



# Adaptive Server® Anywhere Getting Started

Part number: 38122-01-0900-01

Last modified: June 2003

Copyright © 1989–2003 Sybase, Inc. Portions copyright © 2001–2003 iAnywhere Solutions, Inc. All rights reserved.

No part of this publication may be reproduced, transmitted, or translated in any form or by any means, electronic, mechanical, manual, optical, or otherwise, without the prior written permission of iAnywhere Solutions, Inc. iAnywhere Solutions, Inc. is a subsidiary of Sybase, Inc.

Sybase, SYBASE (logo), AccelaTrade, ADA Workbench, Adaptable Windowing Environment, Adaptive Component Architecture, Adaptive Server, Adaptive Server Anywhere, Adaptive Server Enterprise, Adaptive Server Enterprise Monitor, Adaptive Server Enterprise Replication, Adaptive Server Everywhere, Adaptive Server IQ, Adaptive Warehouse, AnswerBase, Anywhere Studio, Application Manager, AppModeler, APT Workbench, APT-Build, APT-Edit, APT-Execute, APT-Library, APT-Translator, ASEP, AvantGo, AvantGo Application Alerts, AvantGo Mobile Delivery, AvantGo Mobile Document Viewer, AvantGo Mobile Inspection, AvantGo Mobile Marketing Channel, AvantGo Mobile Pharma, AvantGo Mobile Sales, AvantGo Pylon, AvantGo Pylon Application Server, AvantGo Pylon Conduit, AvantGo Pylon PIM Server, AvantGo Pylon Pro, Backup Server, BayCam, Bit-Wise, BizTracker, Certified PowerBuilder Developer, Certified SYBASE Professional, Certified SYBASE Professional (logo), ClearConnect, Client Services, Client-Library, CodeBank, Column Design, ComponentPack, Connection Manager, Convoy/DM, Copernicus, CSP, Data Pipeline, Data Workbench, DataArchitect, Database Analyzer, DataExpress, DataServer, DataWindow, DB-Library, dbQueue, Developers Workbench, Direct Connect Anywhere, DirectConnect, Distribution Director, Dynamic Mobility Model, Dynamo, e-ADK, E-Anywhere, e-Biz Integrator, E-Whatever, EC Gateway, ECMAP, ECRT, eFulfillment Accelerator, Electronic Case Management, Embedded SQL, EMS, Enterprise Application Studio, Enterprise Client/Server, Enterprise Connect, Enterprise Data Studio, Enterprise Manager, Enterprise Portal (logo), Enterprise SQL Server Manager, Enterprise Work Architecture, Enterprise Work Designer, Enterprise Work Modeler, eProcurement Accelerator, eremote, Everything Works Better When Everything Works Together, EWA, Financial Fusion, Financial Fusion (and design), Financial Fusion Server, Formula One, Fusion Powered e-Finance, Fusion Powered Financial Destinations, Fusion Powered STP, Gateway Manager, GeoPoint, GlobalFIX, iAnywhere, iAnywhere Solutions, ImpactNow, Industry Warehouse Studio, InfoMaker, Information Anywhere, Information Everywhere, InformationConnect, InstaHelp, InternetBuilder, iremote, iScript, Jaguar CTS, jConnect for JDBC, KnowledgeBase, Logical Memory Manager, M-Business Channel, M-Business Network, M-Business Server, Mail Anywhere Studio, MainframeConnect, Maintenance Express, Manage Anywhere Studio, MAP, MDI Access Server, MDI Database Gateway, media.splash, Message Anywhere Server, MetaWorks, MethodSet, ML Query, MobicATS, My AvantGo, My AvantGo Media Channel, My AvantGo Mobile Marketing, MySupport, Net-Gateway, Net-Library, New Era of Networks, Next Generation Learning, Next Generation Learning Studio, O DEVICE, OASIS, OASIS (logo), ObjectConnect, ObjectCycle, OmniConnect, OmniSQL Access Module, OmniSQL Toolkit, Open Biz, Open Business Interchange, Open Client, Open Client/Server, Open Client/Server Interfaces, Open ClientConnect, Open Gateway, Open Server, Open ServerConnect, Open Solutions, Optima++, Partnerships that Work, PB-Gen, PC APT Execute, PC DB-Net, PC Net Library, PhysicalArchitect, Pocket PowerBuilder, PocketBuilder, Power Through Knowledge, Power++, power.stop, PowerAMC, PowerBuilder, PowerBuilder Foundation Class Library, PowerDesigner, PowerDimensions, PowerDynamo, Powering the New Economy, PowerJ, PowerScript, PowerSite, PowerSocket, Powersoft, Powersoft Portfolio, Powersoft Professional, PowerStage, PowerStudio, PowerTips, PowerWare Desktop, PowerWare Enterprise, ProcessAnalyst, QAnywhere, Rapport, Relational Beans, RepConnector, Replication Agent, Replication Driver, Replication Server, Replication Server Manager, Replication Toolkit, Report Workbench, Report-Execute, Resource Manager, RW-DisplayLib, RW-Library, S.W.I.F.T. Message Format Libraries, SAFE, SAFE/PRO, SDF, Secure SQL Server, Secure SQL Toolset, Security Guardian, SKILS, smart.partners, smart.parts, smart.script, SQL Advantage, SQL Anywhere, SQL Anywhere Studio, SQL Code Checker, SQL Debug, SQL Edit, SQL Edit/TPU, SQL Everywhere, SQL Modeler, SQL Remote, SQL Server, SQL Server Manager, SQL Server SNMP SubAgent, SQL Server/CFT, SQL Server/DBM, SQL SMART, SQL Station, SQL Toolset, SQLJ, Stage III Engineering, Startup.Com, STEP, SupportNow, Sybase Central, Sybase Client/Server Interfaces, Sybase Development Framework, Sybase Financial Server, Sybase Gateways, Sybase Learning Connection, Sybase MPP, Sybase SQL Desktop, Sybase SQL Lifecycle, Sybase SQL Workgroup, Sybase Synergy Program, Sybase User Workbench, Sybase Virtual Server Architecture, SybaseWare, Syber Financial, SyberAssist, SybMD, SyBooks, System 10, System 11, System XI (logo), SystemTools, Tabular Data Stream, The Enterprise Client/Server Company, The Extensible Software Platform, The Future Is Wide Open, The Learning Connection, The Model For Client/Server Solutions, The Online Information Center, The Power of One, TradeForce, Transact-SQL, Translation Toolkit, Turning Imagination Into Reality, UltraLite, UltraLite.NET, UNIBOM, Unilib, Uninull, Unisep, Unistring, URK Runtime Kit for UniCode, Versacore, Viewer, VisualWriter, VQL, Warehouse Control Center, Warehouse Studio, Warehouse WORKS, WarehouseArchitect, Watcom, Watcom SQL, Watcom SQL Server, Web Deployment Kit, Web.PB, Web.SQL, WebSights, WebViewer, WorkGroup SQL Server, XA-Library, XA-Server, and XP Server are trademarks of Sybase, Inc. or its subsidiaries.

All other trademarks are property of their respective owners.

# Contents

<b>About This Manual</b>	<b>vii</b>
SQL Anywhere Studio documentation . . . . .	viii
Documentation conventions . . . . .	xi
The Adaptive Server Anywhere sample database . . . . .	xiii
Finding out more and providing feedback . . . . .	xiv
<b>1 Adaptive Server Anywhere Quick Start</b>	<b>1</b>
Step 1: Start the Adaptive Server Anywhere database server . . . . .	2
Step 2: Start Sybase Central . . . . .	4
Step 3: Start Interactive SQL . . . . .	6
<b>2 Databases and Applications</b>	<b>9</b>
Relational database concepts . . . . .	10
SQL and database computing . . . . .	15
The pieces of a database system . . . . .	18
How the pieces fit together . . . . .	20
<b>3 Introduction to Adaptive Server Anywhere</b>	<b>23</b>
Introduction to Adaptive Server Anywhere . . . . .	24
Adaptive Server Anywhere intended uses . . . . .	25
Adaptive Server Anywhere hallmarks . . . . .	26
The Adaptive Server Anywhere database server . . . . .	28
Adaptive Server Anywhere applications . . . . .	29
<b>4 The Architecture of Database Applications</b>	<b>33</b>
Application programming interfaces . . . . .	34
Inside Adaptive Server Anywhere . . . . .	39
<b>5 Designing and Building Your Database</b>	<b>43</b>
Introduction . . . . .	44
The sample database . . . . .	46
Tutorial: Design and build a simple database . . . . .	51
<b>6 Connecting Your Application to its Database</b>	<b>61</b>
Introduction to connections . . . . .	62
Creating an ODBC data source . . . . .	63

<b>7</b>	<b>Using Interactive SQL</b>	<b>67</b>
	Introduction to Interactive SQL . . . . .	68
	Starting Interactive SQL . . . . .	70
	Using Interactive SQL to display data . . . . .	75
	Working with SQL statements in Interactive SQL . . . . .	80
<b>8</b>	<b>Selecting Data from Database Tables</b>	<b>87</b>
	Introduction . . . . .	88
	Selecting a complete table . . . . .	90
	Selecting columns from a table . . . . .	92
	Ordering query results . . . . .	95
	Selecting rows from a table . . . . .	98
<b>9</b>	<b>Selecting Data from Multiple Tables</b>	<b>105</b>
	Introduction . . . . .	106
	Joining tables using the cross product . . . . .	108
	Using the ON phrase to restrict a join . . . . .	109
	Joining tables using key joins . . . . .	111
	Joining tables using natural joins . . . . .	113
	Joining tables using outer joins . . . . .	115
<b>10</b>	<b>Selecting Aggregate Data</b>	<b>117</b>
	Summarizing data . . . . .	118
	A first look at aggregate functions . . . . .	119
	Applying aggregate functions to grouped data . . . . .	120
	Restricting groups . . . . .	122
<b>11</b>	<b>Selecting Data Using Subqueries</b>	<b>125</b>
	Introducing subqueries . . . . .	126
	Introduction . . . . .	127
	Single-row and multiple-row subqueries . . . . .	129
	Using subqueries instead of joins . . . . .	131
<b>12</b>	<b>Updating the Database</b>	<b>133</b>
	Introduction . . . . .	134
	Adding rows to a table . . . . .	135
	Modifying rows in a table . . . . .	136
	Deleting rows . . . . .	137
	Grouping changes into transactions . . . . .	138
	Integrity checking . . . . .	141

<b>13 System Tables</b>	<b>145</b>
The system tables . . . . .	146
The SYSCATALOG view . . . . .	147
The SYSCOLUMNS view . . . . .	148
Other system tables . . . . .	149
<b>14 Microsoft Visual Basic Quick Start</b>	<b>151</b>
Tutorial: Developing a Visual Basic application . . . . .	152
<b>15 Glossary</b>	<b>155</b>
<b>Index</b>	<b>173</b>



# About This Manual

Subject	This book describes how to build simple databases and database applications using Adaptive Server Anywhere.
Audience	This manual is for beginning users of Adaptive Server Anywhere.
Before you begin	This manual assumes some familiarity with basic programming concepts. It also assumes a working knowledge of the operating system on which you will be using Adaptive Server Anywhere.

# SQL Anywhere Studio documentation

## The SQL Anywhere Studio documentation

This book is part of the SQL Anywhere documentation set. This section describes the books in the documentation set and how you can use them.

The SQL Anywhere Studio documentation is available in a variety of forms: in an online form that combines all books in one large help file; as separate PDF files for each book; and as printed books that you can purchase. The documentation consists of the following books:

- ◆ **Introducing SQL Anywhere Studio** This book provides an overview of the SQL Anywhere Studio database management and synchronization technologies. It includes tutorials to introduce you to each of the pieces that make up SQL Anywhere Studio.
- ◆ **What's New in SQL Anywhere Studio** This book is for users of previous versions of the software. It lists new features in this and previous releases of the product and describes upgrade procedures.
- ◆ **Adaptive Server Anywhere Getting Started** This book is for people new to relational databases or new to Adaptive Server Anywhere. It provides a quick start to using the Adaptive Server Anywhere database-management system and introductory material on designing, building, and working with databases.
- ◆ **Adaptive Server Anywhere Database Administration Guide** This book covers material related to running, managing, and configuring databases and database servers.
- ◆ **Adaptive Server Anywhere SQL User's Guide** This book describes how to design and create databases; how to import, export, and modify data; how to retrieve data; and how to build stored procedures and triggers.
- ◆ **Adaptive Server Anywhere SQL Reference Manual** This book provides a complete reference for the SQL language used by Adaptive Server Anywhere. It also describes the Adaptive Server Anywhere system tables and procedures.
- ◆ **Adaptive Server Anywhere Programming Guide** This book describes how to build and deploy database applications using the C, C++, and Java programming languages. Users of tools such as Visual Basic and PowerBuilder can use the programming interfaces provided by those tools. It also describes the Adaptive Server Anywhere ADO.NET data provider.



- ◆ **Adaptive Server Anywhere Error Messages** This book provides a complete listing of Adaptive Server Anywhere error messages together with diagnostic information.
- ◆ **SQL Anywhere Studio Security Guide** This book provides information about security features in Adaptive Server Anywhere databases. Adaptive Server Anywhere 7.0 was awarded a TCSEC (Trusted Computer System Evaluation Criteria) C2 security rating from the U.S. Government. This book may be of interest to those who wish to run the current version of Adaptive Server Anywhere in a manner equivalent to the C2-certified environment.
- ◆ **MobiLink Synchronization User's Guide** This book describes how to use the MobiLink data synchronization system for mobile computing, which enables sharing of data between a single Oracle, Sybase, Microsoft or IBM database and many Adaptive Server Anywhere or UltraLite databases.
- ◆ **MobiLink Synchronization Reference** This book is a reference guide to MobiLink command line options, synchronization scripts, SQL statements, stored procedures, utilities, system tables, and error messages.
- ◆ **iAnywhere Solutions ODBC Drivers** This book describes how to set up ODBC drivers to access consolidated databases other than Adaptive Server Anywhere from the MobiLink synchronization server and from Adaptive Server Anywhere remote data access.
- ◆ **SQL Remote User's Guide** This book describes all aspects of the SQL Remote data replication system for mobile computing, which enables sharing of data between a single Adaptive Server Anywhere or Adaptive Server Enterprise database and many Adaptive Server Anywhere databases using an indirect link such as e-mail or file transfer.
- ◆ **SQL Anywhere Studio Help** This book includes the context-sensitive help for Sybase Central, Interactive SQL, and other graphical tools. It is not included in the printed documentation set.
- ◆ **UltraLite Database User's Guide** This book is intended for all UltraLite developers. It introduces the UltraLite database system and provides information common to all UltraLite programming interfaces.
- ◆ **UltraLite Interface Guides** A separate book is provided for each UltraLite programming interface. Some of these interfaces are provided as UltraLite components for rapid application development, and others are provided as static interfaces for C, C++, and Java development.

In addition to this documentation set, PowerDesigner and InfoMaker include their own online documentation.

Documentation formats      SQL Anywhere Studio provides documentation in the following formats:

- ◆ **Online documentation**    The online documentation contains the complete SQL Anywhere Studio documentation, including both the books and the context-sensitive help for SQL Anywhere tools. The online documentation is updated with each maintenance release of the product, and is the most complete and up-to-date source of documentation.

To access the online documentation on Windows operating systems, choose Start ► Programs ► SQL Anywhere 9 ► Online Books. You can navigate the online documentation using the HTML Help table of contents, index, and search facility in the left pane, as well as using the links and menus in the right pane.

To access the online documentation on UNIX operating systems, see the HTML documentation under your SQL Anywhere installation.

- ◆ **Printable books**    The SQL Anywhere books are provided as a set of PDF files, viewable with Adobe Acrobat Reader.

The PDF files are available on the CD ROM in the *pdf\_docs* directory. You can choose to install them when running the setup program.

- ◆ **Printed books**    The complete set of books is available from Sybase sales or from eShop, the Sybase online store. You can access eShop by clicking How to Buy ► eShop at <http://www.ianywhere.com>.

# Documentation conventions

This section lists the typographic and graphical conventions used in this documentation.

## Syntax conventions

The following conventions are used in the SQL syntax descriptions:

- ◆ **Keywords** All SQL keywords appear in upper case, like the words ALTER TABLE in the following example:

**ALTER TABLE** [ *owner*.]*table-name*

- ◆ **Placeholders** Items that must be replaced with appropriate identifiers or expressions are shown like the words *owner* and *table-name* in the following example:

**ALTER TABLE** [ *owner*.]*table-name*

- ◆ **Repeating items** Lists of repeating items are shown with an element of the list followed by an ellipsis (three dots), like *column-constraint* in the following example:

**ADD** *column-definition* [ *column-constraint*, ... ]

One or more list elements are allowed. In this example, if more than one is specified, they must be separated by commas.

- ◆ **Optional portions** Optional portions of a statement are enclosed by square brackets.

**RELEASE SAVEPOINT** [ *savepoint-name* ]

These square brackets indicate that the *savepoint-name* is optional. The square brackets should not be typed.

- ◆ **Options** When none or only one of a list of items can be chosen, vertical bars separate the items and the list is enclosed in square brackets.

[ **ASC** | **DESC** ]

For example, you can choose one of ASC, DESC, or neither. The square brackets should not be typed.

- ◆ **Alternatives** When precisely one of the options must be chosen, the alternatives are enclosed in curly braces and a bar is used to separate the options.

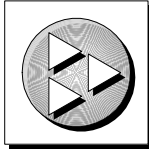
[ **QUOTES** { **ON** | **OFF** } ]

If the QUOTES option is used, one of ON or OFF must be provided. The brackets and braces should not be typed.

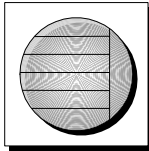
## Graphic icons

The following icons are used in this documentation.

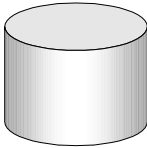
- ◆ A client application.



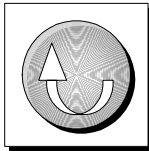
- ◆ A database server, such as Sybase Adaptive Server Anywhere.



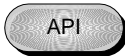
- ◆ A database. In some high-level diagrams, the icon may be used to represent both the database and the database server that manages it.



- ◆ Replication or synchronization middleware. These assist in sharing data among databases. Examples are the MobiLink Synchronization Server and the SQL Remote Message Agent.



- ◆ A programming interface.



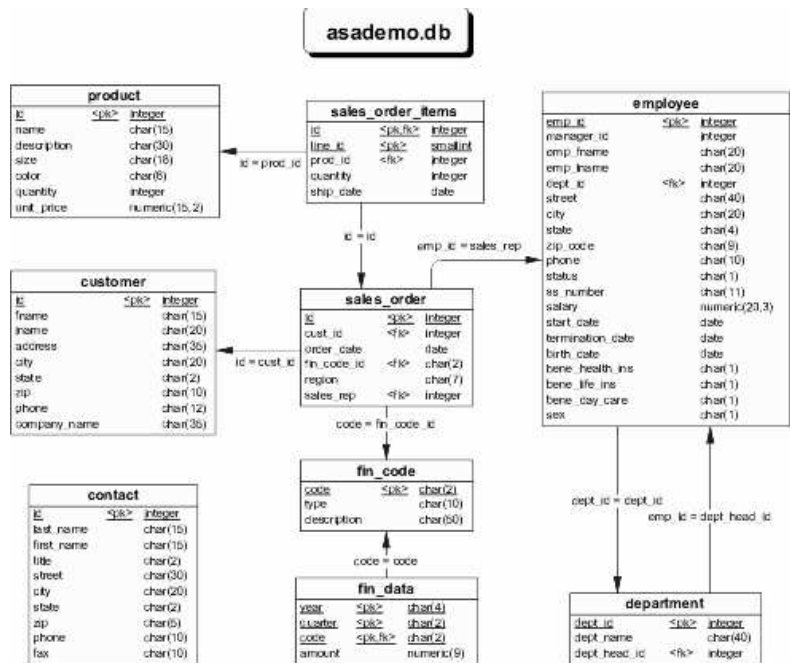
# The Adaptive Server Anywhere sample database

Many of the examples throughout the documentation use the Adaptive Server Anywhere sample database.

The sample database is held in a file named *asademo.db*, and is located in your SQL Anywhere directory.

The sample database represents a small company. It contains internal information about the company (employees, departments, and finances) as well as product information and sales information (sales orders, customers, and contacts).

The following figure shows the tables in the sample database and how they relate to each other.



## Finding out more and providing feedback

We would like to receive your opinions, suggestions, and feedback on this documentation.

You can provide feedback on this documentation and on the software through newsgroups set up to discuss SQL Anywhere technologies. These newsgroups can be found on the *forums.sybase.com* news server.

The newsgroups include the following:

- ◆ sybase.public.sqlanywhere.general.
- ◆ sybase.public.sqlanywhere.linux.
- ◆ sybase.public.sqlanywhere.mobilink.
- ◆ sybase.public.sqlanywhere.product\_futures\_discussion.
- ◆ sybase.public.sqlanywhere.replication.
- ◆ sybase.public.sqlanywhere.ultralite.

### **Newsgroup disclaimer**

iAnywhere Solutions has no obligation to provide solutions, information or ideas on its newsgroups, nor is iAnywhere Solutions obliged to provide anything other than a systems operator to monitor the service and insure its operation and availability.

iAnywhere Solutions Technical Advisors as well as other staff assist on the newsgroup service when they have time available. They offer their help on a volunteer basis and may not be available on a regular basis to provide solutions and information. Their ability to help is based on their workload.

## CHAPTER 1

# Adaptive Server Anywhere Quick Start

### About this chapter

This chapter describes how to start an Adaptive Server Anywhere database server and connect to the sample database from Sybase Central and Interactive SQL.

This chapter is for those who have some familiarity with databases and want to run the software right away.

☞ For introductory descriptions of databases and Adaptive Server Anywhere in particular, see the chapters starting with “[Databases and Applications](#)” on page 9.

### Contents

<b>Topic:</b>	<b>page</b>
<a href="#">Step 1: Start the Adaptive Server Anywhere database server</a>	<a href="#">2</a>
<a href="#">Step 2: Start Sybase Central</a>	<a href="#">4</a>
<a href="#">Step 3: Start Interactive SQL</a>	<a href="#">6</a>

---

## Step 1: Start the Adaptive Server Anywhere database server

In this section you start the Adaptive Server Anywhere database server running the sample database.

Adaptive Server Anywhere includes two versions of the database server. The personal database server is provided for single-user, same-machine use. The network database server supports client/server communication over a network and is provided for multi-user operation. The two database servers are exactly equivalent in their query processing and other internal operations: the personal database server is not less powerful than the network database server.

Adaptive Server Anywhere databases are held in operating system files. The sample database is held in a file named *asdemo.db* in your SQL Anywhere installation directory.

### ❖ To start the personal database server running the sample database (Windows)

1. From the Start menu, choose Programs ► SQL Anywhere 9 ► Adaptive Server Anywhere ► Personal Server Sample.

The database server window appears.

### ❖ To start the database server running the sample database (Command line)

1. Open a command prompt.
2. Change the directory to the SQL Anywhere installation directory.

On Windows operating systems, the default installation directory is *C:\Program Files\Sybase\SQL Anywhere 9*.

3. Start the database server running the sample database.

The way you start the database server depends on your operating system, and on whether you wish to connect to the database from other machines on the network.

- ◆ If you wish to connect only from the same machine on Windows or UNIX operating systems, enter the following command to start the personal database server:

```
dbeng9 -n asdemo9 asdemo.db
```



- ◆ If you wish to connect to the database server from other machines on the network on Windows or UNIX operating systems, enter the following command to start the network database server:

```
dbsrv9 -n asademo9 asademo.db
```

- ◆ On NetWare, only the network database server is provided. Enter the following command:

```
load dbsrv9.nlm -n asademo9 asademo.db
```

The database server window appears.

☞ For a complete list of options you can use when starting the database server, see “The database server” [*ASA Database Administration Guide*, page 124].

---

## Step 2: Start Sybase Central

In this section you start Sybase Central, the graphical database administration tool. These instructions assume that you have carried out the instructions in [“Step 1: Start the Adaptive Server Anywhere database server”](#) on page 2.

### ❖ To start Sybase Central and connect to the sample database (Windows)

1. Choose Start ► Programs ► SQL Anywhere 9 ► Sybase Central.
2. Connect to the sample database.
  - ◆ Choose Tools ► Connect.
  - ◆ If you are prompted to choose a plug-in, choose Adaptive Server Anywhere 9 from the list of plug-ins and then click OK.
  - ◆ On the Database tab, enter the Server name **asademo9**. This identifies the database server you started in the previous section.
  - ◆ On the Identification tab, enter the user ID **DBA** and the password **SQL**.
  - ◆ Click OK to connect.

### ❖ To start Sybase Central and connect to the sample database (Command line)

1. At a command prompt, type the following command:

```
scjview
```

The main Sybase Central window appears.

2. Connect to the sample database.
  - ◆ Choose Tools ► Connect.
  - ◆ If you are prompted to choose a plug-in, choose Adaptive Server Anywhere 9 from the list of plug-ins and then click OK.
  - ◆ On the Database tab, enter the Server name **asademo9**. This identifies the database server you started in the previous section.
  - ◆ On the Identification tab, enter the user ID **DBA** and the password **SQL**.
  - ◆ Click OK to connect.

You can now explore the tables and other objects in the sample database.

With Sybase Central you can carry out many database administration tasks, including creating databases, backing up databases, creating tables and other database objects, and modifying data in database tables.

☞ For a tour of Sybase Central, see “Tutorial: Managing Databases with Sybase Central” [*Introducing SQL Anywhere Studio*, page 47].

---

## Step 3: Start Interactive SQL

In this section you start the Interactive SQL utility, connect to the sample database, and enter a command.

### ❖ To start Interactive SQL and connect to the sample database (Windows)

1. Start Interactive SQL:
  - ◆ Choose Start ► Programs ► SQL Anywhere 9 ► Adaptive Server Anywhere ► Interactive SQL.  
The Connect dialog appears.
2. Connect to the sample database.
  - ◆ On the Database tab, enter the Server name **asademo9**.
  - ◆ On the Identification tab, enter the user ID **DBA** and the password **SQL**.
  - ◆ Click OK to connect.  
The Interactive SQL window appears.
3. Enter a command.

In the SQL Statements pane, enter a SQL statement such as the following and press F5 to execute the query:

```
SELECT * FROM CUSTOMER
```

The result set for the query appears in the Results pane.

### ❖ To start Interactive SQL and connect to the sample database (Command line)

1. From a command prompt, enter the following command:

```
dbisql
```

The Connect dialog appears.
2. Connect to the sample database.
  - ◆ On the Database tab, enter the Server name **asademo9**.
  - ◆ On the Identification tab, enter the user ID **DBA** and the password **SQL**.
  - ◆ Click OK to connect.
3. Enter a command in the Interactive SQL window.

In the SQL Statements pane, enter a SQL statement such as the following and press F5 to execute the query:

```
SELECT * FROM CUSTOMER
```

The result set for the query appears in the Results pane.

☞ You can enter any SQL statement against the database from within Interactive SQL. For a complete list of SQL statements, see “SQL Statements” [*ASA SQL Reference*, page 213].

☞ For more information about Interactive SQL, see [“Using Interactive SQL” on page 67](#).



## CHAPTER 2

# Databases and Applications

### About this chapter

This chapter introduces basic database concepts. It describes what a relational database is, and what it is used for. It also describes how databases and database applications work together.

Other parts of the documentation set assume that you are familiar with the concepts introduced in this chapter.

### Contents

<b>Topic:</b>	<b>page</b>
<a href="#">Relational database concepts</a>	10
<a href="#">SQL and database computing</a>	15
<a href="#">The pieces of a database system</a>	18
<a href="#">How the pieces fit together</a>	20

---

# Relational database concepts

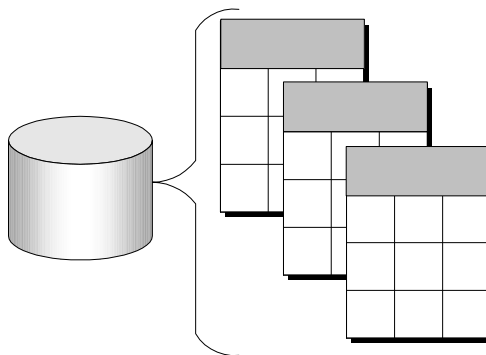
A **relational database-management system** (RDBMS) is a system for storing and retrieving data, in which the data is organized in tables. A relational database consists of a collection of tables that store interrelated data.

This section introduces some of the terms and concepts that are important in talking about relational databases.

☞ For a tutorial illustrating the concepts described here, see [“Designing and Building Your Database” on page 43](#).

## Database tables

In a relational database, all data is held in **tables**, which are made up of **rows** and **columns**.



Each table has one or more columns, and each column is assigned a specific data type, such as an integer, a sequence of characters (for text), or a date. Each row in the table has a single value for each column.

For example, a table containing employee information may look as follows:

emp_ID	emp_lname	emp_fname	emp_phone
10057	Huong	Zhang	1096
10693	Donaldson	Anne	7821

Characteristics of  
relational tables

The tables of a relational database have some important characteristics:

- ◆ There is no significance to the order of the columns or rows.
- ◆ Each row contains one and only one value for each column, or contains NULL, which indicates that there is no value for that column.



- ◆ All values for a given column have the same data type.

The following table lists some of the formal and informal relational database terms describing tables and their contents, together with their equivalent in non-relational databases. This manual uses the informal terms.

Informal relational term	Formal relational term	Non-relational term
Table	Relation	File
Column	Attribute	Field
Row	Tuple	Record

What do you keep in each table?

Each table in the database should hold information about a specific thing, such as employees, products, or customers.

By designing a database this way, you can set up a structure that eliminates redundancy and the possible inconsistencies caused by redundancy. For example, both the sales and accounts payable departments might enter and look up information about customers. In a relational database, the information about customers is stored only once, in a table that both departments can access.

☞ For more information about database design, see “Designing Your Database” [ASA SQL User’s Guide, page 3].

☞ For instructions on how to create a table, see [“Lesson 3: Design and create a table” on page 53](#).

