Sybase Adaptive Server Anywhere for Linux HOWTO

Aylwin Lo

Tom Slee

Sybase Inc.

Tom.Slee@sybase.com

Revision History

Revision 1.0	2001−04−26	Revised by: al
First public release.		

This HOWTO guides you through the installation of SQL Anywhere Studio 7.0.2 for Linux and the basic operation and administration of Adaptive Server Anywhere databases.

Table of Contents

1. Introduction......1
1.1. New versions of this document......1
1.2. Content and Audience......1
1.3. Adaptive Server Anywhere features......1
1.4. Quirks......2
1.4.1. Alt and Function keys......2
1.5. What's a Relational Database?......2
1.5.1. Definition......2
1.5.2. Example......2
1.5.3. Primary and Foreign Keys......4

2. Requirements......5
2.1. System requirements......5
2.2. Supported distributions......5

3. Installation......6
3.1. Process......6
3.2. Distribution−specific considerations (for TurboLinux and Caldera)......6
3.3. Setting the Environment Variables......6
3.4. Where did it get installed?......7

4. Creating, Running and Connecting to Databases......8
4.1. Creating a database......8
4.1.1. Creating a database from the command prompt......8
4.1.2. Creating a database from Sybase Central......9
4.2. Running a database server and starting databases......9
4.2.1. Running the server as a daemon......10
4.3. Stopping the database server......10
4.4. Stopping databases......11
4.5. Connecting to a database......12
4.5.1. Connection strings......12
4.5.2. Connecting from Interactive SQL......13
4.5.3. Connecting via ODBC......13

5. Backing up and Restoring a Database......16
5.1. Creating a Backup of the Database......16
5.1.1. Full vs. Incremental Backups......16
5.1.2. Online vs. Offline Backups......16
5.1.3. Server−side vs. Client−side Backups......16
5.1.4. How to make a backup......17
5.2. Validating the database and its backup......18
5.3. Recovering the database......18

6. Managing a Database......19
6.1. Tables......19
6.1.1. Creating a Table......19
6.1.2. Making Alterations to Tables......20
6.2. Users, permissions, and authorities......20

Table of Contents

6.2.1. User IDs....21
6.2.2. Permissions....21
6.2.3. Authorities....22
6.2.4. Removing Users and Revoking Permissions....23
6.2.5. Changing Passwords....23
6.3. Making the database more secure....23
6.3.1. Increasing password security....24
6.3.2. Views, procedures, and triggers....24
6.3.3. Encrypting client/server communications....25

7. Where to get more information....26

8. Legalities and Acknowledgements....27
8.1. Copyright and Licenses....27
8.2. Names and Contacts....27
8.3. Acknowledgement....27

1. Introduction

This HOWTO guides you through the installation of SQL Anywhere Studio 7.0.2 for Linux and the basic operation and administration of Adaptive Server Anywhere databases.

1.1. New versions of this document

The latest version of this document should always be available at the Linux Documentation project website (http://www.linuxdoc.org/).

1.2. Content and Audience

Within this document, you will find a list of the supported Linux distributions ("Section 2"). It is intended for moderately experienced users of Linux or UNIX. Familiarity with relational database concepts is certainly useful, but not a requirement. "Section 1.5" contains a summary of relational database concepts.

1.3. Adaptive Server Anywhere features

Adaptive Server Anywhere (Adaptive Server Anywhere) is the full SQL relational database management system at the heart of SQL Anywhere Studio. Ideally suited for use as an embedded database, in mobile computing, or as a workgroup server, it includes the following among its features:

- Economical hardware requirements
- Designed to operate without administration
- Designed for mobile computing and synchronization
- Ease of use
- High performance
- Cross−platform solution
- Standalone and network use
- Industry standard interfaces

Some of the more specific features include:

- Stored procedures and triggers
- Java support for logic and datatypes

For further details about Adaptive Server Anywhere, please visit the following links:

- http://www.sybase.com/detail/1,3693,1002624,00.html is a datasheet on SQL Anywhere Studio. It includes some data on Adaptive Server Anywhere, which ships as a component of SQL Anywhere Studio.
- http://www.sybase.com/detail/1,3693,1009210,00.html has some information on the features and system requirements of SQL Anywhere Studio and points you to the download location for SQL Anywhere Studio for Linux 7.0.

1.4. Quirks

1.4.1. Alt and Function keys

Sometimes the Alt keys or the F1−F10 keys may not function in the terminal where you are running Interactive SQL.

To emulate the Alt key, press Ctrl−A. Then press whatever key was to be pressed with the Alt key. For example, instead of pressing Alt−F, you would press Ctrl−A, then F.

To emulate the function keys, press Ctrl−F, followed by the number of the function key you wanted to press. For example, instead of pressing F9, you would press Ctrl−F, then 9. For F10, use the zero key.

1.5. What's a Relational Database?

If you are already familiar with relational databases, you can skip this section.

1.5.1. Definition

A *relational database−management system* (RDBMS) is a system for storing and retrieving data, in which the data is organized in tables. A relational database consists of a collection of tables that store interrelated data.

If that doesn't quite make sense yet, read on.

