a WHEN directive with a TABLENAME clause, due to the complexities of editing or adding records. In this situation, you must add the data, but since the field is hidden, you cannot see it and could add a value that would cause unexpected results. See **section 3.3.15, “WHEN Directive,”** for additional information.

3.3.9 NAME Directive

The NAME directive assigns a database field name to the data item defined on the next line. You can use the NAME directive to prevent duplicate field names.

Syntax

```
$XFD NAME=fieldname
```

or

```
*(( XFD NAME=fieldname ))
```

This directive has several uses, as shown in the following examples.

Example 1 – duplicate names for data items in a single record

Within the Vision database, all field names must be unique. (Multiple database tables may include the same field name, but duplicates may not exist within a single table.) Unique field names are not required in COBOL, because names can be qualified by group items. For example, this is acceptable in COBOL:

```
10  last_visit.
     15  mm                  pic 99.
     15  dd                  pic 99.
     15  yy                  pic 99.
10  last_payment.
     15  mm                  pic 99.
     15  dd                  pic 99.
     15  yy                  pic 99.
```

It is not, however, acceptable for a database to have a table with two columns with the same name. In fact, you will not be able to compile and generate an XFD with this source code in your program.

You need not change the field names in your COBOL program to make them accessible to your Windows or Java application through ODBC or JDBC. Instead, you use the NAME directive to provide unique names for the fields.

Although this is not included in the sample code, here is an example of using the NAME directive when you have duplicate field names:

```
10  last_visit.
     15  mm                  pic 99.
     15  dd                  pic 99.
     15  yy                  pic 99.
10  last_payment.
$XFD NAME=MONTH_PD
     15  mm                  pic 99.
$XFD NAME=DAY_PD
     15  dd                  pic 99.
$XFD NAME=YEAR_PD
     15  yy                  pic 99.
```

The “dates” portion of the Vision database table will look like this:

mm	dd	yy	MONTH_PD	DAY_PD	YEAR_PD
06	18	02	07	04	02

Example 2 – assigning shorter names

You may want to use the NAME directive to assign shorter names than those used in your COBOL programs. This makes the formation of interactive SQL queries easier and quicker. For example:

```
 FILE SECTION.
 FD  jr-file.
$XFD COMMENT  This sample file demonstrates directives.
 01  jr-record.
     03  animal-info.
*(( XFD NAME=PATIENT ))
         05  patient-id                pic x(5).
         05  atype                     pic x.
         05  ctype redefines atype     pic x.
         05  dtype redefines atype     pic x.
         05  otype redefines atype     pic x.
     03  owner-info.
         05  phone                     pic x(8).
         05  owner                     pic x(30).
     03  financial.
         05  acct_no.
             10  year                  pic x(2).
             10  seq_no                pic x(4).
*(( XFD DATE=mmddyyyy, USE GROUP ))
         05  last_visit.
             10  mm                    pic 9(2).
             10  dd                    pic 9(2).
             10  yyyy                  pic 9(4).
$XFD HIDDEN
         05  fee                       pic s9(5)v99.
```

This directive causes the XFD to map PATIENT-ID to PATIENT in the database.

PATIENT	ATYPE	PHONE	OWNER	YEAR	SEQ_NO	LAST_VISIT	DATE_PAID
00001	C	555-0123	Robert Jones	85	5678	5/21/2002	6/2/2002
00002	C	555-7777	Hartt Knox	96	1370	4/21/2002	5/17/2002
00018	D	555-0904	Robert Michaels	1	3174	6/8/2002	6/8/2002
00102	D	555-0145	Ellen Smith	94	9012	6/17/2002	7/21/2002
00503	B	555-0167	Sam Tucker	95	9876	5/8/2002	5/8/2002
00054	C	555-8832	Sidney Leonard	0	7143	2/11/2002	2/15/2002
00160	D	555-6219	Neil Braziel	91	5247	7/6/1998	7/6/1998
00480	T	555-1303	Buddy Green	97	5430	1/29/2002	2/20/2002
00328	D	555-9921	Marcia Adams	2	4319	3/14/2002	3/14/2002
00378	C	555-1011	Alix Parker	88	6002	10/22/1999	11/15/1999
00801	R	555-0198	Bob Hernandez	99	1234	5/23/2002	5/23/2002
00802	R	555-0198	Bob Hernandez	0	2234	2/7/2002	2/7/2002

Additional examples

Here are some other places where you may consider using the NAME directive. In all of these instances, note that your COBOL data does not change. The new name appears only in the Vision database and in any tables that result from your COBOL code.

- If your COBOL data contains field names that are identical within the first 18 characters. Each time you compile your program and specify "-Fx" to create XFDs, any field names longer than 18 characters are checked for uniqueness within the first 18. If you have names that are identical within the first 18 characters, or that would not be meaningful if shortened to the first 18 characters, use the NAME directive to assign them different database field names.

- If you map COBOL data items to Windows or Java applications. If a column name in your Windows/Java application does not match the name used in your COBOL FD, you can use a NAME directive to associate the two names.

- If a field name in your COBOL data begins with a numeric character. Because Windows and Java applications communicate using SQL, the program typically generates a syntax error when it encounters a column name that begins with a numeric character. If your COBOL program uses field names that begin with a numeric character, use the NAME directive to assign a different name for use with your Windows and Java applications.

- If you want to include a “FILLER” item.
- If your COBOL name is an SQL reserved word.

3.3.10 NUMERIC Directive

The NUMERIC directive allows you to treat a data item as an unsigned integer when it is declared as alphanumeric. You might use this when the data stored in the item is always numeric.

Syntax

```
$XFD NUMERIC
```

or

```
*(( XFD NUMERIC ))
```

Example

The PATIENT-ID field (now appearing in the database table as the PATIENT column) is defined in the source code as being a text field. Since this field will always contain a number, you may want to define it in the database table as numeric. (Note the use of two directives for the same data item here.)

```
 FILE SECTION.
 FD  jr-file.
$XFD COMMENT  This sample file demonstrates directives.
 01  jr-record.
     03  animal-info.
*((XFD NAME=PATIENT, NUMERIC ))
         05  patient-id                pic x(5).
         05  atype                     pic x.
         05  ctype redefines atype     pic x.
         05  dtype redefines atype     pic x.
         05  otype redefines atype     pic x.
     03  owner-info.
         05  phone                     pic x(8).
         05  owner                     pic x(30).
     03  financial.
         05  acct_no.
             10  year                  pic x(2).
```

```
            10  seq_no                  pic x(4).
*(( XFD DATE=YYYYMMDD, USE GROUP ))
         05  last_visit.
             10  yyyy                   pic 9(4).
             10  mm                     pic 9(2).
             10  dd                     pic 9(2).
$XFD HIDDEN
         05  fee                        pic s9(5)v99.
```

The following table is the result, and, in the database, PATIENT is now of type NUMBER.

animals : Table

PATIENT	ATYPE	PHONE	OWNER	YEAR
1	C	555-0123	Robert Jones	85
2	C	555-7777	Hartt Knox	96
18	D	555-0904	Robert Michaels	1
102	D	555-0145	Ellen Smith	94
503	B	555-0167	Sam Tucker	95
54	C	555-8832	Sidney Leonard	0
160	D	555-6219	Neil Braziel	91
480	T	555-1303	Buddy Green	97
328	D	555-9921	Marcia Adams	2
378	C	555-1011	Alix Parker	88
801	R	555-0198	Bob Hernandez	99
802	R	555-0198	Bob Hernandez	0

See also

INVALID_NUMERIC_DATA

NULL_NUMERIC_READ

NULL_NUMERIC_WRITE

3.3.11 READ-ONLY Directive

The READ-ONLY directive allows you to make some fields (columns) of data read-only, while preserving normal access to other fields/columns. When placed immediately before a data item in an FD, this directive assigns the read-only attribute to the item. The read-only attribute applies only to elementary items, and not to group items. Subsequent data items return to normal read-write access.

Note: Use the READ-ONLY *directive* if you want to tag fields (columns) of data as read-only. Use the READ_ONLY *configuration variable* if you want to designate all the files belonging to a DSN as read-only. See Chapter 4, **section 4.2.23** for more information on this configuration variable.

Syntax

```
$XFD READ-ONLY
```

or

```
*(( XFD READ-ONLY ))
```

In the code creating the “animals” table, the “owner” data item has been designated as read-only. The code is:

```
       FILE SECTION.
       FD  jr-file.
      $XFD COMMENT  This sample file demonstrates directives.
       01  jr-record.
           03  animal-info.
      *((XFD NAME=PATIENT, NUMERIC ))
               05  patient-id                 pic x(5).
               05  atype                      pic x.
               05  ctype redefines atype      pic x.
               05  dtype redefines atype      pic x.
               05  otype redefines atype      pic x.
           03  owner-info.
               05  phone                      pic x(8).
      *(( XFD READ-ONLY ))
               05  owner                      pic x(30).
           03  financial.
               05  acct_no.
                   10  year                   pic x(2).
                   10  seq_no                 pic x(4).
      *(( XFD DATE=YYYYMMDD, USE GROUP ))
               05  last_visit.
                   10  yyyy                   pic 9(4).
                   10  mm                     pic 9(2).
                   10  dd                     pic 9(2).
      *(( XFD HIDDEN ))
               05  fee                        pic s9(5)v99.
      $XFD DATE=MMDDYYYY
               05  date_paid                  pic 9(8).
```

In the database table, the “owner” column looks the same as the other columns. However, if the user attempts to modify data in the owner column, they receive a message telling them that the update on a linked table failed.

See also

GRANT (Object privileges)

3.3.12 SUBTABLE Directive

This directive modifies the way fields that appear in an OCCURS clause are processed resulting in the creation of subtables. (See **section 3.2.3** for more information on the OCCURS clause). This directive instructs the XFD parsing routines not to append the occurrence number to the field name, as would normally take place with OCCURS clauses, but instead store just the base name along with the name of the subtable as written in the XFD file. The resulting tables will appear as multiple tables in a primary/foreign key relation based on the base table’s primary key.

The SUBTABLE directive is designed for AcuXDBC and cannot be used with Acu4GL.

Syntax

```
$XFD SUBTABLE=name
```

or

```
*(( XFD SUBTABLE=name ))
```

Example

```
*((XFD SUBTABLE=subtab1))
  03 employee-table occurs 10 times.
    05 employee-number           pic 9(3).
```

See also

USE GROUP Directive

3.3.13 USE GROUP Directive

Generally, only elementary data items correspond to columns. The USE GROUP directive indicates that the following group item is to correspond to a column as if it were an elementary item of the same width. This is necessary if the item is stored in your database as a group, rather than as individual fields.

By default, the USE GROUP directive implies that the consolidated field is alphanumeric. If you want a numeric field, add the word "NUMERIC" at the end of the directive.

Grouping data items in this way is efficient if the groups are usually processed as units.

Syntax

```
$XFD USE GROUP
```

or

```
*(( XFD USE GROUP ))
```

Example

You might use this directive with fields like multi-part account numbers or department numbers, or keys that are referenced as a unit but not by their individual pieces. In the sample file, you can group the "year" and "seq_no" fields to make a single account number column ("acct_no") as shown below:

```
 FILE SECTION.
 FD  jr-file.
$XFD COMMENT  This sample file demonstrates directives.
 01  jr-record.
     03  animal-info.
*((XFD NAME=PATIENT, NUMERIC ))
         05  patient-id                pic x(5).
         05  atype                     pic x.
         05  ctype redefines atype     pic x.
         05  dtype redefines atype     pic x.
         05  otype redefines atype     pic x.
     03  owner-info.
         05  phone                     pic x(8).
```

```
*(( XFD READ-ONLY ))
         05  owner                          pic x(30).
     03  financial.
$XFD USE GROUP
         05  acct_no.
             10  year                       pic x(2).
             10  seq_no                     pic x(4).
*(( XFD, USE GROUP, DATE=YYYYMMDD ))
         05  last_visit.
             10  yyyy                       pic 9(4).
             10  mm                         pic 9(2).
             10  dd                         pic 9(2).
$XFD HIDDEN
         05  fee                            pic s9(5)v99.
```

The resulting table looks similar to this:

animals : Table

PATIENT	ATYPE	PHONE	OWNER	ACCT_NO	LAST_VISIT	DATE_PAID
1	C	555-0123	Robert Jones	855678	5/21/2002	6/2/2002
2	C	555-7777	Hartt Knox	961370	4/21/2002	5/17/2002
18	D	555-0904	Robert Michaels	013174	6/8/2002	6/8/2002
102	D	555-0145	Ellen Smith	949012	6/17/2002	7/21/2002
503	B	555-0167	Sam Tucker	959876	5/8/2002	5/8/2002
54	C	555-8832	Sidney Leonard	007143	2/11/2002	2/15/2002
160	D	555-6219	Neil Braziel	915247	7/6/1998	7/6/1998
480	T	555-1303	Buddy Green	975430	1/29/2002	2/20/2002
328	D	555-9921	Marcia Adams	024319	3/14/2002	3/14/2002
378	C	555-1011	Alix Parker	886002	10/22/1999	11/15/1999
801	R	555-0198	Bob Hernandez	991234	5/23/2002	5/23/2002
802	R	555-0198	Bob Hernandez	002234	2/7/2002	2/7/2002

If you are using an existing database in which certain fields are grouped, they must also be grouped in your COBOL FD.

If the database does not yet exist, keep in mind that combining fields into groups typically improves execution speed. Whether to group fields or not also depends on how you want to process them. Do you always store and use the fields together? Someone who really knows how the data is being used might help to identify groups of fields that can be combined to speed processing.

Once you've grouped fields, you can apply other directives, such as **DATE Directive**, **NUMERIC Directive**, and **ALPHA Directive**.

3.3.14 VAR_LENGTH Directive

By default, the compiler generates fixed-length fields in the XFD. The VAR_LENGTH directive requests that the data item that immediately follows the directive be assigned a type that implies variable length, if possible. This can save considerable space in your database.

The VAR_LENGTH directive cannot be used in combination with the **BINARY Directive**.

The precise variable type that is assigned to the data item depends on which RDBMS is in use. Possible variable types that might be assigned are VARCHAR and VARBINARY.

Syntax

```
$XFD VAR_LENGTH
```

or

```
*(( XFD VAR_LENGTH ))
```

Example

For example, the directive in the following code indicates that the employee-name field should be entered into a SYBASE database as a VARCHAR data item.

```
 *(( XFD  VAR_LENGTH ))
      03  employee-notes      pic x(300).
```

3.3.15 WHEN Directive

Use the WHEN directive when you want to include multiple record definitions or REDEFINES in the XFD. The WHEN directive is typically used to force certain columns of data to be available that wouldn't be available otherwise.

Note: You cannot use the WHEN directive in an OCCURS clause.

Recall that the key fields and the fields from the *largest* record are automatically included as explicit columns in the Vision database table (see **section 3.2.2**). So you should use the WHEN directive if you want the user to be able to access all the data in the COBOL file in a way that is understandable.

WHEN declares that the field (or subordinate fields, if it is a group item) that immediately follow the directive *must* appear as a column (or columns) in the Vision database table. It also states one condition under which the columns are to be used. The WHEN directive thus guarantees that the fields will be explicitly included in the table (as long as they aren't FILLER and don't occupy the same area as key fields).

Syntax

```
$XFD WHEN field operator value
```

or

```
*(( XFD WHEN field operator value ))
```

Field is the name of a data item that corresponds to a field. If there is a NAME directive for this data item, the name used in the WHEN directive is the name given to the item by the NAME directive, not its COBOL name.

The operator specifies the relation between the field value and the alphanumeric literal that satisfies the condition. *Operator* can be one of the following:

Operator	**Relation**
=	The field value is equal to the literal value.
!=	The field value is not equal to the literal value.
>	The field value is greater than the literal value.
<	The field value is less than the literal value.
>=	The field value is greater than or equal to the literal value
<=	The field value is less than or equal to the literal value

Value is an alphanumeric literal (if the field is alphanumeric) or a numeric literal (if the field is numeric) or the special word “Other”. “Other” is used only with the “=” operator:

```
$XFD WHEN field = other
```

“Other” is true only when all other conditions for the same field are false. For example, if your FD contains the following lines of code:

```
$XFD WHEN ATYPE = "C"
$XFD WHEN ATYPE = "D"
$XFD WHEN ATYPE = other
```

the “other” condition holds true only if both “atype = ‘c’” and “atype = ‘d’” are false.

Example

The following code is an example of using the WHEN directive.

```
 01  key-record.
* employee-number is a key data item
     03  employee-number     pic 99999.
     03  emp-type            pic x.
$xfd when emp-type="1"
* record has this form when emp-type=″1″
 01  data-record-1.
     03  filler              pic 99999.
     03  filler              pic x.
     03  name-1              pic x(35).
     03  pay-rate-1          pic 99.99.
$xfd when emp-type="2"
* record has this form when emp-type=″2″
 01  data-record-2
     03  filler              pic 99999.
     03  filler              pic x.
     03  name-2              pic x(35).
     03  pay-rate-2          pic 999.99.
     03  subordinates        pic 999.
     03  position            pic x(50).
```

The effect of these directives is to force “emp-type”, “name-1”, and “pay-rate-1” to correspond to columns, even though they are not in the largest record description. Therefore, the corresponding table has the following columns:

EMPLOYEE_NUMBER
EMP_TYPE
NAME_1
PAY_RATE_1
NAME_2
PAY_RATE_2
SUBORDINATES
POSITION

If each data item is subordinate to at most one WHEN directive, as in this example, the following occurs:

- When the condition is true, the data item appears in the database table in the usual way.
- When the condition is false, the special value NULL appears in the corresponding column in the database table, and any value written into the column in the database table is *not* written to the COBOL data file. The exact meaning of a NULL value depends on the database. In some databases, NULL is a blank or zero value. In others, NULL is a special value on which no arithmetic or string operations can be performed, although a value can be tested to determine whether it is NULL.

If a data item is subordinate to two or more WHEN directives, the following applies:

- When *all* conditions are true, the data item appears in the database table in the usual way.
- When *at least one* condition is false, the special value NULL appears in the corresponding column in the database table, and any value written into the column in the database table is *not* written to the COBOL data file.

WHEN Directive With TABLENAME Clause

If you assign a tablename, a VIEW will be created that contains the columns subordinate to the WHEN directive and any columns the WHEN directive depends on.

When an XFD names a condition, such as WHEN, AcuXDBC produces multiple tables from a single XFD file. One table is given the current name of the file, while the VIEWS resulting from any named conditions are given the name specified with the tablename parameter (see syntax below).

You cannot include a HIDDEN field in a WHEN directive with a TABLENAME clause, due to the complexities of editing or adding records. In such a situation, you must add the data, but since the field is hidden, you cannot see it and may add a value that would cause unexpected results.

If an XFD file in an alias contains WHEN directives with TABLENAME phrases, the corresponding tables are defined in the usual way, using the data file specified by its physical file name. You can define two or more file aliases with the same XFD file but with different physical file names if the XFD file does not contain any WHEN directives with the TABLENAME clause.

Syntax

```
$XFD WHEN field operator value TABLENAME=new_table_name
or
*(( XFD WHEN field operator value TABLENAME=new_table_name ))
```

The syntax is essentially the same as for the WHEN directive alone, with the addition of the TABLENAME clause. The word “OTHER” can be used only with “=”. It means “use the following field(s) only if none of the other WHEN condition(s) listed for the same field is met.” In other words, this condition is true only if all other conditions for the same field are false.

For example:

```
.
.
assign to "ar_table"
.
.
01 ar-code-type.
*(( xfd  when ar-code-type = "s" tablename=ship ))
     03  ship-code-record     pic x(4).
*(( xfd  when ar-code-type = "b" tablename=backorder ))
     03  backorder-code-record redefines
         ship-code-record.
*(( xfd  when ar-code-type = other ))
     03  obsolete-code-record redefines
         ship-code-record.
```

If you tried to connect to the system catalog through a program like Access, you would see a table named “ar_table”, and two views named “ship”, and “backorder”. If you placed $XFD READ-ONLY TABLE immediately before the “xfd when ar-code-type = “s” tablename=ship” line, the ship view and ar_table would be read-only, but the backorder view would not. If you create the INFORMATION_SCHEMA during AcuXDBC setup (described in Chapter 5, **section 5.3**), you can see these entries by executing the following SQL query:

```
SELECT * FROM INFORMATION_SCHEMA.VIEWS;
```

OTHER may be used before one record definition, and may be used once at *each level* within each record definition.

Note: A WHEN directive with condition OTHER *must* be used if there is a possibility that the data in the field will not meet any of the explicit conditions specified in the other WHEN directives. If this is not done, results are undefined. Also, WHEN directives may ensure that there will be multiple columns that share the same record area. If you try to modify both columns, an error results.

Example 1

If the following code were compiled without directives, the underlined fields would appear explicitly in the database table. Note that the key fields are included automatically, as are the fields from the largest record. FILLER would be ignored:

```
01  ar-codes-record.
     03  ar-codes-key.
         05  ar-code-type  pic x.
         05  ar-code-num   pic 999.
01  ship-code-record.
     03  filler            pic x(4).
     03  ship-instruct     pic x(15).
01  terms-code-record.
     03  filler            pic x(4).
     03  terms-rate-1      pic s9v999.
     03  terms-days-1      pic 9(3).
     03  terms-rate-2      pic s9v999.
     03  terms-descript    pic x(15).
```

If you added the WHEN directive as shown below, it would cause the fields from the SHIP-CODE-RECORD to be included in the database table, and would determine when specific database columns would be used. The underlined fields would appear as columns in the database table:

```
 01  ar-codes-record.
     03  ar-codes-key.
         05  ar-code-type   pic x.
         05  ar-code-num    pic 999.
$xfd  when ar-code-type = "s"
 01  ship-code-record.
     03  filler             pic x(4).
     03  ship-instruct      pic x(15).
$xfd  when ar-code-type = "t"
 01  terms-code-record.
     03  filler             pic x(4).
     03  terms-rate-1       pic s9v999.
     03  terms-days-1       pic 9(3).
     03  terms-rate-2       pic s9v999.
     03  terms-descript     pic x(15).
```

FILLER data items don't have unique names and are thus not used to form columns in the database table. You could use the NAME directive to give them a name if you really need to see them in the database table. However, in this example the FILLER data items implicitly redefine key fields. Thus, they would be disregarded *even if you provided a name for them*.

Example 2

In the following code, in which no WHEN directives are used, the underlined fields will be explicitly named in the database table. (Key fields have the suffix "key" in their names in this example.)

Note that REDEFINES records simply re-map the same data area and are not explicitly included in the database table by default:

```
01  archive-record.
     03  filler                 pic x(33).
     03  archive-code           pic 9(6).
     03  archive-location       pic 9(2).
     03  filler                 pic x(10).
01  master-record.
     03  animal-id-key.
          05  patient-id        pic 9(6).
```

```
          05  species-code-type  pic 9(5).
          05  species-name       pic x(6).
     03  service-code-key.
          05  service-code-type  pic 9(6).
          05  service-name       pic x(10).
     03  billing-code.
          05  billing-code-type  pic 9(4).
          05  plan-name          pic x(8).
     03  office-info.
          05  date-in-office     pic 9(8).
          05  served-by-name     pic x(10).
     03  remote-info redefines office-info.
          05  van-id             pic 9(4).
          05  proc-code          pic 9(8).
          05  vet-name           pic x(6).
```

If you added the WHEN directives shown below, you would add several columns to the database table. The fields that would appear in the table are underlined:

```
*(( xfd when animal-id-key = "00000000000000000" ))
 01  archive-record.
     03  filler                  pic x(33).
     03  archive-code            pic 9(6).
     03  archive-location        pic 9(2).
     03  filler                  pic x(10).
*(( xfd  when animal-id-key =  other ))
 01  master-record.
*(( xfd  use group ))
     03  animal-id-key.
          05  patient-id         pic 9(6).
          05  species-code-type  pic 9(5).
          05  species-name       pic x(6).
     03  service-code-key.
          05  service-code-type  pic 9(6).
          05  service-name       pic x(10).
     03  billing-code.
          05  billing-code-type  pic 9(4).
          05  plan-name          pic x(8).
*(( xfd when billing-code-type = "1440" ))
     03  office-info.
          05  date-in-office     pic 9(8).
          05  served-by-name     pic x(10).
*(( xfd when billing-code-type = other ))
     03  remote-info redefines office-info.
```

```
05  van-id              pic 9(4).
05  proc-code           pic 9(8).
05  vet-name            pic x(6).
```

Example 3

If your application has a REDEFINES whose field names are more meaningful than the fields they redefine, you might consider switching the order of your code, rather than using a WHEN directive. Use the less significant field names in the REDEFINES.

For example, you might change this:

```
03  code-info.
     05  filler             pic 9(8).
     05  code-1             pic x(10).
03  patient-info redefines code-info.
     05  patient-id         pic 9(4).
     05  service-code       pic 9(8).
     05  server-name        pic x(6).
```

to this:

```
03  patient-info.
     05  patient-id         pic 9(4).
     05  service-code       pic 9(8).
     05  server-name        pic x(6).
03  code-info redefines patient-info.
     05  filler             pic 9(8).
     05  code-1             pic x(10).
```

The fields that would appear in the database table by default are underlined above. This shows how the column names might become more meaningful when the order is reversed. Your application operates the same either way.

Note: If a WHEN condition is false for a particular record, columns corresponding to data items subject to the WHEN directive and in the row corresponding to the record are set to the special database value NULL This means that there is no value provided for those columns. NULL is not equivalent to zero, and it has special properties in the database. For example, you can select all rows for which a given column is NULL.

Example 4

This COBOL code:

```
 01  col-type                          pic x.
     03  col-def.
$xfd  when col-type = "a"
          05  def1                     pic x(2).
$xfd  when col-type = "b"
          05  def2 redefines def1      pic 9(2).
```

results in this database table:

col_type	def1	def2
a	xx	null
b	null	10
a	yy	null

Note that if you try to set the first row so that col_type=a, def1=xx, and def2=20, the value of def2 is not stored.

3.3.16 XSL Directive

If you are using the "-Fxe" or "-Fae" compiler option to generate XML-style XFD files, the XSL directive allows you to include a stylesheet reference in the XML header.

Syntax

```
$XFD XSL=stylesheet
```

or

```
*(( XFD XSL="stylesheet" ))
```

where *stylesheet* is an alphanumeric literal indicating the appropriate stylesheet. The compiler includes the following line in all generated XML-style XFD files:

```
<?xml-stylesheet type="text/xsl" href="stylesheet"?>
```

For example:

```
*(( XFD XSL="myxsl.xsl" ))
```

generates this line:

```
<?xml-stylesheet type="text/xsl" href="myxsl.xsl"?>
```

4 Configuration

Key Topics

4.1 Introduction

Two configuration files are central to AcuXDBC. By default, they are known as “acuxdbc.cfg” and “net.ini”. In stand-alone configurations, only “acuxdbc.cfg” is used. Network installations using AcuXDBC Server require both configuration files. In this case, “acuxdbc.cfg” resides on the server and “net.ini” resides on the client.

Note: The “acuxdbc.cfg” file can be named anything. You specify the name of the file when you set up the DSN on the client or when using various command-line tools. “net.ini”, on the other hand, must retain its default name and location.

Configuration files provide you with two main benefits:

- Universal configuration. When creating DSNs on network clients, you don’t have to configure each DSN individually. Instead, just point to a configuration file on the server.
- Dynamic configuration. When you need to change your AcuXDBC configuration, you can modify the configuration file as needed without changing individual DSNs.

This chapter describes the purpose of these configuration files, and the variables that are supported.

4.2 AcuXDBC Configuration

AcuXDBC is configured through a configuration file, acuxdbc.cfg, located at “c:\Program Files\Acucorp\Acucbl8x\AcuGT\acuxdbc.cfg” on Windows or “/opt/acucorp/8x/acuxdbc.cfg” on UNIX, by default. This configuration file is where you define the location of your Vision data files and system catalog, file case instructions, and other important program functions. You can **create several different configuration files** for different purposes. Configuration files must reside in your AcuXDBC installation directory, specified by the environment variable (registry entry on Windows) GENESIS_HOME. You specify the exact name when setting up your DSNs on the client.

In stand-alone installations, this is the only configuration file that is required. In network installations, “acuxdbc.cfg” must reside on the server. “net.ini” is the configuration file used on network clients.

AcuXDBC supports configuration variables in several key areas, as shown below. AcuServer variables are described in **section 4.4**. The variables marked with an asterisk (*), DICTSOURCE and FILE_PREFIX, are required. If you installed AcuXDBC in a non-default directory, you must modify the DICTSOURCE and FILE_PREFIX variables in the client configuration file or generate a new configuration file with the genxconf utility, which is included in your distribution to make the generation of configuration files easier. This utility takes the format:

```
genxconf [-d directory] [-c catalog] [ -p prefix] [-n config_name]
```

This utility will generate a configuration file for you with default values for the configuration options unless you override them with command-line parameters. For you to be able to customize the generation settings for your site’s particular needs, most of the settings are read from a template called from your install directory called “sample\genxconf\acuxdbc.in”.

AcuXDBC Configuration Variables

Variable	Default	Description
General Setup Options		
DICTSOURCE*	\Acucorp\Acucbl8xx\acugt\syscat (Windows) /opt/acucorp/740/syscat (UNIX)	Location (path) of your system catalog
FILE_PREFIX*	none	Location (path) of your data files
FILE_SUFFIX	none	Extension of your data files
Advanced Options		
FILE_CASE	Default (Mixed)	Indicates the file case of your data files
FILENAME_WILDCARD	none	Define multi-company wildcards and their respective substitution characters.

AcuXDBC Configuration Variables

Variable	Default	Description
IGNORE_OWNER	0	Specifies whether you want the table owner to be included in the listing of tables that appears in some applications.
INVALID_NUMERIC_DATA	error	Indicates how to treat non-numeric data in a numeric field
JULIAN_BASE_DATE	1900/01/01	The base date for Julian date calculation
NULL_ALPHA_READ	null	Defines how to read null alphanumeric data
NULL_ALPHA_WRITE	spaces	Defines how to write null alphanumeric data
NULL_NUMERIC_READ	0	Defines how to read null numeric data
NULL_NUMERIC_WRITE	0	Defines how to write null numeric data
READ_ONLY	No	Read-write status of data files
Vision Options		
LOCKS_PER_FILE	10	The maximum number of record locks that can be held on a file when an SQL transaction has been issued
MAX_FILES	32	Maximum number of files that can be opened
MAX_LOCKS	32	The maximum number of record locks that can be held for all of the files together
V_BUFFERS	32	The number of indexed block buffers to allocate
Transaction Processing Options		
LOG_BUFFER_SIZE	512	The maximum buffer size, in bytes, for the transaction log file
LOG_ENCRYPT	off	Encrypts transaction log file
LOG_FILE	none	Name of transaction log file

AcuXDBC Configuration Variables

Variable	Default	Description
LOG_DEVICE	off	Assumes that the log file is actually a device, rather than a file
LOGGING	off	Enables transaction logging
TEMP_DIR	current directory	Directory to be used for holding the temporary files generated by the transaction management system
TRANSACTIONS	on	Enables transaction processing in the AcuXDBC interface.
TRANSACTION_PROCESSING	off	Enables Vision's transaction processing support.
Logging Setup Options		
DEBUG_LOGFILE	none	Name and location of the log file used for debugging purposes
DEBUG_LOGLEVEL	0	The log level you desire for debugging purposes
VISION_LOGGING_FILE	vision_trace.log	Sets name of Vision log file
VISION_LOGGING_LEVEL	0	Initiates Vision logging/tracing

For your convenience, a sample "acuxdbc.cfg" file is provided on the AcuXDBC distribution media. This file gets its information from a template file located in sample\AcuXDBC\acuxdbc.in in the install directory. **Section 4.2.30** shows the contents of this file. Most of the configuration options are commented out in this sample. To use an option, remove the comment hash sign (#) to apply a setting, and resave the file.