## Relations between tables

A relational database is a set of related tables. You use primary keys and foreign keys to describe relationships between the information in different tables. **Primary keys** identify each row in a table uniquely, and **foreign keys** define the relationships between rows in different tables.

Primary keys and foreign keys let you use relational databases to hold information in an efficient manner, without redundancy.

### Tables have a primary key

Each table in a relational database should have a **primary key**. The primary key is a column, or set of columns, that uniquely identifies each row. No two rows in a table may have the same primary key value.

Examples

In the sample database, the employee table stores information about employees. It has a primary key column named emp\_id, which holds a

---

unique ID number assigned to each employee. A single column holding an ID number is a common way to assign primary keys, and has advantages over names and other identifiers that may not always be unique.

A more complex primary key can be seen in the `sales_order_items` table of the sample database. The table holds information about individual items on orders from the company, and has the following columns:

- ◆ **id** An order number, identifying the order the item is part of.
- ◆ **line\_id** A line number, identifying each item on any order.
- ◆ **prod\_id** A product ID, identifying the product being ordered.
- ◆ **quantity** A quantity, displaying how many items were ordered.
- ◆ **ship\_date** A ship date, displaying when the order was shipped.

A particular sales order item is identified by the order it is part of, and by a line number on the order. These two numbers are stored in the `id` and `line_id` columns. Items may share a single `id` value (corresponding to an order for more than one item) or they may share a `line_id` number (all first items on different orders have a `line_id` of 1). No two items share both values, and so the primary key is made up of these two columns.

☞ For a tutorial example, see [“Lesson 4: Identify and create primary keys” on page 55](#).

## Tables are related by foreign keys

### Example

The information in one table is related to that in other tables by **foreign keys**.

The sample database has one table holding employee information and one table holding department information. The department table has the following columns:

- ◆ **dept\_id** An ID number for the department. This is the primary key for the table.
- ◆ **dept\_name** The name of the department.
- ◆ **dept\_head\_id** The employee ID for the department manager.

To find the name of a particular employee’s department, there is no need to put the name of the employee’s department into the employee table. Instead, the employee table contains a column holding a number that matches one of the `dept_id` values in the department column.

The `dept_id` column in the employee table is called a foreign key to the department table. A foreign key references a particular row in the table containing the corresponding primary key.

In this example, the employee table (which contains the foreign key in the relationship) is called the **foreign table** or **referencing table**. The department table (which contains the referenced primary key) is called the **primary table** or the **referenced table**.

☞ For a tutorial example, see [“Lesson 5: Design column properties”](#) on page 56.

## Other database objects

A relational database holds more than a set of related tables. Among the other objects that make up a relational database are the following:

- ◆ **Indexes** Indexes allow quick lookup of information. Conceptually, an index in a database is like an index in a book. In a book, the index relates each indexed term to the page or pages on which that word appears. In a database, the index relates each indexed column value to the physical location at which the row of data containing the indexed value is stored.

Indexes are an important design element for high performance. You must usually create indexes explicitly, but indexes for primary and foreign keys and for unique columns are created automatically. Once created, the use of indexes is transparent to the user.

- ◆ **Views** Views are computed tables, or virtual tables. They look like tables to client applications, but they do not hold data. Instead, whenever they are accessed, the information in them is computed from the underlying tables.

The tables that actually hold the information are sometimes called **base tables** to distinguish them from views. A view is defined with a SQL query on base tables or other views.

- ◆ **Stored procedures and triggers** These are routines held in the database itself that act on the information in the database.

You can create and name your own stored procedures to execute specific database queries and to perform other database tasks. Stored procedures can take parameters. For example, you might create a stored procedure that returns the names of all customers who have spent more than the amount that you specify as a parameter in the call to the procedure.

A trigger is a special stored procedure that automatically fires whenever a user updates, deletes, or inserts data, depending on how you define the trigger. You associate a trigger with a table or columns within a table. Triggers are useful for automatically maintaining business rules in a database.

- 
- ◆ **Users and groups** Each user of a database has a user ID and password. You can set permissions for each user so that confidential information is kept private and users are prevented from making unauthorized changes. Users can be assigned to groups in order to make the administration of permissions easier.
  - ◆ **Java objects** You can install Java classes into the database. Java classes provide a powerful way of building logic into your database, and a special class of user-defined data types for holding information.

## SQL and database computing

When a client application wants to carry out a database task, such as retrieving information using a query or inserting a new row into a table, it does so using **Structured Query Language (SQL)** statements. SQL is a relational database language that has been standardized by the ANSI and ISO standards bodies.

Depending on how you develop a client application, SQL statements could be supplied in function calls from the programming language. Some application development tools provide user interfaces for building and generating SQL statements.

The programming interface delivers the SQL statement to the database server. The database server receives the statement and executes it, returning the required information (such as query results) back to the application.

Client/server communication protocols carry information between the client application and the database server, and programming interfaces define how an application sends the information. No matter what interface you use, and what network protocol you use, it is SQL statements that are sent to a server, and the results of SQL statements that are returned to the client application.

### Example

This SQL statement extracts the last names of all employees from the employee table in the sample database:

```
SELECT emp_lname
FROM employee
```

You can send queries like this to the database server using Interactive SQL or you can build the query into your own application.

### Example

This SQL statement creates a table called food that lists types of food and the amount in stock in the employee cafeteria:

```
CREATE TABLE food (
    food_id integer primary key,
    food_type char(20),
    quantity integer
)
```

☞ For an introduction to SQL, see the chapters beginning with [“Selecting Data from Database Tables” on page 87](#).

## Queries

The “Q” in “SQL” stands for **query**. You query (or **retrieve**) data from a database with the SELECT statement. The basic query operations in a relational system are projection, restriction, and join. The SELECT

---

## Projections and restrictions

statement implements all of them.

A **projection** is a subset of the columns in a table. A **restriction** (also called **selection**) is a subset of the rows in a table, based on some conditions.

For example, the following SELECT statement retrieves the names and prices of all products that cost more than \$15:

```
SELECT name, unit_price
FROM product
WHERE unit_price > 15
```

This query uses both a projection (SELECT name, unit\_price) and a restriction (WHERE unit\_price > 15).

☞ For more information, see [“Selecting rows from a table” on page 98](#).

## Joins

A join links the rows in two or more tables by comparing the values in columns of each table. For example, you might want to select the order item identification numbers and product names for all order items that shipped more than a dozen pieces of merchandise:

```
SELECT sales_order_items.id, product.name
FROM product JOIN sales_order_items
WHERE sales_order_items.quantity > 12
```

The product table and the sales\_order\_items table are joined together based on the foreign key relationship between them.

☞ For more information, see the chapter [“Selecting Data from Multiple Tables” on page 105](#).

## Other SQL statements

You can do more with SQL than just query. SQL includes statements that create tables, views, and other database objects. It also includes statements that modify tables (the INSERT, UPDATE, and DELETE statements), and statements that perform many other database tasks discussed in this manual.

## The system tables

Every Adaptive Server Anywhere database contains a set of system tables. These are special tables that the system uses to manage data and the system. These tables are also sometimes called the **data dictionary** or the **system catalog**.

System tables contain information about the database. You never alter the system tables directly in the way that you can alter other tables. The system tables hold information about the tables in a database, the users of a

database, the columns in each table, and so on. This information is data about data, or **metadata**.

☞ For more information about the Adaptive Server Anywhere system tables, see “System Tables” [*ASA SQL Reference*, page 611].

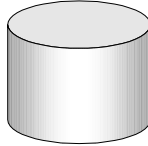
---

# The pieces of a database system

This section describes how database applications and the database server work together to manage databases.

Any information system contains the following pieces:

- ◆ **A database** Data is stored in a database. In diagrams in the documentation, a database is indicated by a cylinder:



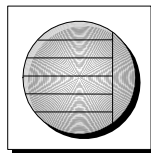
An Adaptive Server Anywhere database is a file, usually with an extension of *.db*. Adaptive Server Anywhere includes a sample database for you to work with: this is the file *asdemo.db* in your Adaptive Server Anywhere installation directory.

☞ For instructions on how to create a database, see [“Designing and Building Your Database” on page 43](#).

- ◆ **A database server** The database server manages the database. No other application addresses the database file directly; they all communicate with the database server.

☞ For instructions on starting database servers, see [“Connecting Your Application to its Database” on page 61](#).

In diagrams in the documentation, a database server is indicated as follows:



Adaptive Server Anywhere provides two versions of its database server: the **personal database server** and the **network database server**. In addition to the features of the personal server, the network database server supports client/server communications across a network, while the personal database server can accept connections only from applications running on the same machine. The request-processing engine is identical in both servers.

☞ For more information, see [“The Adaptive Server Anywhere database server” on page 28](#).



- ◆ **A programming interface** Applications communicate with the database server using a programming interface. You can use ODBC, JDBC, OLE DB, Sybase Open Client, or Embedded SQL.

Many application development tools provide their own programming environment that hides the details of the underlying interface. For example, if you develop an application using Sybase PowerBuilder, you never have to make an ODBC function call. Nevertheless, behind the scenes each of these tools is using one of the programming interfaces.

The programming interface provides a library of function calls for communicating with the database. For ODBC and JDBC, the library is commonly called a **driver**. The library is typically provided as a shared library on UNIX operating systems or a dynamic link library (DLL) on PC operating systems. The JDBC interface uses the Sybase jConnect driver, which is a zip file of compiled Java classes.

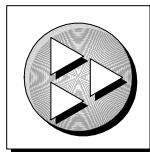
In diagrams in the documentation, a programming interface is indicated as follows:



- ◆ **A client application** Client applications use one of the programming interfaces to communicate with the database server.

If you develop an application using a rapid application development (RAD) tool such as Sybase PowerBuilder, you may find that the tool provides its own methods for communicating with database servers, and hides the details of the language interface. Nevertheless, all applications do use one of the supported interfaces.

In diagrams in the documentation, a client application is indicated by the following icon:



---

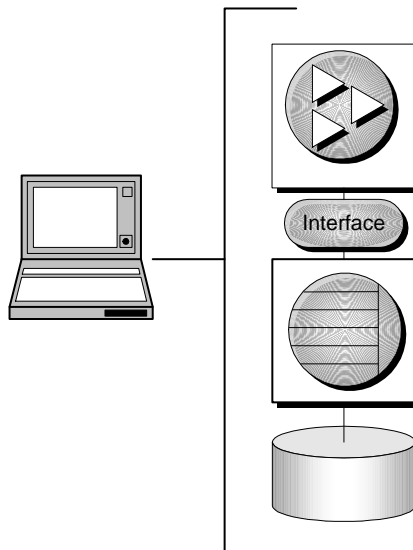
## How the pieces fit together

Database applications can connect to a database server located on the same machine as the application itself, or in the case of the network database server, on a different machine. In addition, with Adaptive Server Anywhere you can build distributed databases, with physically distinct databases on different machines sharing data.

### Personal applications and embedded databases

You can use Adaptive Server Anywhere to build a complete application and database on a single computer. In the simplest arrangement, this is a **standalone application** or **personal application**: it is self-contained with no connection to other databases. In this case, the database server and the database can be started by the client application, and it is common to refer to the database as an **embedded database**: as far as the end user is concerned, the database is a part of the application.

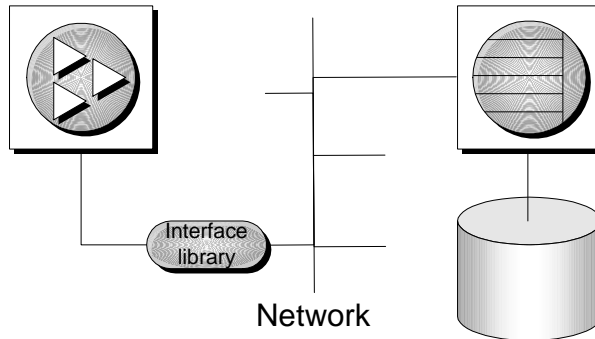
Standalone applications have the following architecture, with a client application connecting through a programming interface to a database server running on the same machine:



The Adaptive Server Anywhere personal database server is generally used for standalone applications, although you can also use applications on the same machine as the network server.

## Client/server applications and multi-user databases

You can use Adaptive Server Anywhere to build an installation with many applications running on different machines, connected over a network to a single database server running on a separate machine. This is a **client/server** environment, and has the following architecture. The interface library is located on each client machine.



In this case, the database server is the Adaptive Server Anywhere network database server, which supports network communications. The database is also called a **multi-user database**.

No changes are needed to a client application for it to work in a client/server environment, except to identify the server to which it should connect.



## CHAPTER 3

# Introduction to Adaptive Server Anywhere

About this chapter

This chapter introduces the Adaptive Server Anywhere relational database system. It describes the uses and features of Adaptive Server Anywhere.

Contents

<b>Topic:</b>	<b>page</b>
<a href="#">Introduction to Adaptive Server Anywhere</a>	24
<a href="#">Adaptive Server Anywhere intended uses</a>	25
<a href="#">Adaptive Server Anywhere hallmarks</a>	26
<a href="#">The Adaptive Server Anywhere database server</a>	28
<a href="#">Adaptive Server Anywhere applications</a>	29

---

# Introduction to Adaptive Server Anywhere

The previous chapter, “[Databases and Applications](#)” on page 9, introduced databases in a general manner. This chapter describes the particular hallmarks of Adaptive Server Anywhere and describes the uses to which it can be put.

Adaptive Server Anywhere provides a series of tools for storing and managing data. You can use these tools to enter data into your database, to change your database structure, and to view or alter your data.

The Adaptive Server Anywhere relational database-management system is the heart of SQL Anywhere Studio. Adaptive Server Anywhere is designed for tasks that require a full-featured SQL database. It is designed to operate in varied environments. It takes advantage of available memory and CPU resources, providing good performance in environments with ample resources. It also operates well in environments with limited physical and database administration resources, including mobile computing environments, embedded database use, and use as a database server for small and medium businesses.

## Adaptive Server Anywhere intended uses

Roles for which Adaptive Server Anywhere is ideally suited include the following:

- ◆ **A small and medium business database server** Adaptive Server Anywhere is built to handle the requirements of small and medium businesses, with anywhere from a few users to several hundred users. It provides a high-performance database for workgroups and companies, well-suited for (but not limited to) environments where administration and hardware resources are limited.

Adaptive Server Anywhere can employ multiple CPUs and use up to 64 Gb of memory. Our customers have Adaptive Server Anywhere databases with tens of gigabytes of data in production use.

- ◆ **An embedded database** Many applications require a database “behind the scenes”. Personal Information Managers, document management systems, network monitoring applications—just about any application that stores information. Adaptive Server Anywhere is intended to be the database for these applications. The UltraLite deployment option is intended for embedded environments that have very restricted resources.

A key feature of embedded databases is that they be able to run entirely without administration. Adaptive Server Anywhere has demonstrated this facility in many demanding commercial applications.

- ◆ **Mobile computing** Laptop and notebook computers are now common in many workplaces. Adaptive Server Anywhere is intended to be the SQL database for these computers. With MobiLink synchronization and SQL Remote replication, Adaptive Server Anywhere extends transaction-based computing throughout the enterprise. The UltraLite deployment option and MobiLink synchronization technology provide full database functionality on devices with limited resources, such as hand-held computers.

---

## Adaptive Server Anywhere hallmarks

Adaptive Server Anywhere is built around the following technological hallmarks:

- ◆ **Full SQL relational database-management system** Adaptive Server Anywhere is a transaction-processing relational database-management system (RDBMS), with full recovery capabilities, online backup, referential integrity actions, stored procedures, triggers, row-level concurrency control, schedules and events, a rich SQL language, and all the features you expect in a full SQL RDBMS.
- ◆ **Economical hardware requirements** Adaptive Server Anywhere requires fewer memory and disk resources than other database-management systems.
- ◆ **Easy to use** Adaptive Server Anywhere is self-tuning and easy to manage. You can use Adaptive Server Anywhere without the extensive database administration efforts usually associated with relational database-management systems.
- ◆ **Standalone and network use** Adaptive Server Anywhere can be used in a standalone manner, for example as an embedded database in a data-centric application, or as a network server in a multi-user client/server or three-tier environment. As an embedded database system, it can be started automatically by an application when required.
- ◆ **High performance** While Adaptive Server Anywhere is designed with simple administration and modest resource requirements in mind, it is a scalable, high-performance DBMS. Adaptive Server Anywhere can run on multiple CPUs, has an advanced query optimizer, and provides performance monitoring and tuning tools.
- ◆ **Industry standard interfaces** Adaptive Server Anywhere provides a native ODBC 3.5 driver for high performance from ODBC applications, and an OLE DB driver for use from ActiveX Data Object (ADO) programming environments. It has an ADO.NET data provider for Adaptive Server Anywhere and it also comes with Sybase jConnect for JDBC, as well as an iAnywhere JDBC driver, and supports embedded SQL and Sybase Open Client interfaces.
- ◆ **A cross-platform solution** Adaptive Server Anywhere can be run on many operating systems, including Windows, Novell NetWare, Sun Solaris, and Linux.

☞ The components available on each platform may differ. For information, see [“Availability of components” on page 30](#).



System requirements      ➞ For more information on supported operating systems for components in SQL Anywhere Studio, see “SQL Anywhere Studio Supported Platforms” [*Introducing SQL Anywhere Studio*, page 121].

## Network software requirements

If you are running an Adaptive Server Anywhere network server, you must have appropriate networking software installed and running.

The Adaptive Server Anywhere network server is available for Windows, Novell NetWare, Linux, and UNIX operating systems.

Adaptive Server Anywhere supports the TCP/IP network protocol and the SPX protocol for Novell NetWare.

---

# The Adaptive Server Anywhere database server

There are two versions of the Adaptive Server Anywhere database server included in the product:

- ◆ **The personal database server** This server is provided for single-user, same-machine use: for example, as an embedded database server. It is also useful for development work.

The name of the personal server executable is as follows:

- On UNIX operating systems, it is *dbeng9*.
- On Windows, except Windows CE, it is *dbeng9.exe*.

- ◆ **The network database server** In addition to the features of the personal server, the network server supports client/server communications across a network. It is provided for multi-user use.

The name of the network server executable is as follows:

- On UNIX operating systems, it is *dbsrv9*.
- On Windows, including Windows CE, it is *dbsrv9.exe*.
- On Novell NetWare, the server is a NetWare Loadable Module (NLM) called *dbsrv9.nlm*.

Same SQL features in each version

The request-processing engine is identical in the two versions of the server. They support exactly the same SQL language, and exactly the same database features. The personal server does not support communications across a network, more than ten concurrent connections, or the use of more than two CPUs. Applications developed against a personal server work unchanged against a network server.

## Adaptive Server Anywhere applications

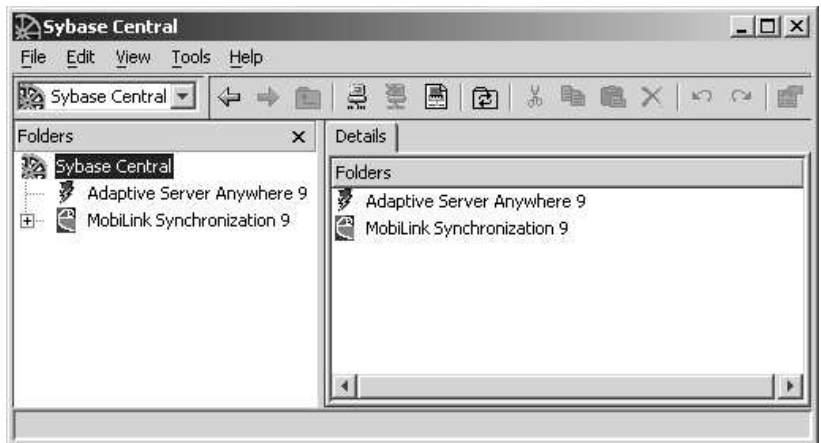
This section introduces some of the database applications that are supplied with Adaptive Server Anywhere in the SQL Anywhere Studio product. These applications help you design, build, and administer your databases.

### Sybase Central

Sybase Central provides a graphical user interface for creating and modifying databases and database objects, for inspecting the structure of databases, and for database server administration.

Sybase Central lets you perform such tasks as creating a new database, adding a table, or adding a column to a table.

☞ For information on using Sybase Central, see [“Lesson 2: Connect to your database” on page 52](#).



The Sybase Central window is similar to Windows Explorer. The main window is split into two vertically-aligned panes. The left pane displays a hierarchical view of database objects or **containers** in a tree-like structure. A container is a database object that can hold other database objects, including other containers.

The right pane displays the contents of the container that has been selected in the left pane. For example, to view a list of tables in the database, expand the Adaptive Server Anywhere 9 plug-in icon in the left pane, and then open the Tables folder in the left pane. A list of tables in the database appears on the Tables tab in the right pane.

☞ For an introduction to Sybase Central, see [“Tutorial: Managing Databases with Sybase Central” \[Introducing SQL Anywhere Studio, page 47\]](#).

---

## Interactive SQL

Interactive SQL is an application for typing and sending SQL statements to a database. Interactive SQL allows you to query and alter data in your database, as well as modify the structure of your database.

Everything that can be done in Sybase Central can be done in Interactive SQL, but administration tasks are easier in Sybase Central. For this reason, this book uses Sybase Central whenever possible, but when a task is important, or is simple to accomplish in Interactive SQL, the Interactive SQL instructions are also included.

☞ For an introduction to Interactive SQL, see [“Using Interactive SQL” on page 67](#).

The Interactive SQL panes

- ◆ **The SQL Statements pane** This is where you type SQL statements to access and modify your data. The title bar above the SQL Statements pane displays the current connection information.
- ◆ **The Results pane** This is a tabbed pane that displays query result sets, messages from the database server, and information about query execution. For example, if you enter a query asking how many customers ordered five or more different types of products, that number appears on the Results tab in the Results pane, and the query optimizer’s plan for executing the statement appears on the Plan tab in the Results pane. You can edit the result set on the Results tab.

☞ For more information about editing the result set, see [“Editing table values in Interactive SQL” on page 76](#).

## Utilities

A set of utilities is available for carrying out administration tasks such as backing up a database. Utilities are useful for including in batch files for repeated use.

☞ For more information about administration utilities, see [“Database Administration Utilities” \[ASA Database Administration Guide, page 455\]](#).

## Availability of components

What components you have installed depends on which operating system you are using, what choices you made when installing the software, and whether you received the Adaptive Server Anywhere product or installed Adaptive Server Anywhere as part of another product.

For example, if you have received Adaptive Server Anywhere as part of another product, you may not have both versions of the database server.

Not all components are available on all operating systems. For example, there is no personal server for NetWare, only a network server.

## The SQL Anywhere program group

For Windows operating systems, after the software is installed, you will have a SQL Anywhere **program group**. You can reach the program group by clicking the Start button and choosing Programs ► SQL Anywhere 9.

Installing SQL Anywhere Studio under UNIX does not provide a program group.

### Program group items

The program group contains some or all of the following items. The items you see depend on the choices you made when installing the software.

- ◆ **Adaptive Server Anywhere** Contains the items:
  - **Interactive SQL** Starts the Interactive SQL utility for sending SQL statements to a database.
  - **Network Server Sample** Starts the network database server and loads the sample database.
  - **ODBC Administrator** Starts the ODBC Administrator program for setting up and editing ODBC data sources.
  - **Personal Server Sample** Starts the personal server and loads the sample database.
- ◆ **MobiLink program group** To access MobiLink synchronization programs and samples.
- ◆ **UltraLite program group** To access UltraLite programs and samples.
- ◆ **Check for updates** To access a web page with information about the latest updates to SQL Anywhere Studio.
- ◆ **iAnywhere Online Resources** Opens a web page with information about iAnywhere Solutions.
- ◆ **Sybase Central** Starts Sybase Central, the database management tool.
- ◆ **Check for updates** Opens a web page where you get information about updates available for your version of SQL Anywhere Studio.
- ◆ **Online Books** Opens the online documentation for Adaptive Server Anywhere.

In addition, you may have items for InfoMaker and PowerDesigner.



## CHAPTER 4

# The Architecture of Database Applications

### About this chapter

This chapter describes how database applications communicate with the Adaptive Server Anywhere database server, and provides a glimpse into the architecture of Adaptive Server Anywhere.

### Contents

<b>Topic:</b>	<b>page</b>
<a href="#">Application programming interfaces</a>	<a href="#">34</a>
<a href="#">Inside Adaptive Server Anywhere</a>	<a href="#">39</a>

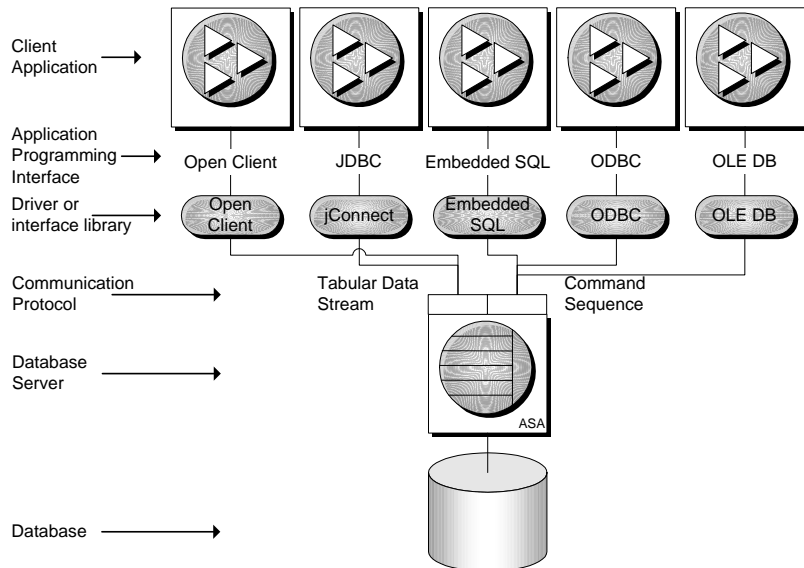
# Application programming interfaces

The current section describes application architecture in more detail. An overview of database application architecture was given in [“How the pieces fit together” on page 20](#).

Adaptive Server Anywhere supports a wide set of programming interfaces to provide flexibility in the kinds of applications and application development environments you can use.

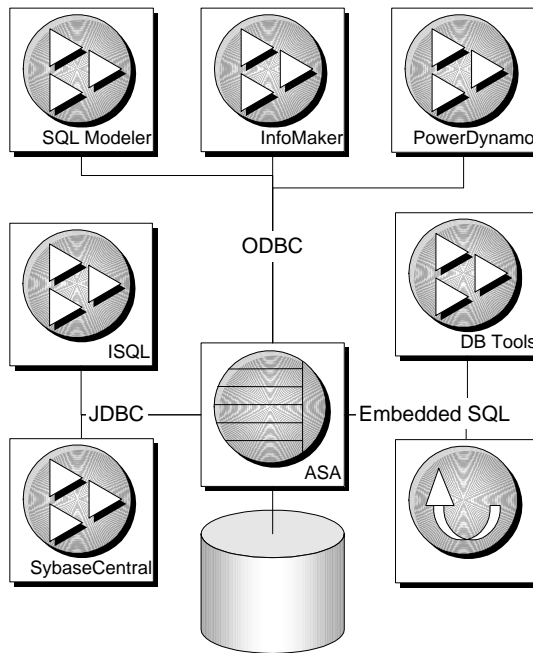
## Supported interfaces and protocols

The following diagram displays the supported interfaces, and the interface libraries used. In most cases, the interface library is the same name as the interface itself.



The applications supplied with SQL Anywhere Studio use several of these interfaces:





## Communications protocols

Each interface library communicates with the database server using a **communication protocol**. Adaptive Server Anywhere supports two communication protocols, **Tabular Data Stream (TDS)** and **Command Sequence**. These protocols are internal, and for most purposes it does not matter which one you are using. Your choice of development environment will be governed by your available tools.

The major differences you will see are upon connecting to the database. The Command Sequence applications and TDS applications use different methods to identify a database and database server, and so connection dialogs are different.

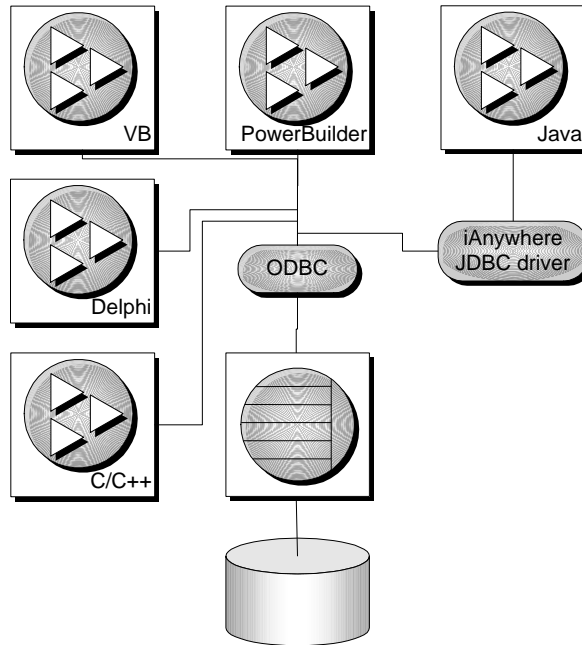
**TDS** This protocol is used by Sybase Adaptive Server Enterprise, Open Client applications, and Java applications that use the jConnect JDBC driver connect using TDS.

**Command Sequence** This protocol is used by Adaptive Server Anywhere, Adaptive Server IQ, and is used by embedded SQL, ODBC, and OLE DB applications.

---

## ODBC applications

You can develop ODBC applications using a variety of development tools and programming languages, as shown in the figure below.

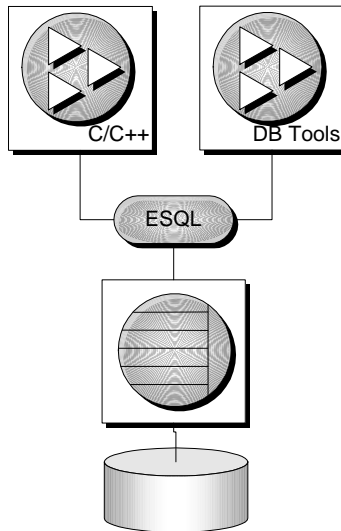


For example, of the applications supplied with SQL Anywhere Studio, InfoMaker and SQL Modeler use ODBC to connect to the database.