1.5.2. Example

Suppose you have some software to keep track of sales orders, and each order is stored in the form of a table, called sales_order. It has information about the customer (for example, her name, address and phone number), the date of the order, and information about the sales representative (for example his name, department, and office phone number). Let's put all this into a table, with the data for a few orders:

Table 1. The sales_order table

cust_name	*cust_address*	*cust_city_state_zip*	*cust_phone*	*order_date*	*emp_name*	*emp_dept*	*emp_phone*
M. Devlin	3114 Pioneer Ave.	Rutherford, NJ 07070	2015558966	19930316	R. Overbey	Sales	5105557255
M. Devlin	3114 Pioneer Ave.	Rutherford, NJ 07070	2015558966	19940405	M. Kelly	Sales	5085553769
J. Gagliardo	2800 Park Ave.	Hull, PQ K1A 0H3	8195559539	19940326	M.Garcia	Sales	7135553431
E. Peros	50 Market St.	Rochester, NY	7165554275	19930603	P. Chin	Sales	4045552341

		14624					
E. Peros	50 Market St.	Rochester, NY 14624	7165554275	19940127	M.Garcia	Sales	7135553431
E. Peros	50 Market St.	Rochester, NY 14624	7165554275	19940520	J. Klobucher	Sales	7135558627

Everything appears nice and ordered, but there's a fair bit of redundancy. M. Devlin's name appears twice, along with his address and phone number. E. Peros' details appear three times. If you look carefully at the employee side of things, you'll notice that M. Garcia is repeated, as well.

Wouldn't it be nice if you could separate that information and only store it once, rather than several times? In the long term, it would certainly save disk space and allow for greater flexibility. Since redundant data entry is minimized, it would also reduce the chances of erroneous data entering the database, increasing consistency. Well, we can see three different entities involved here: the customer, the order, and the employee. So let's take each of the individuals, put them into categories, and give them identification numbers so they can be referenced.

Table 2. The customer table

id	*name*	*address*	*city_state_zip*	*phone*
101	M. Devlin	3114 Pioneer Ave.	Rutherford, NJ 07070	2015558966
109	J. Gagliardo	2800 Park Ave.	Hull, PQ K1A 0H3	8195559539
180	E. Peros	50 Market St.	Rochester, NY 14624	7165554275

Table 3. The employee table

id	*name*	*dept*	*phone*
299	R. Overbey	Sales	5105557255
902	M. Kelly	Sales	5085553769
667	M.Garcia	Sales	7135553431
129	P. Chin	Sales	4045552341
467	J. Klobucher	Sales	7135558627

Table 4. The new sales_order table

id	*cust_id*	*order_date*	*sales_rep_id*
2001	101	19930316	299
2583	101	19940405	902
2576	109	19940326	667
2081	180	19930603	129
2503	180	19940127	667

2640	180	19940520	467

As you can see, each customer's information is stored only once, and the same goes for each employee. The sales_order table is a lot smaller, too. Each row, representing a sales order, refers to a cust_id and an emp_id.

By looking up the customer corresponding to a cust_id (which is unique), one can find all the needed data on that customer, without having to repeat it in sales_order. In addition, an id column has been added. Its purpose will be explained in the next section.

Why do this, you ask? By eliminating redundancy, this kind of structure reduces the opportunities for inconsistencies to seep in, in addition to lowering storage requirements. If you had to change E. Peros' address in the old sales_order table, you'd have to do it three times, which would take three times as long and give you three times as many chances to make an error. In the newer table, all you'd have to do is change her address once, in the customer table. Also, by carefully separating data, you make access control simpler.

Finally, can you spot another redundancy? The employee table has "Sales" all the way down the dept column. For an organization with multiple departments, you'd want to add a department table and reference it from a dept_id column instead.

1.5.3. Primary and Foreign Keys

As described in the previous section, you can separate a table into interrelated tables. But how do you go about relating tables to each other? In relational databases, primary keys and foreign keys help you link tables together. Primary keys are columns that uniquely identify each row of a table, and foreign keys define the relationship between the rows of two separate tables. Proper use of primary and foreign keys will help you efficiently hold information without excessive redundancy.

Every table should have a primary key to ensure that each row is uniquely identified. This often takes the form of an ID number being assigned to each row, as in the previous section's example. The id column forms the primary key.

As long as you can guarantee the uniqueness of the data in a particular column, though, that column can be a primary key. For example, if you only want one entry per day to be put into a particular table, you could use the date as that table's primary key.

Tables are related to one another by foreign keys. In the sales_order example, the cust_id and sales_rep columns would be called foreign keys to the customer and employee tables, respectively. For terminology's sake, you might want to know that in this case, the sales_order table is called the *foreign* or *referencing* table, while the customer and employee tables are called the *primary* or *referenced* tables.

2. Requirements

2.1. System requirements

Adaptive Server Anywhere requires that you have the following installed on your system:

- kernel 2.2.5−15 and up (2.2.x series)
- glibc−2.1 or up
- pthreads−0.8 or higher (included usually as part of glibc)
- libstdc++−2−libc6.1

2.2. Supported distributions

At present, the following Linux distributions are supported:

- Caldera 2.4
- Red Hat 7.0, 6.2, 6.1 or 6.0
- TurboLinux 6.1
- SuSE 7.0, 6.4, 6.3 or 6.2

NOTE: The glibc and gcc released with Red Hat Linux 7.0 require patches before you can use Adaptive Server Anywhere. You can find them at http://www.redhat.com/support/errata/rh7−errata−bugfixes.html.

3. Installation

3.1. Process

1. Log on as root.
2. Place the CD−ROM into the CD−ROM drive.
3. Mount the CD−ROM. Usually, it gets mounted to /mnt/cdrom. If so, enter the following command:

   mount /mnt/cdrom
4. At a command prompt, change to the CD−ROM directory. If the CD−ROM was mounted to /mnt/cdrom/, use the following command:

   cd /mnt/cdrom
5. Start the setup script by entering the following command:

   ./setup
6. The setup script prompts you with information about installing SQL Anywhere Studio for UNIX. Enter any information you are prompted for, and press the Enter key to continue.

