

## CHAPTER 3

# Working with Database Objects

### About this chapter

This chapter describes the mechanics of creating, altering, and deleting database objects such as tables, views, and indexes. The SQL statements for carrying out these tasks are called the **Data Definition Language (DDL)**.

The definitions of the database objects form the database schema: you can think of the schema as the database without any data.

☞ Procedures and triggers are also database objects, but are discussed in "Using Procedures, Triggers, and Batches" on page 221.

### Contents

Topic	Page
Tools for working with database objects	66
Working with databases	68
Working with tables	71
Working with views	77
Working with indexes	83

## Tools for working with database objects

Adaptive Server Anywhere includes two utilities for working with database objects: Sybase Central and Interactive SQL. In addition, SQL Modeler can be used for designing and creating whole databases.

### Sybase Central is the primary tool for working with database objects

You can use Sybase Central to create, modify, and delete all kinds of database objects, including tables, procedures, triggers, views, indexes, users and groups.

This chapter is concerned with the SQL statements for working with database objects. If you are using Sybase Central, these SQL statements are generated for you. The primary source of information about Sybase Central is the Sybase Central online Help. In this chapter, only brief pointers are given for tasks that you can carry out using Sybase Central.

☞ For an introduction to using Sybase Central, see "Managing Databases with Sybase Central" on page 45 of the book *First Guide to SQL Anywhere Studio*.

### Using Interactive SQL to work with database objects

Interactive SQL is a utility for entering SQL statements. If you are using Interactive SQL to work with your database schema, instead of executing the SQL statements one at a time you should build up the set of commands in an Interactive SQL command file. This file can then be executed in Interactive SQL to build the database.

If you are using a tool other than Interactive SQL, all the information in this chapter concerning SQL statements still applies.

If you have not created your database using command files, you can create a command file that would recreate your database by unloading the database.

☞ For a description of the Unload utility, see "The Unload utility" on page 110 of the book *Adaptive Server Anywhere Reference Manual*.

#### Interactive SQL command file

An Interactive SQL command file is a text file with semicolons placed at the end of commands (see "Running command files" on page 80 of the book *First Guide to SQL Anywhere Studio*) as shown below.

```
CREATE TABLE t1 ( .. );  
CREATE TABLE t2 ( .. );
```

```
CREATE INDEX i2 ON t2 ( .. );  
..
```

An Interactive SQL command file is usually given a name with the extension *.sql*. To execute a command file, either paste the contents of the file into the Interactive SQL command window (if the file has less than 500 lines) or enter a command that reads the file into the command window. For example:

```
read makesdb
```

reads the Interactive SQL commands in the file *makesdb.sql*.

## Working with databases

Some application design systems, such as Powersoft PowerBuilder, contain facilities for creating database objects. These tools construct SQL statements that are submitted to the server, typically through its ODBC interface. If you are using one of these tools, you do not need to construct SQL statements to create tables, assign permissions, and so on.

This chapter describes the SQL statements for defining database objects. You can use these statements directly if you are building your database from an interactive SQL tool, such as Interactive SQL. Even if you are using an application design tool, you may want to use SQL statements to add features to the database if they are not supported by the design tool.

For more advanced use, database design tools such as Powersoft PowerDesigner provide a more thorough and reliable approach to developing well-designed databases.

### Initializing a database

Initializing a database creates the file for storing your data and the system tables, which hold the schema definition as you build your database.

You create a database using the database initialization utility or the CREATE DATABASE statement. Once the database is initialized, you can connect to it and build the tables and other objects that you need in the database.

#### Database file compatibility

An Adaptive Server Anywhere database is an operating system file. It can be copied to other locations just like any other file is copied.

Database files are compatible among all operating systems. A database created from any operating system can be used from another operating system by copying the database file(s). Similarly, a database created with a personal server can be used with a network server. Adaptive Server Anywhere servers can manage databases created with earlier versions, but old servers cannot manage new databases.

