

2. Click the Design View button of the toolbar, and then relocate and resize the controls as necessary. Your bound object frame should occupy most of the display area. To view the entire document in its original format, set the Height property of the object frame to 11 inches and the Width property to 8.5 inches.



3. Return to Form view to display the presentation of the document. Figure 22.28 shows the presentation of the initial version of the manuscript for this chapter of the book. The size of the bound object frame of Figure 22.28 is about 8.5×11 inches.

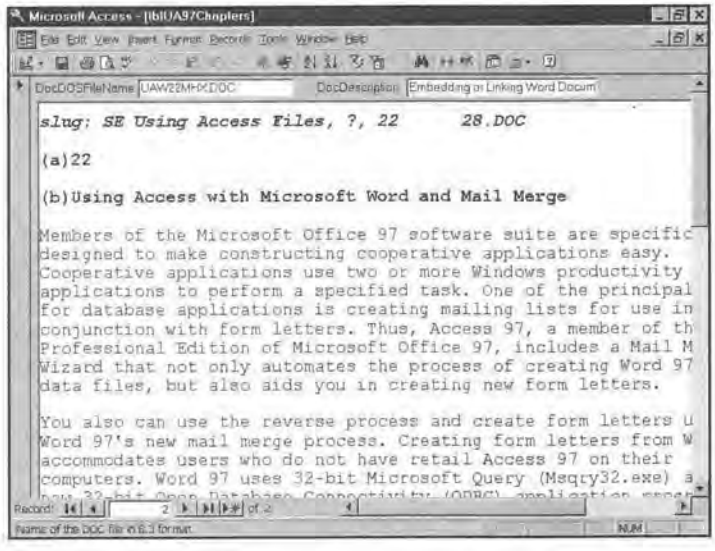


FIG. 22.28 The presentation of a Word 97 document in a bound object frame.

4. Double-click the surface of the object frame to activate the object. Activating the object launches Word 97 if it is not running. If you embedded the document, activation adds Word's toolbars to the display as docked toolbars. Word's menu choices take over Access's Edit and View menus, and Word adds its Insert, Format, Tools, and Table menus to the menubar (see Figure 22.29).

   You can move through the document with the Page Up and Page Down keys. All editing features of Word 97 are available when the document object is activated, but you can only view the document in Page Layout view. You may use the scroll bar of the Access form to view parts of the page that are not visible on your display. (Using 800×600-pixel or higher resolution solves the partial display problem.)

5. Click the surface of the form, outside of the bound object frame area, to deactivate the object and return to Presentation view of the document.

Oracle Ex. 1009, p. 796
Oracle v. DeBellis
IPR2024-00221

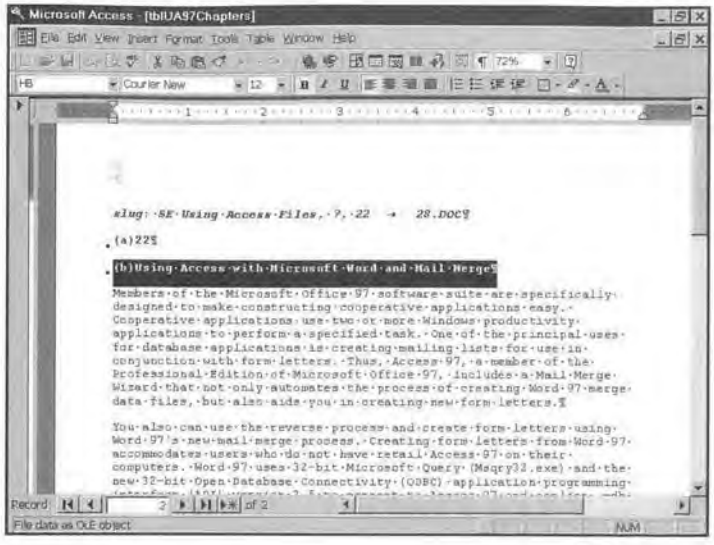


FIG. 22.29 An embedded Word 97 document activated in a bound object frame.

Tip

You can also click the record selection bar at the left edge of the form to deactivate the object and return to Presentation view.

6. To save the document to a file, to alter the page layout, or to print an embedded document, choose Edit, Document Object, and select Open. Microsoft Word opens a separate window in which to edit the embedded document, and you can access the File menu of Word to save changes, as shown in Figure 22.30.

7. Choose File, Close and Return to *FormName* to close Word's window and return to Access.

Note

The only available view of a Word document when the object is embedded is Page Layout view. You can change the layout of the embedded document by opening the document in Word (see the preceding step 8), choosing File, Page Setup, and then making the required adjustments.

Oracle Ex. 1009, p. 797
Oracle v. DeBellis
IPR2024-00221

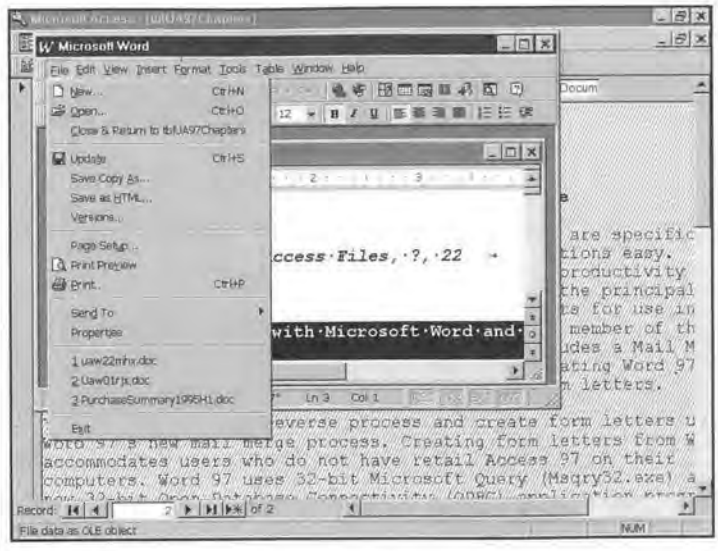
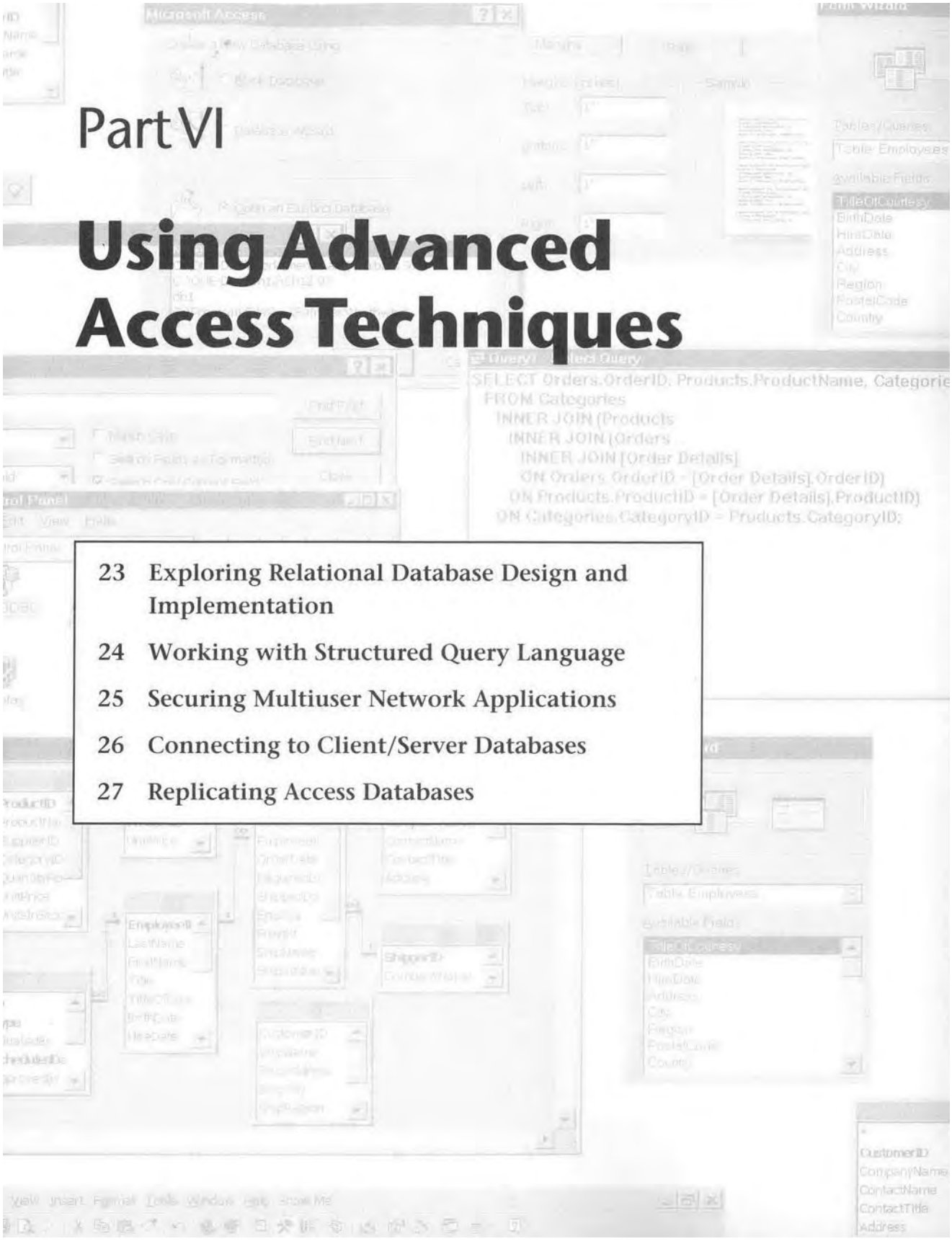


FIG. 22.30 Opening the embedded document in Word 97's window.

You also can insert additional document objects directly into the form. To embed or link an object in Form view, position the record pointer on the blank (tentative append) record. An empty presentation appears in the bound object frame. Choose Insert, Object, and follow steps 2 through 5 of the preceding section to embed or link additional document objects.

Oracle Ex. 1009, p. 798
Oracle v. DeBellis
IPR2024-00221



Part VI

Using Advanced Access Techniques

23 Exploring Relational Database Design and Implementation

24 Working with Structured Query Language

25 Securing Multiuser Network Applications

26 Connecting to Client/Server Databases

27 Replicating Access Databases

Oracle Ex. 1009, p. 799
Oracle v. DeBellis
IPR2024-00221

Chapter 23

Exploring Relational Database Design and Implementation

You were introduced to a few of the elements of relational database design when you created the Personnel Actions table and joined it with the Employees table of the Northwind Traders database in Chapter 10, "Creating Multitable and Crosstab Queries." Chapter 21, "Using Access with Microsoft Excel," gave you a bit more insight into how to create a relational database from information contained in a worksheet. However, when you're presented with the challenge of designing a database from ground zero—especially a complex or potentially complex database—you need to understand the theory of relational database design and its terminology.

Integrating Objects and Relational Databases

This chapter takes a step back and starts with the definition of data objects and how you identify them. Because Access is an object-enabled database development tool, the concepts of database design presented in this chapter have an object-oriented bent. The reason for this approach is twofold:

- Access's relational tables incorporate many of the features of client/server databases. Properties, such as validation rules and indexes, and methods, that include preventing duplicate primary key entries, are combined in the table object. Details of relationships between tables and methods of enforcing referential integrity are stored in the database object.
- Access VBA treats the database itself and each of Access's database elements—tables, queries, forms, and reports—as programming objects.

After you've identified the data objects that are to be included in the tables of your database, you need to design the tables to contain the data objects. You use a process called data *normalization* to create tables that conform to the relational database model. Normalization is the process of eliminating duplicate information in tables by extracting the duplicate data to new tables that contain records with unique data values. You then join the new tables by

Oracle Ex. 1009, p. 800
Oracle v. DeBellis
IPR2024-00221

fields with common data values to create a relational database structure. Normalizing data is the subject of the "Normalizing Data to the Relational Model" section later in this chapter.

The role of indexes in maintaining unique values in primary-key fields and organizing data tables was described briefly in preceding chapters. This chapter provides an explanation of how indexes are constructed and maintained by Access 97 and the Jet 3.5 database engine. Properly designed indexes improve the performance of your applications without consuming excessive amounts of disk space or slowing the appending of new records to a crawl.

This chapter also deals with the rules that establish and maintain referential integrity—one of the most important considerations in designing a database. Referential integrity enforces uniqueness in primary keys and prevents the occurrence of orphaned records, such as records of invoices whose customer data records have been deleted.

Understanding Database Systems

Prior to this chapter, you've used the Northwind Traders demonstration database, created a few simple databases, and perhaps imported your own data in another database format into an Access 97 table. No formal theories were presented to aid or hinder your understanding of the underlying design of the database examples. Now that you've gained some experience using Access, the more theoretical concepts of database design should be easier to understand.

This section takes a systems approach to database design, starting with a generalized set of objectives, outlining the steps necessary to accomplish the objectives, and then explaining the theory and practice behind each step.

The Objectives of Database Design

The strategy of database design is to accomplish the following objectives:

- Fulfill your needs or the needs of the organization for information in a timely, consistent, and economical manner.
- Eliminate or minimize the duplication of database content across the organization. In a large organization, eliminating duplication may require a distributed database. Distributed databases use multiple servers to store individual databases. The individual databases are linked to one another (to use Access terminology) through a local-area network (LAN) or wide-area network (WAN) so that they appear as a single database to the user.
- Provide rapid access to the specific elements of information in the database required by each user category. Operating speed is a function of the relational database management system (RDBMS) itself, the design of the applications you create, the capabilities of the server and client computers, and network characteristics.

Oracle Ex. 1009, p. 801
Oracle v. DeBellis
IPR2024-00221

- Accommodate database expansion to adapt to the needs of a growing organization, such as the addition of new products and processes, complying with governmental reporting requirements, and incorporating new transaction-processing and decision-support applications.
- Maintain the integrity of the database so that it contains only validated, auditable information. Some client/server databases, such as Microsoft SQL Server, provide built-in triggers to maintain database integrity. Triggers are sets of rules that are included in the database. If you violate a rule, the trigger sends an error message instead of performing the transaction. The Enforce Referential Integrity check box in Access's Relationships dialog creates the equivalent of a trigger.
- Prevent access to the database by unauthorized persons. Access provides a security system that requires users to enter a password in order to use a particular database.
- Permit access only to those elements of the database information that individual users or categories of users need in the course of their work. You can permit or deny users the right to view the data in specific tables of the database.
- Allow only authorized persons to add or edit information in the database. Permissions in Access are multilevel; you can selectively allow users to edit tables or alter their structure, as well as edit or create their own applications.
- Ease the creation of data entry, editing, display, and reporting applications that efficiently serve the needs of the users of the database. The design of the RDBMS's front-end features determines the ease with which new applications are created or existing ones can be modified. You have seen in the preceding chapters that Access is especially adept as a front-end application generator.

The first two objectives are independent of the database manager you choose. The RDBMS influences or determines the other objectives. Operating speed, data validation, data security, and application creation are limited by the capabilities built into the RDBMS and the computer environment under which it operates. If your database is shared on a network, you need to consider the security features of the network operating system and the client/server database system (if one is used) in the security strategy.

Note

Database replication presents an exception to the objective of minimizing data duplication. Replication of entire databases or specific tables of databases often is used to improve performance of queries against remote databases that are accessed over a WAN. A local copy of the remote database is maintained on a LAN server. Updates to the tables of the remote database are periodically propagated over the WAN to the local replicate databases. If users don't need current information, replication is scheduled in the middle of the night when WAN traffic is minimal.

Access 97 incorporates a small-scale replication system that is designed primarily to accommodate mobile computer users who maintain a copy of a database on their laptop PCs. Chapter 27, "Replicating Access Databases," describes Access 97's replication capabilities that take advantage of Windows 95's Briefcase feature.

Oracle Ex. 1009, p. 802
Oracle v. DeBellis
IPR2024-00221

The Process of Database Design

The process of designing a relational database system consists of 10 basic steps:

1. Identify the objects (data sources) that the database system is to represent.
2. Discover associations between the objects (when you have more than one object).
3. Determine the significant properties and behaviors of the objects.
4. Ascertain how the properties of the objects relate to one another.
5. Create a preliminary data dictionary to define the tables that comprise the database.
6. Designate the relationships between database tables based on the associations between data objects contained in the tables, and incorporating this information in the data dictionary.
7. Establish the types of updates and transactions that create and modify the data in the tables, including any necessary data-integrity requirements.
8. Determine how to use indexes to speed up query operations without excessively slowing down the addition of data to tables or consuming excessive amounts of disk space.
9. Decide who can access and modify data in each table (data security), and alter the structure of the tables if necessary to assure data security.
10. Document the design of the database as a whole; complete data dictionaries for the database as a whole and each table it contains; and write procedures for database maintenance, including file backup and restoration.

Each step in the design process depends on preceding steps. The sections in this chapter follow steps 1–8 in sequence. Database security is the subject of Chapter 25, "Securing Multiuser Network Applications." A full discussion of database documentation is beyond the scope of this book, but this chapter explains how to use Access 97's improved Documentor feature to create a data dictionary.

The Object-Oriented Approach to Database Design

Databases contain information about objects that exist in the real world. These objects may be people, books in a library, paper invoices or sales orders, maps, money in bank accounts, or printed circuit boards. Such objects are *tangible*. Whatever the object, it must have a physical representation, even if only an image on a computer display that never finds its way to the printer, as in the mythical "paperless office." References to objects in this book, if not preceded by a word describing the type of object—such as "table object" or "OLE object"—indicate real-world, tangible objects.

Tangible objects possess *properties* and *behavior*, just as the OLE objects discussed in Chapter 19, "Using 32-Bit OLE Components," have properties and methods. At first, this combination might appear to be applicable only to databases of persons, not books or bank balances. However, all database objects other than those in archival databases have

Oracle Ex. 1009, p. 803
Oracle v. DeBellis
IPR2024-00221

both properties and behavior. (*Archival databases* store information that never changes—new data is simply added to such databases and the indexes to the data, if any, are updated. An example of an archival database is one containing the text of previously published newspapers.)

◀◀ See "Properties and Methods Encapsulated in Objects," p. 670



Considering Static and Dynamic Properties of Objects. An object's properties determine the content of a database or table that contains object representations of the same type. Books are assigned subject codes, which are derived from the Dewey decimal system. Modern books have an identifying ISBN code, and most now have a Library of Congress catalog number. These numbers are properties of a book, as are the title, author, number of pages, and binding type. Such properties are *static*: they are the same whether the book is in the stacks of a library or checked out by a cardholder. Customer information for a bank account, such as the account number, name, and address, also is considered static, even though customers occasionally change addresses. Book circulation status and bank account balances are *dynamic* properties: they change from day to day or hour to hour.

Describing Data Entities and Their Attributes. A single object, including all of its static properties, is called a *data entity*. Each individual data entity must be unique so that you can distinguish it from others. A bank's checking account customer is a data entity, for example, but money in the customer's account is not, because the money is fungible and cannot (and doesn't need to be) uniquely identified. A customer may have more than one account, so a Social Security number or federal employer identification number doesn't suffice as a unique identifier. An account number must be assigned to ensure the uniqueness of each customer data entity.

Deposit slips and checks are objects that are represented in the database as other data entities that *relate* to the customer entity. Check numbers aren't unique enough to distinguish them as entities; many different customers might use a check numbered 1553. Combining the customer and check number doesn't suffice as a unique identifier because different banking firms might use the same customer number to identify different people. A bank identification number, customer number, and check number together can uniquely identify a debit entity. Each check contains this information printed in magnetic ink. The amount property of each debit (check) or credit (deposit) entity is used to adjust the balance in the customer's account by simple subtraction and addition, a process called a *transaction*.

You don't want to wait while an ATM (originally Automated Transaction Machine, now commonly called an Automatic Teller Machine) calculates your balance by processing every transaction since you opened your account. Therefore, a derived static property, the last statement balance, can be included in the customer data entity and updated once per month. Only last-statement-to-date transactions need to be processed to determine the current balance—a dynamic, *calculated* property. In Figure 23.1, lines connect static properties of bank account objects to the data entities derived from them.

Oracle Ex. 1009, p. 804
Oracle v. DeBellis
IPR2024-00221

Properties of objects included in data entities, such as account number and customer name, are called *attributes*.

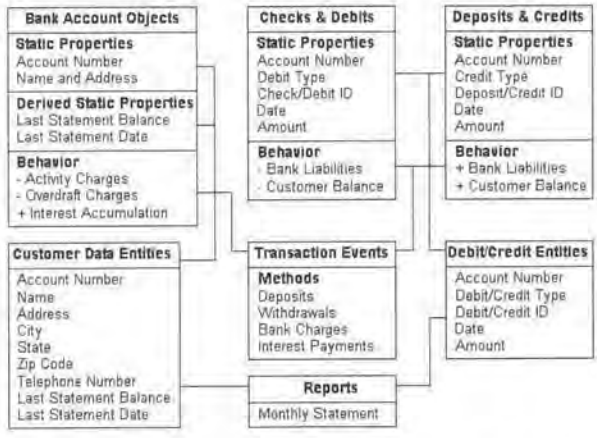


FIG. 23.1 Relationships between objects, entities, events, and methods in a banking database.

Accounting for the Behavior of Objects with Methods. The behaviors of related database objects determine the characteristics of transactions in which their data entities participate. Books in a library may be acquired, checked out, returned, and lost. Bank account behavior is very easy to describe: a customer opens the account; deposit transactions and interest accumulations (credits) increase the account balance; and checks, cash withdrawn from an ATM, and bank charges incurred (debits) reduce the balance. Crediting or debiting a bank account is an example of a transaction. Transactions occur in response to events, such as making a deposit or withdrawal at an ATM. Access implements transactions by using methods in response to events initiated by the user, such as opening a form or clicking a command button.

In conventional relational databases, you can represent tangible objects and object properties as data entities but not an object's real-world behavior. The OLE Object field data type, described in Chapter 20, "Adding Graphics to Forms and Reports," is an exception to this rule. The behavior of an OLE data object is determined by the methods available in the OLE server used to create the object or add the object to the OLE Object field. With conventional data entities, you emulate the behavior of tangible objects by applying the methods that you incorporate into your applications.



◀◀ See "Adding a Bound Object Control to a Form or Report," p. 691

Programs that you write using the RDBMS's native programming language(s) implement database methods. In the case of Access 97, VBA functions and procedures implement the methods. Most users of earlier versions of Access wrote Access macros to implement methods, because they deemed writing macros to be simpler than writing Access 1.x, Access Basic, or Access 2.0 VBA code.

Oracle Ex. 1009, p. 805
Oracle v. DeBellis
IPR2024-00221

Access is unique among today's mainstream PC database managers because, by default, Access saves application objects (the queries, forms, reports, macros, and VBA code that you create for the database) within the database file itself, not in separate SC, PRG, or .exe files as do other PC RDBMSs. Access data objects (tables) have self-contained properties and methods; most other PC RDBMSs require separate programs to validate data, display status text, and create indexes. Therefore, Access database files and the tables they contain conform to the object *paradigm*—a synonym for the word *model* that has become an object-oriented cliché.

Note

As observed in Chapter 25, "Securing Multiuser Network Applications," it has become a generally accepted database design practice (GADDP) to use separate .mdb files to contain application objects and data objects. Keeping your data object (tables) in a *Tables*.mdb file and linking the tables to your *AppObjs*.mdb file lets you update the application objects without affecting the existing data in the table objects.

A separate *Tables*.mdb file in Access 2.0 format is necessary if you must continue to support users running 16-bit Windows 3.1+. (You must also maintain a duplicate of your *AppObjs*.mdb file in Access 2.0 format, which is a major issue among Access developers.) Issues relating to 16-bit and 32-bit Access application versioning are covered in Chapter 33, "Migrating Access 2.0 and Access 95 Applications to Access 97."

▶▶ See "Splitting Databases for File Sharing," p. 873



Combining Different Entities in a Single Table. You can include representations of different types of objects in a single table as long as you can represent their properties and behavior in the same manner and yet distinguish between the different object types. For example, checks and debits are shown as a single data-object type in preceding Figure 23.1, although one originates from a paper check and the other from an Electronic Funds Transfer debit. A Debit Type field can indicate the different sources. You can combine cash deposits and transfers from a savings account into a single data-entity type in a Credits table. You might want to combine both debits and credits in a single table, which you can do by using different codes for Debit Types and Credit Types.

To identify a debit or credit uniquely, you need to include fields for bank ID, customer number, debit/credit type, and transaction number. Although a check number can serve as the transaction number, the system must assign transaction numbers to other types of transactions, such as those conducted at ATMs. Access 97 can use an AutoNumber field, called a Counter field in Access 2.0 and earlier, to add a unique transaction number (either incremented or random) to each data entity, including checks. The check number becomes a separate attribute.

Database Terminology

The terms used to describe formally a database and the elements that comprise it derive from four different sources. The data-description language, of which entity and attribute

Oracle Ex. 1009, p. 806
Oracle v. DeBellis
IPR2024-00221

are members, derives from the terminology of statistics. Another set of terms, which describe the same set of elements, is based on computer terminology and relates to how the elements are stored within disk files. Query by example introduced new terms—for example, *row*, *column*, and *cell*—to the language of databases. Structured Query Language adopted these terms.

Table 23.1 compares words that are used for data description, in QBE and SQL, and for describing data-storage methodologies employed by Access, xBase, and Paradox. The Access VBA language takes an object-oriented approach to programming, so Table 23.1 also includes terms that are applicable to object-oriented programming (OOP).

Table 23.1 A Comparison of Data-Description and Data-Storage Terminology

Data Description	QBE and SQL	Object-Oriented	Access Storage	xBase/ Paradox
Heterogeneous Universe	Database Directory	Base Object Class	File	
Universe (homogeneous)	Table	Object Class	Table (Sub-File)	Data File
Entity (object, instance)	Row	Data Object	Record	Record
Attribute	Cell	Object Property	Field	Field
Attribute Data Type	Datatype	Data Type	Field Data Type	Field Type
Attribute Domain	Validation Rule Statement	Enumeration	Validation Rule	Valid
Attribute Value	Cell Value	Property Value	Field Value	Field Value
Identifier	Primary Key	Property Value	Key, Index	Index File

The real-world object is the basic source of information that is represented in a database as an entity. In explaining the terms included in Table 23.1, therefore, the following definition list begins with an entity, breaks it down into its component parts, and then establishes its position in the hierarchy of databases and tables:

- *Entity.* A unique representation of a single real-world object that is created by using the values of its attributes in computer-readable form. To ensure uniqueness, one or more of an entity's attributes must have values unlike the corresponding values of any other entity of the same class. An entity corresponds to a row in QBE and SQL or a record in data-storage terminology. Entities are also called data entities, data objects, data instances, or instances.
- *Attribute.* A significant property of a real-world object. Every attribute carries a value that assists in identifying the entity of which it is a part and in distinguishing the entity from other members of the same entity class. Attributes are contained in fields (data-storage terminology) or columns (QBE and SQL). An attribute also is called a *cell* or *data cell*, terms that describe the intersection of a row and a column (or a field and a record).

Oracle Ex. 1009, p. 807
Oracle v. DeBellis
IPR2024-00221

- *Attribute data type.* Basic attribute data types consist of all numeric (integer, floating-point, and so forth) and string (text or alphanumeric) data types without embedded spaces or separating punctuation. The string data type can contain letters, numbers, and special characters (such as those used in languages other than English). An attribute with a basic attribute data type is indivisible and called an *atomic type.*

  Text data types with spaces or other separating punctuation characters are called *composite attribute data types.* You can divide most composite types into basic data types by parsing. Parsing means to separate a composite attribute into basic attributes. For example, you can parse "Smith, Dr. John D., Jr." to Last Name (Smith), Title (Dr.), First Name (John), Middle Initial (D.), and Suffix (Jr.) basic attribute types. Special field types, such as Memo and OLE, are composite attribute data types that cannot be parsed to basic data types by conventional methods. You cannot, therefore, create Access indexes that include Memo or OLE Object attribute data types; only attributes with basic attribute data types can be indexed.

- *Attribute domain.* The allowable range of values for an attribute of a given attribute data type. The attribute data type determines the domain unless the domain is limited by a process that is external to the data in the table. As an example of attribute domain limitation, the domain of an employee age attribute that has an integer data type might be limited by a data validation method to any integer greater than 13 and less than 90. In object-oriented terms, the domain consists of an enumeration of acceptable values. A days-of-the-week enumeration (the domain of days) consists of a list of its members: Monday, Tuesday, and so on. Access validation rules, stored in tables, maintain domain integrity, limiting data entry to limits set by the data validation expression.

- *Attribute value.* The smallest indivisible unit of data in an entity. Attribute values are limited to those within the attribute domain. Cell value and data value are synonyms for attribute value.

- *Identifier.* An attribute or combination of attributes required to identify a specific entity (and no others) uniquely. Identifiers are called *primary-key fields* in Access and are used to create the primary index of the entities. When an entity's attribute values are duplicated in other entities' corresponding attributes, you need to combine various attributes to ensure a unique identifier for the entity. When more than one attribute is used as an identifier, the key fields are called a *composite* or *compound primary key.*

- *Homogeneous universe.* The collection (set) of all data entities of a single data entity type. The data entities must have an identical set of attributes, attribute data types, and attribute domains. This set corresponds to an Access or Paradox table or a data file in xBase. The set also is called an *entity class* or *entity type,* and its members are sometimes called *entity instances,* or just *instances.*

- *Heterogeneous universe.* The collection (set) of related entity classes comprising related homogeneous universes—the database. A database is stored as a single file in

Oracle Ex. 1009, p. 808
Oracle v. DeBellis
IPR2024-00221

Access and most client/server databases. Paradox and xBase store databases as collections of related files, usually in a single directory. Access databases include a special table that catalogs the objects the databases contain. You can reveal the content of the catalog by using the techniques described in the section "Access's Integrated Data Dictionary System" later in this chapter.

Much of the formal terminology used to describe data objects in relational databases is quite technical and rather abstract. You need to understand the meaning of these terms, however, when you create the data models that form the basis of the design of your database.

Types of Tables and Keys in Relational Databases

Specific to relational databases are certain types of tables and keys that enable relationships between tables. Understanding these tables and keys is essential to comprehending relational databases and the rules of data normalization, which are discussed in the section "Normalizing Data to the Relational Model" later in this chapter. The following list defines the various relational keys and tables:

- *Base table.* In a relational database, a base table is the table that incorporates one or more columns of an object's properties and contains the primary key that uniquely identifies that object as a data entity. A base table must have a primary key. Base tables are often called *primary tables* because of the requirement for a primary key.

> **Note**
>
> Microsoft often uses the term *base table* to refer to a table in the native Jet MDB database structure. This book uses the term *base table* in accordance with the preceding definition.

- *Relation table.* A table that is used to provide linkages between other tables and isn't a base table (because it doesn't incorporate properties of an object or have a primary key field) is called a *relation table.* Key fields in relation tables each must be foreign keys related to a primary key in a base table.

  Technically, a true relation table is comprised wholly of foreign keys and contains no independent data entities. The Order Details table of the Northwind Traders database is an example of a relation table that contains data values that aren't foreign keys (the UnitPrice and Quantity fields, for example). Its OrderID field is related to the field of the same name in the Orders table. Likewise, the ProductID field is related to the ProductID field of the Products table. Although the Order Details table has a composite key, it isn't a true primary key; its purpose is to prevent duplication of a product entry in a specific order.

- *Primary key.* A primary key consists of a set of values that uniquely specifies a row of a base table, which in Access is the primary table. For any primary-key value, one and only one row in the table matches this value. You can base the primary key on a single field if each data cell's value is unique at all times.

- *Candidate keys.* Any column or group of columns that meets the requirements for a primary key is a candidate to become the primary key for the table. Name and

Oracle Ex. 1009, p. 809
Oracle v. DeBellis
IPR2024-00221

Social Security number are candidate keys for identifying a person in the United States; however, the Social Security number is the more appropriate choice because two people can have the same name but not the same valid Social Security number.

- *Composite keys.* If you need data from more than one column of the table to meet the uniqueness requirement of a primary key, the key is said to be a composite or concatenated key.
- *Foreign keys.* A foreign key is a column whose values correspond to those contained in a primary key, or the far-left portion of a composite key, in another related table. A foreign key can consist of one column or group of columns (a composite foreign key). If the length of a foreign key is less than the corresponding primary key, the key is called a *partial* or *truncated foreign key*.

Examples of the preceding keys and tables occur in the discussions of normal forms in the section "Normalizing Data to the Relational Model" later in this chapter. First, the following sections examine the process of data modeling.

Data Modeling

The first step in designing a database is to determine which objects to represent within the database and which of the objects' properties to include. This process is called *data modeling*. The purpose of a data model is to create a logical representation of the data structure that is used to create a database. Data modeling can encompass an entire organization, a division or department, or a single type of object. Models that deal with objects, rather than the tables that you later create from the objects, are called *conceptual data models*.

Figure 23.2 illustrates two different approaches (conceptual data models) to database design: the bottom-up approach to create an application database, and the top-down method to develop subject databases. These two approaches, discussed in the following sections, result in databases with quite different structures.

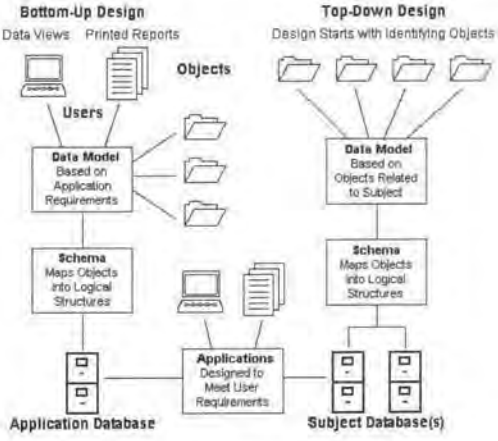


FIG. 23.2 A comparison of bottom-up and top-down database designs.

Oracle Ex. 1009, p. 810
Oracle v. DeBellis
IPR2024-00221

Application Databases. You can base data models on specific needs for data presented in a particular manner. For such a needs-based model, you can use the bottom-up approach and start with a view of the data on a display, a printed report, or both, as shown in the left side of Figure 23.2. This approach results in an application database.

If you're creating a simple database for your own use or dealing with a single type of data object, the bottom-up approach might suffice because the presentation requirements and properties of the objects involved are usually well-defined. The problem with the bottom-up approach is that it often leads to multiple individual databases that may duplicate one another's information. Several persons or groups within an organization might have a requirement for an application database that includes, for example, a customer's table. When a new customer is added—or data for an existing customer is changed—in one application database, you need to update each of the other application databases. The updating process is time-consuming and subject to error.

Conceptual data models, such as those shown in Figure 23.2, are independent of the database manager you use and the type of database files it accesses. Therefore, the same data model accommodates databases in Access's native format as well as others with which Access is compatible. Data models aren't connected with any programming language or tools used to create applications. The applications box in Figure 23.2 isn't a component of conventional data models but is added to show where application design fits into the overall picture.

Subject Databases. A better approach is to base the design of the database on groups of objects that are related by subject matter. For a manufacturing firm, tables are usually grouped into databases devoted to a single department or function. The following lists some database examples:

- Sales database consisting of customer, sales order, sales quota, product discount, and invoice tables.
- Production database including product, price, parts, vendor, and cost accounting tables.
- Personnel database with employee, payroll, and benefits tables (large firms may include tables relating to health care providers and employment applicants).
- Accounting database incorporating general ledger and various journal tables.

Databases that consist of tables relating to a single class of subjects or functions are called *subject databases*. Even if you are creating the first database application for a small organization, starting with an overall plan for the organization's total information requirements in subject databases pays long-term dividends. If you decide or are assigned to create an invoicing application, for instance, you can establish sales, production, and personnel databases from the beginning, rather than split up a single invoice database at a later time and rewrite all of your applications to access tables within multiple databases.

Oracle Ex. 1009, p. 811
Oracle v. DeBellis
IPR2024-00221

Subject databases require top-down design, depicted in the right-hand diagram of Figure 23.2. In this case, the properties of the data objects, not the applications used with them, determine the design. Designing subject databases involves creating a diagram of the relevant objects and the associations between them and then creating models for each database involved. You distribute the model diagrams to users and then interview the users to determine their information needs based on the content of the model databases.

Diagrammatic Data Models. Large, complex data models resemble the work-flow and paper-flow diagrams commonly used in analyzing organizations' administrative procedures. If you have such diagrams or descriptions, they make the data-modeling process much easier. Generating an organization-wide data model may involve a substantial amount of research to determine the needs of the organization as a whole and of individuals using specialized applications. In many cases, users and potential users aren't able to define what information they need or how they want to see it presented.

Many methods exist of creating diagrams to represent data models. One of the more useful methods is the Entity-Relationship (E-R) diagram, developed by Peter Chen in 1976 and expanded on by David R. McClanahan in a series of articles entitled "Database Foundations: Conceptual Designs" in *DBMS* magazine (see Figure 23.3). You can use E-R diagrams to represent relationships between objects and to depict their behavior.



FIG. 23.3 An Entity-Relationship diagram of two data entities from Figure 23.1.

Data entities are enclosed within rectangles, data attributes within ovals, and relationships between entities within diamonds. Relationships between database objects, at the conceptual stage, can be defined by their behavior; therefore, E-R diagrams include at least one verb whose object, unless otherwise indicated, is to the right of the diamond relationship symbol. You add symbols to the diagram as the model's detail increases. One of the advantages of the E-R diagram is that you can use it to represent the conceptual design of very large systems with multiple databases in a relatively small amount of space.

Database Schema. A graphic description of the layout of tables in the form of bars that contain their field names and show a simplified version of relationships between them can be employed to aid users to grasp the concept of the database. A diagram that shows the logical representation of data is called a *schema*. A schema, such as the one shown in Figure 23.4 for an ocean shipping line, is independent of the RDBMS used to implement the database.

Oracle Ex. 1009, p. 812
Oracle v. DeBellis
IPR2024-00221

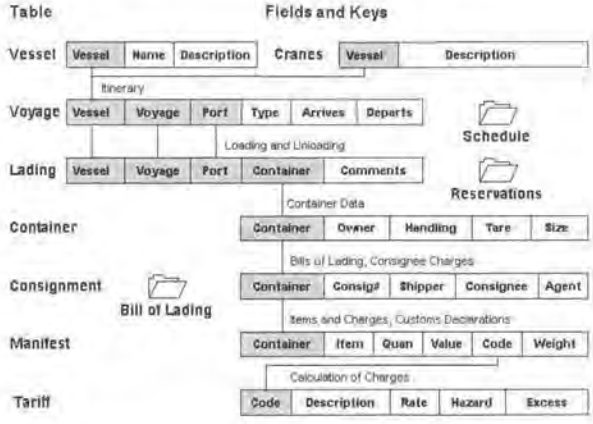


FIG. 23.4 The schema for the operations database of a shipping line.

In Figure 23.4, the primary keys are shaded, and the relationships between the table keys are indicated by lines that connect the keys. Foreign keys are unshaded, except when they correspond to a component of a composite primary key. The descriptions shown between the bars are optional; they are useful in describing the relationships to users. You can expand a schema of this type to include the source documents involved, the reports to be generated, and the applications that pertain to all (or a portion) of the tables.

External Determinants of Database Design. Finding the data objects that provide the information to meet all of an organization's requirements may require extensive detective work. Many objects might not be available within the organization itself. For instance, if you're developing a database application that involves geographic positioning (called *geocoding*), you might need map tables such as the U.S. Census Bureau's TIGER/Line files, or tables derived from those files by others.