To assist you in creating your own configuration files, a utility called "genxconf" is included in the bin directory of "AcuGT". This utility automatically generates an AcuXDBC configuration file that includes the required configuration variables set to the values that you define. It also enables you to specify an alternate directory name for GENESIS_HOME and an alternate configuration file name. The genxconf.bat file takes the following commands:

```
genxconf [-d directory] [-c catalog] [-p prefix]
[-n config_name]
```

where:

Option	Specifies
"-d"	The directory name to use for the environment variable (registry entry on Windows) GENESIS_HOME. See **section 5.2** for more details on this variable/registry entry.
"-c"	An optional directory name to use for the System Catalog. See **section 4.2.3, "DICTSOURCE,"** for more details.
"-p"	A location (path) to your data files. See **Section 4.2.6, "FILE_PREFIX"** for more details.
"-n"	A name for the configuration file that is created.

For information on configuration files in general, refer to the *ACUCOBOL-GT User's Guide*, section 2.7, "Runtime Configuration."

4.2.1 DEBUG_LOGFILE

Use DEBUG_LOGFILE to specify the fully qualified name of the log file generated by the DEBUG_LOGLEVEL variable.

For example:

```
DEBUG_LOGFILE c:\logdir\xdbc.log
```

4.2.2 DEBUG_LOGLEVEL

Set DEBUG_LOGLEVEL to the level of logging that you need for debugging purposes. This variable has three valid values:

Value	Description
0	None. No log file is created. Logging is off.
1	Log control blocks/data. The log file contains information about the connection, the values in your configuration file, and the data from the table/file.
2	Log conversion output.

4.2.3 DICTSOURCE

Use this variable to specify the location of your system catalog. The system catalog is a required component of AcuXDBC and is created using the **xdbcutil** program or **ainit** script. Refer to Chapter 5, **section 5.3** for details.

Example:

```
DICTSOURCE c:\data\Dict
```

By default, DICTSOURCE is set to C:\Program Files\Acucorp\Acucbl8xx\acugt\syscat on Windows and /opt/acucorp/8xx/syscat on UNIX.

Use the FILE_PREFIX variable to specify the location of your Vision data files. Both DICTSOURCE and FILE_PREFIX are required.

4.2.4 FILE_CASE

Using this variable, set the case of the Vision filenames in your destination directory. By default, on Windows systems, the filenames are recognized in either upper, lower, or mixed case. If your files are on a UNIX system, using the default setting enables you to access files regardless of whether they are named in uppercase or lowercase letters.

If you set FILE_CASE to “Upper,” you can access only files that are named in uppercase letters. In the reverse, if you set FILE_CASE to “Lower,” you can access only files that are named in lowercase letters.

For example, setting “FILE_CASE to “Upper” means that AcuXDBC would find a file named “FILEA”, but it would not find a file named “FileA”. Setting FILE_CASE to “Lower” means that “filea” would be found, but “FileA” would not. Accepting the default value means that “FileA”, “filea”, and “FILEA” would all be found (case is ignored).

4.2.5 FILENAME_WILDCARD

Use this variable to define wildcards for multi-company DSNs. For example, a configuration file entry like this:

```
FILENAME_WILDCARD $$=01;**=02
```

tells AcuXDBC to substitute “01” for “$$” and “02” for “**” whenever the wildcard characters “$$”” and “**” are encountered in a configuration file, a “list.txt” file, or on the **xdbcutil** command line.

Wildcards can be any character *except* an equal sign (=), colon (:), semi-colon (;), alphabetic characters A-Z, and numerals 0-9. These are reserved characters.

Separate a wildcard from its substitution character by an equal sign. Separate multiple wildcards from one another by a semi-colon.

Refer to **Section 5.5.2, “Multi-company Support,”** for instructions on setting up multi-company DSNs.

4.2.6 FILE_PREFIX

Use this variable to specify the directories where AcuXDBC should search for data files. You can specify multiple locations for your data files. Until you specify a data directory, AcuXDBC will not let you log on.

You must prepend the line with a semi-colon, use double backslashes (“\\”) or single forward slashes (“/”) between directory names, and separate your paths by semi-colons. If any of your directories contain spaces, place the entire entry in quotation marks:

```
FILE_PREFIX “;c:\my dir1;c:\my dir2”
```

For best performance, list the directories in the order in which you want AcuXDBC to search them. For example:

```
FILE_PREFIX ;C:\\data\\data; \\XYZDomain\\data
```

or

```
FILE_PREFIX ;C:/data/data; /XYZDomain/data
```

Use the DICTSOURCE variable to specify the location of your system catalog. Both DICTSOURCE and FILE_PREFIX are required.

4.2.7 FILE_SUFFIX

Use this variable to specify the file extension used for your data files. For example

```
FILE_SUFFIX dat
```

When AcuXDBC looks for a file, it looks only at files with the extension specified here. If you do not include this variable in your configuration file, AcuXDBC looks for data files with no extension in the data directory indicated with FILE_PREFIX. You can always specify the exact filename with suffix when using the **xdbcutil** program or **addfile** script.

Note that files residing on remote UNIX servers must have lowercase file extensions.

4.2.8 IGNORE_OWNER

All tables in an AcuXDBC database have an owner. By default, the owner of the table is PUBLIC, or you can specify a different authorization ID when you add the file to the database with **xdbcutil**.

If an application such as Microsoft Access tries to link to the table, the table appears with the link table name of the form OWNER_NAME by default. To prevent the owner from being displayed:

1. Specify "-o " "" in **xdbcutil** to specify a blank owner when loading the table. Please note there is a space between the quotes after the “-o”, so it looks something like this on the command line:

   ```
   -o “<space>”
   ```

2. Set the IGNORE_OWNER variable to “1” (on, true, yes). When set to “1”, this variable removes the owner from the listing of the tables in applications that do catalog lookups. The default value is “0” (off, fasle, no).

To have no owner appear in applications that do table look up, you must perform both of these steps. See **section 5.3.1** for information on the **xdbcutil** utility.

As an alternative, you may choose the name of the linked/imported table in Access by using **Edit/Rename** or by right-clicking the table name and clicking “Rename” on the pop-up menu.

Tip: Having the link in the form USER_NAME is useful in distinguishing between separate objects with the same name.

4.2.9 INVALID_NUMERIC_DATA

Use this variable to indicate how AcuXDBC should treat non-numeric data in a numeric field.

If you specify...	AcuXDBC will...
Error (default)	Treat non-numeric data in a numeric field as an assignment error.
Truncate	Treat the first non-numeric character in a numeric field as the end of the numeric data (thus truncating the numeric data). This includes anything that is not a number, decimal, plus, or minus.
Zero	Cause the field to be returned to the application as “0” if the field contains any non-numeric data.

Caution: Be aware that users may inadvertently change data in a record. For example, if a particular record contains non-numeric data in a numeric field and the user changes the data while working in an ODBC-enabled application, the record in the source file changes to reflect what is now in the application. Therefore, exercise caution when setting either the INVALID_NUMERIC_DATA or READ_ONLY variables to a non-default value.

Some applications must read data based on the key. If a numeric column of a key field contains non-numeric data, the application will be unable to find the correct row of the file and may show an error. Note that, in particular, Microsoft Access selects the columns of the primary key and then selects the rest of the data based on those key columns. If Access cannot read the data (because a user operation has changed the data, and therefore the wrong data is in the key), the affected records show as DELETED. In this case, you must fix the data in the primary key.

4.2.10 JULIAN_BASE_DATE

A Julian date is the number of days since a certain, base date. For example, if the Julian base date is June 18, 2006 and today's date is July 4, 2006, the Julian date is 16. The default Julian base date in AcuXDBC is January 1, 1900.

To specify the date from which to start counting Julian dates, use the JULIAN_BASE_DATE variable. Specify a base date in the format "yyyy/mm/dd". For example:

```
JULIAN_BASE_DATE 2006/10/30
```

Note that you can enter any Julian Base Date, regardless of whether or not the date is valid. Also note that when AcuXDBC calculates Julian dates, the Julian base date becomes date 0. For example, if your Julian base date is October 30, 2006, the Julian dates are counted as follows:

October 30	Julian date 0
October 31	Julian date 1
November 1	Julian date 2
November 2	Julian date 3
November 3	Julian date 4

4.2.11 LOCKS_PER_FILE

The LOCKS_PER_FILE variable sets the maximum number of record locks that can be held on a file when an SQL transaction has been issued by the Windows application. This value affects only the files that are maintaining multiple record locks. Typically, all read records for updating or deleting transaction data are locked. This field enables you to increase the number of locks per file so you can manage large transactions. The default setting is "10". The maximum value is "8191" for Vision files. Setting this variable to its maximum value can waste resources and is not recommended.

4.2.12 LOG_BUFFER_SIZE

This variable sets the maximum buffer size, in bytes, for the transaction log file. Acceptable values are from "0" to "32767". Log Buffer Size is examined before each write to the log file. Its default value is "512". If Log Buffer Size is set to "0", writes to the log file are synchronous (unbuffered). That is, if the log buffer size is "0", there is an operating system call after every write to the log buffer. If the number is large, there is a call only when the buffer is full. We recommend that you experiment to determine which size works best in your environment.

4.2.13 LOG_DEVICE

Setting this variable to "1", (on, true, yes) causes the transaction management system to assume that the log file is actually a device, rather than a file. This means that a special device locking method is on the log file. It also guarantees that the log file is opened "append" and that no seeks are performed on it. This allows for the use of a tape device for the log file on many systems. By default, this option is turned off.

See also

LOG_ENCRYPT
LOG_FILE
LOGGING
LOG_BUFFER_SIZE

4.2.14 LOG_ENCRYPT

Set this variable to “1”, (on, true, yes) to encrypt records before writing them to the log file. By default, log encryption is off.

See also

LOG_FILE
LOGGING
LOG_BUFFER_SIZE
LOG_DEVICE

4.2.15 LOG_FILE

Use this variable to specify the name of the log file to be used for transaction management logging. (Activate transaction management logging using the ENABLE_LOGGING variable.) If the file does not exist, one is created.

You may store your transaction log file on a remote machine if your system utilizes AcuServer. To specify a remote filename for Log File, use remote name notation in this field. Note that AcuServer must be running on the remote machine.

Remote name notation has the following format:

```
@server-name:path-name/file-name
```

Log files prepended with *@server-name:path-name* are routed to AcuServer on the host specified by *server-name*, and stored in the directory specified by *path-name*.

See also

LOG_ENCRYPT
LOGGING
LOG_BUFFER_SIZE
LOG_DEVICE

4.2.16 LOGGING

Set this variable to “1”, (on, true, yes) to enable transaction management logging. When this box is selected, file updates are logged to the transaction log file specified in the LOG_FILE variable. Note that logging affects performance and should be used judiciously. By default, logging is off.

See also

LOG_FILE
LOG_ENCRYPT
LOG_BUFFER_SIZE
LOG_DEVICE

4.2.17 MAX_FILES

The MAX_FILES variable sets the maximum number of files that can be opened by AcuXDBC. The default is “32”. The maximum value is “256”. Keep this value small to conserve memory, but large enough to allow for 12 AcuXDBC system tables plus your data tables.

4.2.18 MAX_LOCKS

The MAX_LOCKS variable sets the maximum number of record locks that can be held by AcuXDBC for all of the files together. The default matches the setting of “Max Files,” in this case “32”. The maximum value is “8191” for Vision files. Setting this variable to its maximum value can waste resources.

4.2.19 NULL_ALPHA_READ

COBOL does not have a concept that corresponds directly to SQL’s NULL. The closest candidates in COBOL are data items that contain either SPACES or LOW-VALUES. In SQL, NULL is often used to indicate that the data is missing or not applicable.

To maintain the integrity of the source data and to ensure that any data written from your application back to the COBOL source is accurate, you must provide a representational mapping between COBOL's SPACES and LOW-VALUES and the corresponding SQL column values.

Use the NULL_ALPHA_READ variable to indicate which COBOL alphanumeric data should be represented as NULL. For data coming into your application (READs), if a field contains either SPACES or LOW-VALUES in alphanumeric data in the COBOL files, instruct the application to represent it as either NULL or an empty string. Valid values are:

```
NULL_ALPHA_READ null
NULL_ALPHA_READ empty
```

By default, alphanumeric data is interpreted as null on input to your ODBC-enabled application.

AcuXDBC follows these rules for alphanumeric data coming into an ODBC-enabled application.

If an alphanumeric field contains …	**Then that field …**
All SPACES or all LOW-VALUES	Comes into the application as either NULL or as an empty string, depending on the setting for variables for the data source.
Any other values	Comes into the application unchanged.

4.2.20 NULL_ALPHA_WRITE

See the **NULL_ALPHA_READ** for background on null processing in COBOL.

Use the NULL_ALPHA_WRITE variable to indicate how SQL NULLs in alphanumeric data should be translated into COBOL data. For data *returned* to the COBOL files (WRITEs), indicate whether the NULLs should be interpreted as SPACES or LOW-VALUES. Valid values are:

```
NULL_ALPHA_WRITE spaces
NULL_ALPHA_WRITE low-values
```

By default, alphanumeric data is interpreted as SPACES on output to the COBOL source.

Note: This variable is ignored if your data source is read-only.

4.2.21 NULL_NUMERIC_READ

See the **NULL_ALPHA_READ** for background on null processing in COBOL.

Use the NULL_NUMERIC_READ variable to indicate which COBOL numeric data should be represented as NULL. For data coming into your application (READs), if a field contains either SPACES or LOW-VALUES in numeric data in the COBOL files, instruct the application to represent it as either NULL or a zero. Valid values are:

```
NULL_NUMERIC_READ null
NULL_NUMERIC_READ 0
```

With numeric fields, LOW-VALUES and SPACES are valid values for many numeric types, as shown in the following examples:

If the numeric type is …	LOW-VALUES is equal to …	SPACES is equal to …
PIC 9(4) COMP-3	0	2020
PIC 9(2) COMP-5	0	8192
PIC 9(4) COMP-2	0	INVALID

Therefore, AcuXDBC follows these rules for numeric data coming in to the ODBC-enabled application:

If a field contains …	And …	Then …
All LOW-VALUES	LOW-VALUES is an invalid value for the numeric type	The field comes into the application based on the setting for numeric variables for the data source.

If a field contains …	And …	Then …
All SPACES	SPACES is an invalid value for the numeric type	The field comes into the application based on the setting for numeric variables for the data source.
Any other values	The value is invalid for the numeric type	Undefined data may come in to the application.

Note: SPACES or LOW-VALUES will not be converted to NULL or zero in a numeric data item where SPACES or LOW-VALUES, respectively, are valid numeric values.

By default, numeric data is interpreted as zero on input to your ODBC-enabled application.

4.2.22 NULL_NUMERIC_WRITE

See the **NULL_ALPHA_READ** for background on null processing in COBOL.

Use the NULL_NUMERIC_WRITE variable to indicate how SQL NULLs in numeric data should be translated into COBOL data. For data *returned* to the COBOL files (WRITEs), indicate whether the NULLs in numeric data should be interpreted as SPACES, LOW-VALUES, or zero.

Valid values are:

```
NULL_NUMERIC_WRITE spaces
NULL_NUMERIC_WRITE low-values
NULL_NUMERIC_WRITE 0
```

By default, numeric data is interpreted as zero on output to the COBOL source.

Note: This variable is ignored if your data source is read-only.

4.2.23 READ_ONLY

This variable establishes the default read/write permission for all the files belonging to the associated DSN. By default, this variable is set to "0", (off, false, no), indicating that users can write to the files. Specify "1", (on, true, yes)" if you want to make the files read-only.

What you indicate for read/write permission here applies to all files in the data source. If you want to assign different permissions to any individual files, you can do so using file aliases. See Chapter 5, **section 5.5.1** for more information on file aliases.

File (table) level read/write protection can also be defined by granting object privileges to specific users or to the PUBLIC user. This is done with the SQL GRANT statement as described in Chapter 7, **section 7.5.11**. For example, to make a table read-only to all users, you could issue the following command using **xdbcquery**:

```
GRANT SELECT ON TABLE1 TO PUBLIC
```

Field (column) level read/write protection can be further defined using the READ-ONLY directive. Refer to Chapter 3, **section 3.3.11** for more information on this option.

Note: When executing a SELECT statement, AcuXDBC opens Vision files as INPUT regardless of the how this variable is set. When the user attempts to update a row (from Microsoft Access, for example), if the READ_ONLY configuration variable is off, AcuXDBC attempts to open the Vision file for update. If the COBOL program has the record locked during an AcuXDBC SELECT and that record has met the SELECT conditions, a locked record condition occurs. When the COBOL program updates a record previously retrieved during an AcuXDBC SELECT, the update takes effect. When AcuXDBC attempts to update the record, an error occurs indicating that the record has been changed by another process after the original AcuXDBC SELECT.

4.2.24 TEMP_DIR

This variable allows you to specify a directory to be used for holding the temporary files generated by the transaction management system. If no directory is specified, temporary files are placed in the current directory.

4.2.25 TRANSACTIONS

This variable turns transaction processing support on (1, true, yes) or off (0, no, false) for the AcuXDBC interface. The default setting is “On” and results in AcuXDBC accepting or recoginizing Vision transaction commands. When set to “Off” AcuXDBC ignores the transaction commands.

4.2.26 TRANSACTION_PROCESSING

Use this variable to enable Vision’s transaction processing support in AcuXDBC. If you are in a transaction processing environment, set this variable to “1”, (on, true, yes) to begin transaction management, locking, and other key functions. By default, transaction processing is off. Use the **LOGGING** to turn on transaction logging.

4.2.27 V_BUFFERS

The V_BUFFERS variable lets you set the number of indexed block buffers to allocate. Each buffer is 512 bytes plus some overhead. These buffers are used to improve the performance of indexed files. The value can range from “0” (no buffering) to “256”. The default is “32”. Selecting a larger value generally improves file performance. Setting a lower value saves memory.

4.2.28 VISION_LOGGING_FILE

Use this variable to specify the name of the log file to be used for Vision logging. (Activate Vision logging using the VISION_LOGGING_LEVEL variable.) If the file does not exist, one is created.

You may store your Vision log file on a remote machine if your system utilizes AcuServer. To specify a remote filename for Log File, use remote name notation in this field. Note that AcuServer must be running on the remote machine.

Remote name notation has the following format:

```
@server-name:path-name/file-name
```

Log files prepended with *@server-name:path-name* are routed to AcuServer on the host specified by *server-name*, and stored in the directory specified by *path-name*.

4.2.29 VISION_LOGGING_LEVEL

Set this variable to a value between “1” and “9” to initiate Vision logging/tracing; the higher the number, the more detailed the trace file. The trace file contains all Vision file operations performed at runtime, such as file opens, reads, and writes. It is useful for troubleshooting file errors. By default, the file is named “vision_trace.log”.

A value of “0” or “-1” turns file tracing off.

4.2.30 Sample “acuxdbc.cfg” File

```
# This is a sample AcuXDBC configuration file.
# You should edit it to match your needs.

# The following lines are commented out to show you the default values.
# If you want to use a different value, then uncomment the line and change
# the value.

#-------------------------------------------------------------
#------------------GENERAL SETUP OPTIONS----------------------
#-------------------------------------------------------------

#  The path to your system catalog directory. This is a required variable

dictsource C:\Program Files\Acucorp\Acucbl810\acugt\syscat

#  The path to your data files.  You must prepend the line with a semi-colon,
#  use either double backslashes (“\\”) or forward slashes (“/”),
```

```
#  and separate your paths by semi-colons. This is a required variable

file_prefix ;C:/Program Files/Acucorp/Acucbl810/acugt/sample/AcuXDBC/data

#  Specify your data file extensions; leave blank if no
#  extensions exist.

# file_suffix

#-------------------------------------------------------------
#------------------ADVANCED OPTION SETUP--------------------
#-------------------------------------------------------------

#  Values for File_Case are Default (case ignored),
#  Lower (filename converted to lower case), and
#  Upper (filename converted to upper case).
# file_case         default

#  To specify how AcuXDBC will treat non-numeric values in numeric fields,
#  set this variable to Error (assignment error returned),
#  Truncate (value truncated from first non-numeric value to the end),
#  or Zero (value returned will be zero).

invalid_numeric_data    error
invalid_numeric_data    truncate
invalid_numeric_data    zero

#  Set this value to true if the user should have read-only
#  permissions to this database.  The default is read and write permissions.

# read_only       no

#  These two variables provide a representational mapping between COBOL's
#  SPACES and LOW-VALUES and the corresponding SQL column values.
#  Valid values are null and empty.

# null_alpha_read null

#  Valid values are spaces and low-values.

# null_alpha_write      spaces

#  Valid values are null and 0 (zero).

# null_numeric_read     0

#  Valid values are spaces, low-values, and 0 (zero).

# null_numeric_write    0
```

```
#  Enter a start date for Julian dates using the YYYYMMDD
#  format.  The default date is January 1, 1900.
# julian_base_date      19000101

# This variable when turned off will cause AcuXDBC to ignore the transaction
# options on operations such as an update. The default is on.
#
# transactions        on

# Tables in AcuXDBC have an owner specified. The owner is either public
# or an authorization id. The ignore_owner variable can be used to have
# files show as no owner. This must be used in conjunction with loading
# through xdbcutil with the -o " " option to have a blank owner in the
# database. The default is FALSE.
#
# ignore_owner       off

#-------------------------------------------------------------
#------------------VISION OPTION SETUP---------------------
#-------------------------------------------------------------
# Set Max_Files from 1 to 256.
# max_files       32

# Set Max_Locks from 1 to 8191.
# max_locks        32

# Set Locks_per_file from 1 to 8191.
# locks_per_file     10

# Set V_Buffers from 0 to 256.
# v_buffers      32

#-------------------------------------------------------------
#-----TRANSACTION PROCESSING SETUP OPTIONS------------------
#-------------------------------------------------------------

# transaction_processing Off
# logging off
# log_encrypt off
# log_device off
# log_file
# temp_dir .
# log_buffer_size 512

#-------------------------------------------------------------
#--------------ACUSERVER SETUP OPTIONS--------------------
#-------------------------------------------------------------
```

```
# acu_client_password
# acuserver_port 6532
# security_method none
# default_map_file

#-------------------------------------------------------------
#--------------------Multi-company options-----------------
#-------------------------------------------------------------

#  For multi-company handling, you can specify the value of any
#  wildcards that you used when creating the table in your
#  system catalog by noting the values here.
#
# filename_wildcards

#-------------------------------------------------------------
#--------------LOGGING SETUP OPTIONS----------------------
#-------------------------------------------------------------

# vision_logging_file
# vision_logging_level
# debug_logfile
# debug_loglevel

#-------------------------------------------------------------
#---------------END CONFIGURATION----------------------
#-------------------------------------------------------------
```

4.3 AcuXDBC Server Configuration

AcuXDBC Server requires two configuration files:

- “acuxdbc.cfg” on the server. This file is described in **section 4.2**.
- “net.ini” on the client. By default, AcuXDBC expects this file to be located at c:\Program Files\Acucorp\Acucbl8xx\AcuGT\lib.

  “net.ini” is where you define client-side settings for your network configuration.

For your convenience, a sample “net.ini” is provided on the AcuXDBC distribution media, in the “lib” subdirectory of the install. **Section 4.3.6** shows the contents of this file. Most of the configuration options are commented out in this sample. To use an option, remove the comment hash sign (#) to apply a setting, and resave the file.

AcuXDBC Server supports a variety of configuration variables, as shown below:

AcuXDBC Server Configuration Variables

Variable	Default	Description
KEY_CONNECT	1234	Used to encrypt the user ID and password sent over the network
PACKETSIZE	8192	Size of buffer used to aggregate send operations
READ_TIMEOUT	0	Time to wait for read completion
RETURN_ERRNO	no	Return the OS error code for communication errors
WRITE_TIMEOUT	0	Time to wait for write completion

4.3.1 KEY_CONNECT

Use the “net.ini” configuration variable, KEY_CONNECT, to specify the connect string masking key used to encrypt the OS and DBMS user ID and password information. This number must match the one given with the “-k” option when AcuXDBC Server is started. For example:

```
KEY_CONNECT 12345
```

4.3.2 PACKETSIZE

Use this variable to specify the size (in bytes) of the buffer used to aggregate send operations. The default is 8192. For example:

```
PACKETSIZE 1024
```

4.3.3 READ_TIMEOUT

Use this variable to specify the time to wait (in seconds) for read completion. The default of "0" indicates that there is no timeout.

```
READ_TIMEOUT 60
```

4.3.4 RETURN_ERRNO

When this variable is set to "1", (on, true, yes), AcuXDBC returns the operating system error code for communication errors. By default this variable is set to "0", (off, false, no).

```
RETURN_ERRNO yes
```

4.3.5 WRITE_TIMEOUT

Use this variable to specify the time to wait (in seconds) for write completion. The default of "0" indicates that there is no timeout.

```
WRITE_TIMEOUT 60
```

4.3.6 Sample "net.ini" File

```
#  net.ini file, used by AcuXDBC Server for client-side
#  configuration settings.  Note that the information
#  in the DSN takes precedence over the settings here.

#  This number must match the one given with
#  the -k option when AcuXDBC Server is started.
#  It is used to encrypt the user ID and password
#  information over the network.
#key_connect       1234

#  Size (in bytes) of the buffer used to aggregate send operations.
#packetsize        8192

#  Time to wait for read completion.  The default of 0 (zero) means no timeout.
#read_timeout      0

#  Time to wait for a write completion.  The default of 0 (zero) means no timeout.
#write_timeout     0

#  Return the OS error code for a communication error.
#return_errno      no
```

4.4 AcuServer Configuration

Network installations that use AcuServer rather than AcuXDBC Server require one configuration file: "acuxdbc.cfg" on the client.

Any AcuXDBC configuration variable described in **section 4.2** can be used in this file. The following variables are specific to AcuServer users. They apply only to network environments that use AcuServer for remote data access:

AcuServer Setup Options		
ACUSERVER_PASSWORD	none	Defines the password for connecting to AcuServer
ACUSERVER_PORT	6523	Specifies the port number on which AcuServer is listening
DEFAULT_MAP_FILE	none	Sets the name and path of the map file to be used to map special characters to their decimal or hexadecimal equivalent
SECURITY_METHOD	none	Sets the security method to use when accessing the AcuServer host

4.4.1 ACUSERVER_PASSWORD

Use this variable to specify the password assigned in AcuServer's server access file (AcuAccess file). AcuXDBC will use this password to access the AcuServer host.

4.4.2 ACUSERVER_PORT

Use this variable to specify the port number assigned to the AcuServer host if it is different from the default port number, "6523". This variable is useful for redirecting additional instances of AcuServer or for working around a firewall.

4.4.3 DEFAULT_MAP_FILE

Use this variable to set the name and path of the map file to be used (if any) to map special characters in your character set to their decimal or hexadecimal equivalent in another character set before they are passed to or from the Vision file system. This file lets you reconcile the character encoding between two machines that use different codes for the same characters.

You can specify a local or remote directory for the map file. To specify a remote directory, use the following syntax:

```
@server-name:directory-path
```

where *server-name* is the name of the UNIX or Windows server on which the map file resides.

When creating a map file, you need to re-map only those values that vary between the two character sets (e.g., vowels with a grave accent, acute accent, circumflex, tilde, etc.) You can check the values of specific characters using the Windows Character Map accessory in the PC environment, or by referring to your UNIX manual pages (man pages) in the UNIX environment.

The map file should contain two values per line: the first indicating the decimal or hexadecimal value of the special character on the client machine and the second indicating the decimal or hexadecimal value of the corresponding character on the server machine. (Hexadecimal values use the standard “0x” notation.) For instance:

```
0x90 0xC9
```

maps “É” (E acute) in the IBM PC character set to “É” (E acute) in the ISO8859-1 character set using hexadecimal notation.

```
144 201
```

gives the same mapping using decimal notation.

You can use the pound sign (“#”) to indicate a comment.

Note: The map will be used to translate only alphanumeric fields; but it will translate *all* alphanumeric fields, including group items and items subject to a REDEFINES clause. If this is not a desired behavior, you may need to restructure your program to avoid these clauses by passing the elementary items instead of the group item, or passing an item from the REDEFINES clause instead of the first reference.

4.4.4 SECURITY_METHOD

Use this variable to specify the security method to use when accessing the AcuServer host. AcuXDBC's network security methods are not used with AcuServer.

SECURITY_METHOD can take any of three values. The value specified here must match the value specified in the server configuration file on the AcuServer host. Valid values include:

NONE (false, no)

Do not use the native operating system security. Use AcuServer security instead.

LOGON

Use the system's native security to manage user logons.

NAMED-PIPE (on, true, yes)

Use Windows security based on the connection made from the client to the server via a named pipe.

5 Installing AcuXDBC

Key Topics

5.1 General Setup Procedures

The procedures for installing AcuXDBC in network and stand-alone environments are described in this section. They assume that you have already created your configuration file(s) as instructed in **Chapter 4**. To become familiar with the AcuXDBC software quickly, follow the quick-start procedure. If you just wish to run the sample demonstration database included with your distribution, you can generate a default configuration file with the command genxconf.bat (on Windows) or genxconf.sh (on UNIX).

5.1.1 Quick Start — Demo Application

1. Install AcuXDBC software, accepting the default options in the installation script.
2. Switch to the directory in which AcuXDBC is installed.
3. Make sure that you have a configuration file (**Chapter 4**).
4. Run a script called “demo.bat” (Windows) or “demo.sh” (UNIX). This script:
   a. Creates an empty system catalog, database system tables, and base views of the catalog.
   b. Loads the system catalog with information from XFDs from a sample veterinary office application.
5. You can now use the command-line query tool to retrieve some information from the sample database you have just created. Enter asql (Windows) or asql.sh (UNIX).
6. Click * from pets.

   This should retrieve the 19 records contained in the pets file:

```
SQL (/? for help) ==> select * from pets;

PATIENT_ID PATIENT_NAME ANIMAL_TYPE  BREED     TREATMENT  OWNER_ID
---------- ------------ ------------ ----------  --------- -------
 1         Cinnamon      Cat          Tabby             1      624
 2         Nutmeg        Cat          Tabby             2      550
 18        Shotzi        Dog          Schnauzer         1      704
```

```
36        Cinder        Dog         Poodle            4     221
54        Buster        Cat         Siamese           1     377
72        Missy         Bird        Parakeet          6     309
102       Kit           Dog         Shiba Inu         1     600
160       Milo          Dog         Chow              4     522
161       Puzzle        Reptile     Ball Python       3     522
328       Copper        Dog         Golden Retriever  4     618
377       Scooter       Cat         Domestic Shorthair 2    357
378       Scrapper      Cat         Devon Rex         2     357
379       Abbie         Cat         Maine Coon        2     357
480       Princess      Reptile      Iguana           1     309
503       Polly         Bird        Senegal Parrot    6     625
504       Diego         Bird        Red Lory          4     625
505       Alexi         Bird        African Grey      4     625
801       Hammy         Rodent      Gerbil            3     700
802       Rodney        Rodent      Hamster           4     700

SQL (/? for help) ==&gt;
```

7. You can exit the application by entering “/q” for quit.

5.1.2 Stand-alone Installations

If you will be accessing your Vision data locally, you do not require AcuXDBC Server or AcuServer. The following steps are all performed on the same machine:

1. **Install AcuXDBC software using the installation script found on your product distribution media**.
2. **Edit a configuration file, either by hand or with genxconf**.
3. Create a database:
   - For the demo, use the demo file located in the GENESIS_HOME directory.
   - For an empty database:

     (1) Use the script file **ainit**.

     (2) Run the executable **xdbcutil**:

     ```
     xdbcutil -c -d mydb -pa
     ```

4. If desired, create users.
5. **Load the system catalog** with information from your XFDs. (For details on this, see **section 5.3** and **section 5.5**.)
6. S**et permissions (for details, see section 5.6)**.
7. **Set up a Data Source Name** (DSN) for your Vision data. This step is not required if you plan to use a query tool to issue SQL queries and do not require ODBC integration.

5.1.3 AcuXDBC Server Installations

If you will be using AcuXDBC Server to process Vision data on the server, you will first need to set up and configure AcuXDBC on the server machine for local access. Once you have completed this configuration, you can start the server running and test client/server connections locally.

Server Procedure

1. **Install AcuXDBC Server software and set necessary environment variables**.
2. **Create an empty system catalog**, database system tables, and base views of the catalog.
3. If desired, define users and **grant database privileges** to them. If you do not perform this step, your database will be open to all users.
4. **Load the system catalog** with information from your XFDs.
5. If desired, **set permissions** on your newly created Vision tables.
6. **Start AcuXDBC Server**.