By default, SQL Anywhere Studio is installed into a directory named SYBSsa7 under /opt/sybase on Solaris, Linux, and HP−UX, and under /usr/lpp/sybase on AIX. You can specify another installation directory if you wish.

3.2. Distribution−specific considerations (for TurboLinux and Caldera)

After installation, you should follow these instructions if you are running either TurboLinux 6.0 or Caldera 2.2.

For TurboLinux 6.0 only, change to directory /usr/lib and create a symbolic link using the following command.

ln −s libstdc++−libc6.1−2.so.3 libstdc++−libc6.1−1.so.2

For Caldera 2.2 only, change to directory /usr/lib and create a symbolic link using the following command.

ln −s /usr/lib/libstdc++−2.9.0 /usr/lib/libstdc++−libc6.1−1.so.2

3.3. Setting the Environment Variables

Each user who uses the software must set the necessary environment variables for Adaptive Server Anywhere. To help you do that, the installation program puts two script files, asa_config.sh and asa_config.csh, in the directory /*InstallDir*/SYBSsa7/bin. *InstallDir* is the directory where you chose to install Adaptive Server Anywhere.

Depending on which shell you're using, enter the appropriate command from *InstallDir*.

Table 5.

If you're using this shell...	*...use this command.*
sh, ksh, bash	. ./SYBSsa7/bin/asa_config.sh
csh, tcsh	source ./SYBSsa7/bin/asa_config.csh

You may also want to insert the above commands into your copy of .profile or .bash_profile to have the environment variables ready every time you log in.

3.4. Where did it get installed?

Table 6.

Most Adaptive Server Anywhere command line utilities (names beginning with db)	/*InstallDir*/SYBSsa7/bin
Sybase Central	/*InstallDir*/shared/sybcentral40/java
Sample database	/*InstallDir*/SYBSsa7
Online documentation	/*CDROM*/help/contents.htm or/InstallDir/SYBSsa7/doc/contents.htm

CDROM is the directory where your CD−ROM is mounted, which is usually /mnt/cdrom/.

InstallDir is the directory where you chose to install Adaptive Server Anywhere.

The first two directories are put into the path by asa_config.sh or asa_config.csh, so if you've already executed one of them as mentioned in the previous section, you won't have to change directories to get to most of the executables associated with Adaptive Server Anywhere.

4. Creating, Running and Connecting to Databases

4.1. Creating a database

When you ask Adaptive Server Anywhere to create a database, it creates the main database file, which contains the following objects, among others:

- user tables
- indexes
- views
- system tables

The maximum size of a database file depends on your file system and the page size you choose. Database files are limited to 256 million database pages or the filesize limit, whichever is reached first. UNIX files can be as large as 1 Tb, in some cases−see the Physical Limitations chapter of the Adaptive Server Anywhere Reference Manual or your Linux documentation for more information. You can set pages to be 1, 2, 4, 8, 16, or 32 kb in size, but it is not recommended that you use a page size of 1 kb. The default page size is 2 kb.

By default, Adaptive Server Anywhere also creates a file called the *transaction log*. Besides improving performance, the transaction log is vital to Adaptive Server Anywhere replication systems and database recovery in event of system failures. When possible, it is recommended that the transaction log be placed on a physical device (in most cases, a disk drive) separate from the main database file, to reduce the chances of both the main database file and transaction log being affected in the event of a media failure. You can specify the name and location of the transaction log when you create the database.

This section shows you how to create databases at either the command prompt or in Interactive SQL. You can also create databases through Sybase Central, if you prefer, by opening the Utilities folder under Adaptive Server Anywhere 7.

4.1.1. Creating a database from the command prompt

The command line utility for creating a database is *dbinit*.

Syntax:

dbinit [switches] db−file−name

db−file−name is the name you would like to give to your database file, for example, mydb.db. If you issue the command "dbinit −?" you'll be shown the above syntax, along with a list of options you can use.

To create your first Adaptive Server Anywhere database on Linux, enter the following command:

dbinit −t './logs/mydb.log' p 4096 mydb.db

This command creates a database in the current working directory called mydb.db with a page size of 4096 bytes, specified by the −p switch. Assuming the directory exists, it also creates the transaction log mydb.log in the subdirectory "logs," specified by the −t switch. Adaptive Server Anywhere databases carry the extension ".db" .

4.1.2. Creating a database from Sybase Central

To create a database in Sybase Central, open the Adaptive Server Anywhere section of the left pane, and select Utilities. Double−click Create Database in the right pane, and follow the on−screen instructions.

4.2. Running a database server and starting databases

There are two versions of the database server installed on your machine. If you are just using Adaptive Server Anywhere locally, use the personal database server (dbeng7). If you are going to connect to the Adaptive Server Anywhere database over a network, however, you should use the network database server (dbsrv7). Examples in this document use dbeng7, but the two commands are, for the most part, interchangeable. See the table below for specific differences.