## Embedded SQL applications

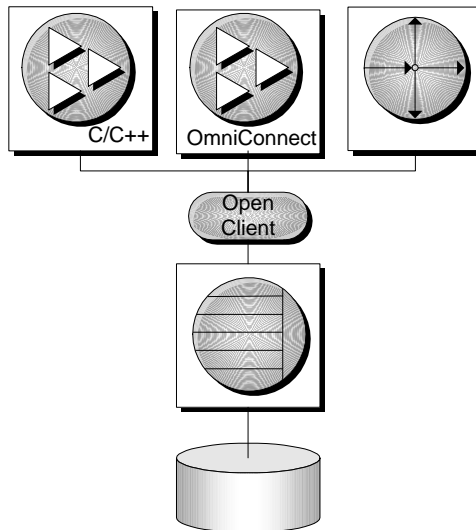
You can develop C or C++ applications using the embedded SQL programming interface. The command-line database tools are examples of applications developed in this manner.

Embedded SQL is also a programming interface for UltraLite applications.



## Open Client applications

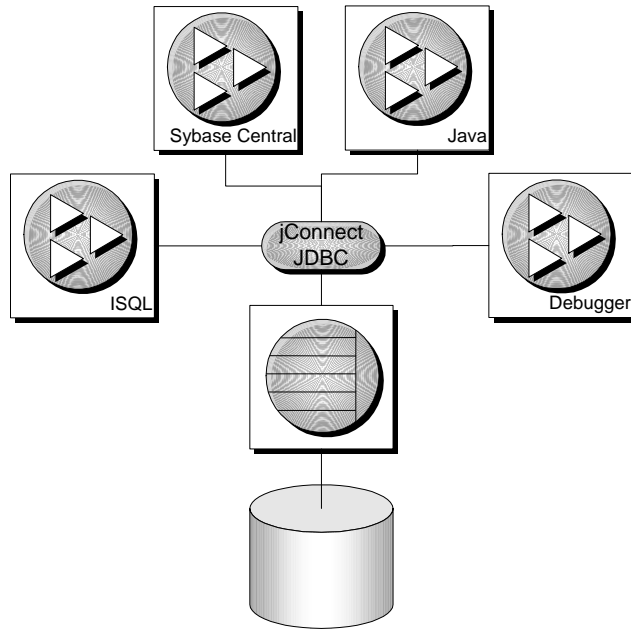
Open Client is an interface that is also supported by Sybase Adaptive Server Enterprise. You can develop Open Client applications in C or C++. Other Sybase applications, such as OmniConnect or Replication Server, use Open Client.



---

## JDBC applications

You can develop applications that use JDBC to connect to Adaptive Server Anywhere using Java. Several of the applications supplied with SQL Anywhere Studio use JDBC: the stored procedure debugger, Sybase Central, and Interactive SQL.



Java and JDBC are also important programming languages for developing UltraLite applications.

## OLE DB applications

Adaptive Server Anywhere includes an OLE DB driver. You can develop applications using Microsoft's OLE DB interface directly, or using the ActiveX Data Objects (ADO) interface. The ADO interface is included in Visual Basic and other Microsoft programming tools. SQL Anywhere Studio also includes an ADO.NET data provider for Adaptive Server Anywhere.

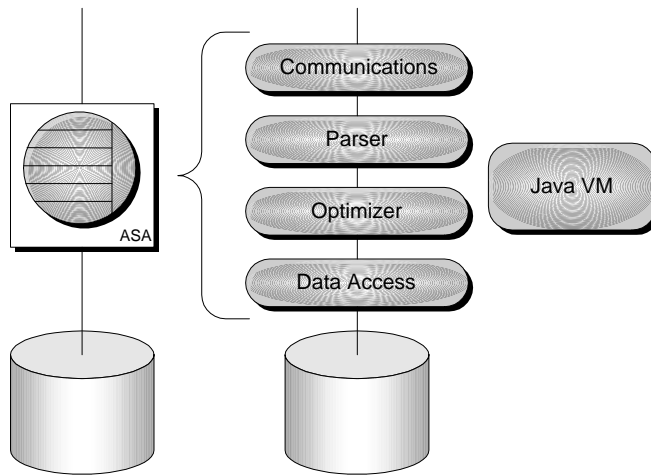
## Inside Adaptive Server Anywhere

While you never need to deal with the internals of the database server, a glimpse behind the scenes may help you understand the process better.

### Inside the database server

The Adaptive Server Anywhere database server has an internal structure that allows many requests to be handled efficiently.

- ◆ A communications layer handles the actual exchange of data with client applications. This layer receives requests from client applications, and returns results. The timing of these actions is governed by a negotiation between client and server to make sure that the network traffic is kept to a minimum, but that the data is made available as soon as possible on the client side.
- ◆ The parser checks each SQL statement sent to the database server, and transforms it into an internal form for processing.
- ◆ If the request is a query, an update, or delete statement, there may be many different ways of accessing the data, which may take massively different times. The job of the optimizer is to select from among all these possibilities the best route to getting the required data quickly.
- ◆ A Java Virtual Machine is built into the database server, and any Java operations sent by client applications, or invoked internally by the database server, are handled by the Java VM.
- ◆ The lowest level of the database server is concerned with reading and writing data from the disk, caching data in memory to avoid unnecessary disk access, and balancing the demands of different users.



## Inside the database

All the information in an Adaptive Server Anywhere database is usually stored in a single database file, which can be copied from one machine to another. It is possible to make databases of several files, but this is generally only required for very large databases.

In addition to the database file, Adaptive Server Anywhere uses two other files when it is running a database. These are the transaction log and the temporary file.

- ◆ **The database file** Internally, the database file is composed of pages: fixed size areas of disk. The data access layer reads and writes data one page at a time. Many pages hold the data that is in the database tables, but other pages hold index information, information about the distribution of data within the database, and so on.
- ◆ **The transaction log** The transaction log is a separate file that contains a record of all the operations performed on the database. Normally, it has the same name as the database file, except that it ends with the suffix *.log* instead of *.db*. It has three important functions.
  - **Record operations on your data to enable recovery** You can recreate your database from a backup together with the transaction log if the database file is damaged.
  - **Improve performance** By writing information to the transaction log, the database server can safely process your statements without writing to the database file as frequently.
  - **Enable database replication** SQL Remote and the MobiLink client

utility use this file to replicate the changes to your database on portable computers which are occasionally connected to the network.

- ◆ **The temporary file** The temporary file is opened when the database server starts, and is closed down when the server stops. As its name suggests, the temporary file is used while the server is running to hold temporary information. The temporary file does not hold information that needs to be kept between sessions.

The temporary file is stored in your temporary directory. The location of this directory is generally identified by your TEMP environment variable.





## CHAPTER 5

# Designing and Building Your Database

### About this chapter

This chapter introduces some principles of database design, and describes how to create a database using Sybase Central. It uses the Adaptive Server Anywhere sample database to illustrate the principles involved.

### Contents

<b>Topic:</b>	<b>page</b>
<a href="#">Introduction</a>	44
<a href="#">The sample database</a>	46
<a href="#">Tutorial: Design and build a simple database</a>	51

---

# Introduction

It is worth spending some time designing even the simplest database: what tables you will have, the keys that relate these tables, and so on.

Designing a database is not a difficult task for small and medium sized databases, but it is an important one. Bad database design can lead to an inefficient and possibly unreliable database system. Database applications are built to work on specific parts of a database, and rely on the database design, so a bad design can be difficult to revise at a later date.

Designing a large database is a complex task. There are formal approaches, such as conceptual data modeling, that help you to create efficient designs. Powerful tools, such as Sybase PowerDesigner and DataArchitect, enable you to apply these techniques to design and maintain large database designs.

This chapter does not attempt to tackle design issues for large databases. Instead, it helps you decide the kind of information you group together in a single table, and the way in which to think about and classify relationships between tables.

☞ For an elementary look at the principles of database design, see “Designing Your Database” [ASA *SQL User’s Guide*, page 3]. For more advanced treatments, see the Sybase PowerDesigner documentation or a book devoted to database design.

This chapter assumes that you are familiar with the concepts of database tables, primary keys, and foreign keys.

☞ For information on primary keys and foreign keys, see [“Relations between tables” on page 11](#).

## About this chapter

If you want to know...	Then see...
Where to find information about the sample database	<a href="#">“The sample database” on page 46</a>
How to connect to the sample database	<a href="#">“Connect to the sample database” on page 47</a>
How to view tables	<a href="#">“View a list of tables in the sample database” on page 48</a>
How to view columns	<a href="#">“View the columns of a table” on page 49</a>

If you want to know...	Then see...
How to create a database file	<a href="#">“Lesson 1: Create a database file” on page 51</a>
How to connect to a database	<a href="#">“Lesson 2: Connect to your database” on page 52</a>
How to create a table	<a href="#">“Lesson 3: Design and create a table” on page 53</a>
How to create a primary key	<a href="#">“Lesson 4: Identify and create primary keys” on page 55</a>
How to set column properties	<a href="#">“Lesson 5: Design column properties” on page 56</a>
How to create relationships between tables	<a href="#">“Lesson 6: Design and create relationships between tables” on page 58</a>

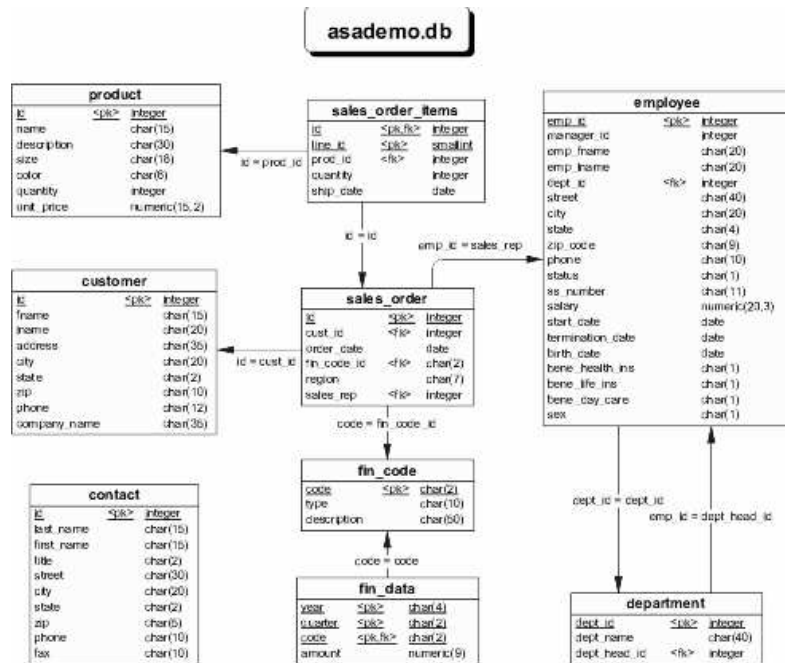
## The sample database

There is a sample database included with Adaptive Server Anywhere. Many of the examples throughout the documentation use this sample database.

The sample database represents a small company that makes a limited range of sports clothing. It contains internal information about the company (employees, departments, and financial data), as well as product information (products) and sales information (sales orders, customers, and contacts). All information in the sample database is fictional.

The following figure displays the tables in the sample database and how they are related to each other. The boxes represent tables, and the arrows represent foreign key relationships. Primary key columns are underlined.

We will use this database to illustrate the concepts described in this chapter.



## Viewing the structure of the sample database using Sybase Central

This section describes how to use Sybase Central to view the contents of the sample database.

## Connect to the sample database

You need to connect to the sample database from Sybase Central in order to view the tables and other objects in the database.

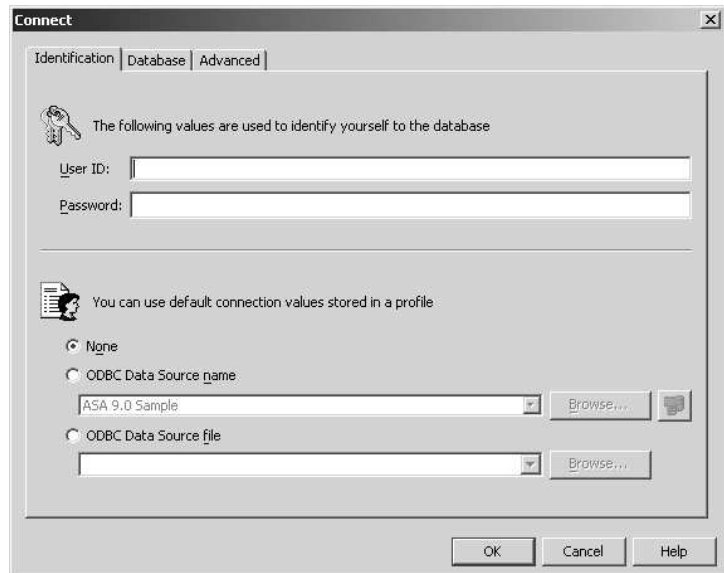
### ❖ To connect to the sample database from Sybase Central

1. Start Sybase Central.
  - ◆ From the Start menu, choose Programs ► SQL Anywhere 9 ► Sybase Central.
  - or
  - ◆ At a command prompt, enter the following command:

```
scjview
```

2. Open the Connect dialog.
  - ◆ Choose Tools ► Connect.
  - ◆ If you are presented with a list of plug-ins, choose Adaptive Server Anywhere from the list.

The Connect dialog appears:




3. Select ODBC Data Source Name and click Browse. From the resulting list, choose ASA 9.0 Sample.
4. Click OK to connect.

---

## View a list of tables in the sample database

Once you are connected to the sample database, you can open the folders in the left pane to view the tables and other objects that make up the database.

 For information on connecting to a database from Sybase Central, see [“Connect to the sample database” on page 47](#).

### ❖ To view a list of tables in the database

1. In Sybase Central, open the Adaptive Server Anywhere 9 plug-in icon.

Each top-level folder on the left pane is a **plug-in**. A separate plug-in is used to manage each software component. The Adaptive Server Anywhere plug-in is used to manage all aspects of Adaptive Server Anywhere.

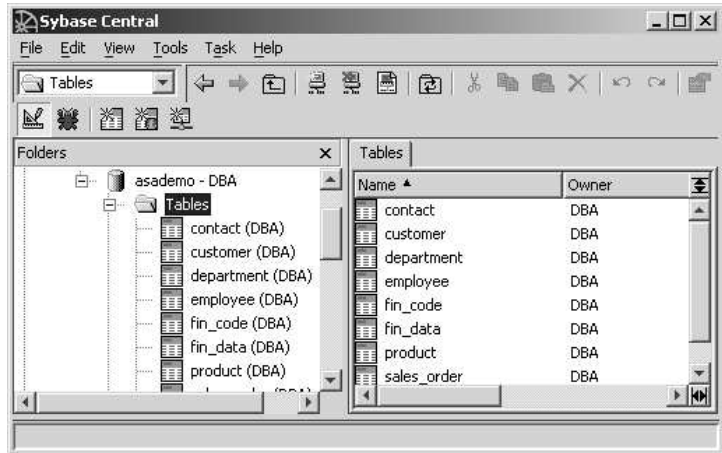
2. Open the sample database.

Under the Adaptive Server Anywhere plug-in is a list of database servers to which you have connections. When the sample database is started using the ASA 9.0 Sample ODBC data source, the database server is named **asademo9**.

Each database server may be running one or more databases. In this case, the sample database is named asademo. The database also has the user name for the current connection next to it. In this case, you are connected as a user named **DBA**.

3. Open the Tables folder.

Each database contains a set of folders describing the different types of database objects. Here, we are interested only in the tables in the database. Open the Tables folder to list the available tables.



### Notes

Each table has an **owner**, who is listed in parentheses beside the table name in the left pane. If you see more tables than appear in the figure, right-click the asademo database, and choose Filter Objects By Owner from the popup menu. Select DBA, and clear the other user IDs in the list. Click OK to restrict the objects displayed to those owned by DBA.

### View the columns of a table

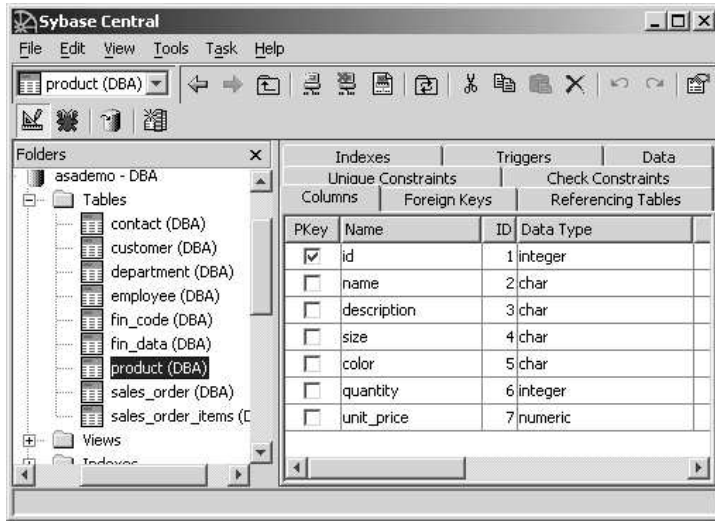
Each table has tabs in the right pane with information about a table's columns, foreign keys, indexes, triggers, and data. Here, we look only at the columns of the product table in the sample database.

☞ For information on viewing tables, see [“View a list of tables in the sample database” on page 48](#).

❖ **To view information about the columns of the product table**

1. In Sybase Central, open the product table. In the left pane, expand the Tables folder, and then select the product table in the left pane.

Information about the product table appears on the tabs in the right pane.



2. In the right pane, click the Columns tab.

The name, data type, and other information about each column appears in the right pane.

3. View the properties of the color column.

Select the color column, and then choose Properties from the File menu.

The color property sheet appears. This property sheet contains a detailed description of the column. Some of the properties are useful for advanced users only, and are not described in this book.

You can use Sybase Central to browse other aspects of the sample database, including foreign keys, primary keys, and the data in each table. You are encouraged to do so in order to be familiar with the sample database for examples in the remainder of the documentation.



## Tutorial: Design and build a simple database

When designing a database, you plan what items you want to store information about, and what information you will keep about each item. You also determine how these items are related. You classify the things in your database as **entities**; the links between these entities are called **relationships**.

In this tutorial, you design and build a very simple database, modeled on the product, sales\_order\_items, sales\_order, and customer tables of the sample database, but simplified. The database you create is not used in other parts of the documentation, but it is still highly recommended that you work through the tutorial to gain familiarity with the software and concepts.

### Lesson 1: Create a database file

In this lesson, you create a database file to hold your database.

☞ For more information, see [“The pieces of a database system” on page 18](#).

#### Concepts

A database file is a container, ready to hold your database. The database file contains system tables and other system objects that are common to all databases, but you must add tables and the data they hold.

The collection of tables, indexes, and so on within the database, and all the relationships between them, make up the database **schema**. The schema is the database without the data. This tutorial describes how to design and create a very simple database schema.

☞ For a conceptual introduction to some of these pieces, see [“Relational database concepts” on page 10](#).

The name of each object within the database, including tables, columns, and indexes, is an **identifier**. There are rules governing what you can use as identifiers. You can use any set of letters, numbers, or symbols. However, you must enclose a column name in double quotes if it contains characters other than letters, numbers, or underscores, if it does not begin with a letter, or if it is the same as a keyword.

If the QUOTED\_IDENTIFIER database option is set to OFF, double quotes are used to delimit SQL strings and cannot be used for identifiers. However, you can always use square brackets to delimit identifiers, regardless of the setting of QUOTED\_IDENTIFIER.

☞ For more information about identifiers, see [“Identifiers” \[ASA SQL Reference, page 7\]](#).

#### Exercise

---

### ❖ To create a new database file

1. Start Sybase Central.
2. In the left pane, select the Adaptive Server Anywhere 9 plug-in, then click the Utilities tab in the right pane. In the right pane, double-click Create Database.

The Create Database wizard opens.

3. Read the information on the introductory page and click Next.
4. Select Create A Database On This Computer and click Next.
5. Choose a location and name for your database file:
  - ◆ Enter the filename `c:\temp\mysample`. If your temporary directory is somewhere other than `c:\temp`, specify a path of your own choice.
6. Click Finish to create the database.

Other options are available when creating a database, which you could have viewed by clicking Next instead of Finish, but the default choices are good for many purposes.

The Creating Database window displays the progress of the task. When the file is created, click OK to close the window.

## Lesson 2: Connect to your database

In this lesson, you connect to the database file you created.

☞ For more information, see [“How the pieces fit together” on page 20](#).

### Exercise

Once your database is created, you can connect to it in order to create tables and other database objects.

### ❖ To connect to your database

1. Start Sybase Central.
2. Choose Tools ► Connect to open the Connect dialog. If you have multiple plug-ins loaded, you may need to choose the Adaptive Server Anywhere plug-in before the Connect dialog opens.
3. Specify the user ID and password.

On the first tab of the Connect dialog (the Identification tab), enter a user ID of **DBA** and a password of **SQL**. These are the values created for new databases, and so grant access to your new database.

Choose **None** in the profile options at the bottom of the tab.

## 4. Specify your database file.

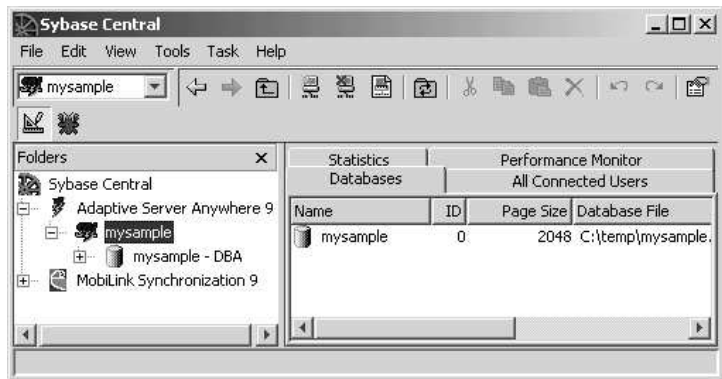
Click the Database tab. Enter the full path of your database file in the Database File field. For example, if you followed the suggestion in the previous lesson, you should enter the following:

```
c:\temp\mysample.db
```

## 5. Connect to the database.

Click OK. Sybase Central connects to the database.

Open the database server container in the left pane to see the mysample database.



## Lesson 3: Design and create a table

In this lesson, you add a table to your database.

### Concepts

Each table in your database should contain information about a single subject. In the language of database design, you are identifying **entities**. For example, the sample database holds information about employees in one table, and information about products in another table: employees and products are entities within the database design.

☞ For an introduction to tables, see [“Database tables” on page 10](#).

Each column in a table describes a particular characteristic of the thing that you would like to store information about. For example, in the sample database, the employee table has columns that hold an employee ID number, first and last names, an address, and other particular information that pertains to a particular employee.

In database diagrams, each table is depicted by a rectangular box. The name of the table is at the top of the box, and the column names are listed inside the box.

product	
id	integer
name	char(15)
description	char(30)
size	char(18)
color	char(6)
quantity	integer
unit_price	numeric(15,2)

In the product table from the sample database, above, each product is an item of sports clothing.

## Exercise

Create a simplified version of the product table, containing only the identifier (id) and name columns.

### ❖ To create the product table

1. In Sybase Central, connect to your database if you are not already connected.
2. Open the Tables folder in your database.  
First open the server and database containers, then open the Tables folder.
3. From the File menu, choose ► New ► Table.  
The Table Creation wizard appears.
4. Name your table **product**.
5. Click Finish.
6. Select the product table in the left pane, then click the Columns tab in the right pane.
7. Create the columns.

From the File menu, choose Add column and add a column with the following properties:

- ◆ **Column Name** Give the column a name of **id**.
- ◆ **PKey** Select the checkbox beside the column name so that a checkmark appears, indicating that the column is a **primary key**.
- ◆ **Data Type** Give the column the **integer** data type.

You can ignore the other properties. Add a second column with the following properties:

- ◆ **Column Name** Give the column a name of **name**. This column holds the product name.
- ◆ **Data Type** Give the column the **char** data type which holds character strings, and enter a maximum length of **15** in the Size column.

8. To finish, click the Save and Close icon on the far right of the toolbar.

You have now created a table in your database. The table data is held in the database file. At present, the table is empty.

The next two lessons have more to say about columns and data types.

## Lesson 4: Identify and create primary keys

In this lesson, you learn more about defining primary keys for your tables. There is no exercise associated with this lesson.

For more information, see [“Tables have a primary key” on page 11](#).

### Concepts

The **primary key** is a special column or columns used to uniquely identify a row in a table. In the product table, the id column uniquely identifies each product.

product	
<u>id</u>	integer
name	char(15)
description	char(30)
size	char(18)
color	char(6)
quantity	integer
unit_price	numeric(15,2)

Each row has a unique value for the id column, and the values in each row pertain only to a single product identifier by its id value. Two products might have the same name or the same size, but not the same id number. In the diagram, the id column is underlined to show that it is a primary key.

Creating a column specifically to hold an identifier which has no other meaning is common practice in database design. You will know from your bank, utility, or credit card statements that each account has a unique identifier.

### Using an AUTOINCREMENT primary key

You can make entering primary keys simple by assigning a primary key column a default value of AUTOINCREMENT. The value for this column is entered automatically each time a new row is added, and its value is one more than the field's value for the last row added.

#### ❖ To create an AUTOINCREMENT primary key

1. Select the product table in the left pane and then click the Columns tab in the right pane.
2. Select the primary key column. From the File menu, choose Properties button to open the property sheet for the column.
3. Click the Value tab.
4. Select the Default Value option.

- 
5. Click System-defined, and choose Autoincrement from the dropdown list.
  6. Click OK to close the column property sheet.
  7. To finish, choose File ► Save Table.

## Lesson 5: Design column properties

In this lesson, you learn more about choosing data types and other attributes for the columns of your tables.

### Concepts

Each column has a data type associated with it. The **data type** defines the type of information the column holds. Choose a data type for the column that is appropriate for the data in the column. For example, identifier columns commonly have an integer data type, while columns holding names or addresses must have character data types.

Data types are organized into the following categories:

- ◆ **Numeric data types** There are several numeric data types. Some are exact (not affected by round-off errors during operations) and some are approximate.

The data type of the column affects the maximum size of the column. For example, if you specify SMALLINT, a column can contain a maximum value of 32,767. If you specify INTEGER, the maximum value is 2,147,483,647.

☞ For a complete list, see “Numeric data types” [ASA SQL Reference, page 56].

- ◆ **Character data types** These are used to hold strings of text, such as names, addresses, and so on. These data types have a length indicating the maximum length of string that can be stored in them.

☞ For a list, see “Character data types” [ASA SQL Reference, page 52].

- ◆ **Binary data types** These can be useful to hold information that may be meaningful to an application, but is encoded in a binary format.

☞ For a list, see “Binary data types” [ASA SQL Reference, page 72].

- ◆ **Date/time data types** These hold times of the day, as well as dates.

☞ For a list, see “Date and time data types” [ASA SQL Reference, page 65].

- ◆ **Long data types** These are sometimes called blobs (binary large objects). They can be used to hold long strings of text (called memo fields in some databases), images, or other binary information.

☞ For more information, see “LONG BINARY data type [BINARY]” [ASA SQL Reference, page 72], and “LONG VARCHAR data type [Character]” [ASA SQL Reference, page 54].

In addition, Adaptive Server Anywhere supports user-defined data types and special Java data types. These are not discussed in this introductory book.

#### NULL and NOT NULL

If every row must contain a value for this column, you should define the column as being NOT NULL. Otherwise, the column is allowed to contain NULL, which represents a missing value. The default is to allow NULL, but you should explicitly declare columns NOT NULL unless there is a good reason to allow NULL.

☞ For a complete description of the NULL value, see “NULL value” [ASA SQL Reference, page 48]. For information on its use in comparisons, see “Search conditions” [ASA SQL Reference, page 22].

#### ❖ To specify a data type for a column

1. Select the product table in the left pane, then click the Columns tab in the right pane.
2. Select the primary key column and then choose File ► Properties to open the property sheet for a column.  
The column’s property sheet opens.
3. Click the Data Type tab.
4. Select a data type from the Built-in Type dropdown list.

#### Exercise

This lesson and the last lesson have introduced the basic concepts you need to know in order to create database tables. You can put these to work by adding some more tables to your database. These tables will be used in the subsequent lessons in this chapter.

Add the following tables to your database:

- ◆ **customer** Add a table named customer, with the following columns:
  - **id** An identification number for each customer. This column has **integer** data type, and is the **primary key**. Make this an autoincrement key.
  - **company\_name** The company name. This column is a **character** data type, with a maximum length of **35** characters.
- ◆ **sales\_order** Add a table named sales\_order, with the following columns:

- **id** An identification number for each sales order. This column has **integer** data type, and is the **primary key**. Make this an autoincrement key.
- **order\_date** The date on which the order was placed. This column has **date** data type.
- **cust\_id** The identification number of the customer who placed the sales order. This column has **integer** data type.
- ◆ **sales\_order\_items** Add a table named `sales_order_items` to hold line item information, with the following columns:
  - **id** The identification number of the sales order of which the line item is a part. This column has **integer** data type, and should be identified as a **primary key** column.
  - **line\_id** An identification number for each sales order. This column has **integer** data type, and should be identified as a **primary key** column.
  - **prod\_id** The identification number for the product being ordered. This column has **integer** data type.

You have now created four tables in your database. The tables are not yet related in any way. In the next lesson, you define foreign keys to relate the tables to one another.

## Lesson 6: Design and create relationships between tables

In this lesson, you learn about designing and creating relationships between tables, using foreign keys.

☞ For more information, see [“Tables are related by foreign keys” on page 12](#).

### Concepts

Although each table contains information about a single subject, two or more tables may contain related information. For example, an employee is a member of a department, or a sales order is for a set of products. Relationships in a database may appear as **foreign key** relationships between tables, or may appear as separate tables themselves. You will see examples of each in this chapter.

You create relationships in your database to encode rules or practices that govern the data in your tables. Once a relationship is built into the structure of the database, there is no provision for exceptions.