Windows Client Procedure

1. **Install AcuXDBC**.
2. **Set up a Data Source Name** (DSN) for your Vision data. This step is not required if you plan to use a query tool to issue SQL queries and do not require ODBC integration.

UNIX Client Procedure

1. **Install AcuXDBC**.
2. **Dowload an ODBC driver** manager for UNIX, such as unixODBC. (For details, see **section 5.2.2**.)
3. Add Data Source Name information to an **ODBC text file** according to the driver's instructions.
4. **Modify environment and configuration variables**.

5.1.4 AcuServer Installations

If you will be using AcuServer to process remote Vision data locally, do the following:

Server Procedure

1. Install AcuServer software on the server.
2. Create a server access file.
3. Start AcuServer.

See the *AcuServer User's Guide* for specifics.

Client Procedure

1. **Install AcuXDBC** software.
2. **Create an empty system catalog**, database system tables, and base views of the catalog.
3. If desired, create users and **grant database privileges** to them. If you do not perform this step, your database will be open to all users.
4. **Load the system catalog** with information from your XFDs.
5. If desired, **set permissions** on your newly created Vision tables.
6. **Set up a Data Source Name** (DSN) for your Vision data. This step is not required if you plan to use a query tool to issue SQL queries and do not require ODBC integration.

5.1.5 Using AcuXDBC

Once you have installed and configured AcuXDBC, you are ready to issue SQL queries to your Vision data. There are two ways to do this:

1. You can issue queries from ODBC-enabled applications like Microsoft Word, Excel, and Access or JDBC-enabled applications like Cold Fusion. Chapter 7 demonstrates how to do this.

2. You can issue queries from the command-line query tool, "xdbcquery.exe". Chapter 6 demonstrates how to do this.

5.2 Installing AcuXDBC/AcuXDBC Server

Following are instructions for installing AcuXDBC and AcuXDBC Server in **Windows** and **UNIX/Linux** environments.

5.2.1 Windows Installations

On Windows, run "setup.exe" on your product CD-ROM to install AcuXDBC on your machine. Select AcuXDBC from the product selection list as appropriate. You will be prompted for product codes and license keys during the installation. Have this information ready.

The installation process sets a registry entry known as GENESIS_HOME with the location for configuration files and the installation directory. The default installation directory is C:\Program Files\Acucorp\Acucbl8*xx*\AcuGT\ on Windows. Though you can have several different configuration files and refer to different ones in your DSNs, they must always reside in the directory pointed to by GENESIS_HOME.

Note: If you install AcuXDBC in a non-default directory, be sure to modify the DICTSOURCE and FILE_PREFIX variables in the client configuration file. See **section 4.2** for more information on these variables.

In network environments, install AcuXDBC on each client in the network, and install AcuXDBC Server or AcuServer on the server.

5.2.2 UNIX Installations

UNIX Server

1. Mount the installation CD-ROM for AcuXDBC Server.
2. Change to the mount directory.
3. Type "./install" and follow the instructions given from the command line.
4. Set an operating system environment variable called "GENESIS_HOME" to the root installation of AcuXDBC, /opt/acucorp/800 by default. The following is an example of how to set this variable:

   ```
   export GENESIS_HOME=/opt/acucorp/800
   ```

   Note: The value of the environment variable should not have a '/' as the last character, because this can interfere with installation and configuration scripts.

5. Set or modify environment variables that point to the location of the AcuXDBC executables and to tell the operating system where to find the AcuXDBC shared libraries. On Linux, use the PATH and LD_LIBRARY_PATH variables, for example.

   ```
   export PATH=/opt/acucorp/800/bin:$PATH
   export LD_LIBRARY_PATH=/opt/acucorp/800/bin:$LD_LIBRARY_PATH
   ```

UNIX Client

1. Mount the installation CD-ROM for AcuXDBC.
2. Change to the mount directory.
3. Type "./install" and follow the instructions given from the command line.
4. Download an ODBC driver for UNIX. We have tested with "unixODBC" from www.unixodbc.org/unixODBC-2.2.11.tar.gz.

5. Copy the “unixODBC*.tar.gz” file to a location where you have permission to create files and directories.

6. Type “gunzip unixODBC*.tar.gz”.

7. Type “tar xvf unixODBC*.tar”.

8. Read the “readme” file located in the directory where the package was extracted and any other “readme” files with a suffix that describes your operating system (for example, “readme.aix” for IBM.AIX). These directions are provided for a basic 32-bit Linux (Intel x86) installation.

9. Read the “install” file for prerequisites and installation instructions.

10. Type “configure” and wait while features on your system are being checked.

11. Type “make” and wait for the package to be compiled.

12. Type “make install” to install programs, data, and documentation.

13. Type “odbcinst -j” to find out where your SYSTEM and USER data sources are located. Here is the location information from a Linux system.

```
DRIVERS.............: /usr/local/etc/odbcinst.ini
SYSTEM DATA SOURCES: /usr/local/etc/odbc.ini
USER DATA SOURCES..: /home/techsup/.odbc.ini
```

14. Add the following to the file where your SYSTEM data sources are located (for example, /usr/local/etc/odbc.ini):

```
[ODBC Data Sources]
vision_sys = VORTEXodbc to VISION
[vision_sys]
Driver = /usr2/lib/acuxdbc.so
Description = VORTEXodbc to VISION
LoginID = system
```

15. In your /home/acucorp80/lib) create another odbc.ini file with the following information:

```
rem -------------- VORTEXodbc
fetch_buffer_size  8192        -- fetch buffer size (in bytes)
columns            256          -- max # of database columns
logical_cursors    1024         -- max # of logical cursors
db_cursors         64           -- max # of DB cursors
```

Then, if the information above is for use on a local machine, enter:

```
dsn_vision_sys      "acuxdbc04:%s/%s/xvision:acuxdbc.cfg"
```

If the information above is for use on a server, enter:

```
dsn_vision_sys     "acuxdbc03:%s/%s/xvision:acuxdbc.cfg@20222:servername!acuxdbc04
```

16. Set two operating environment variables called VORTEX_HOME and GENESIS_HOME to the root installation directory of AcuXDBC, /opt/acucorp/8xx/ by default. The following is an example of how to set these variables:

    ```
    export VORTEX_HOME=/opt/acucorp/800
    export GENESIS_HOME=/opt/acucorp/800
    ```

    Note: The value of the environment variable should not have a '/' as the last character, because this can interfere with installation and configuration scripts.

17. Set or modify environment variables that point to the location of the AcuXDBC executables and to tell the operating system where to find the AcuXDBC shared libraries. On Linux, use the PATH and LD_LIBRARY_PATH variables, for example:

    ```
    export PATH= /opt/acucorp/800/bin
    export LD_LIBRARY_PATH= /opt/acucorp/800/bin
    ```

18. Modify the AcuXDBC configuration file. A sample configuration file called "acuxdbc.cfg" is provided for your convenience. The configuration file must be located at $GENESIS_HOME. There are two variables that are required: DICTSOURCE and FILE_PREFIX. Other variables may be needed depending on your situation. Below is a partial sample of acuxdbc.cfg. showing the syntax for the two required variables. Note that this file also contains sample syntax and instructions for all available variables. If you want to use this file, you can remove the comment indicators from any of the other variables listed in acuxdbc.cfg.

    ```
    # The path to your system catalog directory
    DICTSOURCE /usr2/acuxdbc/syscat
    # The path to your data files. You must prepend the line with a
    semi-colon,
    # use either double backslashes ("\\") or forward slashes ("/"),
    # and separate your paths by semi-colons.
    FILE_PREFIX ;/usr2/acuxdbc/data;/usr2/acuxdbc/sample/acuxdbc/data
    ```

Note: For FILE_PREFIX, the delimiting character is a semi-colon or colon. Spaces are not supported in this AcuXDBC variable.

19. Verify that the AcuXDBC license file, “xvision.alc”, is located in the bin directory.
20. Run the setup script “demo.sh” located in the $GENESIS_HOME/bin directory.
21. Try to access a sample Vision file using a utility that came with unixODBC. To run the test, type “isql vision_sys system manager”.
22. A successful connection looks like this:

```
Connected!
sql-statement
help [tablename]
quit
```

To use the unixODBC driver

1. Set up a Data Source Name (DSN) for the veterinary data.
   a. From the Start menu, select **All Programs/Acucobol 800/ AcuXDBC/ODBC Data Source Administrator**. The ODBC Data Source Administrator appears.
   b. From the User DSN tab, click **Add**.
   c. From the Create New Data Source dialog box, select **Acucorp AcuXDBC Driver**, and then click **Finish**.
   d. On the AcuXDBC Setup: General Tab, complete the fields as follows:

Field	Description
Data Source Name	Enter “Vet Data”
Description (Optional)	Enter “Sample veterinary application data”
Network driver	Leave unchecked
Hostname	N/A
Port	N/A

Field	Description
Configuration File	Accept the default value
Username	System
Password	Manager
Confirm Password	Leave blank

2. Access the veterinary database from an ODBC compliant application as described in **Chapter 8**.

5.2.3 Providing JDBC Access

With AcuXDBC Enterprise Edition, you have access to Vision data from JDBC as well as ODBC. Below are the instructions for setting up a JDBC installation on UNIX:

1. Create a directory on your machine for the AcuXDBC JDBC files.
2. Put the downloaded ".tar" file in that directory and untar it.

   The directory now contains two subdirectories and a "vortex.jar" file:

   - ***sample*** — Sample java and jdbc programs.
   - ***html*** — HTML java and jdbc method descriptions.
3. Move the "vortex.jar" file into a directory pointed to by the Java JVM $CLASSPATH environment variable that is used for running the JDBC-enabled application. Alternatively you can add this directory to the $CLASSPATH environment variable:

C Shell

```
setenv CLASSPATH "$CLASSPATH":"/usr2/acuxdbc"
```

Bourne Shell

```
CLASSPATH=$CLASSPATH:/usr2/acuxdbc
export CLASSPATH
```

5.2.4 Installed Executables and Scripts/Shells

Here is a list of the executable components and batch files/shell scripts that are installed along with AcuXDBC:

File	Description
xdbcutil.exe / xdbcutil	System catalog tool
xdbcquery.exe / xdbcquery	Command-line SQL query tool
xdbcsrvr.exe / xdbcsrvr	AcuXDBC Server network daemon
vortex.jar	JDBC and Java class file
ainit.bat / ainit.sh	Creates system catalog and information-schema views
asql.bat / asql.sh	Starts **xdbcquery** tool
addfile.bat / addfile.sh	Adds named XFD file or file list to system catalog
acuxdbcs.bat / acuxdbcs.sh	Performs functions on **xdbcsrvr** server daemon. Four options: "-help" "-info" Pings daemon. "-kill" Kills daemon. "-start" Starts daemon
demo.bat / demo.sh	Creates system catalog and load it with sample veterinary data.
genxconf.bat / genxconf.sh	Generates an AcuXDBC configuration file that includes the required configuration variables set to the values that you define. It also enables you to specify an alternate directory name for GENESIS_HOME and an alternate configuration file name.

5.3 Creating a System Catalog and Views

If you ran “demo.bat” or “demo.sh”, you do not need to create a system catalog. One has been created for you. Look for an AcuGT/syscat/ directory to confirm whether a system catalog exists.

A system catalog is where database systems store schema metadata, such as information about tables and columns in your data directories, and internal bookkeeping information.

AcuXDBC includes a command-line utility, **xdbcutil**, to create a system catalog for your Vision files. **xdbcutil** reads your XFD files and creates a table, transforming your Vision file system into a Vision database. The system catalog itself is stored in the Vision database.

For your convenience, we have created a script that creates a system catalog and simultaneously generates information-schema *views* of the catalog. The views are “virtual tables” or reflections of the system tables and contain columns that describe your Vision data so that you can use SQL syntax that you’re used to using (standard SQL).

If you plan to port your system catalog to multiple hardware platforms, be aware of the byte order format for both the machine you created your system catalog on and the machine you are going to deploy on. The machine’s byte order format is commonly referred to as big endian or little endian. A system catalog created on a machine that uses little endian (Intel for example) will not be portable to a machine that uses big endian (Motorola for example). In this case, you need to create a system catalog once on each type of machine (big endian and little endian) and then port the appropriate catalog to the appropriate target machine.

An alternate method is to dynamically create and load the database on the target machine. In this case, you would send the XFD files to the end user site and then create an empty database and load the XFD files into it, either one at a time or in batch mode.

Note: This section describes how to create an *empty* system catalog. Once you have a catalog, you must populate it with information from your XFDs. This is described in **section 5.5**.

To create a system catalog using a script:

1. Change to the \AcuGT\bin directory where AcuXDBC or AcuXDBC Server was installed.
2. Type “ainit” and press **Enter**.

The script creates a series of system tables and views of those tables and places them in the \AcuGT\syscat\ directory by default. These are described in Chapter 6. If you did not accept the default catalog path, be sure to amend your COBOL configuration file to point to your system catalog. Do this using the DICTSOURCE variable. See Chapter 4, **section 4.2.3** for details.

To create a system catalog using the xdbcutil utility:

1. Change to the directory where you want your system catalog installed.
2. Enter the following command:

   ```
   xdbcutil -c
   ```

   If you plan to grant database and table privileges to individual users, use the “-pa” option as well to create a user/group catalog and an object permissions catalog.

   ```
   xdbcutil -c -pa
   ```

   The command syntax for **xdbcutil** is described in **section 5.3.1**.

Note: If you use the “-d” option to specify a catalog path, be sure to amend your COBOL configuration file to point to your system catalog. Do this using the DICTSOURCE variable. See Chapter 4, **section 4.2.3** for details.

5.3.1 **xdbcutil** Syntax

The **xdbcutil** command has this syntax:

```
xdbcutil [-l LOGFILE] [-d SYSCAT_PATH] [-x XFD_PATH] [-v] [-p] [-pa] [-n] [-s] <-c |
-a xfd1 [xfd2...] | -f FILE | -u xfd1 [xfd2...]>
```

Note: Use quotes or the short name if specifying a path that contains spaces. XFD filenames, when given, must be of the format:

```
xfilename[#[tablename][#filename]]
```

where *xfilename* is the XFD filename, *tablename* is the SQL tablename to use, and *filename* is the Vision data filename.

As shown in the table below, some of the command options are used when creating the system catalog. Others are used when loading the system catalog with XFDs.

Option	Description
For use when creating the system catalog (see **section 5.3**)	
-c	Required. Creates a new system catalog.
-d	Specifies a directory for your system catalog. If none is specified, the system catalog is stored in the current directory.
-n	Specifies do not overwrite user/group catalog if it exists.
-p	Creates a user/group catalog for granting database privileges.
-pa	Creates a user/group catalog *and* an object permissions catalog. A user/group catalog is for granting database privileges. An object permissions catalog is for setting permissions on your Vision tables. Use the GRANT statement to set permissions in these tables.
For use when loading the system catalog with XFDs (see **section 5.5**)	
-a	Adds an individual table definition from the specified XFD file. You can use either "-a" or "-f" to populate the system catalog with XFDs, but do not use both in the same operation.
-d	Specifies the location of your system catalog. Required if you are not in the system catalog directory when you issue the **xdbcutil** command.
-f	Adds table definitions from the specified file, which contains a list of XFD filenames. You can use either "-a" or "-f" to populate the system catalog with XFDs, but do not use both in the same operation.
-l	Defines log output file name.

Option	Description
-o	Specifies table ownership. This must be a valid authorization ID that was created with the GRANT SQL statement for database privileges (See **section 7.5.10, "GRANT (Database privileges)."**

-s	Says to include data storage method switch. Required if you are using XFDs created with a pre-8.0 version compiler, and used one of the data storage compiler options ("-Dci", "-Dcm", "-Dcn"...) when creating your XFDs.
	
	The "-s" command accepts these storage method arguments:
	
	`If you compiled with:  Use this xdbcutil switch:`
	`-Dca  -sa`
	`-Dci  -si`
	`-Dcm  -sm`
	`-Dcn  -sn`
	`-Dcv  -sv`
	`-Dcb  -sb`
	`-Dcr  -sr`

-u	Updates table definitions from the specified XFD file(s). Use this option if you have already loaded an XFD file into the system catalog but have modified it for some reason. Reloading a file with "-a" results in an error.
-v	Initiates a verbose reporting mode.
-x	Specifies an XFD file directory. Required if your XFD files are not in the current directory.

Note: All tables in an AcuXDBC database have an owner. By default, the owner of the table is PUBLIC, or you can specify a different authorization ID using the "-o" option to **xdbcutil**. If you do not want a table owner to be listed publicly in applications like Microsoft Access, specify "-o " "" and set the **IGNORE_OWNER** to "1" (on, true, yes) in your configuration file.

5.4 Granting Database Privileges

Access to objects can be secured on both the database and the object level with AcuXDBC. If you created your database using **xdbcutil** or **addfile** with the "-p" or the "-pa" options, you will have a database that supports user authorization IDs. (By default, **addfile** will create the database with the "-pa" option, providing both database-level privileges and object-level privileges.)

By default, the system will create a single user "system" with the password "manager." There is also a second special authorization ID called "public," which is used only in the context of object-level privileges. Any object owned by "public" is accessible by all authorization IDs. By default, when you load an XFD into AcuXDBC, the default owner of the object will be "public." You may override this by specifying the "-o" option to **xdbcutil** or the "-u" option to **addfile**.

Once you have created your database with database-level privileges, you will need to create additional user authorization IDs. This is done using SQL "GRANT" command from within the command-line query tool (**asql** or **xdbcquery**) or some other ODBC enabled query tool that will allow the input of SQL commands. (The GRANT statement is described in detail in **section 7.5.10**. The command-line query tool is described in **section 6.3.1**.)

If you wish, you can load the system catalog with your XFDs and send the loaded tables to your customer or end user. A site administrator can grant database and table permissions.

A second approach to setting up a new database is to create an SQL script file to create all of the user authorization IDs and object-level privileges. This script can then be run at the end-user site using the command-line query tool. For example:

```
mfsql -u system -p manager -r my_script.sql
```

where *my_script.sql* is a text file containing the SQL commands to perform. For an example of this, please refer to the "addfile" script, which executes the "cview.sql" script to create the information schema in new databases.

Note: Granting database privileges is optional; however, if you do not grant privileges, your database will be open to all users.

To grant database privileges:

1. Change to the \AcuGT\bin directory where AcuXDBC or AcuXDBC Server was installed.

2. You can modify these parameters from within the file itself to match your settings and then simply run the "asql.bat" file from the same directory where **xdbcquery** is installed. Alternatively, you can manually specify your own connection parameters from the command line by using the following syntax:

   ```
   asql -u system -p manager -c myconfigfile.cfg
   ```

   The following prompt appears:

   ```
   SQL (/? for help) ==>
   ```

3. Grant database administrator privileges to yourself. To do so, issue a GRANT statement such as:

   ```
   GRANT DBA TO system IDENTIFIED BY manager;
   ```

   Note: If you log out now, no one but the DBA will be able to connect. If you do not grant DBA privileges *before* creating any other users, your system will be unusable. Should this happen, you will need to recreate the system catalog as described in **section 5.3**.

4. Grant the privileges CONNECT and/or RESOURCE to each of your users. CONNECT lets grantees connect to the database when the password is correct. RESOURCE lets grantees create objects/tables in the database. (Granting DBA privileges implies both CONNECT and RESOURCE and lets grantees read or modify any table in the database.)

   For example, you might issue the following commands:

   ```
   GRANT CONNECT,RESOURCE TO jsmith IDENTIFIED BY password1;
   GRANT CONNECT TO sjones IDENTIFIED BY password2;
   ```

5. When you are done granting privileges, quit the query tool by entering the following:

   ```
   SQL (/? for help) ==> /q
   ```

See **section 7.5.10** for detailed information on the GRANT statement.

Once you have granted users database privileges, you can modify their object-level privileges as well.

To modify privileges for existing users:

Issue a GRANT statement without the IDENTIFIED BY clause. For example:

```
GRANT CONNECT TO jsmith;
```

5.5 Loading the System Catalog with Your XFDs

The **xdbcutil** utility can be used both to create a system catalog and to populate it with data. The **xdbcutil** command is very flexible, so you have many options for loading your XFDs into the system catalog. For your convenience, we have provided a script—addfile—for loading XFD files into your system catalog as well. This script will allow you to default many of the common options.

To load XFDs using a script:

1. Change to the \AcuGT\bin directory where AcuXDBC or AcuXDBC Server was installed.
2. Enter the following:

   ```
   addfile filename
   ```

where *filename* is the name of the XFD file to add to the system catalog. By default, the script assumes that your XFDs are stored in the current directory and that your system catalog is stored in AcuGT/syscat/. By default, the user PUBLIC is assumed.

If you stored your XFDs and/or system catalog elsewhere or if you want to specify an alternate user, you can edit the script or add command line options as follows:

```
addfile [-d syscat_path] [-x xfd_path] [-u user] filename
```

or

```
addfile [-d syscat_path] [-x xfd_path] [-u user] -f filelist
```

The second form assumes that you have created a text file using your favorite editor listing all of the XFD files that you want to load into the system catalog. When you create a list file, use just the base name of the XFD file; do not use the '.xfd' suffix. List the XFD files one per line. For example, you might create a file called "list.txt" containing:

```
animals
accounts
pets
```

In this case, your addfile command would be:

```
addfile –f list.txt
```

To load XFDs using the xdbcutil utility:

To load the system catalog with your XFDs, minimally, use the "-d" and "-a" options to **xdbcutil**, as shown below:

```
xdbcutil [-d SYSCAT_PATH] <-a xfd1 [xfd2...]>
```

For example:

```
xdbcutil -d c:\data\dict -a c:\data\data\animals
```

results in the system catalog in c:\data\dict being loaded with information from the XFD named "animals.xfd" found in c:\data\data.

Rather than prepending the XFD filenames with their path, however, you can specify the XFD directory with the "-x" option, as in:

```
xdbcutil -d c:\data\dict -x c:\data\data -a animals
```

This would have the same effect as the preceding example.

If desired, you can create a file with a list of XFD names, and then specify it with a "-f" option, along with the "-x" option to specify the directory. For example, this command:

```
xdbcutil -d c:\data\dict -x c:\data\data -f list.txt
```

and a "list.txt" that contains this:

```
animals
accounts
pets
```

results in a system catalog with table definitions from "animals.xfd", "accounts.xfd", and "pets.xfd".

To instruct AcuXDBC that your XFDs use the data storage value specified by the "-Dci" storage method, add "-si" to the **xdbcutil** command line, like this:

```
xdbcutil -d c:\data\dict -x c:\data\data -si -f list.txt
```

If you have several different XFD directories, you can create the list file with the XFD paths pre-pended. For example, this command:

```
xdbcutil -d c:\data\dict -f list.txt
```

and a "list.txt" that contains this:

```
c:\data\data\animals
c:\data\data2\accounts
c:\data\other\pets
```

results in a catalog with three table definitions, coming from XFDs in three separate XFD directories.

Note: **xdbcutil** command syntax is described in **section 5.3.1**.

5.5.1 Setting Up File Aliases

With AcuXDBC, you can define file aliases when adding XFDs to your system catalog with **xdbcutil**.

A file alias is a way to map data files, ".xfd" files, and SQL tables. It is used at runtime to dynamically reassign a file name that is referenced in an ASSIGN clause or, if the ASSIGN clause is a variable, from the FILE directive.

File aliases are useful when:

- A single COBOL file description references multiple physical data files with the same format, such as when the ASSIGN TO clause of a COBOL select statement is a variable. Each of these files represents a distinct table in the database.
- A single table name in SQL can refer to multiple physical COBOL data files.

These situations can easily occur in a large company, in which one COBOL file needs to point to more than one company. For example, Corporation A supports several other companies (say, company01 and company 02) and wants to use the same code to point to each company. Corporation A has named its COBOL file "company," so the XFD file would be called "company.xfd". Because Corporation A wants to use this same code for company01 and company02, it has set up the code to point to data files "company01.dat" and "company02.dat".

Let's also say that Corporation A wants to use a different name when referring to the data files from SQL (maybe the name is too long and/or cumbersome). No problem! Corporation A can specify all three components: the XFD name, the database table name, and the data file name. All of this information, along with the schema into which to load the tables, is specified using the addfile batch file or the tool xdbcutil.exe.

Please note that another issue that you may need to consider is the configuration file variable **FILE_PREFIX**, which points to the directory in which your files are stored.

5.5.2 Multi-company Support

AcuXDBC offers convenient ways to manage multiple company data sets. For example, you may have two sets of data for Company01 and Company02. The layouts of the data files are the same, but the file names have different prefixes or suffixes. For example, ACCT01 and ACCT02 may be the accounting files for Company01 and Company02. They are stored in different directories.

Rather than having to set up system catalogs for each of your companies individually, you can do one of two things:

1. You can use **wildcards** and substitution characters (defined in the configuration file) to identify which company file to access.
2. You can set up a separate schema for each company, and then you can specify which table to access. For example, you could set up a schema for "acme" and a separate schema for "generic," and then qualify your SQL statements.

To set up a multi-company DSN with wildcard substitution:

1. Create a text file listing the XFD filenames for your companies. Name the file "list.txt" or similar. For example, if your data is:

   ```
   main.xfd
   ACCT01.dat
   ACCT02.dat
   ACCT03.dat
   ```

   you might have an entry like this in "list.txt":

   ```
   main#AcctTbl#Acct$$
   ```

   where *main* is the XFD name, *AcctTbl* is the table/repository name, and *Acct$$* represents the Vision file name.

2. In the configuration file, "acuxdbc.cfg" by default, use the **FILENAME_WILDCARD** to define the wildcards and their respective substitution characters. For example, the configuration file might contain:

   ```
   FILENAME_WILDCARD $$=01
   ```

3. Create your system catalog using the “-f” option as shown:

   ```
   xdbcutil -d c:\data\dict -f list.txt
   ```

   When building the system catalog, **xdbcutil** uses the XFD “main” to create a repository table called AcctTbl.

4. At the company site, create a single DSN for Accounting, and point to the configuration file that contains the substitution characters for that company. (See **section 5.8** for instructions on creating DSNs.)

Then when a user accesses the table AcctTbl from your COBOL program, AcuXDBC looks in the configuration file for the FILENAME-WILDCARD variable, and, substituting “01” for “$$” accesses the data files, “Acct01” and “Acct02”. In programming terms, when “SELECT * from AcctTbl;” is issued, “Acct01” is returned.

There is no limit on the number of wildcards/replacements that you can use with AcuXDBC. You can change several characters in a filename. For example:

```
ACCT£$$**
```

where the actual filename would be ACCTZ01AA and the configuration file entry would be:

```
FILENAME_WILDCARD £=Z ;$$=01;**=AA
```

You don’t have to create a list file to specify your company wildcards. If you prefer, you can specify them directly on the **xdbcutil** command line, like this:

```
xdbcutil -d c:\data\dict -x c:\data\data  -a mainxfd#AcctTbl#Acct$$
```

Note: You may have to escape the wildcard characters if they are shell interpreted. For example, the sample above would not work on UNIX, because the shells interpret the “$” character. You would have to use:

xdbcutil -d /data/dict -x /data/data -a mainxfd#AcctTbl#Acct\$\$
or
xdbcutil -d /data/dict -x /data/data -a "mainxfd#AcctTbl#Acct$$"

To set up a multi-company DSN with table ownership:

Alternatively, to support multiple companies, you could specify database table ownership using the "-o" option to **xdbcutil** when creating your system catalog. For example, there could be two tables named AcctTbl, for instance—one owned by Company01 and one by Company 02.

Filename	Database Name	Owner	Table name
ACCT01	dd	company01	AcctTbl
ACCT02	dd	company02	AcctTbl

On the command line, you specify:

```
xdbcutil -d <catalog_dir> -x <xfd_dir> -o <owner's name>
xfdname#tablename#filename
```

Or in this case:

```
xdbcutil -d c:\data\dict -x c:\data\data -o Company01
mainxfd#AcctTbl#ACCT01

xdbcutil -d c:\data\dict -x c:\data\data -o Company02
mainxfd#AcctTbl#ACCT02
```

Then you can access all the data files with one SQL query to one DSN, like this:

```
SELECT * FROM COMPANY01.AcctTbl AND COMPANY02.AcctTbl
WHERE .....
```

You can also use synonyms for AcctTbl, such as "Accts Receivable" or "Billing".

Note: If you do not want a table owner to be listed publicly in applications like Microsoft Access, specify "-o " "" when loading the table with **xdbcutil** and set the **IGNORE_OWNER** to "1" (on, true, yes) in your configuration file.

5.6 Setting Permissions on Your Vision Tables

You set object privileges—permissions on individual tables, views, and so on—by modifying the GENESIS_AUTHS system table. GENESIS_AUTHS is created when you use the "-pa" option to **xdbcutil** or **ainit** when creating the system catalog. (**Ainit** defaults to "-pa"; you must specify "-pa" with **xdbcutil**.)

Like the database-level privileges described in **section 5.4**, this is done by issuing GRANT commands from an SQL query tool like **xdbcquery**. Use the SQL GRANT command to specify permissions for specific users.

This is often done at the user site by a site administrator.

1. From your command-line prompt, return to the /AcuGT/bin/ directory where AcuXDBC was installed.

2. Log in as database administrator. For example, type the following at the prompt and press **Enter**.

   ```
   asql -u system -p manager
   ```

   After a brief message, the following prompt appears:

   ```
   SQL (/? for help) ==>
   ```

3. Grant table-level privileges to individual users or groups of users. For example:

   ```
   GRANT SELECT ON Table_A TO sjones;
   GRANT ALL PRIVILEGES ON Table_B TO public;
   ```

   Other privileges include DELETE, INSERT, and UPDATE.

   If you want to give someone read-only access, grant them the SELECT privilege. This allows them to issue the SELECT statement on the object, but nothing else. If you want them to be able to read, write, delete, and insert data, grant them SELECT, UPDATE, DELETE, and INSERT privileges. ALL PRIVILEGES passes on all applicable privileges that you are entitled to grant. USAGE allows grantees to use the object to define another object.

Refer to Chapter 7, **section 7.5.11** for more information on granting object privileges to users.

5.7 Starting AcuXDBC Server (Network Only)

If you or your users are operating in a network environment, the final step is to start AcuXDBC Server on the server. If you are operating on a stand-alone machine, this step is not necessary. You can begin using ODBC applications to access Vision right away.

Note that to perform the functions described in this section on a UNIX server, you should be logged in as *root* or *superuser*. To perform these functions on a Windows NT or Windows 2000 server, you should be logged in from the *Administrator* account or from an account that belongs to the *Administrators* group.

To start AcuXDBC Server on the server, enter the following at the command prompt:

```
acuxdbcs -start
```

This runs a script that invokes the **xdbcsrvr** daemon on the default port, 20222, with the "-k" option to request data encryption over the network. If the **xdbcsrvr** daemon is already running, AcuXDBC Server displays the message:

```
The AcuXDBC Server 8.0.x on the default port <20222> service is already installed
```

A new **xdbcsrvr** process will not be started. If you want to start the AcuXDBC Server with new options, you must stop and restart **xdbcsrvr**.

If you would like to invoke the server daemon with different security or logging options, you can issue the **xdbcsrvr** command directly. Following is the command syntax for **xdbcsrvr**.

Command Syntax

```
xdbcsrvr [-a[l]] [-p] [-kn] [log]
```

Optional arguments to **xdbcsvr** include:

-a[l]	Activates the operating system (OS) authorization checking. This option lets one user start a process and pass ownership to another. You must pass a valid operating system user ID and password as part of the connect string, so the spawned service knows who owns it. **xdbcsrvr** performs a default OS authentication. If the *l* modifier is specified, **xdbcsrvr** logs the outcome of the authentication request to the system log. On UNIX, this is syslog. On Windows, it is the event log.
-k*n*	Specifies a masking key value used to decrypt incoming OS and DBMS user ID/password information. Clients must have the same value in their "net.ini" file in the KEY_CONNECT variable.
-p*port_number*	Assigns the listener port number for clients to request service. The default is 20222.
log	Starts logging. It puts a log file named `tcm`*`processid`*`.log` in the directory where **xdbcsrvr** is started.