Images to be incorporated as OLE objects might not be available in file formats that are compatible with your OLE server applications; they would require file-type conversion. If the accounting department is using packaged accounting software, you must incorporate representations of the structure of its database into your model. In this case, you also must plan how to exchange information with the accounting data, but you need not include the access methodology in your conceptual data model.

Using Data Modeling Tools to Create Access Databases

Data modeling tools are available from several publishers for designing and then automatically creating the structure of client/server relational databases, such as SQL Server, ORACLE, and SQLBase. Data modeling tools that run under Windows let you use graphic techniques, such as E-R diagrams, to design the structure of the tables and establish relationships between the tables. Database CASE tools save much time and prevent many errors when you implement a large and complex relational database.

Oracle Ex. 1009, p. 813
Oracle v. DeBellis
IPR2024-00221

Note

Data modeling tools often are called *Computer-Aided Software Engineering* (*CASE*) tools. CASE tools designed for enterprise-wide software design usually include data modeling as one of their features. Today's high-end CASE tools, some of which carry license fees of $100,000 or more, offer many additional capabilities. This book positions data modeling tools as a subset of CASE tools.

Most data modeling tools contain a *repository* that stores information about table design, primary and foreign key fields, constraints (validation rules) for fields, and types of relationships between the tables. The repository is a database maintained by the data modeling application itself. You can print database schema and generate data dictionaries from records in the repository. When your database design appears satisfactory, the data modeling tool translates the data in the repository to a SQL Data Definition Language (DDL) statement. You send the DDL statement to the client/server RDBMS on the server. This RDBMS can be used to create the entire database or just to add tables to the database. SQL's DDL commands are one of the subjects of the next chapter.

InfoModeler, published by Asymetrix Corporation of Bellevue, Washington, is the most popular data modeling tool for Access databases. InfoModeler uses a new approach to designing databases called *Object Role Modeling* (*ORM*), developed by Professor Terry Halpin of the University of Queensland, Australia. ORM lets you express the design of a database in simple, English terms through a structured language called *Formal Object Role Modeling Language* (FORML). InfoModeler translates FORML statements into a graphic schema, related to but more flexible than E-R diagrams. Figure 23.5 shows a portion of the ORM database diagram for a tutorial application of InfoModeler 2.0 derived from the pubs sample database included with Microsoft SQL Server 6.x.

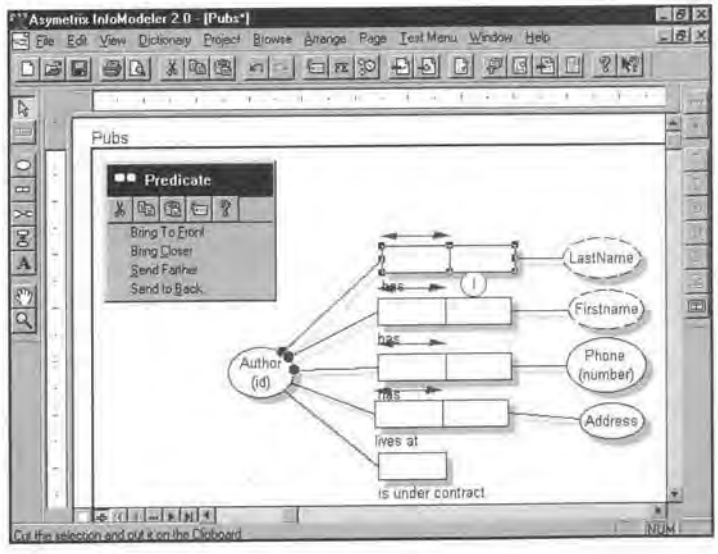


FIG. 23.5 An Object Role Modeling diagram for an Access database.

Oracle Ex. 1009, p. 814
Oracle v. DeBellis
IPR2024-00221

InfoModeler lets you print the ORM graphic schema and creates a data dictionary for the database. This reduces the time required to describe the structure of the database to others, as well as much of the drudgery of creating a comprehensive data dictionary. 32-bit InfoModeler 2.0 for Windows 95 and Windows NT 4.0 offers a number of other useful features, such as reverse-engineering your Access database (creating an ORM schema from an .mdb file or client/server database), database structure version tracking, and automatic restructuring of databases and tables.

Normalizing Data to the Relational Model

Up to this point, most of the subject matter in this chapter has been applicable to any type of database—hierarchical, relational, or even the new class of object database systems. However, because Access is a RDBMS, the balance of the chapter is devoted to relational databases. Access fully implements the relational model in its native database structure and you can link tables from other RDBMSs—including client/server tables—to Access databases. Thus, the discussion that follows is general in nature and applies to any database system with which Access is compatible or for which you have the appropriate 32-bit Open Database Connectivity (ODBC) driver. (Access 97 requires the use of 32-bit ODBC drivers.)

The theory of relational database design is founded in a branch of mathematics called *set theory*, with a great deal of combinatorial analysis and some statistical methodology added. The set of rules and symbols by which relational databases are defined is called *relational algebra*. This chapter doesn't delve into the symbolic representation of relational algebra, nor does it require you to comprehend advanced mathematics. The chapter does, however, introduce you to many of the terms used in relational algebra for the sake of consistency with advanced texts, which you may want to consult on the subject of database design.

Normalization Rules

Normalization is a formalized procedure by which data attributes are grouped into tables and tables are grouped into databases. The purposes of normalization include the following:

- Eliminating duplicate information in tables.
- Accommodating future changes in the structure of tables.
- Minimizing the impact of database structural change on user applications that access the data.

Normalization is performed in steps; the first three and most common steps were described by Dr. E. F. Codd in his 1972 paper, "Further Normalization of the Data Base Relational Model." These steps are depicted in Figure 23.6. The following sections describe each of the five steps that comprise the entire normalizing process.

Oracle Ex. 1009, p. 815
Oracle v. DeBellis
IPR2024-00221

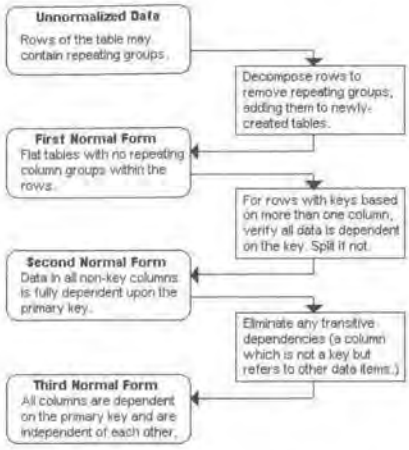


FIG. 23.6 A graphic representation of relational database normalization to the third normal form.

First Normal Form. *First normal form* requires that tables be flat and contain no repeating groups. A flat table has only two dimensions—length (number of records or rows) and width (number of fields or columns)—and cannot contain data cells with more than one value. For a single cell to contain more than one data value, the representation of the cell's contents requires a third dimension, depth, to display the multiple data values. Flat tables and the flat-file databases referred to in Chapter 7, "Linking, Importing, and Exporting Tables," are similar in that both have two dimensions. Flat-file databases, however, consist of only one table and have no restrictions on the content of the data cells within the table.

> **Note**
>
> Access 97's Table Analyzer Wizard does a good job of detecting duplicate information in tables created from flat files, but the files must be in first normal form (no repeating groups) for the analysis to succeed. The "Using Access 97's Table Analyzer Wizard" section later in this chapter describes how to use the Table Analyzer Wizard to check for duplicate data and create a set of related tables to minimize or eliminate the duplication.

An example of unnormalized data for a shipping line appears in Figure 23.7. This presentation often is seen in the schedules published by transportation firms where the stops are displayed across the page. This example is representative of a schedule created by importing the worksheet file that was used to create the printed version of the schedule. In the various examples of tables that follow, missing borders are the equivalent of an ellipsis; that is, a missing right border indicates that additional columns (fields) exist beyond the far-right column. A missing bottom border means that more rows (records) follow. (Those readers who are seasoned mariners will recognize the example as a mythical schedule for vessels of the former Pacific Far East Lines.)

Oracle Ex. 1009, p. 816
Oracle v. DeBellis
IPR2024-00221

Vessel	Name	Voyage	Embarks	From	Arrives	Port	Departs	Arrives	Port	Departs
528	Japan Bear	9203W	5/31/92	SFO	6/6/92	HNL	6/8/92	7/15/92	OSA	7/18/92
603	Korea Bear	9203W	6/05/92	OAK	6/19/92	OSA	6/21/92	6/25/92	INC	6/28/92
531	China Bear	9204W	6/20/92	LAX	7/10/92	PAP	7/11/92	8/28/92	SYD	9/2/92
528	Japan Bear	9204W	8/20/92	SFO	8/27/92	HNL	8/29/92	9/30/92	OSA	10/2/92

FIG. 23.7 A partial schedule of voyages for a shipping line.

Because the vessels stop at a number of ports, the Arrives, Port, and Departs columns are duplicated for each stop in the voyage. This type of data structure is allowed in COBOL, where the repeating group (Arrives, Port, and Departs) `OCCURS` any number of `TIMES`, but not in relational databases. The data in the preceding schedule isn't in first normal form because it contains repeating groups. The table must be decomposed (divided) into two tables, therefore, with the repeating groups (shown in shaded type in Figure 23.7) removed from the Schedule table and placed in two new tables, Ports and Vessel Voyages, as shown in Figure 23.8.

Vessel	Name	Voyage	Embarks	From
528	Japan Bear	9203W	5/31/92	SFO
603	Korea Bear	9203W	6/5/92	OAK
531	China Bear	9204W	6/20/92	LAX
528	Japan Bear	9204W	8/20/92	SFO

Arrives	Port	Departs
6/6/92	HNL	6/8/92
6/19/92	OSA	6/21/92
7/10/92	PAP	7/11/92
8/27/92	HNL	8/29/92
7/15/92	OSA	7/18/92
6/25/92	INC	6/28/92
8/28/92	SYD	9/2/92
9/30/92	OSA	10/2/92

FIG. 23.8 The Ports and Vessel Voyages tables created from the Schedule table.

Now you need to provide for a link between the Ports and Vessel Voyages tables to retain the relationship between the data. This shipping line numbers voyages for each vessel with the year and which voyage this is for the year, as well as the general direction of travel (9204W is the fourth voyage of 1992, westbound). Thus, both Vessel and Voyage need to be used to relate the two tables. Neither Vessel nor Voyage is sufficient in itself because a vessel has multiple voyages during the year and the voyage numbers used here recur for other vessels. Because you must create a new Ports table to meet the requirements of the first normal form, you have the chance to order the columns in the order of their significance. Columns used to establish relationships are usually listed first, in the sequence in which they appear in the composite primary key, when more than one column is included in the key (see Figure 23.9).

Vessel	Voyage	Port	Arrives	Departs
528	9203W	HNL	6/6/92	6/8/92
603	9203W	OSA	6/19/92	6/21/92
531	9204W	PAP	7/10/92	7/11/92
528	9204W	HNL	8/27/92	8/29/92
528	9203W	OSA	7/15/92	7/18/92
603	9203W	INC	6/25/92	6/28/92
531	9204W	SYD	8/28/92	9/2/92
528	9204W	OSA	9/30/92	10/2/92

FIG. 23.9 Linking fields are added to the Ports relation table.

Next, you establish the key fields for the Ports table that uniquely identify a record in the table. You need a primary key for the Ports table because other tables may be dependent on this table. Clearly, Vessel and Voyage must be included because these columns constitute the relationship to the Vessel Voyages table. You need to add the Port field to create a unique key (Vessel + Voyage can have duplicate values). Vessel + Voyage + Port creates

Oracle Ex. 1009, p. 817
Oracle v. DeBellis
IPR2024-00221

a unique composite primary key because the combination takes into account stopping at a port twice—when returning eastbound, the voyage carries an "E" suffix.

> **Note**
>
> A spreadsheet application, such as Microsoft Excel 97, can speed up the process of normalizing existing data, especially when the data contains repeating groups. Import the data into a worksheet, then cut and paste the data in the repeating groups into a new worksheet. When the data for both of the tables is normalized, save the worksheets and then import the files to Access tables. This process is usually faster than creating make-table queries to generate normalized tables.

Second Normal Form. *Second normal form* requires that data in all non-key columns be fully dependent on the primary key and each element (column) of the primary key when it is a composite primary key. *Fully dependent* means that the data value in each non-key column of a record is determined uniquely by the value of the primary key. If a composite primary key is required to establish the uniqueness of a record, the same rule applies to each value of the fields that comprise the composite key of the record. Your table must be in first normal form before examining it for conformity to second normal form. Second normal form removes much of the data redundancy that is likely to occur in a first normal table.

Returning to the Vessel Voyages table, you can see that it requires a composite key, Vessel + Voyage, to create a unique key because the vessel number and vessel name recur. When you create such a key, however, you observe that Vessel and Name aren't dependent on the entire primary key because neither is determined by Voyage. You also find that the vessel name occurs for each of a vessel's voyages; for example, the Japan Bear appears twice. This lack of dependency violates the rules of the second normal form and requires Vessel Voyages to be split into two tables, Vessels and Voyages. One row is required in the Vessels table for each ship, and one row is required in the Voyages table for each voyage made by each ship (eastbound and westbound directions are considered separate voyages for database purposes). As was the case for Ports, a unique key is required to relate voyages to the vessel, so the vessel number column is added to the Voyages table, as shown in Figure 23.10.

Vessel	Vessel Name
528	Japan Bear
603	Korea Bear
531	China Bear

Vessel	Voyage	Embarks	From
528	9203W	5/31/92	SFO
603	9203W	6/5/92	OAK
531	9204W	6/20/92	LAX
528	9204W	8/20/92	SFO

FIG. 23.10 The Vessels and Voyages tables created from the Vessel Voyages table.

Third Normal Form. *Third normal form* requires that all non-key columns of a table be dependent on the table's primary key and independent of one another. Tables must conform to both first and second normal forms to qualify for third normal status.

Your Vessels and Voyages tables are now in third normal form because there are no repeating groups of columns, and the data in non-key columns is dependent on the primary key field. The non-key columns of Ports, Arrives, and Departs are dependent on the

Oracle Ex. 1009, p. 818
Oracle v. DeBellis
IPR2024-00221

composite key (Vessel + Voyage + Port) and independent of one another. Ports, therefore, meets the requirements of first, second, and third normal forms. The departure date is independent of the arrival date because the difference between the two dates is based on the vessel's lading into and out of the port, the availability of berths and container cranes, and the weather.

To demonstrate normalization to the third normal form, suppose that you want to identify the officers of the vessel—master, chief engineer, and so on—in the database. Your first impulse might be to add their employee numbers, the primary key of an Employee table, to the Vessels table (see Figure 23.11).

Vessel	Vessel Name	Master	Chief	1st Mate
528	Japan Bear	01023	01155	01367
603	Korea Bear	00955	01203	00823
531	China Bear	00721	00912	01251

FIG. 23.11 A table with a transitive dependency between vessels and crew members.

This table violates the third normal rule because none of the officers assigned to a vessel is dependent on the vessel itself. This type of dependency is called *transitive*. The master's, chief's, and first mate's maritime licenses allow them to act in their respective capacities on any vessel for which the license is valid. Any officer may be assigned to other vessels, as the need arises, or remain on board for only a portion of the voyage.

One method of removing the transitive dependency might be to add the employee numbers column to the Voyages table. This method doesn't provide a satisfactory solution, however, because the vessel may arrive at a port with one group of crew members and depart with another group. In addition, you need to specify the crew members who remain with the vessel while it is in port. A relation table, such as that shown for the Japan Bear in Figure 23.12, solves the problem. Duplicate values in the Port (departure port) and To (destination port) fields designate records for crew members responsible for the vessel while in port. The Crew table of Figure 23.12 qualifies as a relation table because all of its fields correspond to primary keys or parts of primary keys in the base tables—Vessels, Voyages, Ports, and Employees.

Vessel	Voyage	Port	To	Master	Chief	1st Mate
528	9203W	SFO	HNL	01023	01156	01367
528	9203W	HNL	HNL	01023	01156	01367
528	9203W	HNL	OSA	01023	01156	01367
528	9203W	OSA	OSA	01023	01156	01367
528	9203W	OSA	INC	01023	01156	01367

FIG. 23.12 Removing transitive dependency with a relation table.

All of your tables are now flat, contain no duplicate information other than that in the columns used for keys, and conform to the first through third normal forms.

Fourth Normal Form. *Fourth normal form* requires that independent data entities not be stored in the same table when many-to-many relationships exist between these entities. The table of Figure 23.12 violates fourth normal form because many-to-many relationships exist between the Vessel and the fields that identify crew members. The fourth

Oracle Ex. 1009, p. 819
Oracle v. DeBellis
IPR2024-00221

normal form is discussed in the "Many-to-Many Relationships and Fourth Normal Form" section later in this chapter, because it is the only normalization rule that is dependent on a specific type of relationship.

> **Note**
>
> Many database designers disregard the fourth and fifth normal forms; those designers consider these forms too esoteric or applicable only in specialized cases. Disregarding fourth normal form often results in poorly designed databases, but not necessarily malfunctioning ones.

Fifth Normal Form and Combined Entities. *Fifth normal form* requires that you be able to reconstruct exactly the original table from those tables into which it was decomposed. Re-creating the Excel spreadsheet from the tables in the example in Chapter 21, "Using Access with Microsoft Excel," demonstrates compliance with fifth normal form. Fifth normal form requires that the tables comply with the rules for third normal form and, when many-to-many relationships are present, with the rule for fourth normal form.

◀◀ See "Verifying the Tables by Re-Creating the Worksheet," p. 746



The Voyages table appears to be quite similar to that of Ports. The From column is equivalent to Port, and Embarks is the same as Departure. Therefore, you can move the data in the Voyages table to the Ports table and delete the Voyages table. Figure 23.13 shows the new Ports table. The rows from the Voyages table don't have values in the Arrives column because they represent points of departure.

Vessel	Voyage	Port	Arrives	Departs
528	9203W	HNL	6/6/92	6/8/92
603	9203W	OSA	6/19/92	6/21/92
531	9204W	PAP	7/10/92	7/11/92
528	9204W	HNL	8/27/92	8/29/92
528	9203W	OSA	7/15/92	7/18/92
603	9203W	INC	6/25/92	6/28/92
531	9204W	SYD	8/28/92	9/2/92
528	9204W	OSA	9/30/92	10/2/92
528	9203W	SFO		5/31/92
603	9203W	OAK		6/5/92
531	9204W	LAX		6/20/92
528	9204W	SFO		8/20/92

FIG. 23.13 Records from the Voyages table appended to the Ports table.

However, you cannot explicitly reconstruct the original table from the combined Voyages and Ports tables in all cases, because you cannot distinguish an embarkation row from the other rows by a value in the table. A Null value in the Arrives field is a candidate to distinguish an embarkation, but most PC RDBMSs don't support Null values. You eliminate any ambiguity that using a Null value might cause—and bring the table into fifth normal form—by adding a single-character field, Type, with single-letter codes to define the type of call. In Figure 23.14, the codes E and S represent Embarkation and Scheduled call, respectively. Other codes might include M for Maintenance stop and R for Return voyage.

Oracle Ex. 1009, p. 820
Oracle v. DeBellis
IPR2024-00221

Vessel	Voyage	Port	Type	Arrives	Departs
528	9203W	HNL	S	6/6/92	6/8/92
603	9203W	OSA	S	6/19/92	6/21/92
531	9204W	PAP	S	7/10/92	7/11/92
528	9204W	HNL	S	8/27/92	8/29/92
528	9203W	OSA	S	7/15/92	7/18/92
603	9203W	INC	S	6/25/92	6/28/92
531	9204W	SYD	S	8/28/92	9/2/92
528	9204W	OSA	S	9/30/92	10/2/92
528	9203W	SFO	E		5/31/92
603	9203W	OAK	E		6/5/92
531	9204W	LAX	E		6/20/92
528	9204W	SFO	E		8/20/92

FIG. 23.14 The Type field has been added to comply with fifth normal form.

Figure 23.15 demonstrates that you can reconstruct the content of the original Schedule table from the Vessels and Ports tables. Query1 creates the first five columns of the Schedule table by adding the criterion E for the Type field, which isn't shown. You can re-create the remaining columns of the Schedule table from Query2, which uses the criterion S for the Type field.

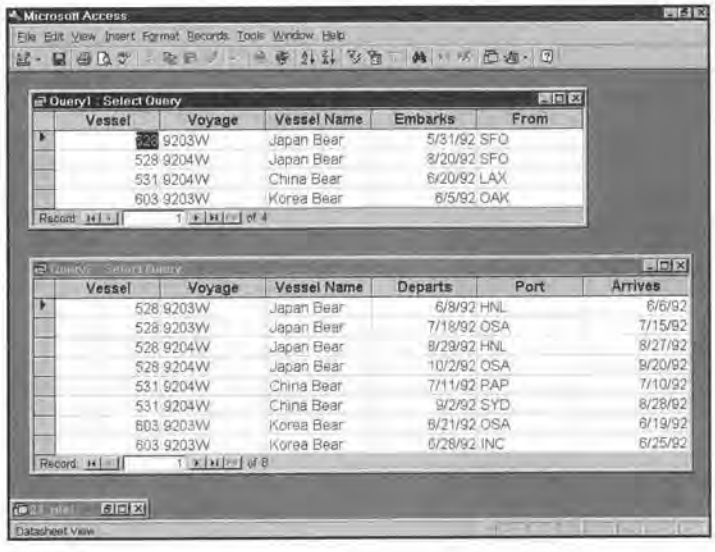


FIG. 23.15 The datasheets of the two queries required to reconstruct the Schedule table.


On the Web

The database used to create Figure 23.15—23_pfel.mdb—is included in the example files for this book, available from the Macmillan Publishing USA Information SuperLibrary Web site. Follow the links from this URL to the *Special Edition Using Access 97* page for downloading instructions:

http://www.mcp.com/que/msoffice/

The example files are archived in ZIP format. You can download an evaluation copy of the latest 32-bit version of WinZip, which expands the ZIP files, from

http://www.winzip.com/

Oracle Ex. 1009, p. 821
Oracle v. DeBellis
IPR2024-00221

Types of Relationships

The subject of relationships between entities usually precedes discussions of normalization. Relationships come second in this book, however, because you can only create valid relationships between tables that have been structured in accordance with at least the first three normalization rules described in the preceding sections. This section describes the four basic types of relationships between tables and employs E-R diagrams to depict the relationships graphically.

One-to-One Relationships. The simplest relationship between tables is a one-to-one relationship. In such a relationship, the tables have exact one-to-one row correspondence; no row in one table has more than one corresponding row in the other table. You can combine one-to-one-related tables into a single table consisting of all the tables' columns.

One-to-one relationships are often used to divide very wide base tables into narrower ones. You might want to divide a wide table to reduce the time needed to view fields containing specific sets of data, such as the stock prices table in the example of Chapter 15, "Preparing Advanced Reports." Often you need to control access to the parts of tables that contain sensitive or confidential data. An example is an employee file: everyone might have read-only access to the employees' names, but only members of the personnel department are authorized to view salary and other payroll information (see Figure 23.16).

Employee	Position	Last	First	MI
00668	Master	Johansson	Lars	F.
00721	Master	Karlsson	Bo	B.
00885	Chief	MacGregor	Paul	C.
00912	Chief	McDermott	John	R.
00955	Master	Olafson	Karl	T.
01023	Master	Kekkonen	Eino	K.
01156	Chief	McDougal	William	U.
01203	Chief	Kashihara	Matsuo	

Employee	Salary
00668	6500.00
00721	6250.00
00885	5100.00
00912	5000.00
00955	6100.00
01023	6050.00
01156	4900.00
01203	4850.00

FIG. 23.16 Two tables with a one-to-one relationship.

Figure 23.17 shows the E-R diagram for the Employees and Salaries tables. The number 1 added to each side of the relationship diamond indicates a one-to-one relationship. The participation of entities in relationships can be mandatory or optional. Optional relationships are symbolized by a circle drawn on the line connecting the optional entity with the relationship diamond. In the figure, the Paid-Salaries relationship is optional because some employees can be paid on an hourly basis and linked to a Wages table. Tables with mandatory one-to-one relationships are base tables. A table with an optional one-to-one relationship to a base table is a related table. Multiple tables with one-to-one relationships where the corresponding records in the other tables are optional can reduce the database's disk space requirement.

Another example of a one-to-one relationship is between an xBase memo field and a corresponding entry in a memo .dbt file. Access treats free text as the content of a data cell in the table, so no relationship is involved.

Oracle Ex. 1009, p. 822
Oracle v. DeBellis
IPR2024-00221



FIG. 23.17 An E-R diagram for an optional one-to-one relationship.

One-to-Many Relationships. One-to-many relationships link a single row in one table with two or more rows in another table through a relationship between the primary key of the base table and the corresponding foreign key in the related table. Although the foreign key in the table containing the many relationships may be a component of a composite primary key in its own table, it is a foreign key for the purposes of the relationship. One-to-many relationships are the most common relationships.

The one-to-many relationship shown in Figure 23.18 links all records in the Ports table to one record in the Vessels table. The one-to-many relationship allows you to display all records in the Ports table for scheduled ports of call of the Japan Bear.

Vessel	Vessel Name
528	Japan Bear

Vessel	Voyage	Port	Type	Arrives	Departs
528	9203W	HNL	S	8/6/92	8/8/92
528	9204W	HNL	S	8/27/92	8/29/92
528	9203W	OSA	E	7/15/92	7/18/92
528	9204W	OSA	S	9/30/92	10/2/92
528	9203W	SFO	E		5/31/92
528	9204W	SFO	E		8/20/92

FIG. 23.18 A one-to-many relationship between the Vessels and Ports tables.

The E-R diagram of Figure 23.19 expresses this relationship, where the degree of the Vessel entity relationships between the two tables is indicated by the "1" and "m" adjacent to their entities.

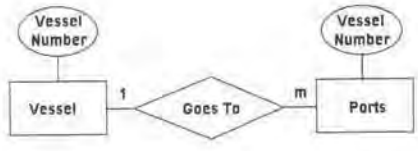


FIG. 23.19 The E-R diagram for the one-to-many relationship of Figure 23.17.

Many-to-One Relationships. Many-to-one relationships are the converse of the one-to-many type. The many-to-one relationship allows you to display the vessel name for any record in the Ports table. If the roles of the participating entities are simply reversed to create the many-to-one relationship, the relationship is said to be *reflexive*; that is, the many-to-one relationship is the reflection of its one-to-many counterpart (see Figure 23.20). All many-to-one relationships in Access are reflexive; you can specify only a one-to-one or one-to-many relationship between the primary table and the related table by using the two option buttons in Access's Relationship dialog.

Oracle Ex. 1009, p. 823
Oracle v. DeBellis
IPR2024-00221

Vessel	Voyage	Port	Type	Arrives	Departs
528	9203W	HNL	S	6/6/92	6/8/92
528	9204W	HNL	S	8/27/92	8/29/92
528	9203W	OSA	S	7/15/92	7/18/92
528	9204W	OSA	S	9/30/92	10/2/92
528	9203W	SFO	E		5/31/92
528	9204W	SFO	E		8/20/92

Vessel	Vessel Name
528	Japan Bear

FIG. 23.20 The Ports and Vessels tables in a reflexive many-to-one relationship.

If you select a record on the many side of the relationship, you can display the record corresponding to its foreign key on the one side. E-R diagrams for reflexive relationships are often drawn like the diagram in Figure 23.21. Reflexive relationships are indicated by the appropriate form of the verb placed outside the diamond that defines the relationship.

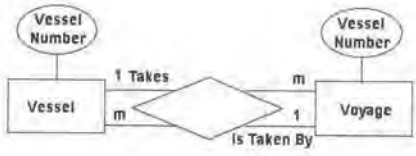


FIG. 23.21 The E-R diagram for a reflexive many-to-one relationship.

Many-to-Many Relationships and Fourth Normal Form. Many-to-many relationships cannot be expressed as simple relationships between two participating entities. You create many-to-many relationships by making a table that has many-to-one relationships with two base tables.

The Crews relation table, created in the "Third Normal Form" section of this chapter, for assigning crew members to legs of the voyage is shown again in Figure 23.22. The Crews table creates a many-to-many relationship between the Vessels table (based on the Vessel entity) and the Employees table (based on the employee number entities in the Master, Chief, and 1stMate fields).

Vessel	Voyage	Port	To	Master	Chief	1st Mate
528	9203W	SFO	HNL	01023	01156	01367
528	9203W	HNL	HNL	01023	01156	01367
528	9203W	HNL	OSA	01023	01156	01367
528	9203W	OSA	OSA	01023	01156	01367
528	9203W	OSA	INC	01023	01156	01367

FIG. 23.22 The first version of the Crews relation table.

The table in Figure 23.22 has a many-to-one relationship with the Vessels table and a many-to-one relationship with the Employees table. This version of the Crews table creates a many-to-many relationship between the Vessels and Employees tables. The employees who crew the vessel are independent of one another; any qualified employee can, in theory, be assigned to fill a crew position on any leg of a voyage. The table in Figure 23.22 violates the fourth normal form, therefore, because it contains independent entities.

Oracle Ex. 1009, p. 824
Oracle v. DeBellis
IPR2024-00221

Figure 23.23 shows the restructured Crews relation table that is needed to assign employees to legs of voyages. The table has one record for each employee for each leg of the voyage.

Employee	Vessel	Voyage	Port	To
01023	528	9203W	SFO	HNL
01156	528	9203W	SFO	HNL
01367	528	9203W	SFO	HNL
01023	528	9203W	HNL	HNL
01156	528	9203W	HNL	HNL
01367	528	9203W	HNL	HNL
01023	528	9203W	HNL	OSA
01156	528	9203W	HNL	OSA
01367	528	9203W	HNL	OSA
01023	528	9203W	OSA	OSA
01156	528	9203W	OSA	OSA
01367	528	9203W	OSA	OSA
01023	528	9203W	OSA	INC
01156	528	9203W	OSA	INC
01367	528	9203W	OSA	INC

FIG. 23.23 The Crews table restructured to fourth normal form.

You can add new entities to this table, provided that the entities are wholly dependent on all of the foreign key fields. An example of a dependent entity is payroll data that might include data attributes such as regular hours worked, overtime hours, and chargeable expenses incurred by each employee on each leg of a voyage. Such entities are called *weak* or *associative entities* because they rely on other base tables for their relevance. The Crews table is no longer considered strictly a relation table when you add associative entities because it no longer consists wholly of fields that constitute foreign keys.

The E-R diagram for the many-to-many relation table relating employees and the legs of a voyage to which the employees are assigned is shown in Figure 23.24. The encircled Date connected to the Assigned Crew relationship expresses cardinality: one employee can be assigned to only one voyage on a given date. The cardinality of the relationship, therefore, is based on the departure and arrival dates for the leg. Automatically enforcing the condition that employees not be in more than one place at one time can be accomplished by creating a no-duplicates index consisting of all fields of the Crews table. Associative entities are shown in E-R diagrams as a relationship diamond within an entity rectangle. If you add payroll data to the Crews table, an associative entity is created. Assignment of an employee to a voyage is optional, as indicated by the circled lines; employees may have shore leave, be indisposed, or be assigned to shoreside duties.

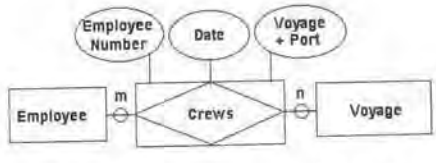


FIG. 23.24 An E-R diagram for a many-to-many relationship with an associative entity.

Using graphic schema and E-R diagrams when you design an Access database helps ensure that the database meets your initial objectives. Schema also are useful in explaining

Oracle Ex. 1009, p. 825
Oracle v. DeBellis
IPR2024-00221

the structure of your database to its users. E-R diagrams can uncover design errors, such as the failure to normalize tables at least to fourth normal form. Few experiences are more frustrating than having to restructure a large table because you realize its design wasn't fully normalized. Forethought, planning, and diagramming are the watchwords of success in database design.

Using Access 97's Table Analyzer Wizard

Access 97's Table Analyzer Wizard detects cells containing repeated data in table columns and proposes to create two new related tables to eliminate the repetition. This wizard uses Access 97's Lookup Wizard, described in Chapter 10, "Creating Multitable and Crosstab Queries," to create the relationship between the two new tables. After the wizard creates the new related tables, tbl*NewName* and tlkp*Lookup*, your original table is renamed to *TableName*_OLD, and the wizard creates a one-to-many `INNER JOIN` query named *TableName* to return a result set that duplicates Datasheet view of the original table. Thus, you need not change the references to *TableName* in your Access application objects.

◀◀ See "Using Lookup Fields in Tables," p. 332



The tlkp*Lookup* table must have a valid primary key field to provide unambiguous association of a single record in the tlkp*Lookup* table with a foreign key field in the tbl*NewName* table. One of the problems associated with repetitious data is data entry errors, such as occasional misspelling of a company name or an address element in tlkp*Lookup*. The Table Analyzer Wizard detects and displays instances of minor mismatch in repeated cell values, such as a missing apostrophe, for correction. If such errors are not corrected, the tlkp*Lookup* table includes spurious, almost-duplicate entries that violate the rules of table normalization.

To demonstrate use of the TableAnalyzer Wizard to eliminate duplicate shipping address information in the Orders table of Northwind.mdb, follow these steps:

1. Open Northwind.mdb, if necessary, and launch the TableAnalyzer Wizard by choosing Tools, Analyze, Table.
2. Skip the two introductory dialogs by clicking the Next button twice to reach the Table Selection dialog shown in Figure 23.25.



3. Select the table with the duplicated data in the Tables list box (for this example, you select the Orders table). Next, clear the Show Introductory Pages check box to skip the two introductory dialogs whenever you use the Table Analyzer Wizard again. Click the Next button to continue.



4. If you let the wizard decide, you can't change the primary key field of tlkp*Lookup*, so select the No, I Want To Decide option (see Figure 23.26). Click Next.

Oracle Ex. 1009, p. 826
Oracle v. DeBellis
IPR2024-00221

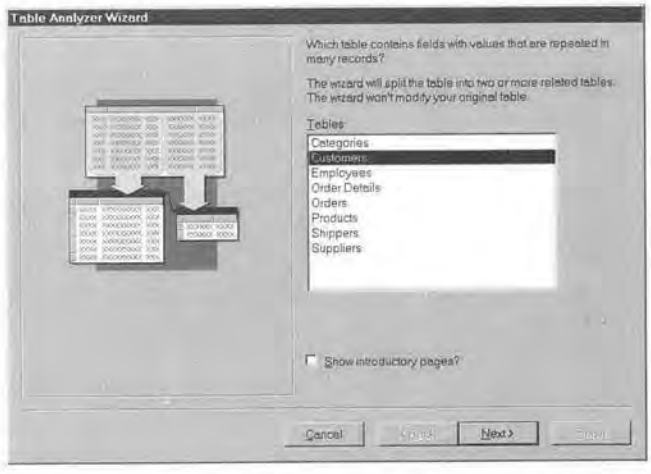


FIG. 23.25 Selecting the table to analyze in the third TableAnalyzer Wizard dialog.

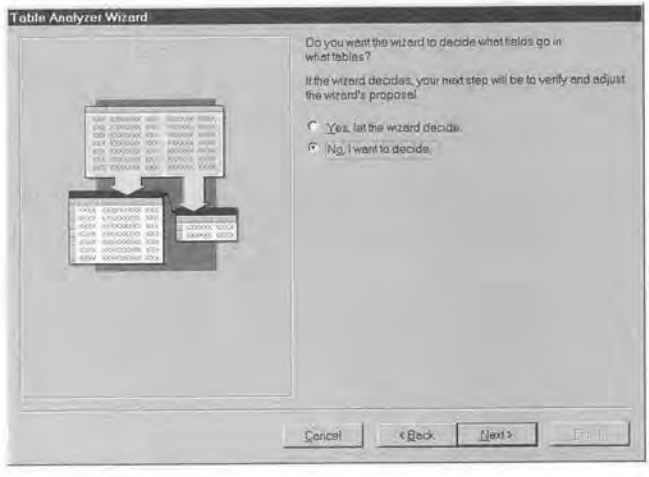


FIG. 23.26 Analyzing the Orders table by deciding yourself which table fields contain duplicate data.

5. The wizard displays a list of fields of the Orders table renamed to Table1. Click to select the first of the fields with duplicated information, ShipName, then press Shift and click the last of the fields to move, ShipCountry (see Figure 23.27).

6. Holding the left mouse button down, drag the selected fields from the field list to an empty area to the right of the Table1 list. When you release the mouse button, the wizard creates a new field list for proposed Table2 with a many-to-one relationship between Table1 and Table2. The relationship is based on a lookup field in Table1 and a Generated Unique ID (AutoNumber) field in Table2. An input box opens to rename the table; type **tlkpShipAddress** in the Table Name text box (see Figure 23.28). Click OK to close the input box.

Oracle Ex. 1009, p. 827
Oracle v. DeBellis
IPR2024-00221

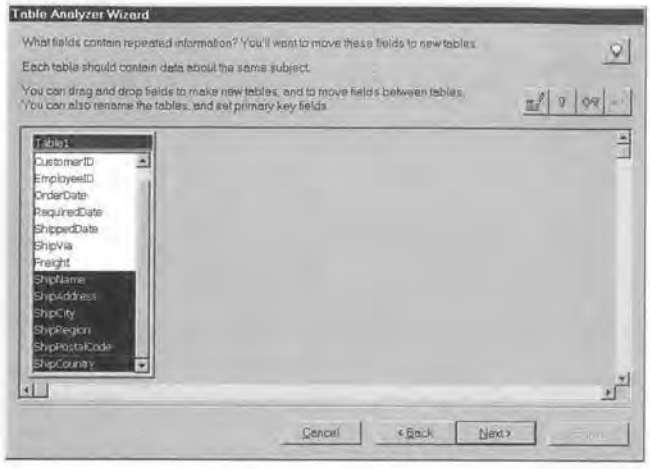


FIG. 23.27 Selecting the fields with duplicate data to move to a new table.

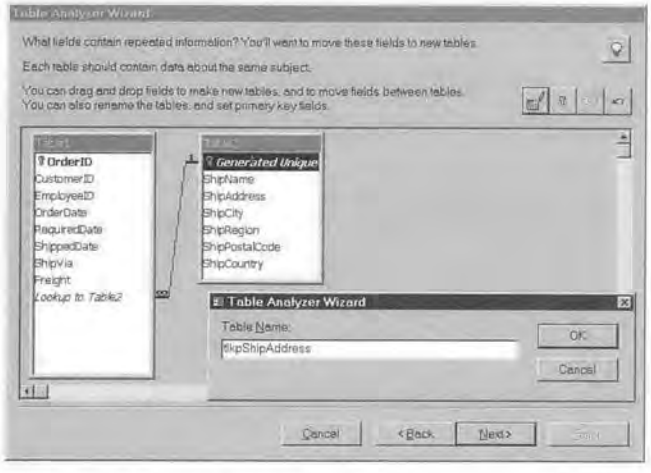


FIG 23.28 Naming the proposed tlkpShipAddress lookup table.

7. CustomerID is a better choice than an AutoNumber field for the primary key field for tlkpShipAddress, because there is only one ShipAddress per customer in the Orders table. Click and drag the CustomerID field from the Table1 field list to the tlkpShipAddress field list. With the CustomerID field selected in the tlkpShipAddress field list, click the Set Unique Key button (with the key icon only). The Generated Unique ID field disappears and the CustomerID field becomes the primary key for the proposed tlkpShipAddress table (see Figure 23.29).