Relationships among tables are classified as follows.

- ◆ **One-to-one relationships** Each item in one entity corresponds to either zero or one entity in another. For example, in the sample database, *one*



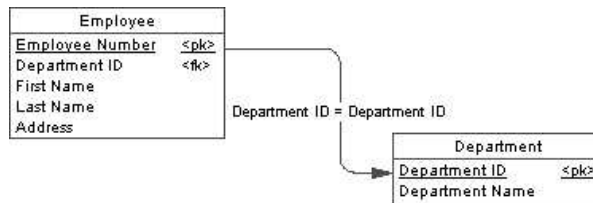
employee manages *one* department. There is nowhere to put a second department manager. Duplicating the department entry would involve duplicating the department ID, which is not possible because it is the primary key.

It is often appropriate to combine the items in a one-to-one relationship into a single table. There is a column in the department table for a manager, rather than having a separate table named manager.

☞ For cases where it is appropriate to keep the items separate, see “Designing Your Database” [ASA SQL User’s Guide, page 3].

- ◆ **Many-to-one relationships** A many-to-one relationship becomes a foreign key relationship between tables. In a many-to-one relationship, the primary key in the *one* entity appears as a new foreign key column in the *many* table.

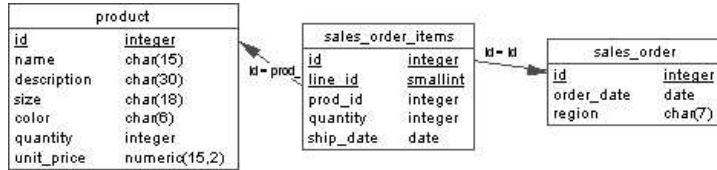
For example, in the database you just created, *one* customer can place *many* orders, but only one customer places each order. To represent the one-to-many relationship, you need a **foreign key column** in the sales\_order table (cust\_id) that maps to the primary key column in the customer table (id). It is often convenient to give the two columns the same name.



Each entry in the cust\_id column of the sales\_order table must match one of the entries in the id column of the customer table. The sales\_order table (which contains the foreign key in the relationship) is called the foreign table or referencing table. The customer table (which contains the referenced primary key) is called the primary table or the referenced table.

- ◆ **Many to many relationships** A many-to-many relationship is represented by an intermediate table, and there is a foreign key relationship from the intermediate table to each of the related entities.

For example, in the sample database, there is a many-to-many relationship between products and sales orders. One sales order can be for many products, and one product can appear on many sales orders.



In some cases, the intermediate table (sales\_order\_items) contains additional information, such as the number of items of the product that were ordered and the date they were shipped. In this case, the intermediate table holds no additional information.

## Exercise

Add foreign keys to relate the tables in your database.

Add the following foreign keys:

- ◆ A foreign key from the id column in sales\_order\_items, referencing the id column in sales\_order. This key builds the many-to-one relationship between sales orders and sales order items into the database.
- ◆ A foreign key from the prod\_id column in sales\_order\_items, referencing the id column in product. This key builds the many-to-one relationship between sales order items and products into the database.
- ◆ A foreign key from the cust\_id column in sales\_order, referencing the id column in customer. This key builds the many-to-one relationship between sales orders and customers into the database.

The first two foreign keys taken together build the many-to-many relationship between sales orders and products into the database.

## ❖ To create a foreign key

1. Select the table for which you wish to create a foreign key.
2. Click the Foreign Keys tab in the right pane.
3. From the File menu, choose New ► Foreign Key to open the Foreign Key Creation wizard.
4. Follow the instructions in the wizard.

This completes this introductory section on designing and building relational databases. Remaining chapters in the book describe how to add and retrieve data from databases. These chapters use the Adaptive Server Anywhere sample database, which is a bigger database than the one you have just created.

## CHAPTER 6

# Connecting Your Application to its Database

About this chapter

This chapter shows you how to establish a connection from your application to the database it is working with.

Contents

<b>Topic:</b>	<b>page</b>
<a href="#">Introduction to connections</a>	<a href="#">62</a>
<a href="#">Creating an ODBC data source</a>	<a href="#">63</a>

---

## Introduction to connections

Any client application that uses a database must establish a connection to that database before any work can be done. The connection forms a channel through which all activity from the client application takes place. For example, your user ID determines permissions to carry out actions on the database—and the database server has your user ID because it is part of the request to establish a connection.

Many client applications, including application development systems, use the **Open Database Connectivity (ODBC)** interface to access Adaptive Server Anywhere. An ODBC data source is a set of connection parameters that are stored in the registry or in a file.

You can use ODBC data sources to connect to Adaptive Server Anywhere databases from any of the following applications:

- ◆ Sybase Central and Interactive SQL.
- ◆ All the Adaptive Server Anywhere utilities.
- ◆ PowerDesigner and InfoMaker.
- ◆ Any application development environment that supports ODBC, such as Microsoft Visual Basic, Sybase PowerBuilder, and Borland Delphi.

Adaptive Server Anywhere client applications on UNIX can use ODBC data sources. On UNIX, the data source is stored as a file.

☞ Adaptive Server Anywhere supports several programming interfaces in addition to ODBC. For more information, see “Introduction to connections” [*ASA Database Administration Guide*, page 38].

## Creating an ODBC data source

ODBC data sources are a convenient way of saving connection parameters for repeated use. Once you have a data source, your connection string can simply name the data source to use:

```
DSN=my data source
```

The Connect dialog in Sybase Central and Interactive SQL has fields for entering an ODBC Data Source Name or ODBC Data Source File.

This section describes how to create a simple ODBC data source in Windows.

☞ For more detailed information, see “Configuring ODBC data sources using the ODBC Administrator” [*ASA Database Administration Guide*, page 56].

☞ For information about setting up ODBC data sources on UNIX, see “Using ODBC data sources on UNIX” [*ASA Database Administration Guide*, page 63].

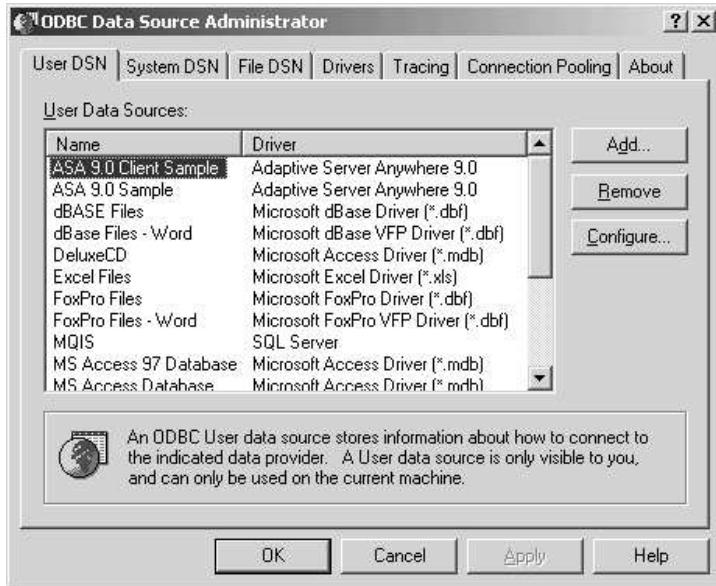
### ❖ To create a simple ODBC data source

1. Start the ODBC Administrator:
  - ◆ From the Windows Start menu, choose Programs ► SQL Anywhere 9 ► Adaptive Server Anywhere ► ODBC Administrator.

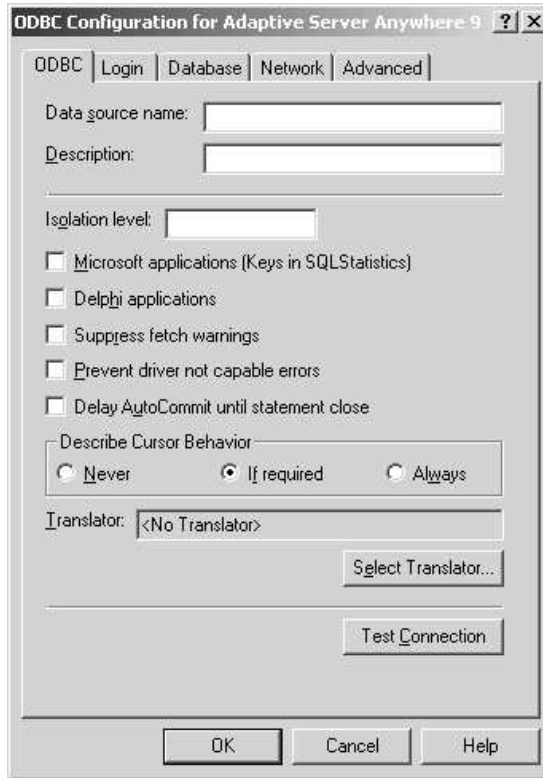
or

From Sybase Central, choose Tools ► Adaptive Server Anywhere 9 ► Open ODBC Administrator.

The ODBC Data Source Administrator appears, displaying a list of the data sources you currently have installed on your computer. For example,



2. On the User DSN tab, click Add.  
The Create New Data Source wizard appears.
3. Select **Adaptive Server Anywhere 9.0** from the list of drivers, and click Finish.  
The Adaptive Server Anywhere ODBC Configuration dialog appears.



Many of the fields in this dialog are optional. Click the Help button at the bottom of each tab for a description of all the fields on that tab. You probably only need to use the following parameters:

- ◆ **Data Source Name (ODBC tab)** Type a name that will appear in the Connect dialog. It can contain spaces, but should be short.  
For example, for the database created in the previous chapter you could enter a data source name of **My Sample**.
- ◆ **User ID (Login tab)** The database user ID you will use to connect. If you omit the user ID, you will be prompted for it when you attempt to connect.  
For newly created databases, the default user ID is **DBA**.
- ◆ **Password (Login tab)** You should omit or encrypt the password if there are security concerns with having passwords stored on your machine. If you omit the password, you will be prompted for it when you attempt to connect.  
The default password for the DBA user ID is **SQL**.
- ◆ **Database File (Database tab)** You can select a database file by browsing your machine.

---

For the database created in the previous chapter you would enter `c:\temp\mysample.db`.

4. When you have specified the parameters you need, click OK to create the data source and close the dialog.

☞ For descriptions of the fields on each tab in the ODBC Configuration dialog, see “Configuring ODBC data sources using the ODBC Administrator” [ASA Database Administration Guide, page 56].

☞ For a full description of database connections, see “Connecting to a Database” [ASA Database Administration Guide, page 37].

☞ For more information about ODBC data sources, see “Working with ODBC data sources” [ASA Database Administration Guide, page 53].



## CHAPTER 7

# Using Interactive SQL

### About this chapter

This chapter discusses how to run and use Interactive SQL. Interactive SQL is a utility shipped with Adaptive Server Anywhere; it lets you execute SQL statements, build scripts, and display database data.

### Contents

<b>Topic:</b>	<b>page</b>
<a href="#">Introduction to Interactive SQL</a>	68
<a href="#">Starting Interactive SQL</a>	70
<a href="#">Using Interactive SQL to display data</a>	75
<a href="#">Working with SQL statements in Interactive SQL</a>	80

# Introduction to Interactive SQL

Interactive SQL is a utility for sending SQL statements to the database server. You can use it for the following purposes:

- ◆ Browsing the information in a database.
- ◆ Trying out SQL statements that you plan to include in an application.
- ◆ Loading data into a database and carrying out administrative tasks.

In addition, Interactive SQL can run **command files** or **script files**. For example, you can build repeatable scripts to run against a database and then use Interactive SQL to execute these scripts in a batch fashion.

## About this chapter

The following table lists the most common information how to run and use Interactive SQL.

If you want to know...	Then see...
How to start Interactive SQL	<a href="#">“Starting Interactive SQL” on page 70</a>
How to connect to a database	<a href="#">“Connecting Your Application to its Database” on page 61</a>
How to use the Interactive SQL toolbar	<a href="#">“Interactive SQL main window description” on page 70</a>
How to open a new Interactive SQL window	<a href="#">“Opening multiple windows” on page 71</a>
What keyboard shortcuts are available in Interactive SQL	<a href="#">“Interactive SQL keyboard shortcuts” on page 72</a>
How to display data	<a href="#">“Using Interactive SQL to display data” on page 75</a>
How to execute SQL commands in Interactive SQL	<a href="#">“Working with SQL statements in Interactive SQL” on page 80</a>
Where to find more detailed information on selecting data	<a href="#">“Selecting Data from Database Tables” on page 87</a>
Where to find information about using the Query Editor to build SELECT statements	<a href="#">“Introducing the Query Editor” [SQL Anywhere Studio Help, page 190]</a>

If you want to know...	Then see...
Where to find more information on loading and unloading data	“Introduction to import and export” [ASA <i>SQL User’s Guide</i> , page 522]
How to set Interactive SQL options	“Options dialog” [ <i>SQL Anywhere Studio Help</i> , page 144]
How to automate common tasks	“Running SQL command files” [ASA <i>SQL User’s Guide</i> , page 553]
How to use JDBC escape syntax	“Using JDBC escape syntax” [ASA <i>Programming Guide</i> , page 131]
How to print from Interactive SQL	<a href="#">“Printing SQL statements” on page 83</a>
How to print the graphical plan from Interactive SQL	“Graphical plans” [ASA <i>SQL User’s Guide</i> , page 429]
How to analyze queries using the Index Consultant	“Starting the Index Consultant” [ASA <i>SQL User’s Guide</i> , page 65]

---

# Starting Interactive SQL

You can start Interactive SQL in two ways: from Sybase Central or on its own. The way that you start Interactive SQL on its own depends on your operating system.

For detailed information on connecting to databases, see “Connecting to a Database” [ASA Database Administration Guide, page 37].

## ❖ To start Interactive SQL from Sybase Central

1. To start Interactive SQL, right-click a database in Sybase Central and choose File ► Open Interactive SQL.

In this case Interactive SQL automatically connects to the database.

or

2. To start Interactive SQL without a connection to a database, choose Tools ► Adaptive Server Anywhere 9 ► Open Interactive SQL.

The Connect dialog appears.

## ❖ To start Interactive SQL (Command line)

1. On Windows, choose Start ► Programs ► SQL Anywhere 9 ► Adaptive Server Anywhere ► Interactive SQL. The Connect dialog appears.

or

Type the following at a command prompt:

```
dbisql
```

The Connect dialog appears.

2. Enter the required information to connect to a database and click OK to connect.

## Interactive SQL main window description

Interactive SQL has the following panes:

- ◆ **SQL Statements** Provides a place for you to type SQL statements.
- ◆ **Results** Displays the results of commands that you execute. For example, if you use SQL statements to search for specific data in the database, the Results tab in this pane displays the columns and rows that match the search criteria. If the information exceeds the size of the pane, scroll bars automatically appear for the pane. You can edit the result set on the Results tab.

☞ For more information about editing the result set, see [“Editing table values in Interactive SQL” on page 76](#).

The Messages tab displays messages from the database server about the SQL statements that you execute in Interactive SQL.

The Plan tab and the UltraLite Plan tab in the Results pane display the query optimizer’s execution plan for a SQL statement.

You can set options for the tabs and panes in the main Interactive SQL window from the Options dialog found in the Tools menu.

The title bar at the top of the window displays connection information, as follows:

```
database-name ( userid ) on server-name
```

For example, if you connect to the sample database using the ASA 9.0 Sample ODBC data source, the title bar is as follows:

```
asademo ( dba ) on asademo9
```

## Using the Interactive SQL toolbar

The Interactive SQL toolbar appears at the top of the Interactive SQL window. It provides you with buttons for executing common commands. With the buttons on this toolbar, you can:

- ◆ Recall the executed SQL statement immediately before your current position in the history list.
- ◆ View a list of up to 50 previously executed SQL statements.
- ◆ Recall the executed SQL statement immediately after your current position in the history list.
- ◆ Execute the SQL statement currently appearing in the SQL Statements pane.
- ◆ Interrupt the execution of the current SQL statement.

As an easy reminder of what these buttons do, you can hold your cursor over each button to see a popup description.

## Opening multiple windows

You can open multiple Interactive SQL windows. Each window corresponds to a separate database connection. You can connect simultaneously to two (or more) different databases on different servers, or you can open concurrent connections to a single database.

---

## ❖ To open a new Interactive SQL window

1. Choose Window ► New Window.

The Connect dialog appears.


2. In the Connect dialog, enter connection options, and click OK to connect.

The connection information (including the database name, your user ID, and the database server name) appears on the title bar above the SQL Statements pane.

You can also connect to or disconnect from a database with the Connect and Disconnect commands in the SQL menu, or by executing a CONNECT or DISCONNECT statement.

## Interactive SQL keyboard shortcuts

Interactive SQL provides the following keyboard shortcuts:

Function key	Description
ALT+F4	Exits Interactive SQL.
ALT+LEFT CURSOR	Displays the previous SQL statement in the history list.
ALT+RIGHT CURSOR	Displays the next SQL statement in the history list.
CTRL+BREAK	Interrupts the SQL statement that is being executed.
CTRL+C	Copies the selected row(s) and column headings to the clipboard in the Results pane. In the SQL Statements pane, copies the selected text to the clipboard.
CTRL+END	Moves to the bottom of the current pane.
CTRL+H	Displays the history of your executed SQL.
CTRL+HOME	Moves to the top of the current pane.
CTRL+N	Clears the contents of the Interactive SQL window.
CTRL+P	Prints the contents of the SQL Statements pane. You can configure the appearance of the printed text in the Interactive SQL Options dialog.  For information about setting print options, see “Print tab” [ <i>SQL Anywhere Studio Help</i> , page 151].

Function key	Description
CTRL+Q	Displays the Query Editor. The Query Editor helps you build SQL queries. When you have finished building your query, click OK to export it back into the SQL Statements pane.
CTRL+S	Saves the contents of the SQL Statements pane.
ESC	Clears the SQL Statements pane.
F1	Opens Help.
F2	Edits the selected value in the result set. You can tab from column to column within the row.
F5	Executes all text in the SQL Statements pane. You can also perform this operation by clicking the Execute SQL Statement button on the toolbar.
F7	Displays the Lookup Table Name dialog. In this dialog, you can find and select a table and then press ENTER to insert the table name into the SQL Statements pane at the cursor position. Or, with a table selected in the list, press F7 again to display the columns in that table. You can then select a column and press ENTER to insert the column name into the SQL Statements pane at the cursor position.
F8	Displays the Lookup Procedure Name dialog. In this dialog, you can find and select a procedure and then press ENTER to insert the procedure name into the SQL Statements pane at the cursor position.
F9	Executes the text that is selected in the SQL Statements pane. If no text is selected, all of the statements are executed.
PGDN	Moves a page down in the current pane.
PGUP	Moves a page up in the current pane.
SHIFT+F5	Displays the plan for the statement in the SQL Statements pane without executing the statement.

The following keyboard shortcuts are available when the SQL Statements pane has the focus:

---

Function key	Description
CTRL+]	Moves the cursor to the matching brace. Brace matching matches parentheses, braces, brackets, and angle brackets.
CTRL+BACKSPACE	Deletes the word to the left of the cursor.
CTRL+DEL	Deletes the word to the right of the cursor.
CTRL+G	Opens the Go To dialog where you can specify the line you want to go to.
CTRL+L	Deletes the current line from the SQL Statements pane and puts the line onto the clipboard.
CTRL+SHIFT+]	Extends the selection to the matching brace. Brace matching matches parentheses, braces, brackets, and angle brackets.
CTRL+SHIFT+L	Deletes the current line.
CTRL+SHIFT+U	Changes the selection to uppercase characters.
CTRL+U	Changes the selection to lowercase characters.
F3	Finds the next occurrence of the selected text.
HOME	Moves the cursor to the start of the current line or to the first word on the current line.
SHIFT+F3	Finds the previous occurrence of the selected text.
SHIFT+HOME	Extends the selection to the start of the text on the current line.



## Using Interactive SQL to display data

One of the principal uses of Interactive SQL is to browse table data. This section shows how to query the information in the sample database.

You can display database information using the `SELECT` statement in Interactive SQL. The following example shows the command to type in the SQL Statements pane. Once you have typed the command, you must click the Execute SQL Statement button on the toolbar to carry out the command.

After you execute the statement, the data (called a result set) appears on the Results tab in the Results pane. You can use the scroll bars to see areas of the table that are outside your current view of the pane. By default, row numbers appear to the left of the result set.

### ❖ To list all the columns and rows of the employee table

1. Start Interactive SQL and connect to the sample database.
2. Type the following in the SQL Statements pane:

```
SELECT *
FROM employee
```

3. On the toolbar, click the Execute SQL Statement button.

emp_id	manager_id	emp_lname	emp_fname	...
102	501	Fran	Whitney	...
105	501	Matthew	Cobb	...
129	902	Philip	Chin	...
148	1293	Julie	Jordan	...
...				

☞ For more information about `SELECT` statements, see [“Selecting Data from Database Tables” on page 87](#).

You can add, delete, and update rows within the result set.

☞ For more information about editing the result set, see [“Editing table values in Interactive SQL” on page 76](#).

---

## Editing table values in Interactive SQL

Once you execute a query in Interactive SQL, you can edit the result set to modify the database. You can also select rows from the result set and copy them for use in other applications. Interactive SQL supports editing, inserting, and deleting rows. These actions have the same effect as executing UPDATE, INSERT, and DELETE statements.

To edit a row or value in the result set, you must have the proper permissions on the table or column you want to modify values from. For example, if you want to delete a row, then you must have DELETE permission for the table the row belongs to.

Editing the result set may fail if you:

- ◆ attempt to edit a row or column you do not have permission on.
- ◆ select columns from a table with a primary key, but you do not select all of the primary key columns.
- ◆ attempt to edit the result set of a JOIN (for example, there is data from more than one table in the result set).
- ◆ enter an invalid value (for example, a string in a numeric column or a NULL in a column that does not allow NULLs).

When editing fails, an Interactive SQL error message appears explaining the error, and the database table values remain unchanged.

Once you make changes to table values, you must enter a COMMIT statement to make the changes permanent. If you want to undo your changes, you must execute a ROLLBACK statement.

☞ For more information, see “COMMIT statement” [ASA SQL Reference, page 284] and “ROLLBACK statement” [ASA SQL Reference, page 537].

## Editing table values from the Interactive SQL result set

From Interactive SQL you can change any or all of the values within existing rows in database tables. You must have UPDATE permission on the columns being modified. When you edit the result set, you can only make changes to the values in one row at a time.

**❖ To edit a row in the result set**

1. Click the value you want to change.
2. Right-click the result set and choose Edit from the popup menu. You can also press F2 to edit the result set.

A blinking cursor appears in the table cell containing the value.

3. Enter the new value.

You cannot enter invalid data types into a column. For example, you cannot enter a string into a column that accepts the INT data type.

If you are done editing values in the row, press Enter to update the database. If you want to change other values in the row, press TAB or SHIFT+TAB to move to the other values.

You can press the ESC key to cancel the change that was made to the selected value.

4. Execute a COMMIT statement to make your changes to the table permanent.

Editing computed  
columns

Once you edit values in the result set, the database is updated with the modified values. Computed columns are recalculated based on the modified values whether or not they are part of the result set. However, if there are computed columns in your result set, and you modify a value in the computed column, the database is updated with the modified value.

**Inserting rows into the database from the Interactive SQL result set**

Interactive SQL also allows you to add new rows to result sets. You tab between columns in the result set to add values to the row. When you add values to the table, characters are stored in the same case as they are entered. You must have INSERT permission on the table to add new rows.

**❖ To insert a new row into the result set**

1. Right-click the result set and choose Add from the popup menu.

A new blank row appears with a blinking cursor in the first value in the row.

Press TAB to move the cursor from column to column across the row. You can also insert a value by clicking on the value within the selected row.

2. Enter the new value.

You cannot enter invalid data types into a column. For example, you cannot enter a string into a column that accepts the INT data type.

- 
3. Press TAB to move to the next column.
  4. Repeat steps 2 and 3 until all the column values are added.
  5. Press ENTER to update the database.
  6. Execute a COMMIT statement to make your changes to the table permanent.

Inserting values into  
computed columns

If the result set contains a computed column and you do not specify a value for the computed column, the value is calculated when the database is updated. However, if you specify a value for the computed column, the database is updated with the specified value, and a value is not calculated for the computed column.

## Deleting rows from the database using Interactive SQL

You can also delete rows from a database table in Interactive SQL. You must have DELETE permission on the table to delete rows. You can select only consecutive rows in the result set.

### ❖ To delete a row from the result set

1. Select the row(s) you want to delete. To select a row(s):
  - ◆ Press and hold the SHIFT key while clicking the row(s)
  - ◆ Press and hold the SHIFT key while using the Up or Down arrowIf you want to delete non-consecutive rows, you must delete each row individually.
2. Right-click the result set and choose Delete from the popup menu. You can also delete the selected row(s) by pressing the DELETE key.

The selected row(s) are removed from the database table.
3. Execute a COMMIT statement to make your changes to the table permanent.

## Copying rows from the Interactive SQL result set

You can copy rows directly from the result set in Interactive SQL and then paste them into other applications. Copying rows also copies the column headings. Copied data is comma-delimited, which allows other applications, such as Microsoft Excel, to format the copied data correctly. Copied data is in ASCII format, and all of the strings are enclosed in single quotes. You can select only consecutive rows in the result set.

❖ **To copy rows from the Interactive SQL result set**

1. Select the row(s) you want to copy. To select a row(s):
  - ◆ Press and hold the `SHIFT` key while clicking the row(s)
  - ◆ Press and hold the `SHIFT` key while using the Up or Down arrow
2. Right-click the result set and choose Copy from the popup menu. You can also copy the selected row(s) by pressing `CTRL+C`.

The selected row(s), including their column headings, are copied to the clipboard. You can paste them into other applications.

Copying individual values  
from the result set

You can copy a single value from the result set by selecting a value, right-clicking the result set and choosing Copy Cell from the popup menu. When you do this, no column headings are copied—only the data is copied.

---

## Working with SQL statements in Interactive SQL

The following sections describe some of the commands you can use in Interactive SQL. This section describes general tasks for working with commands in Interactive SQL.

All SQL statements can be entered as commands in the top pane of the Interactive SQL window. When you are finished typing, you need to execute the statement to view its results.

❖ **To execute a SQL statement, do one of the following**

1. Press the Execute SQL Statement button, or choose SQL ► Execute, or press F5.

❖ **To clear the SQL Statements pane**

1. Choose Edit ► Clear SQL or press ESCAPE.

## Canceling an Interactive SQL command

The Interrupt button on the Interactive SQL toolbar cancels a command.

A Cancel operation stops current processing and prompts for the next command. If a command file was being processed, or if there is more than one statement in the SQL Statements pane, you are prompted for an action to take (Stop Command File, Continue, or Exit Interactive SQL). These actions can be controlled with the Interactive SQL ON\_ERROR option.

☞ For information about the ON\_ERROR option, see “ON\_ERROR option [ISQL]” [ASA Database Administration Guide, page 611].

### Reported messages

When an interruption is detected, one of three different messages is reported depending upon when the interruption is detected.

1. If the interruption is detected when Interactive SQL is processing the request (as opposed to the database server), then the following message appears:

```
ISQL command terminated by user
```

Interactive SQL stops processing immediately and the current database transaction is left alone.

2. If the interruption is detected by the database server while processing a data manipulation command (SELECT, INSERT, DELETE, or UPDATE), then the following message appears:

```
Statement interrupted by user.
```

The effects of the current command are undone, but the rest of the transaction is left intact.

3. If the interruption is detected while the database server is processing a data definition command (CREATE, DROP, ALTER, etc.), the following message appears:


```
Terminated by user - transaction rolled back
```

Since data definition commands all perform a COMMIT automatically before the command starts, the effect of the ROLLBACK is to just cancel the current command.

This message also occurs when the database server is running in bulk operations mode executing a command that modifies the database (INSERT, UPDATE, or DELETE). In this case, ROLLBACK cancels not only the current command, but everything that has been done since the last COMMIT. In some cases, it may take a considerable amount of time for the database server to perform the automatic ROLLBACK.

## Executing multiple statements

The Interactive SQL environment allows multiple statements to be entered at the same time. This can be done by ending each statement with a command delimiter. The command delimiter is a configurable option in Interactive SQL that you can change using the COMMAND\_DELIMITER option. By default, it is a semicolon (;).

 For more information, see “COMMAND\_DELIMITER option [ISQL]” [ASA Database Administration Guide, page 583].

### ❖ To enter multiple statements in SQL Statements pane

1. Enter the following three commands into the SQL Statements pane.

```
UPDATE employee
SET dept_id = 400,
    manager_id = 1576
WHERE emp_id = 467;

UPDATE employee
SET dept_id = 400,
    manager_id = 1576
WHERE emp_id = 195;

SELECT *
FROM employee
WHERE emp_id IN ( 195, 467 );
```

2. On the toolbar, click the Execute SQL Statement button. All three statements are executed. After execution, the commands remain in the SQL Statements pane.
3. Roll back your changes by typing `ROLLBACK` in the SQL Statements pane and executing the statement.

Using `go` as an alternative

An alternative to using the semicolon is to enter `go` on a line by itself, at the beginning of the line.

```
UPDATE employee
SET dept_id = 400,
    manager_id = 1576
WHERE emp_id = 467
go

UPDATE employee
SET dept_id = 400,
    manager_id = 1576
WHERE emp_id = 195
go

SELECT *
FROM employee
WHERE emp_id IN ( 195, 467 )
go
```

**Tip**

You can press F9 to execute only the selected text in the SQL Statements pane.

## Looking up tables, columns, and procedures

While you are entering commands in Interactive SQL, you can look up the names of tables, columns, or procedures stored in the current database and insert them at your cursor position.

❖ **To look up the names of tables in the database**

1. Choose Tools ► Lookup Table Name or press F7.
2. Find and select the table.
3. Click OK to insert the table name into the SQL Statements pane at the current cursor position.



### ❖ To look up column names in the database

1. Choose Tools ► Lookup Table Name or press F7.
2. Find and select the table containing the column.
3. Click Show Columns.
4. Select the column and click OK to insert the column name into the SQL Statements pane at the current cursor position.