Table 7. Differences between the Personal and Network database servers

	Personal database server	*Network database server*
Name of executable	dbeng7	dbsrv7
Local connections	Yes	Yes
Network connections	No	Yes
Maximum number of connections	10	Depends on license
Available communications protocols	Shared memory, TCP/IP	Shared memory, TCP/IP
Maximum number of CPUs for request processing	2	Unlimited
Default/Maximum number of internal threads	10/10	20/Unlimited

Syntax:

(dbeng7 | dbsrv7) [server−switches] [database−file [database−switches],]

database−file specifies the path and filename to the database. You aren't actually required to specify a database file when you start up the database server, but if you don't, you must specify a name for the server using the −n switch. By default, if you do not specify a name for the database, it takes on the name of the database file, minus the path and extension. Similarly, if you do not specify a name for the database server (which you can do in server−switches), it takes on the name of the first database that was started on it.

For full details on the usage of dbeng7 and dbsrv7, see "The database server" in the Adaptive Server Anywhere Reference.

To start up the Adaptive Server Anywhere personal database server, but not a database, and name it MyServer, issue the following command at a prompt:

dbeng7 −n MyServer

To start up the Adaptive Server Anywhere personal database server and name it MyServer, then start a database on MyServer from mydb.db, naming it MyDatabase, issue the following command:

dbeng7 −n MyServer mydb.db −n MyDatabase

In the latter case, if you don't name the database server MyServer, it would be named MyDatabase instead.

There's a plethora of other switches available for the server. You can get a full listing of them by typing "dbeng7 −?" at a command prompt. A few important switches include the following:

- −c, for specifying Adaptive Server Anywhere's cache size
- −x allows you to specify the communications protocols
- −gt allows you to specify the number of processors to be used
- −ud tells the server to run as a daemon in UNIX (explained below)

4.2.1. Running the server as a daemon

Sometimes it's necessary for the server to run outside of the current session (that is, regardless of who, if anyone, is logged in). To do so, use the −ud switch at the command line when starting the server to run it as a daemon.

The following command would start up a database server as a daemon, using the database we created before:

dbsrv7 −ud −n MyDatabase mydb.db

NOTE: Using "&" to run the database server in the background does not work.

4.3. Stopping the database server

Assuming you have the appropriate authority, you can stop the database server using any of the following methods:

- the dbstop command line utility
- using the STOP ENGINE SQL statement
- pressing the Q key when the server display window has the focus

NOTE: While the term *engine* is part of the SQL statement's name, *server* is the common term now used. This document will use the term *server* unless referring explicitly to the STOP ENGINE SQL statement.

By default, any user can stop a personal database server, but only a user with the DBA authority can stop a network database server. (This default can be changed by using the −gk switch when starting the server−see the Adaptive Server Anywhere Reference for details.)

The command line utility syntax is as follows:

dbstop [switches] {name}

If you are issuing dbstop to stop a locally−running server, you can simply specify the name of the database server in {name}. If the server is not running locally, you need to create a connection to the server before you can tell it to stop. The −c switch allows you to specify a connection string for the database running on the server that you would like to stop. To stop MyServer, execute the following command:

dbstop −c "uid=DBA;pwd=SQL;eng=MyServer;dbn=MyDatabase"

In this instance, you could also just give the server name, since the server is running locally:

dbstop MyServer

The first command connects to the database named MyDatabase on the server MyServer, then stops the server named MyServer. In the case that no databases are active on the server, you have to add "dbn=utility_db" to the connection string.

Let's say "Club" is the name of one of the databases running on a server named "Goliath," and you want to stop all the databases running on Goliath, including Club. The following command accomplishes that, as well as shutting down the database server:

dbstop −c "uid=DBA;pwd=SQL;eng=Goliath;dbn=Club"

If you have a database server named "David" running without any databases started on it, you can stop the server using the following command:

dbstop −c "uid=DBA;pwd=SQL;eng=David;dbn=utility_db"

The syntax for the STOP ENGINE statement is as follows:

STOP ENGINE [server−name] [UNCONDITIONALLY]

The server named server−name is stopped. If server−name is omitted, the currently running database server is stopped. If UNCONDITIONALLY is specified, the database server is stopped whether or not there are still connections to the server.

4.4. Stopping databases

It's also possible to stop individual databases without stopping the server, or any of the other databases that might be running on it. To do so, use the STOP DATABASE SQL statement.

Syntax:

STOP DATABASE database−name [ON engine−name] [UNCONDITIONALLY]

You specify the name of the database that you would like to stop in database−name, with the restriction that the database specified cannot be the currently connected one. The "ON engine−name" clause can be used only in Interactive SQL. You use it to specify the server that the database is running on. Outside of Interactive SQL, the database can only be stopped if it is on the current server. The UNCONDITIONALLY keyword forces databases to be stopped, even if there are connections to it. By default, you can't stop a database if there are connections active.

4.5. Connecting to a database

You can connect to an Adaptive Server Anywhere database via any of the following interfaces:

- ODBC
- OLE DB or ADO
- Embedded SQL
- Sybase Open Client
- JDBC

Regardless of how you connect, you must specify some parameters, such as a username and password, to establish a connection to the database. These can be specified in a connection string, the SQLCONNECT environment variable, an ODBC data source configuration, or the fields of a dialog box.

In this section, you'll find explanations on how to connect via SQL and ODBC.

As the Adaptive Server Anywhere network server is a client/server database, you may connect to a Linux−hosted database from Windows−based PCs and other non−Linux devices, as well as Linux applications. Programming interfaces such as OLE DB or ADO are available ony on Windows, but can still be used against a Linux−hosted database.

4.5.1. Connection strings

Connection strings are frequently used when performing actions on a database. They consist of a list of parameter settings, delimited by semicolons and enclosed in double quotes. There should be no extra spaces in a connection string.