To test your connection with the server, use the process query command, "**acuxdbcs** -info" to ping the server.

5.7.1 Pinging AcuXDBC Server

Use "**acuxdbcs** -info" to ping the **xdbcsrvr** process. If no *server* is specified, the **xdbcsrvr** process is pinged on the current host; otherwise the process is pinged on the named host.

5.7.2 Stopping AcuXDBC Server

Use "acuxdbcs -kill [-n port] [server]" to halt (kill) the **xdbcsrvr** process on port 20222 or the port specified by the "-n" option. On UNIX, you may also use "xdbckill [-p<port#>]". If no *port* is specified, the **xdbcsrvr** process is halted on the current host; otherwise the process is halted on the named host.

On Windows, all client connections are terminated. On UNIX, client connections remain active.

Caution: The **xdbcsrvr** process can also be terminated from UNIX using the command "kill -9" (signal #9). However, a signal #9 prevents **xdbcsrvr** from performing an orderly shutdown and should never be used when clients are actively using AcuXDBC.

5.8 Setting Up Data Source Names (DSNs) on Client

Traditionally, the data source is one of the main components of an ODBC installation. Physically, data sources consist of sets of data and their associated environments. This means that the term "data source" actually indicates the data files, the operating system, the file system or DBMS used to manage the physical data, and the optional network software used to access this data. The term *data source name* (DSN), therefore, refers to something rather complicated.

Because part of the ODBC mission is to hide the complexity of underlying software, ODBC architects have chosen to use an abstract name to identify all the components of a data source. This data source name maps all the underlying software components necessary to access the data.

The name you assign to a data source is arbitrary. When users try to access the data files, they will be asked to specify the DSN. Therefore, you should assign a meaningful name that represents the type of data that the files contain or the type of query that will be made against the files.

This section describes how to create local data source names (or DSNs) for your COBOL data files.

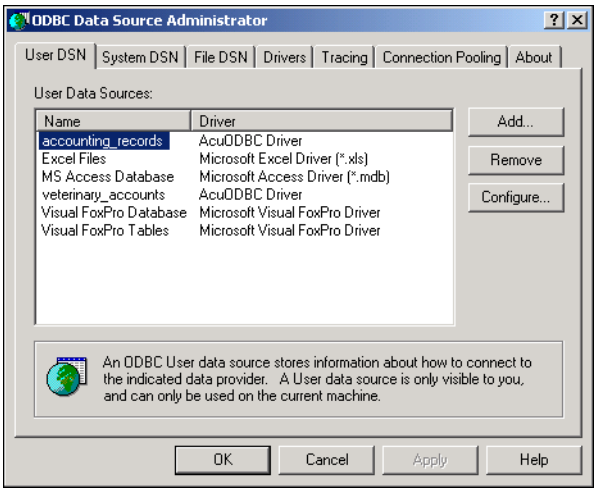
You begin by adding the AcuXDBC driver on the ODBC Data Source Administrator of the Windows control panel. (The ODBC Data Source Administrator, included as part of your operating system, is your interface for adding, removing, or configuring ODBC data sources on your system.) Once you add a driver, you define the data source name and configuration file through the AcuXDBC Setup screen.

Note: If you are upgrading from an earlier version of AcuODBC, you must create new DSNs.

5.8.1 Adding a Data Source Name

To add a DSN to the ODBC Data Source Administrator, follow these steps.

1. From the Start menu, select **All Programs/Acucobol 8.x.x/AcuXDBC/ ODBC Data Source Administrator**. The ODBC Data Source Administrator appears with the User DSN tab selected.



User DSNs are one of three types of DSNs managed by the ODBC Data Source Administrator:

- **User DSN:** Stores information about how to connect to a specific data source. May be used only by the current user on the current machine.

- **System DSN:** The most common type of DSN that you and your users will create for AcuXDBC. Same as the User DSN, but available to all users on a particular machine, including NT services.

Local to the computer, rather than dedicated to a user. The system, or any user having privileges, can use a data source set up with a system DSN.

- **File DSN:** The File DSN is a file-based data source that may be shared among all users who have the same drivers installed. AcuXDBC does not support File DSNs.

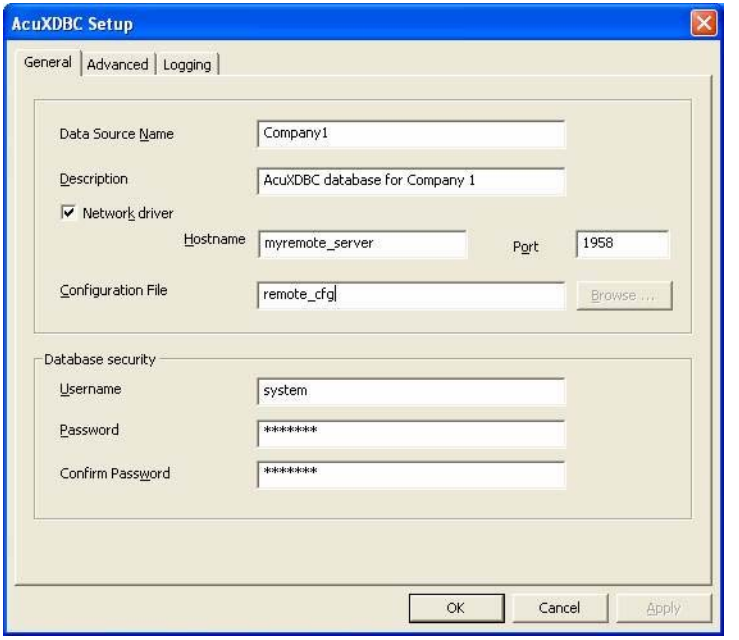
2. Select the User DSN or System DSN tab, depending on your preference, and click **Add**.

3. From the Create New Data Source dialog box, select **AcuXDBC Driver**, and then click **Finish**. (If "AcuXDBC Driver" is not an option in the list box, reinstall AcuXDBC on your system.)

   The AcuXDBC Setup screen appears. It contains three tabs: General, Advanced, and Logging. Complete these tabs as described in **section 5.8.2** to 5.8.4.

5.8.2 AcuXDBC Setup: General Tab

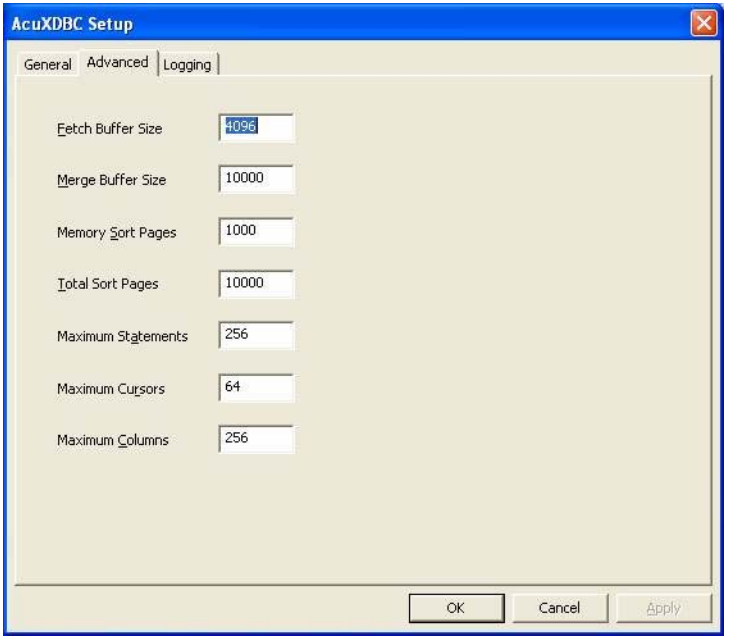
Into the General tab, enter the information that is requested, then click **OK**. Descriptions of each field are shown in the table below.



Field	Description
Data Source Name (required field)	Enter a string that identifies the name of the data source that you are creating. Enter a name that represents the type of data in this group of data files or the type of query that will be made against these files. Since this name is used every time a user opens this data source, select a name that is meaningful to users. (For example, "ACME Payroll" might be used for payroll files for the ACME Corporation.) The name of the data source must be no longer than 63 characters in length. Follow the Windows file-naming conventions when using special characters in data source names. Also, note that the name you choose must be unique. This is a required field. Default value is blank.
Description (Optional)	Enter a description of the data source for additional user clarification. Default value is blank.
Network driver	Check this box if this is a network data source.
Hostname	If this is a network data source, enter the hostname of the server on which the data resides.
Port	If this is a network data source, enter the port number on which the server is listening.
Configuration File (required field)	Browse to the configuration file that you created for the data source (a remote file if this is a network data source). The configuration file defines the location of your Vision files, file case instructions, and much more. This is a required field. By default, it is located in the installation directory and named "acuxdbc.cfg". Refer to Chapter 4, **section 4.2** for more information on this file.
Username (Optional)	A username for the person who will be logging into the database on this machine. Default value is blank.
Password (Optional)	A password to be associated with that username. Default value is blank. Refer to **section 5.4** for information on alternative measures of data security.
Confirm Password	Confirm the password by entering it a second time.

5.8.3 AcuXDBC Setup: Advanced Tab

Into the Advanced tab, enter the following information or accept the defaults, then click **OK**.

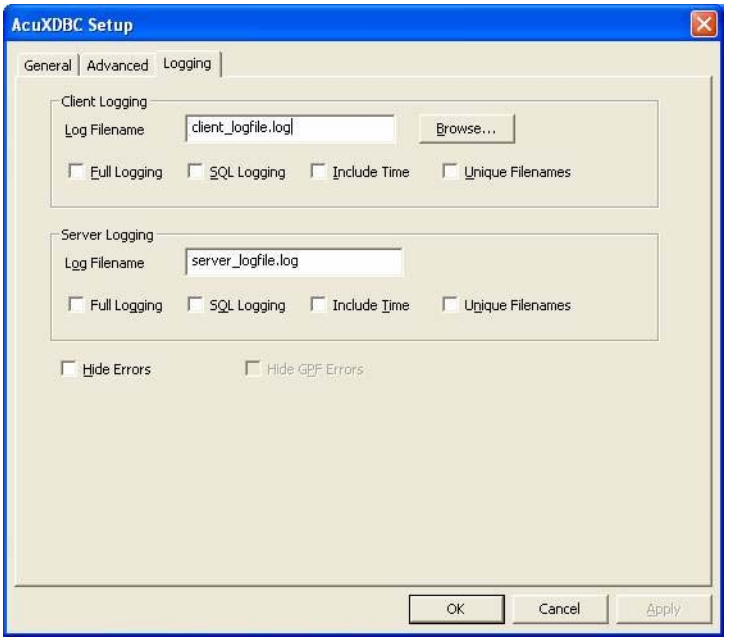


Field	Description
Fetch Buffer Size	Enter the size of the fetch buffer in bytes. The fetch buffer indicates how much data will be *fetched* or retrieved from a query at a time. Allowable values are 1024 through 32766. Default value is 4096.
Merge Buffer Size	Enter the size of the merge buffer in rows. This is the number of unique rows that your query can return. Default value is 10,000.
Memory Sort Pages	Enter the number of memory sort pages. This indicates how many of the "Total Sort Pages" will reside in memory. Allowed values are 100 - 999,999. Default value is 1000.

Field	Description
Total Sort Pages	Enter the total number of sort pages. Total sort pages is the number of virtual memory pages used to sort results sets. Allowed values are 1000 to 99,999,999. Default value is 10,000.
Maximum Statements	Enter the maximum number of statements allowed. values are 10 to 4096. Default value is 256. See Maximum Cursors.
Maximum Cursors	Enter the maximum number of cursors allowed. Allowable values are 10 to 1024. Default value is 64. AcuXDBC has a concept of logical cursors (nlc) that are mapped over a smaller number of actual DB Cursors (ndc). If you create SQL statements equal to or fewer than the number of DB Cursors set, it's a straight one-to-one relationship. However, when you attempt to open more, AcuXDBC searches for the oldest "soft-closed" logical cursor that it can "unhook" from a database cursor and assign to the new statement. If the application tries to use the original statement, then AcuXDBC will look for another unused logical cursor to donate its DB Cursor. If no cursors can be unhooked (because results are pending), then an error message "Too many concurrently open cursors" is returned.
Maximum Columns	Enter the maximum number of columns allowed in a single table. Allowable is 1 to 1024. Default value is 256.

5.8.4 AcuXDBC Setup: Logging Tab

You can gather information from AcuXDBC to help you troubleshoot server processes and debug your client applications. Into the Logging tab, enter the information that is requested or accept the defaults, then click **OK**.



Field	Description
Client Logging:	
Log Filename	Browse to the name of the local log file used for AcuXDBC logging. By default, this is "ACUXDBC_API_LOGFILE.log". Log files are created in the local directory that is current when AcuXDBC starts. If you want your log file to write to another location, you must specify a fully-qualified file name.

Field	Description
Full Logging	Check this box if you want detailed logging on the client. By default, AcuXDBC performs abridged logging. Abridged logging, more space-efficient, records selected items rather than the complete control structure.
SQL Logging	Check this box if you want to create an SQL log file locally.
Include Time	Check this box if you want to include the time to complete each call in the local log file.
Unique Filenames	Check this box if you want unique identifiers to be appended to the name of the local log file.
Server Logging:	These options appear when you check "Network driver" on the General Tab.
Log Filename	Browse to the name of the remote log file used for AcuXDBC logging. By default, this is "ACUXDBC_HOST_LOGFILE_pid.log", where *pid* is the host process ID. Log files are created in the local directory that is current when AcuXDBC starts. If you want your log file to write to another location, you must specify a fully-qualified file name.
Full Logging	Check this box if you want detailed logging on the host. By default, AcuXDBC performs abridged logging. Abridged logging, more space-efficient, records selected items rather than the complete control structure.
SQL Logging	Check this box if you want to create an SQL log file on the host.
Include Time	Check this box if you want to include the time to complete each call in the log file on the host.
Unique Filenames	Check this box if you want unique identifiers to be appended to the name of the remote log file.
Hide Errors	For Windows servers only, check this box if you want AcuXDBC to quit when a GPF error occurs.
Hide GPF Errors	For Windows servers only, check this box if you do not want to display GPF dialog boxes to the end user.

5.8.5 Copying DSNs to Other Network Machines

If desired, you can create a single DSN, then download it to other machines in your network using the ".reg" file.

6 The System Catalog

Key Topics

6.1 Introduction

The system catalog forms the core of your database. It comprises a set of system tables that describe all of the tables and relations described by the database. You can have multiple system catalogs (databases) but only a single catalog may be accessed at one time. The system catalog is portable between machines with the same byte-order.

AcuXDBC creates a system catalog corresponding to information it reads from your XFDs. Once this information is loaded, AcuXDBC no longer requires your XFDs. Instead, it refers to the system catalog to obtain the information needed to construct and display your Vision tables. This gives you the option of providing end users with XFDs or encrypted XFDs (.efd) files for building their own system catalog, or you can provide end users a pre-built system catalog. The ability to supply a pre-built system catalog can lead to other benefits for your applications. For example, you can also pre-build user logins, database object permissions, and complex data handling views.

The primary advantage to a system catalog is that it allows for much greater relational database-like behavior.

This chapter describes the structure of the system catalog, and in doing so, orients you to the environment from which you can access and manage your database. It shows how a newly created Vision database may appear to users after the initial install of AcuXDBC and a system catalog populated with sample customer data. Database administrators and other programmers should find the information in this chapter useful for more effectively managing and querying the database.

Note: This chapter is not intended to teach SQL programming, or how to manage a database. If you do not have a fundamental understanding of these concepts, you should first consult a resource dedicated to SQL programming and database management.

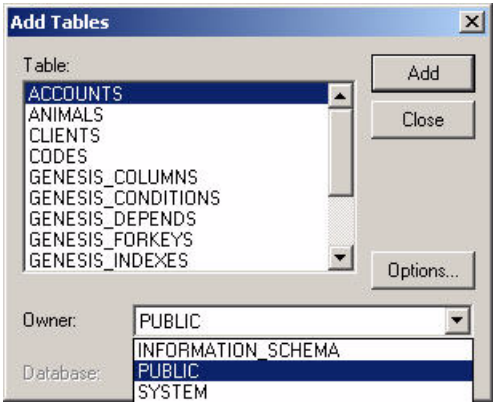
This chapter also provides a description of the **xdbcquery** command-line tool that can be used to access the system catalog and interact with your database tables directly. **xdbcquery** is provided on your distribution media.

6.2 System Catalog Structure

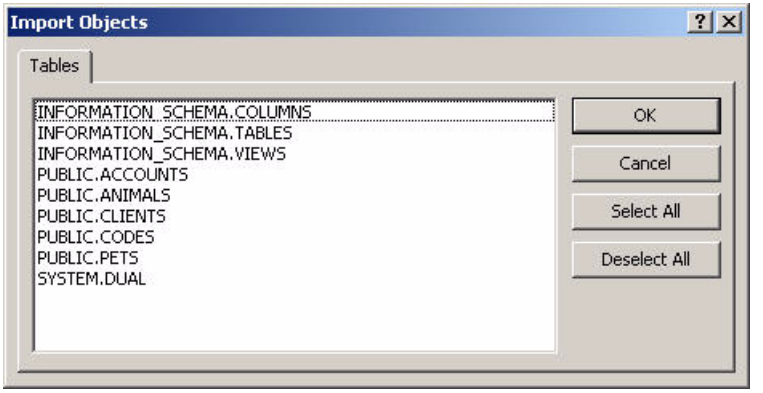
Note: The examples in this section are based on the sample catalog that is created from the "demo.bat" file that is included with AcuXDBC. The examples depict what a user encounters after the system catalog is created and loaded using this installation script and a DSN is created, as described in Chapter 5, **section 5.8** of this manual.

The system catalog displays the database tables that are created from your XFD files. It also contains system tables that you can query to obtain information about the definitions that make up your tables. The initial structuring of the catalog places or assigns your tables to the "PUBLIC" domain, which allows any database user to access the tables. AcuXDBC also creates two other entities that appear as users or owners, depending on the client application being used to view the catalog. These entities are called "INFORMATION_SCHEMA"and "SYSTEM."

For example, the following Microsoft Query Add Tables screen shows a partial list of the catalog tables that appear under PUBLIC for the sample veterinarian database:



From Microsoft Access, the same veterinarian database tables appear as shown below:



6.2.1 PUBLIC

Tables created from your XFDs have an owner. By default, this owner is the authorization ID “PUBLIC”. Tables or other database objects owned by PUBLIC are accessible to any authorized database user. This includes users initially created, as well as any additional users added in the future.

If you do not want a table owner to be listed publicly in applications like Microsoft Access, specify “-o “ “ ” when loading the table with **xdbcutil** and set the **IGNORE_OWNER** to “1” (on, true, yes) in your configuration file.

6.2.2 GENESIS Tables

Although appearing under the PUBLIC owner, GENESIS tables are a special class of tables. These are internal system tables required by the AcuXDBC processor. When you execute SQL statements, the AcuXDBC processor constantly refers to these tables in order to complete the SQL command. You cannot change or modify GENESIS tables directly. The AcuXDBC processor modifies these tables when you issue various SQL commands. For

example, issuing the following command not only results in AcuXDBC creating a new table, it also results in a new entry into the GENESIS_TABLES called “MYTABLE:”

```
CREATE TABLE MYTABLE(COL1 CHAR (10));
```

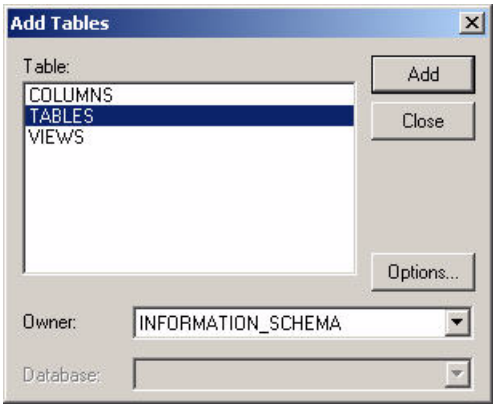
GENESIS tables include the following:

Table Name	Description
GENESIS_AUTHS	Enables the setting of object (e.g. tables, views) authorizations such as SELECT, INSERT, UPDATE, AND DELETE.
GENESIS_COLUMNS	Provides the mapping between SQL tables and views and the Vision record fields.
GENESIS_CONDITIONS	Tracks the conditions used to modify the result set based on the values in certain columns.
GENESIS_DEPENDS	Keeps track of view dependencies.
GENESIS_FORKEYS	Defines foreign key relationships, but is not currently used by Vision.
GENESIS_INDEXES	Provides the mapping between SQL indexes and the Vision keys.
GENESIS_TABLES	Provides the mapping between the SQL tables and views and the Vision file.
GENESIS_USERS	Enables a security method based on user ID and password.
GENESIS_VIEWS	Describes the SQL view definition.
GENESIS_XCOLUMNS	Provides the mapping between SQL indexes and the Vision key segments.

6.2.3 INFORMATION_SCHEMA

The INFORMATION_SCHEMA gives you access to the definitions that make up your database tables. It contains three tables named COLUMNS, TABLES, and VIEWS. You can query these tables to find information about the definitions that make up your database tables, columns, and views.

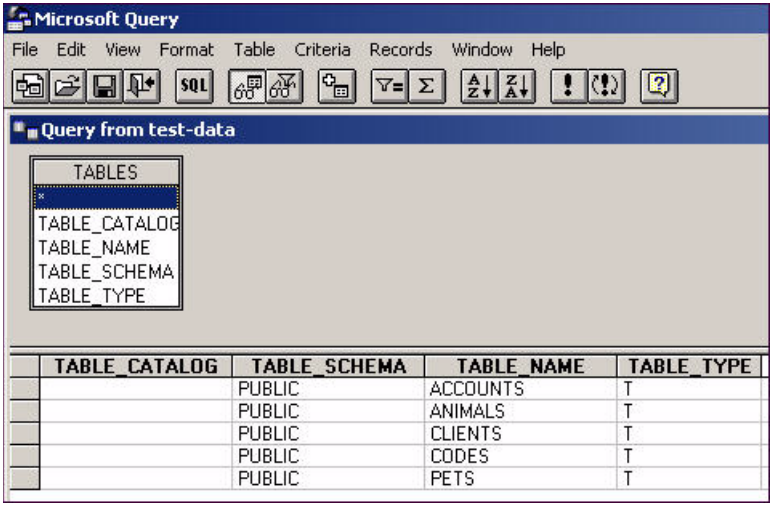
Using Microsoft Query as an example, choosing INFORMATION_SCHEMA owner from the Add Tables dialog box displays the following:



Then double-clicking the "*" or executing the following SQL command:

SELECT * FROM INFORMATION_SCHEMA.TABLES;

Displays the following information for each of the Veterinarian database tables:



Note: You can query and read the column headings, but not modify them.

6.2.3.1 INFORMATION_SCHEMA.COLUMNS

This table contains the following information on each user-defined or publicly created column.

Column Heading	Definition
table_catalog	Database containing the column.
table_schema	Column owner.
table_name	Name of the table containing the column.
column_name	Name of the column.
ordinal_position	The column's physical location in the table.
is_nullable	Indicates if NULL is allowed (Y/N).
data_type	Column data type.
character_octet_length	Applies to char and varchar, the number of bytes in the column.
numeric_precision	The maximum precision column for decimal and numeric data types.
numeric_scale	The maximum scale column for decimal and numeric data types.

6.2.3.2 INFORMATION_SCHEMA.TABLES

This table contains the following information on each user-defined or publicly created table.

Column Heading	Definition
table_catalog	Database containing the table.
table_schema	Table owner.
table_name	Name of the table.

Column Heading	Definition
table_type	Type of table. The possible values are: S - System T - User V - View

6.2.3.3 INFORMATION_SCHEMA.VIEWS

This table contains the following information on each user-defined or publicly created view.

Column Heading	Definition
table_catalog	Database containing the View.
table_schema	View owner.
table_name	Name of the view.
view-definition	SQL that defines the view.

6.2.4 DUAL

DUAL is a system table and has one column called “DUMMY” and one row containing the value “X”. The DUAL table is a convenience tool that enables users, or their programs, to select a single known constant or function.

For example, since DUAL contains only a single column, it makes the following types of commands more sensible:

```
SELECT USER()FROM DUAL;

SELECT NOW()FROM DUAL;

SELECT DATABASE()FROM DUAL;
```

...and for other functions where you only want a single value returned.

6.3 Using the Command-line Query Tool

AcuXDBC includes a command-line SQL query tool called **xdbcquery**. You use **xdbcquery** from either a Windows or UNIX command prompt to connect to your system catalog and execute SQL commands on your database directly, as opposed to connecting to the catalog from a third-party query tool such as Microsoft Query. The query tool is also useful for conducting the following database tasks:

- Assigning users and granting permissions.
- Creating VIEWs.
- Executing AcuXDBC queries and programs in non-interactive mode through an SQL script.
- Testing SQL scripts.

Using the query tool requires you to first start the tool and then specify a command that connects you to your system catalog. You can perform these steps manually from the command prompt or you can run a batch file that includes the **xdbcquery** start and connect command syntax and parameters.

Tip: **xdbcquery** is also accessible from the Windows Start menu through the following menus: All Programs/Acucorp8.1/AcuXDBC/xdbcquery.

6.3.1 Starting xdbcquery from the Command Line

To start the query tool and connect to your system catalog, execute the following **xdbcquery** connect command. (No spaces are allowed between parameters.)

```
xdbcquery /cacuxdbc04:[user name/password/]xvision:
[configuration file name]
```

Where:

Parameter	Definition
/c	xdbcquery connect command.
acuxdbc04	ODBC Driver with which to connect.
user name*	User name.
password*	User password.
xvision	Vision driver with which to connect
configuration file name	File that specifies the data source and catalog location.

* Not required if no database security is in effect.

Example

```
xdbcquery /cacuxdbc04:system/manager/xvision:acuxdbc.cfg
```

Note: If you have not set up database security, you can omit the user name and password, but need to keep the two forward slashes before "xvision" as place holders.

You are now connected to the system catalog, and can execute **xdbcquery**-specific commands, or execute SQL statements on your catalog tables. Chapter 7, **section 7.4** provides the complete list of SQL commands and functions supported by AcuXDBC.

6.3.2 Starting xdbcquery from asql.bat/asql.sh

For your convenience, a file called "*asql.bat*" ("*asql.sh*" on UNIX/Linux) is included with AcuXDBC and is used to start **xdbcquery** and connect to a database and system catalog. asql.bat has the following usage:

Command	Parameter	Default	Description
-c	configuration filename	acuxdbc.cfg	Specifies the configuration file that identifies the database and catalog location.
-u	user name	system	Specifies the user name for whom access is granted.
-p	password	manager	The password for the user.
-r	script name	none	**Runs the script** specified.

Example

```
asql -u mary -p 123
```

Since no configuration file was specified, this command connects the user Mary to the database and catalog specified in acuxdbc.cfg configuration file.

Note: You can modify the default parameters directly within asql.bat to match your settings and then simply run the "asql.bat" file from the same directory where **xdbcquery** is installed.

6.3.3 xdbcquery Commands

Once started, you can view a list of command options and other helpful information by executing the "/?" command. Doing so, displays the following information:

Command	Parameter	Description
/c	connect_string	Connect to a database.
/x	c r [u]	Commit or Rollback transaction and, if "u" specified, start a read/write transaction.
/d	s t<string>	Describe a SQL statement or table.
/r	file name	Run statements/commands from a file. Files can be chained, but not nested.

Command	Parameter	Description
/o	cmd parameters	Execute driver command.
/m	mask	Set date time format mask.
/n	null	What to display for a NULL. Default is blank.
/l	N	N lines displayed for SELECT. 0 for continuous.
/p	[N T [data]]	Parameters. Just "/p" to reset. "N" is parameter number (zero based). "T" is data type (B - binary, C - character, D - date time, I - integer, N - number).
/v	y n	Verbose messages. "N" for errors only.
/b		Display database request block.
/s	N	Sleep for N seconds.
/? or /h		Help screen.
/q		Quit session and exit.

6.3.3.1 Running SQL Scripts

Using a text editor, you can create SQL scripts and run them with **xdbcquery** by using the "/r" command followed by the filename. A sample script file appears below. Note that **xdbcquery** recognizes the pound symbol (#) as the comment identifier.

```
################################################################
#  x.sql
################################################################

create table foo(col1 char(10), col2 char (10) );

#now create an index for the file
create unique index ifoo_0 on foo(col1);

#insert values
insert into foo values ('1','1');

#insert a null value for col2
insert into foo values ('2', '');
```

```
# set the null character
/n !null!

#select some records
select * from foo;

#now exit
/q
```

Run this script by issuing the following command:

```
asql -r x.sql
```

7 SQL Supported Commands

Key Topics

7.1 Introduction

This chapter describes the SQL commands and functions that can be used to query and manage your database tables. **Section 7.4** provides a concise list of the supported SQL commands, while **section 7.5** provides you with a detailed explanation of each supported command including its syntax, usage, and sample coding.

Note: This chapter is not designed to teach SQL programming or to serve as a comprehensive SQL guide. Please consult your preferred SQL-specific programming guide for detailed information on SQL programming techniques.

7.2 Conventions

The following conventions describe the SQL syntax supported by AcuXDBC. There are no rules pertaining to the number of words you put on a line or where the line breaks. The number of words and line breaks used in this section are strictly for readability

Convention	Description
CAPS	Capital letters indicate the word is a keyword (command).
MIXed	Capitals mixed with lower-case letters indicate that the word is a keyword, but you can type either the full word or only the letters in capitals.
lower	Lowercase words are variables; you supply your own.
{ }	Curly braces indicate you must choose at least one of the enclosed options.
[ ]	Brackets indicate you can choose one or more of the enclosed options, or none of them.
( )	Parentheses are part of the command. Type them just as they appear.
|	A vertical bar separates mutually exclusive options. You can only choose one.

Convention	Description
,	A comma separates multiple options. You can choose as many options as you like, but include commas in the command between choices.
…	The ellipses (three dots) mean you can repeat the marked section or commands as many times as you like.

7.3 Limitations and Restrictions

The following section describes SQL programming protocols related to object names, predicates and constraints.

7.3.1 Object Names

Names or identifiers of objects such as columns, tables, and indexes are limited to 30 characters. Identifiers must start with an alphabetic character and can include numbers as well as the "_" character. Avoid using any special characters such as "-" and "+". Using special characters will require that you put double quotation marks ("" "") around identifiers in your SQL statements.

7.3.2 Predicates

Predicates are expressions that apply comparison operators (comp_op elements in the following syntax) and/or SQL predicate operators (IN, EXISTS, and so on) to values to produce a truth value of TRUE, FALSE, or UNKNOWN.

Predicates can be either a single expression or a combination of any number of expressions using Boolean operators (AND, OR, and NOT) as well as the special SQL operator IS, and parentheses to define the order of evaluation.

Predicates are most often used in the WHERE and HAVING clauses of SELECT statements and subqueries to determine the rows or aggregate groups to select, and in UPDATE and DELETE statements to identify the rows on which changes should be made.

Predicates evaluate to TRUE, FALSE, or UNKNOWN. UNKNOWNs arise when NULLs are compared to any value, including other NULLs, since it is impossible to know the value of a data field with NULL value. You can use Boolean operators and SQL IS on UNKNOWN truth values.

7.3.3 Constraints

The ANSI standard defines a list of constraints that can be placed on a table or index. Constraints following the definition of a column apply to that column; those standing alone as table constraints can reference any one or more columns in the table.

At this time, AcuXDBC supports only the NOT NULL constraint. The complete list of possible constraints are:

Constraint	Description
NOT NULL	Forbids NULLs from being entered in a column. According to ANSI standard, this specification can only be a column constraint.
UNIQUE	Mandates that every column value, or combination of column values if a table constraint, be unique.
PRIMARY KEY	Same effect as UNIQUE, except that none of the columns in a PRIMARY KEY constraint can contain NULLs. You can only issue this constraint once in a given table.
CHECK	Followed by a predicate (in parentheses) that uses column values in some expression whose value can be TRUE, FALSE, or in the presence of NULLs, UNKNOWN. The constraint is only violated when the predicate is FALSE.