### ❖ To look up the names of procedures in the database

1. Choose Tools ► Lookup Procedure Name or press F8.
2. Find and select the procedure.
3. Click OK to insert the procedure name into the SQL Statements pane at the current cursor position.

In the tables and procedures lookup dialogs, you can enter the first few characters of the table or procedure you are looking for. After you type something in the field, the dialog narrows the list to include only those items that start with the text you entered.

You can use the SQL wildcard characters ‘%’ (percent sign) and ‘\_’ (underscore) to help narrow your search. ‘%’ matches any string of zero or more characters, while ‘\_’ matches any one character.

For example, to list all the tables that contain the word profile, type **%profile%**.

If you want to search for a percent sign or underscore within a table name, you must prefix the percent sign or underscore with an escape character. The escape character depends on the JDBC driver that you are using. If you are connected via jConnect, the escape character is ‘\’ (backslash) while the escape character for the iAnywhere JDBC driver is ‘~’ (tilde).

## Printing SQL statements

You can print the contents of the SQL Statements pane by pressing CTRL+P or by choosing Print from the File menu. You can add a header or footer and configure other formatting options in the Interactive SQL Options dialog.

☞ For information about configuring print options, see “Print tab” [*SQL Anywhere Studio Help*, page 151].

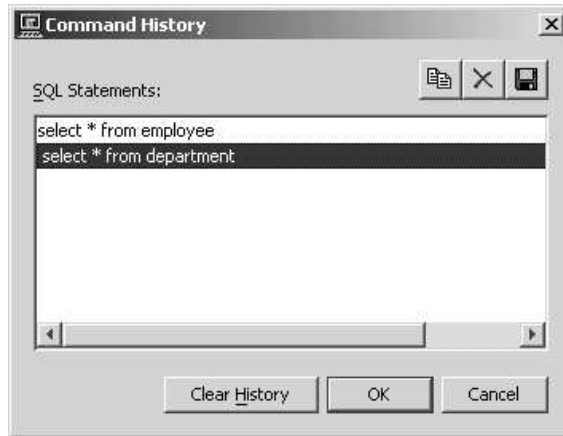
☞ For information about print graphical plans in Interactive SQL, see “Graphical plans” [*ASA SQL User’s Guide*, page 429].

---

## Recalling commands

When you execute a command, Interactive SQL automatically saves it in a history list that persists between Interactive SQL sessions. Interactive SQL maintains a record of up to 50 of the most recent commands.

You can view the entire list of commands in the Command History dialog. To access the Command History dialog, press CTRL+H, or click the book icon in the toolbar.



The most recent commands appear at the bottom of the list. To recall a command, select it and click OK. It will appear in the SQL Statements pane of Interactive SQL.

You can also recall commands without the Command History dialog. Use the arrows in the toolbar to scroll back and forward through your commands, or press ALT+RIGHT ARROW and ALT+LEFT ARROW.

**Note**

If you execute a SQL statement that contains password information (GRANT CONNECT, GRANT REMOTE DBA, CONNECT, or CREATE EXTERNLOGIN), the password information appears in the Command History dialog for the duration of the current Interactive SQL session.

When the command history is viewed in subsequent Interactive SQL sessions, passwords are replaced with ... in any of these statements that contain password information. For example, if you execute the following statement in Interactive SQL:

```
GRANT CONNECT TO testuser IDENTIFIED BY testpassword
```

the following appears in the Command History dialog in subsequent Interactive SQL sessions:

```
GRANT CONNECT TO testuser IDENTIFIED BY ...
```

When Interactive SQL saves the command history between sessions, it removes password information from the statements listed above.

Copying commands in the Command History dialog

You can copy a command to the clipboard, by selecting it in the Command History dialog and then pressing CTRL+C or clicking the Copy button when the dialog has the focus. If you want to copy selected commands to the SQL Statements pane, click OK. When you copy multiple commands, they are separated by the command delimiter (a semicolon by default).

Removing commands from the Command History dialog

The contents of the Command History dialog persist between Interactive SQL sessions. You can remove commands from the dialog in one of two ways:

- ◆ Select one or more commands and click the Delete button or press the DELETE key to remove the selected command(s) from the dialog. This action cannot be undone.
- ◆ Remove all the commands from the dialog by clicking the Clear History button. This action cannot be undone.

You can also save commands in text files so that you can use them in a subsequent Interactive SQL session.

❖ **To save the command history to a file**

1. Open the Command History dialog.
2. Click the Save button or press CTRL+S.
3. In the Save As dialog, specify a location and name for the file.

The command history file has a *.SQL* extension.

- 
4. Click Save when finished.

## Logging commands

With the Interactive SQL logging feature, you can record commands as you execute them. Interactive SQL continues to record until you stop the logging process, or until you end the current session. The recorded commands are stored in a log file.

### ❖ To begin logging Interactive SQL commands

1. Choose SQL ► Start Logging.
2. In the Save As dialog, specify a location and name for the log file.  
A log file must have the *.SQL* extension.
3. Click Save when finished.

### ❖ To stop logging Interactive SQL commands

1. Choose SQL ► Stop Logging.

#### **Tips**

You can also start and stop logging by typing in the SQL Statements pane. To start logging, type and execute **START LOGGING 'c:\filename.sql'**, where *c:\filename.sql* is the path, name, and extension of the log file. A log file must have the *.SQL* extension. You only need to include the single quotation marks if the path contains embedded spaces. To stop logging, type and execute **STOP LOGGING**.


Once you start logging, all commands that you try to execute are logged, including ones that do not execute properly.

## CHAPTER 8

# Selecting Data from Database Tables

### About this chapter

This chapter introduces queries, which retrieve data from a database. It describes how to retrieve data from a single table.

 For information about selecting data from multiple related tables, see [“Selecting Data from Multiple Tables” on page 105](#). In addition, [“Selecting Aggregate Data” on page 117](#) describes how to group your data and perform calculations on the data in one or more columns.

### Contents

Topic:	page
<a href="#">Introduction</a>	88
<a href="#">Selecting a complete table</a>	90
<a href="#">Selecting columns from a table</a>	92
<a href="#">Ordering query results</a>	95
<a href="#">Selecting rows from a table</a>	98

# Introduction

All interaction between applications (clients) and database servers is carried out by sending SQL statements to the database server, which returns information to the client.

The SELECT statement retrieves information from a database for use by the client application. SELECT statements are also called **queries**. The information is delivered to the client in the form of a result set. The client application can then process the result set. For example, Interactive SQL displays the result set in the Results pane. Result sets consist of a set of rows, just like tables in the database.

SELECT statements can become highly complex in applications retrieving very specific information from many tables. This chapter uses only simple SELECT statements: more advanced queries are described in later tutorials.

☞ For the full syntax of the SELECT statement, see “SELECT statement” [ASA SQL Reference, page 541].

## About this chapter

If you want to know...	Then see...
How to order query results	<a href="#">“Ordering query results” on page 95</a>
How to select columns from a table	<a href="#">“Selecting columns from a table” on page 92</a>
How to select rows from a table	<a href="#">“Selecting rows from a table” on page 98</a>
How to compare dates in queries	<a href="#">“Comparing dates in search conditions” on page 99</a>
How to use pattern matching	<a href="#">“Pattern matching in search conditions” on page 100</a>
How to match rows by sound	<a href="#">“Matching rows by sound” on page 101</a>
How to use compound search conditions	<a href="#">“Using compound search conditions” on page 102</a>
Where to find information about shortcuts for search conditions	<a href="#">“Shortcuts for compound search conditions” on page 102</a>

Notes

You should run the Adaptive Server Anywhere software on your computer while you read and work through the examples in this chapter.

Each example instructs you to type commands into Interactive SQL and describes what you will see on your computer screen. If you cannot run the software as you read the tutorials, you will still be able to learn about SQL, but you will not have the opportunity to experiment on your own.

The examples assume that you have started Interactive SQL and are connected to the sample database.

☞ For instructions, see [“Adaptive Server Anywhere Quick Start” on page 1](#).

---

## Selecting a complete table

The simplest `SELECT` statement retrieves all the data in a single table. This `SELECT` statement has the following syntax:

**`SELECT * FROM table-name`**

where *table-name* should be replaced with the name of the table you are querying. The asterisk (\*) is a short form for a list of all columns.

### ❖ List all products sold by the company

1. In Interactive SQL, type the following in the SQL Statements pane and press F5 to execute the statement.

```
SELECT * FROM Product
```

The `SELECT` statement retrieves all the rows and columns of the product table, and displays them on the Results tab in the Results pane:

id	name	description	size	color	quantity	unit_price
300	Tee Shirt	Tank Top	Small	White	28	9
301	Tee Shirt	V-neck	Medium	Orange	54	14
302	Tee Shirt	Crew Neck	One size fits all	Black	75	14
400	Baseball Cap	Cotton Cap	One size fits all	Black	112	9
...	...	...	...	...	...	...

The Product table contains seven columns. Each column has a name, such as color or id. There is a row for each product that the company sells, and each row has a single value in each column. For example, the product with ID 301 is a Tee Shirt. It is a V-neck style in medium size, and is orange in color.

### Notes

- ◆ **Table names are case insensitive** The table name Product starts with an upper case P, even though the real table name is all lower case. Adaptive Server Anywhere databases can be created as case-sensitive or case-insensitive in their string comparisons, but are always case insensitive in their use of identifiers such as table names and column names.

☞ For information on creating databases, see “Creating a database” [ASA SQL User’s Guide, page 27], or “The Initialization utility” [ASA Database Administration Guide, page 485].



- ◆ **SQL keywords are case insensitive** You can enter **select** or **Select** instead of **SELECT**. In the documentation, upper case letters are generally used for SQL keywords.
- ◆ **Line breaks are not important** You can type the statements all on one line, or break them up by pressing Enter at the end of each line. Some SQL statements, such as the **SELECT** statement, consist of several parts, called **clauses**. In many examples, each clause is placed on a separate line for easier reading, so the statement in the example is commonly written as follows in the documentation:

```
SELECT *  
FROM product
```

- ◆ **Row order in the result set is insignificant** There is no guarantee of the order in which rows are returned from the database, and no meaning to the order. If you wish to retrieve rows in a particular order, you must specify the order in the query.

☞ For more information, see [“Ordering query results” on page 95](#).

#### Exercise

Try querying other tables in the sample database, such as the employee, customer, contact, or sales\_order tables.

---

## Selecting columns from a table

You can limit the columns that a **SELECT** statement retrieves by listing the desired columns immediately after the **SELECT** keyword. This **SELECT** statement has the following syntax:

```
SELECT column-name-1, column-name-2,...
FROM table-name
```

where *column-name-1*, *column-name-2*, and *table-name* should be replaced with the names of the desired columns and table you are querying.

The list of result-set columns is called the **select list**. It is separated by commas. There is no comma after the last column in the list, or if there is only one column in the list. Limiting the columns in this way is sometimes called a **projection**.

### ❖ List the name, description, and color of each product

1. In Interactive SQL, type the following in the SQL Statements pane and press F5 to execute the statement.

```
SELECT name, description, color
FROM product
```

name	description	color
Tee Shirt	Tank Top	White
Tee Shirt	V-neck	Orange
Tee Shirt	Crew Neck	Black
Baseball Cap	Cotton Cap	Black
...	...	...

#### Rearranging columns

The columns appear in the order in which you type them in the **SELECT** statement. If you want to rearrange the columns, simply change the order of the column names in the statement. For example, to put the **description** column on the left, use the following statement:

```
SELECT description, name, color
FROM product
```

## Using calculated columns

The columns of the result set do not need to be just columns in tables. They can also be expressions calculated from the underlying data. You can

combine table columns into a single result-set column, and you can use a wide variety of functions and operators to control the results you get.

### ❖ List the value in stock of each product

1. In Interactive SQL, type the following in the SQL Statements pane and press F5 to execute the statement.

```
SELECT id, ( unit_price * quantity ) AS value
FROM product
```

id	value
300	252
301	756
302	1050
400	1008
...	...

Notes

- ◆ **Columns can be given an alias** By default the column name is the expression listed in the select list, but for calculated columns the expression is cumbersome and not very informative. Here the calculated column is renamed in the select list as value. value is the **alias** for the column.
- ◆ **Other operators are available** In the above example, the multiplication operator is used to combine the columns. You can use other operators, including the standard arithmetic operators as well as logical operators and string operators.

For example, the following query lists the full names of all customers:

```
SELECT id, (fname || ' ' || lname ) AS "Full name"
FROM customer
```

The || operator concatenates strings. In this query, the alias for the column has spaces, and so must be surrounded by double quotes. This rule applies not only to column aliases, but to table names and other identifiers in the database.

☞ For a complete list of operators, see “Operators” [ASA SQL Reference, page 10].

- ◆ **Functions can be used** In addition to combining columns, you can use a wide range of built-in functions to produce the results you want.

For example, the following query lists the product names in upper case:

---

```
SELECT id, UCASE( name )  
FROM product
```

id	UCASE(product.name)
300	TEE SHIRT
301	TEE SHIRT
302	TEE SHIRT
400	BASEBALL CAP
...	...

☞ For a complete list of functions, see “Alphabetical list of functions” [ASA *SQL Reference*, page 97].

## Ordering query results

Unless otherwise requested, the database server returns the rows of a table in an order that has no meaning. Often it is useful to look at the rows in a table in a more meaningful sequence. For example, you might like to see products in alphabetical order.

You order the rows in a result set by an **ORDER BY** clause to the end of the **SELECT** statement. This **SELECT** statement has the following syntax:

```
SELECT column-name-1, column-name-2,...
FROM table-name
ORDER BY order-by-column-name
```

where *column-name-1*, *column-name-2*, and *table-name* should be replaced with the names of the desired columns and table you are querying, and where *order-by-column-name* is a column in the table. As before, you can use the asterisk as a short form for all the columns in the table.

### ❖ List the products in alphabetical order

1. In Interactive SQL, type the following in the SQL Statements pane:

```
SELECT id, name, description
FROM product
ORDER BY name
```

id	name	description
400	Baseball Cap	Cotton Cap
401	Baseball Cap	Wool cap
700	Shorts	Cotton Shorts
600	Sweatshirt	Hooded Sweatshirt
...	...	...

Notes

- ◆ **The order of clauses is important** The **ORDER BY** clause must follow the **FROM** clause and the **SELECT** clause.
- ◆ **You can specify either ascending or descending order** The default order is ascending. You can specify a descending order by adding the keyword **DESC** to the end of the clause, as in the following query:

```
SELECT id, quantity
FROM product
ORDER BY quantity DESC
```

---

id	quantity
400	112
700	80
302	75
301	54
600	39
...	...

- ◆ **You can order by several columns** The following query sorts first by size (alphabetically), and then by name:

```
SELECT id, name, size
FROM product
ORDER BY size, name
```

id	name	size
600	Sweatshirt	Large
601	Sweatshirt	Large
700	Shorts	Medium
301	Tee Shirt	Medium
...	...	...

- ◆ **The ORDER BY column does not need to be in the select list** The following query sorts products by unit price, even though the price is not included in the result set

```
SELECT id, name, size
FROM product
ORDER BY unit_price
```

id	name	size
500	Visor	One size fits all
501	Visor	One size fits all
300	Tee Shirt	Small
400	Baseball Cap	One size fits all
...	...	...

- ◆ **If you do not use an ORDER BY clause, and you execute a query**

**more than once, you may appear to get different results** This is because Adaptive Server Anywhere may return the same result set in a different order. In the absence of an ORDER BY clause, Adaptive Server Anywhere returns rows in whatever order is most efficient. This means the appearance of result sets may vary depending on when you last accessed the row and other factors. The only way to ensure that rows are returned in a particular order is to use ORDER BY.

## Using indexes to improve ORDER BY performance

Sometimes there is more than one possible way for the Adaptive Server Anywhere database server to execute a query with an ORDER BY clause. You can use indexes to enable the database server to search the tables more efficiently.

Queries with WHERE and ORDER BY clauses

An example of a query that can be executed in more than one possible way is one that has both a WHERE clause and an ORDER BY clause.

```
SELECT *
FROM customer
WHERE id > 300
ORDER BY company_name
```

In this example, Adaptive Server Anywhere must decide between two strategies:

1. Go through the entire customer table in order by company name, checking each row to see if the customer id is greater than 300.
2. Use the key on the id column to read only the companies with id greater than 300. The results would then need to be sorted by company name.

If there are very few id values greater than 300, the second strategy is better because only a few rows are scanned and quickly sorted. If most of the id values are greater than 300, the first strategy is much better because no sorting is necessary.

Solving the problem

Creating a two-column index on id and company\_name could solve the example above. Adaptive Server Anywhere can use this index to select rows from the table in the correct order. However, keep in mind that indexes take up space in the database file and involve some overhead to keep up to date. Do not create indexes indiscriminately.

☞ For more information, see “Using indexes” [ASA SQL User’s Guide, page 163].

## Selecting rows from a table

You can limit the rows that a `SELECT` statement retrieves by adding a `WHERE` clause to your query. This is sometimes called applying a **restriction** to the result set. The `WHERE` clause includes a **search condition** that specifies the rows to be returned. This `SELECT` statement has the following syntax:

```
SELECT column-name-1, column-name-2,...
FROM table-name
WHERE search-condition
```

where, as before, *column-name-1*, *column-name-2*, and *table-name* should be replaced with the names of the desired columns and table you are querying. The *search-condition* is described more below. If you use an `ORDER BY` clause, it must come after the `WHERE` clause.

### ❖ List all products colored black

1. In Interactive SQL, type the following in the SQL Statements pane:

```
SELECT *
FROM product
WHERE color = 'black'
```

id	name	description	size	color	quantity	unit_price
302	Tee Shirt	Crew Neck	One size fits all	Black	75	14
400	Baseball Cap	Cotton Cap	One size fits all	Black	112	9
501	Visor	Plastic Visor	One size fits all	Black	28	7
...	...	...	...	...	...	...

### Notes

- ◆ The `WHERE` clause includes a search condition to select rows. In this case the search condition is `color = 'black'`. Other search conditions are described in the following sections.

☞ For information on search conditions, see “Search conditions” [ASA *SQL Reference*, page 22].

- ◆ The single quotes around **black** are required. They indicate that **black** is a character string. Double quotes have a different meaning. Double quotes can be used to make otherwise invalid strings valid for column names and other identifiers.

☞ For information about strings, see “Strings” [ASA *SQL Reference*, page 8].



- ♦ The sample database is not case sensitive, so you would get the same results whether you searched for **BLACK**, **black**, or **Black**.
- ♦ How you order clauses is important. The **SELECT** list is followed by the **FROM** clause, followed by the **WHERE** clause, and then the **ORDER BY** clause. Typing the clauses in a different order gives a syntax error.

### Exercise

Try some queries that combine what you have learned in this chapter. Here is one query that lists the names and birth dates of all employees named John.

```
SELECT ( emp_fname || ' ' || emp_lname) AS Name,
       birth_date
FROM employee
WHERE emp_fname = 'John'
ORDER BY birth_date
```

Name	birth_date
John Letiecq	4/27/1939
John Sheffield	9/25/1955

## Comparing dates in search conditions

You can use operators other than equals to select a set of rows that satisfy the search condition. The inequality operators (< and >) can be used to compare numbers, dates, and even character strings.

### ❖ List all employees born before March 13, 1964

1. In Interactive SQL, type the following in the SQL Statements pane:

```
SELECT emp_lname, birth_date
FROM employee
WHERE birth_date < 'March 13, 1964'
ORDER BY birth_date DESC
```

emp_lname	birth_date
Ahmed	12/12/1963
Dill	7/19/1963
Rebeiro	4/12/1963
Garcia	1/23/1963
Pastor	7/14/1962
...	...

- ◆ **Automatic conversion to dates** The Adaptive Server Anywhere database server knows that the `birth_date` column contains dates, and automatically converts the string `'March 13, 1964'` to a date.
- ◆ **Ways of specifying dates** There are many ways of specifying dates. The following are all accepted by Adaptive Server Anywhere:

```
'March 13, 1964'  
'1964/03/13'  
'1964-03-13'
```

You can tune the interpretation of dates in queries by setting a database option. Dates in the format `yyyy/mm/dd` or `yyyy-mm-dd` are always recognized unambiguously as dates, regardless of the `DATE_ORDER` setting.

☞ For information on controlling allowable date formats in queries, see “`DATE_ORDER` option [compatibility]” [ASA Database Administration Guide, page 588], and “Setting options” [ASA Database Administration Guide, page 556].

- ◆ **Other comparison operators** For a complete list of available comparison operators, see “Comparison operators” [ASA SQL Reference, page 10].

## Pattern matching in search conditions

Pattern matching is a versatile way of identifying character data. In SQL, the `LIKE` keyword is used to search for patterns. Pattern matching employs wildcard characters to match different combinations of characters.

### ❖ List all employees whose last name begins with BR

1. In Interactive SQL, type the following in the SQL Statements pane:

```
SELECT emp_lname, emp_fname  
FROM employee  
WHERE emp_lname LIKE 'br%'
```

emp_lname	emp_fname
Breault	Robert
Braun	Jane

The `%` in the search condition indicates that any number of other characters may follow the letters `BR`.

❖ **List all employees whose last name begins with BR, followed by zero or more letters and a T, followed by zero or more letters**

1. In Interactive SQL, type the following in the SQL Statements pane:

```
SELECT emp_lname, emp_fname
FROM employee
WHERE emp_lname LIKE 'BR%T%'
```

emp_lname	emp_fname
Breault	Robert

The first % sign matches the string **ea**ul, while the second % sign matches the empty string (no characters).

Another special character that can be used with LIKE is the \_ (underscore) character, which matches exactly one character. For example, the pattern 'BR\_U%' matches all names starting with BR and having U as the fourth letter. In Braun the \_ character matches the letter A and the % matches N.

☞ For more information, see “LIKE conditions” [ASA SQL Reference, page 24].

## Matching rows by sound

With the SOUNDEX function, you can match rows by sound. For example, suppose a phone message was left for a name that sounded like “Ms. Brown”. Which employees in the company have names that sound like Brown?

❖ **List employees with a last name that sound like Brown**

1. In Interactive SQL, type the following in the SQL Statements pane:

```
SELECT emp_lname, emp_fname
FROM employee
WHERE SOUNDEX( emp_lname ) = SOUNDEX( 'Brown' )
```

emp_lname	emp_fname
Braun	Jane

The algorithm used by SOUNDEX makes it useful mainly for English-language databases.

☞ For more information, see “SOUNDEX function [String]” [ASA SQL Reference, page 187].

---

## Using compound search conditions

Search conditions can be combined using the AND and OR logical operators to make compound search conditions. Each individual piece of the search condition is sometimes called a **predicate**.

❖ **List all employees born before March 13, 1964, except the employee named Whitney**

1. In Interactive SQL, type the following in the SQL Statements pane:

```
SELECT emp_lname, birth_date
FROM employee
WHERE birth_date < '1964-3-13'
AND emp_lname <> 'whitney'
```

emp_lname	birth_date
Cobb	12/4/1960
Jordan	12/13/1951
Breault	5/13/1947
Espinoza	12/14/1939
Dill	7/19/1963
Francis	9/12/1954

## Shortcuts for compound search conditions

Using the short form  
**BETWEEN**

SQL has two short forms for compound search conditions. The first, **BETWEEN**, is used when you are looking for a range of values. For example the following two queries are equivalent:

```
SELECT emp_lname, birth_date
FROM employee
WHERE birth_date BETWEEN '1963-1-1' AND '1965-3-31'
```

and

```
SELECT emp_lname, birth_date
FROM employee
WHERE birth_date >= '1963-1-1'
AND birth_date <= '1965-3-31'
```

Using the short form **IN**

The second short form, **IN**, may be used when looking for one of a number of values. The following two statements are equivalent:

```
SELECT emp_lname, emp_id
FROM employee
WHERE emp_lname IN ('yeung','bucceri','charlton')
```

and

```
SELECT emp_lname, emp_id
FROM employee
WHERE emp_lname = 'yeung'
OR emp_lname = 'bucceri'
OR emp_lname = 'charlton'
```



## CHAPTER 9

# Selecting Data from Multiple Tables

### About this chapter

This chapter describes database queries that look at information in more than one table. To do this, SQL provides the **JOIN** operator. There are several different ways to join tables together in queries, and this chapter describes some of the more important ones.

### Contents

<b>Topic:</b>	<b>page</b>
<a href="#">Introduction</a>	106
<a href="#">Joining tables using the cross product</a>	108
<a href="#">Using the ON phrase to restrict a join</a>	109
<a href="#">Joining tables using key joins</a>	111
<a href="#">Joining tables using natural joins</a>	113
<a href="#">Joining tables using outer joins</a>	115

---

# Introduction

Sometimes it is necessary to view data from multiple related tables. This chapter explains how to use a **join** to view the data in a useful and meaningful way.

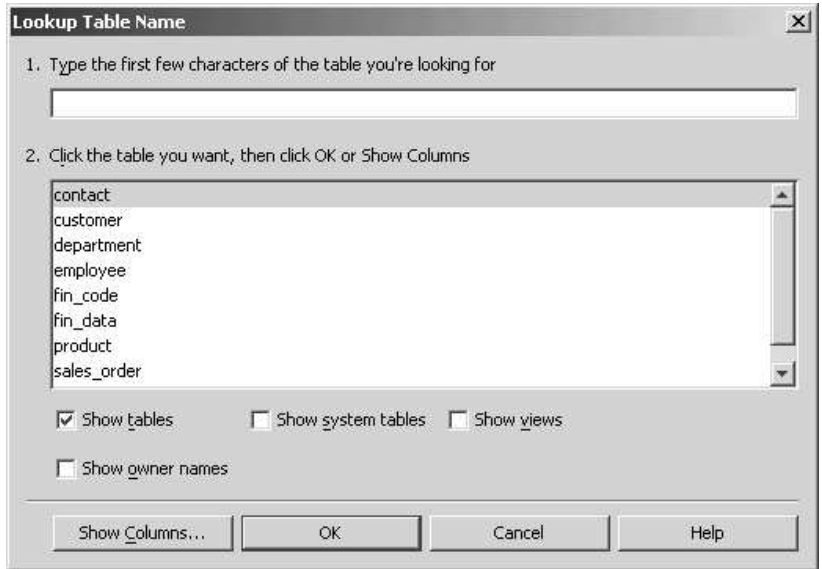
## About this chapter

If you want to know...	Then see...
How to display a list of tables	<a href="#">“Displaying a list of tables” on page 106</a>
How to display all combinations of data from two tables	<a href="#">“Joining tables using the cross product” on page 108</a>
How to make a join useful	<a href="#">“Using the ON phrase to restrict a join” on page 109</a>
How to join two tables in which the join’s foreign keys have the same name	<a href="#">“Joining tables using key joins” on page 111</a>
How to join tables on columns with the same names	<a href="#">“Joining tables using natural joins” on page 113</a>

## Displaying a list of tables


In Interactive SQL, you can display a list of tables by pressing the F7 key.





Select a table and click Show Columns to see the columns for that table. The ESCAPE key takes you back to the table list, and pressing it again will take you back to the SQL Statements pane. Pressing ENTER instead of ESCAPE copies the selected table or column name into the SQL Statements pane at the current cursor position.

Press ESCAPE to leave the list.

 For a diagram of the tables in the sample database, see [“The sample database” on page 46](#).

---

## Joining tables using the cross product

One of the tables in the sample database is `sales_order`, which lists the orders placed to the company. Each order has a `sales_rep` column, containing the employee ID of the sales representative responsible for the order. There are 648 rows in the `sales_order` table and 75 rows in the `employee` table.

The cross product join is a simple starting point for understanding joins, but not very useful in itself.

### ❖ List all data in the `employee` and `sales_order` tables

1. In Interactive SQL, type the following in the SQL Statements pane and press F5 to execute the statement.

```
SELECT *  
FROM sales_order CROSS JOIN employee
```

The results of this query, which appear on the Results tab in the Interactive SQL Results pane, match every row in the `employee` table with every row in the `sales_order` table. Since there are 75 rows in the `employee` table and 648 rows in the `sales_order` table, there are  $75 \times 648 = 48,600$  rows in the result of the join. Each row consists of all columns from the `sales_order` table followed by all columns from the `employee` table. This join is called a **full cross product**.

Subsequent sections describe how to construct more selective joins. The more selective joins can be thought of as applying a restriction to the cross product table.

☞ For more information, see “Cross joins” [*ASA SQL User’s Guide*, page 272].

## Using the ON phrase to restrict a join

The ON phrase applies a restriction to the rows in a join, in much the same way that the WHERE clause applies restrictions to the rows of a query.

The ON phrase allows more useful joins than the CROSS JOIN to be constructed. For example, you can apply the ON phrase to a join of the sales\_order and employee table to retrieve only those rows for which the sales\_rep in the sales\_order table is the same as the one in the employee table in every row of the result. Then each row contains information about an order and the sales representative responsible for it.

### ❖ List all sales orders with their dates, and the employee responsible for each

1. In Interactive SQL, type the following in the SQL Statements pane and press F5 to execute the statement.

```
SELECT employee.emp_lname,
       sales_order.id,
       sales_order.order_date
FROM sales_order JOIN employee
     ON sales_order.sales_rep = employee.emp_id
```

emp_lname	id	order_date
Chin	2008	4/2/2001
Chin	2020	3/4/2001
Chin	2032	7/5/2001
Chin	2044	7/15/2000
Chin	2056	4/15/2001
...	...	...

#### Notes

The table name is given as a prefix to identify the columns. Using the table name prefix clarifies the statement, and is required when two tables have a column with the same name.

The results of this query contain only 648 rows (one for each row in the sales\_order table). Of the 48,600 rows in the cross product, only 648 of them have the employee number equal in the two tables.

The ordering of the results has no meaning. You could add an ORDER BY clause to impose a particular order on the query.

The ON clause includes columns that are not included in the final result set.

---

☞ For more information, see “Explicit join conditions (the ON phrase)”  
[*ASA SQL User’s Guide*, page 269].

## Joining tables using key joins

Many common joins are between two tables related by a foreign key. The most common join restricts foreign key values to be equal to primary key values.

The KEY JOIN operator joins two tables based on foreign key relationship. In other words, Adaptive Server Anywhere generates an ON clause that equates the primary key column from one table with the foreign key column of the other.