Example:

"uid=DBA;pwd=SQL"

The short strings of letters just before each equal sign (in this example, uid, pwd, and dbf) are called *keywords*, which each correspond to a connection parameter. There are many connection parameters available, and they are listed in the Connecting to a Database chapter of the Adaptive Server Anywhere User's Guide. They are also described in detail in the Connection and Communication Parameters chapter of the Adaptive Server Anywhere Reference.

When Adaptive Server Anywhere utilities are looking for connection parameters, they check the SQLCONNECT environment variable for any parameters that were left out of the connection string. If you're putting connection parameters into the SQLCONNECT environment variable, replace the equal signs with number (#) signs. In bash you would use the following command:

SQLCONNECT='uid#DBA;pwd#SQL'

The single quotes are necessary in the above command because semicolons can be used to separate bash commands. You can also use double quotes.

To make SQLCONNECT available in subsequent shells, you'd need to use "export SQLCONNECT" to export the SQLCONNECT variable to the environment. You may also want to put these commands into your .bash_profile (or .profile, if you're using another shell) if you want the same connection parameters to be available each time you log in.

4.5.2. Connecting from Interactive SQL

To connect to a database from Interactive SQL, go to the Command menu, and choose "Connect...", then fill in the dialog box as appropriate.

4.5.3. Connecting via ODBC

ODBC (which stands for Open Database Connectivity) is an industry−standard interface for connecting client applications to relational and non−relational DBMSes. When you create an ODBC data source, it encapsulates the data and any other information required to get the data, including connection parameters.

4.5.3.1. Setting up ODBC with Adaptive Server Anywhere

To connect to Adaptive Server Anywhere from ODBC applications on Linux, you can either use Sybase's ODBC driver as a driver manager, or use a third−party ODBC driver manager such as iODBC or unixODBC. If you choose the latter route, follow the installation instructions for the driver manager you've chosen and choose dbodbc7.so (which resides in the sybase/SYBSsa7/lib directory) as the ODBC driver for Adaptive Server Anywhere.

If you choose the former route, you can use Adaptive Server Anywhere's ODBC driver as a driver manager if you will only be connecting to Adaptive Server Anywhere databases. To do so, you need to create a few symbolic links so that ODBC driver manager requests get routed to the Sybase ODBC driver. From the sybase/SYBSsa7/lib subdirectory, enter the following commands:

$ ln −s dbodbc7.so libodbc.so

$ ln −s dbodbc7.so libodbc.so.1

$ ln −s dbodbc7.so libodbcinst.so

$ ln −s dbodbc7.so libodbcinst.so.1

That's it!

4.5.3.2. About ODBC data sources

Data sources exist on the client computer, with at least one for each database accessible via ODBC. They reside in the .odbc.ini file or in a separate .dsn file.

If the client computer is running Linux or another UNIX operating system, ODBC data sources can be used both for ODBC applications as well as for the Interactive SQL and Sybase Central utilities.

NOTE: The database server looks for *.odbc.ini* in the following locations, among several others:

1. ODBCINI environment variable
2. ODBCHOME and HOME environment variables
3. The user's home directory
4. The current directory
5. The path
6. The root directory

If no .odbc.ini file exists in your home directory, you'll have to create one in your home directory. You can check if one exists by using the command "ls −a ~/.odbc.ini".

You manage ODBC data sources using the *dbdsn* command line utility.

Syntax:

```
dbdsn [ modifier−switches ]
 { −l
 | −d dsn
 | −g dsn
 | −w dsn [details−switches]
 | −cl                          }
```

dbdsn has four main modes of operation, and its behaviour depends on whether you choose the −l, −d, −g, or −w switch. Where applicable, the name of the data source to be operated on is specified by dsn.

- the −l switch lists the data sources that have been defined
- the −d switch deletes the specified data source
- the −g switch gives you the details of the specified data source
- the −w switch creates a new DSN using parameters specified in details−switches

The most important details−switch is the −c switch, which allows you to specify the usual database connection parameters. You can also specify the name of a database server as a details−switch. Type "dbdsn −cl" to display a list of available connection parameters.

To create a new data source named MyNewDSN for the server MyServer, execute the following command at a shell prompt:

dbdsn −w MyNewDSN −c "uid=dba;pwd=sql;eng=MyServer"

If there is a data source named MyNewDSN already existing, dbdsn asks if you would like to overwrite it.

Conversely, to delete MyNewDSN, execute the following command:

dbdsn −d MyNewDSN

The modifier−switches control how dbdsn outputs its messages to screen, and whether or not data sources can be overwritten without confirmation. For more information on other dbdsn options, see "The Data Source utility" under the Database Administration Utilities chapter of the Adaptive Server Anywhere Reference.

4.5.3.3. Connecting to an ODBC data source

Once you've created an ODBC data source, you can access it through the DSN (DataSourceName) connection string keyword.

For an ODBC data source called mydatasrc, for example, use the following connection string to connect to the database associated with it:

"dsn=mydatasrc"

NOTE: Explicitly−provided connection parameters and SQLCONNECT override any parameters provided in the ODBC data source, in that order.

NOTE: The FileDSN connection parameter is not yet available in version 7.0.2 of Adaptive Server Anywhere. Future versions of Adaptive Server Anywhere should support File DSNs.

---

5. Backing up and Restoring a Database

Creating a backup of your data is a simple, essential component of any serious installation. Adaptive Server Anywhere includes utilities to help minimize data loss in case your data becomes corrupt as a result of media failure, power outage, or other failure.