#### Using the CREATE DATABASE statement

You can use the CREATE DATABASE statement to create databases. For example, the following statement creates a database file *c:\temp\temp.db*.

```
CREATE DATABASE 'c:\\temp\\temp.db'
```

The directory path is relative to the database server. The permissions required to execute this statement are set on the server command line, using the `-gu` command-line option. The default setting is to require DBA authority.

**Accessing the initialization utility**

For a full description, see "CREATE DATABASE statement" on page 385 of the book *Adaptive Server Anywhere Reference Manual*.

A full description of the initialization utility, with the options available when you create a database is given in "The Initialization utility" on page 84 of the book *Adaptive Server Anywhere Reference Manual*. The initialization utility can be accessed in the following ways:

- ◆ In Sybase Central, click the Database Utilities folder in the left panel, then double-click Create Database to start the Create Database Wizard, which leads you through the process.
- ◆ Use the *dbinit* command (*dbinitw* for Windows 3.x). For a full description of *dbinit* command, see "The Initialization utility" on page 84 of the book *Adaptive Server Anywhere Reference Manual*.

For example, the following command will create a database called *company.db*:

```
dbinit company.db
```

- ◆ Command line parameters allow different options for the database. For example, the following command creates a database with a 4K page size:

```
dbinit -p 4096 company.db
```

**Creating Windows CE databases from Sybase Central**

Sybase Central has features to make database creation easy for Windows CE databases. If you have Windows CE services installed on your Windows 95 or Windows NT desktop, you get an option to create a Windows CE database when you create a database from Sybase Central (Windows Edition). Sybase Central enforces the requirements for Windows CE databases, and optionally copies the resulting database file to your Windows CE machine.

**Erasing a database**

Erasing a database deletes all tables and data from disk, including the transaction log that records alterations to the database.

All database files are marked as read-only to prevent accidental modification or deletion of the database files.

You can erase database files using the Erase utility. For a full description of the Erase utility, see "The Erase utility" on page 80 of the book *Adaptive Server Anywhere Reference Manual*.

## Accessing the Erase utility

You can access the Erase utility using any of the following methods:

- ◆ **Using Sybase Central** Click the Database Utilities folder, and double-click Erase Database to display the Erase Database Wizard, which leads you through the process.
- ◆ **Using the DBERASE command-line utility** The following command erases the database *company.db* and its transaction log:

```
dberase company.db
```

You will be asked to confirm that you really want to erase the files. To erase the files, type **y** and press ENTER.

- ◆ **Using the DROP DATABASE statement** For information on this statement, see "DROP DATABASE statement" on page 453 of the book *Adaptive Server Anywhere Reference Manual*.
- ◆ **Using the Interactive SQL Database Tools window** To open this window, select the Database Tools menu item from the Window menu. Select Erase Database or Write File from the Tools list, and enter the name of the database file in the Database File field. The database is erased when you press the Erase button.

🔗 For information on using multiple files for a database, and managing other optional files, such as write files, see "Working with Database Files" on page 613.

## Working with tables

When the database is initialized, the only tables in the database are the **system tables**, which hold the database schema.

This section describes how to create, alter, and delete tables from a database. The examples can be executed in Interactive SQL, but the SQL statements are independent of the administration tool you are using.

You should create command files containing the CREATE TABLE and ALTER TABLE statements that define the tables in your database. In this way, you can re-create the database when necessary.

### Creating tables

Creating tables in Sybase Central

Sybase Central provides a tool called the **table editor**. In the table editor, you can create a table definition by filling out a spreadsheet-like form.


❖ **To create a table using Sybase Central:**

- 1 Connect to the database.
- 2 Click the Tables folder for that database.
- 3 Double-click Add Table in the right panel.
- 4 Enter the features you want in the Table Editor.
- 5 Click OK to create the table.

SQL statement for creating tables

The SQL statement for creating tables is the CREATE TABLE statement.

This section describes how to use the CREATE TABLE statement. The examples in this section use the sample database. To try the examples, run Interactive SQL and connect to the sample database with user ID **DBA** and password **SQL**.

 For information on connecting to the sample database from Interactive SQL, see "Connecting to the sample database from Interactive SQL" on page 35.