You can define constraints so that they are not checked until the end of the current transaction. This approach is useful when you want to update a table that references itself as a parent key. This operation usually creates intermediate states where referential integrity must be violated. By default, constraints are not deferrable.

7.4 Summary of Supported SQL Commands

This section provides a concise list of each supported command's syntax. **Section 7.5** provides detailed descriptions of each command including syntax, descriptions, use, and examples.

If you are writing commands from an ODBC or JDBC application, that application must also support the same commands as AcuXDBC in order for the command to function properly.

```
CREATE (UNIQUE) INDEX index_name
   ON table_name
    (column_name (ASC | DESC) (,column_name (ASC | DESC)) ... )

CREATE SYNONYM synonym_name
  FOR object_name

CREATE TABLE table_name
   (column_name datatype (NOT NULL)
      (, column_name datatype (NOT NULL)) ...)

CREATE VIEW view_name
      ((column_name(, column_name) ...))
   AS subselect
   (WITH CHECK OPTION)

DELETE FROM {table_name | view_name} (correlation_name)
   (WHERE search_condition)

DROP INDEX index_name

DROP SYNONYM synonym_name

DROP TABLE table_name

DROP VIEW view_name

GRANT privilege., ..TO { grantee }
    IDENTIFIED by { password }

INSERT INTO table_name
   [ (column_list)]
   { VALUES (constant_list)
```

```
REVOKE { privilege., ..} FROM  { grantee }

    privilege ::=
       { CONNECT
       | DBA
       | RESOURCE }

SELECT [ALL | DISTINCT ] select_list
   FROM {table_name | view_name} (corr_name)
      (, {table_name | view_name} (corr_name)) ...
   (WHERE search_condition)
   (GROUP BY column_name (, column_name) ...)
   (HAVING search_condition)
   (ORDER BY {column_name | select_list_number } (ASC | DESC))
... )

SET PASSWORD {old_password} {new_password}

UPDATE {table_name | view_name} (correlation_name)
   SET
   column_name = {expression | NULL}
      (column _name = {expression | NULL} ) ...
   (WHERE search_condition)
```

7.5 Detailed SQL Support Descriptions

This section provides detailed information on each AcuXDBC supported command including the syntax, keywords and descriptions, use, and sample code.

If you are writing commands from an ODBC or JDBC application, that application must also support the same commands as AcuXDBC in order for the command to function properly.

Commands are ordered alphabetically.

7.5.1 CREATE INDEX

Creates an index on a base table.

Syntax

```
CREATE [UNIQUE] INDEX index_name
  ON table_name
   (column_name [ASC | DESC] [,column_name [ASC | DESC]]...)
```

Keyword	Description
UNIQUE	Creates a unique index
index_name	The name of the index to create.
table_name	The name of the table for the index
column_name	Column name to use in the index key. The index key is built using the columns in the order you specify in this list.

Use

You can create indexes on initial tables that have yet to be used or referenced within a SELECT or INSERT statement. Once you use a table, you cannot go back to the table and create an index.

Example

```
CREATE UNIQUE INDEX STAFF_IX1 ON STAFF(ID)
CREATE INDEX STAFF_IX2 ON STAFF(DEPT,NAME)
```

7.5.2 CREATE SYNONYM

Creates a PUBLIC synonym name for a database object.

Syntax

```
CREATE SYNONYM synonym_name
  FOR object_name
```

Keyword	Description
SYNONYM	Creates a synonym name for a table name
synonym_name	The name of the synonym to create.
object_name	The name of the table or view to which the synonym name applies

Use

You can create synonyms for an object in the database. Synonyms can currently be created for base tables or views. The synonym name can then be used in place of the object name in queries. Note that users of the synonym name must have appropriate privileges to the base object to which the synonym applies. Synonyms are useful in that a user can access data from tables existing in other user schemas (Authorization IDs) without having to specify that user's schema.

The **DROP SYNONYM** command is used to remove a synonym name.

Example

```
CREATE SYNONYM REPS FOR Mary.ALL_REPS_LOCATIONS
```

7.5.3 CREATE TABLE

Creates a permanent table.

Syntax

```
CREATE TABLE table_name
   (column_name datatype (NOT NULL)
      (, column_name datatype (NOT NULL)) ...)
```

Keyword	Description
table_name	The name of the table to create.
column_name	Column name to create in table. Columns are created in the order you specify in this list.
datatype	Data type for the specified column. You must specify a valid data type for each named column to successfully create the table. Data types allowed: • CHAR[*n*] — fixed length character string (default: *n = 1)* • VARCHAR[*n]* — variable-length character string • DATETIME — date and time (to the second) • DECIMAL(*p*[,*s*]) — decimal of precision *p*, scale *s* • FLOAT — DECIMAL(8,6) • REAL — DECIMAL(8,6) • DOUBLE — DECIMAL(16,6) • INTEGER — four-byte integer • SMALLINT — two-byte integer

Use

AcuXDBC supports the creation of permanent base tables only. Tables contain one or more columns, which must be defined when you create the table. Thus, a table definition is a series of one or more comma-separated column definitions

To define a column you specify a *column_name*, a data type for the data in that column, and whether the data can be NULL or not.

The order of the columns in this statement determines their order in the table. A column definition must include:

- The name of the column (The column must be named.)
- A data type that applies to all column values.

NULL or NOT NULL designation.

Some data types accept size arguments indicating, for example, the length of a fixed-length character string, or the scale and precision of a decimal number. The meaning and format of these vary with the data type, but defaults exist.

Default values and collations

If you specify a NOT NULL constraint for a column, either you must define a default value or every INSERT or UPDATE command on the column must leave it with a specified value. The default can be specified as a literal, a user value function, a datetime value function, or NULL (if you don't specify a default, the column is assumed to be nullable).

Ownership and access control

Tables and other database objects are created and owned by authorization IDs, which means users in most contexts. An object's owner controls the privileges others have on it. In a sense, all privilege flows from the right to create objects. Tables are grouped into schemas and can only be created by the owner of the schema in which they reside or a user with DBA privileges.

Example

```
CREATE TABLE STAFF(ID INTEGER NOT NULL,
            NAME VARCHAR(10) NOT NULL, DEPT INTEGER NOT NULL,
             JOB VARCHAR(6) NOT NULL, YEARS INTEGER,
             SALARY FLOAT(8,2) NOT NULL,COMM FLOAT(8,2))
```

7.5.4 CREATE VIEW

The CREATE VIEW command defines a view.

Syntax

```
CREATE VIEW view_name (column_list)
     AS ( SELECT statement );
```

Keyword	Description
view_name	Name of view.
column_name	List of columns to display in view.
statement	Criteria by which you identify rows that you want to retrieve.

Use

This statement creates a view. A view is an object that is treated as a table, but whose definition contains a query — a valid SELECT statement. Because the query may access more than one base table, a view may combine data from several tables. Views do not contain their own data. Because the rows of a view are, by definition, unordered, you cannot use ORDER BY when creating a view.

You reference a view in SQL statements just like base tables. When you reference the view in a statement, the output of the query becomes the content of the view for the duration of that statement. In cases where views can be updated, the changes are transferred to the underlying data in the base table(s).

The tables or views directly referenced in a query are called the *simply underlying* tables of the query or view. These combined with all the tables they reference, and all the subsequently referenced tables all the way down to and including the base tables that contain the data, are called the *generally*

underlying tables. The base tables — the ones that do not reference any other tables, but actually contain the data — are called the *leaf underlying* tables. View definitions cannot be circular. That is, no view can be among its own generally underlying tables.

Views also cannot contain target specifications or a dynamic parameter specifications. The list of columns is used to provide the columns with names that are used only in given view. You can use it if you do not want to retain the names that the columns have in the underlying base table(s). You must use it whenever:

- Any of the two columns would otherwise have identical names.
- Any of the columns contain computed values or any values other than column values directly extracted from the underlying tables, unless an AS clause is used in the query to name them.
- There are any joined columns with distinct names in their respective tables, unless an AS clause is used in the query to name them.

If you do name the columns, you cannot use the same column name twice in the same view. If you name the columns, you must name all of them, so the number of columns in the name list is the same as the SELECT clause of the contained query. You can use SELECT * in the query to select all columns; this command is converted internally to a list of all columns so that if a column is added to an underlying table (using ALTER TABLE), your view remains valid.

Views can base their queries on other views, as long as the definition is not circular. Views cannot reference declared temporary tables, although global and created local ones are acceptable.

Inserting, updating, and deleting values in views

When you perform any of the above operations on a view the changes are transferred to the base table that contains the data. Such operations are only permitted if the changes that must be made to the underlying table are unambiguous. The principle is that an insertion or change to one row in the view must translate to an insertion or change to one row in the leaf underlying table. If this is the case, the view is said to be updatable. The specific conditions outlined in the standard for a view to be updatable are:

- It must be drawn on one and only one simply underlying table. Joins are not allowed.
- It must contain one and only one query.
- If the simply underlying table is itself a view, that view must also be updatable.
- The SELECT clause of the contained query may only specify column references, not value expressions or aggregate functions, and no column can be referenced more than once.
- The contained query can not specify GROUP BY or HAVING.
- The contained query cannot specify DISTINCT.
- Subqueries are permissible, but only if they do not refer to any of the generally underlying tables on which the view is based.

Example

```
CREATE VIEW STAFF_VIEW (Employee_id, Employee_name,
Employee_dept)
            AS SELECT ID,NAME,DEPT FROM STAFF
```

7.5.5 DELETE

This command deletes rows from a table.

Syntax

```
DELETE FROM table_name (correlation_name)
       [ (WHERE search_condition) |
       { (WHERE CURRENT OF cursor_name) } ]
```

Keyword	Description
table_name	Name of table or view from which to delete data rows.
correlation_name	Also called range variable or alias, provides alternative name for the table whose name it follows; the definition lasts only for the duration of the statement. *Correlation names* are optional for base tables and views, but required for tables produced by subqueries.
search_condition	Criteria by which you identify rows on which you want to act.

Use

This statement can be coded directly, or in dynamic SQL, coded as a prepared statement, which is a statement whose text is generated at runtime. The DELETE statement removes rows from permanent base tables, views, or cursors. In the last two cases, the deletions are transferred to the base table from which the view or cursor extracts its data.

The WHERE CURRENT OF form is used for deletions from cursors. The row currently in the cursor is removed. This is called a *positioned deletion*. The WHERE predicate form is used for deletions from base tables or views. All rows that satisfy the predicate are removed at once. This is called a *searched deletion*. If the WHERE clause is absent, it is also a searched deletion, but all rows of the table or view are removed. The following restrictions apply to both types:

- You must have DELETE privilege on the table to delete it.
- If the deletion is performed on a view or cursor, that view or cursor must be updatable.
- The current transaction mode cannot be read-only.

Searched deletions

The predicates used in DELETE statements, like those in SELECT and UPDATE, use one or more expressions. For example, location =’Bahrain’. This will test whether TRUE, FALSE, or if NULLs exist, UNKNOWN for each row based on the values within that row. Each row for which the predicate is TRUE is deleted.

Positioned deletions

Positioned deletions use cursors and therefore only apply to static or dynamic SQL, but not to interactive SQL. You can use a positioned deletion if:

- A cursor is within the current module or one of its compilation unit emulations that references the table.
- This cursor has been opened within the current transaction.
- This cursor has had at least one row fetched.
- The cursor has not yet been closed.

The last row fetched is deleted.

Prepared DELETE statements

The PREPARE statement lets you generate the text of dynamic SQL statements at runtime. When you use PREPARE to generate a positioned deletion, you can omit the FROM *table_name* clause of the DELETE statement. The table underlying the cursor is assumed.

Example

```
Downsize department 15:
DELETE FROM STAFF WHERE DEPT = 15
```

7.5.6 DROP INDEX

This command removes an index from a base table.

Syntax

```
DROP INDEX index_name;
```

Keyword	Description
index_name	Name of the index to remove.

Use

This statement is used to remove an index and can be used currently only before the table is accessed the first time by a non-DDL statement (which causes the physical file to be created).

Example

```
DROP INDEX STAFF_IX1
```

7.5.7 DROP SYNONYM

Removes a PUBLIC synonym from the database.

Syntax

```
DROP SYNONYM synonym_name
```

Keyword	Description
synonym_name	The name of the synonym to drop.

Use

This statement is used to remove a synonym referencing a table or view from the database. The removal of a synonym has no effect on the base object to which it refers.

The **CREATE SYNONYM** statement is used to create a synonym name.

Example

```
DROP SYNONYM REPS
```

7.5.8 DROP TABLE

This command removes a base table.

Syntax

```
DROP TABLE table_name;
```

Keyword	Description
table_name	Name of the table to remove.

Use

This statement is used to remove the same kinds of tables that are created with a **CREATE TABLE** statement. To drop views use the DROP VIEW statement. To drop a table you must own the schema in which the table resides or have DBA privileges.

The definition of the table is eliminated and all users lose their privileges on that table.

Example

```
DROP TABLE STAFF
```

7.5.9 DROP VIEW

This command removes a view.

Syntax

```
DROP VIEW  view_name ;
```

Keyword	Description
View_name	Name of the view to remove.

Use

This statement drops a view, which must previously have been created with a CREATE VIEW statement. To drop a view you must own the schema within which the view resides or have DBA privileges.

Example

```
DROP VIEW STAFF_VIEW
```

7.5.10 GRANT (Database privileges)

This command gives users database access rights.

Syntax

```
GRANT privilege., ..TO { grantee } IDENTIFIED by { password }

privilege ::=
     { CONNECT
     | DBA
     | RESOURCE }
```

Keyword	Description
privilege	Type of privilege to grant: • CONNECT: lets grantees connect to and read from the database. • RESOURCE: lets grantees create objects in the database. • DBA: implies both CONNECT and RESOURCE, and gives grantees total control over all tables in the database.

grantee	User name to allow privilege.
password	Password for user name.

Use

Only DBAs can use the GRANT command. If you are a DBA, it lets you give grantees (the authorization ID that represents a user or group of users) the right to perform specified actions on the database.

You may use **SET PASSWORD** to change a password.

Example

```
GRANT CONNECT,RESOURCE TO CLERK IDENTIFIED BY MY42
```

7.5.11 GRANT (Object privileges)

This command gives users database object privileges.

Syntax

```
GRANT privilege., ..ON object_name
    TO { grantee., ...} | PUBLIC
    [ WITH GRANT OPTION ];

privilege ::=
     { ALL PRIVILEGES }
     | { SELECT
         | DELETE
         | { INSERT [ ( column name.,..) ] }
         | { UPDATE [ ( column name.,..) ] }

object name ::=
         [ TABLE ] table name
```

Keyword	**Description**
privilege	Type of access, action, or privilege to grant.
object_name	Name of the object on which to grant privileges.
grantee	User name(s) to allow privilege.

Use

This statement gives grantees (the authorization IDs that represent a user or group of users) the right to perform specified actions on named objects.

USAGE

To grant an object privilege, you must have been given (granted) the privilege yourself. You can then grant that particular privilege to other users. Note that the users to whom you grant this privilege are able to grant this same privilege to other users.

ALL PRIVILEGES

ALL PRIVILEGES passes on all applicable privileges that you are entitled to grant. PUBLIC denotes all authorization IDs, present and future.

Other Privileges

SELECT, INSERT, UPDATE, and DELETE let grantees execute the statements of the same names on the object.

Privileges Cascade

Privileges can cascade up; that is, privileges granted on some object can imply grants of privileges on other objects. These situations are covered by the following principles:

- If the grantee owns an updatable view, and is being GRANTed privileges on its leaf underlying table (the base table wherein the data finally resides, regardless of any intervening tables or views), these privileges are GRANTed for the view as well. If specified, the grant option also cascades up. There is only one leaf underlying table for an updatable view. (See CREATE VIEW.)

- If the grantee owns an updatable view that immediately references the table on which privileges are being GRANTed (in other words, if the reference appears in the FROM clause without an intervening view), these privileges can also cascade up, including the grant option, if applicable.

- If the grantee owns a view, updatable or not, that grantee already has the SELECT privilege on all tables referenced in its definition as well as on the view itself. If the grantee gains the grant option on SELECT on all the referenced tables, he also acquires the grant option on the SELECT privilege on the view.

In all these situations, the grantor of the privilege is ‘‘_SYSTEM,’’ which denotes an automatic grant.

For each privilege that is granted, a privilege descriptor is effectively created. (This is a theoretical construct used by the ISO standard to specify privilege behavior; it may not actually exist.) The privilege descriptor indicates:

- The grantee that has received the privilege.
- The privilege itself (the action that can be performed).
- The object on which the privilege is granted, which may be one of those listed above or a column.
- The grantor that conferred the privilege. For automatic grants, this is the built-in value ‘‘_SYSTEM.’’
- Whether the privilege is grantable (GRANTed with the grant option).

Multiple identical privilege descriptors are combined, so that a privilege granted twice by the same grantor need be revoked only once. Likewise, if two privilege descriptors differ only in that one confers grant option and the other does not, they are merged into a single privilege with grant option. If the grantor lacks the ability to grant the privileges attempted, a completion condition is raised — a warning that privileges were not granted.

Example

```
GRANT SELECT ON STAFF TO CLERK
```

7.5.12 INSERT

This command inserts rows into a table.

Syntax

```
INSERT INTO table_name
   [ (column_list)]
   { VALUES (constant_list)
)
```

Keyword	Description
table_name	Name of the table into which values are inserted.
column_list	Identifies the columns of the table into which the values are inserted. All columns not in the list receive their default values automatically. If any such column cannot receive defaults (for example, if they have the NOT NULL constraint, but have no other default value specified) the INSERT fails. If you omit the list, all columns of the table are the target of the insert. The number and order in which you list the columns must match the number and order of the output columns of the corresponding query.
constant_list	A simple list of values to insert.

Use

This statement enters one or more rows into *table_name*. The rows are the output rows of the subquery. These rows must have the same data types as the columns being inserted into.

You must have INSERT privileges on all named columns to issue an INSERT statement. The table may be a view. If so, the view must be updatable, in which case the new rows are inserted into the base table that contains the data from which the view is derived (the *leaf underlying table*).

The values to be inserted may be directly specified with a table value constructor (whose elements may include variables or parameters passed from applications) or derived from a query from information already present in the database.

Example

```
INSERT INTO STAFF VALUES
(10,'Sanders',15,'Clerk',7,12345.67,543.54)
```

7.5.13 REVOKE (Database privileges)

This command removes the privilege to perform an action on a database.

Syntax

```
REVOKE { privilege., ..} FROM  { grantee }

privilege ::=
     { CONNECT
     | DBA
     | RESOURCE }
```

Keyword	Description
privilege	Type of access, action, or privilege to remove.
grantee	User name(s) to allow privilege.

Use

This statement removes privileges from authorization IDs that have previously received them with the GRANT statement. Only a DBA can execute this statement.

Removing CONNECT privilege means that the grantee can no longer access the database. It has no effect on the objects owned by that grantee. Removing RESOURCE privilege means that the grantee can no longer create new objects.

Note: Be very careful removing DBA privilege: If there are no more DBAs then the GRANT and REVOKE statements can no longer be used.

Example

```
REVOKE CONNECT FROM CLERK
```

7.5.14 REVOKE (Object privileges)

This command removes the privilege to perform an action on a database object.

Syntax

```
REVOKE [ GRANT OPTION FOR ]
    { ALL PRIVILEGES } | { privilege., ..}
    ON object_name
    FROM PUBLIC | { grantee .,.. };
```

Keyword	Description
privilege	Type of access, action, or privilege to revoke.
object_name	Name of the object on which to revoke privileges.
grantee	User name(s) to revoke.

Use

This statement removes privileges from authorization IDs that have previously received them with the GRANT statement. Authorization IDs refer to users. The privileges follow the definitions and rules outlined under GRANT. The GRANT option is the ability to grant the privileges received in turn to others.

In any case, the revoker of the privilege is the same authorization ID that granted it, and all dependent privileges may be revoked. A privilege (privilege A) depends directly on another (privilege B) if either of the following sets of conditions is met:

1. Privilege A is grantable (has GRANT option)

and

2. The grantee of A is PUBLIC or the same as the grantee of B

and

3. A and B are both privileges for the same action on the same object.

OR

1. B is an automatically generated privilege, indicated by a grantor value of ‘‘_SYSTEM’’

and

2. The actions of the two privileges are the same

and

3. The grantee of A owns the object (which must be a table, translation or collation) on which the privileges exist.

and

4. Either privilege B is on a view referencing a table on which privilege A is the SELECT privilege (if it is a read-only view) or the privilege at hand (if it is an updatable one)

or

5. B is the USAGE privilege on a collation defined on a character set on which A is the USAGE privilege

or

6. B is the USAGE privilege on a translation that uses the character set on which A is the USAGE privilege as either source or target.

Example

```
REVOKE SELECT ON STAFF FROM CLERK
```

7.5.15 SELECT

Syntax

```
SELECT [ALL | DISTINCT ] select_list
   FROM [table_name | subquery] (corr_name)
      (, [table_name | subquery] (corr_name)) ...
   (WHERE search_condition)
   (GROUP BY column_name (, column_name) ...)
   (HAVING search_condition)
   (ORDER BY {column_name | select_list_number } (ASC | DESC))
...
```

Keyword	Description
select_list	List of columns to retrieve.
table_name	Name of table or tables in which the columns reside.

correlation_name	Also called range variable or alias, provides alternative name for the table whose name it follows; the definition lasts only for the duration of the statement. *Correlation names* are optional for base tables and views, but required for tables produced by subqueries.
search_condition	Criteria by which you identify rows on which you want to act.
column_name	Name of the column to evaluate with the search_condition.
select_list_number	Position in the select list of the column to evaluate with the search_condition.

Use

This is the statement used to formulate queries, which are requests for information from the database. To issue this statement you must have the SELECT privilege on all tables accessed. Queries may be stand-alone or used in the definitions of views and cursors. In addition, you can use them as subqueries, to produce values that are used within other statements including the SELECT statement itself. Sometimes a subquery is evaluated separately for each row processed by the outer query. Values from that outer row are used in the subquery. Queries of this type are called *correlated subqueries*.

The output of a query is itself a table, and the SELECT clause defines the columns of that table (the *output columns*).

Clauses of the SELECT statement are evaluated in the following order:

1. FROM
2. WHERE
3. GROUP BY
4. HAVING
5. SELECT
6. ORDER BY*

Note: * The order of records returned by AcuXDBC is always undefined unless you specify an ORDER BY clause. If a specific returning order is required, you must use the ORDER BY clause.

7.5.15.1 SELECT list (SELECT statement)

The SELECT list appears as the first clause in a SELECT statement, but it is not the first logical step. The other clauses produce a set of rows, the *source rows*, from which the output is derived.

The SELECT list determines which columns from these rows are output. It may directly output these columns, or it may use them in aggregate functions or value expressions. Value expressions can be NUMERIC, STRING, DATETIME, or INTERVAL. They may include aggregate functions and subqueries.

If DISTINCT is specified, the rows are compared and if any duplicate rows are found, only one copy appears in the output. The SELECT clause may contain any of the following:

- ***Aggregate functions*** — Functions that extract single values from groups of column values, for example, SUM or COUNT.
- ***An asterisk (*)*** — Causes all the columns of all tables listed in the FROM clause to be output in the order in which they appear in the FROM clause.
- ***qualifier*** — Where the *qualifier* is the table or correlation name referenced in the FROM clause. All columns of that (possibly derived) table are output, excluding common columns of joined tables.
- ***A value expression*** — Normally is (or includes) a column name from one of the tables identified in the FROM clause. Either the column's value is directly output or it becomes part of some expression, such as `AMOUNT * 3`.
- ***A specified column name*** — If the output columns are directly taken from one and only one column referenced in the FROM clause, it inherits the name of that column by default. You can override this name by using the AS clause. The names of columns not directly taken from input columns are implementation-dependent. You are not required to name any output columns by the SELECT clause, but may be required to by the context of the way the output columns are to be used (for example, in a view). The keyword AS is not required because it is implied.

If aggregate functions and value expressions are mixed, all the value expressions must be specified in a GROUP by clause.

7.5.15.2 FROM clause (SELECT statement)

The FROM clause names the source tables for the query. These tables may be:

- Tables or views named and accessed directly.
- Derived with a subquery that is part of the main query.
- Explicit joins.

The FROM clause determines the tables from which the data is taken or derived. These sources can include temporary or permanent base tables, views, results of subqueries, and other operations that return tables.

You can use correlation names to qualify ambiguous column references in the rest of the statement. You can choose to join a table to itself, which is treated as a join of two identical tables. In this case, you must use correlation names to distinguish between the two tables. The correlation names prefix the *column_name* and are separated by a period. The column name list is for renaming columns, just as they are in the SELECT clause.

The names used here, however, are not for the output; they are for references to the columns made in the remainder of the statement, particularly in the WHERE clause. They are optional, but may be required to clarify column references in some cases.

7.5.15.3 Joins

If more than one table is named in the FROM clause, they are all implicitly joined. This means that every possible combination of rows (one from each table) is derived. In addition, this concatenation is the table on which the rest of the query operates. The concatenated table is called a *Cartesian product* or *cross join*. Typically, you want to eliminate most of the rows and focus on the data you want. To do this, use the WHERE clause.

The standard join is also referred to as an Inner join and is simply the regular SQL WHERE clause. For example:

```
WHERE [table1.]column1 = [constant | [table2.]column2]
```

7.5.15.4 Outer Joins

AcuXDBC supports OUTER joins, including LEFT, RIGHT, and FULL. The outer joins follow the FROM clause and are of the form:

<LEFT | RIGHT | FULL> OUTER JOIN <tablename> ON <column1> = <column2>

where <column1> is a column in <tablename> and <column2> is a column in another FROM clause table.

7.5.15.5 WHERE clause (SELECT statement)

The WHERE clause defines the criteria that rows must meet in order to be selected for output.

The WHERE clause contains a predicate, which is a set of one or more expressions that can be TRUE, FALSE, or UNKNOWN. Values are compared according to:

Predicate Type	**Description**
NULLS	Compared to any value, including other NULLs, produce UNKNOWNs.
Character string types	Collating sequence.
Numeric types	Numerical order.
Date-time types	Chronological order.
Interval types	Magnitude.

These comparisons are expressed using the following operators: =, <, <=, >, =>, and < > (does not equal).

Operators such as "*" (multiplication) or "||" (concatenation) may be applied depending on the data type. In most situations, row value constructors may be used instead of simple value expressions.

In addition to the standard comparison operators, SQL provides the following special predicate operators. Assume that B and C are all value expressions, which can be column names or direct expressions (possibly using column names or aggregate functions) in the appropriate data type:

Predicate Operator	**Description**
B BETWEEN A AND C	Equal to (A <= B) AND (B <= C). A and C must be specified in ascending order. B BETWEEN C AND A is interpreted as (C <= B) AND (B <= A), which is FALSE if the first expression is TRUE, unless all three values are the same. If any of the values is NULL the predicate is UNKNOWN.
A IN (C, D, ., …)	This is true if A equals any value in the list.
A LIKE 'string'	This assumes that A is a character string and searches for the specified substring. Fixed and varying-length wild cards can be used.
A IS NULL	Specifically tests for NULLs. It can only be TRUE or FALSE, not UNKNOWN.
A *comp op* SOME | ANY *subquery*	SOME and ANY have equivalent meanings. The subquery produces a set of values. If, for any value V so produced, A *comp op* V is TRUE, then the ANY predicate is TRUE.
A *comp op* ALL *subquery*	Similar to ANY except that all the values produced by the subquery have to make A *comp op* V true.
EXISTS *subquery*	Evaluates to TRUE if the *subquery* produces any rows, otherwise, evaluates to FALSE. It is never UNKNOWN. To be meaningful, this phrase must use a correlated subquery.
UNIQUE *subquery*	If the *subquery* produces no identical rows, UNIQUE is TRUE, otherwise, it is false. For the purposes of this predicate, identical rows are devoid of NULLs, otherwise, they are not identical.
row value constructor MATCH *arguments subquery*	Tests for the presence of the constructed row among those of the table produced by the subquery. The arguments allow you to specify FULL or PARTIAL matches, and whether the matched row must be unique.

Predicate Operator	Description
row value constructor OVERLAPS *row value constructor*	This allows you to determine when two date or time periods overlap. You must use it with DATETIME data types, possibly in conjunction with INTERVAL data types.

These predicates are combined using the conventional Boolean operators AND, OR, and NOT. For TRUE and FALSE values, these have the conventional results. The rows selected by the WHERE clause go on to be processed by subsequent clauses.

7.5.15.6 GROUP BY clause (SELECT statement)

The GROUP BY clause groups identical output values in the named columns. Every value expression in the output column that includes a table column must be named in it unless it is an argument to aggregate functions. GROUP BY is used to apply aggregate functions to groups of rows defined by having identical values in specified columns.

If you don't use GROUP BY, either all or none of the output columns in the SELECT clause must use aggregate functions. If all of them use aggregate functions, all rows satisfying the WHERE clause, or all rows produced by the FROM clause (if there is no WHERE clause) are treated as a single group for deriving the aggregates.

The GROUP BY clause defines groups of output rows to which aggregate functions (COUNT, MIN, AVG, and so on) can be applied. If you do not use this clause and elect to use aggregate functions, the column names in the SELECT clause must all be contained in aggregate functions, and the functions are applied to all rows to satisfy the query.

Otherwise, each column referenced in the SELECT list outside an aggregate function must be a grouping column and be referenced in this clause. All rows output from the query that have all grouping column values equal, constitute a group. (For the purposes of GROUP BY, all NULLs are considered equal). The aggregate function is applied to each such group.

7.5.15.7 HAVING clause (SELECT statement)

The HAVING clause defines criteria that the groups of rows defined in the GROUP BY clause must satisfy to be output by the query.

Just as the WHERE clause defines a predicate to filter rows, HAVING is applied after grouping to define a similar predicate to filter the groups based on the aggregate values. It is needed to test for aggregate function values, as these are not derived from single rows of the Cartesian product defined by the FROM clause, but from groups of such rows, and therefore cannot be tested in a WHERE clause.

7.5.15.8 ORDER BY clause (SELECT statement)

ORDER BY forces the output of queries to emerge in a particular sequence.

The ORDER BY clause sorts the output. The rows are sorted according to the values in the columns listed. The first column listed gets the highest priority, and the second column determines the order within duplicate values of the first, the third within duplicate values of the second, and so on. You can specify “ASC” (for ascending, the default) or “DESC” (descending) independently for each column.

Character sets are sorted according to their collations. You can also use integers rather than names to indicate columns. The integers refer to the placement of the column among those in the output, so that the first column is indicated with a “1”, the fifth with a “5”, and so on. If any output columns are unnamed, you must use a number.

Note: The order of records returned by AcuXDBC is always undefined unless you specify an ORDER BY clause. If a specific returning order is required, you must use the ORDER BY clause.

7.5.15.9 Possibly Nondeterministic Queries

In some cases the same query can produce different output tables on different implementations because of subtle implementation-dependent behaviors. Such queries are called *possibly nondeterministic queries*. A query is possibly nondeterministic if any of the following is true:

- It specifies DISTINCT and the data type of at least one column of the source row is a character string.
- One column of the source rows is a character string data type and is used in either the MIN or the MAX aggregate function.
- A character set column is used as a grouping column or in a UNION.
- A HAVING clause uses a character string column within a MIN or MAX function.
- It uses UNION without specifying ALL.

Possibly nondeterministic queries cannot be used in constraints.

Examples

```
SELECT DEPT, AVG(SALARY) FROM STAFF
GROUP BY DEPT
```

7.5.16 SET OPTION

This command enables and dis-enables numerous administrative options.

Syntax

`SET OPTION` *`optiontype  param1 [param2]`*

Option	Param1	Param2	Description
COMPSORT	ON | OFF		Sets sort page compression (default is “ON”).
DATETIME	[n] ‘string’		Allows the user to modify the DATETIME formatting string. The default is DD-MON-RR and the maximum size for the string is 64 bytes. The user can define up to three DATETIME formatting strings. These are identified by the optional *n* parameter.
ERROR	ON | OFF		Sets internal error dumping.
EXPR	ON | OFF		Sets internal expression dumping.
HASH	ON | OFF		Sets internal hash dumping.