5.1. Creating a Backup of the Database

Backups of Adaptive Server Anywhere databases can be performed through the dbbackup command line utility, SQL, or Sybase Central. Both full backups and incremental backups can be performed, and they can be performed either online or offline (that is, whether the server is running or not, respectively). In addition, backups can be performed both from the server side and from the client side.

5.1.1. Full vs. Incremental Backups

A full backup makes copies of the main database file and the transaction log file. While it's the most basic and essential type of backup, it usually isn't practical to regularly perform full backups of large databases. As a result, incremental backups are commonly used.

An incremental backup makes a copy of the transaction log alone. It takes place as part of a cycle that begins with a full backup, which is then followed by a given number of incremental backups. Since only the transaction log is copied, an incremental backup uses less time and resources, making it particularly suited for large databases. Keep in mind, though, that the more time you leave between full backups, the greater the risk of losing data in the event that one of the transaction logs becomes unusable.

5.1.2. Online vs. Offline Backups

An online backup is performed without stopping the database server. It provides a consistent snapshot of the database, even as the database is modified. Online backups are useful for databases with high availability requirements, but they won't complete until all active transactions are complete.

In contrast, offline backups are performed once the database server has been shut down. They're useful for when the database can be taken down on a regular basis. You make offline backups simply by copying the pertinent files to another location using the cp command in a terminal window.

In either case, both full and incremental backups can be performed.

5.1.3. Server−side vs. Client−side Backups

An online backup can be performed from a client using the *dbbackup* command line utility. This is known as a client−side backup, and it puts a backup of the database on the client machine.

An online backup can also be performed on the server by issuing the BACKUP statement in SQL. Server−side backups are generally faster, owing to the fact that client−side backups usually depend upon transport across networks.

5.1.4. How to make a backup

5.1.4.1. From the command line

The command line utility for making a backup of your database is *dbbackup*. Its syntax is as follows:

dbbackup [switches] directory

directory specifies a destination directory for the backup files. Some useful switches include the following:

- −c is used to specify a connection string to the database to be backed up
- −d creates a backup of the main database file only
- −t creates a backup of the transaction log only
- −r renames any previous transaction log backups and creates a new one. It is necessary for replication systems.
- −x deletes any previous transaction log backups and creates a new one. It should not be used in replication systems.

For example, if you were creating your first backup, you would want to create a full backup of MyDatabase. To put it in ./backups, use the following command:

dbbackup −c "uid=DBA;pwd=SQL;dbn=MyDatabase" ./backups

The next few backups could be incremental backups, so use the following:

dbbackup −t −r −c "uid=DBA;pwd=SQL;dbn=MyDatabase" ./backups

5.1.4.2. From SQL

If you prefer to back up your database from Interactive SQL, the SQL statement is BACKUP DATABASE. You must have DBA authority to use BACKUP DATABASE, whose syntax is as follows:

```
BACKUP DATABASE DIRECTORY backup−directory
  [ WAIT BEFORE START ]
  [ DBFILE ONLY ]
  [ TRANSACTION LOG ONLY ]
  [ TRANSACTION LOG RENAME [ MATCH ] ]
  [ TRANSACTION LOG TRUNCATE ]
```

5.1.4.3. From Sybase Central

To make a backup from Sybase Central, open the Utilities folder under "Adaptive Server Anywhere 7" and double−click "Backup Database" to open a dialog box which will guide you through the backup process.

5.2. Validating the database and its backup

You should regularly use either Sybase Central, SQL, or the dbvalid command line utility to validate a backup of your database in read−only mode, and, if errors are found, make repairs against the original database. *Never* make changes to a backup database! To read more about validation, see "Validating a database" and "Validating a transaction log" under the Backup and Data Recovery chapter of the Adaptive Server Anywhere User's Guide.

5.3. Recovering the database

Depending on the way your database and its backups are set up, and the status of your files after a media failure, there are several possible processes involved in how you go about recovering data. For information on how to recover data in various situations, see the Backup and Data Recovery chapter of the Adaptive Server Anywhere User's Guide.

6. Managing a Database

6.1. Tables

All data in relational databases is held in tables. Each column is assigned a data type, and each row of a table holds a value for each column. The following are true for any table in a relational database:

- There is no significance to the order of rows and columns.
- Each row contains exactly one value per column.
- All values in a column are of the same type.

Here are some things to keep in mind when designing your database:

- give every table a primary key
- make sure that each table holds information about one specific entity
- foreign keys form the relationships between tables (and therefore entities)

6.1.1. Creating a Table

When you first create a database in Adaptive Server Anywhere, the only tables it contains are the system tables. To create tables to hold your data, use either the CREATE TABLE statement in SQL or the Sybase Central Table Editor. You must have the DBA or RESOURCE authority to create a table, and you must have the DBA authority make another user its owner.

The CREATE TABLE statement has an extremely broad range of options that are documented in the Adaptive Server Anywhere Reference, so only a small subset of options are described here. The basic syntax is as follows:

```
CREATE TABLE owner.table-name
  (column-name datatype [, column-name datatype]...)
```

The "owner." portion before tablename is optional, and is used by a user with the DBA authority to make another user the owner of the new table. table−name and column−name, respectively, are the names of the table and its columns. Insert the words PRIMARY KEY after datatype to make it the primary key.

See the SQL Data Types chapter of the Adaptive Server Anywhere Reference for a list of the types available and their characteristics.

To create a table named customer with columns id, name, address, city_state_zip, and phone, with id as the primary key, for example, use the following CREATE TABLE statement:

```
create table customer
 (id integer not null primary key,
  name char ( 35 ),
  address char ( 35 ),
  city_state_zip char ( 35 ),
  phone char ( 12 )
 )
```

It's also important to add "not null" in the case of id, since it's the primary key.