You can create tables with other tools in addition to Interactive SQL. The SQL statements described here are independent of the tool you are using.

Example

The following statement creates a new table to describe qualifications of employees within a company. The table has columns to hold an identifying number, a name, and a type (say **technical** or **administrative**) for each skill.

```
CREATE TABLE skill (
    skill_id INTEGER NOT NULL,
```

```
skill_name CHAR( 20 ) NOT NULL,  
skill_type CHAR( 20 ) NOT NULL  
)
```

You can execute this command by typing it into the Interactive SQL command window, and pressing the execute key (F9).

- ◆ Each column has a **data type**. The **skill\_id** is an **integer** (like 101), the **skill\_name** is a CHARACTER string containing up to 20 characters, and so on.
- ◆ All columns are mandatory as indicated by the phrase NOT NULL after their data types.
- ◆ In general, you would not create a table that has no primary key. Creating primary keys is dealt with separately, below.

Before creating the table, all previous changes to the database are made permanent by internally executing the COMMIT statement. There is also a COMMIT after the table is created.

🔗 For a full description of the CREATE TABLE statement, see "CREATE TABLE statement" on page 415 of the book *Adaptive Server Anywhere Reference Manual*. For information about building constraints into table definitions using CREATE TABLE, see "Ensuring Data Integrity" on page 347.

## Altering tables

This section describes how to change the structure of a table using the ALTER TABLE statement.

### Example 1

The following command adds a column to the **skill** table to allow space for an optional description of the skill:

```
ALTER TABLE skill  
ADD skill_description CHAR( 254 )
```

This statement adds a column called **skill\_description** that holds up to a few sentences describing the skill.

### Example 2

Column attributes can also be modified with the ALTER TABLE statement. The following statement shortens the **skill\_description** column of the sample database from a maximum of 254 characters to a maximum of 80:

```
ALTER TABLE skill  
MODIFY skill_description CHAR( 80 )
```

Any current entries that are longer than 80 characters are trimmed to conform to the 80-character limit, and a warning is displayed.



- Example 3** The following statement changes the name of the **skill\_type** column to **classification**:
- ```
ALTER TABLE skill
RENAME skill_type TO classification
```
- Example 4** The following statement deletes the **classification** column.
- ```
ALTER TABLE skill
DROP classification
```
- Example 5** As a final example, the following statement changes the name of the entire table:
- ```
ALTER TABLE skill
RENAME qualification
```

These examples show how to change the structure of the database. The ALTER TABLE statement can change just about anything pertaining to a table—foreign keys can be added or deleted, columns can be changed from one type to another, and so on.

For a complete description of the ALTER TABLE command, see "ALTER TABLE statement" on page 351 of the book *Adaptive Server Anywhere Reference Manual*. For information about building constraints into table definitions using ALTER TABLE, see "Ensuring Data Integrity" on page 347.

#### Altering tables in Sybase Central

The property sheets for tables and columns display all the table or column attributes. You can alter a table definition in Sybase Central by displaying the property sheet for the table or column you wish to change, altering the property, and clicking OK to commit the change.

## Deleting tables

The following DROP TABLE command deletes all the records in the **skill** table and then removes the definition of the **skill** table from the database

```
DROP TABLE skill
```

Like the CREATE statement, the DROP statement automatically executes a COMMIT statement before and after dropping the table. This makes all changes to the database since the last COMMIT or ROLLBACK permanent.

For a full description of the DROP statement, see "DROP statement" on page 451 of the book *Adaptive Server Anywhere Reference Manual*.

#### ❖ To drop a table in Sybase Central:

- 1 Connect to the database.

- 2 Click the Tables folder for that database.
- 3 Right-click the table you wish to delete, and select Delete from the pop-up menu.

## Creating primary and foreign keys

The CREATE TABLE and ALTER TABLE statements allow many attributes of tables to be set, including column constraints and checks. This section shows how to set table attributes using the primary and foreign keys as an example.