8. Select Table1 and click the Rename Table button (the leftmost button in the group of three buttons near the top of the dialog) to display the Table Name input box. Give the table a new name—in this case, tblSalesOrders—and click OK to close the input box. Click Next to continue.



Oracle Ex. 1009, p. 828
Oracle v. DeBellis
IPR2024-00221

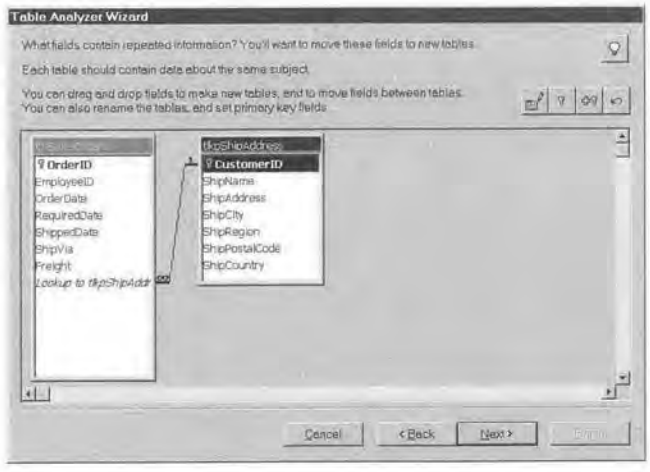


FIG. 23.29 Specifying CustomerID as the primary key field for the tlkpShipAddress table.

Next >

9. If the wizard detects a misspelling of an entry in the lookup table, the dialog shown in Figure 23.30 appears. The wizard bases the marked check box in the Correct Record column on the frequency of exact duplication of records ("Alfred's Futterkiste" appears several times, and "Alfreds Futterkiste" appears only once in the ShipName column). Verify which record is correct, and mark the Correct Record check box for the correct record(s), if necessary. The wizard automatically corrects the misspelled entries. Click Next to continue.

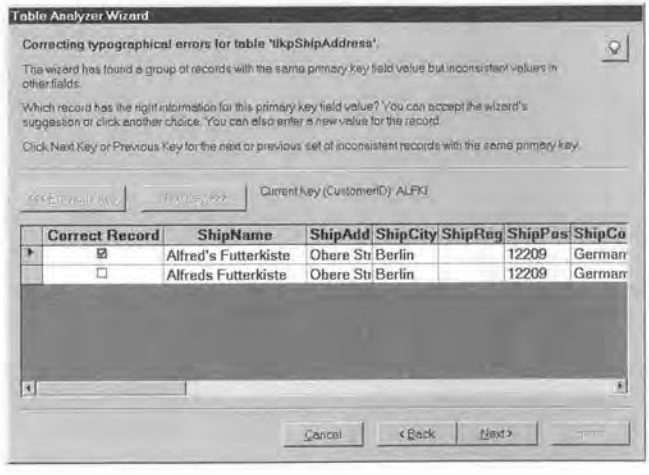


FIG. 23.30 Correcting a misspelled entry in the ShipName field.

10. The wizard proposes to create a query, in this case named Orders, that substitutes for the original Orders table. Accept the Yes, Create the Query option. Clear the

Oracle Ex. 1009, p. 829
Oracle v. DeBellis
IPR2024-00221

Show Me Help check box to prevent two Lookup Wizard Help screens from appearing when you complete the operation (see Figure 23.31).

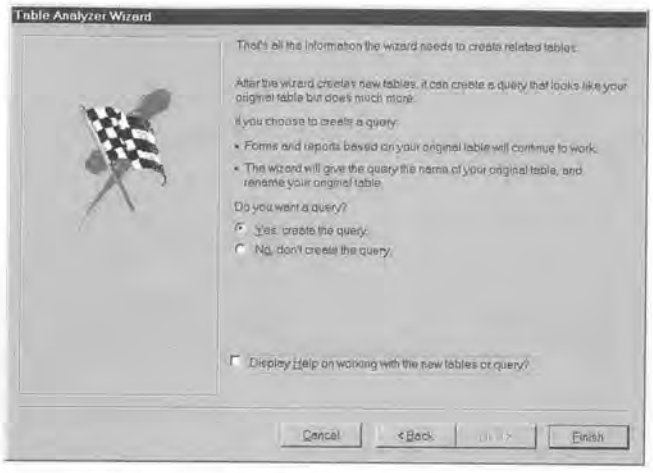


FIG. 23.31 Creating an Orders query as a substitute for the original Orders table.

11. Click Finish to create the Orders query and display the temporary Orders query datasheet as shown in Figure 23.32. Click OK in the message box to create the final Orders query.



FIG. 23.32 The temporary Orders query displaying the Lookup to tlkpShipAddress column.

12. Verify that the Datasheet view of the final Orders query result set, which does not include the Lookup to Ship Addresses column, duplicates the original orders table. Figure 23.32 shows the Orders query in Query Design view.

13. The wizard has renamed the original Orders table to Orders_OLD. To return Northwind.mdb to its original state, open the Database window and delete the Orders query plus the tblSalesOrders and tlkpShipAddress tables, then rename the Orders_OLD table to Orders.

Oracle Ex. 1009, p. 830
Oracle v. DeBellis
IPR2024-00221

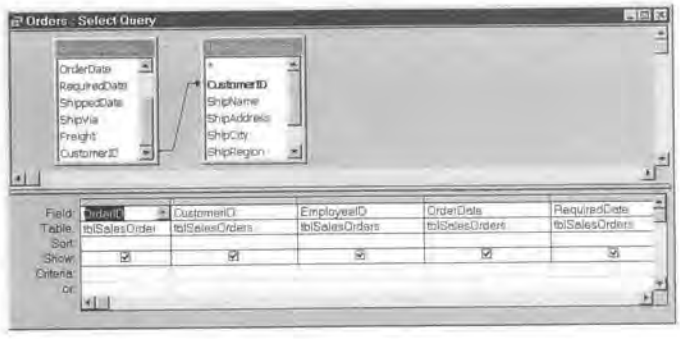


FIG. 23.33 The final Orders query in Query Design view.

Caution

Extracting the duplicate shipping address information from the Orders table to a new tlkpShipAddress table is useful to demonstrate use of the Table Analyzer Wizard. But this extraction is not practical in the real world, where individual customers may have several shipping addresses. To make the tlkpShipAddress table useful, you must add a field, such as ShipToID, to identify multiple shipping addresses for a single customer. Assign a value of 0 for the ShipToID field for the default shipping information created by the wizard. Additional shipping addresses for a particular CustomerID are numbered 1, 2, 3, ... You need to redesign forms that specify shipping addresses to allow adding new tlkpShipAddress records for customers. You must change the primary key to a composite primary key consisting of CustomerID + ShipToID, and you must use Access VBA code to create successive ShipToID values automatically for a particular CustomerID.

Working with Data Dictionaries

After you've determined the individual data entities that comprise the tables of your database and established the relationships between them, the next step is to prepare a preliminary written description of the database, called a *data dictionary*. Data dictionaries are indispensable to database systems; an undocumented database system is almost impossible to administer and maintain properly. Errors and omissions in database design often are uncovered when you prepare the preliminary data dictionary.

When you've completed and tested your database design, you prepare the final detailed version of the data dictionary. As you add new forms and reports to applications or modify existing forms and reports, you update the data dictionary to keep it current. Even if you're making a database for your personal use, a simplified version of a data dictionary pays many dividends on your time investment.

Conventional Data Dictionaries

Data dictionaries contain a text description of the database as a whole, each table the database contains, the fields that comprise the table, primary and foreign keys, and values that may be assigned to fields when they contain coded or enumerated information.

Oracle Ex. 1009, p. 831
Oracle v. DeBellis
IPR2024-00221

The purpose and description of each application that uses the database is included. Data dictionaries shouldn't be dependent on the particular RDBMS used to create and manipulate the database. Because data dictionaries are hierarchical in nature, they lend themselves to the use of traditional outline formats that are implemented in Windows word processing applications. The following illustrates the structure of a conventional data dictionary using legal-style outline headings:

1. DATABASE—Proper name and file name

   A text description of the purpose and general content of the database and who may use it. A list of applications that operate on the database is useful, along with references to any other databases that use the information that the database contains. If any graphic schema of the database have been prepared, they appear in this section.

   1.1. DATA AREA—Name of the group of which tables are a member

   When tables are classified by group, such as the Payroll group within the Human Resources database, include a description of the group.

   1.1.1. TABLE—Individual tables that comprise the data area

   1.1.1.1. PERMISSIONS—User domains with access to the table

   1.1.1.2. RECORD—General definition of the data entities

   1.1.1.2.1. PRIMARY KEY—Field(s) in the primary key

   1.1.1.2.1.1. INDEX—Primary key index specification

   1.1.1.2.2. FOREIGN KEY(S)—Other key fields

   1.1.1.2.2.1 INDEX—Indexes on foreign keys

   1.1.1.2.3. FIELDS—Non-key fields

   1.1.1.2.3.1 ENUMERATIONS—Valid codes for fields

Text follows each heading and describes the purpose of the database element to which the heading refers. Subsequent headings include descriptions of the applications that use the database tables, with subheadings for queries, forms, and reports. Captured images of displays and copies of reports add to the usefulness of the data dictionary. Printouts of programming code usually are contained in appendixes. Complete data dictionaries are essential for database maintenance. An alternative format consists of content descriptions of each table in tabular form.

Access's Integrated Data Dictionary System

Access 97's Database Documentor Add-In was introduced in Access 2.0. The Documentor creates a report that details the objects and values of the properties of the objects in the current database.

Oracle Ex. 1009, p. 832
Oracle v. DeBellis
IPR2024-00221



Note

The Access 2.0 version of Documentor printed reports, but couldn't export its data to tables or files. Access 97's Publish It with MS Word feature of the Office Links toolbar button allows you to save the report as an .rtf file (doc_rptObjects.rtf) for importing into Microsoft Word or any other word processor (such as WordPad) that handles rich-text files. Alternatively, you can export the report in BIFF format to an Excel doc_rptObjects.xls file by choosing the Analyze It with MS Excel option.

In many cases, Documentor tells you more than you want to know about your database; the full report for all objects in Northwind.mdb, for example, requires about 400 printed pages. Most often, you only want to document your tables and, perhaps, your queries to create a complete data dictionary. The following steps show you how to create a data dictionary with Database Documentor:

1. Open the database you want to document, and then choose Tools, Analyze, Documentor. Documentor's Select Objects opening tabbed dialog appears with Tables selected, as shown in Figure 23.34 for Nwind.mdb.

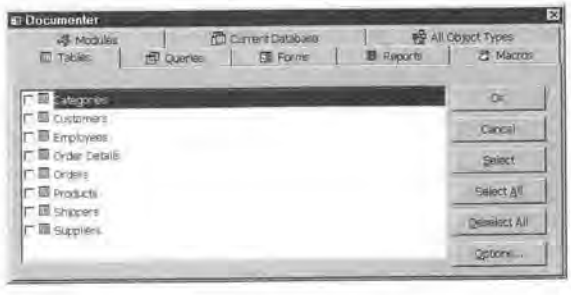


FIG. 23.34 Database Documentor's opening tabbed dialog.

2. Click the tab for the type of object you want to document to display the list of objects. The All Object Types tab adds every object in the database to the list when you click Select All.

3. With the Tables list selected, click Options to display the Print Table Definition dialog. The most detailed set of information for tables and indexes is the default. If your Access database is not secure, you can clear the Permissions by User and Group check box to eliminate reporting permissions data (see Figure 23.35). Click OK to return to the Documenter dialog.

4. Select the table(s) you want to document in the Tables list, and then click Select. This marks the table's check box, as shown in Figure 23.36. (Clicking the check box for an item has the same effect.) Alternatively, if you want to document all tables of your database, click Select All. For this example, only the Orders and Order Details tables are documented.

Oracle Ex. 1009, p. 833
Oracle v. DeBellis
IPR2024-00221

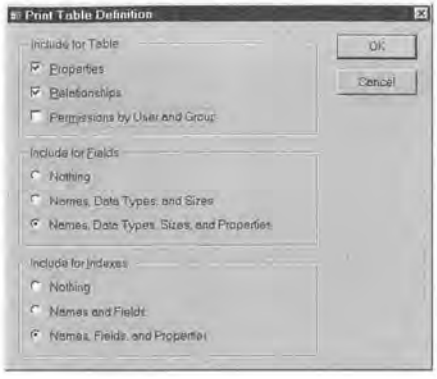


FIG. 23.35 Setting options for documenting tables.

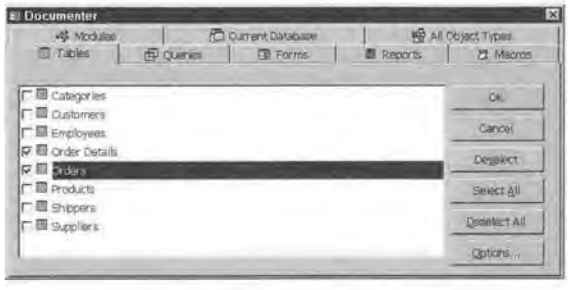


FIG. 23.36 Two tables selected for documenting.

5. Click the Queries tab to display the QueryDef objects in your database. Click the Options button to display the Print Query Definition dialog. Clear the Permissions by User and Group check box to eliminate security data from the report, and select the other options shown in Figure 23.37 to minimize duplication of table field data. Click OK to continue.
6. Click the Order Details Extended check box to add this query to the documentation, and then click OK to create the report, which is 13 pages long.
7. After a short period, the Object Definition print preview window appears (see Figure 23.38) with the Order Details table at the beginning of the list. Click the Last Page button to view the documentation for the Order Information query.



8. Click the Print button on the toolbar to print the report, or click the OfficeLinks button to create a doc_rptObjects.rtf or doc_rptObjects.xls file. Figure 23.39 shows part of a doc_rptObjects.xls file pertaining to the Order Details table.



Oracle Ex. 1009, p. 834
Oracle v. DeBellis
IPR2024-00221

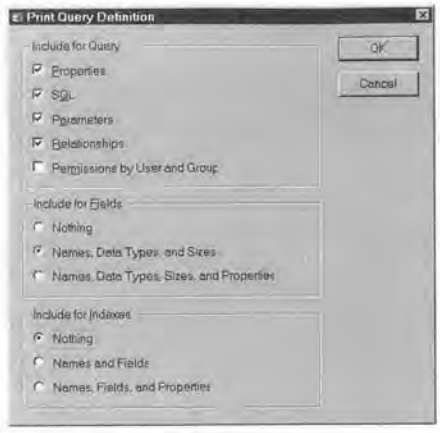


FIG. 23.37 Selecting the options for printing query definitions.

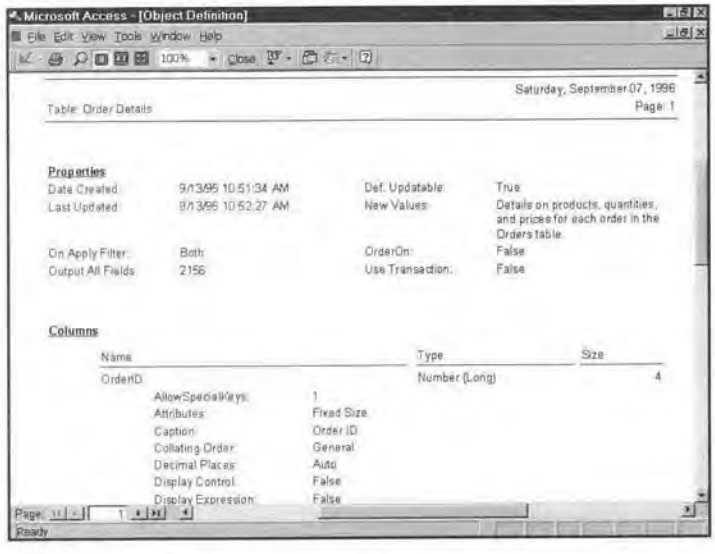


FIG. 23.38 The first page of Database Documentor's report.

9. Click the Close button on the toolbar, or double-click the Document Control-menu box to close the print preview window.

Documenting other objects in your database follows the same method outlined in the preceding steps. You can print data for the database itself by clicking the Current Database tab, or you can print data for selected forms, reports, macros, and modules.

Oracle Ex. 1009, p. 835
Oracle v. DeBellis
IPR2024-00221



FIG. 23.39 Part of the doc_rptObjects.xls displayed in Microsoft Excel 97.

Using Access Indexes

Database managers use indexes to relate the values of key fields to the location of the data entity on the disk. The basic purpose of an index is to speed up access to specific rows or groups of rows within a database table. You also use indexes to enforce the uniqueness of primary keys and establish upper and lower limits for queries. Using an index eliminates the necessity of re-sorting the table each time you need to create a sequenced list based on a foreign key.

Different PC database managers create and use indexes in a variety of ways. Paradox uses a mandatory primary index (.px) to speed up queries and ensure nonduplicate keys. Secondary indexes on nonprimary-key fields are permitted by Paradox (.x## and .y##) and created either by the QuerySpeedUp menu choice or the PAL INDEX instruction. dBASE and some of its xBase dialects allow any number of indexes to be created in the form of individual .ndx files, for a single file or table. The number of xBase indexes that you can have open at once, so as to keep them current, is determined by the xBase RDBMS you choose. In xBase and FoxPro, you select the index you want to use with the `SET ORDER TO` *`IndexFileName`* instruction. Several xBase languages have their own index structures, such as Clipper's .ntx and FoxPro's .idx. dBASE IV+ and FoxPro go one step beyond with their multiple index structures (.mdx and .cdx, respectively) that combine several indexes in a single file. You specify a `TAG` name to identify which index is to be used to find the records you want.

Oracle Ex. 1009, p. 836
Oracle v. DeBellis
IPR2024-00221

Indexed Sequential Access Method

Indexed Sequential Access Method (ISAM) describes a file structure in which the records are logically located (sorted) in their primary key's sequence of values, with an index used to provide random access to individual records. The term logically is applied to record location because the records' physical location on the disk may not be sequential; their physical location is determined by the disk's file allocation table (FAT, for Windows 95) and the degree of file fragmentation on the drive. ISAM is often used to describe any database structure that uses indexes for searching. This book adheres to the original definition, in which the records must be in the order of their primary key.

Classic mainframe ISAM databases have file structures that use *overflow sectors*, space reserved on the disk to handle insertion of new records. The database administrator periodically sorts the file to insert the data from the overflow sectors into the body of the table structure at appropriate locations. The periodic sorting clears the overflow sectors for future additions. The process is called *file maintenance*. Improved insertion techniques have been applied to databases created by PC and client/server RDBMSs. These methods are described in the sections that follow.

DOS RDBMSs duplicate ISAM structures by using an insertion technique. For instance, many xBase dialects allow you to `INSERT` a record in the middle of a file (the ISAM method), rather than `APPEND a BLANK` to the end of a file (the heap technique). When you `INSERT` a record near the top of a large, indexed xBase file, you can catch up on your sleep while the RDBMS moves all of the following records to make room for the new one, adjusting all index entries to refer to new locations. Figure 23.40 shows the difference between a record `INSERT` and `APPEND`. Paradox's native mode is ISAM, which explains why some of its operations, such as canceling an edit on a large table, take so long to complete.

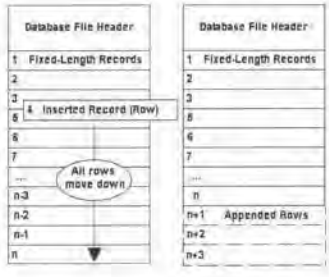


FIG. 23.40 Inserting versus appending new records in an xBase file.

Certain types of files created by DOS RDBMSs inherently fit the ISAM mold—sales order. Invoice records with a numeric key incremented by one for each addition are the best examples. Access's AutoNumber field data type performs the numbering function automatically. Other types of tables, such as lists of customers, are often sorted alphabetically. You have to INSERT each record in the proper location or re-sort the file each time you add a new customer to maintain a true ISAM structure. The faster method with xBase is to `APPEND BLANK`s and `REPLACE` the blanks `WITH` data; this procedure adds the new

Oracle Ex. 1009, p. 837
Oracle v. DeBellis
IPR2024-00221

records in a heap at the bottom of the file. Periodically, DOS adds another cluster to the file to accommodate the newly added data. Some xBase dialects, such as Clipper, don't include the `INSERT` command.

The header of an xBase .dbf file, such as the one in Figure 23.40, includes the name of each field, its field data type, its length in bytes, and some additional data. All data records in the file are the same length, representing the sum of the field lengths plus one byte, to indicate whether the record is marked for deletion. xBase files are called the *fixed-length record type*. Values that don't fill the length of a field are padded with spaces; character fields are padded with spaces to the right of the text, and numbers are right-justified by padding to the left. xBase files often incorporate much more padding than data.

The record numbers shown in Figure 23.40 aren't present in the data but are deduced by calculating the offset (the intervening number of bytes) of the record from the beginning of the file. If the header is 300 bytes long, record 1 begins at offset 300, corresponding to the 301st byte (the offset of the first byte is 0). Assuming the fields total 80 bytes in width, record 2 begins at offset 380, record 3 begins at offset 460, and so forth. The location of a data item's value is determined by calculating the offset of the desired record, then adding the offset to the beginning of the column (field) containing the data.

Access Data Pages and Variable Length Records

Access, like Microsoft SQL Server and many other SQL databases, divides the data stored in its table structures into 2K data pages, corresponding to the size of a conventional DOS fixed-disk file cluster. A header, similar to the one in Figure 23.41, is added to each page to create the foundation for a linked list of data pages. The header contains a reference, called a *pointer*, to the page that precedes it. The header contains another pointer to the page that follows. Linked lists use pointers to link the data pages to one another in order to organize the data table. If no indexes are in use, new data is added to the last page of the table until the page is full, and then another page is added at the end. The process is much like the heap method used by xBase or the manner in which DOS creates entries that link fixed-disk data clusters in its file allocation table.

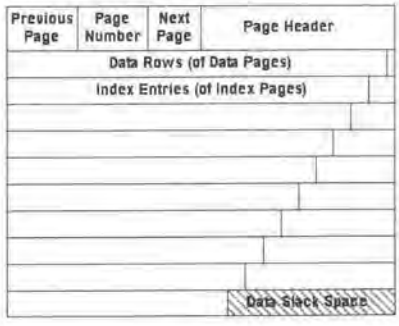


FIG. 23.41 The structure of a data page in an Access table.

Oracle Ex. 1009, p. 838
Oracle v. DeBellis
IPR2024-00221

Data pages contain only integral numbers of rows. The space that remains after the last row that fits in the page is called slack. You may be familiar with the concept of slack from the characteristic of current DOS versions that allocate fixed-disk file space in 2K clusters or larger. A small batch file, for example, may be displayed as having a file size of 120 bytes, but the file actually occupies a minimum of 2,048 bytes of disk space; the unused space is slack. Access uses variable-length records for its data rows instead of the fixed-length record structure of xBase. Variable-length records don't require padding for data that is shorter than the designated field size.

Data rows longer than 2K are contained in multiple pages. Avoid long rows if possible, as they can reduce storage efficiency by increasing the percentage of slack space in data pages. Access .mdb files with relatively short rows store data, especially character-based data, more efficiently than xBase or Paradox. Special fields containing text and images are stored in separate data structures linked to the data item in the data page. The storage concept is similar to that for xBase memo files, but the implementation differs in Access.

The advantage of data pages with their own headers (over the single-header, record-based xBase structure shown earlier in Figure 23.40) is that you can keep a table's data pages in ISAM order by altering only the pointers in the page header and not the structure of the file itself. This process, which uses a nonclustered index (discussed later in this chapter in the section "Nonclustered and Clustered Indexes"), is much faster than the `INSERT` method for xBase files and usually matches the speed of the `APPEND` technique.

Balanced Binary Trees, Roots, and Leaves

Most database managers use an indexing method called a *binary tree*, or *B-tree*. In describing an index structure, the tree is inverted, with its root and trunk at the top, progressing downward to branches and leaves—the direction taken by the searching process. A binary tree is defined as a tree in which the trunk divides into two branches, with each branch dividing into two sub-branches and then into further twofold divisions until you reach the leaves, which are indivisible. The points of the two-way divisions are called *nodes*. B-trees for computer-based searching were first proposed in 1946 by John Mauchly, one of the pioneers of electronic computers.

When you make many insertions and deletions in a database, conventional B-tree structures can become very lopsided, with many sub-branches and leaves stemming from one branch and few from another. The reason for this pattern is explained by mathematical theory that is beyond the scope of this book. Lopsided B-trees slow the searching process for records that are in an especially active area of the database. This situation causes undesirable effects in, for example, an airline reservation system where passenger reservations are being added to or deleted from a flight at a rapid rate immediately prior to its scheduled departure.

To solve the lopsided B-tree problem, two Russian mathematicians, G. M. Adelson-Velski and E. M. Landis, proposed a balanced B-tree structure in 1963. In a balanced B-tree structure, the length of the search path to any leaf is never more than 45 percent longer than the optimum. Each time a new row is added to the index, a new node is inserted

Oracle Ex. 1009, p. 839
Oracle v. DeBellis
IPR2024-00221

and the tree is rebalanced if necessary. A small, balanced B-tree structure appears in Figure 23.42. Its nodes are labeled + (plus) or – (minus), called the balance factor, according to whether the right subtree height minus the left subtree height is +1 or –1. If the subtrees are the same height, the node is empty. Balance factors are used to determine where a new node is added to maintain the tree in balance.

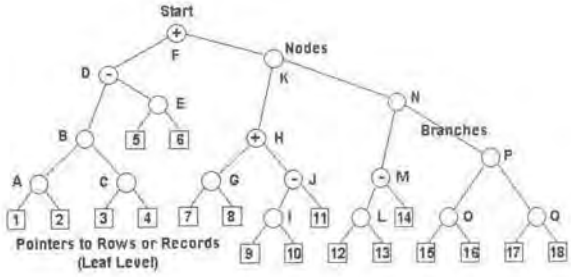


FIG. 23.42 A diagram of a simple balanced B-tree index.

Access and most other modern RDBMSs use the balanced B-tree structure to create indexes. Balanced B-trees improve search speed at the expense of increasing the time necessary to add new records, especially when the table is indexed on several keys. In a multitasking or client/server environment, however, the server conducts the addition process independently. The user can then perform other client operations, such as entering the key to search for another record, while the server's insertion and index rebalancing operations are going on.

Nonclustered and Clustered Indexes

Most RDBMSs, including Access, use nonclustered indexes to locate records with specific key values. Nonclustered means that the RDBMS adds data by the heap or APPEND method. Furthermore, nonclustered means that the rows of the table aren't in the sequence of their primary key—that is, the table isn't structured as an ISAM table. Figure 23.43 shows the structure, compressed and truncated, of a nonclustered Access index. xBase indexes have a similar structure, substituting records for data pages.

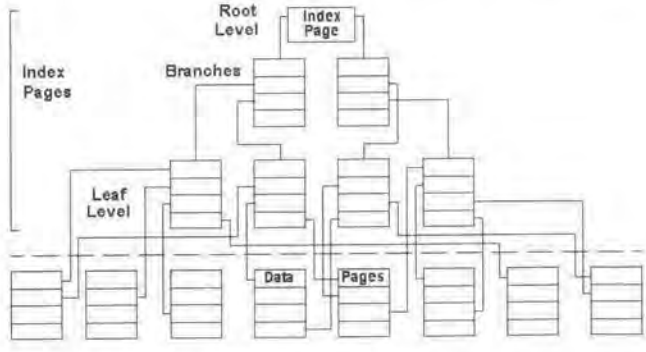


FIG. 23.43 A diagram of a conventional nonclustered index.

Oracle Ex. 1009, p. 840
Oracle v. DeBellis
IPR2024-00221

Notice the more-or-less random association of the data pages to the location of the pointers at the leaf level of the index. This lack of order is typical for indexes on foreign keys in all types of databases and for indexes on primary keys in non-ISAM files. If organized into an ISAM structure with the index created on the primary key, the file would have the more organized appearance of the diagram in Figure 23.44.

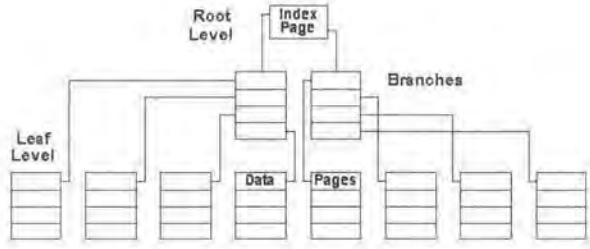


FIG. 23.44 A diagram of a clustered index.

Many client/server databases, such as Microsoft SQL Server, can use a clustered index to create ISAM order out of heap-induced chaos. When you use a clustered index, its table converts from heap-based row-insertion structure to the balanced B-tree structure you saw in Figure 23.42. In this case, the leaf level of the index consists of the data pages themselves. This organization is accomplished by rewriting the pointers in each data page's header in the order of the key on which the clustered index is based. Because the header can have only one set of pointers, fore and aft, you can have only one clustered index per table. In almost all cases, you create the clustered index from the primary key, but a clustered index need not be unique.

To retain the balanced B-tree organization of data pages, an RDBMS needs a balancing technique for insertions. Instead of using overflow sectors, the RDBMS adds a new page, readjusts the header pointers to include the new page in the linked list, and then moves the last half of the rows in the original page to the new page. The RDBMS then updates the index to reflect the changes. This process speeds up data access but slows updating; therefore, the process is practical only for an RDBMS running on a high-speed server computer under an advanced operating system, such as Windows NT 4.0, and that contains a lot of RAM.

Query Speed versus Update Performance

Access automatically creates an index on the primary key field. Adding other indexes to Access tables is a doubled-edged sword. You speed up the performance of queries because the index assists the sort sequence; you don't need to sort the query's `Recordset` object on a primary key index. Furthermore, sorts in the order of other indexes you specify are faster. On the other hand, when you append a new record, Access must take the time to update all of the table's indexes. When you edit a field of a record that is included in one or more indexes, Access has to update each of the indexes affected by the edit.

Beginning with version 2.0, Access incorporates FoxPro's Rushmore technology that optimizes query operations on indexed tables with large numbers of records. Many

Oracle Ex. 1009, p. 841
Oracle v. DeBellis
IPR2024-00221

decision-support queries can be speeded by a factor of 10 to 25 or more by adding the appropriate indexes, in addition to the index on the primary key field that is created for you. You can improve the performance of Access applications, especially when tables with large numbers of records or queries that join several tables are involved, by observing these guidelines:

- Minimize the number of indexes used with transaction-based tables, especially in networked multiuser applications that share tables. Access locks pages so that they aren't editable by other users while you are editing records and during the time it takes Access to update the indexes when you are finished.
- Minimize the number of indexes in tables that are used regularly with `INSERT` and `DELETE` queries. The time required to update indexes is especially evident when making changes to the data in bulk.
- Add indexes judiciously to tables that have large numbers of records and are used primarily in decision-support applications.
- Add indexes to foreign key fields of tables that participate in joins with primary tables. However, when specifying a selection criterion on the key field, always use the key field of the primary table rather than the foreign key field of the related table.
- Add indexes to fields on which you set criteria. If transaction-processing performance is more important than the speed of decision-support applications, add indexes only to those fields that occur most often in the criteria of your select queries.

Indexing becomes more important as the number of records in your tables increases. You might need to experiment to determine whether an index is effective in significantly increasing the speed of a query. If you find the index is warranted by improved query performance, check the speed of transaction processes with the new index before committing to its use.

Note

If you are using a shared database on a peer-to-peer network, such as Windows 95 networking, and the database is located on your computer, use another workgroup member's computer to test the effectiveness of indexes. Network characteristics may significantly affect the performance of indexes. Try to make the test during periods of maximum network traffic, not during off hours, when no one is contributing to network congestion.

Enforcing Database Integrity

The integrity of a database is comprised of two elements: entity and referential integrity. Entity integrity requires that all primary keys must be unique within a table, and referential integrity dictates that all foreign keys must have corresponding values within a base table's primary key. Although the normalization process creates entity and referential

Oracle Ex. 1009, p. 842
Oracle v. DeBellis
IPR2024-00221

integrity, either the RDBMS itself or your application must maintain that integrity during the data-entry process. Failure to maintain database integrity can result in erroneous data values and, ultimately, widespread corruption of the entire database.

Ensuring Entity Integrity and Auditability

Database managers differ widely in their capabilities to maintain entity integrity through unique primary key values. Paradox, for instance, enforces unique primary keys within the RDBMS by flagging as a key violation any attempt to insert a row with an identical primary key. Paradox then places the offending record in the KeyViol table. Access uses a similar technique when you specify a no-duplicates index. If you paste or append records that have duplicate primary keys, Access appends those records to a Paste Errors or Append Errors table.

> **Note**
>
> In xBase, you can add as many records with duplicate index keys as you want. If you use SET UNIQUE ON, a SEEK finds only the first record with the same key. However, any duplicate keys remain in the file and, for example, appear in an indexed LIST operation. Indexed DELETEs affect only the first undeleted record found for the SEEK parameter; you must perform a DELETE for each duplicate. You need to write xBase code, therefore, to test for data duplicates before you APPEND the record that adds the data to the file.

Enforcing entity integrity within the table itself, the process used by Access, is more reliable than using application programming code to prevent the duplication of primary key values. Access provides two methods of ensuring entity integrity that are independent of the applications employing the tables:

- A key field that uses the AutoNumber data type that creates unique values based on an automatically incremented long integer. Using incremental AutoNumber fields is the most common method for creating primary key fields. You cannot create a duplicate primary key in this case because you cannot edit the values in fields of the AutoNumber data type. Random AutoNumber fields are not suitable for creating primary key values, because there is no assurance that a randomly-assigned number is not duplicated in tables with very large numbers of records.
- An index on the primary key field with the No Duplicates property. If you attempt to enter a duplicate value in the key field, Access displays an error message.

Either of these methods ensures unique key fields, but an AutoNumber field is helpful so that documents such as sales orders, invoices, and checks are sequentially numbered. Sequential numbering is necessary for internal control and auditing purposes. AutoNumber (Incremental) fields normally begin with 1 as the first record in a table, but rarely does a real-world cash disbursement or invoice table need 1 as its starting number. You cannot create a table with a Long Integer Number field, enter the beginning number, and then change the field data type to AutoNumber. Access issues a warning message if you attempt this procedure. You can use an append query, however, to establish a specific beginning AutoNumber value.

Oracle Ex. 1009, p. 843
Oracle v. DeBellis
IPR2024-00221

Note

Access increments the value of the AutoNumber field even if Access prevents you from adding a record to a table with an AutoNumber field. Such a situation can occur if domain or referential integrity rules are violated when the addition of a record is attempted. For this reason, an Access AutoNumber field is not likely to be satisfactory for applications where sequential documents must be accounted for.

To create a starting AutoNumber field value of 123456 in the Invoice field of the tblInvoices table's first record, perform the following steps:

1. Open the Database window, select the Orders table, and press Ctrl+C to copy the table to the Clipboard.
2. Press Ctrl+V and type **tblInvoices** as the name of the table to create. Then select the Structure Only option button to create the new tblInvoices table with no records.
3. Open the tblInvoices table in Design mode, click the select button of the Order ID field, and change the field data type from AutoNumber to Number (Long Integer).



4. Click the Datasheet View button on the toolbar, and click OK to save your changes. Alternatively, choose File, Save. (Access 97 does not allow two or more operations on AutoNumber fields in a single design operation.)



5. If you changed to Datasheet view, Reopen the table in Design mode, select the OrderID field, and press Insert to add a new field.



6. Enter InvoiceID as the Field Name, and choose AutoNumber (Increment) as the Data Type.
7. Click the Indexes button on the toolbar. Delete the Primary Key index on OrderID, which enables you to append a record that doesn't have a value for the key field (Null values aren't allowed in key fields). The tblInvoices table design appears as shown in Figure 23.45. Close the Indexes window.



   You also need to set the value of the Required property of the CustomerID field to No and the Allow Zero Length property to Yes. Otherwise, the append query that follows will not execute.
8. Close the tblInvoices table and save your changes. Don't create a primary key field at this point.
9. Create a new temporary table named tblFirstInvoice with one field, named InvoiceID.



10. Set the InvoiceID field's Data Type property to Number and the Field Size property to Long Integer.
11. Change to Datasheet view; don't create a primary key field when asked; and enter a value in InvoiceID that is one less than the starting number you want (for this example, type **123455**).

Oracle Ex. 1009, p. 844
Oracle v. DeBellis
IPR2024-00221



FIG. 23.45 A table designed for adding an AutoNumber field with an arbitrary starting number.

12. Close the tblFirstInvoice table and save your changes.

New **13.** Create a new query and add the tblFirstInvoice table. Click and drag the InvoiceID field symbol to the first column of the Query Design grid.

14. Choose Query, Append and type **tblInvoices** as the table to which to append the record. Click OK. Invoice automatically appears as the Append To field (see Figure 23.46). Click the Run button on the toolbar to add the single record in tblFirstInvoice to the tblInvoices table.

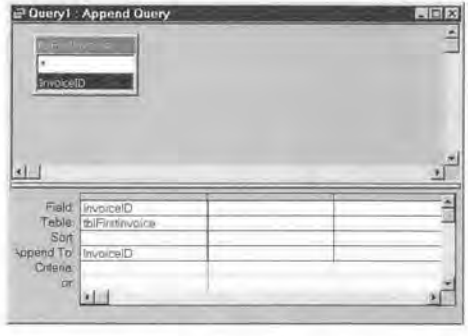


FIG. 23.46 The append query to set the first value of an AutoNumber field.

15. Close Query1 without saving your changes.

The next record you append to tblInvoices is assigned the value 123456 in the InvoiceID (AutoNumber) field. To verify that this technique works properly for appended records, perform the following steps:

New **1.** Create a new query, and add the Orders table. Click and drag the * symbol to the first column of the Query Design grid.

Oracle Ex. 1009, p. 845
Oracle v. DeBellis
IPR2024-00221

2. Choose Query, Append and enter tblInvoiceData as the table to which to append the records from the Orders table (see Figure 23.47). Click the Run Query button on the toolbar to add the records from the Orders table.



3. Close Query1 and don't save the changes.
4. Open the tblInvoices table.



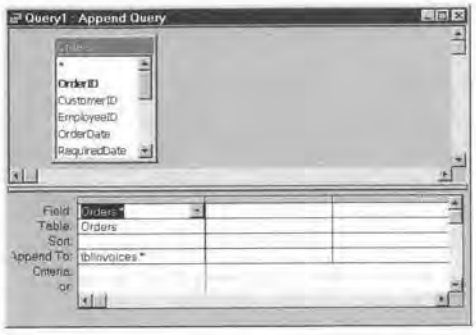


FIG. 23.47 The append query to add records from the Orders table to the tblInvoices table.

Access has added numbers beginning with 123456 to the new Invoice field, corresponding to OrderID values of 10000 and higher, as shown in Figure 23.48.

If you were adding an AutoNumber field to real-world data, you would delete the first blank record and then make the InvoiceID field the primary key field. Access creates a no-duplicates index when you assign an AutoNumber field as a primary key field.

tblInvoices : Table

InvoiceID	Order ID	Customer	Employee
123455	0		
123456	10248	Vins et alcools Chevalier	Buchanan, Steven
123457	10249	Toms Spezialitäten	Suyama, Michael
123458	10250	Hanari Carnes	Peacock, Margaret
123459	10251	Victuailles en stock	Leverling, Janet
123460	10252	Suprêmes délices	Peacock, Margaret
123461	10253	Hanari Carnes	Leverling, Janet
123462	10254	Chop-suey Chinese	Buchanan, Steven
123463	10255	Richter Supermarkt	Dodsworth, Anne
123464	10256	Wellington Importadora	Leverling, Janet
123465	10257	HILARIÓN-Abastos	Peacock, Margaret
123466	10258	Ernst Handel	Davolio, Nancy
123467	10259	Centro comercial Moctezuma	Peacock, Margaret
123468	10260	Ottilies Käseladen	Peacock, Margaret
123469	10261	Que Delicia	Peacock, Margaret
123470	10262	Rattlesnake Canyon Grocery	Callahan, Laura
123471	10263	Ernst Handel	Dodsworth, Anne
123472	10264	Folk och fä HB	Suyama, Michael

Record: 1 of 832

FIG. 23.48 The Invoice Data table including AutoNumber values starting with 123456.