The example in the previous section restricts foreign key values in the sales\_order table to be equal to the primary key values in the employee table.

```
SELECT employee.emp_lname,
       sales_order.id,
       sales_order.order_date
FROM sales_order JOIN employee
   ON sales_order.sales_rep = employee.emp_id
```

The query can be more simply expressed using a KEY JOIN:

```
SELECT employee.emp_lname,
       sales_order.id,
       sales_order.order_date
FROM sales_order KEY JOIN employee
```

KEY JOIN is just a shortcut for typing the ON clause; the two queries are identical. Key join is the default when you specify JOIN, but do not specify CROSS, NATURAL, KEY, or use an ON clause.

If you look at the diagram of the employee database, lines between tables represent foreign keys. You can use the KEY JOIN operator anywhere two tables are joined by a line in the diagram.

☞ For a diagram of the sample database, see [“The sample database” on page 46](#).

Joining more than two tables

Two or more tables can be joined using join operators. The following query uses four tables to list the total value of the orders placed by each customer. It connects the four tables customer, sales\_order, sales\_order\_items, and product using the single foreign-key relationships between each pair of these tables.

---

## ❖ List companies and the total value of their orders

1. In Interactive SQL, type the following in the SQL Statements pane and press F5 to execute the statement.

```
SELECT company_name,  
       SUM( sales_order_items.quantity *  
            product.unit_price) AS value  
FROM ( ( customer KEY JOIN sales_order )  
      KEY JOIN sales_order_items )  
      KEY JOIN product  
GROUP BY company_name
```

company_name	value
Bensoul's Boutique	1332
Bush Pro Shop	2940
Sterling & Co.	6804
Ocean Sports	3744
...	...

### Notes

Your result set may appear in a different order. There is no significance to the order of the rows in the result set.

The example uses the SUM operator, which is an aggregate function.

Aggregate functions work with GROUP BY clauses to return values for each row group. In this example, the sum of `sales_order_items.quantity * product.unit_price`—that is, the total amount of money paid per product type—is calculated for each `company_name`, thereby returning each company's sales.

The parentheses in the FROM clause help to clarify the order in which the joins are made.

☞ For more information on aggregate functions, see [“A first look at aggregate functions” on page 119](#).

☞ For more information on complex key joins, see [“Key joins” \[ASA SQL User's Guide, page 292\]](#).

## Joining tables using natural joins

The NATURAL JOIN operator joins two tables based on common column names. In other words, Adaptive Server Anywhere generates an ON clause that equates the common columns from each table.

### ❖ List all employees and their departments

1. In Interactive SQL, type the following in the SQL Statements pane and press F5 to execute the statement.

```
SELECT emp_lname, dept_name
FROM employee NATURAL JOIN department
```

emp_lname	dept_name
Whitney	R & D
Cobb	R & D
Breault	R & D
Shishov	R & D
Driscoll	R & D
...	...

Adaptive Server Anywhere looks at the two tables and determines that the only column name they have in common is dept\_id. The following ON CLAUSE is internally generated and used to perform the join:

```
FROM employee JOIN department
ON employee.dept_id = department.dept_id
```

NATURAL JOIN is just a shortcut for typing the ON clause; the two queries are identical.

### Errors using NATURAL JOIN

This join operator can cause problems by equating columns you may not intend to be equated. For example, the following query generates unwanted results:

```
SELECT *
FROM sales_order NATURAL JOIN customer
```

The result of this query has no rows. Adaptive Server Anywhere internally generates the following ON clause:

```
FROM sales_order JOIN customer
ON sales_order.id = customer.id
```

The id column in the sales\_order table is an ID number for the *order*. The id

---

column in the customer table is an ID number for the *customer*. None of them matched. Of course, even if a match were found, it would be a meaningless one.

☞ For more information, see “Natural joins” [*ASA SQL User’s Guide*, page 288].



## Joining tables using outer joins

In the previous examples, you created joins that returned rows only if they satisfied the join conditions. These joins are called **inner joins**, and are the default. Sometimes, you may wish to preserve all the rows in one table. To do this, you use an **outer join**.

You can specify a **right outer join**, which preserves all the rows in the right table; a **left outer join**, which preserves the left table; or a **full outer join**, which preserves all the rows in both tables.

### ❖ List all customers, and the dates of any orders they have placed

1. In Interactive SQL, type the following in the SQL Statements pane and press F5 to execute the statement.

```
SELECT lname, order_date, city
FROM customer KEY LEFT OUTER JOIN sales_order
WHERE customer.state = 'NY'
ORDER BY order_date
```

lname	order_date	city
Thompson	(NULL)	Bancroft
Reiser	1993-01-22	Rockwood
Clarke	1993-01-27	Rockwood
Mentary	1993-01-30	Rockland
...	...	...

The statement includes all customers, whether or not they have placed an order. If a particular customer has placed no orders, each column in the result that corresponds to order information contains NULL.

☞ For more information, see “Outer joins” [ASA SQL User’s Guide, page 274].



## CHAPTER 10

# Selecting Aggregate Data

### About this chapter

This chapter describes how to construct queries that tell you aggregate information. Examples of aggregate information are as follows:

- ◆ The total of all values in a column.
- ◆ The number of distinct entries in a column.
- ◆ The average value of entries in a column.

### Contents

Topic:	page
<a href="#">Summarizing data</a>	118
<a href="#">A first look at aggregate functions</a>	119
<a href="#">Applying aggregate functions to grouped data</a>	120
<a href="#">Restricting groups</a>	122

---

# Summarizing data

Some queries examine aspects of the data in your table that reflect properties of groups of rows rather than of individual rows. For example, you may wish to find the average amount of money that a customer pays for an order, or to see how many employees work for each department. For these types of tasks, you use **aggregate** functions and the GROUP BY clause.

## About this chapter

If you want to know...	Then see...
How to view summary information about an entire table	<a href="#">“A first look at aggregate functions” on page 119</a>
How to view summary information about groups in a table	<a href="#">“Applying aggregate functions to grouped data” on page 120</a>
How to view summary data about a restricted set of groups in a table	<a href="#">“Restricting groups” on page 122</a>

## A first look at aggregate functions

Aggregate functions return a single value for a set of rows. If there is no **GROUP BY** clause, an aggregate function returns a single value for all the rows that satisfy other aspects of the query.

### ❖ List the number of employees in the company

1. In Interactive SQL, type the following in the SQL Statements pane and press F5 to execute the statement.

```
SELECT COUNT( * )
FROM employee
```

---

**COUNT(\*)**

75

The result set consists of only one column, with title **COUNT( \* )**, and one row, which contains the total number of employees.

### ❖ List the number of employees in the company and the birth dates of the oldest and youngest employee

1. In Interactive SQL, type the following in the SQL Statements pane and press F5 to execute the statement.

```
SELECT COUNT(*), MIN(birth_date), MAX(birth_date)
FROM employee
```

COUNT(*)	MIN(employee.birth_date)	MAX(employee.birth_date)
75	1/2/1936	1/18/1973

The functions **COUNT**, **MIN**, and **MAX** are called **aggregate functions**, which are functions that summarize information. Other aggregate functions include statistical functions such as **AVG**, **STDDEV**, and **VARIANCE**. All but **COUNT** have the name of a column as a parameter.

☞ For more information, see “Aggregate functions” [ASA SQL Reference, page 84].

# Applying aggregate functions to grouped data

In addition to providing information about an entire table, aggregate functions can be used on groups of rows. The GROUP BY clause arranges rows into groups, and aggregate functions return a single value for each group of rows.

## Example

❖ **List the sales representatives and the number of orders each has taken**

1. In Interactive SQL, type the following in the SQL Statements pane and press F5 to execute the statement.

```
SELECT sales_rep, count( * )
FROM sales_order
GROUP BY sales_rep
ORDER BY sales_rep
```

sales_rep	count(*)
129	57
195	50
299	114
467	56
...	...

A GROUP BY clause tells Adaptive Server Anywhere to partition the set of all the rows that would otherwise be returned. All rows in each partition, or group, have the same values in the named column or columns. There is only one group for each unique value or set of values. In this case, all the rows in each group have the same sales\_rep value.

Aggregate functions such as COUNT are applied to the rows in each group. Thus, this result set displays the total number of rows in each group. The results of the query consist of one row for each sales rep ID number. Each row contains the sales rep ID, and the total number of sales orders for that sales representative.

Whenever GROUP BY is used, the resulting table has one row for each column or set of columns named in the GROUP BY clause.

👉 For more information, see “The GROUP BY clause: organizing query results into groups” [ASA SQL User’s Guide, page 237].

A common error with GROUP BY

A common error with GROUP BY is to try to get information that cannot

properly be put in a group. For example,

```
-- This query is incorrect
SELECT sales_rep, emp_lname, COUNT( * )
FROM sales_order KEY JOIN employee
GROUP BY sales_rep
```

gives the following error:

Function or column reference to 'emp\_lname' in the select list must also appear in a GROUP BY

An error is reported because Adaptive Server Anywhere cannot be sure that each of the result rows for an employee with a given ID all have the same last name.

To fix this error, add the column to the GROUP BY clause.

```
SELECT sales_rep, emp_lname, COUNT( * )
FROM sales_order KEY JOIN employee
GROUP BY sales_rep, emp_lname
ORDER BY sales_rep
```

If this is not appropriate, you can instead use an aggregate function to select only one value, as shown:

```
SELECT sales_rep, MAX( emp_lname ), COUNT( * )
FROM sales_order KEY JOIN employee
GROUP BY sales_rep
ORDER BY sales_rep
```

The MAX function chooses the maximum (last alphabetically) last name from the detail rows for each group. This statement is valid because there can be only one distinct maximum value. In this case, the same last name appears on every detail row within a group.

# Restricting groups

You have already seen how to restrict rows in a result set using the WHERE clause. You restrict the rows in groups using the HAVING clause.

❖ **List all sales representatives with more than 55 orders**

- 1. In Interactive SQL, type the following in the SQL Statements pane and press F5 to execute the statement.

```
SELECT sales_rep, count( * ) AS orders
FROM sales_order KEY JOIN employee
GROUP BY sales_rep
HAVING count( * ) > 55
ORDER BY orders DESC
```

sales_rep	orders
299	114
129	57
1142	57
467	56

**Order of clauses**  
A GROUP BY must always appear before a HAVING clause. If both are present, a WHERE clause must appear before a GROUP BY clause.

HAVING clauses and WHERE clauses can both be used in a single query. Conditions in the HAVING clause logically restrict the rows of the result only after the groups have been constructed. Criteria in the WHERE clause are logically evaluated before the groups are constructed, and so save time.

☞ For more information, see “The HAVING clause: selecting groups of data” [ASA SQL User’s Guide, page 242].

## Combining WHERE and HAVING clauses

Sometimes you can specify the same set of rows using either a WHERE clause or a HAVING clause. In such cases, one method is not more or less efficient than the other. The optimizer always automatically analyzes each statement you type and selects an efficient means of executing it. It is best to use the syntax that most clearly describes the desired result. In general, that means eliminating undesired rows in earlier clauses.

Example

To list all sales reps with more than 55 orders and an ID of more than 1000,



type the following statement.

```
SELECT sales_rep, count( * )  
FROM sales_order KEY JOIN employee  
WHERE sales_rep > 1000  
GROUP BY sales_rep  
HAVING count( * ) > 55  
ORDER BY sales_rep
```

The following statement produces the same results.

```
SELECT sales_rep, count( * )  
FROM sales_order KEY JOIN employee  
GROUP BY sales_rep  
HAVING count( * ) > 55 AND sales_rep > 1000  
ORDER BY sales_rep
```

Adaptive Server Anywhere detects that both statements describe the same result set, and so executes each efficiently.



## CHAPTER 11

# Selecting Data Using Subqueries

About this chapter

This chapter shows how to use the results of one query as part of another `SELECT` statement. This is a useful tool in building more complex and informative queries.

Contents

<b>Topic:</b>	<b>page</b>
<a href="#">Introducing subqueries</a>	126
<a href="#">Introduction</a>	127
<a href="#">Single-row and multiple-row subqueries</a>	129
<a href="#">Using subqueries instead of joins</a>	131

---

# Introducing subqueries

A relational database allows you to store related data in more than one table. The chapter [“Selecting Data from Multiple Tables” on page 105](#) outlines one way of extracting data from related tables. A second method involves **subqueries**—queries that appear in another query’s WHERE clause or HAVING clause. Subqueries make some queries easier to write than joins, and there are queries that cannot be written without using subqueries.

## About this chapter

If you want to know...	Then see...
When subqueries are used	<a href="#">“Introduction” on page 127</a>
How to compare column values to a single value returned by a subquery	<a href="#">“Single-row and multiple-row subqueries” on page 129</a>
How to use a subquery instead of a join	<a href="#">“Using subqueries instead of joins” on page 131</a>

## Introduction

Subqueries use the results of one query as part of another query. This section illustrates a situation where subqueries can be used by building a query that lists order items for products that are low in stock.

There are two queries involved in producing this list. This section first describes them separately, and then shows the single query that produces the same result.

### ❖ List all products for which there are less than 20 items in stock

1. In Interactive SQL, type the following in the SQL Statements pane and press F5 to execute the statement.

```
SELECT id, description, quantity
FROM product
WHERE quantity < 20
```

id	description	quantity
401	Wool cap	12

The query shows that only wool caps are low in stock.

### ❖ List all order items for wool caps

1. In Interactive SQL, type the following in the SQL Statements pane and press F5 to execute the statement.

```
SELECT *
FROM sales_order_items
WHERE prod_id = 401
ORDER BY ship_date DESC
```

id	line_id	prod_id	quantity	ship_date
2082	1	401	48	7/9/2001
2053	1	401	60	6/30/2001
2125	2	401	36	6/28/2001
2027	1	401	12	6/17/2001
...	...	...	...	...

This two-step process of identifying items low in stock and identifying orders for those items can be combined into a single query using subqueries.

---

❖ **List all order items for items that are low in stock**

1. In Interactive SQL, type the following in the SQL Statements pane and press F5 to execute the statement.

```
SELECT *
FROM sales_order_items
WHERE prod_id IN
    ( SELECT id
      FROM product
      WHERE quantity < 20 )
ORDER BY ship_date DESC
```

id	line_id	prod_id	quantity	ship_date
2082	1	401	48	7/9/2001
2053	1	401	60	6/30/2001
2125	2	401	36	6/28/2001
2027	1	401	12	6/17/2001
...	...	...	...	...

The subquery in the statement is the phrase enclosed in parentheses:

```
( SELECT id
  FROM product
  WHERE quantity < 20 )
```

The subquery makes a list of all values in the id column in the product table, satisfying the WHERE clause search condition.

The subquery returns a set of rows, but only a single column. The IN keyword treats each value as a member of a set and tests whether each row in the main query is a member of the set.

## Single-row and multiple-row subqueries

There are constraints on the number of rows and columns that a subquery may return. If you use IN, ANY, or ALL, the subquery may return several rows, but only one column. If you use other operators, the subquery must return a single value.

A multiple-row subquery

Two tables in the sample database are concerned with financial results. The `fin_code` table is a small table holding the different codes for financial data and their meanings:

To list the revenue items from the `fin_data` table, type the following:

```
SELECT *
FROM fin_data
WHERE fin_data.code IN
  ( SELECT fin_code.code
    FROM fin_code
    WHERE type = 'revenue' )
```

year	quarter	code	amount
1999	Q1	r1	1023
1999	Q2	r1	2033
1999	Q3	r1	2998
1999	Q4	r1	3014
2000	Q1	r1	3114

This example has used qualifiers to clearly identify the table to which the code column in each reference belongs. In this particular example, the qualifiers could have been omitted.

Two other keywords can be used as qualifiers for operators to allow them to work with multiple rows: ANY and ALL.

The following query is identical to the successful query above:

```
SELECT *
FROM fin_data
WHERE fin_data.code = ANY
  ( SELECT fin_code.code
    FROM fin_code
    WHERE type = 'revenue' )
```

While the `=ANY` condition is identical to the IN condition, ANY can also be used with inequalities such as `<` or `>` to give more flexible use of subqueries.

The ALL keyword is similar to the word ANY. For example, the following

---

query lists financial data that is not revenues:

```
SELECT *
FROM fin_data
WHERE fin_data.code <> ALL
    ( SELECT fin_code.code
      FROM fin_code
      WHERE type = 'revenue' )
```

This is equivalent to the following command using NOT IN:

```
SELECT *
FROM fin_data
WHERE fin_data.code NOT IN
    ( SELECT fin_code.code
      FROM fin_code
      WHERE type = 'revenue' )
```

A common error using subqueries

In general, subquery result sets are restricted to a single column. The following example does not make sense because Adaptive Server Anywhere would not know which column from fin\_code to compare to the fin\_data.code column.

```
-- this query is incorrect
SELECT *
FROM fin_data
WHERE fin_data.code IN
    ( SELECT fin_code.code, fin_code.type
      FROM fin_code
      WHERE type = 'revenue' )
```

Single-row subqueries

While subqueries used with an IN condition may return a set of rows, a subquery used with a comparison operator must return only one row. For example the following command results in an error since the subquery returns two rows:

```
-- this query is incorrect
SELECT *
FROM fin_data
WHERE fin_data.code =
    ( SELECT fin_code.code
      FROM fin_code
      WHERE type = 'revenue' )
```



## Using subqueries instead of joins

Suppose you need a chronological list of orders and the company that placed them, but would like the company name instead of their customer ID. You can get this result using a join as follows:

Using a join

To list the order id, date, and company name for each order since the beginning of 2001, type the following:

```
SELECT    sales_order.id,
          sales_order.order_date,
          customer.company_name
FROM sales_order
  KEY JOIN customer
WHERE order_date > '2001/01/01'
ORDER BY order_date
```

id	order_date	company_name
2473	1/4/2001	Peachtree Active Wear
2474	1/4/2001	Sampson & Sons
2106	1/5/2001	Salt & Pepper's
2475	1/5/2001	Cinnamon Rainbow's
2036	1/5/2001	Hermanns

☞ For more on joins, see [“Selecting Data from Multiple Tables” on page 105](#).

Using a subquery

The following statement obtains the same results using a subquery instead of a join:

```
SELECT sales_order.id,
       sales_order.order_date,
       ( SELECT company_name FROM customer
         WHERE customer.id = sales_order.cust_id )
FROM sales_order
WHERE order_date > '2001/01/01'
ORDER BY order_date
```

The subquery refers to the `cust_id` column in the `sales_order` table even though the `sales_order` table is not part of the subquery. Instead, the `sales_order.cust_id` column refers to the `sales_order` table in the main body of the statement. This is called an **outer reference**. Any subquery that contains an outer reference is called a **correlated subquery**.

A subquery can be used instead of a join whenever only one column is required from the other table. (Recall that subqueries can only return one

---

column.) In this example, you only needed the `company_name` column so the join could be changed into a subquery.

If the subquery might have no result, this method is called an outer join. The join in previous sections of the tutorial is more fully called an inner join.

#### Using an outer join

To list all customers in Washington state, together with their most recent order ID, type the following:

```
SELECT      company_name, state,
            ( SELECT MAX( id )
              FROM sales_order
              WHERE sales_order.cust_id = customer.id )
FROM customer
WHERE state = 'WA'
```

company_name	state	MAX(sales_order.id)
Custom Designs	WA	2547
It's a Hit!	WA	(NULL)

The It's a Hit! company placed no orders, and the subquery returns NULL for this customer. Companies who have not placed an order are not listed when inner joins are used.

You could also specify an outer join explicitly. In this case, a GROUP BY clause is also required.

```
SELECT company_name, state,
       MAX( sales_order.id )
FROM customer
      KEY LEFT OUTER JOIN sales_order
WHERE state = 'WA'
GROUP BY company_name, state
```

## CHAPTER 12

# Updating the Database

### About this chapter

This chapter describes how to make changes to the contents of your database. It includes descriptions of how to add rows, remove rows, and change the contents of rows, as well as how to make changes permanent or correct changes you have made.

### Contents

<b>Topic:</b>	<b>page</b>
<a href="#">Introduction</a>	134
<a href="#">Adding rows to a table</a>	135
<a href="#">Modifying rows in a table</a>	136
<a href="#">Deleting rows</a>	137
<a href="#">Grouping changes into transactions</a>	138
<a href="#">Integrity checking</a>	141

---

# Introduction

This chapter describes how to make changes to the data in your database. There are three basic operations:

- ◆ **Insert** You can add rows to tables to include new data.
- ◆ **Delete** You can delete rows in tables to remove data.
- ◆ **Update** You can modify existing rows in tables.

Each operation is carried out by executing a SQL statement.

## About this chapter

If you want to know...	Then see...
How to add rows to your table	<a href="#">“Adding rows to a table” on page 135</a>
How to update rows in your table	<a href="#">“Modifying rows in a table” on page 136</a>
How to delete rows in your table	<a href="#">“Deleting rows” on page 137</a>
How to implement transactions	<a href="#">“Grouping changes into transactions” on page 138</a>
How Adaptive Server Anywhere checks for errors in your data	<a href="#">“Integrity checking” on page 141</a>

## Adding rows to a table

Suppose that the company decides to sell a new product, a brown acrylic vest. This action requires you to add data to the product table of the sample database.

### ❖ Add a brown acrylic vest to the product table

1. In Interactive SQL, type the following in the SQL Statements pane and press F5 to execute the statement.

```
INSERT
INTO product ( id, name, description, size, color,
               quantity, unit_price )
VALUES ( 800, 'Vest', 'Acrylic Vest', 'Extra Large',
        'Brown', 42, 20.00 )
```

If you make a mistake and forget to specify one of the columns, Adaptive Server Anywhere reports an error.

You can also add new rows to database tables from the result set in Interactive SQL.

☞ For more information, see [“Editing table values in Interactive SQL” on page 76](#).

**NULL values in columns** The NULL value is a special value used to indicate that something is either not known or not applicable. Some columns are allowed to contain the NULL value, and others are not.

---

## Modifying rows in a table

In most databases, you need to update records that are already stored in the database. For example, the manager ID should change when employees are transferred between departments, as well as the department ID.

### ❖ Transfer employee #195 to department 400 in Interactive SQL

1. In Interactive SQL, type the following in the SQL Statements pane and press F5 to execute the statement.

```
UPDATE employee  
SET dept_id = 400, manager_id = 1576  
WHERE emp_id = 195
```

The statement carries out both updates at the same time for employee Marc Dill (employee ID 195).

Since Adaptive Server Anywhere updates all rows that satisfy the conditions of the WHERE clause, sometimes more than one row is updated at one time. For example, if a group of sales employees are transferred into marketing and have their dept\_id column updated, the following statement sets the manager\_id for all employees in the marketing department to 1576.

```
UPDATE employee  
SET manager_id = 1576  
WHERE dept_id = 400
```

For employees already in the marketing department, no change is made.

You can also modify rows from the result set in Interactive SQL.

☞ For more information, see [“Editing table values in Interactive SQL” on page 76](#).

## Deleting rows

Sometimes you will want to remove rows from a table. Suppose Rodrigo Guevara (employee ID 249) leaves the company. The following statement deletes Rodrigo Guevara from the employee table.

```
DELETE
FROM employee
WHERE emp_id = 249
```

With UPDATE and DELETE, the search condition can be as complicated as you need. For example, if the employee table is being reorganized, the following statement removes from the employee table all male employees hired between March 3, 1989 and March 3, 1990.

```
DELETE
FROM employee
WHERE sex = 'm'
      AND start_date BETWEEN '1988-03-03'
      AND '1989-03-03'
```

You can also delete rows from database tables from the Interactive SQL result set.

☞ For more information, see [“Editing table values in Interactive SQL” on page 76](#).

Since you have made changes to the database that you do not want to keep, you should undo the changes as follows:

```
ROLLBACK
```

---

## Grouping changes into transactions

Adaptive Server Anywhere expects you to group your commands into transactions. You commit a transaction to make changes to your database permanent. When you alter your data, your alterations are not made permanent right away. Instead, they are stored in your **transaction log** and are made permanent when you enter the COMMIT command.

Knowing which commands or actions signify the start or end of a transaction lets you take full advantage of transactions.

### Starting transactions

Transactions start with one of the following events:

- ◆ The first statement following a connection to a database.
- ◆ The first statement following the end of a transaction.

### Completing transactions

Transactions complete with one of the following events:

- ◆ A COMMIT statement makes the changes to the database permanent.
- ◆ A ROLLBACK statement undoes all the changes made by the transaction.
- ◆ A statement with a side effect of an automatic commit is executed: Database definition commands, such as ALTER, CREATE, COMMENT, and DROP all have the side effect of an automatic commit.
- ◆ A disconnection from a database performs an implicit rollback.

### Options in Interactive SQL

Interactive SQL provides you with two options which let you control when and how transactions end:

- ◆ If you set the option AUTO\_COMMIT to ON, Interactive SQL automatically commits your results following every successful statement and automatically performs a ROLLBACK after each failed statement.
- ◆ The setting of the option COMMIT\_ON\_EXIT controls what happens to uncommitted changes when you exit Interactive SQL. If this option is set to ON (the default), Interactive SQL does a COMMIT; otherwise it undoes your uncommitted changes with a ROLLBACK statement.

#### **If you are using a data source**

By default, ODBC operates in autocommit mode. Even if you have set the AUTO\_COMMIT option to OFF in Interactive SQL, ODBC's setting will override Interactive SQL's. You can change ODBC's setting using the SQL\_ATTR\_AUTOCOMMIT connection attribute. ODBC autocommit is independent of the CHAINED option.



Adaptive Server Anywhere also supports Transact-SQL commands, such as `BEGIN TRANSACTION`, for compatibility with Sybase Adaptive Server Enterprise.

☞ For further information, see “An overview of Transact-SQL support” [ASA *SQL User’s Guide*, page 440].

## Making changes permanent

The `COMMIT` statement makes all changes permanent.

You should use the `COMMIT` statement after groups of statements that make sense together. For example, if you want to transfer money from one customer’s account to another, you should add money to one customer’s account, then delete it from the other’s, and then commit, since in this case it does not make sense to leave your database with less or more money than it started with.

You can instruct Adaptive Server Anywhere to commit your changes automatically by setting the `AUTO_COMMIT` option to `ON`. This is an Interactive SQL option. When `AUTO_COMMIT` is set to `ON`, Adaptive Server Anywhere must update the database after every insert, update, and delete statement you make to it. This can slow down performance considerably. Therefore, it is a good idea to leave the `AUTO_COMMIT` option set to `OFF`.

### Use COMMIT with care

When trying the examples in this tutorial, be careful not to `COMMIT` any changes until you are sure that you want to change the database permanently.

☞ For more information about Interactive SQL options, see “Interactive SQL options” [ASA *Database Administration Guide*, page 571].

## Canceling changes

Any uncommitted change you make can be cancelled. SQL allows you to undo all of the changes you made since your last commit with the `ROLLBACK` statement.

The `ROLLBACK` statement undoes all changes you have made to the database since the last time you made changes permanent (see “[Making changes permanent](#)” on page 139).

☞ For more information on Interactive SQL options, see “Interactive SQL options” [ASA *Database Administration Guide*, page 571].

---

## Transactions and data recovery

Suppose that a system failure or power outage suddenly takes your database server down. Adaptive Server Anywhere is carefully designed to protect the integrity of your database in such circumstances. It provides you with a number of independent means of restoring your database. For example, it provides you with a **log file** which you may store on a separate drive so that should the system failure damage one drive, you still have a means of restoring your data. In addition, when you use a log file, Adaptive Server Anywhere does not have to update your database as frequently, which improves your database's performance.

Transaction processing allows the database server to identify states in which your data is in a consistent state. Transaction processing ensures that if, for any reason, a transaction is not successfully completed, then the entire transaction is undone, or rolled back. The database is left entirely unaffected by failed transactions.

Adaptive Server Anywhere's transaction processing ensures that the contents of a transaction are processed securely, even in the event of a system failure in the middle of a transaction.

☞ For a detailed description of data recovery mechanisms, see “Backup and Data Recovery” [*ASA Database Administration Guide*, page 337].

## Integrity checking

Adaptive Server Anywhere automatically checks for some common errors in your data.

### Inserting duplicate data

For example, suppose you attempt to create a department, but supply a dept\_id value that is already in use:

To do this, enter the command:

```
INSERT
INTO department ( dept_id, dept_name, dept_head_id )
VALUES ( 200, 'Eastern Sales', 902 )
```

The INSERT is rejected as it would make the primary key for the table not unique. Since dept\_id field is a primary key, duplicate values are not permitted.

### Inserting values that violate relationships

The following statement inserts a new row in the sales\_order table, but incorrectly supplies a sales\_rep ID that does not exist in the employee table.

```
INSERT
INTO sales_order ( id, cust_id, order_date,
sales_rep)
VALUES ( 2700, 186, '1995-10-19', 284 )
```

There is a one-to-many relationship between the employee table and the sales\_order table, with a join between the sales\_rep field of the sales\_order table and the emp\_id field of the employee table. Only after a record in the table containing the primary key for the join (the employee table) has been entered can a corresponding record on table containing the foreign key (the sales\_order table) be inserted.

#### Foreign keys

The primary key for the employee table is the employee ID number. The sales rep ID number in the sales\_rep table is a **foreign key** for the employee table, meaning that each sales rep number in the sales\_order table must match the employee ID number for some employee in the employee table.

When you try to add an order for sales rep 284 you get an error message:

```
No primary key value for foreign key 'ky_so_employee_id'
in table 'sales_order'
```

There isn't an employee in the employee table with that ID number. This prevents you from inserting orders without a valid sales rep ID. This kind of

---

validity checking is called **referential integrity** checking as it maintains the integrity of references among the tables in the database.

☞ For more information on primary and foreign keys, see “[Relations between tables](#)” on page 11.