Option	Param1	Param2	Description
HEAPBLOCK SIZE	bytes		Allows the user to set the heap block size used in allocating memory. Larger sizes require less CPU overhead but may result in excessive memory usage. The range is 0 to 1000000. Setting "0" means that the exact required size will be used.
LOGFILE	'filename'		Sets the name of the debugging logfile.
MERGESIZE	records		Sets the number of records used to check for duplicates while processing an IN or UNION clause. The default is 1000. Increase this value if you get a "Multi RID overflow" error.
OJNESTON	ON | OFF		Allows nested ON clauses (default is "OFF").
OPTIMIZE	ON | OFF		Allows the user to control whether or not Optimization is used to evaluate statements. (Default is "ON").
PLAN	ON | OFF		Sets query plan dumping.
SORTPAGES	totalpages	mempages	Allows the user to modify the amount of disk and memory storage used for sort operations. The *totalpages* parameter is the total number of 4096 byte pages to use and *mempages* is the number of these pages kept in memory. *totalpages* must be greater than or equal to *mempages*. The default values are 10000 and 1000. The default values are fine for most users. If you get a message indicating that Virtual Memory has been exceeded, then increase the *totalpages* value.

Option	Param1	Param2	Description
TRACE	ON | OFF		Sets internal trace dumping.
TREE	ON | OFF		Sets internal tree dumping.

Use

This statement is used to set a number of database administrative options, as described in the table above.

Examples

```
SET OPTION SORT 8000 2000
SET OPTION DATETIME DD-MM-YYYY
SET OPTION LOGFILE mylogfile
SET OPTION ERROR ON
```

7.5.16.1 SET PASSWORD

This statement is used to change the user's password.

Syntax

```
SET PASSWORD old_password new_password
```

Keyword	Description
old_password	Current password.
new_password	New password.

Use

This statement is used to change the user's password.

Examples

```
SET PASSWORD mypass newpass
SET PASSWORD mypass ''42pass''
```

7.5.17 UPDATE

This command changes the data in a table.

Syntax

```
{UPDATE table_name (correlation_name)
   SET column_name = {expression | NULL}
      (column_name = {expression | NULL} ) ...
   (WHERE search_condition)
```

Keyword	Description
table_name	The name of an existing table that you can access. May include the owner's name if it is not you. For example, owner.table.
correlation_name	Also called an alias. Used to relabel the name of the reference in other clauses in the statement.
column_name	A column within the table. Parentheses are only required if the column list contains more than one column.
expression	The operation or function to execute on the specified column_names.
search_condition	A valid condition that evaluates to TRUE, FALSE, or UNKNOWN.

Use

This statement changes one or more column values in an existing row of a table. The table may be a base table or view. You can set any number of columns to values and follow the whole *column_name = value_expression* clause with a comma if there is another such to follow. As an alternative to an explicit value, you can set the column to NULL or to the DEFAULT defined for the column (see CREATE TABLE).

You can use *value_expression* to refer to the current values in the table being updated. Any such references refer to the values of all of the columns before any of them were updated. This allows you to do such things as double all column values (if numeric) by specifying:

```
column_name = column_name * 2
```

You can also swap values between columns. Value expressions can also use subqueries.

The UPDATE is applied to all rows that fulfill the WHERE clause, which is one of two types. The WHERE predicate form is like the WHERE predicate clause in the SELECT statement; it uses an expression that can be TRUE, FALSE or UNKNOWN for each row of the table to be updated, and the UPDATE is performed wherever it is TRUE.

Be careful of omitting the WHERE clause; if you do, the UPDATE is performed on every row in the table. You can use the WHERE CURRENT OF form in static or dynamic SQL if the cursor direction is updatable (in other words, not through views) and provided the target table is open and positioned on a row. The UPDATE is applied to the row on which it is positioned. When using WHERE CURRENT OF in dynamic SQL, you can omit the table name from the UPDATE clause, as the table in the cursor is implied.

In either case, for the UPDATE to be successful, the following conditions must be met:

- The statement issuer must have the UPDATE privilege on each column of the table being set.
- If the target table is a view, it must be updatable.
- If the current transaction is read-only, the target table must be temporary.
- If the UPDATE is done through a cursor that specifies ORDER BY, it may not set the values of any columns specified in the ORDER BY clause.
- If the target table is a view, the *value_expression* in the SET clause must not directly, or though views, reference its leaf-underlying table (the base table where the data ultimately resides).
- The *value_expression* may not use aggregate functions except in subqueries.
- Each column of the target table can only be altered once by the same UPDATE statement.
- If the UPDATE is on a cursor that specified FOR UPDATE, each column being set must have been specified or implied by that FOR UPDATE.
- If the UPDATE is made through a view, it may be constrained by a WITH CHECK OPTION clause.

Example

The following statement updates every salary in department 15:

```
UPDATE STAFF SET SALARY = SALARY * 1.10 WHERE DEPT = 15
```

7.6 Functions Supported by AcuXDBC

AcuXDBC has a number of built-in functions that help you create your SQL statements. You can use them in the select-list to modify the result table or in the WHERE clause to limit the number of qualifying rows. They are also valid in INSERT/UPDATE/DELETE statements.

7.6.1 ASCII

```
ASCII(char)
```

Returns the integer value of *char*.

7.6.2 CHAR_LENGTH

```
CHAR_LENGTH(expr)
```

Returns the length of the character expression.

7.6.3 CHR

```
CHR(n)
```

Returns the character value of *n*.

7.6.4 CONCAT

```
CONCAT(char1,char2)
```

Returns the concatenation of *char1* and *char2*.

7.6.5 CONVERT

```
CONVERT(expr1,datatype)
```

Returns the value of *expr1* converted into *datatype*, which is one of the following:

SQL_BIGINT
SQL_BINARY
SQL_BIT
SQL_CHAR
SQL_DATE
SQL_DECIMAL
SQL_DOUBLE
SQL_FLOAT
SQL_INTEGER
SQL_LONGVARBINARY
SQL_LONGVARCHAR
SQL_REAL
SQL_SMALLINT
SQL_TIME
SQL_TIMESTAMP
SQL_TINYINT
SQL_VARBINARY
SQL_VARCHAR

7.6.6 CURDATE

```
CURDATE()
```

Returns the current date.

7.6.7 CURTIME

```
CURTIME()
```

Returns the current time.

7.6.8 DATABASE

```
DATABASE()
```

Returns the name of the database.

7.6.9 DAYNAME

```
DAYNAME(expr)
```

Returns the name of the day of the week for the date specified by *expr.*

7.6.10 DECODE

```
DECODE(expr,value,result[,value,result]...,default)
```

Compares *expr* with each value and returns either the first matching value's result or the default value if no values match.

7.6.11 HOUR

```
HOUR(expr)
```

Returns the hour for the datetime specified by *expr.*

7.6.12 IFNULL

```
IFNULL(expr1,expr2)
```

Returns *expr2* if *expr1* is NULL, otherwise it returns *expr1*.

See also

NVL

7.6.13 INSTR

```
INSTR(char1,char2[,n[,m]])
```

Returns the position of *char2* within *char1*. If *n* is specified and positive, then the search begins *n chars* into *char1*. If *n* is negative, then the search begins *n chars* from the end of *char1*. If *m* is specified, then the *m*th occurrence of *char2* in *char1* is located. If *char2* does not exist within *char1*, then "0" is returned.

See also

LOCATE

7.6.14 LEFT

```
{fn LEFT(expr,n)}
```

Returns the first *n* characters of *expr.*

The LEFT and RIGHT built-in functions are also SQL keywords. You must enclose them in {fn <LEFT | RIGHT>(parameters)}.

7.6.15 LENGTH

```
LENGTH(expr)
```

Returns the length of the character representation of *expr.*

7.6.16 LOCATE

```
LOCATE(char1,char2[,n[,m]])
```

Returns the position of *char2* within *char1*. If *n* is specified and positive, then the search begins *n chars* into *char1*. If *n* is negative, then the search begins *n chars* from the end of *char1*. If *m* is specified, then the *m*th occurrence of *char2* in *char1* is located. If *char2* does not exist within *char1*, then "0" is returned.

See also

INSTR

7.6.17 LCASE

```
LCASE(expr)
```

Returns the lowercase representation of *expr.*

7.6.18 LTRIM

```
LTRIM(expr)
```

Returns the character representation of *expr* trimmed of leading blanks.

7.6.19 NOW

```
NOW()
```

Returns the current date and time.

7.6.20 NVL

```
NVL(expr1,expr2)
```

Returns *expr2* if *expr1* is NULL, otherwise it returns *expr1*.

See also

IFNULL

7.6.21 POSITION

```
POSITION(char1 IN char2)
```

Returns the position of *char1* within *char2*. If *char1* does not exist within *char2*, then “0” is returned.

7.6.22 RIGHT

```
{fn RIGHT(expr,n)}
```

Returns the last *n* characters of *expr.* The LEFT and RIGHT built-in functions are also SQL keywords. You must enclose them in {fn <LEFT | RIGHT>(parameters)}.

7.6.23 ROUND

```
ROUND(n[,m])
```

Returns the rounded value of *n* based on the value of *m*. If *n* is a numeric value, it can be either positive or negative.

A positive value *m* specifies the digits to the right of the decimal point. A negative value *m* specifies the digits to the left of the decimal point. If *m* is *0* or not specified, the value is rounded at the decimal point.

If *n* is a date value, then *m* can be one of the following:

SCC,CC	Century
SYYY, YYYY, YEAR, SYEAR, YYY, YY, Y	Year
Q	Quarter
MONTH, MON, MM	Month
WW,W	Start of Week
DDD,DD,J (default)	Day
DAY, DY, D	Nearest Sunday
HH, HH12, HH24	Hour
MI	Minute

7.6.24 RTRIM

```
RTRIM(expr)
```

Returns the character representation of *expr* trimmed of trailing blanks.

7.6.25 SQRT

```
SQRT(n)
```

Returns the square root of *n*.

7.6.26 SUBSTR

```
SUBSTR(char,m[,n])
```

Returns a substring of *char*, beginning at position *m* for *n* characters. If you specify *m=0*, the whole string is returned. If you specify a negative number, the function returns the number of characters specified from the end of the string. If you don't specify *n*, the default is to return all characters starting from *m*.

7.6.27 SUBSTRING

```
SUBSTRING(char,m[,n])
```

See SUBSTR().

7.6.28 SYSDATE

```
SYSDATE
```

Returns the current system date and time.

7.6.29 TO_CHAR

```
TO_CHAR(expr[,fmt])
```

Returns the character representation of *expr* based on the *fmt* string or the default for expr's data type. If *expr* is already a character string, then *expr* is not converted.

If *expr* is a numeric value, then fmt can be:

%	Percent sign at right of number.
$	Dollar sign at left of number.
B	Display zero as blank.
0	Display leading zeros.
9	A digit position.
other	Delimiting character (not leading).

The default mask is as many nines (9s) as required for the number's precision and scale.

If *expr* is a date value, then *fmt* can be:

YYYY	Four digit year.
YY	Two digit year.
RR	Two digit year in another century.
MM	Two digit month of year (01-12).
MON	Three character month (all uppercase).
mon	Three character month (all lowercase).
Mon	Three character month (initial cap).
MONTH	Fully named month (all uppercase).
month	Fully named month (all lowercase).
Month	Fully named month (initial cap).
DDD	Three digit day of year (001-356).
DD	Two digit day of month (01-31).
D	Single digit day of week (1-7).
DY	Three character day (all upper case).
dy	Three character day (all lowercase).
Dy	Three character day (initial cap).
DAY	Fully named day (all uppercase).
day	Fully named day (all lowercase).

Day	Fully named day (initial cap).
HH12	Two digit hour (00-11).
HH,HH24	Two digit hour (00-23).
MI	Two digit minutes (00-59).
SS	Two digit seconds (00-59).
SSSSS	Seconds past midnight (0000-86399).
J	Julian day.
Q	Single digit quarter of year (0-4).
W	Single digit week of month (1-4). The week begins on Sunday.
WW	Two digit week of year (01-52).
other	Delimiting character.

To add character extensions to the values that represent counting, such as *ST*, *ND*, *RD*, or *TH*, simply add *th* to any uppercase digit mask. The function correctly interprets the extension based on the last digit and the case based on the mask's case.

Put embedding characters that are valid masks inside double quotes (").

The default mask is DD-MON-YY.

7.6.30 TO_DATE

```
TO_DATE(expr[,fmt])
```

Returns the datetime representation of *expr* based on the *fmt* string or the default for expr's data type. If *expr* is already a datetime, then the value is not converted.

The *expr* can be an integer or numeric value. If it is an integer value, it represents the number of days since year zero. If *expr* is a numeric, then the integer portion represents the number of days since year zero and the fractional portion represents the part of the last day.

If *expr* is a char value, then fmt can be:

YYYY	Four digit year.
YY	Two digit year.
RR	Two digit year in another century.
MM	Two digit month of year (01-12).
MON	Three character month (all uppercase).
mon	Three character month (all lowercase).
Mon	Three character month (initial cap).
DDD	Three digit day of year (001-356).
DD	Two digit day of month (01-31).
HH12	Two digit hour (00-11).
HH,HH24	Two digit hour (00-23).
MI	Two digit minutes (00-59).
SS	Two digit seconds (00-59).
SSSSS	Seconds past midnight (0000-86399).
J	Days since 1/1/1.
other	Delimiting character.

7.6.31 TO_NUMBER

```
TO_NUMBER(expr[,fmt])
```

Returns the numeric representation of *expr* based on the *fmt* string or the default for expr's data type.

If *expr* is already a numeric, then the value is not converted. If *expr* is a char value, then *fmt* can be:

%	Percent sign at right of number.
$	Dollar sign at left of number.
B	Display zero as blank.
0	Display leading zeros.

9	A digit position.
other	Delimiting character (not leading).

The default mask is as many nines (9s) as required for the number's precision and scale.

If *expr* is a date value, then the returned numeric represents the number of days since year 0 and the portion of the last day.

7.6.32 TRANSLATE

```
TRANSLATE(char,from,to)
```

Returns *char* with all characters in from replaced with the corresponding ones in to. If the number of characters in *to* is a multiple of the number of characters in *from*, for each character in *from*, that multiple of characters from *to* replaces that single character. If *to* is empty, all characters found in *from* are deleted.

7.6.33 TRUNC

```
TRUNC(value)
```

Returns the truncated value. If *value* is of type datetime then the hours, minutes, and seconds are set to zero. Otherwise it is treated as a number and the fractional part is removed.

7.6.34 UCASE

```
UCASE(expr)
```

Returns the uppercase representation character representation of *expr*.

7.6.35 USER

```
USER()
```

Returns the username of the current connection.

8 Working with Windows and Java Applications

Key Topics

8.1 Working With Windows Applications

The most common way to interact with your Vision tables (now an ODBC data source) is using Microsoft Query, a database query tool that comes with Microsoft Office. You can invoke Query directly from Windows applications like Microsoft Word and Excel. Once in Query, you can use the point-and-click interface to build up queries graphically, or you can bring up a texxt box to manually enter SQL statements for execution by AcuXDBC. ODBC calls to the Vision tables are totally transparent when you use Query. Other Windows programs, like Microsoft Access, Crystal Reports Professional, and the Microsoft Visual Basic development system, have their own interfaces to ODBC data sources.

This section describes how you can access Vision data from three of the most popular Windows applications: Microsoft Word, Excel, and Access. For instructions on accessing an ODBC data source from other Windows applications, refer to the application-specific user documentation. The examples in this chapter assume you have performed all the setup tasks applicable to your environment, such as creating the sample database and a data source name (DSN).

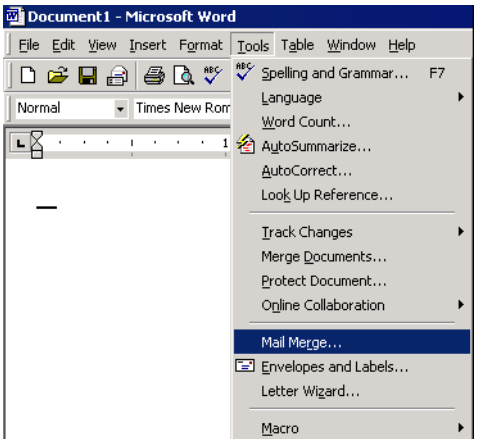
Note: It is important to be aware of how many fields your application supports for a primary key. For example, Vision files can support 16 columns in a primary key. Microsoft Access, on the other hand, supports only ten columns in a primary key. If your application does not support as many fields as your data, you can receive unexpected results.

8.1.1 Accessing Data From Word 2000

The following procedure describes how to access your Vision data from Microsoft Word 2000, the word processing component of Microsoft Office. Accessing Vision tables from Microsoft Word 2003 is described in the **Section 8.1.2**. If you have a different version of Word, your steps may be slightly different.

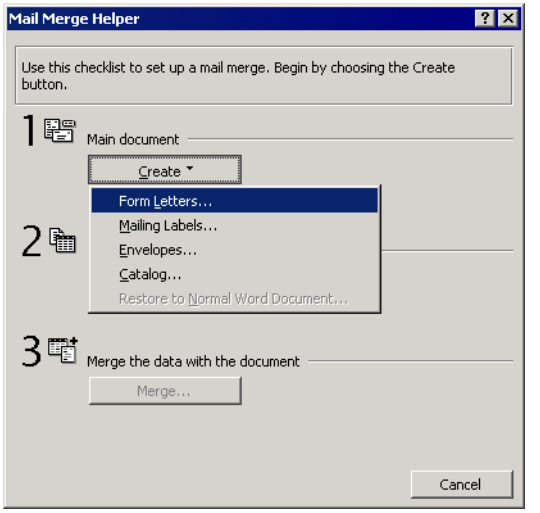
1. Start Microsoft Word and open the document into which you want to insert data.

2. Select **Mail Merge** from the Tools menu. If your Microsoft Office menus are set to "Show recently used commands first," you may need to click on the expansion arrows at the bottom of the Tools menu to see the Mail Merge option.

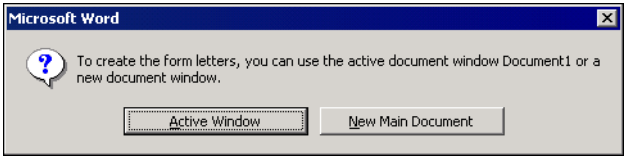


The Mail Merge Helper dialog box appears.

3. Under "Main Document," click **Create** and select **Form Letters**.

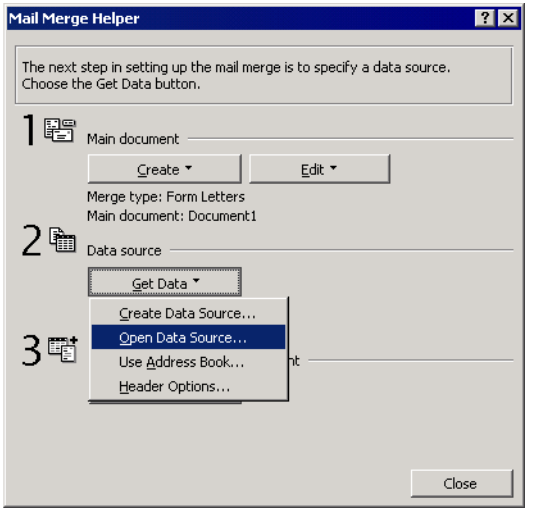


4. Because you are already working in the document into which you want to insert data, click **Active Window**.

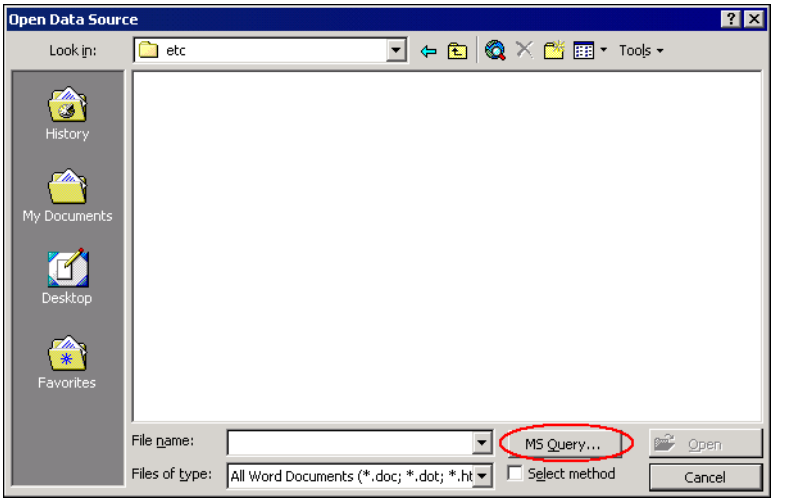


You are then returned to the Mail Merge Helper dialog box.

5. Under "Data Source," click **Get Data**, and select **Open Data Source**.



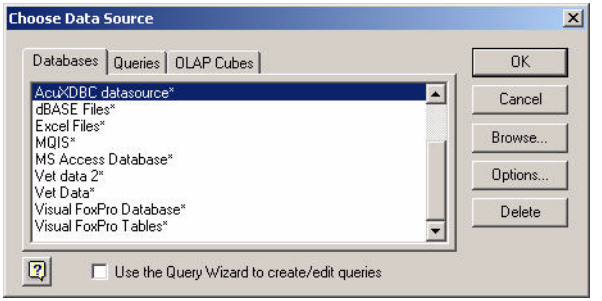
6. Click **MS Query** in the Open Data Source dialog box. (If you do not have an MS Query button, you need to install Microsoft Query on your machine.)



This starts Microsoft Query, and opens the Choose Data Source dialog box. If you want to use Microsoft Query's "Query Wizard" to create your SQL query, leave the "Use Query Wizard" check box selected. Refer to Microsoft Query's online documentation for instructions on using the Query Wizard.

For simplicity, clear the Query Wizard check box.

7. Select the DSN corresponding to the data source you want to access and click **OK**. (Refer to **section 5.8, "Setting Up Data Source Names (DSNs) on Client,"** for instructions on creating DSNs.).



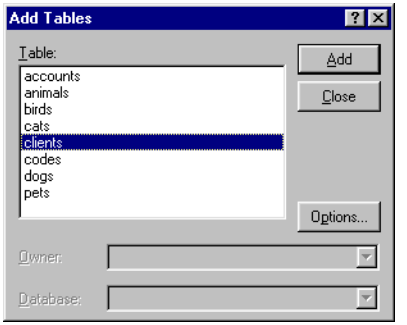
If you do not see your DSN listed, do one of the following:

- Click **Browse** and navigate to "c:\Program Files\Common Files\ odbc\DataSources", then double-click the name of your DSN. This adds the DSN to the list.

- Click **Options** and enter the path to the directory containing your DSN, then click **Add**. Now the contents of that directory will be listed. For more information, see the Microsoft Query documentation.

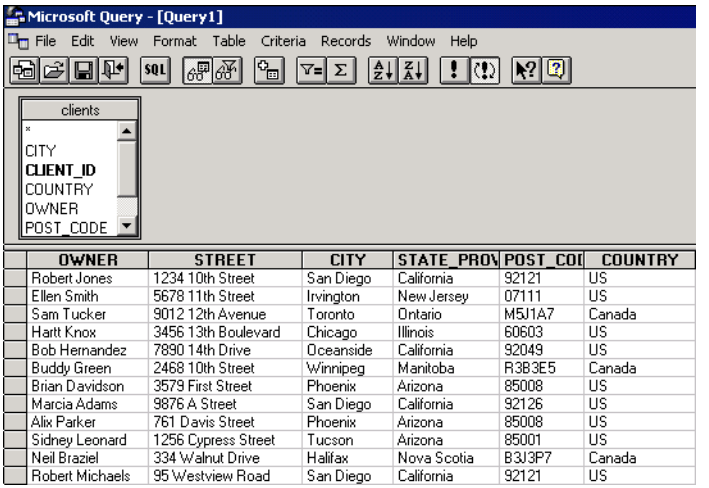
8. The AcuXDBC Login dialog box appears if you or an administrator established database security methods during AcuXDBC setup Enter a valid User ID and Password, and click **OK**. If no security methods have been established, a User ID and Password are not required and the login dialog box will not appear.

9. Select the table or tables that you want to add (for example, “clients”), and then click **Add**. Close the box when you’re done.



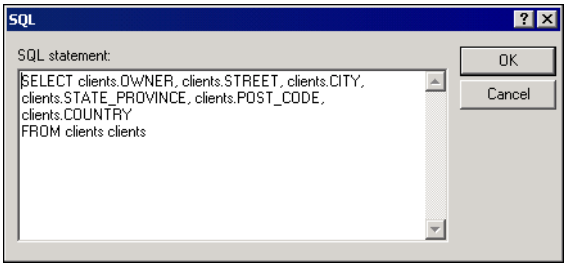
For each table you selected, a list box is displayed on the Microsoft Query screen. This box lists all of the columns in the associated table.

10. Double-click each column that you want to read into your Word document, and that column appears on the screen. If you want to add all columns, double-click the “*” at the top of the list box.



OWNER	STREET	CITY	STATE_PROV	POST_COI	COUNTRY
Robert Jones	1234 10th Street	San Diego	California	92121	US
Ellen Smith	5678 11th Street	Irvington	New Jersey	07111	US
Sam Tucker	9012 12th Avenue	Toronto	Ontario	M5J1A7	Canada
Hartt Knox	3456 13th Boulevard	Chicago	Illinois	60603	US
Bob Hernandez	7890 14th Drive	Oceanside	California	92049	US
Buddy Green	2468 10th Street	Winnipeg	Manitoba	R3B3E5	Canada
Brian Davidson	3579 First Street	Phoenix	Arizona	85008	US
Marcia Adams	9876 A Street	San Diego	California	92126	US
Alix Parker	761 Davis Street	Phoenix	Arizona	85008	US
Sidney Leonard	1256 Cypress Street	Tucson	Arizona	85001	US
Neil Braziel	334 Walnut Drive	Halifax	Nova Scotia	B3J3P7	Canada
Robert Michaels	95 Westview Road	San Diego	California	92121	US

If you prefer, you can select columns or further refine your query by clicking the SQL button and entering your own SQL statements such as the one shown below.



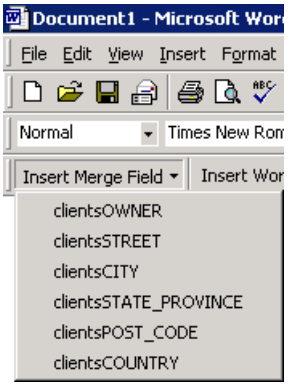
Click **OK** when you're done. (For a description of the SQL commands supported by AcuXDBC, refer to Chapter 7, **section 7.4**)

11. Select **Return Data to Microsoft Word** from the Microsoft Query File menu.

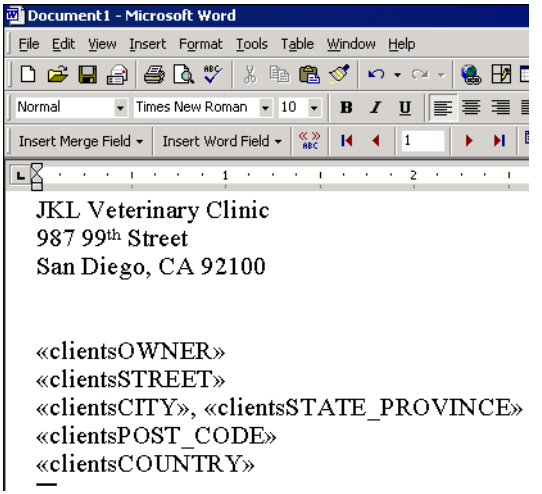
12. If no database security methods have been established, the AcuXDBC Login dialog box does not appear. If database security methods have been established and you previously supplied a correct user ID and password, this step will not be necessary.

13. Click **Edit Main Document** to continue preparing your document. Notice that new mail merge buttons are added to the toolbar.

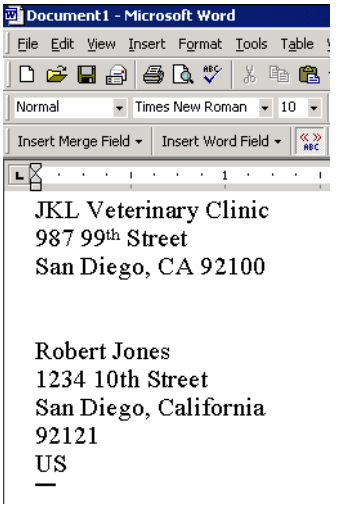
14. Place the cursor in your document where you want to insert the data. Click **Insert Merge Field** to see a drop-down list of table columns. These are the columns that you selected in Microsoft Query.



15. Select the merge field you wish to insert. The control characters for this field are placed in your document.



16. To display the data that corresponds to each field code, click the **<<ABC>>** button on the Mail Merge toolbar. Use the control buttons on the Mail Merge toolbar to move to the next, previous, first, or last record in the table.



17. Finish your document as you normally would. Whenever you want to insert a data field, click **Insert Merge Field** once again and select the appropriate field.

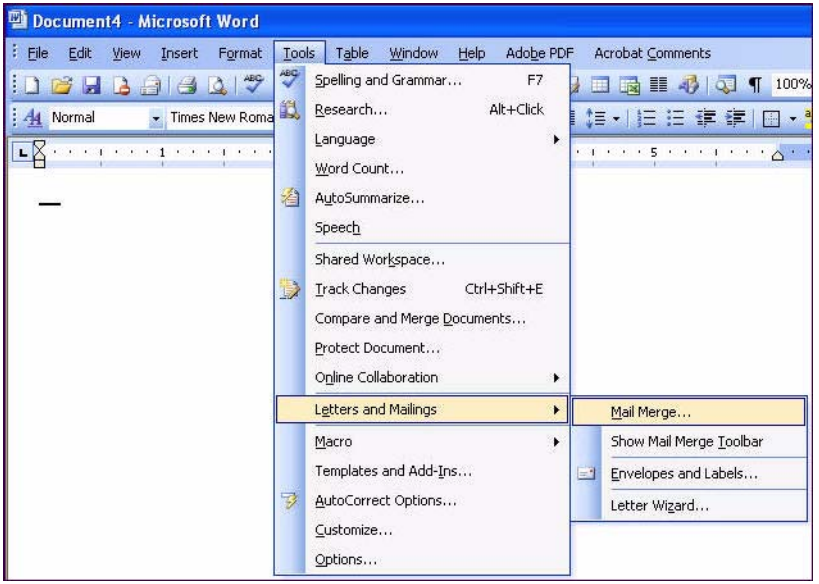
For more instructions on using Word's mail merge feature, refer to your Microsoft documentation.

8.1.2 Accessing Data From Word 2003

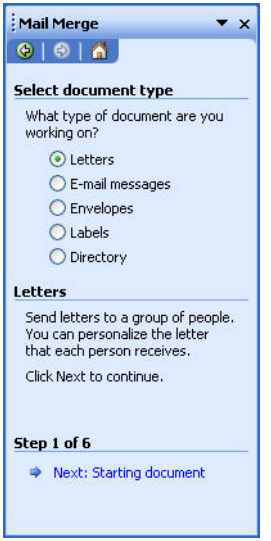
The following procedure describes how to access your Vision data from Microsoft Word 2003. This procedure uses Word's Mail Merge task pane and wizard to create form letters. You can also create the letters using Word's Mail Merge toolbar. If you are an experienced Word user, you may favor this method over the Mail Merge task pane approach, but be aware your steps will differ from what is depicted here.

1. Start Microsoft Word and open a new document, or if you have a form letter already prepared, open that document.

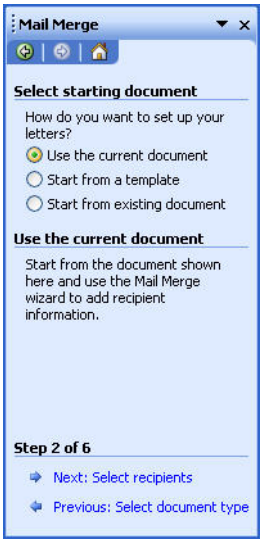
2. Select **Letters and Mailings** from the Tools menu and then **Mail Merge** from the submenu. If your Microsoft Office menus are set to "Show recently used commands first," you may need to click the expansion arrows at the bottom of the Tools menu to see the Letters and Mailings option.