Oracle Ex. 1009, p. 846
Oracle v. DeBellis
IPR2024-00221

Troubleshooting

When I try to execute the append query that provides the starting value minus 1 to my table structure, I get an error message.

You have constraints on the fields of your destination table that do not allow Null values or empty strings in fields. Open your destination table in Design mode, and make sure that the Required property of each field is set to No and that the Allow Zero Length property of each field of the Text data type is set to Yes. Also, make sure that your destination table does not specify a primary key field.

Maintaining Referential Integrity

Prior chapters have discussed the use of Access's database-level referential integrity enforcement capabilities, which were introduced in Access 2.0. Maintaining referential integrity requires strict adherence to a single rule: *Each foreign key field in a related table must correspond to a primary key field in a base or primary table.* This rule requires that the following types of transactions be prevented:

- Adding a record on the many side of a one-to-many relationship without the existence of a corresponding record on the one side of the relationship.
- Deleting a record on the one side of a one-to-many relationship without first deleting all corresponding records on the many side of the relationship.
- Deleting or adding a record to a table in a one-to-one relationship with another table without deleting or adding a corresponding record in the related table.
- Changing the value of a primary key field of a base table on which records in a relation table depend.
- Changing the value of a foreign key field in a relation table to a value that doesn't exist in the primary key field of a base table.

A record in a relation table that has a foreign key with a value that doesn't correspond to the value of a primary key in a relation table is called an *orphan record*.

Whenever possible, use Access's built-in `INNER JOIN`, cascading updates, and cascading deletions features to maintain referential integrity at the database level. Don't rely on applications to test for referential integrity violations when adding records to relation tables or deleting records from base tables. Access gives you the opportunity to enforce referential integrity automatically between tables in a database by marking the Enforce Referential Integrity check box in the Relationships dialog. As noted in Chapter 11, "Using Action Queries," you can specify cascade updates and deletes when you use Access's referential integrity enforcement capabilities. Access 97 also enforces referential integrity in linked Access tables.

Note

Most xBase RDBMSs don't have the capability to enforce referential integrity automatically. You need to write xBase code that tests for the required records with SEEK commands on indexed files.

Oracle Ex. 1009, p. 847
Oracle v. DeBellis
IPR2024-00221

Chapter 24

Working with Structured Query Language

This chapter describes Structured Query Language (SQL), the structure and syntax of the language, and how Access translates queries you design with Access's graphical query-by-example (QBE) technique into SQL statements. An SQL background helps you understand the query process and design more efficient queries. A knowledge of SQL syntax is necessary to use the new subquery and `UNION` query capabilities introduced by Access 2.0 and for many of the applications you write in Access VBA. Examples of SQL have been presented in other chapters in this book. These examples—usually figures that illustrate an SQL statement created by Access—demonstrate what occurs behind the scenes when you create a query or a graph.

In this chapter, you learn more about:

- SQL Reserved Words
- SQL Query Syntax
- Basic SQL `SELECT` Queries
- SQL `INNER JOIN`s, `OUTER JOIN`s, and `UNION` Queries
- SQL `INSERT`, `UPDATE`, and `DELETE` Queries

Note

There are no significant differences between the Access 2.0, Access 95, and Access 97 versions of SQL, previously known as *Access SQL*. The new name for Access SQL is *Microsoft Jet Database Engine SQL*, called *Jet SQL* in this book for brevity. The primary additions to Access SQL occurred during the upgrade from Access 1.1 to Access 2.0. The new Jet SQL terminology is more appropriate than Access SQL, because all members of Microsoft Office, Visual Basic, and any other application that supports VBA can use the Jet database engine.

Using Access to Learn SQL

Access is a useful learning tool for gaining fluency in SQL. This chapter shows you how to create Access QBE queries from SQL statements entered in the SQL dialog. If you use SQL with another RDBMS, such as Microsoft SQL Server, this chapter can help you make the transition from ANSI/Access/Jet SQL to Transact-SQL, the extended version of SQL used by Microsoft SQL Server. A knowledge of SQL also is necessary for Visual Basic programmers because Visual Basic doesn't include a graphical QBE feature.

Oracle Ex. 1009, p. 848
Oracle v. DeBellis
IPR2024-00221

Many users of Access decision-support applications want to be able to define their own queries. When you open an Access 97 database with the run-time version of Access, the Query Design window is hidden. Thus, you must design forms that include control objects that users can manipulate to construct a Jet SQL statement that returns the desired query result set. You write Access VBA code to translate users' choices on the form into a Jet SQL statement; then create a `QueryDef` object (a persistent query definition whose name appears in the Database window) in the current database.

Note

This book uses the term *Access VBA* to mean the Access "flavor" of VBA that includes preestablished references to Microsoft Access 97 object collections, such as `Forms` and `Reports`, and elements of the Jet 3.5 Data Access Object (DAO), as well as VBA itself. The "flavor" of other VBA-enabled applications—such as Excel VBA, Word VBA, or Project VBA—is determined by the application's object collections that are referenced by default.

What Is Structured Query Language?

Structured Query Language, abbreviated SQL (usually pronounced "sequel" or "seekel," but more properly "ess-cue-ell"), is the common language of client/server database management. The principal advantage of SQL is that it's standardized—you can use a common set of SQL statements with all SQL-compliant database management systems. The first U.S. SQL standard was established in 1986 as ANSI X3.135-1986. The current version is ANSI X3.135-1992, usually known as SQL-92.

Note

ANSI, an acronym for the *American National Standards Institute*, is an organization devoted to establishing and maintaining scientific and engineering standards. ANSI-standard SQL was first adopted as a worldwide standard in 1987 by the International Standards Organization (ISO), a branch of the United Nations. Microsoft states that Jet SQL "generally is ANSI SQL-89 Level 1-compliant."

SQL is an application language for relational databases, not a system or programming language. SQL is a set-oriented language; thus, ANSI SQL includes neither a provision for program flow control (branching and looping), nor keywords to create data entry forms and print reports. Programming functions usually are implemented in a system language such as xBase, C, C++, or COBOL. Some implementations of SQL, such as Transact-SQL used by Microsoft SQL Server, add flow control statements (`IF...ELSE` and `WHILE`) to the language. Publishers of ANSI SQL-compliant RDBMSes are free to extend the language if the basic ANSI commands are supported. The ANSI/ISO implementation of SQL is independent of any system language with which it might be used.

ANSI SQL includes a set of standard commands (keyword verbs) that are broadly grouped into six categories:

Oracle Ex. 1009, p. 849
Oracle v. DeBellis
IPR2024-00221

- Data definition
- Data query
- Data manipulation
- Cursor control
- Transaction processing
- Administration or control

Provisions for SQL keywords that maintain data integrity were added in a 1989 revision of the original standard as ANSI X3.135-1989, *Database Language—SQL with Integrity Enhancement*. Jet 3.5's implementation of SQL includes the data integrity keywords.

SQL has three different methods of implementation:

- *Direct Invocation*. Sends a series of SQL statements to the RDBMS. The RDBMS responds to the query by creating a table that contains the result and displays the table. Entering SQL commands at the dBASE IV SQL prompt is an example of Direct Invocation.
- *Module Language*. Lets you write a text file of SQL statements that later are executed by an application; you begin the SQL statement with the `MODULE` keyword and specify the programming `LANGUAGE`.
- *Embedded SQL*. The most common implementation; the SQL statements are generated by the application or included as strings of text in a command of an application language. Access queries, whether created by graphical QBE, by an SQL property in Access VBA, or by the Row Source property of a graph, use Embedded SQL.

Looking at the Development of SQL

SQL was created because, early in the 1970s, IBM wanted a method with which nonprogrammers could extract and display the information they wanted from a database. Languages that nonprogrammers can use are called *fourth generation*, or *4GL*, and sometimes are referred to as *structured English*. The first commercial result of this effort was Query By Example (QBE), developed at IBM's laboratories in Yorktown Heights, New York. QBE was used, beginning in the late 1970s, on terminals connected to IBM System 370 mainframes. A user could obtain a result with less than an 80-character line of QBE code that required 100 or more lines to implement in COBOL or the other 3GL languages of the day. Access, dBASE IV and 5, and Paradox use QBE to display selected data from tables.

At the other end of the country, programmers at IBM's San Jose, California facility were developing System R, the progenitor of SQL/DS and IBM's DB2 relational database. In the mid-1970s, IBM scientist Dr. E.F. Codd proposed SQL (then known as SEQUEL for *S*tructured *E*nglish *Qu*ery *L*anguage) as a means of accessing information from the relational database model he had developed in 1970. Relational databases based on the Codd model that use the SQL language to retrieve and update data within them have become, like QBE, computer-industry standards.

Oracle Ex. 1009, p. 850
Oracle v. DeBellis
IPR2024-00221

SQL has achieved the status of being the exclusive language of relational client/server databases. A *database server application* (the back end) holds the data. *Client applications* (front ends) add to or edit the data. The client application generates SQL statements. If you deal regularly with databases of any type, the odds are great that you ultimately will need to learn SQL. You need to learn Jet SQL *now* if you plan to create applications with user-defined queries that work with the run-time version of Access. You also need to learn SQL if you want to take full advantage of the Internet Database Connector (IDC/HTX files) described in Chapter 18, "Creating Dynamic Web Pages."

Comparing ANSI and Jet SQL

Jet SQL doesn't include many of the approximately 100 keywords incorporated in the ANSI standard for SQL. Few, if any, commercial SQL-compliant RDBMSes for the PC implement much more than half of the standard SQL keywords. The majority of the common SQL keywords missing from Jet's implementation are provided by the expressions you create with operators, built-in functions, or user-defined functions you write in Access VBA. The effect of many unsupported ANSI SQL keywords related to tables is achieved by making selections from Access's Database window or from menus.

When you learn a new language, it's helpful to categorize the vocabulary of the language by usage and then into the familiar parts of speech. SQL commands, therefore, first are divided into six usage categories:

- *Data Query Language (DQL).* Sometimes referred to as *data retrieval commands*, obtain data from tables and determine how the results of the retrieval are presented. The `SELECT` command is the principal instruction in this category. DQL commands often are considered members of the Data Manipulation Language category.
- *Data Manipulation Language (DML).* Provide `INSERT` and `DELETE` commands, which add or delete entire rows, and the `UPDATE` command, which can change the values of data in specified columns within rows.
- *Transaction Processing Language (TPL).* Include `BEGIN TRANS[ACTION]`, `COMMIT [WORK]`, and `ROLLBACK [WORK]`, which group multiple DML operations. If one DML operation of a transaction fails, the preceding DML operations are canceled (rolled back). Jet SQL doesn't support TPL; instead, you use the Access VBA `BeginTrans`, `CommitTrans`, and `Rollback` methods to support transaction processing.
- *Data Definition Language (DDL).* Include `CREATE TABLE` and `CREATE VIEW` instructions that define the structure of tables and views. DDL commands are used also to modify tables and to create and delete indexes. The keywords that implement data integrity are used in conjunction with DDL statements. Jet SQL supports the `CREATE TABLE` and `CREATE INDEX` instructions.

Oracle Ex. 1009, p. 851
Oracle v. DeBellis
IPR2024-00221

- *Cursor Control Language (CCL).* Can select a single row of a query result set for processing. Cursor control constructs, such as `UPDATE WHERE CURRENT`, are implemented by the Jet database engine, so these commands are not discussed in this chapter.
- *Data Control Language (DCL).* Perform administrative functions that grant and revoke privileges to use the database, such as `GRANT` and `REVOKE`, a set of tables within the database, or specific SQL commands. Jet SQL does not include DCL; instead, you use Access's security objects for implementing security.

Keywords that comprise the vocabulary of SQL are identified further in the following categories:

- *Commands*, such as `SELECT`, are verbs that cause an action to be performed.
- *Qualifiers*, such as `WHERE`, limit the range of values of the entities that comprise the query.
- *Clauses*, such as `ORDER BY`, modify the action of an instruction.
- *Operators*, such as =, <, or >, compare values and are used to create joins when `JOIN` syntax is not used. Jet SQL uses `JOIN` syntax by default, but lets you use operators to create joins.
- *Group aggregate functions*, such as `MIN()`, return a single result for a set of values.
- Other keywords modify the action of a clause or manipulate cursors that are used to select specific rows of queries.

Note

SQL keywords usually are capitalized, but the keywords aren't case-sensitive. The uppercase convention is used in this book, and SQL keywords are set in the monospace type. You use *`parameters`*, such as *`column_list`*, to define or modify the action specified by keywords. Names of replaceable parameters are printed in lowercase italicized monospace type.

SQL Reserved Words in Access

Access doesn't support all the ANSI SQL keywords with identical reserved words in the Jet SQL language. In this chapter, *keywords* are defined as the commands and functions that comprise the ANSI SQL language. Jet SQL commands and functions, however, are referred to here as *reserved words* to distinguish them from ANSI SQL.

The tables in the following section are intended to acquaint readers who are familiar with ANSI or similar implementations of SQL in other DBMs or database front-end applications with the Access implementation of SQL. If you haven't used SQL, the tables demonstrate that SQL is a relatively sparse language, having far fewer keywords than programming languages like VBA, and that Jet SQL is even more sparse. Jet SQL has few reserved words to learn. You learned in Chapter 9, "Understanding Operators and Expressions in Access," to use the Access operators and functions in expressions that Access substitutes for ANSI SQL keywords.

Oracle Ex. 1009, p. 852
Oracle v. DeBellis
IPR2024-00221

Jet SQL Reserved Words Corresponding to ANSI SQL Keywords

Access supports the ANSI SQL keywords listed in Table 24.1 as identical reserved words in Jet SQL. Don't use these Jet SQL reserved words as the names of tables, fields, or variables. The reserved words in Table 24.1 appear in all capital letters in the Jet SQL statements Access creates for you when you design a query or when you add a graph to a form or report. Reserved words in are marked with an asterisk were introduced by Access 95:

ADD*	CONSTRAINT*	HAVING	MAX	REFERENCES*
ALL	COUNT	IN	MIN	RIGHT
ALTER*	CREATE*	INDEX*	NOT	SELECT
ANY*	DELETE	INNER	NULL	SET
AS	DESC	INSERT	ON	SOME*
ALIAS	DISALLOW*	INTO	OR	UNION*
ASC	DISTINCT	IS	ORDER	UNIQUE*
AVG	DROP*	JOINR	OUTE	UPDATE
BETWEEN	EXISTS*	KEY*	PARAMETERS	VALUE*
BY	FOREIGN*	LEFT	PRIMARY*	VALUES*
COLUMN*	FROM	LIKE	PROCEDURE	WHERE

The keywords that relate to data types, `CHAR[ACTER]`, `FLOAT`, `INT[EGER]`, and `REAL`, aren't included in Table 24.1 because Jet SQL uses a different reserved word to specify these SQL data types. The comparison operators (=, <, <=, >, and =>) are common to both ANSI SQL and Jet SQL. Access substitutes the <> operator for ANSI SQL's not-equal (!=) operator.

As in ANSI SQL, the `IN` reserved word in Jet SQL can be used as an operator to specify a list of values to match in a `WHERE` clause or a list created by a subquery. (Beginning with Access 2.0, Jet SQL supports subqueries.) You also can use `IN` to identify a table in another database; this use is discussed near the end of this chapter in the section "Adding `IN` to Use Tables in Another Database."

Access Functions and Operators Used in Place of ANSI SQL Keywords

Table 24.1 shows reserved words in Jet SQL that correspond to ANSI SQL keywords but are operators or functions used in Jet SQL expressions. Jet doesn't use ANSI SQL syntax for its aggregate functions; you cannot use the `SUM(DISTINCT field_name)` syntax of ANSI SQL, for instance. Jet, therefore, distinguishes between its use of the `Sum()` aggregate function and the SQL implementation, `SUM()`. Expressions that use operators such as `And` and `Or` are enclosed in parentheses in Jet SQL statements; Jet SQL uses uppercase `AND` and `OR` (refer to the previous section) when criteria are added to more than one column.

Oracle Ex. 1009, p. 853
Oracle v. DeBellis
IPR2024-00221

Table 24.1 Jet SQL Reserved Words that Substitute for ANSI SQL Keywords

Jet SQL	ANSI SQL	Jet SQL	ANSI SQL
And	AND	Max()	MAX()
Avg()	AVG()	Min()	MIN()
Between	BETWEEN	Not	NOT
Count()	COUNT()	Null	NULL
Is	IS	Or	OR
Like	LIKE	Sum()	SUM()

The Jet IsNull() function that returns True (-1) or False (0), depending on whether IsNull()'s argument has a Null value, has no equivalent in ANSI SQL, and isn't a substitute for IS NULL or IS NOT NULL qualifiers in WHERE clauses. Jet SQL does not support distinct aggregate function references, such as AVG(DISTINCT *field_name*); the default DISTINCTROW qualifier added to the SELECT statement by Jet serves this purpose.

Jet SQL Reserved Words, Operators, and Functions Not in ANSI SQL

Jet SQL contains a number of reserved words that aren't ANSI SQL keywords (see Table 24.2). Most of these reserved words define Jet data types; some reserved words have equivalents in ANSI SQL, and others don't. You use Jet DDL reserved words to modify the properties of tables. Access VBA's SQL property DISTINCTROW is described in the following section. PIVOT and TRANSFORM are used in creating crosstab queries that are unique to Jet databases. (SQL Server 6.5's Transact-SQL adds the ROLLUP and CUBE functions that provide aggregation capabilities similar to PIVOT and TRANSFORM.)

Table 24.2 Jet SQL Reserved Words Not in ANSI SQL

Jet SQL	ANSI SQL	Category	Purpose
BINARY	No equivalent	DDL	Not an official Jet field data type.
BOOLEAN	No equivalent	DDL	Jet Yes/No field data type.
BYTE	No equivalent	DDL	Byte field data type, 1 byte integer.
CURRENCY	No equivalent	DDL	Jet Currency field data type.
DATETIME	No equivalent	DDL	Jet Date/Time field data type.
DISTINCTROW	No equivalent	DQL	Updatable Jet Recordset objects.
DOUBLE	REAL	DDL	High-precision decimal numbers.
LONG	INT[EGER]	DDL	Long Integer field data type.
LONGBINARY	No equivalent	DDL	OLE Object field data type.
LONGTEXT	VARCHAR	DDL	Memo field data type.
OWNERACCESS	No equivalent	DQL	Run with owner's privileges parameters.
PIVOT	No equivalent	DQL	Used in crosstab queries.
SHORT	SMALLINT	DDL	Integer field data type, 2 bytes.
SINGLE	No equivalent	DDL	Single-precision real number.

(continues)

Oracle Ex. 1009, p. 854
Oracle v. DeBellis
IPR2024-00221

Table 24.2 Continued

TEXT	CHAR[ACTER]	DDL	Text field data type.
TRANSFORM	No equivalent	DQL	Creates crosstab query.
? (LIKE wildcard) with LIKE.	_(wild card)	DQL	Single character.
* (LIKE wild card)	% (wild card)	DQL	Zero or more characters.
# (LIKE wild card)	No equivalent	DQL	Single digit, 0–9.
# (date specifier)	No equivalent	DQL	Encloses date/time values.
<> (not equal)	!=	DQL	Jet uses ! as a separator.

Jet provides four statistical aggregate functions that aren't incorporated in ANSI SQL. These functions are listed in Table 24.3.

Table 24.3 Aggregate SQL Functions Added in Jet SQL

Jet Function	Category	Purpose
StdDev()	DQL	Standard deviation of a population sample.
StdDevP()	DQL	Standard deviation of a population.
Var()	DQL	Statistical variation of a population sample.
VarP()	DQL	Statistical variation of a population.

Jet's *DISTINCTROW* and SQL's *DISTINCT* Keywords

The DISTINCTROW reserved word that follows the SQL SELECT keywords causes Jet to eliminate duplicated rows from the query's result. The effect of DISTINCTROW is especially dramatic in queries used to display records in tables that have indirect relationships. To create an example of a query that you can use to demonstrate the effect of Jet's DISTINCTROW SQL reserved word, follow these steps:

New

1. Open a new query in Nwind.mdb by clicking the Query tab in the Database window and then clicking the New button. Select Design View in the list box of the New Query window, and then click OK to bypass the Query wizards.
2. In the Show Table dialog, add the Customers, Orders, Order Details, Products, and Categories tables to the query, in sequence. Access automatically creates the required joins. Click Close to close the Show Table dialog.
3. Drag the CompanyName field from the Customers field list to the Field row of the first column of the Query Design grid. Select the Sort cell, open the drop-down list with F4, and choose Ascending Sort Order.

Oracle Ex. 1009, p. 855
Oracle v. DeBellis
IPR2024-00221

4. Drag the CategoryName field from the Categories field list to the Field row of the second column of the grid. Add an ascending sort to this field (see Figure 24.1).

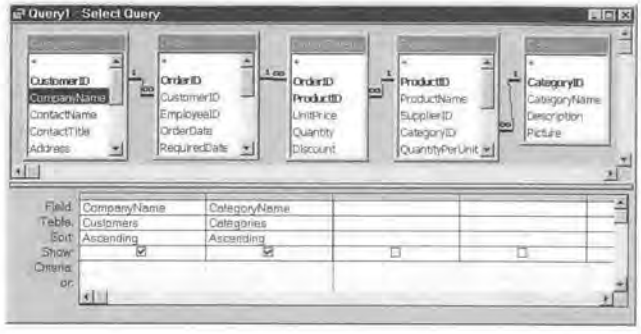


FIG. 24.1 The QBE design of a query to determine customers purchasing categories of products.

5. Choose View, SQL View. The SQL statement that creates the query is shown in the SQL window in Figure 24.2.

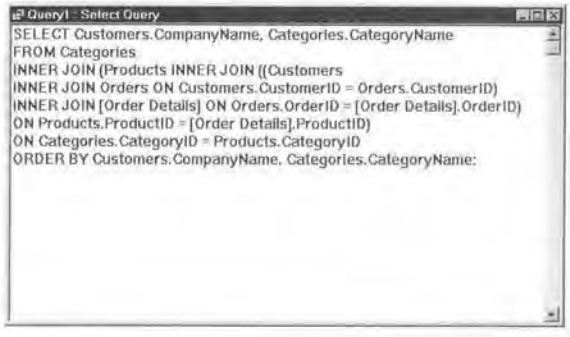


FIG. 24.2 The SQL statement corresponding to the query design of Figure 24.1.

6. Click the Run Query button on the toolbar to execute the query. The number of rows (records) returned by the query—2,156 at the time of this writing—appears to the right of the navigation buttons at the bottom of the Query Datasheet window (see Figure 24.3). There are many duplicate rows in the query result set.



Oracle Ex. 1009, p. 856
Oracle v. DeBellis
IPR2024-00221

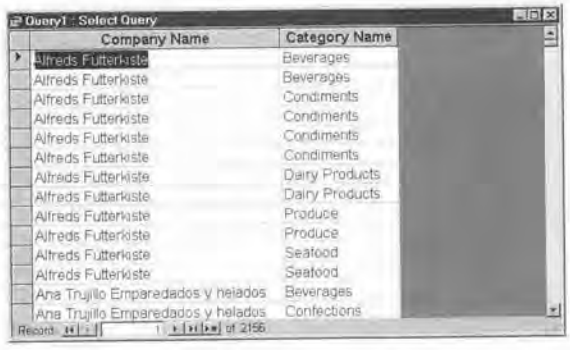


FIG. 24.3 The query result set for the SQL statement of Figure 24.2.

To demonstrate the effect of adding the `DISTINCTROW` reserved word to the SQL statement and to verify, in this case, that the effect of ANSI SQL's `DISTINCT` keyword and Jet SQL's `DISTINCTROW` reserved word is the same, follow these steps:

1. Choose View, SQL to edit the SQL statement.
2. Add the `DISTINCTROW` keyword to the SQL statement after the `SELECT` command.
3. Click the Run Query button. The new query result set has 598 rows with no duplicated rows (see Figure 24.4).

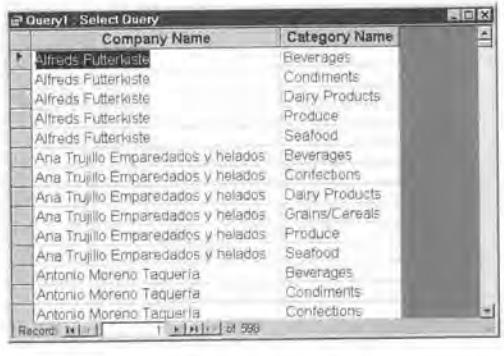


FIG. 24.4 Eliminating duplicate rows with the Jet SQL `DISTINCTROW` modifier.

4. Choose View, SQL to edit the SQL statement again. Replace `DISTINCTROW` with `DISTINCT`.
5. Click the Run Query button again. You get the same result of 598 records that you obtain when you use the `DISTINCTROW` keyword.
6. Close, but do not save, the query.

Oracle Ex. 1009, p. 857
Oracle v. DeBellis
IPR2024-00221

Note

As an alternative to choosing View, SQL View from the Query menu, you can click the arrow adjacent to the View button of the toolbar, and then select SQL View from the drop-down menu. The number of mouse clicks is the same in either case.



A second option is to right-click the mouse button on a blank area of the upper pane of the Query Design window and choose SQL View from the pop-up menu.

`DISTINCTROW` is a special Jet SQL reserved word and is unavailable in standard (ANSI) SQL; `DISTINCTROW` is related to, but not the same as, the `DISTINCT` keyword in ANSI SQL. Both words eliminate duplicate rows of data in query result tables, but they differ in execution. `DISTINCT` in ANSI SQL eliminates duplicate rows based only on the values of the data contained in the rows of the query, from left to right. You cannot update values from multiple-table queries that include the keyword `DISTINCT`.

`DISTINCTROW`, available only in Access, eliminates duplicate rows based on the content of the underlying table, regardless of whether additional field(s) that distinguish records in the table are included. `DISTINCTROW` allows values in special kinds of multiple-table `Recordset` objects to be updated.

To distinguish between these two keywords, assume that you have a table with a Last_Name field and a First_Name field and only 10 records, each with the Last_Name value, Smith. Each record has a different First_Name value. You create a query that includes the Last_Name field but not the First Name field. `DISTINCTROW` returns all 10 Smith records because the First Name values differ in the table. `DISTINCT` returns one record because the First_Name field that distinguishes the records in the table is absent in the query result table.



Earlier versions of Access included the default reserved word `DISTINCTROW`, unless you purposely replaced it with the `DISTINCT` keyword by using the Query Properties dialog's Unique Values Only option. Access 97's Query Properties dialog sets the value of the Unique Values (`DISTINCT`) and Unique Rows (`DISTINCTROW`) properties to No (see Figure 24.5). Set either of these query property values to Yes to automatically add the modifier to `SELECT` queries.



FIG. 24.5 Access 97's Query Properties dialog with default query property values.

Oracle Ex. 1009, p. 858
Oracle v. DeBellis
IPR2024-00221

Creating Tables with Jet DDL

You can create new tables in your current database with Jet 3.5's Data Definition Language reserved words. Using SQL to create new tables is of primary interest to developers, not users, of Access applications, because it is much easier to create new tables with the Access user interface. For the sake of completeness, however, a brief description of Access 95 SQL DDL statements follows:

- `CREATE TABLE table_name (field_name data_type [field_size][, field_name data_type...])`. Creates a new table with the fields specified by a comma-separated list. Properties of fields are space-delimited, so you need to enclose entries for *field names* with spaces in square brackets ([]). The *data_type* can be any valid Jet SQL field data type, such as `TEXT` or `INTEGER`. The *field_size* entry is optional for `TEXT` fields only. (The default value is 50 characters.)
- `CONSTRAINT index_name {PRIMARYKEY¦UNIQUE¦REFERENCES foreign_table [(foreign_field)]}`. Creates an index on the field name that precedes the expression. You can specify the index as the `PRIMARYKEY` or as an `UNIQUE` index. You also can establish a relationship between the field and the field of a foreign table with the `REFERENCES foreign_table [foreign_field]` entry. (The `[foreign_field]` item is required if the *foreign_field* is not a primary-key field.)
- `CREATE [UNIQUE] INDEX index_name ON table_name (field_name [ASC¦DESC][, field_name [ASC¦DESC], ...]) [WITH {PRIMARY¦DISALLOW NULL¦IGNORE NULL}]`. Creates an index on one or more fields of a table. If you specify the `WITH PRIMARY` modifier, `UNIQUE` is assumed (and not required). `DISALLOW NULL` prevents addition of records with `NULL` values in the indexed field; `IGNORE NULL` doesn't index records with `NULL` field_name values.
- `ALTER TABLE`. Allows you to add new fields (`ADD COLUMN field_name...`) or delete existing fields (`DROP COLUMN field_name...`).
- `DROP INDEX index_name ON table_name`. Deletes the index from a table specified by *table_name*.
- `DROP TABLE table_name`. Deletes a table from the database.

Common ANSI SQL Keywords and Features Not Supported by Jet SQL Reserved Words

The majority of the ANSI SQL keywords that aren't supported by Jet 3.5 are elements of SQL's Data Control Language. Transactions, which are implemented automatically for most operations by the Jet 3.5 database engine, can be explicitly declared only in Access VBA code. The record position buttons of Access queries and forms substitute for most Cursor Control Language statements in ANSI SQL that choose a particular row in a query. Table 24.4 lists these substitutes.

Oracle Ex. 1009, p. 859
Oracle v. DeBellis
IPR2024-00221

Table 24.4 Common ANSI SQL Reserved Words Not Supported in Jet SQL

Reserved Word	Category	Substitute
AUTHORIZATION	DCL	Privileges dialog
BEGIN	TPL	Access VBA BeginTrans method
CHECK	DQL	Table Validation Rule property
CLOSE	CCL	Document Control menu of query
COMMIT	TPL	Access VBA CommitTrans method
CREATE VIEW	DDL	Query Design mode and filters
CURRENT	CCL	Query Run mode, record position buttons
CURSOR	CCL	Query Run mode
DECLARE	CCL	Query Run mode (cursors are automatic)
DROP VIEW	DDL	Query Design mode
FETCH	DQL	Text boxes on a form or report
GRANT	DCL	Privileges dialog
PRIVILEGES	DCL	Privileges dialog
REVOKE	DCL	Privileges dialog
ROLLBACK	TPL	Access VBA Rollback method
TRANSACTION	TPL	Access VBA transaction methods
VALUES	DML	Data values entered in tables or forms
WORK	TPL	Access VBA BeginTrans method
: (variable)	DQL	Access VBA **Dim** statement prefix
!= (not equal)	DQL	Access <> not-equal operator

The Jet 3.5 database engine uses transaction processing for all Jet DML commands executed by action queries. You implement transaction processing (SQL's COMMIT and ROLLBACK [WORK]) on Recordset objects you create with code by writing Access VBA functions or procedures that contain the BeginTrans, CommitTrans, and Rollback reserved words. Many other less commonly used SQL keywords, such as COBOL and PASCAL, don't have Jet SQL reserved word equivalents.

Writing Select Queries in SQL

When you create a SELECT query in Query Design mode, Access translates the QBE query design into a Jet SQL SELECT statement. You can view the Jet SQL equivalent of your design at any point by choosing View, SQL View. Displaying and analyzing the SQL statements that correspond to queries you design or queries in the Northwind Traders sample database is useful when you are learning SQL.

The heart of SQL is the SELECT statement used to create a select query. Every select query begins with the SELECT statement. The following lines of syntax are used for a SQL SELECT

Oracle Ex. 1009, p. 860
Oracle v. DeBellis
IPR2024-00221

statement that returns a query table (called a *result set*, usually a `Recordset` object of the `Dynaset` type) of all or selected columns (fields) from all or qualifying rows (records) of a source table:

```
SELECT [ALL¦DISTINCT¦DISTINCTROW] [TOP n [PERCENT]] select_list
   FROM table_names
  [WHERE search_criteria]
  [ORDER BY column_criteria [ASC¦DESC]]
```

The following list shows the purpose of the elements in this basic select query statement:

- `SELECT`. The basic command that specifies a query. The *`select_list`* parameter determines the fields (columns) that are included in the result table of the query. When you design an Access QBE query, the *`select_list`* parameter is determined by the fields you add to the Fields row in the Query grid. Only those fields with the Show check box marked are included in *`select_list`*. Multiple field names are separated by commas.

  The optional `ALL`, `DISTINCT`, and `DISTINCTROW` qualifiers determine how rows are handled. `ALL` specifies that all rows are to be included, subject to subsequent limitation. `DISTINCT` eliminates rows with duplicate data. As discussed in the section "Jet's `DISTINCTROW` and SQL's `DISTINCT` Keywords" earlier in the chapter, `DISTINCTROW` is a Jet SQL keyword, similar to `DISTINCT`, that eliminates duplicate rows but also enables you to modify the query result set.

  The optional `TOP` *`n`* `[PERCENT]` modifier limits the query result set to returning the first *`n`* rows or *`n`* percent of the result set prior to the limitation. `TOP` and `PERCENT` are Jet SQL, not ANSI SQL, keywords. You use the `TOP` modifier to speed display when you only want to display the most significant rows of a query result set.

- `FROM` *`table_name`*. Specifies the name or names of the table or tables that form the basis for the query. The *`table_names`* parameter is created in Access QBE by the entries you make in the Add Table dialog. If fields from more than one table are included in the *`select_list`*, each table must be specified in the *`table_names`* parameter. Commas are used to separate the names of multiple tables.

- `WHERE` *`search_criteria`*. Determines which records from the select list are displayed. The `search_criteria` parameter is an expression with a text (string) operator, such as `LIKE`, for text fields or a numeric operator, such as `>=`, for fields with numeric values. The `WHERE` clause is optional; if you don't add a `WHERE` clause, all the rows that meet the `SELECT` criteria are returned.

- `ORDER BY` *`column_criteria`*. Specifies the sorting order of a `Recordset` object of the `Dynaset` or `Snapshot` type created by the query. A `Recordset` object of the `Snapshot` type is a query result set that is not updatable. Like the `WHERE` clause, `ORDER BY` is optional. You can specify an ascending or descending sort by the optional `ASC` or `DESC` keywords. If you don't specify a sort direction, ascending is the default.

Oracle Ex. 1009, p. 861
Oracle v. DeBellis
IPR2024-00221

The following lines show an example of a simple Jet SQL query statement:

```
SELECT CompanyName, CustomerID, PostalCode
   FROM Customers
   WHERE PostalCode LIKE '9*'
   ORDER BY CompanyName;
```

You must terminate a Jet SQL statement by adding a semicolon immediately after the last character on the last line.

Note

Examples of SQL statements in this book are formatted to make the examples more readable. Access doesn't format the SQL statements. When you enter or edit SQL statements in the Jet SQL window, formatting these statements so that commands appear on individual lines makes the SQL statements more intelligible. Use Ctrl+Enter to insert newline pairs (the return and newline characters) before SQL keywords. Jet ignores spaces and newline pairs when it processes the SQL statement.

The preceding query results in a Jet `Recordset` object of three columns and as many rows as the number of records in the Customers table for companies located in ZIP codes with values that begin with the character 9, sorted alphabetically by the company name. You don't have to specify the table name with the field name in the *`select_list`* because only one table is used in this query. When Access creates a SQL statement, the table name always precedes the field name. Usually, Access processes queries you write in either ANSI SQL or Jet SQL syntax. This example differs from ANSI SQL only in the substitution of the Jet SQL `*` (asterisk) for ANSI SQL's `%` wild card.

Using SQL Punctuation and Symbols

In addition to the comparison operators used for expressions, SQL uses commas, periods, semicolons, and colons as punctuation. The following list of symbols and punctuation is used in ANSI SQL and the Jet SQL dialect; differences between the two forms of SQL are noted where appropriate:

- Commas are used to separate members of lists of parameters, such as multiple field names, as in `Name, Address, City, ZIP`.
- Square brackets surrounding field names are required only when the field name includes spaces or other symbols—including punctuation—not allowed by SQL, as in `[Company Name]`.
- If fields of more than one table are involved in the query, a period is used to separate the table name from the field name, as in `Customers.[Company Name]`.
- ANSI SQL uses the single quote symbol (`'`) to enclose literal string values. You can use the double quote (`"`) or the single quote symbol to enclose literal values in Jet SQL statements. Using the single quote makes writing SQL statements in Access VBA easier.

Oracle Ex. 1009, p. 862
Oracle v. DeBellis
IPR2024-00221

- ANSI SQL uses % and _ (underscore) symbols as the wild cards for the LIKE statement, rather than the * (asterisk) and ? used by Jet SQL to specify zero or more characters and a single character, respectively. The Jet wild cards correspond to the wild cards used in specifying DOS group file names.
- Jet provides the # wild card for the LIKE statement to represent any single digit. VBA also uses the # symbol to enclose date/time values in expressions. This symbol isn't available in ANSI SQL.
- The end of an Jet SQL statement is indicated by a optional semicolon (;).
- In Jet SQL, you cannot use colons as prefixes to indicate user-declared variables you create in ANSI SQL. You cannot create variables with Jet SQL; user-declared variables in Access are limited to the Access VBA functions and procedures you write. You can, however, pass variable values as parameters to stored procedures of client/server RDBMSes. Using stored procedures requires employing Access's SQL pass-through option.
- ANSI SQL uses the ! (bang symbol or exclamation mark) as a `not in` operator for character lists used with LIKE. ANSI SQL uses != for not equal; Jet SQL uses <>.



▶▶ See "Using Access 97's SQL Pass-Through Queries," p. 940

As the preceding list demonstrates, relatively minor differences exist in the availability and use of punctuation and symbols between ANSI and Jet SQL.

> **Note**
>
> Indentation often is used in writing multiple-line SQL statements. Indented lines indicate continuation of a preceding line or a clause that is dependent on a keyword in a preceding line.

Using SQL Statements to Create Access Queries

You can enter SQL statements in Query Design mode to create simple Access queries that are reflected in changes to the design of the Query grid. This method is another useful way to learn the syntax of SQL. If your entries contain errors in spelling or punctuation, Access displays a message box that describes the error and its approximate location in the statement. When you choose OK in the SQL dialog, Access translates your SQL statement into a QBE design.

To create an Access QBE select query with the SQL statement, follow these steps:



1. Open the Northwind Traders database, if necessary, and then open a new query in Design view.
2. Close the Show Table dialog without adding a table name.
3. Choose View, SQL View to open the SQL window.
4. Delete any text, such as `SELECT;`, that may appear in the SQL window.

Oracle Ex. 1009, p. 863
Oracle v. DeBellis
IPR2024-00221

5. Enter the following SQL statement in the SQL window. Use Ctrl+Enter to create new lines. Your SQL statement appears as shown in Figure 24.6.

```
SELECT CompanyName, CustomerID, PostalCode
   FROM Customers
   WHERE PostalCode LIKE '9*'
   ORDER BY CompanyName;
```

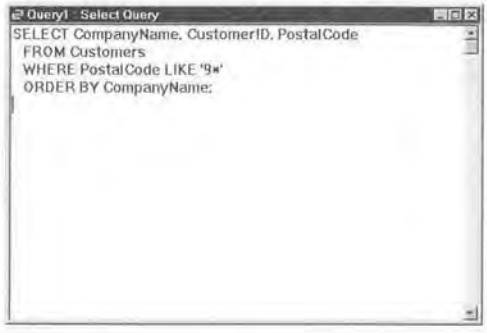


FIG. 24.6 An SQL statement for a simple select query.