### Creating a primary key

The following statement creates the same **skill** table as before, except that a **primary key** is added:

```
CREATE TABLE skill (  
    skill_id INTEGER NOT NULL,  
    skill_name CHAR( 20 ) NOT NULL,  
    skill_type CHAR( 20 ) NOT NULL,  
    primary key( skill_id )  
)
```


The primary key values must be unique for each row in the table which, in this case, means that you cannot have more than one row with a given **skill\_id**. Each row in a table is uniquely identified by its primary key.

Columns in the primary key are not allowed to contain NULL. You must specify NOT NULL on the column in the primary key.

### Creating a primary key in Sybase Central

#### ❖ To create a primary key in Sybase Central:

- 1 Connect to the database.
- 2 Click the Tables folder for that database.
- 3 Right-click the table you wish to modify, and select Properties from the pop-up menu to display its property sheet.
- 4 Click the Columns tab, select the column name, and either click Add to Key or Remove from Key.

 For more information, see the Sybase Central online Help.

#### **Column order in multi-column primary keys**

Primary key column order is based on the order of the columns during table creation. It is not based on the order of the columns as specified in the primary key declaration.

### Creating foreign keys

You can create a table named **emp\_skill**, which holds a description of each employee's skill level for each skill in which they are qualified, as follows:

```
CREATE TABLE emp_skill(
    emp_id INTEGER NOT NULL,
    skill_id INTEGER NOT NULL,
    "skill level" INTEGER NOT NULL,
    PRIMARY KEY( emp_id, skill_id ),
    FOREIGN KEY REFERENCES employee,
    FOREIGN KEY REFERENCES skill
)
```

The **emp\_skill** table definition has a primary key that consists of two columns: the **emp\_id** column and the **skill\_id** column. An employee may have more than one skill, and so appear in several rows, and several employees may possess a given skill, so that the **skill\_id** may appear several times. However, there may be no more than one entry for a given employee and skill combination.

The **emp\_skill** table also has two foreign keys. The **foreign key** entries indicate that the **emp\_id** column must contain a valid employee number from the **employee** table, and that the **skill\_id** must contain a valid entry from the **skill** table.

A table can only have one primary key defined, but it may have as many foreign keys as necessary.

The **skill level** column name contains a space and is surrounded by quotation marks ("double quotes"). You can use any characters in column names and table names, but the names must be enclosed in quotation marks under the following circumstances:

- ◆ If any characters other than letters, digits or the underscore are used
- ◆ If the name does not begin with a letter
- ◆ If the name is the same as a keyword.

### Single and double quotes in SQL


Remember, in SQL:

- ◆ Single quotes (apostrophes) are used to indicate strings. For example:
 

```
'SMITH', '100 Apple St.', '1988-1-1'.
```
- ◆ Double quotes (quotation marks) are used to indicate table or column names (for example, "skill level", "emp\_id", "skill\_type").
- ◆ To include a single quote inside a string, use two single quotes:
 

```
'''Plankton''', said the cat'
```


### Creating a foreign key in Sybase Central

 For more information about using primary and foreign keys, see "Ensuring Data Integrity" on page 347.

Each foreign key relationship lists a primary key in one column to a column in another table, which becomes the foreign key.


#### ❖ To create a foreign key in Sybase Central:

- 1 Connect to the database.
- 2 Click the Tables folder for that database.
- 3 Click the table holding the primary key, and drag it to the foreign key table.
- 4 When the primary key table is dropped on the foreign key table, the Foreign Key Wizard is displayed, which leads you through the process of creating the foreign key.

 For more information, see the Sybase Central online Help.

## Table information in the system tables

All the information about tables in a database is held in the system tables. The information is distributed among several tables.

 For more information, see "System Tables" on page 771 of the book *Adaptive Server Anywhere Reference Manual*.

You can use Sybase Central or Interactive SQL to browse the information in these tables. Type the following command in the Interactive SQL command window to see all the columns in the SYS.SYSTABLE table:

```
SELECT *  
FROM SYS.SYSTABLE
```

#### ❖ To display the system tables in Sybase Central:

- 1 Connect to the database.
- 2 Right-click the database, and select Filter Objects from the pop-up menu.
- 3 Select SYS and OK.
- 4 When you view the database tables or views with Show System Objects checked, the system tables or views are also shown.