To create a table in Sybase Central, connect to your database and open its Tables folder. If you double−click "Add Table," Sybase Central Table Editor will be opened and using the button bar, you can set up the table as you wish. Hover the mouse pointer over each button to find out what it does. Don't forget to make a primary key before you close the Table Editor!

Some table creation options documented in the Adaptive Server Anywhere Reference but not here that you might be interested in include automatic incrementation (often used on the primary key), constraints, and foreign keys.

6.1.2. Making Alterations to Tables

You can make many kinds of changes to a table once it's been created. Some of the things you can do include the following:

- rename a table
- add, remove, or rename columns
- change the datatype, default value, or length of a column

As with creating tables, you can alter them through SQL or Sybase Central. To alter a table in SQL, you use the ALTER TABLE statement. ALTER TABLE has a great variety of options, which are described in detail in the Adaptive Server Anywhere Reference. You'll see a few basic examples here just to get you started.

To rename the customer table to cust:

```
alter table customer
  rename cust
```

To add a company_name column to cust, with a maximum length of 35 characters:

```
alter table cust
  add (company_name char (35) )
```

To give company_name a default value of "n/a" :

```
alter table cust
  alter company_name set default 'n/a'
```

6.2. Users, permissions, and authorities

NOTE: Before putting an Adaptive Server Anywhere database into serious usage, your first order of business as the database administrator (DBA) should be to change the DBA password from the default password, "SQL." For details on how to do this, see section 6.2.5.

This section describes the user IDs that are created for each database, briefly describes how to create new user IDs, and goes over some of the ways you can use user IDs to control outsiders access of data. For more information on user IDs, groups, and permissions, see the Managing User IDs and Permissions chapter of the

Adaptive Server Anywhere User's Guide.

6.2.1. User IDs

6.2.1.1. Special user IDs

When Adaptive Server Anywhere databases are initialized, two groups and two user IDs are created. The two groups created are SYS and PUBLIC. The two user IDs created are DBA and dbo.

SYS is a user as well as a group, but no one can connect to the database using the user ID SYS. SYS owns the system tables and the system views, and only SYS can update the system tables.

PUBLIC is a member of the SYS group, and has only SELECT permissions on most system tables and system views. Since new user IDs are, by default, members of PUBLIC, you should revoke PUBLIC's membership in SYS if you want new users to have no permissions by default.

The DBA user can directly modify any part of an Adaptive Server Anywhere database except the system tables. This is why it's important to change the default DBA password from "SQL." You should be cautious when giving DBA authority to a user (see the DBA Authority section below). If a user needs DBA authority, s/he should be given DBA authority, rather than the DBA's password.

6.2.1.2. Creating new user IDs

The SQL statement to add a new user ID is GRANT CONNECT.

Syntax:

```
GRANT CONNECT TO userid1
  IDENTIFIED BY password1
```

To add a user ID with the name Mortimer, execute the following SQL statement:

```
grant connect to mortimer identified by
monkey
```

6.2.2. Permissions

This section explains permissions on tables that can be granted to users. Permissions are granted on a user−by−user basis.

There are a few different table permissions that can be granted to a user, and they are each granted separately.

- *SELECT* allows the user to *read* data, and can be restricted to particular columns.
- *INSERT* allows the user to *add* data.
- *UPDATE* allows the user to *change* data, and can be restricted to particular columns.
- *DELETE* allows the user to *remove* data.
- *ALTER* allows the user to *modify the structure* of a table.

- *REFERENCES* allows the user to add indexes, primary keys, and foreign keys.
- *ALL* includes all the above permissions.

With the exceptions of ALTER and REFERENCES, which apply to tables exclusively, the table permissions apply to both tables and views. The SQL syntax for granting permissions is as follows:

```
GRANT [ SELECT (column-name, ...)
      | INSERT
      | UPDATE (column-name, ...)
      | DELETE
      | ALTER
      | REFERENCES
      | ALL                                          ]
ON table-name
TO userid
```

The user userid is given the specified permission(s) on the table identified by table−name. If the permissions granted include SELECT and/or UPDATE, they are granted only on the columns specified in column−name.

Let's say a list of available banana types is stored in the type and quantity columns of a table named banana_supply. To allow Mortimer to see a list of available banana types along with their quantities, use the following SQL statement:

grant select on banana_supply (type, quantity) to mortimer

When you grant a permission to a user, you have the option of granting him the ability to grant that same permission to others. To grant a user the permission to do so, add WITH GRANT OPTION to the end of your users GRANT statement when you're granting them their permissions.

To allow Mortimer to see a list of banana types available along with the quantities of each, as well as allowing him to grant others the same SELECT permission, use this SQL statement:

```
grant select on banana_supply (type, quantity)
to mortimer
      with grant option
```

6.2.3. Authorities

An authority is a different level of permission. There are two types of authority.

6.2.3.1. RESOURCE authority

A user with the RESOURCE authority can create and drop database objects such as tables, views, stored procedures, and functions. The RESOURCE authority also allows the user to create and remove user IDs and passwords. To give userid the RESOURCE authority, execute the following SQL statement:

GRANT RESOURCE TO userid

6.2.3.2. DBA authority

A user with the DBA authority can perform any database operation, and automatically has all permissions on all tables, except the system tables. The DBA can create and remove user IDs and passwords, grant RESOURCE and DBA authority, and unload and reload the database.