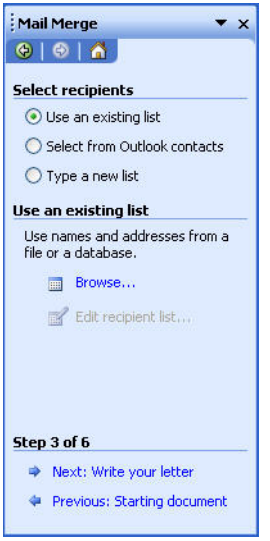
3. The Mail Merge task pane appears. Select the **Letters** radio button, then click **Next** at the bottom of the pane.



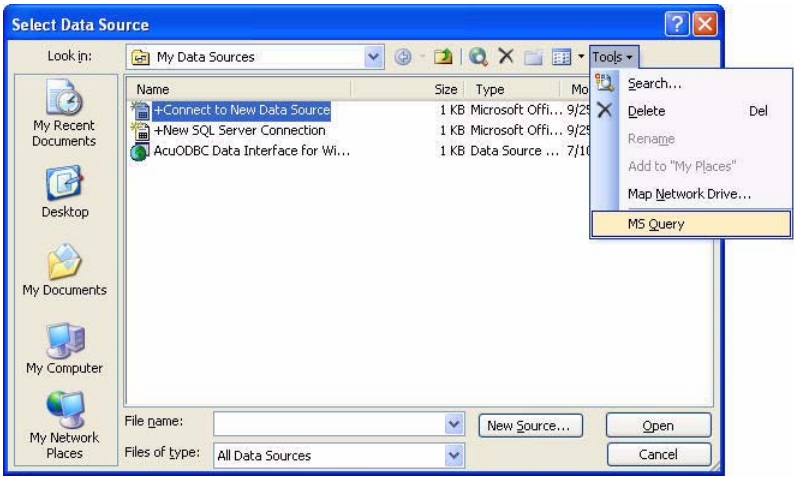
4. Select **Use the current document**, then click **Next**.



5. Select **Use an existing list**, then click **Browse**.



6. The Select Data Source dialog box appears. From this dialog box, select the **Tools** menu, then select **MS Query**. (If you do not have an MS Query option, you need to install MS Query on your machine.)



This starts Microsoft Query, and opens the Choose Data Source dialog box.

7. For simplicity, clear the **"Use the Query Wizard"** check box if it is checked. Select the DSN that corresponds to the data source you want to access and click **OK**. (Refer to **section 5.8, "Setting Up Data Source Names (DSNs) on Client,"** for instructions on creating DSNs.)

If you do not see your DSN listed, do one of the following:

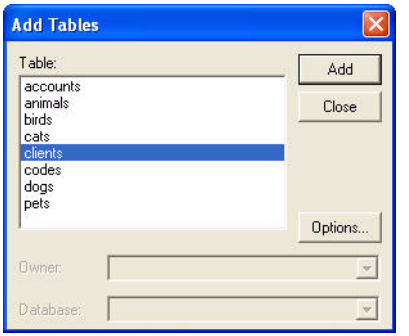
- Click **Browse** and navigate to "c:\Program Files\Common Files\odbc\DataSources", then double-click the name of your DSN. This adds the DSN to the list.

- Click **Options** and enter the path to the directory containing your DSN, then click **Add**. Now the contents of that directory will be listed. For more information, see the Microsoft Query documentation.

8. The AcuXDBC Login dialog box appears if you or an administrator established database security methods during AcuXDBC setup. Enter a valid User ID and Password, and click **OK**. If no security methods have been established, a User ID and Password are not required, and the login dialog box does not appear.

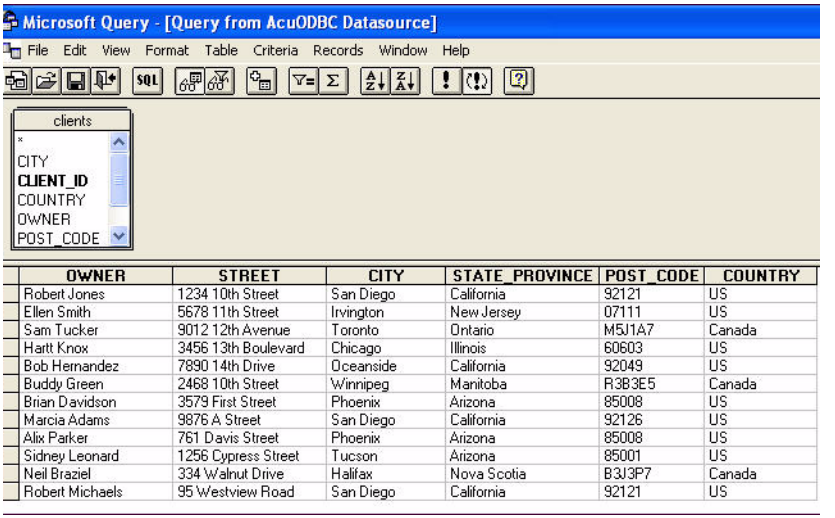
9. The Add Tables dialog box appears. Select the table or tables that you want to add (for example, "clients"), and then click **Add**. Close the box when you're done.

Tip: You can display just tables of a given user by selecting from the owner drop-down-list.



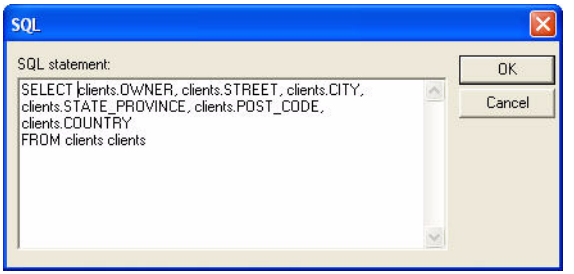
For each table you selected, a list box is displayed on the Microsoft Query screen. This box lists all of the columns in the associated table.

10. Double-click each column that you want to read into your Word document, and that column appears on the screen. If you want to add all columns, double-click the "*" at the top of the list box.



OWNER	STREET	CITY	STATE_PROVINCE	POST_CODE	COUNTRY
Robert Jones	1234 10th Street	San Diego	California	92121	US
Ellen Smith	5678 11th Street	Irvington	New Jersey	07111	US
Sam Tucker	9012 12th Avenue	Toronto	Ontario	M5J1A7	Canada
Hartt Knox	3456 13th Boulevard	Chicago	Illinois	60603	US
Bob Hernandez	7890 14th Drive	Oceanside	California	92049	US
Buddy Green	2468 10th Street	Winnipeg	Manitoba	R3B3E5	Canada
Brian Davidson	3579 First Street	Phoenix	Arizona	85008	US
Marcia Adams	9876 A Street	San Diego	California	92126	US
Alix Parker	761 Davis Street	Phoenix	Arizona	85008	US
Sidney Leonard	1256 Cypress Street	Tucson	Arizona	85001	US
Neil Braziel	334 Walnut Drive	Halifax	Nova Scotia	B3J3P7	Canada
Robert Michaels	95 Westview Road	San Diego	California	92121	US

If you prefer, you can select columns or further refine your query by clicking the SQL button and entering your own SQL statements such as the one shown below.

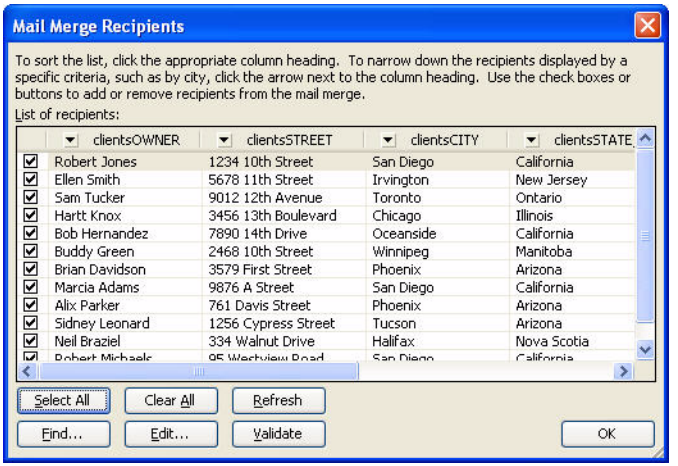


Click **OK** when you're done. (For a description of the SQL commands supported by AcuXDBC, refer to Chapter 7, **section 7.4**.)

11. Select **Return Data to Microsoft Word** from the Microsoft Query File menu.

12. If no database security methods have been established, (the database is open to all users) the AcuXDBC Login dialog box reappears. Simply click **OK**. If database security methods have been established and you previously supplied a correct user ID and password, this step will not be necessary.

13. The Mail Merge Recipient dialog box appears. Make sure all the records are checked and click **OK**. Note that no records actually appear after you click OK. This is expected behavior, as this step is simply making the data records available to Microsoft Word. Additional steps are necessary to actually insert the data records into your document.



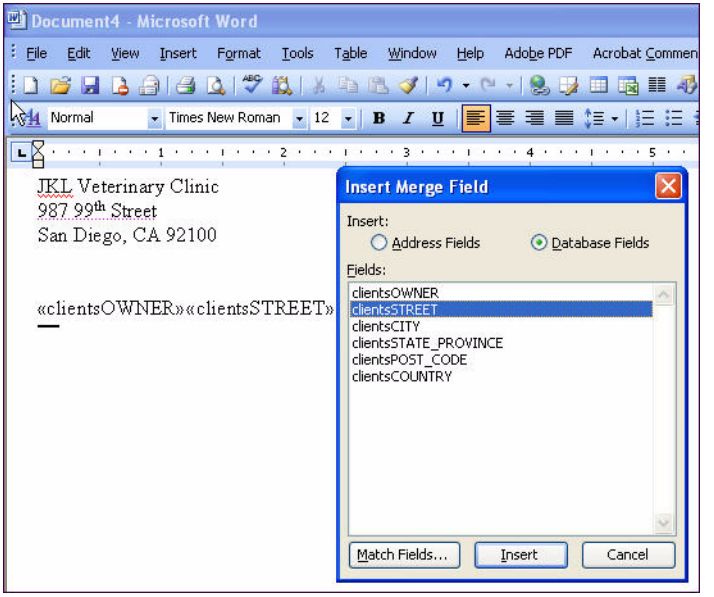
The Mail Merge Recipients dialog box provides options for refining your mail merge list. For example, you can add or delete records, and specify certain record criteria. Refer to Microsoft Word documentation for help on using the commands appearing in this dialog box.

14. Click **Next** on the Mail Merge task pane and continue preparing the body of your letter as needed.

15. Place the cursor in your document where you want your customer data to appear. Click **More items** on the Mail Merge task pane.

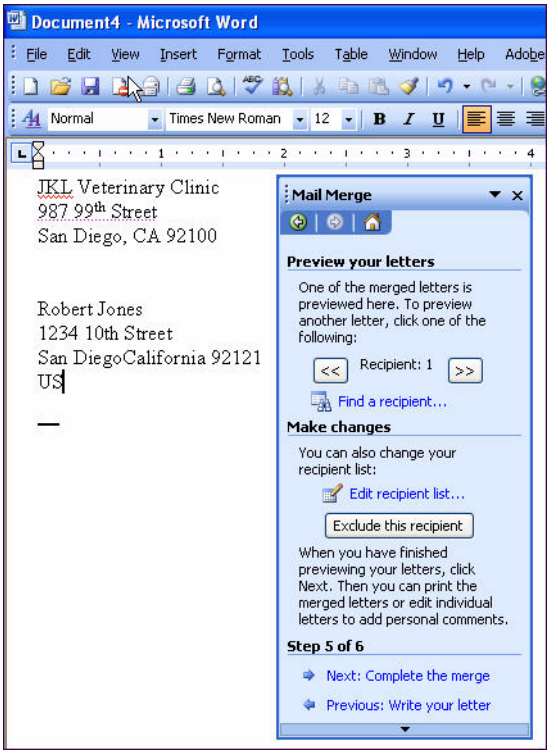
16. The Insert Merged Field dialog box appears. Make sure the Database Fields radio button is selected. Click on the field name you want to appear first in your document and click **Insert**. Click on the next field you want and click **Insert** again. Repeat this process for each field that you want to add to your document. Once you have added all desired fields, **Close** the Insert Merge Field dialog box.

Note: This dialog box does not allow you to select multiple fields and insert them all at once. You must select and insert them individually.



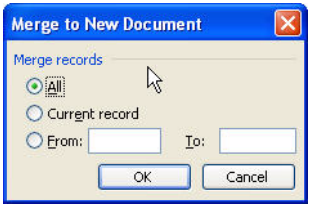
17. Format your fields as desired, then click **Next** on the Mail Merge task pane.

18. The data corresponding to each field is displayed. Notice the options on the Mail Merge task pane to preview or edit your letters. Once satisfied with your letters, click **Next** on the Mail Merge task pane.



19. Click **Edit Individual Letters** on the Mail Merge task pane.

20. The Merge to New Document dialog box appears. Notice your options for selecting which records to merge. Select **All,** then click **OK**.



21. A new document containing individual letters for each record is created. Edit and save your letter document as you normally would. You can also save your original document containing the mail merge query and reuse it to create additional mail merge letters.

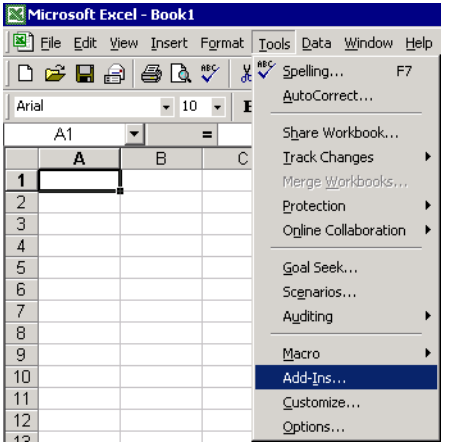
For more instructions on using Word's mail merge feature, refer to your Microsoft documentation.

8.1.3 Accessing Data From Excel 2000 and 2003

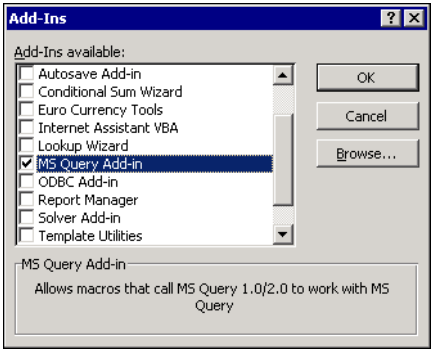
The following procedure describes how to access your Vision data from Microsoft Excel 2000, 2003 and later (these are the spreadsheet components of Microsoft Office). If you have a different version of Excel, your steps may be slightly different.

1. Start Excel. Excel automatically opens a new spreadsheet.

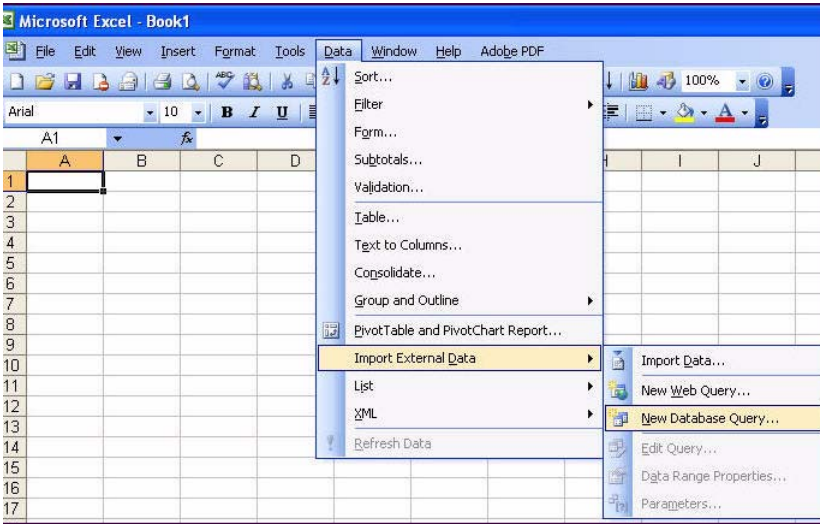
2. Excel 2000 users select **Add-Ins** from the Tools menu. Excel 2003 users go to step 4. If your Microsoft Office menus are set to "Show recently used commands first," you may need to click the expansion arrows at the bottom of the Tools menu to see the Add-Ins option.



3. In the Add-Ins dialog box, select the **MS Query Add-In** check box, and click **OK**. If this add-in does not appear on your list, reinstall Excel, being sure to select "MS Query" from the list of functions to install.



4. From the Data menu, select **Import External Data** or **Get External Data,** and then **New Database Query** from the submenu.

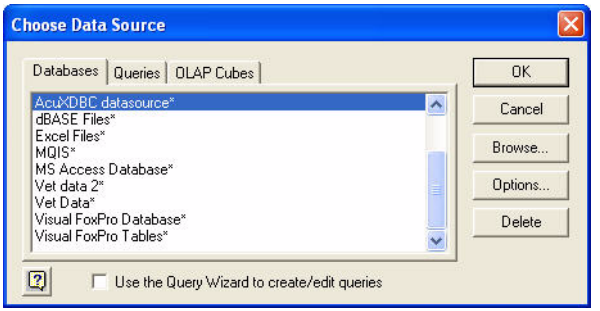


This starts Microsoft Query and opens the Choose Data Source dialog box.

If you want to use Microsoft Query "Query Wizard" to create your SQL query, leave the "Use Query Wizard" check box selected. Refer to Microsoft Query's online documentation for instructions on using the Query Wizard.

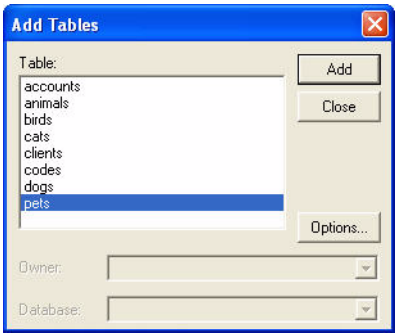
For this exercise, deselect the Query Wizard check box.

5. Select the DSN that corresponds to the data source you want to access and click **OK**. (Refer to **section 5.8, "Setting Up Data Source Names (DSNs) on Client,"** for instructions on creating DSNs.)



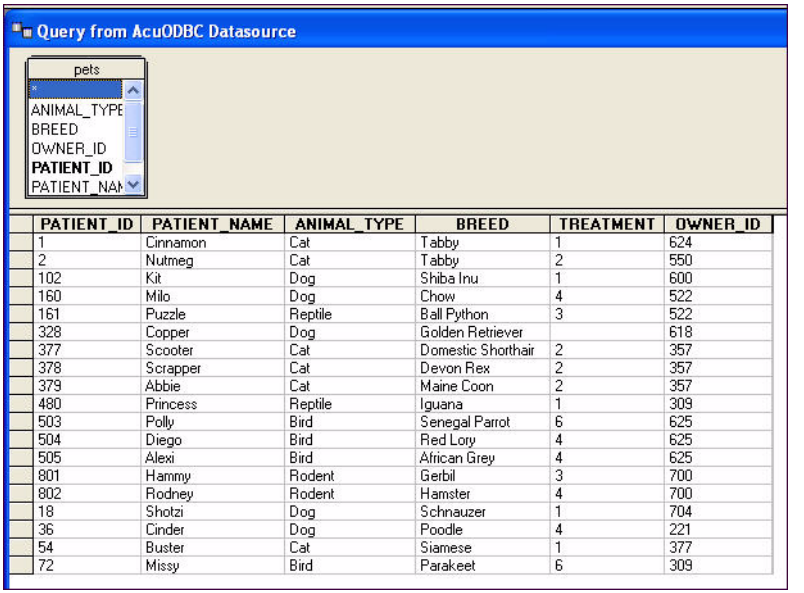
6. The AcuXDBC Login dialog box appears if you or an administrator established database security methods during AcuXDBC setup. Enter a valid User ID and Password, and click **OK**. If no security methods have been established, a User ID and Password are not required, and the login dialog box will not appear.

7. From the Add Tables dialog box, select the table or tables that you want to add (for example, "pets"), and click **Add**. Close the box when you're done.



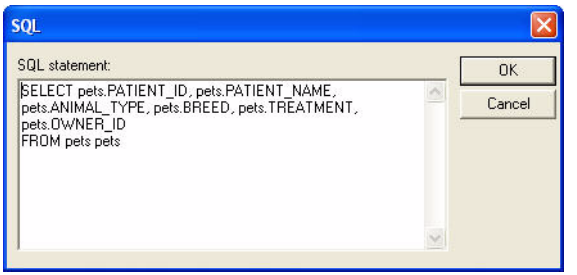
For each table you selected, a list box is displayed on the Microsoft Query screen. This list box lists all of the columns in the associated table.

8. Double-click each column that you want to read into your Excel spreadsheet, and that column is displayed on the screen. If you want to add all columns, double-click the "*" at the top of the list box.



PATIENT_ID	PATIENT_NAME	ANIMAL_TYPE	BREED	TREATMENT	OWNER_ID
1	Cinnamon	Cat	Tabby	1	624
2	Nutmeg	Cat	Tabby	2	550
102	Kit	Dog	Shiba Inu	1	600
160	Milo	Dog	Chow	4	522
161	Puzzle	Reptile	Ball Python	3	522
328	Copper	Dog	Golden Retriever		618
377	Scooter	Cat	Domestic Shorthair	2	357
378	Scrapper	Cat	Devon Rex	2	357
379	Abbie	Cat	Maine Coon	2	357
480	Princess	Reptile	Iguana	1	309
503	Polly	Bird	Senegal Parrot	6	625
504	Diego	Bird	Red Lory	4	625
505	Alexi	Bird	African Grey	4	625
801	Hammy	Rodent	Gerbil	3	700
802	Rodney	Rodent	Hamster	4	700
18	Shotzi	Dog	Schnauzer	1	704
36	Cinder	Dog	Poodle	4	221
54	Buster	Cat	Siamese	1	377
72	Missy	Bird	Parakeet	6	309

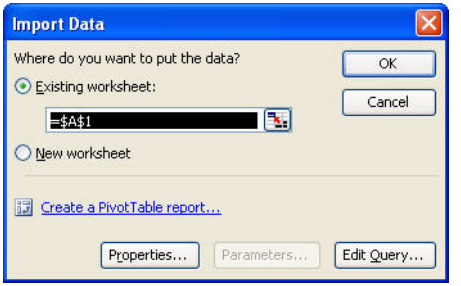
If you prefer, you can select columns or further refine your query by clicking the SQL button and entering your own SQL statements such as the one shown below.



Click **OK** when you're done. (For a description of the SQL commands supported by AcuXDBC, refer to **section 7.5, "Detailed SQL Support Descriptions."**)

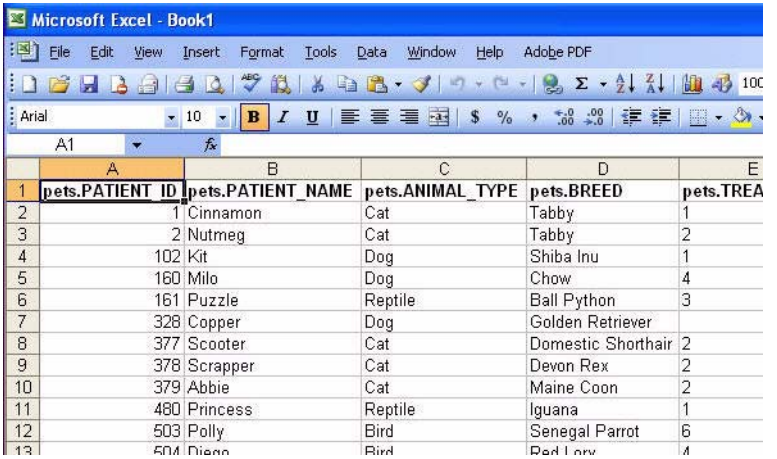
9. Select **Return Data to Microsoft Excel** from the Microsoft Query File menu.

10. The Import Data dialog box appears. Select where to put the data, and then click **OK**. If you want to select further options, click **Properties**.



11. If no database security methods have been established, the AcuXDBC Login dialog box will not appear. If database security methods have been established and you previously supplied a correct user ID and password, this step will not be necessary.

You are returned to Excel. The data you selected is displayed in the current Excel spreadsheet.



	A	B	C	D	E
1	pets.PATIENT_ID	pets.PATIENT_NAME	pets.ANIMAL_TYPE	pets.BREED	pets.TREA
2	1	Cinnamon	Cat	Tabby	1
3	2	Nutmeg	Cat	Tabby	2
4	102	Kit	Dog	Shiba Inu	1
5	160	Milo	Dog	Chow	4
6	161	Puzzle	Reptile	Ball Python	3
7	328	Copper	Dog	Golden Retriever	
8	377	Scooter	Cat	Domestic Shorthair	2
9	378	Scrapper	Cat	Devon Rex	2
10	379	Abbie	Cat	Maine Coon	2
11	480	Princess	Reptile	Iguana	1
12	503	Polly	Bird	Senegal Parrot	6
13	504	Diego	Bird	Red Lory	4

8.1.4 Accessing Data From Access 2000 and 2003

The following procedure describes how to access your Vision data from Microsoft Access 2000 and 2003, the database components of Microsoft Office. If you have a different version of Access, your steps may be slightly different.

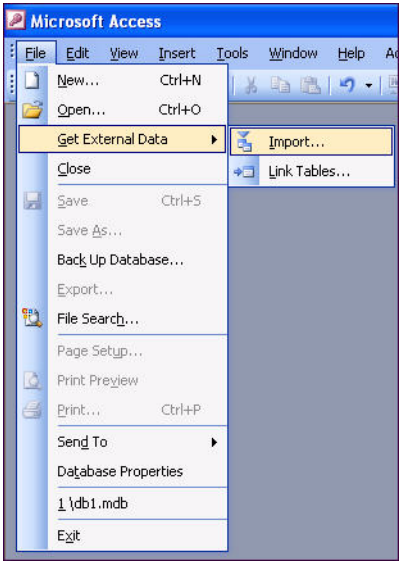
Note: When you are accessing relative files, you do not need to specify a unique key for Access. When Access displays the dialog box requesting that you enter a unique key, just click **OK** without selecting any field.

If you are using the Multi-company feature and are linking tables, you must relink every time you start Access and use this data source. See Chapter 5, **section 5.5.2** for more information on this feature.

---

1. Start Access.
2. Open an existing database or create a new one. If you create a new database, name and save it.

3. Select **Get External Data** from the File menu then select **Import** or **Link Tables**.



Here's a brief comparison of working with imported or linked data. For more information, refer to your Microsoft Access documentation.

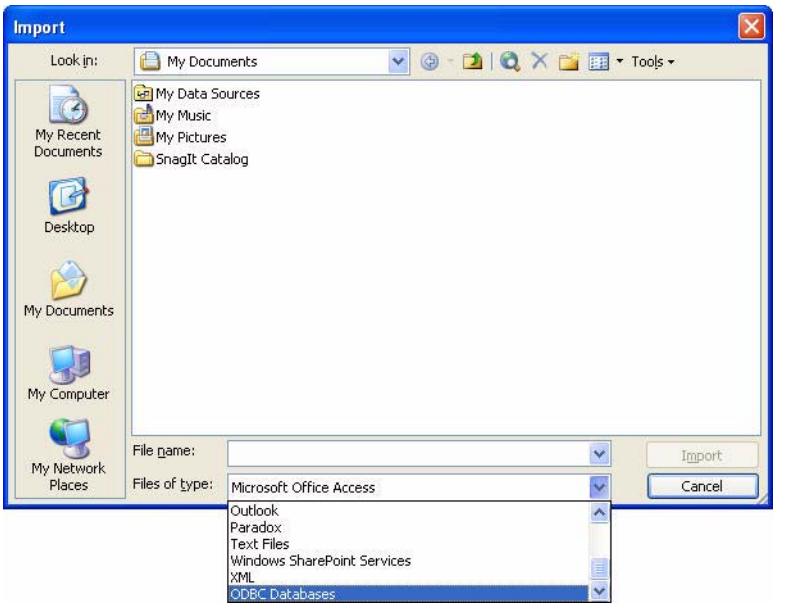
Imported Tables	**Linked Tables**
Data is local. The entire contents of an imported table is read into a copy on local storage.	Data is remote. Only rows requested from the linked table are read into local storage. Note that you can link to a local table, rather than creating a second copy by importing it.
Importing can take several seconds, or minutes, depending on the size of the database.	Creating a link is almost instantaneous, depending on the network connection.
Space is required for local copies of imported tables.	The link requires very little local storage.

Imported Tables	Linked Tables
Data is static. If there are changes to the database, they are not reflected in local data. If the database is subject to frequent changes, linking can be preferable.	Data is dynamic. You have access to any changes.
If the network connection is broken, users still have access to data.	If network connection is broken, users have no access to data until connection is restored.
Users cannot write to the Vision source.	Users can write to the Vision source *if* the DSN is read/write and the user has INSERT and/or UPDATE permissions on the object.

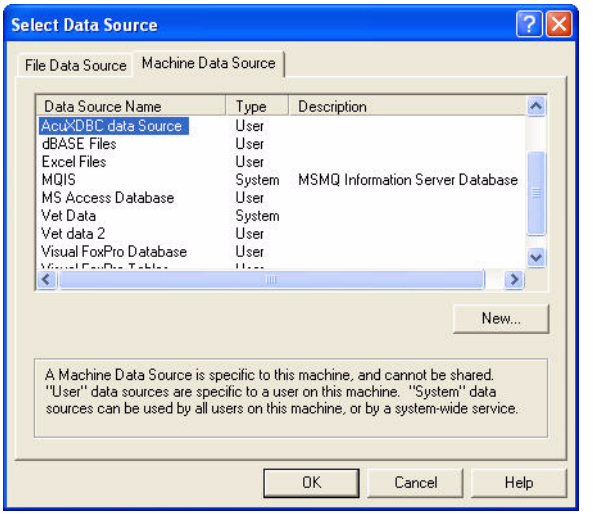
As a guideline, re-linking or re-importing is required when the table structure (such as the number of columns, or their names, sizes, and types) changes.

Note that in the examples shown in this section, the data is imported, not linked.

4. In the Import dialog box, select **ODBC Databases** from the "Files of type" list box.

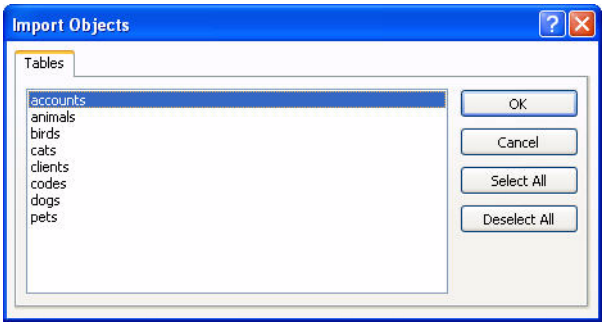


5. In the Select Data Source dialog box, select the DSN that you established, and click **OK**. If you created System or User DSNs, they will appear under the Machine Data Source tab. (Click **New** to create a new DSN. Refer to **section 5.8, "Setting Up Data Source Names (DSNs) on Client,"** for instructions.)



6. The AcuXDBC Login dialog box appears if you or an administrator established database security methods during AcuXDBC setup. Enter a valid User ID and Password, and click **OK**. If no security methods have been established, a User ID and Password are not required, and the login dialog box will not appear.

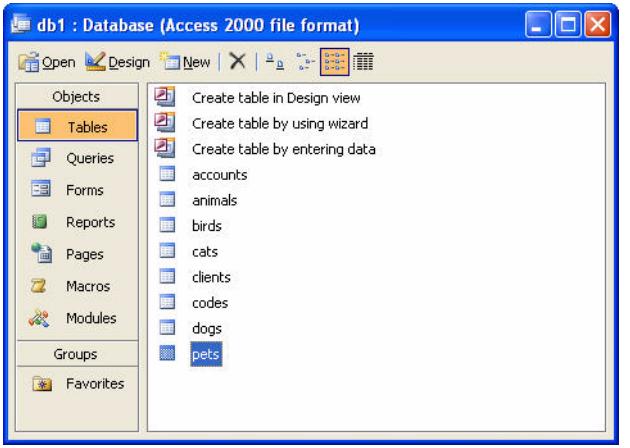
7. The Import Objects dialog box opens. Select the table or tables that you want to read into Access, and click **OK**.