## Errors on DELETE or UPDATE

Foreign key errors can also arise when doing update or delete operations. For example, suppose you try to remove the R&D department from the department table. The dept\_id field, being the primary key of the department table, constitutes the ONE side of a one-to-many relationship (the dept\_id field of the employee table is the corresponding foreign key, and hence forms the MANY side). A record on the *one* side of a relationship may not be deleted until all corresponding records on the MANY side are deleted.

```
DELETE
FROM department
WHERE dept_id = 100
```

Example: DELETE errors

An error is reported indicating that there are other records in the database that reference the R&D department, and the delete operation is not carried out.

primary key for row in table 'department' is referenced in another table

In order to remove the R&D department, you need to first get rid of all employees in that department:

```
DELETE
FROM employee
WHERE dept_id = 100
```

You can now perform the deletion of the R&D department.

You should cancel these changes to the database (for future use) by entering a ROLLBACK statement:

```
ROLLBACK WORK
```

All changes made since the last successful COMMIT WORK will be undone. If you have not done a COMMIT, then all changes since you started Interactive SQL will be undone.

Example: UPDATE errors

The same error message is generated if you perform an update operation that makes the database inconsistent.

For example, the following UPDATE statement causes an integrity error:

```
UPDATE department
SET dept_id = 600
WHERE dept_id = 100
```

In all of the above examples, the integrity of the database was checked as each command was executed. Any operation that would result in an inconsistent database is not performed.

Example: checking the integrity after the COMMIT WORK is complete

It is possible to configure the database so that the integrity is not checked until the COMMIT WORK is done. This is important if you want to change the value of a referenced primary key; for example, changing the R&D department's ID from 100 to 600 in the department and employee tables. In order to make these changes, the database has to be inconsistent in between the changes. In this case, you must configure the database to check only on commits.

☞ For more information, see “WAIT\_FOR\_COMMIT option [database]” [ASA Database Administration Guide, page 634].

You can also define foreign keys in such a way that they are automatically fixed. In the above example, if the foreign key from employee to department were defined with ON UPDATE CASCADE, then updating the department ID would automatically update the employee table.

In the above cases, there is no way to have an inconsistent database committed as permanent. Adaptive Server Anywhere also supports alternative actions if changes would render the database inconsistent.

☞ For more information, see the chapter “Ensuring Data Integrity” [ASA SQL User's Guide, page 75].



## CHAPTER 13

# System Tables

### About this chapter

This chapter describes the system tables, several special tables found in every Adaptive Server Anywhere database. These system tables describe all the tables and columns in the database. The database server automatically updates the system table as the database structure is changed.

### Contents

<b>Topic:</b>	<b>page</b>
<a href="#">The system tables</a>	146
<a href="#">The SYSCATALOG view</a>	147
<a href="#">The SYSCOLUMNS view</a>	148
<a href="#">Other system tables</a>	149

---

# The system tables

Adaptive Server Anywhere stores important information about your database in **system tables**. The data from tables can be viewed in the same way that the data from other tables can be viewed, but you can not update data from the system tables.

## About this chapter

If you want to know...	Then see...
Where to find a listing of all the tables in the database	<a href="#">“The SYSCATALOG view” on page 147</a>
Where to find information about the columns in a table	<a href="#">“The SYSCOLUMNS view” on page 148</a>
How to find other information about the tables in your database	<a href="#">“Other system tables” on page 149</a>



## The SYSCATALOG view

SYSCATALOG is a view that holds a more readable form of the system table SYSTABLE. It lists all the tables in the database. You can view the data from the SYSCATALOG view in the same way that you view the data from any other table in your database.

☞ For more information on viewing data, see [“Selecting a complete table” on page 90](#).

The owner of the system tables

The owner of the system tables and views is the special user ID **SYS**, and the owner of the company tables is **DBA**. In addition, there is a set of views owned by the special user ID **dbo**, which provide an emulation of the Sybase Adaptive Server Enterprise system catalog; these tables are not discussed in this section.

Recall that DBA is the user ID you used when connecting to the database from Interactive SQL. So far, you have simply typed the table names `employee` and `department`; SQL looked in SYSCATALOG for tables with those names created by DBA. In this example, by typing `SYS.SYSCATALOG`, you specified that SYSCATALOG was created by the user ID SYS. Note the similarity to the way column names are qualified, such as `employee.emp_id`.

Other columns in the system table

The other columns in this table contain other important information. For example, the column named **Ncols** is the number of columns in each table, and the column named **tabletype** identifies the table as a base table or a view.

---

## The SYSCOLUMNS view

Another important system table is a view called SYSCOLUMNS. This is a readable form of the system table SYSCOLUMN. It describes all the columns in all the tables in the database. To see the contents of a table, type the following command, in which *tablename* represents the name of the table whose columns you wish to list:

```
SELECT *
FROM sys.syscolumns
WHERE tname = tablename
```

For example,

```
SELECT *
FROM sys.syscolumns
WHERE tname = 'employee'
ORDER BY colno
```

This statement lists all the columns in the employee table. If you look at the columns to the right, you can see from the **Coltype** column that some columns in the employee table contain character information while others contain integer and date information.

You can also view columns in the Interactive SQL Lookup Table Name dialog, or in Sybase Central.

### ❖ To view the columns in a table or view ( Interactive SQL )

1. Invoke the list of tables by pressing F7.
2. Select the table whose columns you wish to view.
3. Click Show Columns.

### ❖ To view the columns in a table or view ( Sybase Central )

1. In the left pane of Sybase Central, select a database and open its Tables folder.
2. In the left pane, select a table.
3. In the right pane, click the Columns tab.

You can now view and modify the columns.

## Other system tables

There are several other system tables in the database that are not be described in the tutorial. You can find out their names by examining **SYS.SYSCATALOG** and looking at them if you want.

☞ For a full description of each of the system tables, see “System Tables” [ASA *SQL Reference*, page 611].



## CHAPTER 14

# Microsoft Visual Basic Quick Start

About this chapter

This chapter describes how to develop a simple database application using Adaptive Server Anywhere and Microsoft Visual Basic.

Contents

<b>Topic:</b>	<b>page</b>
<a href="#">Tutorial: Developing a Visual Basic application</a>	<a href="#">152</a>

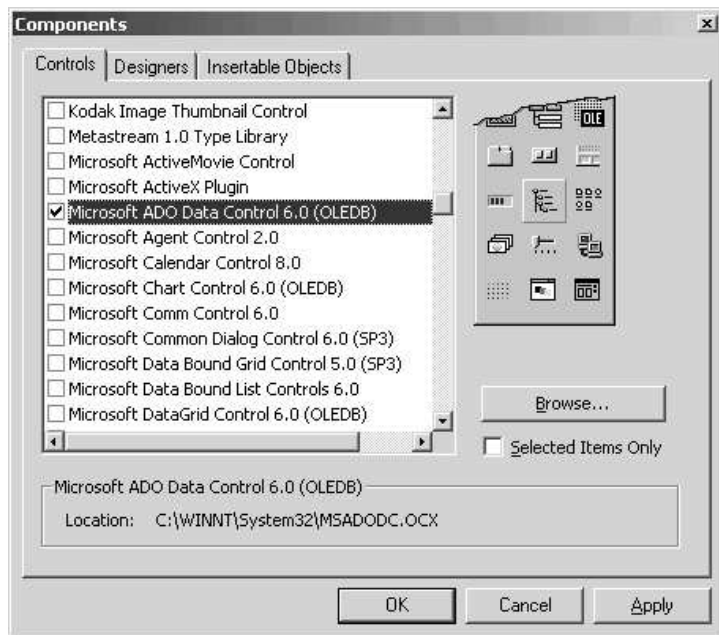
# Tutorial: Developing a Visual Basic application

This brief tutorial is based on Visual Basic 6.0. The complete application can be found in the Visual Basic project *Samples\ASA\VBStarter\ASASTarter.vbp*.

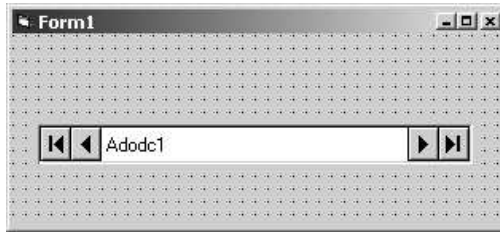
Visual Basic provides several data access technologies. In this tutorial, we use the Microsoft ADO Data Control with the Adaptive Server Anywhere OLE DB provider to access the Adaptive Server Anywhere sample database from Visual Basic.

## ❖ To develop a database application with Visual Basic

1. Start Visual Basic, choosing a Standard Executable project.
2. Add the Microsoft ADO Data Control 6.0 to your tool palette:
  - ◆ From the Project menu, choose Components.
  - ◆ Select the Microsoft ADO Data Control 6.0 component from the list.



- ◆ Click OK to add the control to the palette.
3. Add the ADO Data Control to the form, as follows:

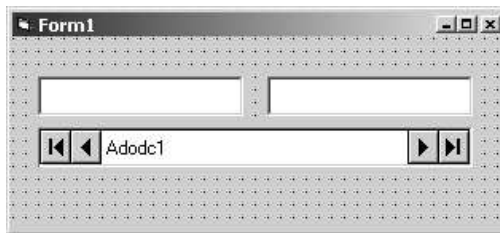


4. Configure the ADO Data Control:

Property	Value
ConnectionString	Provider=ASAPROV;DSN=ASA 9.0 Sample
CursorLocation	2 - asUseServer
CursorType	1 - adOpenKeyset
RecordSource	SELECT * FROM EMPLOYEE

The ConnectionString uses the Adaptive Server Anywhere OLE DB Provider (ASAProv) to connect to the ASA 9.0 Sample data source. The cursor settings take advantage of Adaptive Server Anywhere cursors rather than using the client-side cursors.

5. Add two text boxes to the form, as follows:



6. Bind the text boxes to the ADO Data Control:

- ◆ Set the DataSource property for each to be **Adodc1**.
- ◆ Set the DataField property for the left-hand text box to **emp\_fname**, which is the column holding the employee's first name.
- ◆ Set the DataField property for the right-hand text box to **emp\_lname**, which is the column holding the employee's last name.

7. Save the project.

8. Run the sample:

- 
- ◆ Choose Run ► Start to run the application.

The application connects to the Adaptive Server Anywhere sample database and puts the name of the first employee in the text boxes, as follows:



- ◆ You can use the buttons on the ADO Data Control to scroll through the rows of the result set.

You have now created a simple Visual Basic application that works with Adaptive Server Anywhere.



## CHAPTER 15

# Glossary

Adaptive Server Anywhere (ASA)	The relational database server component of SQL Anywhere Studio, intended for use in mobile and embedded environments or as a server for small and medium-sized businesses.
article	<p>In SQL Remote or MobiLink, an article is a database object that represents a whole table, or a subset of the columns and rows in a table. Articles are grouped together in a publication.</p> <p>☞ See also: <a href="#">“replication” on page 166</a>, <a href="#">“publication” on page 165</a>.</p>
base table	<p>A permanent table for data. Tables are sometimes called <b>base tables</b> to distinguish them from temporary tables and views.</p> <p>☞ See also: <a href="#">“temporary table” on page 169</a>, <a href="#">“view” on page 170</a>.</p>
business rule	<p>A guideline based on real-world requirements. Business rules are typically implemented through check constraints, user-defined data types, and the appropriate use of transactions.</p> <p>☞ See also: <a href="#">“constraint” on page 157</a>, <a href="#">“user-defined data type” on page 170</a>.</p>
check constraint	<p>A restriction that enforces specified conditions on a column or set of columns.</p> <p>☞ See also: <a href="#">“constraint” on page 157</a>, <a href="#">“foreign key constraint” on page 159</a>, <a href="#">“primary key constraint” on page 164</a>, <a href="#">“unique constraint” on page 170</a>.</p>
checkpoint	The point at which all changes to the database are saved to the database file. At other times, committed changes are saved only to the transaction log.
client/server	A software architecture where one application (the client) obtains information from and sends information to another application (the server). The two applications often reside on different computers connected by a network.
collation	A combination of a character set and a sort order that defines the properties of text in the database. For Adaptive Server Anywhere databases, the default collation is determined by the operating system and language on which the server is running; for example, the default collation on English Windows






---

	systems is 1252LATIN1. A collation, also called a collating sequence, is used for comparing and sorting strings.
command file	A text file containing SQL statements. Command files can be built manually, or they can be built automatically by database utilities. The dbunload utility, for example, creates a command file consisting of the SQL statements necessary to recreate a given database.
communication stream	In MobiLink, the network protocol used for communication between the MobiLink client and the MobiLink synchronization server.
compressed database file	A database file that has been compressed to a smaller physical size using the dbshrink utility. Compressed databases are read-only. To make changes to a compressed database file, you must use an associated write file. You can expand compressed database files into normal database files using the dbexpand utility.
concurrency	<p>The simultaneous execution of two or more independent, and possibly competing, processes. Adaptive Server Anywhere automatically uses locking to isolate transactions and ensure that each concurrent application sees a consistent set of data.</p> <p>☞ See also: <a href="#">“transaction” on page 169</a>, <a href="#">“lock” on page 162</a>, <a href="#">“isolation level” on page 161</a>.</p>
conflict trigger	<p>In SQL Remote replication, a trigger that fires when an update conflict is detected, before the update is applied. Conflict triggers are fired when the values in the VERIFY clause of an UPDATE statement fail to match the current values in the database.</p> <p>☞ See also: <a href="#">“replication” on page 166</a>, <a href="#">“trigger” on page 170</a>.</p>
connection ID	<p>A unique number that identifies a given connection between a client application and the database. You can determine the current connection ID using the following SQL statement:</p> <pre>SELECT connection_property( 'Number' )</pre>
connection profile	A set of parameters that are required to connect to a database, such as user name, password, and server name, that is stored and used as a convenience.
consolidated database	<p>In database replication, a database that stores the master copy of the data. The consolidated database contains all of the data, while remote databases usually contain only subsets of the data. In case of conflict or discrepancy, the consolidated database is considered to have the primary copy of all data.</p> <p>In MobiLink, the consolidated database can be Oracle, IBM DB2, Microsoft SQL Server, Adaptive Server Anywhere, or Adaptive Server Enterprise.</p> <p>☞ See also: <a href="#">“replication” on page 166</a>.</p>







constraint	<p>A restriction on the values contained in a particular database object, such as a table or column. For example, a column may have a uniqueness constraint, which requires that all values in the column be different. A table may have a foreign key constraint, which specifies how the information in the table relates to data in some other table.</p> <p>☞ See also: <a href="#">“check constraint” on page 155</a>, <a href="#">“foreign key constraint” on page 159</a>, <a href="#">“primary key constraint” on page 164</a>, <a href="#">“unique constraint” on page 170</a>.</p>
contention	<p>The act of competing for resources. For example, in database terms, two or more users trying to edit the same row of a database contend for the rights to edit that row.</p>
correlation name	<p>The name of a table or view that is used in the FROM clause of a query—either its original name, or an alias that is defined in the FROM clause.</p>
cursor	<p>A named linkage to a result set, used to access and update rows from a programming interface. In Adaptive Server Anywhere, cursors support forward and backward movement through the query results. Cursors consist of two parts: the cursor result set, typically defined by a SELECT statement; and the cursor position.</p> <p>☞ See also: <a href="#">“cursor result set” on page 157</a>, <a href="#">“cursor position” on page 157</a>.</p>
cursor position	<p>A pointer to one row within the cursor result set.</p> <p>☞ See also: <a href="#">“cursor” on page 157</a>, <a href="#">“cursor result set” on page 157</a>.</p>
cursor result set	<p>The set of rows resulting from a query that is associated with a cursor.</p> <p>☞ See also: <a href="#">“cursor” on page 157</a>, <a href="#">“cursor position” on page 157</a>.</p>
data definition language (DDL)	<p>The subset of SQL statements for modeling the structure of a database. DDL statements create, modify, and remove database objects, including users.</p>
data type	<p>The format of data, such as CHAR or NUMERIC. In the ANSI SQL standard, data types can also include a restriction on size, character set, and collation.</p> <p>☞ See also: <a href="#">“user-defined data type” on page 170</a>.</p>
data manipulation language (DML)	<p>The subset of SQL statements for retrieving and updating the contents of a database.</p>
database	<p>A collection of tables that are related by primary and foreign keys. The tables hold the information in the database. The tables and keys together define the structure of the database. A database-management system accesses this information.</p>

---

	<p>☞ See also: <a href="#">“foreign key” on page 159</a>, <a href="#">“primary key” on page 164</a>, <a href="#">“database-management system (DBMS)” on page 158</a>, <a href="#">“relational database-management system (RDBMS)” on page 166</a>.</p>
database administrator (DBA)	The user with the permissions required to maintain the database. The DBA is generally responsible for all changes to a database schema, and for managing users and user groups. The role of database administrator is automatically built into databases as user ID DBA with password SQL.
database connection	A communication channel between a client application and the database. A valid user ID and password are required to establish a connection. The privileges granted to the user ID determine the actions that can be carried out during the connection.
database file	<p>A database is held in one or more database files. There is an initial file, and subsequent files are called dbspaces. Each table, including its indexes, must be contained within a single database file.</p> <p>☞ See also: <a href="#">“dbspace” on page 158</a>.</p>
database-management system (DBMS)	<p>A collection of programs that allow you to create and use databases.</p> <p>☞ See also: <a href="#">“relational database-management system (RDBMS)” on page 166</a>.</p>
database name	<p>The name given to a database when it is loaded by a server. The default database name is the root of the initial database file.</p> <p>☞ See also: <a href="#">“database file” on page 158</a>.</p>
database object	A component of a database that contains or receives information. Tables, indexes, views, procedures, and triggers are database objects.
database owner (dbo)	<p>A special user that owns the system objects not owned by SYS.</p> <p>☞ See also: <a href="#">“database administrator (DBA)” on page 158</a>, <a href="#">“SYS” on page 169</a>.</p>
database server	A computer program that regulates all access to information in a database. Adaptive Server Anywhere provides two types of servers: network servers and personal servers.
DBA authority	<p>The level of permission that enables a user to carry out administrative activity in the database. The DBA user has DBA authority by default.</p> <p>☞ See also: <a href="#">“database administrator (DBA)” on page 158</a>.</p>
dbspace	An additional database file that creates more space for data. A database can be held in up to 13 separate files (an initial file and 12 dbspaces). Each table, together with its indexes, must be contained in a single database file. The SQL command CREATE DBSPACE adds a new file to the database.

	 See also: <a href="#">“database file” on page 158</a> .
download	The stage in synchronization where data is transferred from the consolidated database to a remote database.
embedded SQL	A programming interface for C programs. Adaptive Server Anywhere embedded SQL is an implementation of the ANSI and IBM standard.
external login	An alternate login name and password used when communicating with a remote server. By default, Adaptive Server Anywhere uses the names and passwords of its clients whenever it connects to a remote server on behalf of those clients. However, this default can be overridden by creating external logins. External logins are alternate login names and passwords used when communicating with a remote server.
extraction	<p>In SQL Remote replication, the act of unloading the appropriate structure and data from the consolidated database. This information is used to initialize the remote database.</p> <p>In MobiLink synchronization, the act of unloading the appropriate structure and data from a reference database.</p> <p> See also: <a href="#">“replication” on page 166</a>.</p>
failover	Switching to a redundant or standby server, system, or network on failure or unplanned termination of the active server, system, or network. Failover happens automatically, and is often built-in to continuously available systems.
FILE	<p>In SQL Remote replication, a message system that uses shared files for exchanging replication messages. This is useful for testing and for installations without an explicit message-transport system (such as MAPI).</p> <p> See also: <a href="#">“replication” on page 166</a>, <a href="#">“MAPI” on page 162</a>.</p>
file-based download	In MobiLink, a way to synchronize data in which downloads are distributed as files, allowing offline distribution of synchronization changes.
foreign key	<p>One or more columns in a table that duplicate the primary key values in another table. Foreign keys establish relationships between tables.</p> <p> See also: <a href="#">“primary key” on page 164</a>, <a href="#">“foreign table” on page 159</a>.</p>
foreign key constraint	<p>A restriction on a column or set of columns that specifies how the data in the table relates to the data in some other table. Imposing a foreign key constraint on a set of columns makes those columns the foreign key.</p> <p> See also: <a href="#">“constraint” on page 157</a>, <a href="#">“check constraint” on page 155</a>, <a href="#">“primary key constraint” on page 164</a>, <a href="#">“unique constraint” on page 170</a>.</p>
foreign table	The table containing the foreign key.

---

	 See also: <a href="#">“foreign key” on page 159.</a>
full backup	A backup of the entire database, and optionally, the transaction log. A full backup contains all the information in the database and thus provides protection in the event of a system or media failure.   See also: <a href="#">“incremental backup” on page 160.</a>
generated join condition	A restriction on join results based on the keyword KEY or NATURAL. For a natural join, the generated join condition is based on common column names in the two tables. For a key join, the condition is based on a foreign key relationship between the two tables.   See also: <a href="#">“join” on page 161</a> , <a href="#">“join condition” on page 161.</a>
global temporary table	A type of temporary table for which data definitions are visible to all users until explicitly dropped. Global temporary tables let each user open their own identical instance of a table. By default, rows are deleted on commit, and rows are always deleted when the connection is ended.   See also: <a href="#">“temporary table” on page 169</a> , <a href="#">“local temporary table” on page 161.</a>
grant option	The level of permission that allows a user to grant permissions to other users.
identifier	A string of characters used to reference a database object, such as a table or column. An identifier may contain any character from A through Z, a through z, 0 through 9, underscore (_), at sign (@), number sign (#), or dollar sign (\$).
incremental backup	A backup of the transaction log only, typically used between full backups.   See also: <a href="#">“transaction log” on page 169.</a>
index	A sorted set of keys and pointers associated with one or more columns in a base table. An index on one or more columns of a table can improve performance.
InfoMaker	A reporting and data maintenance tool that lets you create sophisticated forms, reports, graphs, cross-tabs, and tables, as well as applications that use these reports as building blocks.
inner join	A join in which rows appear in the result set only if both tables satisfy the join condition. Inner joins are the default.   See also: <a href="#">“join” on page 161</a> , <a href="#">“outer join” on page 163.</a>
integrated login	A login feature that allows the same single user ID and password to be used for operating system logins, network logins, and database connections.
integrity	Adherence to rules that ensure that data is correct and accurate, and that the

relational structure of the database is intact.

☞ See also: [“referential integrity” on page 165](#).

Interactive SQL	<p>An Adaptive Server Anywhere application that allows you to query and alter data in your database, and modify the structure of your database.</p> <p>Interactive SQL provides a pane for you to enter SQL statements, as well as panes that display information about how the query was processed and the result set.</p>
isolation level	<p>The degree to which operations in one transaction are visible to operations in other concurrent transactions. There are four isolation levels, numbered 0 through 3. Level 3 provides the highest level of isolation. Level 0 is the default setting.</p>
JAR file	<p>Java archive file. A compressed file format consisting of a collection of one or more packages used for Java applications. It includes all the resources necessary to install and run a Java program in a single compressed file.</p>
Java class	<p>The main structural unit of code in Java. It is a collection of procedures and variables grouped together because they all relate to a specific, identifiable category.</p>
jConnect	<p>A Java implementation of the JavaSoft JDBC standard. It provides Java developers with native database access in multi-tier and heterogeneous environments.</p> <p>☞ See also: <a href="#">“JDBC” on page 161</a>.</p>
JDBC	<p>Java Database Connectivity. A SQL-language programming interface that allows Java applications to access relational data.</p>
join	<p>A basic operation in a relational system that links the rows in two or more tables by comparing the values in specified columns.</p>
join condition	<p>A restriction that affects join results. You specify a join condition by inserting an ON clause or WHERE clause immediately after the join. In the case of natural and key joins, Adaptive Server Anywhere generates a join condition.</p> <p>☞ See also: <a href="#">“join” on page 161</a>, <a href="#">“generated join condition” on page 160</a>.</p>
join type	<p>Adaptive Server Anywhere provides four types of joins: cross join, key join, natural join, and joins using an ON clause.</p> <p>☞ See also: <a href="#">“join” on page 161</a>.</p>
local temporary table	<p>A type of temporary table that exists only for the duration of a compound statement or until the end of the connection. Local temporary tables are useful when you need to load a set of data only once. By default, rows are</p>

---

deleted on commit.

☞ See also: [“temporary table” on page 169](#), [“global temporary table” on page 160](#).

lock

A concurrency control mechanism that protects the integrity of data during the simultaneous execution of multiple transactions. Adaptive Server Anywhere automatically applies locks to prevent two connections from changing the same data at the same time, and to prevent other connections from reading data that is in the process of being changed.

You control locking by setting the isolation level.

☞ See also: [“isolation level” on page 161](#), [“concurrency” on page 156](#), [“integrity” on page 160](#).

log file

A log of transactions maintained by Adaptive Server Anywhere. The log file is used to ensure that the database is recoverable in the event of a system or media failure, to improve database performance, and to allow data replication using SQL Remote.

☞ See also: [“transaction log” on page 169](#), [“transaction log mirror” on page 169](#), [“full backup” on page 160](#).

LTM

Log Transfer Manager. See [“Replication Agent” on page 166](#).

MAPI

Microsoft’s Messaging Application Programming Interface. A message system used in several popular e-mail systems such as Microsoft Mail.

message system

In SQL Remote replication, a protocol for exchanging messages between the consolidated database and a remote database. Adaptive Server Anywhere includes support for the following message systems: FILE, MAPI, FTP, SMTP, and VIM.

☞ See also: [“replication” on page 166](#), [“FILE” on page 159](#), [“MAPI” on page 162](#).

message type

In SQL Remote replication, a database object that specifies how remote users communicate with the publisher of a consolidated database. A consolidated database may have several message types defined for it; this allows different remote users to communicate with it using different message systems.

☞ See also: [“replication” on page 166](#), [“consolidated database” on page 156](#), [“MAPI” on page 162](#).

metadata

Data about data. Metadata describes the nature and content of other data.

☞ See also: [“schema” on page 167](#).

MobiLink

A session-based synchronization technology designed to synchronize UltraLite and Adaptive Server Anywhere databases with many



industry-standard SQL database-management systems from Sybase and other vendors.

☞ See also: [“UltraLite” on page 170.](#)

MobiLink client	There are two kinds of MobiLink clients. For Adaptive Server Anywhere remote databases, the MobiLink client is the dbmlsync command line utility. For UltraLite remote databases, the MobiLink client is built in to the UltraLite runtime library.
MobiLink user	A MobiLink user is a name that uniquely identifies a MobiLink remote database in the synchronization system. The client supplies this name and, optionally, an associated password when it connects to the MobiLink synchronization server. MobiLink user names are entirely independent of database user names.
NetWare	A widely-used network operating system defined by Novell. NetWare generally employs the IPX/SPX protocol, although the TCP/IP protocol may also be used.
network server	A database server that accepts connections from computers sharing a common network.  ☞ See also: <a href="#">“personal server” on page 164.</a>
normalization	The refinement of a database structure to eliminate redundancy and improve organization according to rules based on relational database theory.
object tree	In Sybase Central, the hierarchy of database objects. The top level of the object tree shows all products that your version of Sybase Central supports. Each product expands to reveal its own sub-tree of objects.  ☞ See also: <a href="#">“Sybase Central” on page 168.</a>
ODBC	Open Database Connectivity. A standard Windows interface to database-management systems. ODBC is one of several interfaces supported by Adaptive Server Anywhere.
ODBC Administrator	A Microsoft program included with Windows operating systems for setting up ODBC data sources.
ODBC data source	A specification of the data a user wants to access via ODBC, and the information needed to get to that data.
outer join	A join that preserves all the rows in a table. Adaptive Server Anywhere supports left, right, and full outer joins. A left outer join preserves the rows in the table to the left of the join operator, and returns a null when a row in the right table does not satisfy the join condition. A full outer join preserves all the rows from both tables.

---

	☞ See also: <a href="#">“join” on page 161</a> , <a href="#">“inner join” on page 160</a> .
package	In Java, a collection of sets of related classes.
passthrough	In SQL Remote replication, a mode by which the publisher of the consolidated database can directly change remote databases with SQL statements. Passthrough is set up for specific remotes. In normal passthrough mode, all database changes made at the consolidated database are passed through to the selected remote databases. In passthrough only mode, the changes are made at the remote databases, but not at the consolidated database.
performance statistic	A value reflecting the performance of the database system. The CURRREAD statistic, for example, represents the number of file reads issued by the engine that have not yet completed.
personal server	A database server that runs on the same computer as the client application. A personal database server is typically used by a single user on a single computer, but it can support several concurrent connections from that user.
plug-in module	In Sybase Central, a way to access and administer a product. Plug-ins are usually installed and registered automatically with Sybase Central when you install the respective product. Typically, a plug-in appears as a top-level container, in the Sybase Central main window, using the name of the product itself; for example, Adaptive Server Anywhere.  ☞ See also: <a href="#">“Sybase Central” on page 168</a> .
PowerDesigner	A database modeling application. PowerDesigner provides a structured approach to designing a database or data warehouse.
PowerDynamo	A Sybase product for building and managing a Web application linked to a database.
PowerJ	A Sybase product for developing Java applications.
predicate	A conditional expression that is optionally combined with the logical operators AND and OR to make up the set of conditions in a WHERE or HAVING clause. In SQL, a predicate that evaluates to UNKNOWN is interpreted as FALSE.
primary key	A column or list of columns whose values uniquely identify every row in the table.  ☞ See also <a href="#">“foreign key” on page 159</a> .
primary key constraint	A uniqueness constraint on the primary key columns. A table can have only one primary key constraint.  ☞ See also: <a href="#">“constraint” on page 157</a> , <a href="#">“check constraint” on page 155</a> ,

[“foreign key constraint” on page 159](#), [“unique constraint” on page 170](#), [“integrity” on page 160](#).

primary table	The table containing the primary key in a foreign key relationship.
proxy table	<p>A local table containing metadata used to access a table on a remote database server as if it were a local table.</p> <p>☞ See also: <a href="#">“metadata” on page 162</a>.</p>
publication	<p>In SQL Remote or MobiLink, a database object that identifies replicated data. In MobiLink, publications exist only on the clients. A publication consists of articles. Periodically, the changes made to each publication are replicated to all subscribers to that publication. SQL Remote users can receive a publication by subscribing to it. MobiLink users can synchronize a publication by creating a synchronization subscription to it.</p> <p>☞ See also: <a href="#">“replication” on page 166</a>, <a href="#">“article” on page 155</a>, <a href="#">“publication update” on page 165</a>.</p>
publication update	<p>In SQL Remote replication, a list of changes made to one or more publications in one database. A publication update is sent periodically as part of a replication message to the remote database(s).</p> <p>☞ See also: <a href="#">“replication” on page 166</a>, <a href="#">“publication” on page 165</a>.</p>
publisher	<p>In SQL Remote replication, the single user in a database who can exchange replication messages with other replicating databases.</p> <p>☞ See also: <a href="#">“replication” on page 166</a>.</p>
query	<p>A SQL statement or group of SQL statements that access and/or manipulate data in a database.</p> <p>☞ See also: <a href="#">“SQL” on page 168</a>.</p>
Redirector	A web server plug-in that routes requests and responses between a client and the MobiLink synchronization server. This plug-in also implements load-balancing and failover mechanisms.
reference database	In MobiLink, an Adaptive Server Anywhere database used in the development of UltraLite clients. You can use a single Adaptive Server Anywhere database as both reference and consolidated database during development. Databases made with other products cannot be used as reference databases.
referential integrity	Adherence to rules governing data consistency, specifically the relationships between the primary and foreign key values in different tables. To have referential integrity, the values in each foreign key must correspond to the primary key values of a row in the referenced table.