## Working with views

|                                                |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
|------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|                                                | Views are computed tables. You can use views to show database users exactly the information you want to present, in a format you can control.                                                                                                                                                                                                                                                                                                                                                                       |
| Similarities between views and base tables     | <p>Views are similar to the permanent tables of the database (a permanent table is also called a <b>base table</b>) in many ways:</p> <ul style="list-style-type: none"> <li>◆ You can assign access permissions to views just as to base tables.</li> <li>◆ You can perform SELECT queries on views.</li> <li>◆ You can perform UPDATE, INSERT, and DELETE operations on some views.</li> <li>◆ You can create views based on other views.</li> </ul>                                                              |
| Differences between views and permanent tables | <p>There are some differences between views and permanent tables:</p> <ul style="list-style-type: none"> <li>◆ You cannot create indexes on views.</li> <li>◆ You cannot perform UPDATE, INSERT, and DELETE operations on all views.</li> <li>◆ You cannot assign integrity constraints and keys to views.</li> <li>◆ Views are recomputed each time they are invoked. Views refer to the information in base tables, but do not hold copies of that information.</li> </ul>                                        |
| Benefits of tailoring access                   | <p>Views are used to tailor access to data in the database. Tailoring access serves several purposes:</p> <ul style="list-style-type: none"> <li>◆ <b>Improved security</b> By not allowing access to information that is not relevant.</li> <li>◆ <b>Improved usability</b> By presenting users and application developers with data in a more easily understood form than in the base tables.</li> <li>◆ <b>Improved consistency</b> By centralizing in the database the definition of common queries.</li> </ul> |

## Creating views

|         |                                                                                                                                                                                                                                                                                     |
|---------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|         | A SELECT statement operates on one or more tables and produces a result set that is also a table: just like a base table, a result set from a SELECT query has columns and rows. A view gives a name to a particular query, and holds the definition in the database system tables. |
| Example | Suppose that you frequently need to list the number of employees in each department. You can get this list with the following statement:                                                                                                                                            |

```
SELECT dept_ID, count(*)
FROM employee
GROUP BY dept_ID
```


You can create a view containing the results of this statement as follows:

```
CREATE VIEW DepartmentSize AS
SELECT dept_ID, count(*)
FROM employee
GROUP BY dept_ID
```

The information in a view is not stored separately in the database. Each time you refer to the view, the associated SELECT statement is executed to retrieve the appropriate data. On one hand, this is good because it means that if someone modifies the **employee** table, the information in the **DepartmentSize** view will be automatically up to date. On the other hand, if the SELECT statement is complicated it may take a long time for SQL to find the correct information every time you use the view.

❖ **To create a view in Sybase Central:**

- 1 Connect to the database.
- 2 Click the Views folder for that database.
- 3 Double-click Add View.
- 4 Enter the table and the columns to be used. In this case employee and dept\_ID. From the File menu select Execute Script and from the File menu select Close.

 For more information, see the Sybase Central online Help.

## Using views

### Restrictions on SELECT statements

There are some restrictions on the SELECT statements that you can use as views. In particular, you cannot use an ORDER BY clause in the SELECT query. It is a characteristic of relational tables that there is no significance to the ordering of the rows or columns, and using an ORDER BY clause would impose an order on the rows of the view. You can use the GROUP BY clause, subqueries, and joins in view definitions.

To develop a view, you should tune the SELECT query by itself until it provides exactly the results you need in the format you want. Once you have the SELECT query just right, you can add a

```
CREATE VIEW viewname AS
```

phrase in front of the query to create the view.

**Updating views**

UPDATE, INSERT, and DELETE statements are allowed on some views, but not on others, depending on its associated SELECT statement.

Views containing aggregate functions, such as COUNT(\*), cannot be updated. Views containing a GROUP BY clause in the SELECT statement cannot be updated. Also, views containing a UNION operation cannot be updated. In all these cases, there is no way to translate the UPDATE into an action on the underlying tables.