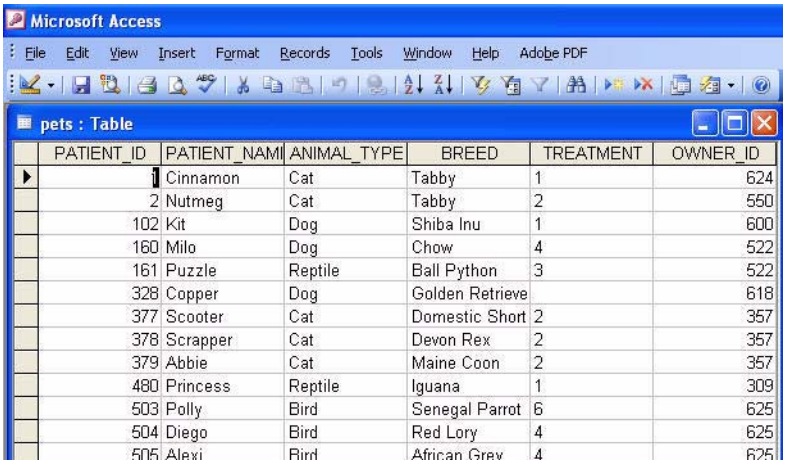
   Click **Select All** to add all of the listed tables to your Access database.

8. Double-click the table icon to open the table in Access.

If you have imported the data, the table name or names you selected appear on the Access screen next to an icon of a table. You will see a different icon next to the table names if you have linked the data.



9. You may now perform Access operations on this data.



PATIENT_ID	PATIENT_NAMI	ANIMAL_TYPE	BREED	TREATMENT	OWNER_ID
1	Cinnamon	Cat	Tabby	1	624
2	Nutmeg	Cat	Tabby	2	550
102	Kit	Dog	Shiba Inu	1	600
160	Milo	Dog	Chow	4	522
161	Puzzle	Reptile	Ball Python	3	522
328	Copper	Dog	Golden Retrieve		618
377	Scooter	Cat	Domestic Short	2	357
378	Scrapper	Cat	Devon Rex	2	357
379	Abbie	Cat	Maine Coon	2	357
480	Princess	Reptile	Iguana	1	309
503	Polly	Bird	Senegal Parrot	6	625
504	Diego	Bird	Red Lory	4	625
505	Alexi	Bird	African Grey	4	625

8.2 Working with Java Applications

The enterprise edition of AcuXDBC includes the “vortex.jar” file that enables a Java client application to connect to your Vision database. The general Java-specific steps and requirements involved in connecting to a Vision database include the following:

1. A Java source file that uses the “DriverManager.getConnection” method to initiate a Vision database connection.
2. String URL within the Java source file that includes the appropriate connection snytax.
3. Java source file compiled using Java 1.5 or later.
4. The path of “vortex.jar” specified in the CLASSPATH of the Java application you wish to run.

Source File and String URL Syntax Sample

Included with the enterprise edition of AcuXDBC is the “sample.java” file, which is located in the “samples” directory of AcuXDBC. This file demonstrates the use of the “DriverManager.GetConnection” method along with the String URL syntax to connect to a remote system catalog.

Run the demo program, “demo.bat” on Windows and “demo.sh” on UNIX, to create a sample veterinarian database. The “sample.java” program will issue SQL commands to access the sample veterinarian database tables.

Note: No spaces are allowed between the String URL connection parameters. If access to the system catalog is local, then “*@port no*”, “*machine name*”, and “*ip-address*” is not required.

---

sample.java

```
import java.sql.*;

class sample {
    public static void main (String argv[]) {
        try {

    /*  Note:  The format is

jdbc:vortex:userid//password/driver:configfile@port:hostname!databaseid

        For example
        (using defaults for driver, configfile, port, and databaseid):

jdbc:vortex:system//manager/xvision:acuxdbc.cfg@20222:myserver!acuxdbc04

    */

            String connectionURL =
                "jdbc:vortex://system/manager/xvision:" +
                "acuxdbc.cfg" +
                "@20222:myserver.acucorp.com" +
                "!acuxdbc04";

            String sqlStatement =
                "  SELECT                          \n" +
                "      patient_id,                 \n" +
                "      patient_name,               \n" +
                "      animal_type                 \n" +
                "    FROM pets                     \n" +
                "    WHERE                         \n" +
                "      LCASE(animal_type) = 'dog'  \n" +
                "    ORDER BY patient_name";

            System.out.println("\nConnection URL: \n" + connectionURL);
            System.out.println("\nSQL Statement:  \n" + sqlStatement);

            Class.forName("vortex.sql.vortexDriver");
            Connection conn =
                DriverManager.getConnection(connectionURL);

            Statement stmt =
                conn.createStatement(   ResultSet.TYPE_FORWARD_ONLY,
                                        ResultSet.CONCUR_UPDATABLE );
            ResultSet data;
            data = stmt.executeQuery(sqlStatement);

            System.out.print("\nRESULTS:\n\n");
            System.out.print(
```

```
                "Patient ID   Patient Name         Animal Type\n" +
                "============ ==================== ====================\n");
            while (data.next()) {
                System.out.printf( "%-12s %-20s %-20s\n",
                                   data.getString(1),
                                   data.getString(2),
                                   data.getString(3));
            }
        }
        catch(Exception e) {
            e.printStackTrace();
        }
    }
}
```

Compile the source file

Compile the source file as follows:

```
<java home directory>/javac  sample.java
```

Run the application with vortex.jar

To run your application, the path of "vortex.jar" must appear in the CLASSPATH of the Java application. For example:

```
export CLASSPATH=<directory path>/vortex.jar:.
<java home directory>/java sample
```

Note: You must have read and execute permission on vortex.jar in order to run your program. If you get the error message, "vortex.sl. vortex driver", check the permissions that you have on vortex.jar.

---

9 Troubleshooting

Key Topics

9.1 Introduction

When installing, configuring, or using AcuXDBC components, you may encounter error messages along the way. This chapter can help you find the cause of the error by explaining what the error message means. If you encounter an error message, locate it in this chapter and see if the information provided can help you uncover the underlying cause of the error. If you still cannot determine the cause, use the logging features of AcuXDBC to trace AcuXDBC and Vision activities. Refer to Chapter 4, **section 4.2.1** for details on using the logging features.

Error messages may originate from the following sources:

- AcuXDBC Client
- AcuXDBC Server
- AcuXDBC SQL Processor
- Vision File System
- Client Applications

9.2 AcuXDBC Client Error Messages

Messages are listed in alphabetical order.

Message	Description
BADCONV	Cause: The requested data conversion failed. Action: Check that the requested data is of the appropriate type. For example, this error occurs when a character column is fetched into an integer and the character data are not all digits.
BADINI	Cause: The ".ini" file is missing or its contents are invalid. Action: Check that the file exists and that its settings are correct.

Message	Description
CANCEL	Cause: Operation was cancelled by the driver manager. Action: Informational message; no action necessary.
CANFREE	Cause: No processing was being done on the statement, so the call was treated as a call to SQLFreeStmt with the SQL_CLOSE option. Function returns SQL_SUCCESS_WITH_INFO. Action: Informational message; no action is necessary.
CURDUP	Cause: The cursor name is already in use. Action: Specify a different name.
DATATRUNC	Cause: Data has been truncated. Either the data specified is too long or supplied output buffers are too small. Action: Modify the size of the output buffers.
DIALOG	Cause: The connect dialog failed. Action: Notify your system administrator.
FUNCSEQ	Cause: The sequence of functions called is invalid. Action: Make sure that you follow the sequence specified by the ODBC documentation.
GENONLY	This version only supports GENESIS. Cause: This version of AcuXDBC only supports GENESIS. Action: Make sure you are attempting to connect to a Vision database.
INVARG	Cause: Invalid arguments specified. Action: Consult the ODBC documentation for the correct syntax.
INVAUTH	Cause: User not authorized to connect to specified data source. Action: Check your userid and password or contact your system administrator.
INVBUFLEN	Cause: The string or buffer length specified is invalid. Negative values such as SQL_NTS have special meaning, but not all negative values are valid. Action: Check the ODBC documentation for valid length specifiers.

Message	Description
INVCOLNUM	Cause: The specified column number is out of range. Action: Verify that the correct column number is being used.
INVCURNAM	Cause: The specified cursor name is invalid. Action: See the ODBC documentation for the maximum allowed length.
INVCURSTA	Cause: The state of the cursor (OPENed, CLOSEd, etc.) is not valid for the current operation. Action: Make sure you've executed the necessary steps before calling this function.
INVDATA	Cause: The data for the specified operation is invalid. Action: Verify that the data being used is correct.
MANYCONN	Cause: The concurrent connections limit has been exceeded. Action: Reduce the number of concurrent connections.
MANYSTMT	Cause: The number of allowable statements has been exceeded. Action: Either increase the limit in the Advanced tab of the Setup Dialog screen, or free any statements you are not using or do not need.
MISSENV	Cause: The environment variable (registry entry on Windows) GENESIS_HOME is missing. Action: Set the GENESIS_HOME environment variable/ registry entry.
MISSINDV	Cause: NULL data supplied without an indicator variable. If the data is NULL, an address of an indicator variable must be supplied. Action: Supply an address for the indicator variable.
NOCONN	Cause: No connection established. A connect must be performed before any other operations. Action: Connect to the data source before attempting any database operations.
NOCURNAM	Cause: Cursor name was never assigned. Action: Assign a cursor name.

Message	Description
NODRV	Cause: No AcuXDBC driver was specified in the connect. Action: Verify that the connect string specifies the AcuXDBC driver.
NODSN	Cause: The Data Source Name (DSN) was not specified. Action: Verify that the connect string specifies the DSN.
NOMEM	Cause: A memory allocation failed. This is a fatal error. Either there is no more heap memory available (rare) or the heaps have been corrupted. Action: Notify your system administrator immediately.
NOTCAP	Cause: AcuXDBC does not support the requested capability. Action: Modify your program to not request this capability.
NOTIMP	Cause: The feature has not been implemented yet. Action: Submit an enhancement request to Technical Support.
NOWHDL	No window handle available. Cause: No window handle available to open connect dialog. Action: Notify your system administrator.
OPTCHG	Cause: A value of an option has been changed. Action: Informational message. No action required.
PARMCNT	Cause: The number of parameters specified does not match the number of parameters required by the statement. Action: Modify your program to use the correct number of parameters for the statement.
PARMDTY	Cause: The data type specified for the parameter is unknown. Action: Consult the ODBC documentation for valid data types.
PARMNUM	Cause: The parameter number specified is out of range. Action: Verify that your program uses the correct parameter number.
TIDUSED	Cause: Another thread is currently using the statement. Action: Verify that your program does not use the same statement as is being used in another thread.

Message	Description
UNDESTYP	Cause: The fDescType for SQLColAttributes() is unknown. Action: Modify your program to use the correct descriptor.
UNFETTYP	Cause: Unknown fetch type. Currently only SQL_FETCH_NEXT is supported. Action: Modify your program to use only SQL_FETCH_NEXT.
UNINTYP	Cause: The Info Type is unsupported. Action: Consult the ODBC documentation for valid values.
UNOPT	Cause: The option is unknown. Action: Consult the ODBC documentation for valid options.
UNUNOPT	Cause: The Uniqueness option is unknown. Action: Consult the ODBC documentation for valid values.
UNXACOPR	Cause: The transaction operation is unknown. Action: Consult the ODBC documentation for valid values.

9.3 AcuXDBC Server Error Messages

All messages are listed in alphabetical order.

Message	Description
AUTHBAD	Cause: The authentication syntax is invalid. Action: Follow the host name with the userid/password.
AUTHFAIL	Cause: Authentication on *service* failed. You are not authorized to execute the requested host service. Action: Verify that the userid/password are correct or contact your system administrator.
AUTHREQ	Cause: The host you are connecting to requires additional authentication. Action: Follow the host name with the userid/password.

Message	Description
BADINI	Cause: The “net.ini” file is missing or is an invalid file. Action**:** Verify that “net.ini” exists and is valid.
CONFIG	Cause: This assertion error notifies you that the first call must always be a CONFIG. Action: Make sure a CONFIG call precedes any other call.
DLLENTRY	Cause: The loaded DLL or shared library does not contain the expected entry point. This error occur only on machines that support DLLs or shared libraries and usually signals that the wrong DLL has been loaded. Action: Verify that the AcuXDBC DLLs have not been overwritten by other DLLs or that there are not other DLLs in the path with the same name.
DLLLOAD	Cause: Could not load the specified DLL or shared library. The DLL is either missing or invalid. This error occurs only on machines that support DLLs. Action: Ensure that the DLL or shared library specification is correct (including spelling) and check that the correct DLL is installed.
DLLSAFE	Cause: Loaded DLL is not thread safe. Not all database drivers are thread safe. Action: To avoid this error, run the single-threaded daemon: “xdbcsrvr.exe”.
EVALCORR	Cause: The evaluation license ID has been corrupted. Action: Download the evaluation software from the Web site again or contact Micro Focus to purchase the software.
EVALEXP	Cause: The evaluation license has expired. The timeout period is 4 hours. Action: To continue the evaluation, restart the AcuXDBC daemon. If you want a copy of the software that doesn’t expire, contact Micro Focus to purchase the software.

Message	Description
EXECFAIL	Cause: Exec *name* failed on host *name*. The service (program) specified in the network connection string could not be started. This error occurs if the service could not be found, does not have the correct permissions, or is not listed as a valid service. Action: Check that the service exists, is listed as a valid service, and that you are connecting with the correct user name and password.
HOSTNOT FOUND	Cause: The host *name* you are trying to connect to is not found. Action: Make sure the spelling of the host is correct and that it really exists.
INVHOSTSYN	Cause: The host/service syntax is invalid. Action: Review the connect string documentation in Chapter 5, **section 5.3.1** of this manual..
INVVER	Cause: NET version mismatch (host: *number*, client: *number*). The version of AcuXDBC server is not the same as AcuXDBC client. Action: Make sure that both sides of the network connection are at the same version level.
KEEPALIVE	Cause: This assertion error indicates that the socket option KEEPALIVE failed or was not set. Action: Notify Micro Focus Technical Support or your system administrator.
LINGER	Cause: This assertion error indicates that the socket option LINGER failed or was not set. Action: Notify Micro Focus support or your system administrator.
NOINTR	Cause: Your program requested a cancel operation. The host you are trying to cancel cannot handle interrupts. Action: Modify your program so that it does not call the cancel operation.

Message	Description
NOMEM	Cause: A memory allocation failed. This is a fatal error. Either there is no more heap memory available (rare) or the heaps have been corrupted. Action: Notify your system administrator immediately.
SERVNOT FOUND	Cause: The service and/or protocol *name* cannot be found. Action: Ensure that xdbcsrvr is specified in "/etc/services". If it is not, you must add it, or explicitly specify the port number in the connect string.
SOCKET	Cause: Call to the socket() function failed. The operating system may have run out of file descriptors. Action: Notify your system administrator.
UNDBID	Cause: Unknown database ID specified in "net.ini". Action: Use a valid database ID (from 0 to 3) in "net.ini".

9.4 AcuXDBC SQL Processing Error Messages

Following are error messages related to issuing SQL statements:

Message	Description
Cannot open file '*filename*'	Cause: The specified file cannot be opened. The path may be wrong, it may not exist, or the permissions are not set correctly. Action: Check the spelling and verify the location of filename.
Catalog table '*name*' corrupted or out of date	Cause: The specified catalog table cannot be read. It has either been modified directly or it is from a different version of AcuXDBC. Action: Rebuild the catalog with the correct catalog utility.
Character array too big (max: *number*)	Cause: The SQL statement contains a character array that is too big. Action: Correct the statement.

Message	Description
Column *name* already defined	Cause: The SQL CREATE TABLE/VIEW statement has duplicate column names. Action: Correct the statement.
Column *name* undefined	Cause: The SQL statement refers to a column that is not defined in the catalog. Action: Correct the statement or define the column/table in the catalog.
Create view column count mismatch (create: *number*, select *number*)	Cause: The SQL CREATE VIEW statement's column list does not match the number of columns in the SELECT statement's select list. Action: Correct the statement.
Data truncation (max: *number*)	Cause: Data was truncated. Action: Notify Micro Focus Technical Support.
End of buffer reached	Cause: The SQL statement ended prematurely. Action: Check the syntax for the command you are using.
Ending quote missing	Cause: The SQL statement is missing an ending quote. Action: Add the missing quote.
Function *name* not implemented yet	Cause: Not implemented yet. Action: Contact Micro Focus with an enhancement request.
GENESIS_HOME environment variable not found	Cause: The GENESIS_HOME environment variable (registry entry on Windows) is not set. Action: Set the GENESIS_HOME environment variable, following instructions in the installation procedure.
Identifier too long	Cause: The SQL statement contains an identifier that is too long. Identifiers are limited to 30 characters. Action: Rename the identifier.

Message	Description
If any numeric operand is NULL then only '==' and '!=' are valid	Cause: The SQL statement's WHERE clause is using an invalid operator with a NULL value. Action: Correct the statement.
Illegal character	Cause: The SQL statement contains an illegal character at the given position. Action: Check the statement for legality.
Illegal number of parameters for built-in function	Cause: The SQL statement has the wrong number of parameters for the built-in function. Action: Correct the statement.
Invalid parameter	Cause: The AcuXDBC COMMAND sent an invalid parameter for the specified command. Action: Verify that the parameters you use are all valid.
Invalid password	Cause: The connection password is incorrect. Action: Ensure that you are using a valid password. Contact your DBA.
Invalid predicate result (NULL or invalid datatype)	Cause: The SQL statement's WHERE clause returned a NULL or invalid datatype result. Action: Correct the statement.
No data source specified	Cause: The connect string does not contain a data source specification. Action: Make sure you supplied a valid data source name. If you didn't create a data source name, refer to Chapter 5, **section 5.8** of this manual for information on creating a data source name.

Message	Description
No locks available on UPDATE	Cause: This sometimes happens when you link a file into Microsoft Access and perform an update that affects a large number of rows. Action: If the number of records affected by an update is less than 8191, set the configuration options “MAX_LOCKS” and/or “LOCKS_PER_FILE” to a value slightly higher then the number of records. The maximum setting is 8191. If the number of records is higher than 8191, set the TRANSACTIONS variable to “FALSE”. This causes the interface to ignore the transaction command and update the records individually.
Non aggregates require a GROUP BY expression	Cause: The SQL SELECT statement contains aggregate and non-aggregate select list items and this requires a GROUP BY expression. Action: Edit the statement and use GROUP BY or change the statement’s structure.
Notify tech support.	Cause: Internal error. Action: Notify Micro Focus Technical Support.
NULL not allowed for column	Cause: The SQL INSERT/UPDATE statement is using a NULL value for a not-NULL column. Action: Correct the statement.
Number of columns does not match number of values	Cause: The SQL INSERT statement’s values do not match the number of columns defined for the table or listed in the column list. Action: Correct the statement.
Operation requires *named* authorization	Cause: The SQL statement requires the specified authorization. Action: Ensure that you have authority to issue the statement. Contact your DBA.
Sort column *name* out of range (1 - *number*)	Cause: The SQL SELECT statement’s ORDER BY clause refers to a column number that is out of range. Action: Correct the statement.

Message	Description
String too long	Cause: The SQL statement contains a string constant that is too long. Action: Use a bind variable.
Sub-query must return a single column	Cause: The SQL statement contains a sub-query whose select list has more than one column. Action: Correct the statement.
Table *name* undefined	Cause: The SQL statement refers to a table that is not defined in the catalog. Action: Correct the statement to refer to a defined table or define the table in the catalog.
to_char/date/ number's format mask must be a constant string	Cause: The SQL statement uses a data conversion function with a non-constant format mask string. Action: Correct the statement.
Too many columns *number* (max: *number*)	Cause: The SQL statement refers to a table with too many columns. Action: Correct the statement.
Too many columns specified	Cause: The SQL statement has too many columns defined. Action: Correct the statement.
Too many cursors opened	Cause: Your application has too many concurrently opened cursors. Action: Explicitly close cursors when they are no longer needed. If this action does not solve the problem, notify Micro Focus Technical Support.
Too many sort columns (max: number)	Cause: The SQL SELECT statement has too many columns in the ORDER BY clause. Action: Reduce the number of columns in the ORDER BY clause.
Too many sub-queries at level *number* (max: *number*)	Cause: The SQL statement contains too many sub-queries. Action: Correct the statement.

Message	Description
Too many tables in SELECT (max: *number*)	Cause: The SQL SELECT statement contains too many tables. Action: Correct the statement.
Unknown error code	Cause: Internal error. Action: Notify Micro Focus Technical Support.
Unknown node (type: *name*)	Cause: Internal error. Action: Notify Micro Focus Technical Support.
yacc: *msg*	Cause: The SQL statement cannot be parsed correctly. Action: Check for invalid keywords.

9.5 Vision File System Error Messages

Vision error codes have two numbers. The first number corresponds to a general "SQLSTATE" error. You can find information on SQLSTATE errors in general ODBC manuals, like the *ODBC Programmer's Reference*. The second number in the native Vison error code. It lists the native error associated with AcuXDBC operation.

For example, in the following error code:

```
S1003   102
```

"S1003" is the SQLSTATE ODBC error number, and "102" is the native Vision error number.

Error Code	Description
100	A duplicate key was detected where duplicates are not allowed.
102	This indicates that a routine was called with an illegal parameter. For example, specifying a key number that is larger than the number of keys in the file would result in this error.

Error Code	Description
104	An attempt was made to open more files than the system allows open at once.
105	The index file is corrupt or the physical disk file does not match the type of file being opened. If the index file is corrupt, it should be reconstructed using the appropriate host system utility.
107	The requested record is locked by another process or (for some systems) by another file handle used by this process.
111	The requested record was not found.
112	The current key of reference was in the "undefined" state when the "i_next" or "i_previous" routines were called. There is no current record from which to read forwards or backwards.
113	The file is locked against the current open mode.
116	The system ran out of dynamic memory.
117	An operation was requested that the current open mode doesn't allow. For example, attempting to add a record to a file that is open for input-only results in this error.
126	The requested operation is not supported by this host system.
127	The disk became full while adding a new record.
128	The size of the new record doesn't match the type of file being opened.
129	The system ran out of lock-table entries.
130	The file is missing.
131	Invalid permission for the operation.

9.6 Application Errors

You may occasionally receive an application-specific error message when trying to access data through AcuXDBC. For instance, when using Microsoft Query, you may receive an error like:

```
MSQry error:  This program has performed an illegal operation
and will be shut down.  If the problem persists, contact the
program vendor.
```

If you receive an error like this, try removing and re-adding your DSN in the ODBC Data Source Administrator. Occasionally, the application may have difficulty recognizing the DSN, particularly if you have upgraded from a previous version of AcuODBC. Recreating the DSN should solve the problem.

For specific information regarding other application-specific errors, refer to your application's user documentation or on-line help.

A Compatibility Guide

Key Topics

A.1 Migrating from AcuODBC to AcuXDBC

Due to the increased functionality of AcuXDBC, the procedures for setting up and configuring AcuXDBC differ from AcuODBC. The following chart compares set up methods and highlights key differences between AcuXDBC versus AcuODBC. It also describes where to look for detailed information pertaining to the stated difference.

AcuXDBC	AcuODBC	Differences
Install software.	Install software.	AcuXDBC uses Microsoft Installer. Refer to Chapter 5, **section 5.2** for AcuXDBC installation instructions.
Create XFDs.	Create XFDs.	No major difference in the method used for creating your XFDs. If you already have XFDs from previous versions of AcuODBC or other products, these can be reused.
Set configuration variables in the configuration file.	Set configuration variables in AcuODBC DSN Setup tabs.	To simplify client setup and enable universal configuration, AcuXDBC is configured via a configuration file rather than on each client machine during DSN setup. Many of the options previously configured on the AcuODBC screens are now configured variables that you set in "acuxdbc.cfg." This method allows you to use the same configuration file for the entire network by just pointing to the configuration file from each client's DSN setup screen. Refer to **Section A.2** of this appendix for information on variable locations.

AcuXDBC	AcuODBC	Differences
Create a system catalog and load it with information from the XFD.	Not required for AcuODBC.	In order to provide greater relational database functionality, AcuXDBC refers to the system catalog to construct and display your Vision tables. After this loading phase, your XFDs are no longer needed by AcuXDBC. Refer to Chapter 6, **section 6.2**.for a detailed descripion of the system catalog and the role of XFDs.
Set system security via SQL GRANT statement.	Set system security via AcuAccess file.	The AcuAccess file and manager utility are no longer used with AcuXDBC. Instead, the SQL GRANT command and its variants are used to manage database security. Refer to Chapter 5, **section 5.4**.
Set up DSNs.	Set up DSNs.	If you are migrating from AcuODBC, you need to create new DSNs. You accomplish this in generally the same manner in AcuXDBC; however, many of the variables that once appeared on the AcuODBC configuration screen now appear in the AcuXDBC configuration file (acuxdbc.cfg). This process allows for universal settings for DSNs and simplifies the process of setting up multiple clients in a network setting. For large network installs, you can also create a .reg file and download the DSN information to your clients. Refer to **Section A.2** of this appendix for details on DSN-related variable locations.

A.2 AcuODBC Configuration Screen Changes

Many of the configuration variables previously located in the AcuODBC Configuration screen (now called the “AcuXDBC Setup” screen) are now located in one of two configuration files included with AcuXDBC. By default, they are known as “acuxdbc.cfg” and “net.ini”. “acuxdbc.cfg” is required for all installations. Network installations using AcuXDBC Server require both “acuxdbc.cfg” and “net.ini”.

The following charts list each configuration option that appeared on the tabs of the AcuODBC configuration screen and then lists its corresponding variable name and location in AcuXDBC. For details on actual AcuXDBC variable settings, refer to **Chapter 4, section 4.2, “AcuXDBC Configuration”**.

A.2.1 General Tab

AcuODBC variable	**Corresponding AcuXDBC variable**	**AcuXDBC Location/ Comments**
Data Source Name	Data Source Name	AcuXDBC DSN Setup screen, General tab.
XFD Directory	DICTSOURCE	acuxdbc.cfg. AcuXDBC refers to the system catalog (specified by DICTSOURCE), instead of your XFDs, for information it uses to form table views of your data. After the initial loading of your XFD, the XFD file itself is no longer needed by AcuXDBC.
Data Directory	FILE_PREFIX	acuxdbc.cfg.
Data File Extension	FILE_SUFFIX	acuxdbc.cfg.

A.2.2 Advanced Tab

AcuODBC variable	Corresponding AcuXDBC variable	AcuXDBC Location/ Comments
File Case	FILE_CASE	acuxdbc.cfg
Compiler Option	None	The data storage compiler options (“-Dci”, “-Dcm”, “-Dcn” ...) are set using the **xdbcutil** “-s” command when loading your system catalog. Refer to Chapter 5, **section 5.3.1** for details on using this command.
Character Set	None	OEM support is not provided in AcuXDBC.
Open Tables	None	Not applicable in AcuXDBC.
Character Types	None	Not applicable in AcuXDBC.
Invalid Numeric Data	INVALID_NUMERIC_ DATA	acuxdbc.cfg
Division by Zero	DIVISION_BY_ZERO	acuxdbc.cfg
Signed Index	None	Not applicable in AcuXDBC.
Read Only	READ_ONLY	acuxdbc.cfg
GUI Dialog	None	Not applicable in AcuXDBC.
Table Name Cache File	None	Not applicable in AcuXDBC.
Use DOUBLE for Numeric Key Data	None	Not applicable in AcuXDBC.

AcuODBC variable	Corresponding AcuXDBC variable	AcuXDBC Location/ Comments
Null Processing	NULL_ALPHA_READ NULL_ALPHA_WRITE NULL_NUMERIC_READ NULL_NUMERIC_WRITE	acuxdbc.cfg
Julian Base Date	JULIAN_BASE_DATE	acuxdbc.cfg
Reset Configuration	None	No longer required since configuration occurs in a text file.

A.2.3 Vision Tab

AcuODBC variable	Corresponding AcuXDBC variable	AcuXDBC Location/ Comments
Max Files	MAX_FILES	acuxdbc.cfg
Max Locks	MAX_LOCKS	acuxdbc.cfg
Locks Per File	LOCKS_PER_FILE	acuxdbc.cfg
Buffers	V_BUFFERS	acuxdbc.cfg
Enable Transaction Processing Support	TRANSACTION_ PROCESSING	acuxdbc.cfg
Enable Logging	LOGGING	acuxdbc.cfg
Encrypt Log File	LOG_ENCRYPT	acuxdbc.cfg
Log to Device	LOG_DEVICE	acuxdbc.cfg
Log File	LOG_FILE	acuxdbc.cfg

AcuODBC variable	Corresponding AcuXDBC variable	AcuXDBC Location/ Comments
Temp File Directory	TEMP_DIR	acuxdbc.cfg
Log Buffer Size	LOG_BUFFER_SIZE	acuxdbc.cfg

A.2.4 Tracing Tab

AcuODBC variable	Corresponding AcuXDBC variable	AcuXDBC Location/ Comments
Vision Trace Level	VISION_LOGGING_ LEVEL	acuxdbc.cfg
Vision Trace File	VISION_LOGGING_FILE	acuxdbc.cfg
ODBC Driver Trace Level	Full Logging SQL Logging Include Time Unique Filenames	AcuXDBC Setup screen, Logging tab, Client Logging frame.
ODBC Driver Trace File	Log Filename	AcuXDBC Setup screen, Logging tab, Client Logging frame.
SQL Tracing Level	DEBUG_LOGLEVEL	acuxdbc.cfg
SQL Trace File	DEBUG_LOGFILE	acuxdbc.cfg

A.2.5 Server Tab

AcuODBC variable	Corresponding AcuXDBC variable	AcuXDBC Location/ Comments
Local SQL Processing	Network Driver	AcuXDBC Setup screen, General tab. In AcuXDBC, deselecting "Network Driver" has the same effect as selecting "Local SQL Processing" in AcuODBC.
Remote SQL Processing	Network Driver	AcuXDBC Setup screen, General tab. In AcuXDBC, selecting "Network Driver" has the same effect as selecting "Remote SQL Processing" in AcuODBC.
Hostname	Hostname	AcuXDBC Setup screen, General tab.
IP Address	Hostname	AcuXDBC Setup screen, General tab. Use Hostname to set either the hostname or IP address.
Port Number	Port	AcuXDBC Setup screen, General tab.
NT Security	None	AcuXDBC has multi-level security options that are set using SQL standards. Refer to Chapter 5, **section 5.4** for information on setting database security.

A.2.6 File Alias Tab

AcuODBC variable	**Corresponding AcuXDBC variable**	**AcuXDBC Location/ Comments**
Table name XFD Data File Write Permission	None.	With AcuXDBC, you define file aliases when adding XFDs to your system catalog with **xdbcutil**. Refer to Chapter 5, **section 5.5.1**.

A.2.7 Multi-company Tab

AcuODBC variable	**Corresponding AcuXDBC variable**	**AcuXDBC Location/ Comments**
Company Data Directory Write Permission	None	In AcuXDBC, Multi-company setup is managed via wildcards or assigning table ownership using SQL commands**.** Refer to Chapter 5,**section 5.5.2** for details on two methods available for setting up multi-company data sets.

A.2.8 AcuServer Tab

AcuODBC variable	Corresponding AcuXDBC variable	AcuXDBC Location/ Comments
AcuServer Password	AcuServer Password User name/Password	acuxdbc.cfg AcuXDBC Setup screen, General tab. AcuXDBC uses a security method based on SQL standards, which involves establishing user IDs and passwords. Refer to Chapter 5, **section 5.4**.
NT Security	None	No longer applies. This method is done automatically.
International Character Handling Default Map File	DEFAULT_MAP_FILE	acuxdbc.cfg
AcuServer Port	AcuServer Port	acuxdbc.cfg

Index

Symbols

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

X