GRANT DBA TO userid

6.2.4. Removing Users and Revoking Permissions

The SQL statement to delete a user ID is REVOKE CONNECT.

Syntax:

REVOKE CONNECT FROM userid [, userid]

As suggested by the portions in square parentheses, it's possible to remove multiple user IDs in a single statement. For example, to remove the user IDs for Mortimer and Chestington, execute this statement:

revoke connect from mortimer, chestington

To revoke permissions or authorities given to a particular user, you take the original granting statement, replace the GRANT with REVOKE, and replace the TO with FROM. To take away Mortimer's permission to view the banana_supply table, for example, use this REVOKE statement:

revoke select on banana_supply (type, quantity) from mortimer

6.2.5. Changing Passwords

To change the password associated with a particular user ID, use a GRANT CONNECT statement again:

GRANT CONNECT TO userid IDENTIFIED BY newpassword

For example, to change the DBA's password from "SQL" to "d0n13xw9," use this statement:

grant connect to DBA identified by d0n13xw9

6.3. Making the database more secure

Some of the Adaptive Server Anywhere features you may wish to use in building a secure environment for your data include the following:

- *User identification and authentication* control access to databases.
- *Permissions and authorities*, which have already been explained in previous sections, control the actions a user can carry out while connected to a database.

* *Views and stored procedures* allow you to carefully tune the data a user can access and the operations a user can execute.
* *Connection encryption* can prevent unauthorized persons from snooping.

Some of these features have already been mentioned in this HOWTO, and some of them will be elaborated upon in the following sections. While the concepts of triggers, procedures, and views will be introduced so you can decide if and how you'll use them, their implementation won't be discussed. You can find indepth information on them, as well as details on their implementation, in the sections of the Adaptive Server Anywhere User's Guide listed below:

Table 8.

Chapter	Section
Using Procedures, Triggers, and Batches	Benefits of procedures and triggers
Managing User IDs and Permissions	Using views and procedures for extra security

6.3.1. Increasing password security

By default, passwords can be any length. For greater security, you can enforce a minimum length on all new passwords, to make them more difficult to guess. You do this by setting the MIN_PASSWORD_LENGTH database option to a greater value. The following statement enforces a minimum password length of 8 characters:

set option public.min_password_length = 8

Check the "Changing Passwords" section of this document to learn how to change a user's password, and don't forget to change the DBA's password!

6.3.2. Views, procedures, and triggers

Views are useful when it is appropriate to give a user access to just one portion of a table. The portion can be defined in terms of rows or in terms of columns. For example, you may wish to prevent a group of users from seeing the quantity column of the banana_supply table, or you may wish to limit a user to see information on a particular type of banana.

While views restrict access based on the data, procedures and triggers restrict access based on the actions a user can take. Procedures and triggers store SQL statements in a database for use by all applications. They execute under the table permissions of the associated table's owner, regardless of the permissions of the user who either executes the procedure or fires the trigger.

Procedures are invoked by a CALL statement, and can take values as well as return them. Unlike procedures, however, triggers are can neither take values nor return them, and are invoked by insertions, updates, or deletions in the table it is associated with. Permissions are not associated with triggers. They execute when the action defined to fire them is performed, regardless of the user.

For strict security, you can prevent all access to the tables, and grant permission to users to execute certain stored procedures that carry out specific tasks. This approach strictly defines the manner in which the database can be modified.

6.3.3. Encrypting client/server communications

Encrypting client/server communications prevents third parties from reading messages being sent between the client and the server. It can be enabled from either the server side or the client side. To enable encryption from the server, use the −e option at server startup. For example, use the following command to start up the database server to accept encrypted connections to mydb.db over TCP/IP:

dbsrv7 −e −x tcpip mydb.db

To enable encryption from a particular client, use the ENC keyword in the connection string. For example, to encrypt a connection over TCP/IP to mydb.db, your connection string would appear as follows:

"uid=mortimer;pwd=monkey;links=tcpip;eng=MyServer;dbf=mydb.db;enc=true"

For more information about client/server communications encryption, look for the −e command−line option under "The database server" in the Adaptive Server Anywhere Reference Manual, and for "Encryption connection parameter" under "Connection parameters" .

7. Where to get more information

On−line help is available on your cdrom. If your computer is set up to mount the CD−ROM to /mnt/cdrom/ the help is located in /mnt/cdrom/help/contents.htm. Open it with Netscape Navigator, or any other web browser that supports tables. Style sheets support is recommended, but not necessary.

A *FAQ* is available for the UNIX version of Adaptive Server Anywhere at http://www.sybase.com/detail/1,3693,1011965,00.html

Check if there have been any *bug fixes* or *updates* posted at http://downloads.sybase.com/swx/sdmain.stm.

Newsgroups can be read from the web or with a news reader. The newsgroups sybase.public.sqlanywhere.general and sybase.public.sqlanywhere.linux are most likely to be relevant. To view newsgroups on the web, visit http://www.sybase.com/support/newsgroups. Be sure to search old threads for similar problems. It may already have been resolved.

8. Legalities and Acknowledgements

8.1. Copyright and Licenses

8.2. Names and Contacts

This document was initiated by Michael Moller and (mostly) written by Aylwin Lo with assistance from Michael Heal and Tom Slee. We work at Sybase.

Since the author is a co−op student, the best way to contact someone regarding this document is by posting to the sybase.public.sqlanywhere.linux newsgroup, available on the forums.sybase.com news server.

8.3. Acknowledgement

Thanks to the folks at http://www.commandprompt.com/ for getting the text of this HOWTO into workable SGML for us.