Note

Access uses double quotes (") to identify string (text) values in SQL statements. As noted earlier in the chapter, the preferred SQL string identifier is the single quote (').

6. Choose View, Design View. Access creates the equivalent of your SQL statement in graphical QBE (see Figure 24.7).

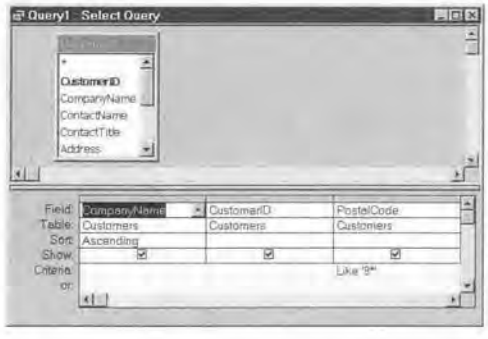


FIG. 24.7 The QBE design created by Access from the query in Figure 24.6.

7. Click the Run Query button on the toolbar. The result of your query in Datasheet view appears as shown in Figure 24.8.



Oracle Ex. 1009, p. 864
Oracle v. DeBellis
IPR2024-00221



FIG. 24.8 The query in Figure 24.7 in Datasheet view.

To change the order by which the query result is sorted, follow these steps:

1. Choose View, S_QL View to open the SQL window.
2. Change `ORDER BY CompanyName` to `ORDER BY PostalCode`, and choose View, Design View to open the Query Design window.

   The Query grid in Design mode displays `Ascending` in the PostalCode column rather than in the CompanyName column, indicating that the query result set is sorted by ZIP code.



3. Click the Run button of the toolbar to display the result set sorted in ZIP code sequence (see Figure 24.9).
4. Close, but do not save, the query.



FIG. 24.9 The query of Figure 24.7 in ZIP code order.

Using the SQL Aggregate Functions

If you want to use the aggregate functions to determine totals, averages, or statistical data for groups of records with a common attribute value, you add a `GROUP BY` clause to your SQL statement. You can further limit the result of the `GROUP BY` clause with the optional `HAVING` qualifier:

Oracle Ex. 1009, p. 865
Oracle v. DeBellis
IPR2024-00221

```
SELECT [ALL¦DISTINCT¦DISTINCTROW]
      aggregate_function(field_name) AS alias_name
     [, select_list]
  FROM table_names
 [WHERE search_criteria]
  GROUP BY group_criteria
    [HAVING aggregate_criteria]
 [ORDER BY column_criteria]
```

The *select_list* includes the *aggregate_function* with a field_name as its argument. The field used as the argument of an aggregate function must have a numeric data type. The additional SQL keywords and parameters required to create a GROUP BY query are described in the following list:

- AS *alias_name*. Assigns a caption to the column. The caption is created in an Access QBE query by the *alias:aggregate_function(field name)* entry in the Field row of the Query grid.
- GROUP BY *group_criteria*. Establishes the column on which the grouping is based. In this column, GROUP BY appears in the Totals row of the Query grid.
- HAVING *aggregate_criteria*. One or more criteria applied to the column that contains the *aggregate_function*. The *aggregate_criteria* of HAVING is applied after the grouping is completed.
- WHERE *search_criteria*. Operates before the grouping occurs; at this point, no aggregate values exist to test against *aggregate_criteria*. Access substitutes HAVING for WHERE when you add criteria to a column with the *aggregate_function*.

Note

Not all client/server RDBMSes use the ANSI SQL AS *alias_name* construct. Microsoft SQL Server, Sybase System 10+, and IBM DB2, as examples, substitute a space for the AS keyword, as in SELECT *field_name alias_name, ...*. The ODBC driver for these databases uses special codes (called *escape syntax*) to change from Access/ANSI use of AS to the space separator. If you use Access's SQL pass-through feature with Access VBA, you must use the space separator, not the AS keyword, for most client/server RDBMSes.

▶▶ See "Using Access 97's SQL Pass-Through Queries," p. 940



The following GROUP BY query is written in ANSI SQL, except for the # symbols that enclose date and time values:

```
SELECT ShipRegion, SUM(Freight) AS [Total Freight]
   FROM Orders
   WHERE ShipCountry="USA"
      AND OrderDate BETWEEN #01/1/94# AND #12/31/96#
   GROUP BY ShipRegion
   HAVING SUM(Freight)  > 50
   ORDER BY SUM(Freight) DESC;
```


Oracle Ex. 1009, p. 866
Oracle v. DeBellis
IPR2024-00221

The query returns a result set that consists of two columns: Ship Region (states) and the totals of Freight for each Ship Region in the United States, for the years 1994 through 1996. The result set is sorted in descending order.

To create an SQL `GROUP BY` query in Access, follow these steps:

New

1. Open a new query in Design view, close the Show Table dialog, and choose View, SQL View. Type the preceding `GROUP BY` example code in the SQL window (see Figure 24.10).

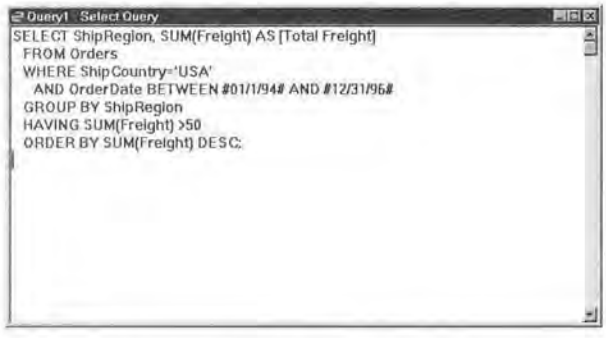


FIG. 24.10 An SQL statement that uses the `SUM()` aggregate function.

2. Choose View, Design View. Your QBE `GROUP BY` query design appears as shown in Figure 24.11.

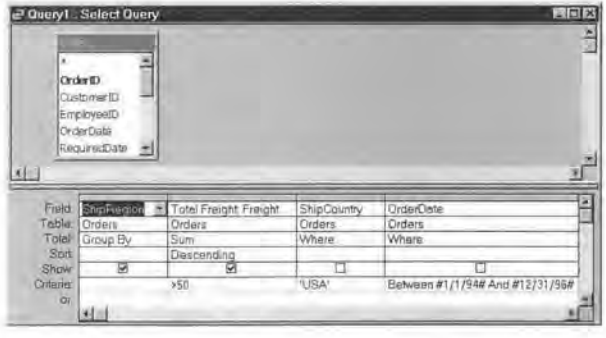


FIG. 24.11 Access's QBE design for the query shown in Figure 24.10.

3. Click the Run Query button on the toolbar. The states with orders having freight charges equal to $50 or more during the period 1/1/94 through 12/31/96 are shown ranked by total freight costs and are displayed in Datasheet view (see Figure 24.12).

4. Close, but do not save, your query.

Oracle Ex. 1009, p. 867
Oracle v. DeBellis
IPR2024-00221



FIG. 24.12 The aggregate query design in Figure 24.7 in Datasheet view.

Creating Joins with SQL

Joining two or more tables with Access QBE uses the `JOIN_ON` structure that specifies the table to be joined and the relationship between the fields on which the `JOIN` is based:

```
SELECT [ALL¦DISTINCT¦DISTINCTROW]select_list
   FROM
   table_name {INNER¦LEFT¦RIGHT} JOIN join_table
      ON join_criteria
  [table_name {INNER¦LEFT¦RIGHT} JOIN join_table
      ON join_criteria]
  [WHERE search_criteria]
  [ORDER BY column_criteria]
```

The elements of the `JOIN` statement are shown in the following list:

- *`table_name`* `{INNER¦LEFT¦RIGHT} JOIN` *`join_table`*. Specifies the name of the table that is joined with other tables listed in *`table_name`*s. Each of the tables participating in a join must be included before the `JOIN` clause. When you specify a self-join by including two copies of the field list for a single table, the second table is distinguished from the first by adding an underscore and a digit to the table name.

  One of the three types of joins, `INNER`, `LEFT`, or `RIGHT` must precede the `JOIN` statement. INNER specifies an equi-join; LEFT specifies a left outer join; RIGHT specifies a right outer join. The type of join is determined in Access QBE by double-clicking the line connecting the joined fields in the table and clicking option button 1, 2, or 3 in the Join Properties dialog.

- `ON` *`join_criteria`*. Specifies the two fields to be joined and the relationship between the joined fields. One field is in `join_table` and the other is in a table in `table_names`. The *`join_criteria`* expression contains an equal sign (=) comparison operator and returns a `True` or `False` value. If the value of the expression is `True`, the record in the joined table is included in the query.

The number of `JOIN` statements you can add to a query usually is the total number of tables participating in the query minus one. You can create more than one `JOIN` between a pair of tables, but the result often is difficult to predict.

Oracle Ex. 1009, p. 868
Oracle v. DeBellis
IPR2024-00221

Access QBE creates nested JOIN statements for multiple JOINs. As an example, Figure 24.13 illustrates a query design with successive JOINs between the Orders, Order Details, Products, and Categories tables of Nwind.mdb. The SQL syntax for the three nested JOINs appears in Figure 24.14. The SQL statement has been formatted so that the ON clause for each JOIN statement is aligned vertically. The JOIN reserved word in Jet SQL creates the lines that connect the joined fields in Query Design view.

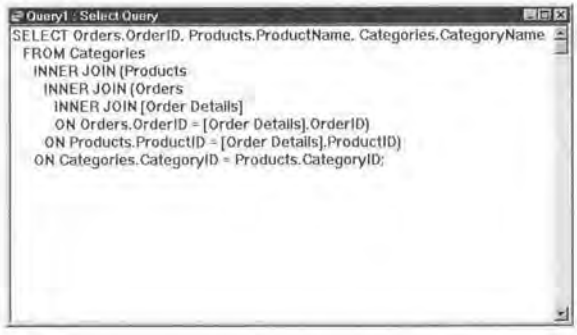


FIG. 24.13 The Jet SQL implementation of three nested equi-joins.

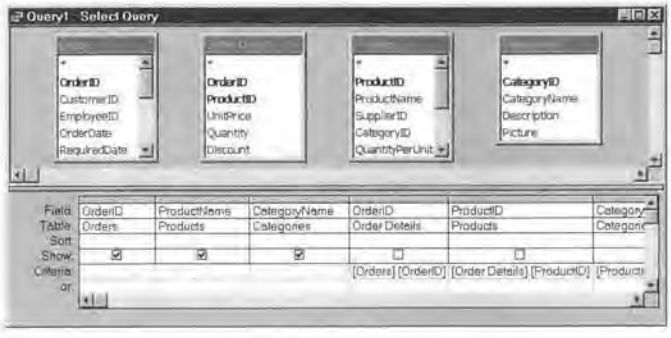


FIG. 24.14 The equi-join of Figure 24.13 created by a WHERE clause.

You create equi-joins in ANSI SQL with the WHERE clause, using the same expression to join the fields as that of the ON clause in the JOIN command. It's much simpler to write SQL statements using WHERE clauses for joins than employing the JOIN syntax. The WHERE clause also is more flexible than the JOIN_... ON structure because you can use other operators such as BETWEEN_AND, LIKE, >, and <. These operators result in error messages when they are substituted for the equal sign (=) in the ON clause of the JOIN statement.

The following ANSI SQL-89 statement gives the same result as the Jet SQL statement in Figure 24.13:

```
SELECT Orders.OrderID, Products.ProductName,
      Categories.CategoryName
   FROM Orders, [Order Details], Products, Categories
   WHERE [Order Details].OrderID = Orders.OrderID
```


Oracle Ex. 1009, p. 869
Oracle v. DeBellis
IPR2024-00221

```
AND Products.ProductID = [Order Details].ProductID
AND Categories.CategoryID = Products.CategoryID;
```

You create multiple joins with WHERE clauses by separating each join expression with an AND operator. Figure 24.14 shows the Query Design view of the preceding SQL statement. JOIN syntax isn't used, so no join lines appear between the field lists in the upper pane. In this case, the Criteria row of the QBE grid displays only the left element of the argument of each WHERE statement.

Using *UNION* Queries

UNION queries let you combine the result set of two or more SELECT queries into a single result set. Nwind.mdb includes an example of a UNION query, which has the special symbol of two overlapping circles, in the Database window. You can create UNION queries only with SQL statements; if you add the UNION keyword to a query, the Query Design Mode button on the toolbar and the Query Design choice on the View menu are disabled. The general syntax of UNION queries is as follows:

```
SELECT select_statement
   UNION SELECT select_statement
     [GROUP BY group_criteria]
     [HAVING aggregate criteria]
  [UNION SELECT select_statement
     [GROUP BY group_criteria]
     [HAVING aggregate criteria]]
  [UNION. . .]
  [ORDER BY column_criteria]
```

The restrictions on statements that create UNION queries are the following:

- The number of fields in the field_list of each SELECT and UNION SELECT query must be the same. You receive an error message if the number of fields is not the same.
- The sequence of the field names in each field_list must correspond to similar entities. You don't receive an error message for dissimilar entities, but the result set is likely to be unfathomable. The field data types in a single column need not correspond; however, if the column of the result set contains both numeric and Text data types, the data type of the column is set to Text.
- Only one ORDER BY clause is allowed, and it must follow the last UNION SELECT statement. You can add GROUP BY and HAVING clauses to each SELECT and UNION SELECT statement, if needed.

Figure 24.15 shows the SQL statement to create a UNION query derived from the Union Query included in Nwind.mdb. The syntax of the SQL statement illustrates the ability of UNION queries to include values from two different field data types, Text (CustomerID) and Long Integer (SupplierID), in the single, aliased ID column. The query result set appears in Figure 24.16 with the single record included from the Suppliers table selected.

Oracle Ex. 1009, p. 870
Oracle v. DeBellis
IPR2024-00221

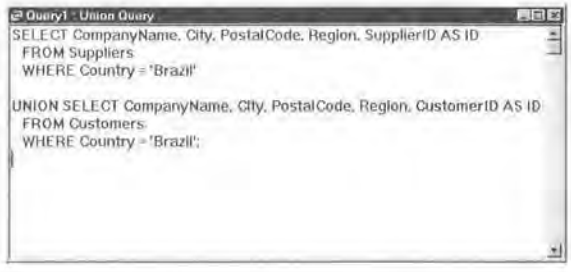


FIG. 24.15 Creating a multiple-column `UNION` query with a SQL statement.

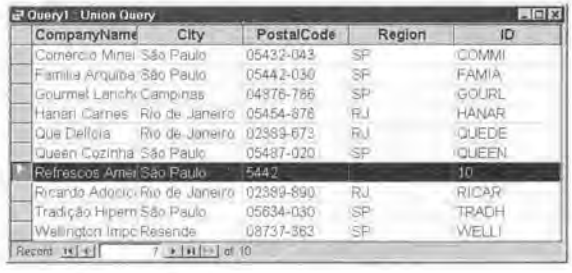


FIG. 24.16 The result of the `UNION` query of Figure 24.15.

Implementing Subqueries

Versions of Access prior to 2.0 used nested queries to emulate the subquery capability of ANSI SQL. (A *nested query* is a query executed against the result set of another query.) Access 97 lets you write a `SELECT` query that uses another `SELECT` query to supply the criteria for the `WHERE` clause. Depending on the complexity of your query, using a subquery instead of nested queries often improves performance. The general syntax of subqueries is as follows:

```
SELECT field_list
   FROM table_list
   WHERE [table_name.]field_name
      IN (SELECT select_statement
  [GROUP BY group_criteria]
     [HAVING aggregate_criteria]
  [ORDER BY sort_criteria]);
```

Figure 24.17 shows the SQL statement for an example subquery that returns names and addresses of Northwind Traders customers who placed orders between January 1, 1996 and June 30, 1996. The `SELECT` subquery that begins after the `IN` predicate returns the CustomerID values from the Orders table against which the CustomerID values of the Customers table are compared.

Oracle Ex. 1009, p. 871
Oracle v. DeBellis
IPR2024-00221

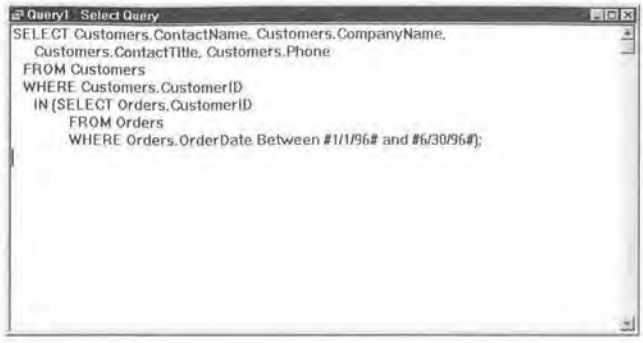


FIG. 24.17 The SQL statement for a simple subquery.

Unlike `UNION` queries, you can create a subquery in Query Design mode. You type **`IN`**, followed by the `SELECT` statement as the criterion of the appropriate column, enclosing the `SELECT` statement within the parenthesis required by the `IN` predicate. Figure 24.18 shows the query design with part of the `IN (SELECT` statement in the Criteria row of the Customer ID column. Figure 24.19 shows the result set returned by the SQL statement of Figure 24.17.

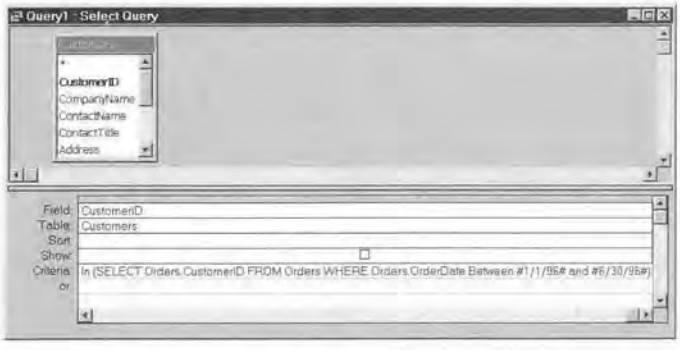


FIG. 24.18 Entering the SQL statement for a subquery in the Criteria row.



FIG. 24.19 The query result set from the subquery of Figures 24.17 and 24.18.

Oracle Ex. 1009, p. 872
Oracle v. DeBellis
IPR2024-00221

Specifying Action Query Syntax

Data Manipulation Language (DML) commands are implemented by Access's action queries: append, delete, make-table, and update. Jet SQL reserved words that create crosstab queries—TRANSFORM and PIVOT—are included in this section because crosstab queries are related to DML queries. This sections shows the syntax for each type of Access action query.

Append queries use the following syntax:

```
INSERT INTO dest_table
   SELECT [ALL¦DISTINCT¦DISTINCTROW] select_list
   FROM source_table
  [WHERE append_criteria]
```

If you omit the WHERE clause, all the records of *source_table* are appended to *dest_table*.

Delete queries take the following form:

```
DELETE FROM table_name
  [WHERE delete_criteria]
```

If you omit the optional WHERE clause in a delete query, you delete all data in *table_name*.

Make-table queries use the following syntax:

```
SELECT [ALL¦DISTINCT¦DISTINCTROW] select_list
   INTO new_table
   FROM source_table
  [WHERE append_criteria]
```

To copy the original table, substitute an asterisk (*) for select_list and omit the optional WHERE clause.

Update queries use the SET command to assign values to individual columns:

```
UPDATE table_name
   SET column_name = value [, column_name = value[, ...]]
  [WHERE update_criteria]
```

Separate multiple *column_name* entries and corresponding values by commas if you want to update the data in more than one field. Jet SQL supports the ANSI SQL VALUES keyword for adding records to tables the hard way (specifying the VALUE of each column of each record).

Crosstab queries use the Jet SQL keywords TRANSFORM and PIVOT to create various types of summary queries using the SQL aggregate functions. The following syntax applies to time-series crosstab queries:

```
TRANSFORM aggregate_function(field_name) [AS alias]
SELECT [ALL¦DISTINCT¦DISTINCTROW] select_list
   FROM table_name
   PIVOT Format(field_name,"format_type")
  [IN (column_list)]
```


Oracle Ex. 1009, p. 873
Oracle v. DeBellis
IPR2024-00221

TRANSFORM defines a crosstab query, and PIVOT specifies the GROUP BY characteristics plus the fixed column names specified by the optional IN predicate. Crosstab queries, like queries with multiple or nested JOINs, are better left to Access QBE to create the query. You can edit the query as necessary after Access has written the initial SQL statement.

▶▶ See "Creating Crosstab Queries" p. 361



Troubleshooting

When I try to execute a query from my SQL statement, an Enter Parameter dialog appears.

You misspelled one of the table names in your `table_list`, one of the field names in your `field_list`, or both. If the Jet engine's query parser can't match a table name or a field name with those specified in the FROM clause, Jet assumes that the entry is a parameter and requests its value. Check the spelling of the database objects in your SQL statement. (If you misspell a SQL keyword, you usually receive a syntax error message box.)

Adding *IN* to Use Tables in Another Database

Access enables you to open only one database at a time, unless you write code to open another table with an Access VBA function or subprocedure. However, you can use Jet SQL's IN clause with a make-table, append, update, or delete query to create or modify tables in another database. Access provides the capability to make a table or append records to a table in another Access database through graphical QBE only. You click the Another Database option in the Query Properties dialog for the make-table or append query and type the file name of the other database.

You have to write an SQL query or edit a query created by Access to update data or delete records in tables contained in another database of any type, or to perform any operation on a dBASE, Paradox, or Btrieve file that isn't attached to your database. The SQL query uses the IN clause to specify the external database or table file. The advantage of using the IN clause is simplicity—you don't have to attach the table before using it. The disadvantage of using the IN clause is that indexes associated with dBASE and Paradox tables aren't updated when the content of the table is modified.

Working with Another Access Database

You can create a table in another Access database, delete all the records, and then append the records back to the table from which the records were deleted, using the IN clause to specify the name of the other database that contains the table. To try using the IN clause, open a new query or an existing query and follow these steps:

1. Choose View, SQL View to open the SQL window.
2. Delete any existing text if you have a query open, and type the following line in the SQL window:

Oracle Ex. 1009, p. 874
Oracle v. DeBellis
IPR2024-00221

SELECT * INTO Customers IN "OLE Objects.mdb" FROM Customers;

SELECT_INTO creates a make-table query. If you haven't created OLE Objects.mdb, choose File, New Database and create a new blank database named OLE **Objects.mdb** or whatever you like in your ...\Office\Samples folder. If the new database is in a different location, you need to add the path to the IN string. Reopen Nwind.mdb.



3. Click the Run Query button of the toolbar to make the new Customers table in your OLE Objects database. Click OK when the message box advises you of the number of records that are copied to the new Customers table created in OLE Objects. You can open OLE Objects.mdb to verify the existence of the new table.

4. Choose View, SQL View again, delete the existing text, and type the following line:

DELETE * FROM Customers IN "OLE Objects.mdb"

DELETE...FROM creates a delete query.



5. Click the Run Query button on the toolbar to delete the records in the OLE_Objects Customers table. Click OK to confirm the deletion of the records.

Tip

You cannot use the IN identifier with Access DDL statements, such as DROP TABLE or CREATE TABLE.

To append the records you deleted back into the Customers table of the OLE Objects database, follow thesdßÀteps:

1. Choose File, Open Database and select your OLE Objects database, which contains the Customers table with no records.



2. Click the Query tab of the Database window, and then click the New button. Close the Show Table dialog without adding a table.

3. Choose View, SQL View and type the following line in the SQL text box:

INSERT INTO Customers SELECT * FROM Customers IN "Northwind.mdb"

INSERT INTO creates an append query. If your OLE Objects database is located elsewhere, add the path to Northwind.mdb to the preceding statement.



4. Click the Run Query button on the toolbar to add the records from Northwind's Customers table. Confirm the append by clicking OK in the message box.

Although you accomplish the same objectives by attaching a table from another database or copying tables to the Clipboard and pasting the table into another database, using a SQL query for this purpose is a more straightforward process.

Oracle Ex. 1009, p. 875
Oracle v. DeBellis
IPR2024-00221

Using the *IN* Clause with Other Types of Databases

You can create or modify xBase and Paradox tables by specifying in an IN statement the path to the file and the file type using the following special Jet SQL syntax reserved for foreign database file types:

```
IN "[drive:\]path" "database_type"
```

The path to the file is required, even if the database is located in your \Access folder; you receive an error if you omit the path entry. You can use an empty string, "", to identify the current folder.

The *database_type* expression must be enclosed in quotation marks. It consists of one of the seven foreign file types supported by ISAM DLLs supplied with Access 97, followed by a semicolon:

- dBASE III;
- dBASE IV;
- dBASE 5;
- FoxPro; (DBF)
- FoxPro 3.0; (DBC)
- Paradox; (DB)

The semicolon after the file type name is required, but the database file type names are not case-sensitive—dbase iii; is acceptable to Access.

You can create a dBASE IV table from a query by using the following statement:

```
SELECT *
   INTO supplier
      IN "c:\dbase\samples" "dBASE IV;"
   FROM Suppliers;
```

You can append records to a dBASE IV file with the following statement:

```
INSERT INTO supplier
      IN "c:\dbase\samples" "dBASE IV;"
   SELECT *
      FROM Suppliers;
```

When deleting and updating records in foreign tables, use the syntax shown in the "Specifying Action Query Syntax" section, with the IN clause added.

Using SQL Statements in Forms, Reports, and Macros

If you create a large number of forms and reports based on queries or that use queries, or if you use macros to run Select and Action queries, the query list in your Database window can become cluttered. You can use SQL queries you write or copy from the SQL dialog in place of the names of query objects and then delete the query from your database. You can use SQL statements for the following purposes:

- *Record Source property of forms and reports.* Substitute the SQL query text for the name of the query in the Record Source text box.

Oracle Ex. 1009, p. 876
Oracle v. DeBellis
IPR2024-00221

- *Row Source property in lists and drop-down combo lists on a form.* Using an SQL statement rather than a query object gives you greater control over the sequence of the columns in your list.
- *Value of the* `SQL` *property of a* `QueryDef` *object or the* `strSource` *argument of the* `OpenRecordset` *method in Access VBA code.* You use SQL statements extensively as property and argument values when programming applications with Access VBA, especially for SQL pass-through queries.
- *Argument of the* `RunSQL()` *macro action.* Only SQL statements that create action queries can be used with the `RunSQL()` macro action. Using macros is discouraged in Access 97.



▶▶ See "Using Access 97's SQL Pass-Through Queries," p. 940

You can create and test your Jet SQL statement in Query Design mode and then copy the statement to the Clipboard. Paste the text into the text box for the property or into your Access VBA module. Then close the test query design without saving it.

Oracle Ex. 1009, p. 877
Oracle v. DeBellis
IPR2024-00221

Chapter 25

Securing Multiuser Network Applications

Personal computer networking continues to be one of the fastest-growing areas in the PC marketplace. Many large organizations are downsizing database applications to personal computer networks. *Downsizing* means moving a database system that runs on a costly, large-scale computer, such as a mainframe, to smaller, lower-priced computers—usually server PCs running network operating systems, such as Windows NT Server 4.0. These networks use a dedicated server PC; PCs connected by the network to the server are called *clients* or *workstations*.

Small- to moderate-size organizations have discovered that installing a simple PC network can increase productivity and reduce the required investment in computing hardware. An organization need purchase only a single laser printer if all users can share it on a network. The cost of purchasing large-fixed-disk drives is reduced when users share large files rather than keep multiple independent copies on local disk drives. Even families with more than one home PC are installing low-cost networks in order to share peripherals and files among family members.

Reducing capital expenditures and operating costs are the principal incentives for installing PC networks. In many cases, savings in the amount spent on peripheral equipment is the sole consideration in the decision to use a network. When you use a database application, however, productivity plays the most important role in the network decision-making process. The capability to share up-to-date information contained in a database file among many users is a strong incentive to install a PC network because it increases productivity. Increasing productivity with a networked database system can, in turn, reduce operating costs by many times the savings offered by the reduction of the investment when you share computer peripheral equipment.

Networking Access Applications

Access 97 is likely to be the first networked Windows application many readers of this book will use. Windows 95's simple installation of low-cost and

Oracle Ex. 1009, p. 878
Oracle v. DeBellis
IPR2024-00221

easy-to-administer Windows Networking should appeal to first-time network users. If you don't have a network now and you plan to create Access applications for other users to share, Windows 95's built-in network operating system is a logical choice as a peer-to-peer "starter" network. A *peer-to-peer network* is a network in which any PC connected to the network is capable of sharing files in all—or designated—folders with any other PC connected to the network. PCs connected by peer-to-peer networks often are called *members of a workgroup*. Access applications that share .mdb files on a network are termed *multiuser applications*.

Microsoft Corporation designed Access specifically for multiuser operation in a networked, workgroup environment. For example, Microsoft added a very sophisticated security system to Access designed for multiuser applications. If your network is already set up, you can choose to install Access on the network or install only the .mdb files you intend to share. (Sharing only .mdb files is the most practical approach.)

If you don't have a network when you begin using Access, the process is simple to change your database files from single-user to shared status when you do install a network. This chapter explains how to set up and use Access in a variety of network environments, share database files, establish database security, and administer a multiuser database system.

Caution

Access 2.0 users should be aware that Access 97 has a new .mdb file structure and uses entries in the Windows 95 or Windows NT Registry, not MSACC??0.INI, to hold the locations and names of workgroup system files.

If you're upgrading to Access 97 from a prior version and plan to share existing .mdb and workgroup system (.mda) files, don't convert the .mdb or .mda files to Access 97 format unless all members of the workgroup have installed and are running Access 97. Access 2.0 and earlier versions cannot open Access 95 or Access 97 .mdb or .mdw files. However, Access 97 can link files created by earlier versions.

Access 97 also can open Access 95 .mdb files, but you cannot change design features of an Access 95 .mdb file unless you convert the file to the Access 97 format.



▶▶ See "Using Access 1.x and 2.0 Data .MDB Files with Access 97," p. 1154

Installing Access in a Networked Environment

If you're using a network operating system with application server capabilities, such as Windows NT Server 3.x/4.0 or Novell NetWare 3.x/4.x, you can use either of the following methods to install Access in a network environment:

- Install Access 97 on the network server. All workstations run the server's copy of Access 97 and don't require a copy of Access on their local disk drives. This

Oracle Ex. 1009, p. 879
Oracle v. DeBellis
IPR2024-00221

approach saves disk space on the workstations but results in much slower operation of Access. If workstations also run Windows 95 from the server, operation can slow down even more. The degree to which the operating speed is affected depends on your network's performance and the number of users accessing the network simultaneously.

- Install a copy of Windows 95 and Access 95 on each workstation. In this case, users share only Access .mdb files. Access requires between 30M and 80M of disk space, depending on the features you include in the workstation installation and whether you previously have installed OLE 2+ server applications such as Microsoft Word 97 and/or Excel 97, or are installing Access 97 as a component of Microsoft Office 97 Professional Edition. Use this installation method for computers connected in a peer-to-peer network.

Tip

If many users need to install or upgrade to Access 97, it is usually faster to use the administrator's installation on a network server and then install Access 97 on the workstations from the administrator's installation, rather than from the distribution disks. (See Acread80.wri and other .wri files on the CD-ROM.)

You need an individual copy of the Access software for each workstation that uses Access or a license for each workstation that runs an Access application with the retail version of Access 95. For additional details, refer to the license information that Microsoft supplies with Access.

The Office Developer Edition (ODE) lets you distribute a run-time version of Access 97. Run-time Access 97 enables users to run applications you create, but not to create or modify applications. Run-time Access enables multiple workstations to run Access applications without an individual license for each workstation.

Unlike Access 1.1 and 2.0, which used a separate run-time executable file (MSARN??0.EXE), Access 97 run-time uses the retail executable (Msaccess.exe) with a setting in the user's Registry that turns off the user's ability to use Msaccess.exe in Design mode. You can install run-time Access on the server or local workstations; installation of run-time Access on each workstation is recommended because you gain operating speed. A run-time Access installation consumes less disk space than the retail version of Access because run-time Access does not include (and cannot be used with) the Access help file and the Access wizards and builders.

Note

Do not attempt to share the retail or run-time version of Access on a peer-to-peer network. Peer-to-peer networks are designed for sharing printers and data files, not the executable (.exe) and help (.hlp) files of large applications such as Access. The computer and network resources required to run Access from a peer server slow applications running on the server to a crawl and greatly increase network traffic.

Oracle Ex. 1009, p. 880
Oracle v. DeBellis
IPR2024-00221

Sharing Your Access Database Files with Other Users

While you're learning how to use Access and designing your first Access applications, you use Access in single-user mode and maintain exclusive use of the database files you create. If your application is designed for use in a networked environment, however, you need to set up a workgroup for the users who will share the database you created. The sections that follow describe how to create a directory for sharing files, modify Access applications for a multiuser environment, and set up a workgroup to provide security for shared .mdb files.



▶▶ See "Examining the Content of the Office Developer's Kit," p. 1135

Tip

If you intend to share your Access applications, make a backup copy of System.mdw, preferably on disk. Use the backup copy to make a new System.mdw file in case your original System.mdw file becomes corrupted.

Creating a Folder and System File for File Sharing

Sharing a database application requires that each user of the database share a common system file, derived from Access' System.mdw workgroup file, that contains information on the members of the workgroup, such as their logon names, passwords, and the groups of which they are members. Permissions for individual users to open and to modify objects are stored in the .mdb file. Permissions are discussed in the section "Maintaining Database Security" later in this chapter.

When you develop an application intended for shared use, it's a common practice to create a new local folder to hold the application's .mdb file(s). You then use the Workgroup Administrator application (Wrkgadm.exe) to create a new system file specifically for the application and develop the application in its own folder. Access 97's default extension for workgroup system files is .mdw. When the application is completed, you can copy the .mdb and .mdw files in this folder to the workgroup folder of the network or peer-to-peer server that is used to share them.

The location and name of the system file that Access 97 uses when it is launched is specified by the SystemDB entry of the `HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Office\8.0\Access\Jet\3.5\Engines` hive of the Registry. For Windows 95, the default Registry entry (key) for SystemDB is your *Windows*\System folder, as shown in Figure 25.1. For Windows NT, the default location is the *WinNT*\System 32 folder.

Oracle Ex. 1009, p. 881
Oracle v. DeBellis
IPR2024-00221

Caution

Unless you're familiar with editing Windows 95's or Windows NT's Registry with the Registry Editor application (RegEdit.exe), don't edit Registry entries manually. An incorrect entry can cause Access or Windows to behave unexpectedly.

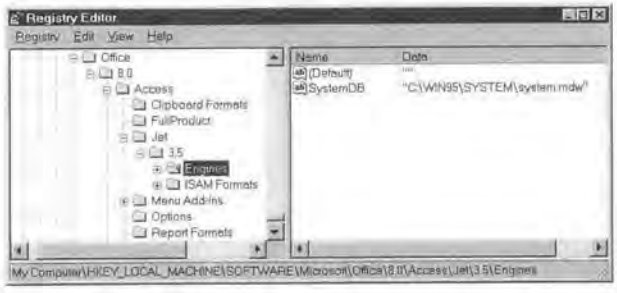


FIG. 25.1 The Windows 95 Registry Editor displaying the Registry value for the SystemDB key that specifies the name and location of your Access 97 system file.

To establish a folder and a workgroup system file for the development of an example shared database application, complete the following steps:

1. Launch Explorer and add a new folder, called \Shared in this example, to the root folder of a local drive on your PC.
2. Open the \Shared folder and create a new subfolder, \Shared\Nwind.
3. Create a copy of Northwind.mdb in \Shared\Nwind, rename the file to **Nwind.mdb**, and then close Explorer. Alternatively, you can use Access's Compact feature to create a compacted copy of Northwind.mdb to Nwind.mdb in the \Shared\Nwind folder.

◀◀ See "Compacting Databases," p. 102

4. Launch Workgroup Administrator (Wrkgadm.exe). Setup does not install Workgroup Administrator in your Start menu hierarchy. Wrkgadm.exe is located in Windows 95's *Windows*\System folder or in Windows NT's *WinNT*\System32 folder. Create a Desktop shortcut to Wrkgadm.exe to make the Workgroup Administrator easily accessible. The opening dialog of Workgroup Administrator displays the name and location of the default system file, System.mdw (see Figure 25.2).

Oracle Ex. 1009, p. 882
Oracle v. DeBellis
IPR2024-00221

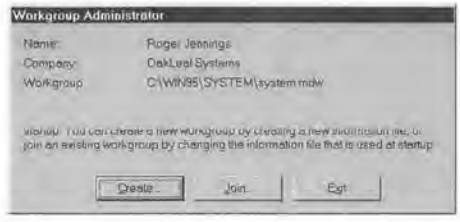


FIG. 25.2 Workgroup Administrator's opening dialog when running under Windows 95.

> **Note**
>
> Previous versions of Access used the ...\Access directory or folder as the default location for the workgroup file (System.mda or System.mdw), as well as Wrkgadm.exe.

5. Click the Create button to open the Workgroup Owner Information dialog in which you specify the Name, Organization, and optional Workgroup ID for the new workgroup system file (see Figure 25.3). The Name and Organization entries default to entries you made when installing Office 97 or Access 97. Write down the Name, Organization, and Workgroup ID entries, which are case-sensitive, and keep them in a safe place. If you need to re-create the workgroup system file in the future, your entries must exactly match the original entries.



FIG. 25.3 Specifying the Name, Organization, and optional Workgroup ID for a new workgroup system file.

6. Click the OK button to display the Workgroup Information File dialog. Type the path and name of your workgroup folder, **\SHARED\NWIND\SYSTEM.mdw** in this case, in the Database text box (see Figure 25.4).

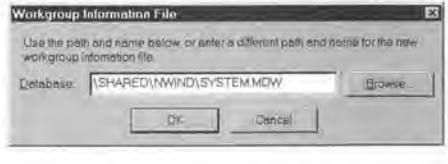


FIG. 25.4 Entering the location and name of the shared workgroup file in the Workgroup Information File dialog.

Oracle Ex. 1009, p. 883
Oracle v. DeBellis
IPR2024-00221

7. Click the OK button to display the Confirm Workgroup Information dialog (see Figure 25.5). If the workgroup information is correct, click the OK button. A message appears to confirm that the new workgroup system file has been created (see Figure 25.6). Click OK to close the message box and return to the initial Workgroup Administrator dialog (see Figure 25.7).



FIG. 25.5 Confirming the owner information for the new workgroup system file.



FIG. 25.6 Access confirming that the new System.mdw workgroup system file is created.



FIG. 25.7 The Workgroup Administrator dialog, displaying the name and location of the new workgroup system file.

8. Click Exit to close Workgroup Administrator. Access doesn't use the new workgroup system file until you close and relaunch Access.
9. Launch Explorer to verify that the copy of Northwind.mdb and System.mdw appear in \Shared\Nwind, as shown in Figure 25.8.
10. Close Access, if it is open. Launch Access and open the \Shared\Nwind\Northwind.mdb database file with the new System.mdw system file active.

The dedicated System.mdw file contains information pertaining only to the database applications that you open when System.mdw is your active system file. You can develop an application using Access' default operating options and System.mdw and then change the options and create a new workgroup system file to provide for file sharing when you complete the application.