---

	<p>☞ See also: <a href="#">“primary key” on page 164</a>, <a href="#">“foreign key” on page 159</a>.</p>
relational database-management system (RDBMS)	<p>A type of database-management system that stores data in the form of related tables.</p> <p>☞ See also: <a href="#">“database-management system (DBMS)” on page 158</a>.</p>
remote database	<p>In SQL Remote replication or MobiLink synchronization, a database that exchanges data with a consolidated database. Remote databases may share all or some of the data in the consolidated database.</p> <p>☞ See also: <a href="#">“replication” on page 166</a>, <a href="#">“consolidated database” on page 156</a>.</p>
remote DBA authority	<p>In SQL Remote, a level of permission required by the Message Agent. In MobiLink, a level of permission required by the Adaptive Server Anywhere synchronization client (<i>dbmsync</i>). When the Message Agent or synchronization client connects as a user who has this authority, it has full DBA access. The user ID has no additional permissions when not connected through the Message Agent or synchronization client.</p> <p>☞ See also: <a href="#">“DBA authority” on page 158</a>.</p>
remote permission	<p>In SQL Remote replication, the permission to exchange replication messages with the publishing database. Granting remote permissions to a user makes that user a remote user. You must specify a message type, an appropriate remote address, and a replication frequency. In general terms, remote permissions can also refer to any user involved in SQL Remote replication (for example, the consolidated publisher and remote publisher).</p> <p>☞ See also: <a href="#">“replication” on page 166</a>.</p>
remote user	<p>In SQL Remote replication, a database user in the consolidated database that has been granted remote permissions and is associated with one particular remote database in the replication setup. To create a remote user, an ordinary database user is granted remote permissions. Doing so not only identifies the existence of a particular remote database, but also specifies the message type and address with which to communicate with that particular remote site.</p> <p>When remote databases are created by means of extraction from a consolidated database, each remote user in the consolidated database becomes the publisher of the data in one particular remote database.</p> <p>☞ See also: <a href="#">“SQL Remote” on page 168</a>, <a href="#">“consolidated database” on page 156</a>, <a href="#">“publisher” on page 165</a>.</p>
replication	<p>The sharing of data among physically distinct databases. Sybase has three replication technologies: MobiLink, SQL Remote, and Replication Server.</p>
Replication Agent	<p>In Replication Server, the program, also called Log Transfer Manager</p>

	(LTM), that reads a database transaction log and sends committed changes to Replication Server.
replication frequency	<p>In SQL Remote replication, a setting for each remote user that determines how often the publisher's message agent should send replication messages to that remote user.</p> <p>☞ See also: <a href="#">“replication” on page 166</a>, <a href="#">“remote user” on page 166</a>.</p>
replication message	<p>In SQL Remote or Replication Server, a communication sent between a publishing database and a subscribing database. Messages contain data, passthrough statements, and information required by the replication system.</p> <p>☞ See also: <a href="#">“passthrough” on page 164</a>, <a href="#">“replication” on page 166</a>, <a href="#">“publication update” on page 165</a>.</p>
Replication Server	<p>A Sybase connection-based replication technology that works with Adaptive Server Anywhere and Adaptive Server Enterprise. It is intended for near-real time replication between a small number of databases.</p>
role	<p>In conceptual database modeling, a verb or phrase that describes a relationship from one point of view. You can describe each relationship with two roles. Examples of roles are “contains” and “is a member of.”</p>
role name	<p>The name of a foreign key. This is called a role name because it names the relationship between the foreign table and primary table. By default, the role name is the table name, unless another foreign key is already using that name, in which case the default role name is the table name followed by a three-digit unique number. You can also create the name yourself.</p> <p>☞ See also: <a href="#">“foreign key” on page 159</a>.</p>
rollback log	<p>A record of the changes made during each uncommitted transaction. In the event of a ROLLBACK request or a system failure, uncommitted transactions are reversed out of the database, returning the database to its former state. Each transaction has a separate rollback log, which is deleted when the transaction is complete.</p> <p>☞ See also: <a href="#">“transaction” on page 169</a>.</p>
row-level trigger	<p>A trigger that executes once for each row that is changed.</p> <p>☞ See also: <a href="#">“trigger” on page 170</a>, <a href="#">“statement-level trigger” on page 168</a>.</p>
schema	<p>The structure of a database, including tables, columns, and indexes, and the relationships between them.</p>
scripts	<p>In MobiLink, code written to handle MobiLink events. Scripts programmatically control data exchange to meet business needs.</p>
server-initiated synchronization	<p>A way to initiate MobiLink synchronization programmatically from the</p>

---


	consolidated database.
service	In Windows operating systems, a way of running applications when the user ID running the application is not logged on.
session-based synchronization	A type of synchronization where synchronization results in consistent data representation across both the consolidated and remote databases. MobiLink is session-based.
SQL	The language used to communicate with relational databases. ANSI has defined standards for SQL, the latest of which is SQL-99 (also called SQL3). SQL stands, unofficially, for Structured Query Language.
SQL Remote	A message-based replication technology for two-way replication between consolidated and remote databases. The consolidated database must be Adaptive Server Anywhere or Adaptive Server Enterprise. The remote databases must be Adaptive Server Anywhere.
SQL statement	<p>A string containing SQL keywords designed for passing instructions to a DBMS.</p> <p>☞ See also: <a href="#">“schema” on page 167</a>, <a href="#">“SQL” on page 168</a>, <a href="#">“database-management system (DBMS)” on page 158</a>.</p>
statement-level trigger	<p>A trigger that executes after the entire triggering statement is completed.</p> <p>☞ See also: <a href="#">“trigger” on page 170</a>, <a href="#">“row-level trigger” on page 167</a>.</p>
stored procedure	A program comprised of a sequence of SQL instructions, stored in the database and used to perform a particular task.
subquery	<p>A SELECT statement that is nested inside another SELECT, INSERT, UPDATE, or DELETE statement, or another subquery.</p> <p>There are two types of subquery: correlated and nested.</p>
subscription	<p>In SQL Remote replication, a link between a publication and a remote user, allowing the user to exchange updates on that publication with the consolidated database.</p> <p>In MobiLink synchronization, a synchronization subscription is a link in a client database between a publication and a MobiLink user allowing the data described by the publication to be synchronized.</p> <p>☞ See also: <a href="#">“publication” on page 165</a>, <a href="#">“remote user” on page 166</a>, <a href="#">“MobiLink user” on page 163</a>.</p>
Sybase Central	A database management tool that provides Adaptive Server Anywhere database settings, properties, and utilities in a graphical user interface. Sybase Central can also be used for managing other Sybase products, including MobiLink.

synchronization	<p>The process of replicating data between databases using MobiLink technology.</p> <p>In SQL Remote, synchronization is used exclusively to denote the process of initializing a remote database with an initial set of data.</p> <p>☞ See also: <a href="#">“MobiLink” on page 162</a>, <a href="#">“SQL Remote” on page 168</a>.</p>
SYS	<p>A special user that owns most of the system objects. You cannot log in as SYS.</p>
system object	<p>Database objects owned by SYS or dbo.</p>
system table	<p>A table, owned by SYS or dbo, that holds metadata. System tables, also known as data dictionary tables, are created and maintained by the database server.</p>
system view	<p>A type of view, included in every database, that presents the information held in the system tables in an easily understood format.</p>
temporary table	<p>A table that is created for the temporary storage of data. There are two types: global and local.</p> <p>☞ See also: <a href="#">“local temporary table” on page 161</a>, <a href="#">“global temporary table” on page 160</a>.</p>
transaction	<p>A sequence of SQL statements that comprise a logical unit of work. A transaction is processed in its entirety or not at all. Adaptive Server Anywhere supports transaction processing, with locking features built in to allow concurrent transactions to access the database without corrupting the data. Transactions end either with a COMMIT statement, which makes the changes to the data permanent, or a ROLLBACK statement, which undoes all the changes made during the transaction.</p>
transaction log	<p>A file storing all changes made to a database, in the order in which they are made. It improves performance and allows data recovery in the event the database file is damaged. For best results, the transaction log should be kept on a different device from the database files.</p>
transaction log mirror	<p>An optional identical copy of the transaction log file, maintained simultaneously. Every time a database change is written to the transaction log file, it is also written to the transaction log mirror file.</p> <p>A mirror file should be kept on a separate device from the transaction log, so that if either device fails, the other copy of the log keeps the data safe for recovery.</p> <p>☞ See also: <a href="#">“transaction log” on page 169</a>.</p>
transactional integrity	<p>In MobiLink, the guaranteed maintenance of transactions across the</p>

---

	synchronization system. Either a complete transaction is synchronized, or no part of the transaction is synchronized.
trigger	<p>A special form of stored procedure executed automatically when a user executes a query that modifies the data.</p> <p>☞ See also: <a href="#">“row-level trigger” on page 167</a>, <a href="#">“statement-level trigger” on page 168</a>, <a href="#">“conflict trigger” on page 156</a>, <a href="#">“integrity” on page 160</a>.</p>
UltraLite	A deployment technology for Adaptive Server Anywhere databases, aimed at small, mobile, and embedded devices. Intended platforms include cell phones, pagers, and personal organizers.
unique constraint	<p>A restriction on a column or set of columns requiring that all non-null values are different. A table can have multiple unique constraints.</p> <p>☞ See also: <a href="#">“foreign key constraint” on page 159</a>, <a href="#">“primary key constraint” on page 164</a>, <a href="#">“constraint” on page 157</a>.</p>
unload	<p>Unloading a database exports the structure and/or data of the database to text files (SQL command files for the structure, and ASCII comma-separated files for the data). You unload a database with the Unload utility.</p> <p>In addition, you can unload selected portions of your data using the UNLOAD statement.</p>
upload	The stage in synchronization where data is transferred from a remote database to a consolidated database.
user-defined data type	<p>A data type that users create to specify a base data type, and optionally a default value, check condition, and nullability. User-defined data types, also called user-defined domains, can be applied to columns to enforce consistency throughout the database.</p> <p>☞ See also: <a href="#">“data type” on page 157</a>.</p>
validate	To test for particular types of file corruption of a database, table, or index.
view	A SELECT statement that is stored in the database as an object. It allows users to see a subset of rows or columns from one or more tables. Each time a user uses a view of a particular table, or combination of tables, it is recomputed from the information stored in those tables. Views are useful for security purposes, and to tailor the appearance of database information to make data access straightforward.
Windows	The Microsoft Windows family of operating systems, including Windows 95, Windows 98, Windows Me, Windows CE, Windows NT, Windows 2000, and Windows XP.
Windows CE	A family of operating systems produced by Microsoft for mobile devices.



work table	An internal storage area for interim results during query optimization.
write file	A file used to record changes to a read-only database. Often used with compressed databases.
	 See also: <a href="#">“compressed database file” on page 156.</a>

---

# Index

## A

Adaptive Server Anywhere	
applications	29
glossary definition	155
hallmarks	26
intended uses	25
internals	39
introduction	24
programming interfaces	34
quick start	1
system requirements	27
adding	
new rows in Interactive SQL	77
rows	135
ADO	
data control	152
development tools	38
aggregate functions	
applying to grouped data	120
introduction	119
aliases	
for columns	92
alphabetical order	
ORDER BY clause	95
ALTER statement	
automatic commit	138
APIs	
Adaptive Server Anywhere	34
ADO	38
embedded SQL	36
JDBC	38
ODBC	36
OLE DB	38
Open Client	37
architecture of database applications	33
articles	
glossary definition	155
ASA	
glossary definition	155
asademo.db file	
about	xiii, 46
attributes	

tables	11
AUTO_COMMIT option	
grouping changes in Interactive SQL	138
automatic commit	
ALTER statement	138
COMMENT statement	138
data definition statements	138
DROP statement	138
availability	
Adaptive Server Anywhere	
components	30
<b>B</b>	
base tables	13
glossary definition	155
BETWEEN conditions	
WHERE clause	102
binary large objects	
about	56
bitmaps	
storing as blobs	56
BLOBs	
about	56
business rules	
glossary definition	155
<b>C</b>	
canceling Interactive SQL commands	80
cardinality	
relationships and	58
case sensitivity	
SQL	90
table names	90
check constraints	
glossary definition	155
checking	
data integrity	141
checkpoints	
glossary definition	155
clearing	
SQL Statements pane	80

client/server		communication protocols	
glossary definition	155	Adaptive Server Anywhere	35
collations		communication stream	
glossary definition	155	glossary definition	156
columns		comparisons	
about	10	about	98
aliases	92	introduction	99
allowing NULL	57	using subqueries	129
calculated	92	completing transactions	138
data types	56	components	
looking up in Interactive SQL	82	availability	30
ordering	92	compound search conditions	
selecting from a table	92	using	102
combining		compressed database files	
multiple statements in Interactive SQL		glossary definition	156
81		computed columns	
command files		adding to new rows in Interactive SQL	
building	81	78	
glossary definition	156	recalculated in Interactive SQL	77
overview	81	updating in Interactive SQL	77
SQL Statements pane	81	conceptual database models	
command history dialog		definition of	51
recalling commands in Interactive		concurrency	
SQL	84	glossary definition	156
using in Interactive SQL	84	conditions	
command line utilities		GROUP BY clause	122
introduction	30	pattern matching	100
command sequence communication		search	98, 102
protocol		conflict triggers	
about	35	glossary definition	156
diagram	34	connecting your application to its	
commands		database	61
canceling in Interactive SQL	80	connection IDs	
editing in Interactive SQL	84	glossary definition	156
executing in Interactive SQL	80, 90	connection profiles	
getting in Interactive SQL	90	glossary definition	156
interrupting in Interactive SQL	80	connections	
loading in Interactive SQL	90	introduction	62
logging in Interactive SQL	86	consolidated databases	
recalling in Interactive SQL	84	glossary definition	156
saving in Interactive SQL	90	constraints	
stopping in Interactive SQL	80	glossary definition	157
COMMENT statement		contention	
automatic commit	138	glossary definition	157
COMMIT statement		conventions	
about	139	documentation	x
transactions	138	copying	

rows in Interactive SQL	78	glossary definition	158
correlated subqueries		database servers	
defined	131	connecting to	61
correlation names		differences between personal and	
glossary definition	157	network	28
COUNT function		glossary definition	158
applying to grouped data	120	internals	39
create database wizard		quick start	2
using	51	running	61
creating		database sizes	
databases	51	multi-gigabyte databases	25
simple ODBC data sources	63	databases	
cross products		client application	18
introduction	108	components	18
cursor positions		creating	51
glossary definition	157	design concepts	51
cursor result sets		designing	43
glossary definition	157	files	40
cursors		glossary definition	157
glossary definition	157	language interface	18
<b>D</b>		objects	13
data definition statements		queries	15
automatic commit	138	relational	10
data recovery		server	18
transactions	140	SQL	15
data sources		system tables	16
introduction	63	databases and applications	9
data types		dates	
about	10	combining	102
choosing	56	compound	102
glossary definition	157	search conditions	102
database administrator		search conditions introduction	99
glossary definition	158	DBA authority	
database applications		glossary definition	158
architecture	33	dbeng9	
database connections		limitations	28
glossary definition	158	dbisql utility	
database files		about	67
glossary definition	158	DBMS	
introduction	40	glossary definition	158
database name		dbspaces	
glossary definition	158	glossary definition	158
database objects		DDL	
about	13	glossary definition	157
glossary definition	158	DELETE statement	
database owner		about	137
		errors	142

examples	142	feedback	
deleting		documentation	xiv
rows from tables	78	providing	xiv
rows using Interactive SQL	78	FILE	
designing		glossary definition	159
databases	43, 51	FILE message type	
designing and building your database	43	glossary definition	159
developing		file-based downloads	
SQL statements	68	glossary definition	159
dialog boxes		finishing transactions	138
command history	84	foreign key constraints	
DML		glossary definition	159
glossary definition	157	foreign key creation wizard	
documentation		using	60
conventions	x	foreign keys	
SQL Anywhere Studio	viii	about	12
downloads		defined	11
glossary definition	159	glossary definition	159
DROP statement		inserts	141
automatic commit	138	foreign tables	
<b>E</b>		glossary definition	159
editing		full backups	
table values in Interactive SQL	76	glossary definition	160
embedded databases		function keys	
requirements	25	Interactive SQL	72
embedded SQL		functions	
development tools	36	SOUNDEX function	101
glossary definition	159	<b>G</b>	
ending transactions	138	generated join conditions	
entering		glossary definition	160
Interactive SQL commands	80	global temporary tables	
multiple statements in Interactive SQL		glossary definition	160
81		glossary	155
errors		go	
Interactive SQL	80	statement delimiter	82
executing		grant options	
commands in Interactive SQL	80	glossary definition	160
queries more than once	96	GROUP BY clause	
external logins		aggregate functions	120
glossary definition	159	errors	120
extraction		grouped data	119
glossary definition	159	grouping changes into transactions	138
<b>F</b>		<b>H</b>	
failover		hardware requirements	
glossary definition	159		

- 
- |                                     |         |                                      |                            |
|-------------------------------------|---------|--------------------------------------|----------------------------|
| Adaptive Server Anywhere hallmarks  |         | executing all text in SQL Statements |                            |
| 26                                  |         | pane                                 | 72                         |
| HAVING clause                       |         | executing commands                   | 80, 90                     |
| GROUP BY clause and                 | 122     | executing only selected text in SQL  |                            |
| WHERE clause and                    | 122     | Statements pane                      | 82                         |
| <b>I</b>                            |         | function keys                        | 72                         |
| icons                               |         | getting commands                     | 90                         |
| used in manuals                     | xii     | glossary definition                  | 161                        |
| identifiers                         |         | grouping changes into transactions   | 138                        |
| glossary definition                 | 160     | inserting rows                       | 77                         |
| IN conditions                       | 102     | interrupting commands                | 80                         |
| incremental backups                 |         | introduction                         | 30                         |
| glossary definition                 | 160     | keyboard shortcuts                   | 72                         |
| indexes                             |         | loading commands                     | 90                         |
| glossary definition                 | 160     | logging commands                     | 86                         |
| introduction                        | 97      | looking up column names              | 82                         |
| inequality                          |         | looking up procedure names           | 82                         |
| testing for                         | 99      | looking up table names               | 82                         |
| InfoMaker                           |         | main window description              | 70                         |
| glossary definition                 | 160     | opening multiple windows             | 71                         |
| inner joins                         |         | overview                             | 68                         |
| glossary definition                 | 160     | quick start                          | 6                          |
| INSERT statement                    |         | recalling commands                   | 84                         |
| examples                            | 141     | reported errors                      | 80                         |
| introduction                        | 135     | saving commands                      | 90                         |
| inserting                           |         | SQL Statements pane                  | 90                         |
| rows into tables in Interactive SQL | 77      | starting                             | 70                         |
| integrated logins                   |         | stopping commands                    | 80                         |
| glossary definition                 | 160     | toolbar description                  | 71                         |
| integrity                           |         | updating computed columns            | 77                         |
| checking                            | 141     | internals                            |                            |
| glossary definition                 | 160     | Adaptive Server Anywhere             | 39                         |
| Interactive SQL                     |         | database server                      | 39                         |
| about                               | 67      | introduction to Adaptive Server      |                            |
| canceling commands                  | 80      | Anywhere                             | 23                         |
| combining multiple statements       | 81      | isolation levels                     |                            |
| command history dialog              | 84      | glossary definition                  | 161                        |
| commands overview                   | 80      | ISQL                                 | <i>see</i> Interactive SQL |
| copying rows                        | 78      | <b>J</b>                             |                            |
| deleting rows                       | 78      | JAR files                            |                            |
| displaying a list of procedures     | 82      | glossary definition                  | 161                        |
| displaying a list of tables         | 82, 106 | Java classes                         |                            |
| displaying data                     | 75      | glossary definition                  | 161                        |
| displaying the Query Editor         | 72      | jConnect                             |                            |
| editing table values                | 76      | glossary definition                  | 161                        |
| effects of exiting                  | 138     | JDBC                                 |                            |

development tools	38	MAPI message type	
glossary definition	161	glossary definition	162
join conditions		message systems	
glossary definition	161	glossary definition	162
join types		message types	
glossary definition	161	glossary definition	162
joins		metadata	
glossary definition	161	glossary definition	162
introduction	106	system tables	16
or subqueries	131	Microsoft Visual Basic quick start	151
<b>K</b>		mobile computing	
key joins		requirements	25
introduction	111	MobiLink	
keyboard shortcuts		glossary definition	162
Interactive SQL	72	MobiLink clients	
keys		glossary definition	163
about	11	MobiLink users	
foreign	11	glossary definition	163
primary	11	<b>N</b>	
<b>L</b>		natural joins	
LIKE conditions		errors	113
introduction	100	introduction	113
local temporary tables		NetWare	
glossary definition	161	glossary definition	163
locks		network server	
glossary definition	162	about	28
log files		glossary definition	163
glossary definition	162	platform support	28
logging		network software requirements	27
commands in Interactive SQL	86	newsgroups	
looking up		technical support	xiv
columns in Interactive SQL	82	normalization	
procedures in Interactive SQL	82	glossary definition	163
tables in Interactive SQL	82	NULL	
lookup table name dialog		allowing in columns	57, 135
displaying a list of tables	106	<b>O</b>	
LTM		object trees	
glossary definition	166	glossary definition	163
<b>M</b>		ODBC	
many-to-many relationships		development tools	36
defined	59	glossary definition	163
MAPI		introduction to data sources	63
glossary definition	162	ODBC Administrator	
		glossary definition	163
		ODBC data sources	



glossary definition	163	glossary definition	164
OLE DB		personal server	
development tools	38	about	28
OLE DB and ADO programming		glossary definition	164
interfaces	151	limitations	28
ON phrase		platform support	28
introduction	109	platforms	
one-to-many relationships		supported	27
definition of	59	plug-in modules	
one-to-one relationships		glossary definition	164
definition of	58	PowerDesigner	
Open Client		glossary definition	164
development tools	37	PowerDynamo	
opening multiple Interactive SQL		glossary definition	164
windows	71	PowerJ	
operating systems		glossary definition	164
supported	27	predicates	
optimization of queries		glossary definition	164
Adaptive Server Anywhere hallmarks		introduction	102
26		primary key constraints	
ORDER BY clause		glossary definition	164
examples	95	primary keys	11
required to ensure rows always appear		glossary definition	164
in same order	96	primary tables	
using indexes to improve performance		glossary definition	165
97		procedures	
outer joins		looking up in Interactive SQL	82
glossary definition	163	program group	
introduction	115	Adaptive Server Anywhere	31
outer references		programming interfaces	
defined	131	Adaptive Server Anywhere	34
<b>P</b>		ADO	38
packages		embedded SQL	36
glossary definition	164	JDBC	38
parameters		ODBC	36
to functions	119	OLE DB	38
passthrough mode		Open Client	37
glossary definition	164	supported in Adaptive Server	
passwords		Anywhere	34
connecting to a new database	52	projections	
pattern matching		defined	16
introduction	100	proxy tables	
performance monitoring		glossary definition	165
Adaptive Server Anywhere hallmarks		publication updates	
26		glossary definition	165
performance statistics		publications	
		glossary definition	165

publisher			
glossary definition	165		
<b>Q</b>			
queries			
defined	15		
glossary definition	165		
Interactive SQL	80		
SELECT statement	88		
Query Editor			
displaying in Interactive SQL	72		
quick start			
Adaptive Server Anywhere	1		
database server	2		
developing a Visual Basic application	152		
Interactive SQL	6		
Sybase Central	4		
<b>R</b>			
RDBMS			
defined	10		
glossary definition	166		
recalling			
commands in Interactive SQL	84		
recovery			
Adaptive Server Anywhere hallmarks	26		
Redirector			
glossary definition	165		
reference databases			
glossary definition	165		
referential integrity			
glossary definition	165		
relational database-management system			
defined	10		
relational databases			
about	11		
concepts	10		
terminology	11		
relations			
entities	11		
relationships			
about	58		
cardinality of	58		
many-to-many	59		
one-to-many	59		
one-to-one	58		
remote databases			
glossary definition	166		
remote DBA authority			
glossary definition	166		
remote permissions			
glossary definition	166		
remote users			
glossary definition	166		
replication			
glossary definition	166		
replication frequency			
glossary definition	167		
replication messages			
glossary definition	167		
Replication Server			
glossary definition	167		
requirements			
Adaptive Server Anywhere	27		
restrictions			
defined	16		
result sets			
copying rows	78		
deleting rows	78		
editing table values in Interactive SQL	76		
executing a query more than once	96		
inserting rows	77		
troubleshooting	96		
retrieving			
commands in Interactive SQL	84		
role names			
glossary definition	167		
roles			
glossary definition	167		
rollback logs			
glossary definition	167		
ROLLBACK statement			
about	139		
introduction	137		
transactions	138		
rolling back			
transactions	138		
row-level locking			
Adaptive Server Anywhere hallmarks	26		
row-level triggers			

glossary definition	167	glossary definition	168
rows		session-based synchronization	168
about	10	glossary definition	168
adding	135	sorting	
adding using Interactive SQL	77	query results	95
copying in Interactive SQL	78	SOUNDEX function	
deleting using Interactive SQL	78	about	101
editing values in Interactive SQL	76	SQL	
inserting in Interactive SQL	77	about	15
selecting from a table	98	developing queries	68
running		glossary definition	168
Interactive SQL commands	80	SQL and database computing	15
<b>S</b>		SQL Anywhere Studio	
sample database		documentation	viii
about	xiii, 46	SQL Remote	
saving		glossary definition	168
transaction results	138	SQL statements	
schemas		glossary definition	168
defined	51	Start menu	
glossary definition	167	Adaptive Server Anywhere	31
scripts		starting	
glossary definition	167	Interactive SQL	70
search conditions		transactions	138
date comparisons	99	statement-level triggers	
GROUP BY clause	122	glossary definition	168
introduction	98	stored procedures	
pattern matching	100	glossary definition	168
shortcuts for	102	subqueries	
subqueries	127	comparisons	129
select list		correlated subqueries	131
calculated columns	92	glossary definition	168
column names	92	introduction	127
SELECT statement		or joins	131
Interactive SQL	75	troubleshooting	130
introduction	88	subscriptions	
subqueries	127	glossary definition	168
selecting aggregate data	117	summarizing data	118
selecting data from database tables	87	support	
selecting data from multiple tables	105	newsgroups	xiv
selecting data using subqueries	125	supported platforms	
selecting rows from a table	98	Adaptive Server Anywhere	27
selections		Sybase Central	
defined	16	glossary definition	168
server-initiated synchronization		introduction	29
glossary definition	167	quick start	4
services		synchronization	
		glossary definition	169

SYS		Adaptive Server Anywhere hallmarks	
glossary definition	169	26	
SYSCATALOG view		data recovery	140
about	147	transactional integrity	
SYSCOLUMNS view		glossary definition	169
about	148	transactions	
system failures		completing	138
transactions	140	data recovery	140
system objects		glossary definition	169
glossary definition	169	grouping changes	138
system requirements		starting	138
Adaptive Server Anywhere	27	triggers	
system tables		glossary definition	170
defined	16	troubleshooting	
glossary definition	169	GROUP BY clause	120
introduction	146	natural joins	113
SYSTABLE	149	result set appears to change	96
system views		subqueries	130
glossary definition	169	tuples	11
SYSCATALOG	147	tutorials	
SYSCOLUMNS	148	designing a database	51
		developing a Visual Basic application	152
<b>T</b>		<b>U</b>	
table values		UltraLite	
editing in Interactive SQL	76	glossary definition	170
tables		unique constraints	
about	10	glossary definition	170
characteristics	10	unload	
designing	53	glossary definition	170
foreign keys	12	UPDATE statement	
looking up in Interactive SQL	82	errors	143
TDS communication protocol		examples	143
diagram	34	introduction	136
technical support		updating	
newsgroups	xiv	data	133
temporary files		values in Interactive SQL	76
introduction	40	updating the database	133
temporary tables		upload	
glossary definition	169	glossary definition	170
toolbars		user IDs	
Interactive SQL	71	new databases	52
transaction log		user-defined data types	
glossary definition	169	glossary definition	170
introduction	40	using Interactive SQL	67
transaction log mirror		using Interactive SQL to display data	75
glossary definition	169		
transaction processing			

utilities	
introduction	30

## V

validate	
glossary definition	170
values	
editing in Interactive SQL	76
views	
glossary definition	170
SYSCATALOG	147
SYSCOLUMNS	148
Visual Basic	
quick start	152

## W

WHERE clause	
BETWEEN conditions	102
date comparisons introduction	99
deleting rows	137
examples	98
HAVING clause and	122
modifying rows in a table	136
pattern matching	100
wildcards	
pattern matching	100
Windows	
glossary definition	170
supported operating systems	27
Windows CE	
glossary definition	170
wizards	
create database	51
foreign key creation	60
work tables	
glossary definition	171
workgroup computing	
requirements	25
working with SQL statements in	
Interactive SQL	80
write files	
glossary definition	171