**Using the WITH CHECK OPTION clause**

Even when INSERT and UPDATE statements are allowed against a view, it is possible that the inserted or updated rows in the underlying tables may not meet the requirements for the view itself: the view would have no new rows even though the INSERT or UPDATE does modify the underlying tables.

**Examples using the WITH CHECK OPTION clause**

The following set of examples illustrates the meaning and usefulness of the WITH CHECK OPTION clause. This optional clause is the final clause in the CREATE VIEW statement.

❖ **To create a view displaying the employees in the sales department.**

- ◆ Type the following statements:

```
CREATE VIEW sales_employee
AS SELECT emp_id,
        emp_fname,
        emp_lname,
        dept_id
FROM employee
WHERE dept_id = 200
```

The contents of this view are as follows:

```
SELECT *
FROM sales_employee
```

| emp_id | emp_fname | emp_lname | dept_id |
|--------|-----------|-----------|---------|
| 129    | Philip    | Chin      | 200     |
| 195    | Marc      | Dill      | 200     |
| 299    | Rollin    | Overbey   | 200     |
| 467    | James     | Klobucher | 200     |
| 641    | Thomas    | Powell    | 200     |
| ...    |           |           |         |

- ◆ **Transfer Philip Chin to the marketing department.** This view update causes the entry to vanish from the view, as it no longer meets the view selection criterion.

```
UPDATE sales_employee
SET dept_id = 400
WHERE emp_id = 129
```

- ◆ **List all employees in the sales department** Inspect the view.

```
SELECT *
FROM sales_employee
```

| emp_id | emp_fname | emp_lname | dept_id |
|--------|-----------|-----------|---------|
| 195    | Marc      | Dill      | 200     |
| 299    | Rollin    | Overbey   | 200     |
| 467    | James     | Klobucher | 200     |
| 641    | Thomas    | Powell    | 200     |
| 667    | Mary      | Garcia    | 200     |
| ...    |           |           |         |

When a view is created WITH CHECK OPTION, any UPDATE or INSERT statement on the view is checked to ensure that the new row matches the view condition. If it does not, the operation causes an error and is rejected.

The following modified sales\_employee view rejects the update statement, generating the following error message:

Invalid value for column 'dept\_id' in table 'employee'

- ◆ **Create a view displaying the employees in the sales department (second attempt)** Use WITH CHECK OPTION this time.

```
CREATE VIEW sales_employee
AS SELECT emp_id, emp_fname, emp_lname, dept_id
FROM employee
WHERE dept_id = 200
WITH CHECK OPTION
```

The check option is inherited

If a view (say V2) is defined on the sales\_employee view, any updates or inserts on V2 that cause the WITH CHECK OPTION criterion on sales\_employee to fail are rejected, even if V2 is defined without a check option.



## Modifying views

You can modify a view using the ALTER VIEW statement. The ALTER VIEW statement replaces a view definition with a new definition; it does not modify an existing view definition.

The ALTER VIEW statement maintains the permissions on the view.

### Example

For example, to replace the column names with more informative names in the **DepartmentSize** view described above, you could use the following statement:

```
ALTER VIEW DepartmentSize
  (Dept_ID, NumEmployees)
AS
  SELECT dept_ID, count(*)
  FROM Employee
  GROUP BY dept_ID
```

## Permissions on views

An INSERT, DELETE, or UPDATE operation is allowed either if permission on the view has been granted or if permission on the underlying tables has been granted.

UPDATE permissions can be granted only on an entire view. Unlike tables, UPDATE permissions cannot be granted on individual columns within a view.

### Behavior change

There was a behavior change with Version 5 of the software concerning the permission requirements. Previously, permissions on the underlying tables were required in order to grant permissions on views.


## Deleting views

To delete a view from the database, you use the DROP statement. The following statement removes the DepartmentSize view:

```
DROP VIEW DepartmentSize
```

### Dropping a view in Sybase Central

To drop a view in Sybase Central, right-click the view you wish to delete and select Delete from the pop-up menu.