Oracle Ex. 1009, p. 884
Oracle v. DeBellis
IPR2024-00221

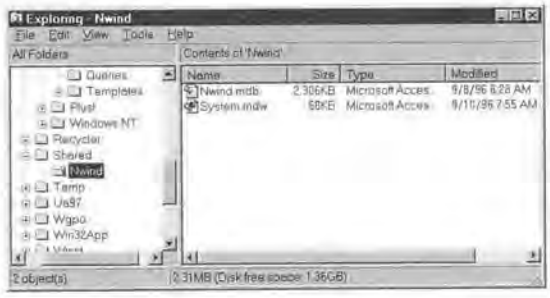


FIG. 25.8 Windows Explorer displaying the files in the \Shared\Nwind folder.

You can change with the Workgroup Administrator the system database file that Access uses when launched. The procedure is described in the section "Choosing Workgroups with the Workgroup Administrator" later in this chapter.

> **Note**
>
> Minimize the number of database objects you create when you develop applications to be shared. You must specifically grant or revoke permissions to groups or individual users for each object you create. Use SQL statements to replace query objects when possible (see Chapter 24, "Working with Structured Query Language"). Minimizing the number of objects also reduces the number of entries you need to make when you establish database security restrictions (permissions) for your application.

Preparing to Share Your Database Files

To set up Access to share database files, you should verify the settings in the Advanced page of the Options properties sheet. You can open the Advanced page by choosing Tools, Options and then clicking the Advanced tab (see Figure 25.9). The changes you make to Access options (called preferences) are stored for your account (Admin, the default) in the system database and apply to all databases you open thereafter using that system database.

Following is a list of the Advanced options that affect multiuser applications:

- *Default Open Mode* set to Shared by default. Changing this setting to Exclusive so that only one user can open files improves performance. If you've changed the file open mode to Exclusive, you must return to the default Shared mode so that more than one user can open the file.
- *Default Record Locking* has three options designed to prevent more than one user from making simultaneous changes to the same record. No Locks, the default, only causes a lock when the edit occurs (*optimistic locking*). All Records locks the entire table when a user opens it for editing (*table locking*). Edited Records locks only the record(s) during the editing process (*pessimistic locking*).

Oracle Ex. 1009, p. 885
Oracle v. DeBellis
IPR2024-00221

- *Number of Update Retries* specifies the number of times Access attempts to update a locked record, at a rate determined by the value in the Update Retry Interval text box, before issuing a message box that the record is locked and cannot be updated.
- *Refresh Interval* determines how often the data displayed in a datasheet or form is rewritten automatically to reflect changes made by other members of the workgroup. *ODBC Refresh Interval* applies only to tables attached using the ODBC driver for tables linked from a foreign database, usually a client/server database.

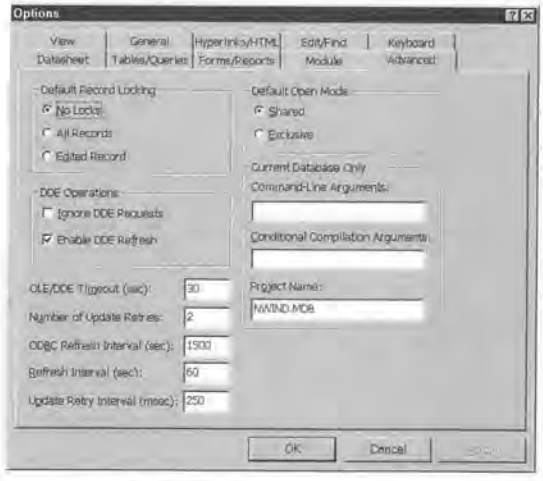


FIG. 25.9 The Advanced page of the Options properties sheet.

Access 97's default values are suitable for most multiuser applications. Unless you have a specific reason for making changes, accept the default values.

Note

Record locking is a misnomer in Access. An Edited Record lock is applied to a 2K page that may contain many records if the fields of the records have small Size property values. Using No Locks, the default, speeds operation of Access in a multiuser environment because the time required to lock and unlock table pages is saved. No Locks is called optimistic locking, because No Locks assumes that the probability is low that two or more users might attempt to alter the same record simultaneously.

If you attempt to update a record in a table with No Locks that is being updated simultaneously by another user, Access displays a message box that lets you choose to accept or overwrite the other user's changes. The conservative approach is to use Edited Record locking, but doing so may impair performance when many users simultaneously update data in a single table.

Splitting Databases for File Sharing

All Access developers agree that Access applications should be divided into two .mdb files: one containing only Access tables (also called *data objects*) and the other containing

Oracle Ex. 1009, p. 886
Oracle v. DeBellis
IPR2024-00221

all other objects (called *application objects*). Splitting Access applications lets you link (attach) tables from the shared .mdb to your application objects in a local .mdb file. Keeping the application objects on the user's computer minimizes network traffic and improves performance, especially on peer-to-peer networks. The major benefit of splitting Access applications, however, is the ability to easily update a user's application .mdb file without affecting current data stored in Jet tables.

Microsoft recognized that most production database applications created with Access use the split design, so Access 97 has an add-in to automate the process. Follow these steps to separate the tables from the application objects of the Northwind.mdb file using the Database Splitter Add-In:

1. Open Nwind.mdb in your \Shared\Nwind folder with \Shared\Nwind\System.mdw as your system database. (Do not use your original copy of Northwind.mdb in ...\Office\Samples for this example.)
2. With the Database window active, choose Tools, Add-Ins, Database Splitter to open the Database Splitter Add-Ins dialog, as shown in Figure 25.10.

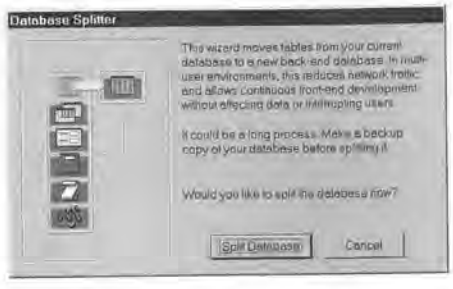


FIG. 25.10 The Database Splitter Add-Ins opening dialog.

3. Verify that you have the copy of Nwind.mdb open, and then click the Split Database button to display the Database Splitter's Save As dialog. The default file name is Nwind_be.mdb; "be" is an abbreviation for *back-end*.
4. Move to the \Shared\Nwind folder and type **NwindData.mdb** as the name of your back-end database file containing tables to link to application objects in Nwind.mdb (see Figure 25.11).
5. Click the Split button to create the back-end database. After a few seconds (depending on your computer's speed) of disk activity, you receive the message shown in Figure 25.12.
6. Click OK to close the Database Splitter Add-In. The tables page of your Database window appears as shown in Figure 25.13. Arrows to the left of the table icons indicate linked files.
7. To verify that the links to the tables are correct, choose Tools, Add-Ins, Linked Table Manager to display the Linked Table Manager's dialog, as shown in Figure 25.14.

Oracle Ex. 1009, p. 887
Oracle v. DeBellis
IPR2024-00221



FIG. 25.11 Selecting the directory and naming the back-end database file to contain the table objects.



FIG. 25.12 The message informing you that the tables are split from your source .mdb file and links to the tables are created.



FIG. 25.13 Newly created links to the tables moved to the back-end database.

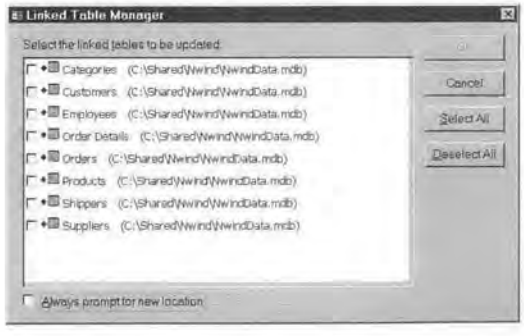


FIG. 25.14 The Linked Table Manager Add-In, confirming the location and name of the back-end .mdb file containing the linked tables.

Oracle Ex. 1009, p. 888
Oracle v. DeBellis
IPR2024-00221

Choosing Workgroups with the Workgroup Administrator

If you have several workgroups that have overlapping user membership, you might want to place all of the workgroup system databases in a single directory, such as \Shared, to make it easy for users to choose a particular workgroup. (The workgroup system database does not have to be in the same directory as the shared .mdb file(s).) To use the Workgroup Administrator to change workgroups, follow these steps:



1. Launch Workgroup Administrator. Your current workgroup is identified by the .mdw file that appears in Workgroup Administrator's opening dialog.
2. Click the Join button to open the Workgroup Information File dialog. Your current workgroup system file appears in the Database text box (see Figure 25.15).

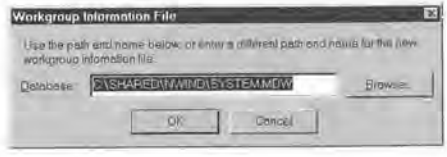


FIG. 25.15 Your current workgroup displayed in the Workgroup Information File dialog.

3. Type the well-formed path and name of the workgroup database file for the workgroup you want to join in the Database text box, or click the Browse button to display the Select Workgroup Information File dialog shown in Figure 25.16. In this case, choose the System.mdw in your \Program Files\Microsoft Office\Office folder to return to the normal Access configuration.



FIG. 25.16 Selecting the workgroup database file for the new workgroup.

4. Select the drive and directory in which the System.mdw file for the new workgroup is located, and click Open to close the Select Workgroup Information File dialog. Your selection appears in the Select Workgroup Information File dialog shown in Figure 25.17.
5. Click OK to confirm your new workgroup selection; click OK when the message box confirms that you have joined the workgroup, and then click the Exit button

Oracle Ex. 1009, p. 889
Oracle v. DeBellis
IPR2024-00221

of the Microsoft Workgroup Administrator's dialog (see Figure 25.18) to complete the process.

6. If Access is running, close it. Restart Access to use the new workgroup system file.

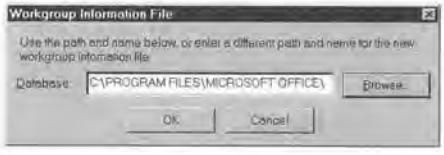


FIG. 25.17 Confirming the workgroup system file for the new workgroup.

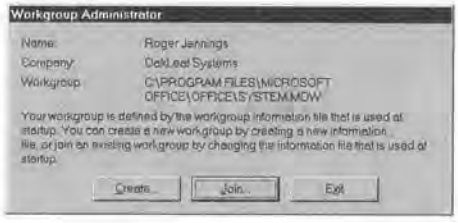


FIG. 25.18 The Workgroup Administrator dialog after changing to a new workgroup.

Using Command-Line Options to Open a Shared Database

Access provides a number of options that you can employ to customize how Access starts for each user. All users in a workgroup may share a common database, but you may want individual users to start Access with a different form. You can open a workgroup database automatically, execute a macro that opens a specific form, and supply a user name or password when you start Access by entering options on the command line that you use to start Access for each workgroup member, as in the following example:

```
d:\msa_path\msaccess.exe [n:\mdb_path\mdb_name.mdb]
 [/User user_name ][/Pwd pass_word ][/X macro_name]
 [/Ro ][/Excl ] [{/Profile user_profile¦/Ini profile.ini}]
 [/Repair] [/Nostartup]
 [/Compact [target.mdb]] [/Convert target.mdb] [/Run-time]
 [/Wrkgrp w:\mdw_path\system.mdw] [/Cmd cmd_value]
```

Spaces separate each of the optional command-line parameters. Table 25.1 describes the elements of the Access startup command-line options.

Table 25.1 Command-Line Options for Launching Access

Command-Line Element	Function
`d:\msa_path\msaccess.exe`	Command to launch Access.
`n:\mdb_path\mdb_name.mdb`	Path and name of startup database file.

(continues)

Oracle Ex. 1009, p. 890
Oracle v. DeBellis
IPR2024-00221

Table 25.1 Continued

Command-Line Element	Function
/User *user_name*	Start with *user_name* as user name.
/Pwd *pass_word*	Start with *pass_word* as password.
/X *macro_name*	Run *macro_name* on startup.
/Ro	Open *mdb_name* for read-only use.
/Excl	Open *mdb_name* for exclusive use.
/Profile *user_profile*	Specify a user profile stored in the Registry named *user_profile*.
/Ini *profile.ini*	Open \Windows*profile*.ini instead of using Registry entries.
/Repair	Repair the database, and then close Access.
/NoStartup	Don't display the Access startup dialog.
/Compact *target*.mdb	Compact into *target*.mdb or into the startup database if you omit *target*.mdb.
/Convert *target*.mdb	Converts a prior version .mdb file to an Access 97 .mdb file with the name specified by *target*.mdb, which is required and cannot be the same name as *mdb_name*.
/Run-time	Starts Msaccess.exe in Run-time mode for testing or to restrict users from entering Design mode.
/Wrkgrp *system*.mdw	Specifies a workgroup system file, *system*.mdw, on startup.
/Cmd *cmd_value*	Specifies a value to be returned by the VBA **Command** function.

Tip

Opening the local application .mdb with the /Excl parameter speeds operation of the application. The /Excl option applies to the local database, not the attached tables of the shared data .mdb file. Do not use the /Excl option if you want other users to be able to share the database. When you omit the /Excl option, shared or exclusive use of databases is determined by the Default Open Mode for Databases choice of the Multiuser Options, discussed in the previous section "Preparing to Share Your Database Files."

Caution

Do not use the /Ro option if you want some members of the workgroup to be able to modify the tables or other objects of the database. If you specify the /Ro option for one workstation, all workstations in the workgroup are restricted to read-only use of the database. Use the permissions features of Access, described in the "Maintaining Database Security" section of this chapter, to designate the users who can update the data in tables and those who cannot.

The sequence in which you enter the command-line options doesn't affect the options' operation; however, a convention is that the name of the file to open always immediately follows the command that launches the application. The /Cmd *cmd_value* entry must be the last command-line option.

Oracle Ex. 1009, p. 891
Oracle v. DeBellis
IPR2024-00221

The `/Profile` parameter replaces the `/Ini` option of the run-time version of Access 2.0, but requires a substantial amount of editing of your Registry to use. The `/Profile` parameter is of limited usefulness, unless you're a full-fledged Access developer with experience editing the Registry. In Access 97, you still can use the argument of Access 2.0's `/Ini` parameter to point to a private profile (.ini) file, such as Nwind.ini, in your \Windows folder. (You can locate the .ini file in another folder by prefixing the well-formed path to the .ini file.) This feature is an alternative to the `/Wrkgrp` option, which is new to Access 97, letting you specify a particular workgroup system file in the `[Options]` section of the .ini file. To create and use a profile with the `/Profile` option, follow these steps:

1. Launch Notepad and type **[Options]** as the first line.
2. Type **SystemDB=** and the well-formed path and name of your workgroup system database as the second line.
3. Choose File, Save and save your text file in your \Windows or \Winnt directory as **Nwind.ini** (see Figure 25.19), and then close Notepad.



FIG. 25.19 Specifying a workgroup database file in a profile (.ini) file.

4. Create a desktop shortcut to Msaccess.exe, and then left-click the shortcut and choose Rename from the pop-up menu. Give the shortcut an appropriate name, in this case **Northwind.**
5. Right-click the shortcut and choose Properties from the pop-up menu. Click the Shortcut tab to display the Shortcut properties page.
6. After ...**Msaccess.exe**, add the path and name of your .mdb file, **/Ini** and the name of your .ini file, and then add **/Nostartup**. For this example, the full command line for the Windows 95 shortcut is **C:\Program Files\Microsoft Office\Office\Msaccess.exe C:\Shared\Nwind\Nwind.mdb /Ini Nwind.ini /Nostartup** (see Figure 25.20).

Oracle Ex. 1009, p. 892
Oracle v. DeBellis
IPR2024-00221

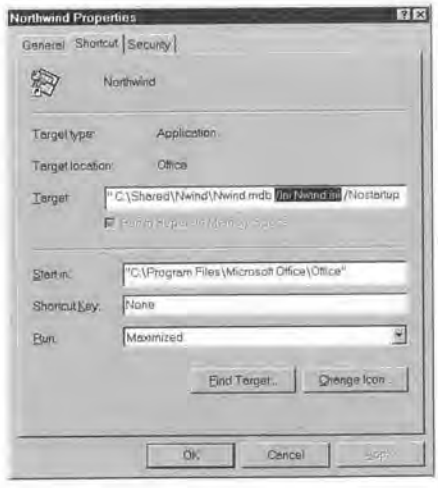


FIG. 25.20 Entering command-line parameters for an Access shortcut in Windows NT 4.0.

> **Note**
>
> If you are using Windows NT 4.0, the name of the executable file must be enclosed within double quotes (`"....exe"`). The full command line for the Windows NT 4.0 shortcut is **"C:\Program Files\Microsoft Office\Office\Msaccess.exe" C:\Shared\Nwind\Nwind.mdb /Ini Nwind.ini /Nostartup**.

7. If you want to change the icon for your shortcut, click the Change Icon button to display a collection of icons in Shell32.dll from which to choose (see Figure 25.21). Select the icon you want, then click OK to close the Change Icon dialog.



FIG. 25.21 Changing the icon for an Access shortcut.

When you double-click the new shortcut, Access launches, uses the workgroup system database specified by the `/Profile` command-line parameter, and automatically opens the file specified on the command line parameter. When you distribute your Access application .mdb to other users, include a copy of the shortcut and the *Profile*.ini file.

Oracle Ex. 1009, p. 893
Oracle v. DeBellis
IPR2024-00221

Caution

Adding a user's password as an option to the startup command line violates one of the basic rules of database security: do not disclose your password to any other person. The preceding example that uses a password as a command-line option does so only for the purpose of completely defining the options available. You should not use the /Pwd command-line option under any circumstances.

Maintaining Database Security

Database security prevents unauthorized persons from accidentally or intentionally viewing, modifying, deleting, or destroying information contained in a database. Database security is of primary concern in a multiuser environment, although you may want to use Access' security features to prevent others from viewing or modifying databases stored on your single-user computer. This section describes the multi-layered security features of networked Access databases and how you use these features to ensure a secure database system.

Specifying the Principles of Database Security on a LAN

Ten basic principles of database security exist for databases installed on a LAN. Five of these principles are associated with the network operating system:

- Each user of a network must be positively identified before the user can gain access to the network. Identification requires a unique user name and secret password for each user. Users must not share their passwords with one another, and all passwords used should be changed every 60 to 90 days.
- Each identified user of the network must be authorized to have access to specific elements of the network, such as server directories, printers, and other shared resources. Each user has a network account that incorporates the user's identification data and authorizations. The network file that contains this information is always encrypted and is accessible only by the network administrator(s).
- Actions of network users should be monitored to determine whether users are attempting to access elements of the network for which they don't have authorization. Users who repeatedly attempt to breach network security should be locked out of the network until appropriate administrative action can be taken.
- The network should be tamper-proof. Tamper-proofing includes installing security systems immune to hacking by ingenious programmers and testing routinely for the presence of viruses.
- Data stored on network servers must be protected against hardware failure and catastrophic destruction (fires, earthquakes, hurricanes, and so on) by adequate and timely backup. Backup systems enable you to reconstruct the data to its state at the time the last backup occurred.

The measures required to establish the first five principles are the responsibility of the network administrator for a server-based system. In a peer-to-peer network, network

Oracle Ex. 1009, p. 894
Oracle v. DeBellis
IPR2024-00221

security measures are the responsibility of each person who shares his or her resources with others. The remaining five principles of database security are determined by the security capabilities of the database management system and the applications you create with it:

- The contents of tables in a database should be encrypted to prevent viewing the data with a file-reading or other snooping utility.
- Users must be further identified before they are allowed to open a database file. A secret password, different from a user's network access password, should be used. The database file that contains user identification and password data (database user accounts) must be encrypted. The encryption technique used should be sophisticated enough to prevent hackers from deciphering it. Only the database administrator(s) has access to this file.
- Users must be assigned specific permission to use the database and the tables it contains. If users are to be restricted from viewing specific columns of a table, access to the table should be in the form of a query that includes only the fields that the user is authorized to view. The RDBMS must provide for revoking specific permissions as the need arises.
- The data in tables should be auditable. Lack of auditability is an incentive to computer-based embezzling. Updates made by users to tables that contain financial data should be maintained in a log—preferably in another database—that identifies the user who made the entry and the date and time the update was made. Logs are useful in reconstructing database entries that occurred between the time the database was last backed up and the time data was restored from the backup copy.
- Operations that update records in more than one table should be accomplished by transaction techniques that can be reversed (or *rolled back*) if updates to all the tables involved cannot be completed immediately.

Most network operating systems in use on PCs provide for the first five database security principles, but enforcement of password secrecy, monitoring of user transgressions, and virus surveillance often are ignored, especially in peer-to-peer networks. Access provides all five of the database security principles, but you must take specific actions to invoke and maintain these principles.

Note

One of the most frequent breaches of database security occurs when a temporary worker is hired to stand in for a user who is ill or on vacation. Instead of establishing a new network account, including a user name, password, and new user (or guest) account for the database, the employee's user names and passwords are divulged to the temporary worker for the sake of expediency. A temporary worker should be assigned his or her own identification for the network and database; the temporary worker's authorizations should be removed when the regular employee returns to the job.

Oracle Ex. 1009, p. 895
Oracle v. DeBellis
IPR2024-00221

Password-Protecting a Single Database

Access 97 adds password protection for individual database files. Setting a database password is the easiest way to partially secure a database while allowing others who don't know the password to use the copy of Access on your PC with other databases. To activate the database password for a specific .mdb file, complete the following steps:

1. You cannot set a database password in shared access mode, so close the open database, and then choose File, Open Database to display the Open dialog. Select your .mdb file, mark the Exclusive check box, and click Open to open the database in exclusive access mode.
2. Choose Tools, Security, Set Database Password to display the Set Database Password dialog.
3. Type a password in the Password text box. Your entry is shown as a series of asterisks to prevent disclosing your password to others as you enter it. Passwords are case-sensitive, so `Uxmal` is a different password from `uxmal`.
4. Type the password in the Verify text box to test your entry (see Figure 25.22). The verification test is not case-sensitive. Click OK.



FIG. 25.22 The Set Database Password dialog, establishing a password required to open a specific database.

5. Close the database, and then reopen it. The Password Required dialog shown in Figure 25.23 appears.



FIG. 25.23 Attempting to open a password-protected database.

6. Enter the password exactly as you typed it in step 3. Press Enter or click OK. If you enter the password correctly, Access continues the startup procedure. If you type an incorrect password, Access displays an error message and won't open the database.

To remove a password protection from a database, open the database for exclusive access, and then choose Tools, Security, Unset Database Password. Type the password in the Password text box of the Unset Database Password dialog, and then click OK.

Oracle Ex. 1009, p. 896
Oracle v. DeBellis
IPR2024-00221

Caution

Don't use database password protection for databases that you intend to replicate using Access 97's Briefcase Replication, described in Chapter 27, "Replicating Access Databases." Database password protection causes the replication process to fail.

Managing Groups and Users

Most client/server databases establish the following three groups of database users:

- *Administrators* (Admins) have the authority to view and update existing tables and add or delete tables and other database objects from the database. Members of the Admins group usually have permission to modify the applications contained in databases.
- *Regular members of workgroups* (Users) are assigned permission to open the database and are granted permission to view and modify databases on a selective basis. Users ordinarily aren't granted permission to modify Access applications. In Access, Admins also are members of the Users group.
- *Occasional users of databases* (Guests) are granted limited rights to use a database and the objects it contains but aren't assigned a user account. Guest privileges often are assigned to persons being trained in the use of a database application. Access 97 does not define a Guests group.

When you install Access 97, you are automatically made a member of the Admins and Users groups with the name Admin and have all permissions granted. You have an empty password and Personal ID Number (PIN), which means that you don't need to enter a password to log onto the database(s) associated with the System.mdw database installed in the ...\Office folder. When you are learning Access, you have little reason to establish database security. After you begin to create a useful database application, especially if it contains confidential information, you should implement basic security provisions on your own computer.

Establishing Your Own Admins Name, Password, and PIN. Access has two levels of security: application level and file level. The *application-level security system* requires each user of Access to enter a user name and a password to start Access.

File level security is established by the network operating system, such as Windows NT 4.0 Server, and grants users permissions for access to shared folders and/or individual files. Establishing single-user application-level security and preparing for multiuser security requires that you perform the following sequence of tasks:

1. Activate the logon procedure for Access. This action requires that you add a password for the Admin user. To remain Admin, you need not complete the remaining steps, but your only security is your password.
2. Create a new account for yourself as a member of the Admins group.
3. Log onto Access using your new Admins user account.

Oracle Ex. 1009, p. 897
Oracle v. DeBellis
IPR2024-00221

4. Remove the default Admin user account from the Admins group. The Admins group should include entries for active database administrators only. You cannot remove the Admin user from the Users group.

Before you begin the following procedure, make a disk backup copy of the System.mdw file in use and any database files that you created or modified while using this System.mdw. If you forget the user name or password you assigned to yourself after deleting the Admin user, you cannot log onto Access. In this case, you must restore the original version of the System.mdw file. Then you may not be able to open the database files with which the original version of the System.mdw file is associated unless you restore the backed-up versions. It is recommended that you modify the System.mdw file you created in the \Shared\Nwind folder as your system database for all of the examples that follow in this chapter.

Caution

Do not use the Northwind.mdb database in your ...\Office\Samples folder for the examples that follow. You should preserve Northwind.mdb and the System.mdw file of your ...\Office\Access folder in the original state. Use the Nwind.mdb file created earlier in this chapter with the Database Splitter and use the System.mdw workgroup file located in the \Shared\Nwind directory.

Tip

You don't need to open a database to add or modify user accounts. All user account information is stored in System.mdw, which Access automatically opens when launched.

To activate the logon procedure for Access, complete the following steps:

1. A temporary password to the Admin user is necessary to activate Access's logon procedure. Choose Tools, Security, User and Group Accounts to open the User and Group Accounts properties sheet (see Figure 25.24). You are logged on as Admin, a member of the Admins and Users group, by default.

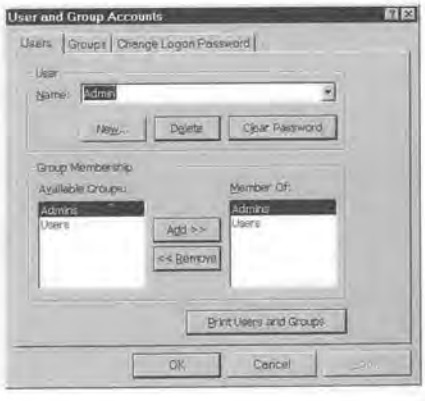


FIG. 25.24 The default opening page, Users, of the User and Group Accounts properties sheet.

Oracle Ex. 1009, p. 898
Oracle v. DeBellis
IPR2024-00221

2. Click the Change Logon Password tab to display the Change Logon Password page.

> **Note**
>
> If you don't change the Admin user's password, you automatically are logged on as Admin with a blank password each time you start Access.

3. Press the Tab key to bypass the Old Password text box (this enters the equivalent of an empty password), and enter a difficult-to-guess password, such as **Xy8zW3ab**, in the New Password text box. Your entry is shown as a series of asterisks to prevent disclosing your password to others as you enter it. Passwords are case-sensitive, so `Xy8zW3ab` is a different password from `xy8zw3ab`.

4. Type the password in the Verify text box to test your entry, as shown in Figure 25.25. The verification test is not case-sensitive. Click OK to close the properties sheet.

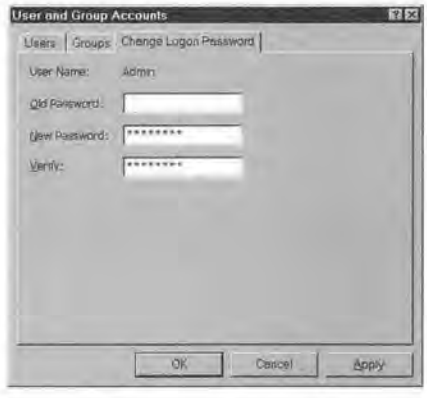


FIG. 25.25 The Change Logon Password page used to establish your new Admin password.

5. Exit Access and launch it again. The Admin account is password-protected, so the Logon dialog appears.

6. Type **admin** in the Name text box, press Tab, and type the password exactly as you typed it in step 3 (see Figure 25.26). Press Enter or click OK. If you enter the password correctly, Access continues the startup procedure.



FIG. 25.26 The Logon dialog that appears when you protect the Admin account with a password.

Oracle Ex. 1009, p. 899
Oracle v. DeBellis
IPR2024-00221

To add your new user account in the Admins group, perform the following steps:

1. Choose Tools, Security, User and Group Accounts to display the User and Group Accounts properties sheet (refer to Figure 25.24). All members of the Admins group automatically are included (and must be included) in the Users group. Both Admins and Users appear in the Member Of list.
2. Click the New button to add your new account. The New User/Group dialog appears.
3. Type the name you want to use to identify yourself to Access in the Name text box and enter a four-digit PIN in the Personal ID text box (see Figure 25.27). The PIN, with the Name entry, uniquely identifies your account. This precaution is necessary because two people may use the same logon name; the Name and PIN values are combined to create a no-duplicates index on the Users table in your current system database file. Click OK to close the New User/Group dialog and return to the User and Group Accounts page.



FIG. 25.27 Adding a new account for the Admins group with the New User/Group dialog.

4. This is a critical step. Select Admins in the Available Groups list and click the Add button to add the Admins group to your new user name (see Figure 25.28). If you fail to do this, you cannot remove the Admin account from the Admins group. (Access requires that there be at least one member of the Admins group in each system database file.)

> **Note**
>
> When you log on with your new user name, you can't see the names of the last four databases you opened as Admin when you choose File, Open. Prior database selections are specific to each user.

Oracle Ex. 1009, p. 900
Oracle v. DeBellis
IPR2024-00221



FIG. 25.28 Adding the Admins group to your new user account.

5. You don't enter a password for the new user at this time because you still are logged onto Access as Admin. Click the OK button to close the User and Group Accounts properties sheet and then exit Access.

6. Launch Access, type your new user name in the Logon dialog, and press Enter or click OK. Do not enter a password because you have an empty password at this point. User names aren't case-sensitive; Access considers NewAdmin and newadmin to be the same user.

7. Choose Tools, Security, User and Group Accounts, select your new user name from the Name drop-down list, and click the Change Logon Password tab. Press Tab to bypass the Old Password text box and type the password you plan to use until it is time to change your password (to maintain system security). Passwords can be up to 14 characters long and can contain any character, except ASCII character 0, the Null character. Verify your password, and then press Enter or click OK to close the Password sheet.

8. Close and reopen Access and log on with your new user name and password. This step verifies that your new Admins user name and password are valid.

9. Choose Tools, Security, User and Group Accounts. Open the Users list of the User and Group Accounts page and select your new user name from the list. Verify that you are a member of the Admins and Users group.

10. Open the Users list again and select the Admin user. Select Admins in the Member Of list; then click Remove. Admin remains a member of the Users group, as shown in Figure 25.29. Click OK to close the properties sheet.

You use the same procedure to add other users as members of the default Admins, Users, or Guests group or of new workgroups you create. You have not fully secured the open database because the Admin user still has full permissions for the objects in the database. Revoking the Admin user's permissions is discussed in the section "Changing the Ownership of Database Objects" later in the chapter.

Oracle Ex. 1009, p. 901
Oracle v. DeBellis
IPR2024-00221

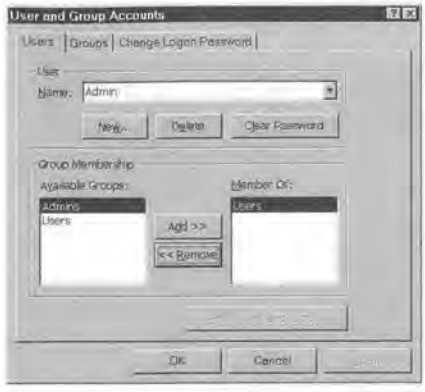


FIG. 25.29 Removing the Admin user from the Admins group.

> **Note**
>
> Write down and save your PIN and the PIN of every user you add to the workgroup for future reference. User names and PINs aren't secure elements, so you can safely keep a list without compromising system security. This list should be accessible only to database administrators. You need a user's PIN so that the user can be recognized as a member of another workgroup when the need arises. (See the section "Granting Permissions for a Database in Another Workgroup" near the end of this chapter.)

Establishing Members of Access Groups. Groups within Access's security system are not the same as workgroups. As discussed previously, a workgroup shares the same system database file that is located in a designated directory. The entries you made in the preceding steps were saved in the workgroup or system database file that was active when you launched Access. This section describes how to add new users to a group, a process similar to the one you used to add your new Admins account.

To add a new user to a group, you must be logged onto Access as a member of the Admins group and complete the following steps:

1. Choose Tools, Security, User and Group Accounts to open the User and Group Accounts dialog.
2. With the Users page active, click New. The New User/Group dialog appears. Type the new user's name and PIN. Click OK to create the account and close the dialog. The Users dialog reappears. Make a note of the PIN you used to add the new user. You need to know the user's PIN so that you can duplicate an entry for the new user in other workgroups.
3. The default group for all new users is Users. To add the user to the Admins group, select Admins in the Available Groups list and click the Add button to add Admins to the Member Of list (refer to Figure 25.24). All Access users must be members of the Users group. Click OK to return to Access's main window when your selections are complete.

Oracle Ex. 1009, p. 902
Oracle v. DeBellis
IPR2024-00221

4. Request the new user log onto Access with the user name and change his or her password from the default empty value to a legitimate password.

   You can improve the level of security by typing the new user's password yourself, so that users cannot bypass the password step by leaving their password blank. To enter a password for a new user, close Access, log on as the new user, and enter the user's chosen password in the Change Logon Password page.

Before you add a significant number of users, decide whether you need additional groups and determine the permissions that should be assigned to each group other than Admins. These aspects of database security are discussed in the following sections.

Tip

When requesting new users enter their first password, emphasize the advantage of the use of longer passwords that combine upper- and lowercase characters and numbers because they improve system security. Users should not use their initials, names of spouses or children, birth dates, or nicknames; these are the entries that unauthorized users try first to gain access to the system.

Adding a New Group. In most cases, Admins and Users are the only groups necessary for each workgroup you create. Members of a group usually share the same permissions to use database objects (which is the subject of the next section). Adding a new Access group is not necessary, therefore, unless you have a category of users who are to have a different set of permissions than members of the Users group. Such a category may distinguish Users (who may be limited to viewing data) from members of a Data Entry group who have permission to update the data in tables.

To add a new group, perform the following steps:

1. Choose Tools, Security, User and Group Accounts. Then click the Groups tab. Currently defined groups appear in the Name drop-down list, as shown in Figure 25.30.



FIG. 25.30 The Groups page for adding a new user group to a database.

Oracle Ex. 1009, p. 903
Oracle v. DeBellis
IPR2024-00221

2. Click the New button to open the New User/Group dialog.
3. Type the name of the group in the Name dialog and a four-digit Personal ID Number, as shown in Figure 25.31. Group names can be up to 20 characters long and can contain spaces, but punctuation symbols aren't allowed. You don't need to make a note of the PIN in the case of groups because the PIN is used only for indexing purposes.



FIG. 25.31 Adding a new Data Entry group to the workgroup.

4. Press Enter or click OK. The Groups dialog reappears.
5. Click OK from the Groups dialog. You can delete the newly added group by clicking the Delete button.

After you add a new group, you need to assign the default permissions that apply to all members of the group by the procedure outlined in the "Granting and Revoking Permissions for Database Objects" section that follows.

Deleting Users and Groups. Members of the Admins group have the authority to delete users from any group and to delete any group except Admins, Users, and Guests. To delete a user or group, choose Tools, Security, User and Group Accounts, select the user or group to delete from the list box, and click Delete. You are asked to confirm the deletion. Admins and Users groups must each contain one user account; you cannot delete all users for either of these groups.

Clearing Forgotten Passwords. If a user forgets his or her password and you are logged into Access as a member of the Admins group, you can delete the user's password, so that you or the user can enter a new password.

To clear a user's password, complete the following steps:

1. Choose Tools, Security, User and Group Accounts. Make sure the Users tab is active.
2. Open the Name list and select the user whose password you want to clear.

Oracle Ex. 1009, p. 904
Oracle v. DeBellis
IPR2024-00221

3. Click the Clear Password button (refer to Figure 25.29).
4. Make sure that the user whose password you cleared enters a new password, or log onto Access as the new user and enter a new password for the user.

As mentioned previously, entering the user's password as the database administrator is the only means of ensuring that the database security is enforced. There is no other means of ensuring that users assign themselves passwords. (Of course, perverse users can change their passwords to empty strings or "password" if they choose to do so.)

Understanding Database Object Ownership

The user who creates an object becomes the owner of the object. (Access calls the owner of an object the object's *creator*.) Object owners have special status within the Access security system. The following two sections briefly describe owners' permissions and how to change the ownership of database objects. A more detailed description of object ownership is contained in the file SECURE.ZIP, written for Access 2.0, that you can download from the **MSACCESS** forum on CompuServe.

Owner Permissions for Objects

The owner of an object has full (Administer) permissions for the object. No other user, including members of the Admins group, can alter the object owner's permissions for the object directly. For example, the Admin user is the owner of all the database objects in Northwind.mdb. Thus, anyone who uses the Admin user account has full permissions for all objects in Northwind.mdb. This is one of the reasons for assigning a password for the Admin account.

When a user other than the object's creator adds a new object to the database or to one of the existing objects in the database, this user becomes the owner of the object. For example, if user Margaret adds a control object to a form created by Larry, Margaret is the owner of the control object, not Larry. Mixed ownership of objects can lead to bizarre situations, such as the inability of the owner of a query to execute the query because the owner of the underlying tables has changed. (You can overcome this problem, however, by adding the `WITH OWNERACCESS OPTION` to the Jet SQL statement for the query.)

When you create new database objects using the default Admin user ID, anyone else who has a retail copy of Access 97 and uses the default Admin user ID also has full permissions for these objects. Thus, when you begin development of an application that you intend to share with others or that you want to prevent others from using or modifying, create a new account in the Admins group as described earlier in the chapter. Use your new Admins account when you create new applications.

Changing the Ownership of Database Objects

Following are the three methods of changing the ownership of existing Access database objects:

- Create a new database file, and then choose File, Get External Data, Import. Open the .mdb file containing the objects, and import all of the objects into the new

Oracle Ex. 1009, p. 905
Oracle v. DeBellis
IPR2024-00221

.mdb file. The user who creates the new .mdb file becomes the owner of the imported objects.

- Use the Change Owner page of the User and Group Permissions dialog.
- Use the Security Wizard to create a new secure database file, import the objects, and then encrypt the new database.

The following two sections describe the second and third methods for changing database object ownership.

Using Access 97's Change Owner Feature. To use the Change Owner feature that originated in Access 95, you must be a member of the Admins group for the database, and must follow these steps:

1. Open the database containing the objects whose ownership you want to change.
2. Choose Tools, Security, User and Group Permissions to open the User and Group Permissions dialog, and then click the Change Owner tab.
3. Choose the class of object you want to change in the Object Type drop-down list.
4. If you want to change the ownership of all of the objects of the selected class, select the first item in the Object list, move to the bottom of the list, press the Shift key, and click the last item of the list.
5. Select the new owner's name from the New Owner drop-down list.
6. Click the Change Owner button to change the ownership of the selected items, from Admin to **RogerJ** in this example (see Figure 25.32).

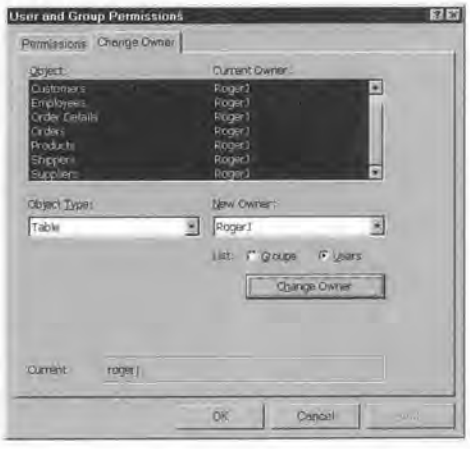


FIG. 25.32 Changing the ownership of all Table objects from Admin to a new owner.

7. Repeat steps 3–6 for each class of objects whose ownership you want to change.

Oracle Ex. 1009, p. 906
Oracle v. DeBellis
IPR2024-00221

The preceding process is the fastest way to remove permissions of the Admin user accrued from ownership of the original objects.

Using the Security Wizard. You can change the ownership of all the objects in a database for which you have Administer permissions by importing all the database objects into a new database you create with a user ID other than Admin. Access 2.0 made it easy to import all the database objects from one .mdb file into another .mdb file with its Import Database Add-In. Access 97's Security Wizard goes the Import Database Add-In one better by letting you choose the database objects to secure and encrypting the new secure copy of the database in a single (long) step. The Security Wizard automatically imports every object in the source database into the new encrypted destination database.

Caution

Do not use the Security Wizard with the Northwind.mdb database in your ...\Office\Samples directory. Use the NwindData.mdb file created earlier in this chapter with the Database Splitter, and use the System.mdw file in your \Shared\Nwind directory. The Security Wizard can take a long time to perform its operations with a slow processor, so using the smaller NwindData.mdb file is recommended.

To test the Security Wizard, follow these steps:

1. If you aren't logged onto Access, launch Access and log on with your new user ID that includes Admins group membership and open the database to secure, NwindData.mdb in this case.
2. Choose Tools, Security, User-Level Security Wizard to display the Security Wizard's opening dialog.
3. Clear the check box that corresponds to the class of database objects that you don't want to make secure (see Figure 25.33). If you want to secure all database objects, accept the wizard's default.



FIG. 25.33 Setting the types of objects to secure in the new database.

4. Click the OK button to open the Destination Database dialog. Select the folder in which to store the new secure database file and give the file a new name. The default is Secure *database*.mdb (see Figure 25.34).

Oracle Ex. 1009, p. 907
Oracle v. DeBellis
IPR2024-00221



FIG. 25.34 Specifying the path and file name for the new secure database file.

5. Click the Save button to put the Security Wizard to work. After a few seconds, the message shown in Figure 25.35 appears, indicating successful creation of the new secure database. Click OK to close the dialog.



FIG. 25.35 The message indicating successful completion of the Security Wizard's task.

The owner of all the objects in the new database is the user ID you used when you opened the source database. Only members of the Admins group have access of any kind to the newly secured database.

Note

This chapter uses the term *user ID* to identify users of Access. Internally, Access uses a *system ID* (*SID*) to identify users. The SID is a value that Access computes from the user ID, password, and PIN. The SID is stored in the MSysUsers table of System.mda as an encrypted binary value in a field of the Binary (varbinary) data type.

Granting and Revoking Permissions for Database Objects

The second layer of Access security is at the *database level*. Access lets the database administrator grant or revoke permissions to use specific database objects to all members of a group or to specific members of a group. Permissions grant authority for users to view or alter specific database objects. The permissions granted to the group are inherited by

Oracle Ex. 1009, p. 908
Oracle v. DeBellis
IPR2024-00221

each member as he or she is added to the group. Thus, it is important that you establish the group permissions you want before adding users to a group.

Users who are members of more than one group, such as Admins and Users, inherit database object permissions from each group. You can grant additional permissions to individual members of a group, but you cannot revoke permissions that individual members inherit from the group. Permissions are stored within the database file as properties of individual database objects. Only members of the Admins group or users who have Administer permission can grant or revoke permissions for database objects.

Caution

If you use the split-database design, you do not need to add groups and users to the data .mdb. Permissions for use of the linked data .mdb file are managed by the permissions for links to the tables in your application .mdb file. However, it is important that you change the ownership of Table objects in the data .mdb file from the Admin account to your secure account in Admins. If you do not change ownership, any user of retail Access who uses the default Admin account can open the data .mdb file and make changes at will to the file.

Table 25.2 lists the permissions offered by Access for database objects, ranked in descending level of authority. Full Permissions allow the user to use all the features of Access, including design functions. The description of the specific action allowed by a permission is listed in the Explicit Permissions column. Permissions at an authority level below Full Permissions require other permissions to operate; these required permissions are called implicit permissions.

Table 25.2 Permissions to Use Access Database Objects

Permission	Database Objects	Explicit Permissions	Implicit Permissions
Open/Run	Forms, reports, macros	Use or run objects	Read Data
Read Design	All	View objects	Execute for macros only
Modify Design	All	Alter, replace, or delete objects	Update Data and Execute
Administer	All database objects	All permissions	Not applicable
Read Data	Tables, queries, forms	View data in objects	Read Design
Update Data	Tables, queries, forms	Edit table data	Read Data
Insert Data	Tables, queries, forms	Append data in tables	Read Data
Delete Data	Tables, queries, forms	Delete data in tables	Read Data

If, for example, you allow a user to modify design, this user also must be able to modify data and execute objects. Therefore, Update Data and Open/Run permissions are implied by the Modify Design permission. This user, and any other users allowed to modify data, must be able to read data. All users having permission to read data must be able to read designs. When you establish permissions for a database object, Access adds the implicit permissions automatically.

Oracle Ex. 1009, p. 909
Oracle v. DeBellis
IPR2024-00221

The Admins and Users groups have full permissions for any new database objects you create. If you intend to share the database with other users, you probably don't want all members of the Users group to have permission to update database tables, and certainly you don't want members of the Users Group to be able to modify the design of your database objects. A more conservative set of permissions for the two groups follows:

- The Admins group has full permissions for all objects. Admins privileges should be assigned to as few individuals as possible. Make sure you have enough backup database administrators with Admins privileges to cover for the absence of the primary administrator. Members of the Admins group are also members of the Users group.
- The Users group has Open/Run and Read Data permissions. Update, Insert, and Delete Data permissions are granted for specific forms and reports. Users are never granted Modify Design permission in databases.

You can add new groups with specific group permissions, such as Data Entry or Developers, to make assigning individual user permissions for database objects simpler.

Note

You can use the Run with Owner's Permissions check box or add the `WITH OWNERACCESS` option to Jet SQL statements to enable users without the required permissions to execute a query.

Altering Group Permissions

After you design your hierarchy of permissions and add any new user groups you need, you are ready to assign group permissions for each of the objects in your database. Only members of the Admins group can alter permissions for Groups or Users. The Permissions check boxes that are enabled depend on the type of object you choose. Open/Run, for example, is enabled only for database, form, report, and macro objects.

Note

When you first select the Admins group, none of the Admins group's permission check boxes are marked. Members of the Admins group inherit full object permissions from membership in the Users group. (By default, Users have full permissions for all objects.) If you revoke object permissions for the Users group, you add Administer permissions for the Admins group.

To change the permissions for a group, complete the following steps:

1. Open the database for which group permissions are to be granted or revoked with the appropriate workgroup system database active.
2. Choose Tools, Security, User and Group Permissions to open the User and Group Permissions dialog.
3. Click the Groups option button to display the permissions for groups of users; then select Users in the User/Group Name list.

Oracle Ex. 1009, p. 910
Oracle v. DeBellis
IPR2024-00221

4. Open the Object Type drop-down list and select the type of database object whose permissions you want to change.
5. Select the specific object to which the new permissions will apply in the Object Name list. To select all objects, click the first item in the list, press the Shift key, and then click the last item in the list. Do not include `<New ObjectType>` in your multiple selection.
6. In the User/Group Names list, select the Group whose permissions you want to revise, Users for this example. Figure 25.36 shows the full permissions for Table objects that Access assigns by default to the Users group.

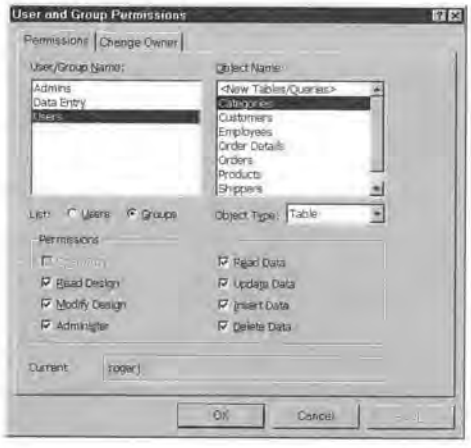


FIG. 25.36 The default permissions for the Users group.

7. Permissions currently granted to the group are shown by a check mark in the Permissions check boxes. Click the Modify Design, Update Data, Insert Data, and Delete Data check boxes to allow the users groups only to display forms and read table data. If you have made multiple selections, you may need to click the check box twice to make a selection effective. When you remove a permission, Access automatically removes the Administer permission. Your Permissions dialog appears as shown in Figure 25.37.
8. Click the Apply button to make the new permissions effective for the selected database object(s).
9. With the object(s) selected in step 5, select the Admins group, mark the Administer check box, and click the Apply button. This step assures that the Admin users continue to have full permissions for the objects (see Figure 25.38).
10. Repeat steps 4–9 for each database object and object type whose User group permissions you want to change.

Oracle Ex. 1009, p. 911
Oracle v. DeBellis
IPR2024-00221

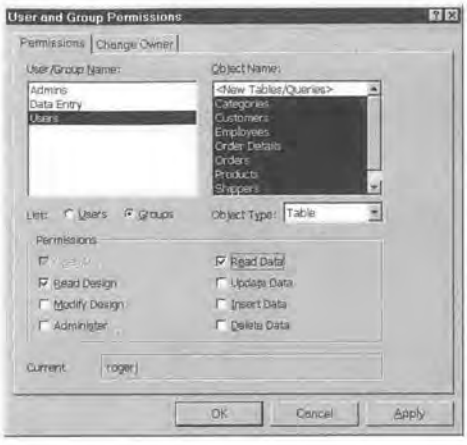


FIG. 25.37 Revising permissions for all Table objects for the Users group.



FIG. 25.38 Granting Administer permissions for all Table objects to the Admins group.

Note

If you create macro objects that contain several individually named macros (to minimize the number of objects in the database), make sure that each macro object contains named macros that correspond to a specific category of permissions. Named macros, for example, that invoke action queries to modify tables or add new records should be grouped in one macro object, and named macros that only display the contents of database objects should be located in a different macro object.

When you assign permissions to execute macro objects, you need to assign Modify Data permission to those users who can execute macro objects that run action queries. Microsoft discourages the use of macros with Access 97, and macros may not be supported in future versions of Access.

Oracle Ex. 1009, p. 912
Oracle v. DeBellis
IPR2024-00221

Granting Additional Permissions to Specific Users

The process of granting additional permissions to a specific user is similar to the process used to alter group permissions. Permissions inherited by the user from the group to which the user is assigned are not shown in the Permissions dialog. To grant additional permissions to a specific user, complete the following steps:

1. Choose Tools, Security, User and Group Permissions. The Users option is the default for the User and Group Permissions dialog.
2. Select the user to whom additional permissions are to be granted in the User/Group Name list (see Figure 25.39).

   TestUser is a member of the Users group whose account was added after the changes to User group permissions were made in the preceding section. As mentioned in the introduction to this section, the Read Design and Read Data permissions that were inherited by TestUser from the modified permissions of the Users group aren't shown in the Permissions check boxes.

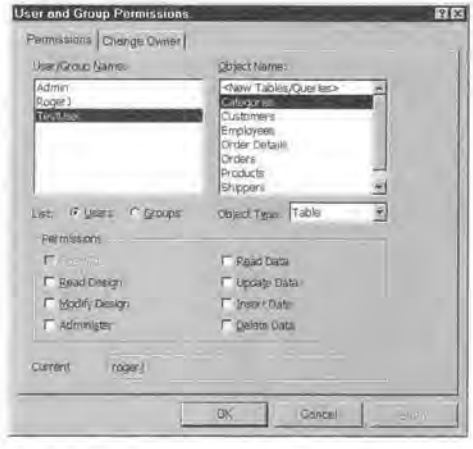


FIG. 25.39 The Permissions dialog for a new user with inherited permissions.

3. To assign permissions to a specific user so that the user can update data for an object, select the object using the Object Type and Object Name lists, and then click the Update Data, Insert Data, and Delete Data check boxes. Access automatically marks the implicit permissions, Read Design and Read Data, associated with the explicit permission, Update Data (see Figure 25.40). Click the Apply button after selecting each object whose permissions you want to change.

   Implicit permissions for individual users are displayed in the Permissions check boxes regardless of whether the implicit permissions also were inherited from group membership.

Oracle Ex. 1009, p. 913
Oracle v. DeBellis
IPR2024-00221

4. Repeat step 3 for each user who requires permissions for an object that aren't inherited from the user's group permissions. Click the OK button when you complete the permission changes for all users who require such changes.

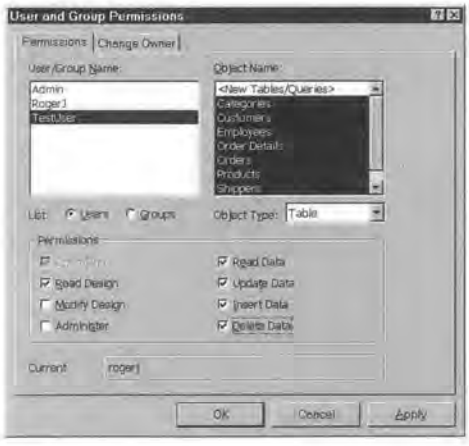


FIG. 25.40 The Permissions dialog for a new user with the new data permissions added.

Note

You can use the /Ro command-line option, described in the preceding section, "Using Command-Line Options to Open a Shared Database," to revoke Update Data, Insert Data, and Delete Data permissions for the database on specific workstations. This method isn't secure because the user can edit the command-line option, remove the /Ro entry, and log on again with read-write privileges. If you use this method, Access displays a message box indicating that the database is being opened in read-only mode and that the user cannot modify data.

Granting Permissions for a Database in Another Workgroup

If your application requires that you attach a table in a secure database used by a different workgroup, the user needs to be a member of a group in the other workgroup and needs to be assigned appropriate permissions for the attached table. At this point, you need the list of PINs for users, mentioned in the section "Establishing Your Own Admins Name, Password, and PIN" earlier in this chapter.

To grant permission for a user to modify data in a table attached from another workgroup's database, perform the following steps:

1. Close Access; you need to relaunch Access when you select another workgroup.
2. Launch the Workgroup Administrator application and specify the path to the workgroup database file of the workgroup that uses the database that contains the table to be attached.



Oracle Ex. 1009, p. 914
Oracle v. DeBellis
IPR2024-00221



3. Launch Access and open the database that contains the table to be attached.
4. Add an account for the user to the Users group with exactly the same user name and PIN as was used to add the user account to his or her workgroup.
5. If you don't want this user to join the other workgroup, enter a password and don't disclose the password to the user.
6. Open the Permissions properties sheet, select the table object to be attached, and assign the appropriate data permission for the table to the user.

You need to use the same PIN for the user in both workgroups because the account for the user is created from the user name and PIN, and the accounts must be identical in both databases. You also must use the same PIN number to reinstate the user's account if the workgroup system file becomes corrupted; you don't have a current backup, and Access cannot repair it.

Sharing Databases on the Network

Once you've set up your user groups and modified the database object permissions for the groups as necessary, you can safely share the workgroup system database and your data .mdb file, and then distribute copies of your application .mdb file to the users. Before sharing the files, make sure to create a backup copy of each of the shared files and store the copies in a safe location. The following sections describe how to share files.

Sharing Database Files on a Windows 95 Network

With a peer-to-peer Windows 95 network, you need only set up a network share of the folder in which you developed the application, \Shared\Nwind for the example of this chapter. The most common method of sharing folders with Windows 95 networking is to use share-level security. To share a folder on your computer with share-level security, follow these steps:



1. Launch Control Panel and double-click the Network tool to open the Network properties sheet.
2. Click the Access Control tab and select the Share-Level Access Control option (see Figure 25.41). Click OK to close the Network properties sheet, then close Control Panel.

> **Note**
>
> If you previously have specified the User-Level Access Control option and change to Share-Level Access Control, all of your current shares will be removed, and you must reboot Windows 95 for the new access control option to take effect.



3. Launch Explorer and select the folder to be shared, \Shared\Nwind for this example.
4. Choose File, Properties or right-click the folder icon and choose Sharing to open the *FileName* Properties sheet. Click the tab of the Sharing properties page.

Oracle Ex. 1009, p. 915
Oracle v. DeBellis
IPR2024-00221

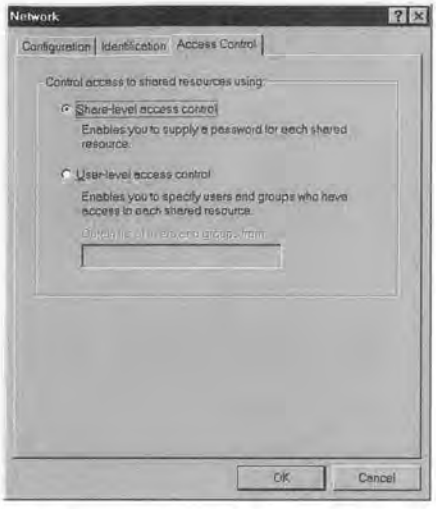


FIG. 25.41 Setting share-level access control in Control Panel's Network properties sheet.

5. Click the Shared As option button. The name of the folder appears as the default share name, NWIND in this example. Share names are limited to 12 characters and must not contain names, punctuation, or other special characters. Add an optional brief description of the share in the Comment text box.

6. To update data, users must have read-write access to the shared folder (often simply called a share). Other users only need read access to the folder. Click the Depends on Password option button in the Access Type frame.

7. Type the Read-Only Password and the Full Access Password in the two text boxes, as shown in Figure 25.41, and then click OK to create the share.



FIG. 25.42 Creating a shared subfolder with share-level access for a peer-to-peer Windows 95 network server.

Oracle Ex. 1009, p. 916
Oracle v. DeBellis
IPR2024-00221

In the examples presented in this chapter, both Nwind.mdb (the application database) and NWData.mdb (the data .mdb file) are shared for read-write access, allowing users to launch Nwind.mdb from the server if they choose. Users should copy Nwind.mdb from the server share to their local computer and run the local copy to minimize network traffic. When connecting to the server share, users must enter the appropriate password for full or read-only access.

Sharing Files with User-Level Security

Specifying individual users with access to shares is a more secure alternative to simple share-based access control. You can specify users that are members of the Windows 95 workgroup to which you log on. If you're connected to a Windows NT network, you can grant share access to designated Windows NT groups or individual users. To set up user-level security, follow these steps:



1. Launch Control Panel and double-click the Network tool to open the Network properties sheet.
2. Click the Access Control tab and select the User-Level Access Control option (refer to Figure 25.41). Type your workgroup name or the Windows NT domain name in the Obtain List of Users and Groups From text box. Click OK to close the Network properties sheet, then close Control Panel.

> **Note**
>
> As noted in the preceding section, if you previously have specified the Share-Level Access Control option and change to User-Level Access Control, all of your current shares will be removed and you must reboot Windows 95 for the new Access Control option to take effect.



3. Launch Explorer and select the folder to be shared, \Shared\Nwind for this example.
4. Choose File, Properties or right-click the folder icon and choose Sharing to open the *ShareName* Properties sheet. Click the tab of the Sharing properties page. The Sharing page contains an empty list of names and access rights.
5. Click the Shared As option button. Add an optional brief description of the share in the Comment text box (see Figure 25.43).
6. To specify groups or users having access to the share, click the Add button to open the Add Users dialog. Members of your workgroup appear in the left-hand list if you're using Windows 95 networking. If you're connected to a Windows NT Server domain, which is the case for this example, all of the Windows NT groups and users appear in the list.
7. Select a group or user, then click the Read Only, Full Access, or Custom button, as appropriate for your selection. Ordinarily, only you and Domain Admins should be granted Full Access to the share (see Figure 25.44).

Oracle Ex. 1009, p. 917
Oracle v. DeBellis
IPR2024-00221



FIG. 25.43 The Sharing properties page for user-level access control.

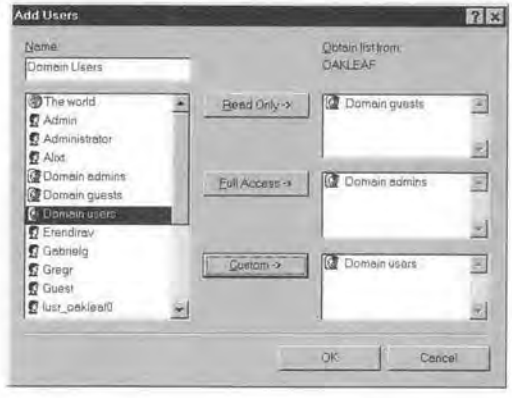


FIG. 25.44 Adding Windows NT Server groups for Read Only, Full, and Custom access to the share.

8. When you add a user or group with Custom access to the share, the Change Access Rights dialog appears. Mark the check boxes for the specific rights you want to grant, as shown for Domain Users in Figure 25.45, then click OK to close the dialog.

9. When you finish adding groups or users, click OK to close the Add Users dialog. The groups (and users, if any) you added for the share appear in the Sharing page, as shown in Figure 25.46. Click OK to close the *ShareName* Properties sheet.

Oracle Ex. 1009, p. 918
Oracle v. DeBellis
IPR2024-00221



FIG. 25.45 Specifying custom share access rights for a user or group.

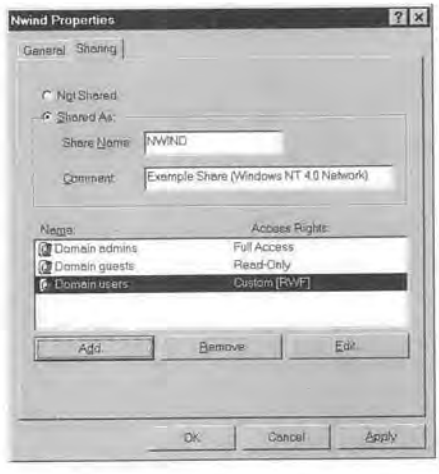


FIG. 25.46 Three Windows NT groups with varying rights to a Windows 95 share.

The advantage to user-level access is that users don't need to enter a password when connecting to the share; the user ID and password used to log onto the network grants access to the share.

> **Note**
>
> It is a relatively uncommon practice to share files from a Windows 95 peer-to-peer server when workgroup members are connected to a Windows NT server. Sharing the files from a Windows NT server that uses the Windows NT File System (NTFS), the subject of the next section, offers improved security.

Oracle Ex. 1009, p. 919
Oracle v. DeBellis
IPR2024-00221

Sharing Database Files from a Network Server

The specific method of creating a server share on a dedicated network server depends on the network operating system (NOS) in use. Ordinarily, the network administrator will create the server share for you, and you need only move the files to be shared from your local folder to the shared server directory. If your NOS supports permissions for individual files, request full access rights to your workgroup database and data .mdb files. Windows NT Server supports permissions for individual files, if the files are located on an NTFS partition.

If you want users to be able to run the application .mdb from the server, grant read and execute access; otherwise, grant copy-only access so users can copy the application .mdb file to their local computer. Make sure, however, that you (the share owner) have full network permissions for all of the shared files. Do not grant users more permissions for the shared files than they need.

Accessing the Shared Files

Users access the server share by mapping the server share to a drive letter or by using Uniform Naming Convention (UNC). Using UNC eliminates the problem with users assigning different logical drive letters to server shares when mapping the share to their computer. Unlike Access 2.0, Access 97 supports UNC, as well as long file names (LFNs).

The network share you create, whether from your computer or from a network server, appears in the Network Neighborhood window for the selected server. In this example, Oakleaf0 is a Windows NT 4.0 Server (Primary Domain Controller) and Oakleaf1 is the workstation used to write this edition. Oakleaf3 is another Windows NT 4.0 Server that acts as a Backup Domain Controller. The \Shared\Nwind directory of Oakleaf0 is shared as `NWIND`, but appears as `NWIND` in the list of shares available from Oakleaf0 (see Figure 25.47). Double-clicking the nwind share displays icons for the shared files. (Use of the term *directory*, rather than *folder*, is more common when referring to file servers.)

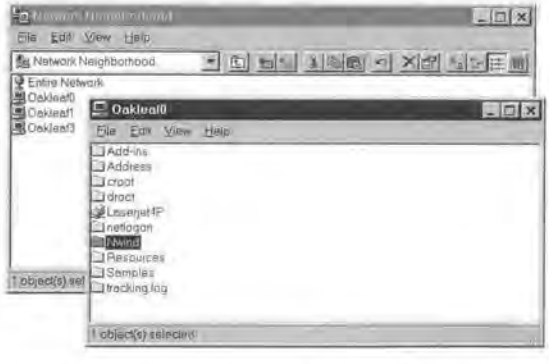


FIG. 25.47 The \Shared\Nwind directory shared as NWIND on the Oakleaf0 server appears in Network Neighborhood as `NWIND`.

Oracle Ex. 1009, p. 920
Oracle v. DeBellis
IPR2024-00221

Attaching the Shared Workgroup System File. Users of your application ordinarily use Workgroup Administrator to join the workgroup using the shared workgroup database file. You copy the shared data .mdb file and the System.mdw file to the server's shared directory. To access files using UNC, type **\\ServerName\ShareName\FileName.ext**. If you're sharing files from your computer, *ServerName* is the name you assigned to your computer when you installed Windows 95 or Windows NT. Figure 25.48 shows how to specify a workgroup database file using UNC.

Caution

Do not use a logical drive letter mapped from a share to specify the location of System.mdw. Users are likely to map shares to different logical drive letters.

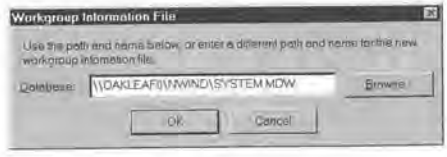


FIG. 25.48 Specifying the workgroup to join using UNC.

Troubleshooting

After changing the location of the workgroup database file, Access opens with a `'d:\path\filename.mdw' isn't a valid path` *message, and then Access closes.*

Either the drive letter, path, or file name entry isn't valid, or you cannot connect to the server share specified by the UNC name, or, if mapped to a drive letter *n*:[*\path*]. Use Explorer or Network Neighborhood to verify that your entry is correct and that your network connection to the server is working. In Explorer, choose View, Refresh to verify that the server connection currently is valid.

Refreshing the Links to the Shared Data File. Prior to distributing your application .mdb file, you must change the links to point to the shared data .mdb file. This step is especially important if you have revoked the design mode permissions for the Users group because the revocation prevents members of the Users group from refreshing the links. To refresh the links to point to the shared data .mdb file, follow these steps:

1. As a member of the Admins group, open the application .mdb file and choose Tools, Add-Ins, Linked Table Manager.

Note

If you share the files from your computer with the Windows 95 network, you must perform this step on another networked computer because your share does not appear in Network Neighborhood and you do not have access to files on your computer through UNC file names. In this case, open the shared application .mdb to refresh the links.

Oracle Ex. 1009, p. 921
Oracle v. DeBellis
IPR2024-00221

2. Click the Select All button, and then mark the Always Prompt for New Location check box. (If you don't mark this check box and the existing links are valid, you won't be able to refresh the links.) The Linked Table Manager's dialog appears as shown in Figure 25.49, assuming you performed the database splitting example earlier in this chapter.

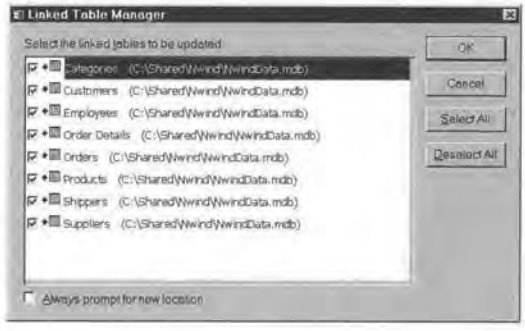


FIG. 25.49 Preparing to move table links to a shared data .mdb file with the Linked Table Manager.

3. Click the OK button to open the Select New Location of Categories dialog.

> **Note**
>
> If a local copy of the linked file specified in the opening dialog of Linked File Manager exists and you don't mark the Always Prompt for New Location check box, the links are simply refreshed and the Select New Location of *TableName* dialog doesn't appear.

4. Open the Look In list and select Network Neighborhood, and then select the server and share to display the files in the share (see Figure 25.50). Select the data .mdb file and click the Open button.



FIG. 25.50 Selecting the shared data .mdb file in the \Network Neighborhood*ServerName*\ *ShareName* folder.

Oracle Ex. 1009, p. 922
Oracle v. DeBellis
IPR2024-00221

5. The Linked File Manager automatically refreshes links for all of the linked tables it finds in the selected data .mdb file. On completion of the refresh process, a message confirms that all linkages were refreshed and the Linked File Manager's dialog appears as shown in Figure 25.51. Click the OK button to close both the message box and the dialog.



FIG. 25.51 The Linked Table Manager confirms the links to the data .mdb on the file server.

Once you refresh the links with the UNC location of the shared data .mdb file, you can distribute the application .mdb file to users.

Administering Databases and Applications

Administering a multiuser database involves a number of duties besides adding and maintaining user accounts. The most important function of the database administrator is to ensure that periodic valid backup copies are made of database and system files. The database administrator's other responsibilities consist of routine database maintenance, periodic compacting of database files, and repairing databases.

Backing Up and Restoring Databases

The following maxims relate to maintaining backup copies of database files:

- The time interval between successive backups of databases is equal to the amount of data you are willing and able to reenter in the event of a fixed disk failure. Except in unusual circumstances, such as little or no update activity, daily backup is the rule.
- Rotate backup copies. Make successive backups on different tapes or disk sets. One of the tapes or disks may have defects that could prevent you from restoring the backup. The backup device, such as a tape drive, can fail without warning you that the recorded data isn't valid.
- Test backup copies of databases periodically. You should test one different copy in the backup rotation sequence for restorability. If you rotate five daily backup tapes, for example, you should randomly choose one of the tapes, restore the database file from the tape, and open it with Access to ensure its validity every fifth day. Access tests each database object for integrity when you open the database file.

Oracle Ex. 1009, p. 923
Oracle v. DeBellis
IPR2024-00221

- Maintain off-site backups that you can use to restore data in case of a disaster, such as a fire or flood. The copy of the backup tape or disk that you test for restorability is a good candidate for an off-site backup copy.

You can back up database files on a network server by copying them to a workstation that has a fixed disk, but this technique doesn't provide backup security. The user of the workstation can erase or damage the backup copy if you don't create the off-site copy required for security against disasters.

Backing up data on network and peer-to-peer servers usually is accomplished with a tape drive device. These devices usually include an application that backs up all data on a network server or selected files on peer-to-peer servers at intervals and times you select. The simpler the backup operation, the more likely you are to have current backups. Regardless of how automated the backup procedure is, however, you need to manually restore the test copy.

Compacting and Repairing Database Files

Compacting and repairing database files was discussed in Chapter 3, "Navigating Within Access." You should compact database files in which applications add and delete data to recover the disk space occupied by the deleted data. The procedure for compacting a database is similar to that described for encrypting and decrypting databases, the subject of the next section, except that you choose Tools, Database Utilities, Compact Database rather than Tools, Security, Encrypt/Decrypt Database.

◀◀ See "Compacting Databases," p. 102
◀◀ See "Repairing Databases," p. 104



Note

You can improve the operating speed of Access if you periodically defragment database files. Windows 95's Disk Defragmenter utility tells you whether a disk drive has sufficient fragmentation to justify running the utility.

If you receive a message that a database is corrupted or if the database behaves in an irregular manner, one or more of the objects it contains may be corrupt as the result of a hardware error. Databases can become corrupt as the result of a power failure when the computer is writing to the database file. The Repair Database choice of Access's File menu attempts to repair the damage. If Access cannot repair the corruption, you must restore the latest backup copy. Test the backup copy with the existing System.mda or Workgroup.mdw file; in some cases, you may need to restore the prior .mda or .mdw file that contains the user account data for the database.

Encrypting and Decrypting Database Files

File-level security isn't complete until you encrypt the database. Encrypting the database prevents others from reading its contents with a text editing or disk utility application, such as is included with Symantec's Norton Utilities. Encryption of databases causes

Oracle Ex. 1009, p. 924
Oracle v. DeBellis
IPR2024-00221

Access' operations on tables to slow perceptibly because of the time required to decrypt the data. Only members of the Admins group can encrypt or decrypt a database.

> **Note**
>
> If you are using a fixed disk data-compression utility, you will find that encrypting your database files reduces the percentage of compression to zero or a very small number. Encrypting files eliminates the groups of repeating characters that form the basis of most data-compression algorithms.

To encrypt or decrypt an Access database file, complete the following steps:

1. Make sure that the disk drive of the computer on which the database is stored has sufficient free space to create a copy of the database you intend to encrypt or decrypt. Access makes a new copy of the file during the process.
2. All other workstations, including your own, need to close the database file to be encrypted. You cannot encrypt or decrypt a database file that is in use on any workstation.
3. Choose Tools, Security, Encrypt/Decrypt Database. The Encrypt/Decrypt Database dialog appears.
4. Select the name of the database file to be encrypted and click OK.
5. If the file already is encrypted, it is decrypted, and vice-versa. The title bar of the dialog that opens indicates whether the file will be encrypted or decrypted in this operation. If you are interested only in whether the file has been encrypted, you can click Cancel now.
6. Type the name of the encrypted or decrypted file to create in the Encrypt FileName As dialog, and click OK. Normally, you type the same name as the original file; Access does not replace the original copy of the file if the process does not succeed.

Databases are compacted by Access when they are encrypted or decrypted.

> **Note**
>
> You don't need to encrypt files while you're developing applications using files that aren't shared with others unless the files contain sensitive information. After the files are made shareable, a good security practice is to encrypt them, even if they don't contain confidential data.

Oracle Ex. 1009, p. 925
Oracle v. DeBellis
IPR2024-00221

Chapter 26

Connecting to Client/ Server Databases

One of the computer buzzwords of the mid-1990s, *downsizing*, was mentioned in the context of local area networks (LANs) in the preceding chapter. Downsizing has another element: moving database management systems from mainframe computers to client/server RDBMSs that run on PCs and reduced instruction set computing (RISC) workstation-servers.

Another newly minted term, *rightsizing*, means choosing the best combination of computer platforms to ensure maximum availability of corporate data to those who need it. Rightsizing often involves retaining mainframe computers as giant database servers but moving the applications that access and manipulate the data from the mainframe to PCs.

The final member of the sizing trio is upsizing. *Upsizing* means moving tables from a desktop database, such as Access 97, to a client/server RDBMS. You can continue to use the application components of your desktop database if the desktop database can link to the tables of the client/server RDBMS. This chapter introduces you to Access front-ends for client/server databases and shows you how to use the Access Upsizing Wizard to move Access databases to Microsoft SQL Server 6.5.

Creating Scalable Access Applications

Regardless of the linguistic legitimacy of terms such as rightsizing, these words have become ingrained in today's computerese. Each of these expressions relates to the *scalability* of applications. Scalable applications can run on a variety of platforms, communicate by industry-standard LAN and wide area networking (WAN) protocols, and access data stored in a variety of different types of databases. 32-bit Access 97 runs under Windows 95 or Windows NT 3.51+, so Access itself is moderately scalable. Access 97 is a threaded application and Jet 3.5 is multithreaded; thus, Access can take advantage of the symmetrical multiprocessing (SMP) capabilities of Windows NT running on workstations with multiple processors. Windows NT runs on Intel-based PCs

Oracle Ex. 1009, p. 926
Oracle v. DeBellis
IPR2024-00221

and RISC systems, such as DEC Alpha servers and IBM PowerPC platforms, but you need a version of Access 97 that's compiled for the specific RISC processor in use. It remains to be seen whether Microsoft intends to release versions of Office 97 for use with Windows NT running on RISC machines.

On the networking front, Windows 95 and Windows NT support the three most common networking protocols: NetBEUI, IPX/SPX, and TCP/IP. Following is a brief description of each of these protocols:

- *NetBEUI (NetBIOS Extended User Interface)* is the foundation of Windows Networking and the simplest LAN protocol to implement. NetBEUI isn't a routable protocol, so it isn't suitable for large-scale networks. NetBEUI, however, is quite efficient and useful for setting up LANs in small- to medium-sized organizations.
- *IPX/SPX (Internetwork Packet Exchange/Sequenced Packet Exchange)* is the primary protocol for Novell NetWare LANs. IPX/SPX is the default network protocol for Windows 95 because of the large installed base of NetWare 3.x/4.x LANs.
- *TCP/IP (Tranmission Control Protocol/Internet Protocol)* is the protocol of the Internet and UNIX systems. TCP/IP is a routable protocol that enables the creation of very large LANs and WANs. The remarkable growth of the Internet in the mid-1990s has established TCP/IP as the standard protocol for scalable networking.

Fortunately, Windows 95 and Windows NT 3.5+ let you run all three of these protocols simultaneously. Unless you're connected to a NetWare LAN, it's more efficient to use only NetBEUI and TCP/IP. As an example, you might connect to SQL Server 6.5 running under Windows NT Server 4.0 with NetBEUI and to Sybase System 11 or Oracle7 running on a UNIX server with TCP/IP.

The third element of scalability, the capability to access data that resides in a variety of SQL-compliant databases, is provided by Microsoft's Open Database Connectivity (ODBC) products. Office 97 includes the new 32-bit ODBC 3.0 Administrator application and a 32-bit ODBC driver for Microsoft SQL Server. There are about 150 third-party suppliers of ODBC drivers. ODBC provides Access with the capability to connect virtually to any popular mainframe-, minicomputer-, RISC-, and PC-resident SQL-compliant database through Microsoft and third-party 32-bit ODBC drivers. This chapter describes how ODBC works and how you can connect to client/server databases with ODBC drivers.

Note

Access 97 requires 32-bit ODBC drivers to connect to client/server RDBMSs. Existing 16-bit ODBC 2.0 drivers used with Access 2.0 and its predecessors do not work with Access 97. If you are upgrading from Access 2.0 (or an earlier version), and are using 16-bit ODBC drivers for RDBMSs other than Microsoft SQL Server, you must obtain 32-bit versions of ODBC drivers from the RDBMS vendor or a third-party publisher.

Oracle Ex. 1009, p. 927
Oracle v. DeBellis
IPR2024-00221

Defining the Client/Server Environment

Client/server databases are designed specifically for use on application server-based networks. An *application server* uses a network operating system, such as Windows NT Server, that is optimized specifically for running applications rather than sharing files or peripheral devices. Client/server databases have many advantages over conventional database systems, including increased database security, the incorporation of all components of the database. In a single file, and faster access to networked data. The clients of a client/server database are workstations, often called *front-ends*, that are connected to the server, called the *back-end*. In these respects, the "split" Access databases, described in the preceding chapter, and client/server databases are similar. The principal difference between Access and a typical client/server database manager, such as Microsoft SQL Server, is that the client/server RDBMS performs many operations on the server that traditionally are carried out by database applications running on the client workstation.

Client/server database managers accept SQL statements from client applications. The client/server RDBMS interprets the SQL statement and executes the actions specified in the statement. If you send a SELECT query SQL statement to the server, the server returns only the result set to the client; processing of the query occurs on the server computer. This action speeds query generation two ways: The amount of information traveling over the network is reduced, and server computers often have much more powerful and faster microprocessors than the workstation clients.