 For more information, see the Sybase Central online Help.

## Views in the system tables

All the information about views in a database is held in the system table SYS.SYSTABLE. The information is presented in a more readable format in the system view SYS.SYSVIEWS. For more information about these, see "SYSTABLE system table" on page 814 of the book *Adaptive Server Anywhere Reference Manual*, and "SYSVIEWS system view" on page 850 of the book *Adaptive Server Anywhere Reference Manual*.

You can use Interactive SQL to browse the information in these tables. Type the following statement in the Interactive SQL command window to see all the columns in the SYS.SYSVIEWS view:

```
SELECT *  
FROM SYS.SYSVIEWS
```

To extract a text file containing the definition of a specific view, use a statement such as the following:

```
SELECT viewtext FROM SYS.SYSVIEWS  
WHERE viewname = 'DepartmentSize';  
OUTPUT TO viewtext.sql  
FORMAT ASCII
```

## Working with indexes

Performance is an important consideration when designing and creating your database. Indexes can dramatically improve the performance of database searches (operations using SELECT, UPDATE, and DELETE statements) on specified columns.

### When to use indexes

An index is similar to a telephone book which first sorts people by their last name, and then sorts all the people with the same last name by their first name. Telephone books are indexed on the last name and first name. This speeds up searches for phone numbers given a particular last name. Just as a standard telephone book is no use at all for finding the phone number at a particular address, so an index is useful only for searches on a specific column or columns.

Indexes get more useful as the size of the table increases. The average time to find a phone number at a given address increases with the size of the phone book, while it does not take much longer to find the phone number of, say, K. Kaminski, in a large phone book than in a small phone book.

### Use indexes for frequently-searched columns

Indexes share one other feature with a phone book: they can take up a great deal of space for large data sets. For this reason, you should build indexes only for columns that are searched frequently or when disk space is not an issue.

If a column is already a **primary key** or **foreign key**, searches will be fast on this column because Adaptive Server Anywhere has facilities to optimize searches on these key columns. Thus, creating an index on a key column is not necessary and generally not recommended. If a column is only part of a key, an index may help.

#### When indexes on primary keys may be useful

One case where an index on a key column may assist performance is when a large number of foreign keys reference a primary key.

Adaptive Server Anywhere automatically uses indexes to improve the performance of any database statement whenever it can. There is no need to refer to indexes once they are created. Also, the index is updated automatically when rows are deleted, updated or inserted.

Indexes are created on a specified table. You cannot create an index on a view.

If an index is no longer required, you can remove it from the database using the DROP statement.

**Example**

In order to speed up a search on employee surnames in the sample database, you could create an index called **EmpNames** with the following statement:

```
CREATE INDEX EmpNames
ON employee (emp_lname, emp_fname)
```

The following statement removes the index from the database:

```
DROP INDEX EmpNames
```

**For more information**


- ◆ For more information about improving database performance, including the use of indexes, see "Monitoring and Improving Performance" on page 623.
- ◆ For a detailed description of the CREATE INDEX statement, including syntax and permission requirements, see "CREATE INDEX statement" on page 399 of the book *Adaptive Server Anywhere Reference Manual*.
- ◆ For a detailed description of the DROP statement, including syntax and permission requirements, see "DROP statement" on page 451 of the book *Adaptive Server Anywhere Reference Manual*.

**Creating and dropping indexes in Sybase Central**

❖ **To create an index on a table in Sybase Central:**

- 1 Connect to the database.
- 2 Double-click the table you wish to modify.
- 3 Double-click the Indexes folder, and then double-click Add Index.
- 4 Fill in the dialog box and click OK to complete.

You can drop an index in Sybase Central by right-clicking it, and selecting Delete from the pop-up menu.

 For more information, see the Sybase Central online Help.

## Indexes in the system tables

All the information about indexes in a database is held in the system tables SYS.SYSINDEX and SYS.SYSIXCOL. The information is presented in a more readable format in the system view SYS.SYSINDEXES. You can use Sybase Central or Interactive SQL to browse the information in these tables.