Note

SQL Server isn't limited to traditional client/server architecture in which clients connect to the server via a network. You can use Access front-ends on the same Windows NT server that runs SQL Server. Microsoft also offers a developer's version of SQL Server that runs as a service on Windows NT Workstation. Microsoft announced in the fall of 1996 its intention to port SQL Server to Windows 95, making SQL Server a direct competitor to the Jet database engine.

To understand many of the examples in this chapter, you need to know how the computers that create the examples are set up. The following list describes the computers and network used to create this chapter's examples of employing ODBC for connecting to client/server databases:

- The primary server (OAKLEAF0) is a 133-MHz Pentium clone with 64M of RAM and a 4.3G Seagate Barracuda Fast, Wide SCSI-2 fixed-disk drive that is connected to an Adaptec AHA-2940UW fixed-disk controller on the PCI bus. Windows NT Server 4.0 is installed as a Primary Domain Controller (PDC) on the 2G C: drive formatted with NTFS (Windows NT's New Technology File System). The 2.3G D: partition is used for file sharing. The E drive is a Toshiba double-speed SCSI CD-ROM drive. A 1G Tandberg tape backup drive also connects to the SCSI bus.
- A second server (OAKLEAF3) is a 166-MHz Pentium PC with a PCI bus, 32M of RAM, and two 4.3G Seagate Barracuda Wide, Fast SCSI-2 drives that are connected

Oracle Ex. 1009, p. 928
Oracle v. DeBellis
IPR2024-00221

to an Adaptec AHA-2940UW. \\OAKLEAF3, which dual-boots Windows 95 and Windows NT Server 4.0 (as a Backup Domain Controller or BDC), is used primarily for digital video capture with an Interactive Images Plum card and non-linear digital video editing by using Adobe Premiere 4.2. OAKLEAF3 has a 4x SCSI-2 CD-ROM and a Hewlett-Packard ScanJet IIcx scanner that are connected to the narrow SCSI port of the AHA-2940UW.

- Microsoft SQL Server 6.5 runs as a service under Windows NT Server 6.5 (NTS) on the OAKLEAF0 and OAKLEAF3 servers. (A Windows NT service is an application that starts automatically when you boot Windows NT, before the user logs on.) Exchange Server 4.0, Internet Information Server 3.0 (beta version), and Microsoft Media Server 1.0 also run as services on OAKLEAF0. SQL Enterprise manager, the administrative application for SQL Server 6.5, is installed on all computers except OAKLEAF2.
- The client used to write this edition (OAKLEAF1) is an 80486DX4-100 PCI-bus clone with 16M of RAM, and 300M, 600M, and 1.2G IDE fixed disks, plus a 2X SCSI CD-ROM drive connected to the SCSI port of a Sound Blaster 16 audio adapter. OAKLEAF1 dual-boots Windows NT Workstation 4.0 and Windows 95. OAKLEAF1 is scheduled for retirement at the end of 1996.
- Another client (OAKLEAF2) is an 80486DX2-66 ISA clone with 8M of RAM and a 300M IDE disk drive that runs Windows 95. OAKLEAF2 is used primarily for composing and sequencing music. It is equipped with a variety of legacy (pre-Plug and Play) sound cards and several MIDI synthesizer modules. The 80486 client is used for testing the performance of custom applications (except Access 97 applications) on yesterday's average client workstation configuration. 16-bit MIDI sequencing applications run fine under Windows 95 on OAKLEAF2, but trying to run Access 97 applications on 80486DX-33 PCs with 8M of RAM definitely is not recommended.
- All servers and clients are equipped with Intel EtherExpress 16 Pro network interface cards (NICs) that are connected via thin Ethernet coaxial cabling. The LAN uses both NetBEUI and TCP/IP protocols. OAKLEAF0 and OAKLEAF3 also have Intel 100BaseT Fast Ethernet adapters for transferring compressed digital video data at rates up to about 42 Mbps. Remote Access Services (RAS), which allow dial-up (modem and ISDN) connections to the OAKLEAF0 server, are also implemented. Figure 26.1 shows the configuration of the network used for the examples discussed in this chapter.

If the preceding description appears to be written in Greek, don't despair; knowing the computer setup is necessary only to explain the entries in the text boxes of the dialogs that are illustrated in this chapter. Network systems of the complexity of the above configuration are common where a variety of operating systems, sometimes in both retail and beta versions, are used.

Oracle Ex. 1009, p. 929
Oracle v. DeBellis
IPR2024-00221

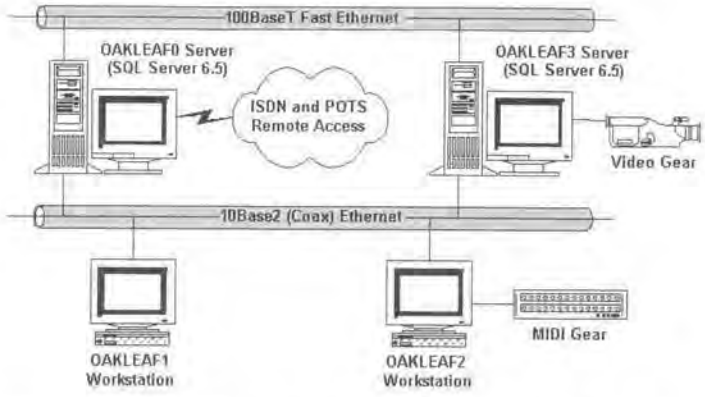


FIG. 26.1 The network topology of the computers used to write this book.

Defining Open Database Connectivity

Access uses the Microsoft Open Data Base Connectivity (ODBC) application programming interface (API) to provide access to any database system for which ODBC drivers are available. An API is a standardized method by which an application communicates with elements of the computer's operating system or environment. For example, applications use the Windows 95 (Win32) API in GDI32.EXE, a mostly 32-bit Windows 95 dynamic link library (DLL), to perform all display operations. The ODBC API enables a standard set of SQL statements in any application to be translated to commands that are recognized by the server database. The role of ODBC drivers is explained in the following section, "Understanding ODBC Drivers."

The ODBC API was the first element of Microsoft's Windows Open Services Architecture (WOSA) used to create a variety of classes of commercial Windows applications. (32-bit ODBC 3.0 is the fourth iteration of the ODBC API, and ODBC 3.5 is expected to be available by mid-1997). Messaging API (MAPI) and Telephony API (TAPI) also are members of WOSA, which includes industry-specific APIs such as WOSA/XRT for handling real-time stock market data.

Today, WOSA includes a group of APIs that enable Windows applications to manipulate data that resides in virtually any format on any type of computer, anywhere in the world. Enterprise-wide data sharing through LANs and WANs employs large mainframe computers as centralized database servers that feed data to or through client/server RDBMSs. A recent trend in enterprise-scale computing is for PC clients to connect to large client/server RDBMSs that are configured as data warehouses or data marts, rather than to connect directly to databases stored on mainframe computers.

One of today's trends in enterprise-wide computing is the use of distributed database systems. Distributed database systems enable elements of a large database to be stored on servers in different locations that act as if they were a single, large server. SQL Server 6.5 includes replication services and a Distributed Transaction Controller (DTC) to

Oracle Ex. 1009, p. 930
Oracle v. DeBellis
IPR2024-00221

implement the first phase of Microsoft's distributed database architecture. As advanced Windows operating systems—such as the next version of Windows NT (presently called Cairo)—are developed, and as additional members of WOSA become a commercial reality, PCs using Intel 80 × 86 architecture and RISC processors will capture a larger share of the server market. Access is designed to play an important role in enterprise-wide, distributed database systems: to create the applications that users need to view and update the myriad of databases to which they can connect. The advantage of Access as a client/server front-end is the ease with which you can create relatively simple decision-support and transaction-processing applications.

Understanding ODBC Drivers

The ODBC API consists of a driver manager and one or more ODBC drivers, as illustrated by the shaded boxes in Figure 26.2. Windows uses drivers to adapt its standard API to specific combinations of hardware such as displays, keyboards, and printers. Likewise, the ODBC API uses drivers to translate instructions passed from the application through the driver manager to instructions that are compatible with various RDBMSs. When the ODBC driver manager receives instructions from Access that are intended for a data source, such as a SQL Server database, the driver manager opens the appropriate ODBC driver for the database. The relationship of the ODBC driver manager and ODBC drivers parallels the relationship of Access 97's built-in, 32-bit Jet 3.5 database engine and the 32-bit ISAM drivers that are used to connect to Access, dBASE, FoxPro, and Paradox files.

ODBC drivers are classified as one of the following two types:

- *Single-tier drivers* translate SQL statements into low-level instructions that operate directly on files. Single-tier drivers are required for RDBMSs that don't process SQL statements directly. The widely used PC RDBMSs fall into this category. The 32-bit ISAM drivers included with Access for connecting to dBASE, FoxPro, and Paradox databases are single-tier drivers (but they aren't ODBC drivers). The Access, dBASE, FoxPro, Paradox, Excel, and Text ODBC drivers included with Microsoft Office 97 are 32-bit ODBC drivers.
- *Multiple-tier drivers* process ODBC instructions but pass SQL statements directly to the data source using SQL syntax that is acceptable to the back-end RDBMS. All popular client/server RDBMSs that can run on PCs—and most mini- and main-frame RDBMSs—process SQL statements directly. The SQL Server ODBC driver is a multiple-tier driver because SQL Server processes SQL statements directly.

One 32-bit ODBC driver is required for each type of client/server database whose tables you want to attach to an Access 97 database, but a single driver can be used to connect to several databases of the same type. Each connection to a database is called an *instance* of a driver. In addition to the ODBC driver, you may need additional files (usually license and/or communication files supplied by the RDBMS publisher) to connect to the back-end RDBMS. As an example, Microsoft supplies the required named pipes DLLs—16-bit Dbnmp3.dll and 32-bit Dbnmpntw.dll—to connect to Microsoft SQL Server. Named Pipes is the protocol through which you make connections to SQL Server. To connect to an Oracle database, for example, you need a local copy of Oracle's SQLNet.dll.

Oracle Ex. 1009, p. 931
Oracle v. DeBellis
IPR2024-00221

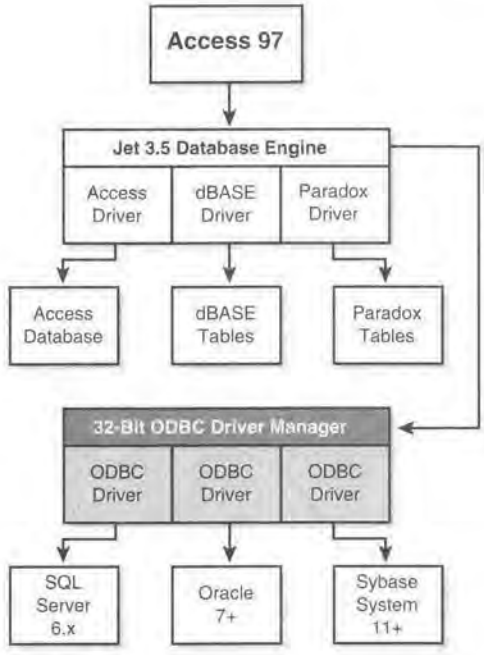


FIG. 26.2 A comparison of the ODBC 3.0 and Jet 3.5 driver systems used with Access.

The ODBC API is based on a standard called the *X/Open SQL Call Level Interface (CLI)*, which was developed by the SQL Access Group (an organization comprised of hardware manufacturers, software suppliers, and users of SQL databases). Microsoft published the standards for creating ODBC drivers, so any RDBMS supplier can make its database product compatible with Access by writing the appropriate driver. Microsoft provides a 32-bit ODBC driver with Office 97 that is compatible with Microsoft SQL Server 4.21a and later.

Most publishers of major client/server database management systems provide ODBC drivers for their database products. Oracle Corporation (Redwood Shores, CA) provides and supports ODBC drivers for its Oracle Server RDBMSs. Intersolv, Inc. (Rockville, MD) offers its DataDirect ODBC drivers for a variety of client/server RDBMSs, desktop databases, and file types.

You can obtain up-to-date information on Oracle and Intersolve ODBC drivers at the following URLs:


On the Web

http://www.oracle.com/products/oracle7/odbc/overview.html

http://www.intersolv.com/solution/dw_odbc.htm



Each workstation that needs to attach SQL Server tables to Access databases must be connected to the SQL Server through the network: Microsoft Windows Network, TCP/IP, Microsoft LAN Manager, IBM LAN Server, Banyan VINES, or Novell NetWare. In most cases, you set up your network adapter card and established the network protocol(s) in

Oracle Ex. 1009, p. 932
Oracle v. DeBellis
IPR2024-00221

use when you installed Windows 95 or Windows NT. You can modify your network configuration, if necessary, with Control Panel's Network option.

Installing the Microsoft ODBC Driver for SQL Server



The members of Office 97, except Access 97, depend on 32-bit ODBC drivers for interoperability with Jet, dBASE, FoxPro, Paradox, and SQL Server databases, plus Excel and text files. Microsoft Query also uses ODBC to connect to the preceding data sources. Thus, Office 97 automatically installs the 32-bit ODBC 3.0 administrator application, called the ODBC Data Source Administrator, as a Control Panel tool when you choose the Typical or Custom Setup option. Figure 26.3 shows the 32-bit ODBC tool installed in Windows 95's Control Panel. In Windows NT 4.0's Control Panel, the 32-bit ODBC icon does not have the `32-bit` caption prefix.



FIG. 26.3 Windows 95's Control Panel with the 32-bit ODBC 3.0 Administrator tool installed.



Note

If you installed Microsoft Access 2.0, Word 6.0, or Excel 5.0 under Windows 95, and if you have elected to install the ODBC driver for Access or the Desktop Database Drivers for Word or Excel, you see an ODBC icon in Control Panel. This is the icon for the 16-bit ODBC Administrator and drivers, which do not work with Access 97. You must install the 32-bit ODBC Administrator and drivers for use with Access 97, Word 97, or Excel 97.

To install the SQL Server ODBC driver on a Windows 95 or Windows NT 4.0 workstation on which Office 97 is installed, perform the following steps:



1. Close all open Windows applications and launch Control Panel's Add/Remove Programs tool. When you install Office 97, the Setup program creates an entry in the list of installed programs. Click to select the Microsoft Office 97 Professional entry (see Figure 26.4).

Oracle Ex. 1009, p. 933
Oracle v. DeBellis
IPR2024-00221



FIG. 26.4 The Add/Remove Programs feature of Control Panel lists previously installed 32-bit applications.

2. With the Microsoft Office 97 Professional item selected, click the Add/Remove button to launch the Office 97 Setup maintenance program.
3. Insert the Office 97 CD-ROM when prompted and click OK. The first dialog of the Setup maintenance program appears, as shown in Figure 26.5.



FIG. 26.5 The first dialog of the Access 95 Setup maintenance program.

4. Click the Add/Remove button to display the Microsoft Office 97—Maintenance dialog. Select the Data Access item in the Options list, as shown in Figure 26.6. The appearance of other items in the Options list varies, depending on how you originally installed Office 97.

Oracle Ex. 1009, p. 934
Oracle v. DeBellis
IPR2024-00221

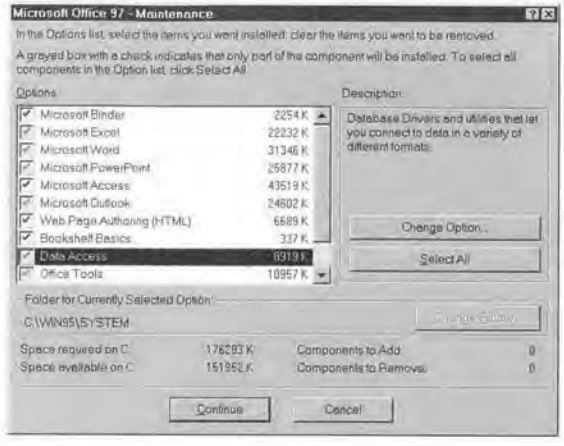


FIG. 26.6 The Microsoft Office 97—Maintenance dialog.

5. With the Data Access entry in the Options list selected, click the Change Option button to display the Microsoft Office 97—Data Access dialog. Select the Database Drivers item in the Options list (see Figure 26.7) and click the Change Option button.

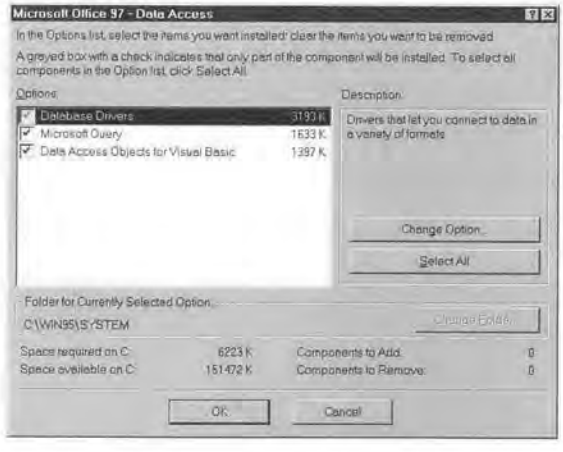


FIG. 26.7 The Microsoft Office 97—Data Access dialog.

6. Double-click the Microsoft SQL Server Driver item in the Options list to mark the check box for installation (see Figure 26.8).

7. Click OK to close the Database Drivers and Data Access dialogs, and click Continue to install the SQL Server driver from the Office 97 CD-ROM. Click OK to close the Setup maintenance application when you receive the message that installation is complete.

Oracle Ex. 1009, p. 935
Oracle v. DeBellis
IPR2024-00221

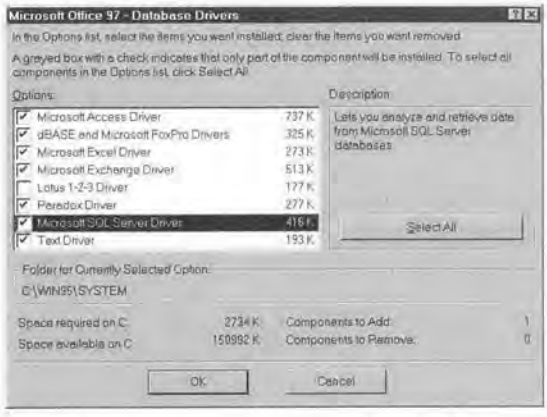


FIG. 26.8 Selecting the SQL Server driver for installation in the Microsoft Office 97—Database Drivers dialog.

8. Click the OK button of the Add/Remove Programs tool to close it, then close Control Panel.

When Setup installs the SQL Server ODBC driver, it creates the required entries for SQLsrv32.dll in the Registry in the `HKEY_LOCAL_MACHINE\SOFTWARE\ODBC` hive (see Figure 26.9).

> **Note**
>
> Previous 16-bit versions of ODBC stored data in ODBCINST.INI (which lists the installed ODBC drivers) and ODBC.INI files (which list installed ODBC data sources) in your \WINDOWS directory. The Registry entries include similar information in the `\ODBC\ODBCINST.INI` and `ODBC\ODBC.INI` groups.

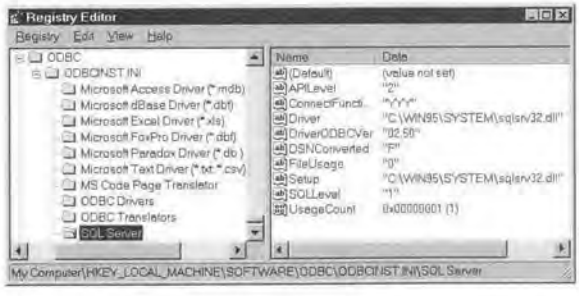


FIG. 26.9 The Registry Editor (Regedit.exe) displaying the Registry entry for the 32-bit SQL Server driver.

Oracle Ex. 1009, p. 936
Oracle v. DeBellis
IPR2024-00221

Adding and Removing SQL Server Data Sources

An ODBC data source for a client/server RDBMS is a definition of a database—including the name of the server on which the RDBMS and database is located, the name of the ODBC driver required to connect to the database, and, optionally, the name of the database. You need at least one SQL Server data source to enable Access to attach, export, or import SQL Server tables. You can use Control Panel's ODBC Administrator tool or Access's built-in ODBC data source feature to create SQL Server data sources.



Following are the three different types of ODBC data sources supported by Access 97:

- *User data sources* are specific to each user of the workstation. The definitions of these data sources are stored in the `\HKEY_CURRENT_USER\Software\ODBC\ODBC.INI` section of the Registry. You can create conventional data sources using the ODBC Administrator tool or from within Access.
- *System data sources (System DSNs)* are stored in the `HKEY_LOCAL_MACHINE\SOFTWARE\ODBC\ODBC.INI` section of the Registry. All users of the workstation can choose from available System DSNs. Like user data sources, you can create system data sources using the ODBC Administrator tool or from within Access. Windows NT Server 4.0 requires System DSNs for operations that run as Windows NT services, such as logging usage of an Internet Information Server 2.0 Web site.
- *File data sources* are individual text files that define the ODBC data source, which you can share with other Access 97 users. File data sources are a standard feature of ODBC 3.0; Access is unique in its ability to create file data sources for ODBC 2.5. File data sources don't appear in the Registry.

The following sections describe how to use ODBC Administrator and Access 97 to create SQL Server data sources for the pubs example database of Microsoft SQL Server 6.x.

> **Note**
>
> It's a tradition in SQL Server to use all lowercase names for databases and most other database objects. Early versions of SQL Server used case-sensitive object names by default; the default for SQL Server 6.5's installation is case-insensitive object names.

Using the ODBC Data Source Administrator to Create a SQL Server Data Source

The traditional approach is to use Control Panel's ODBC Administrator tool to add or remove SQL Server data sources. To add a SQL Server data source with ODBC Administrator, follow these steps:

1. Double-click the 32-bit ODBC icon in Control Panel to launch the ODBC Data Source Administrator and display the default User DSN page (see Figure 26.10).

Oracle Ex. 1009, p. 937
Oracle v. DeBellis
IPR2024-00221

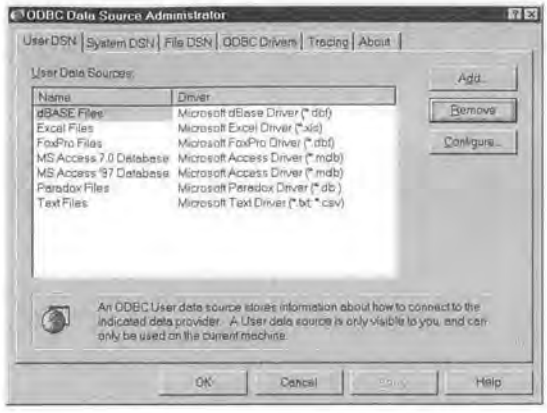


FIG. 26.10 The Data Sources dialog of the ODBC Administrator with default data sources installed.

Figure 26.10 shows the user ODBC data sources created during the installation of Office 97 ODBC drivers. Microsoft Office 97 installs the ODBC drivers for dBASE, Excel, FoxPro, and Access 97, plus optional drivers for Paradox and Text files. The list in Figure 26.10 reflects the ODBC driver choices made in Figure 26.8. There is no default data source for the SQL Server driver.

2. Click the Add button to add a new user ODBC data source with the Create New Data Source dialog. Alternatively, click the System DSN tab, then click Add to add a system data source. (Default system data sources aren't added when you install Office 97.) Select the SQL Server item in the Installed ODBC Drivers list of the Add Data Source dialog, as shown in Figure 26.11.

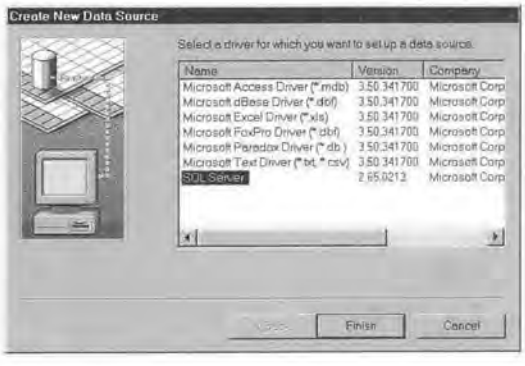


FIG. 26.11 Selecting the SQL Server ODBC driver in the Create New Data Source dialog.

3. Click Finish to display the ODBC SQL Server Setup dialog. Enter a short descriptive name of the SQL Server database in the Data Source Name text box. Unless you have a particular database that contains the tables you want to attach to Access,

Oracle Ex. 1009, p. 938
Oracle v. DeBellis
IPR2024-00221

type **pubs** to specify the pubs sample database supplied with SQL Server. Type a description of the database, such as **SQL Server Sample Database**, in the Description text box.

4. Select the name of the server from the Server combo list that holds the SQL Server database you want to use. If no entries appear in the combo list, type the name of the server (without the preceding \\) in the text box. If you are creating the data source on the Windows NT server running SQL Server, select (local) from the drop-down list. Unless you are using a special network library (netlib) for the data source, accept the (Default) entries for the Network Address and Network Library text boxes (see Figure 26.12). (The default 32-bit network library for SQL Server is Dbnmpntw.dll.) Mark the Use Trusted Connection check box if SQL Server is set up with Windows NT integrated security and you want the user to log on with his or her network user ID and password instead of a SQL Server user ID and password.

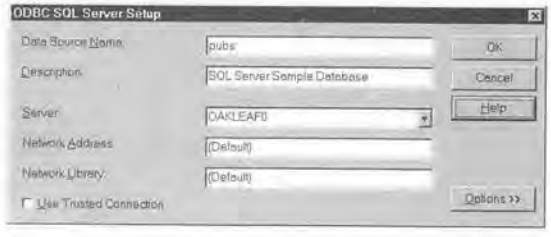


FIG. 26.12 Beginning the definition of a new ODBC data source.

5. Click the Options button to expand the dialog to specify a default database. Type **pubs** in the Database Name text box, as shown in Figure 26.13.

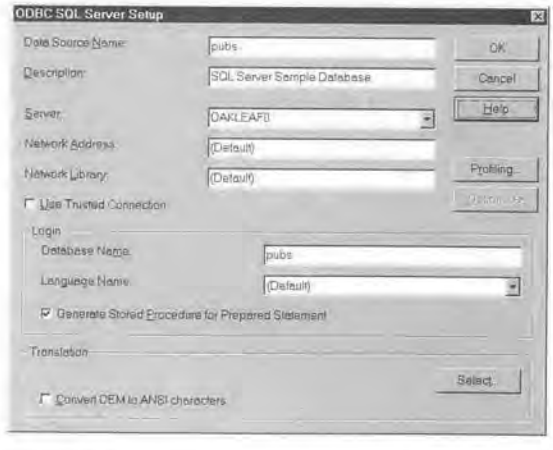


FIG. 26.13 Completing the definition of the ODBC data source.

Oracle Ex. 1009, p. 939
Oracle v. DeBellis
IPR2024-00221

The Generate Stored Procedure for Prepared Statement check box, when marked (the default), creates a temporary SQL Server stored procedure (a precompiled query) from prepared statements written in Transact-SQL, the SQL dialect of SQL Server. Access 97's new ODBCDirect feature, which you can implement only with Access VBA, is capable of creating multiple prepared statements that exist until you close the connection with SQL Server.

The Language Name drop-down list displays the national languages that the database supports. (The national language of the database is not necessarily the same as the Locale of Windows 95.) Accept the (Default) entry.

The Convert OEM to ANSI characters check box changes special characters in SQL Server database tables to their Windows (ANSI) equivalents when an ANSI equivalent is available. If you or the database administrator installed SQL Server with the default code page and you're using Windows 95 or Windows NT clients, you don't need to be concerned with OEM-to-ANSI conversion. Clear the check box if it is marked.

6. Click OK to add the new data source and close the Data Source dialog. Your new data source is added to the User Data Sources (Driver) list of the Data Sources dialog, as shown in Figure 26.14.

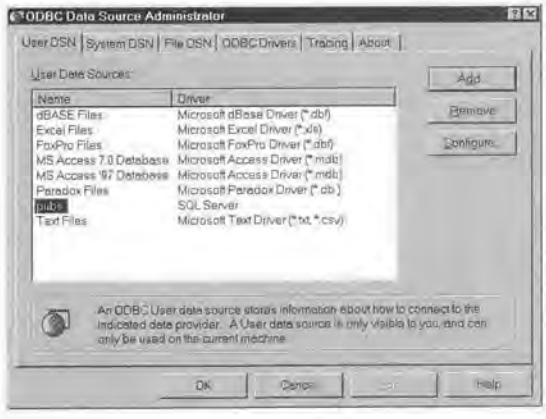


FIG. 26.14 The new pubs SQL Server data source added to the ODBC User Data Sources List.

7. Click OK to close the ODBC Data Source Administrator tool, and then close Control Panel.

You can add additional SQL Server data sources for a workstation by repeating steps 1 through 6 before exiting the ODBC Administrator application. The entries you make for each new conventional data source are added to the HKEY_CURRENT_USER\Software\ODBC\ODBC.INI section of the Registry, as shown in Figure 26.15. The ODBC data source entries are located under the HKEY_CURRENT_USER key, because other users of your PC may have different sets of ODBC data sources.

Oracle Ex. 1009, p. 940
Oracle v. DeBellis
IPR2024-00221

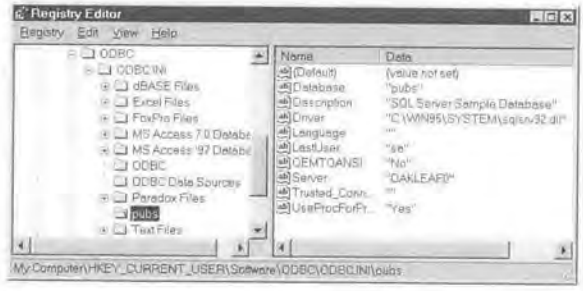


FIG. 26.15 The Registry entry for the pubs data source on the OAKLEAF0 server.

> **Note**
>
> Examples of entries in the Windows 95 Registry are provided in this chapter so that you can compare Registry entries for your computer in the event that you have problems making a connection to a client/server database. Each ODBC driver for a particular client/server RDBMS has its own set of properties, but the entries for the driver and data source should resemble Figures 26.9 and 26.15, respectively. If you have problems while connecting to a client/server database that you can't solve yourself, it's likely that the vendor's technical support person will ask for the values of your Registry entries for the ODBC driver and/or data source.

If you have problems connecting to your ODBC data source, you can turn ODBC tracing on by clicking the Tracing tab of the ODBC Create Data Source Administrator's dialog to display the Tracing page. Select either the All the Time or One-time Only options to create an ODBC log in the default Sql.log file (see Figure 26.16). You can choose a different file name and location by clicking the Browse button. Entries in the log file can pinpoint the source of a problem if the ODBC error messages you receive aren't sufficiently explicit.

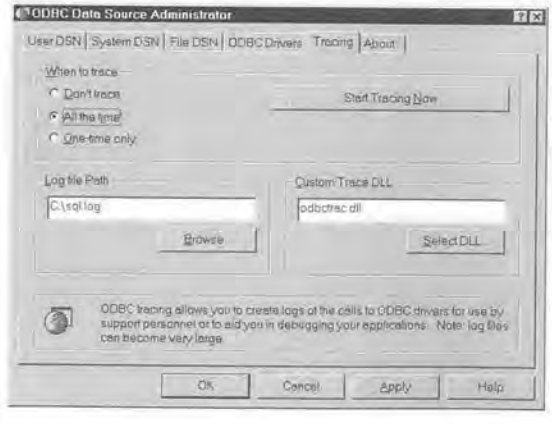


FIG. 26.16 Turning on the trace feature of the SQL Server ODBC driver.

Oracle Ex. 1009, p. 941
Oracle v. DeBellis
IPR2024-00221

Caution

Only turn on ODBC tracing when you are debugging a problem. The Sql.log file can become very large, and adding entries to the log reduces front-end performance dramatically.

The File Data Source page lets you create a sharable ODBC data source in the form of a .dsn file. The section "Creating a File Data Source Within Access 97," later in this chapter, describes how to create and use a file data source. The ODBC Drivers page provides details about the current set of 32-bit ODBC drivers installed on your PC. The About page displays the version numbers of the ODBC 3.0 files that are common to all ODBC drivers. The Windows 95 or Windows NT 4.0 Registry provides the information that appears in the ODBC Drivers and About pages.

Creating a User or System SQL Server Data Source Within Access 97

Access 97's File, Get External Data, Import and Link Tables choices let you create ODBC data sources without opening Control Panel and launching the ODBC Administrator tool. To create the data source within Access 97, follow these steps:

1. Create a new database named pubs.mdb, and then choose File, Get External Data, Link Tables to open the Link dialog. Select ODBC Databases () from the Files of Type drop-down list (see Figure 26.17) to open the Select Data Source dialog.



FIG. 26.17 Selecting ODBC Databases in the Link dialog.

2. Click the Machine Data Source tab. The default data sources for the ODBC drivers previously installed, and the pubs data source you created in the preceding section, appear as shown in Figure 26.18. The Machine Data Source page displays both user and system data sources.

3. Click the New button to open the Create New Data Source dialog, and accept the default User Data Source option (see Figure 26.19). Click Next to open the dialog in which you select the ODBC driver.

Oracle Ex. 1009, p. 942
Oracle v. DeBellis
IPR2024-00221

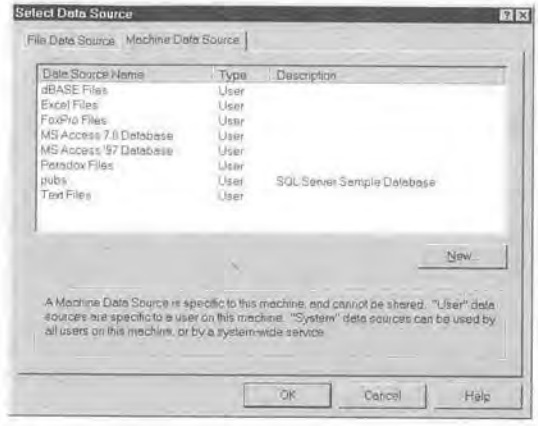


FIG. 26.18 The default data sources added with installation of the Office 97 desktop ODBC database drivers and the pubs data source.

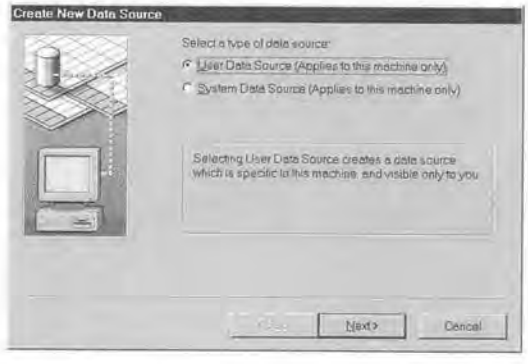


FIG. 26.19 Specifying the creation of a new user data source.

4. Select the SQL Server driver in the ODBC drivers list (see Figure 26.20), and click Next to continue.
5. The last Create New Data Source dialog lets you confirm the choice you made in the preceding dialogs (see Figure 26.21). Click Finish to open the ODBC SQL Server Setup dialog.
6. Click the Options button to expand the ODBC SQL Server Setup dialog and complete the entries as described in steps 3, 4, and 5 of the preceding section (refer to Figure 26.13).
7. Click OK to create the pubs data source and close the ODBC SQL Server Setup dialog. If you created the pubs data source in the preceding section, a message box appears asking whether you want to overwrite the existing data source. The data source is added as a User data source to the Machine Data Source list or replaces the existing entry (refer to Figure 26.18).

Oracle Ex. 1009, p. 943
Oracle v. DeBellis
IPR2024-00221

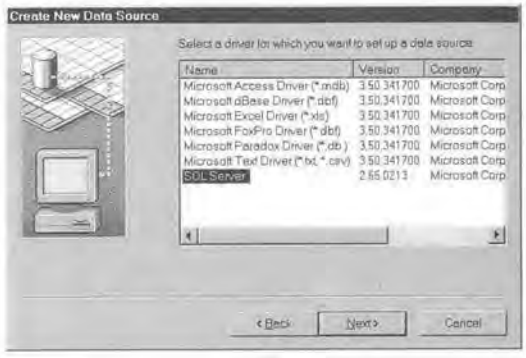


FIG. 26.20 Selecting the SQL Server driver in the third Create New Data Source dialog.



FIG. 26.21 Confirming creation of a user datasource with the SQL Server ODBC driver.

8. Click cancel to return to Access. The section "Using Databases Connected by ODBC" later in this chapter describes how to import tables from or link tables to SQL Server databases.

Creating a File Data Source Within Access 97

Access 97 supports 32-bit ODBC 3.0 file data sources. A *file data source* contains the Registry information for the data source but in conventional text format. File data sources are roughly equivalent to a single set of entries for a 16-bit ODBC data source stored in Windows 3.x's ODBC.INI. The advantage of a file data source is that it isn't machine- or user-dependent; you can share a *Filename*.dsn file with other users across the network. (The .dsn extension is the default for file data sources.)

To create a file data source for SQL Server's pubs database in a shared folder on a server, follow these steps:

1. Create a new pubs.mdb database, if you didn't create this database in preceding section.

Oracle Ex. 1009, p. 944
Oracle v. DeBellis
IPR2024-00221

2. Choose File, Get External Data, Link Tables. Select ODBC Databases () in the Files of Type drop-down list of the Link dialog to open the Select Data Source dialog with the default File Data Sources page open (see Figure 26.22).

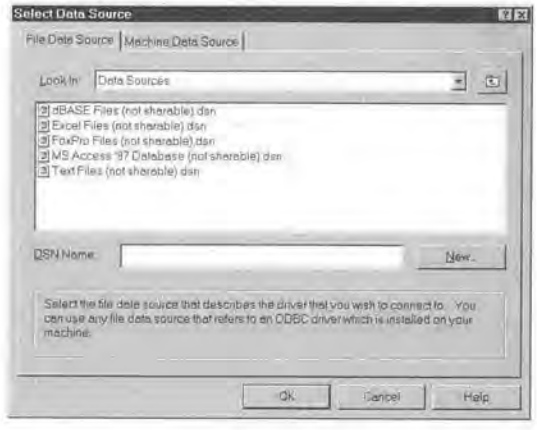


FIG. 26.22 The File Data Sources page of Access 97's Select Data Source dialog.

The file data sources installed by Office 97 appears with each data source described as not sharable. These data source files are located in your \Program Files\Common Files\ODBC\Data sources folder.

3. Click the New button to open the first Create New Data Source dialog, and select the SQL Server item in the list (refer to Figure 26.20).

4. Click the Advanced button to open the Advanced File DSN Creating Settings dialog.

5. In the Enter Driver-Specific Keywords and Values text box, type the additional lines **`SERVER=`*`ServerName`*** and **`DATABASE=pubs`** (see Figure 26.23). The `Driver={SQL Server}` line is added automatically from the selection you made in step 3. Make sure the Verify This Connection check box is marked (the default). Click OK to close the dialog, then click Next in the Create New Data Source dialog to continue.

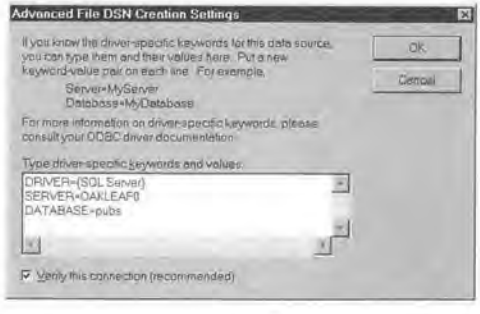


FIG. 26.23 Specifying the server and database names for an ODBC file data source.

Oracle Ex. 1009, p. 945
Oracle v. DeBellis
IPR2024-00221

6. Type the UNC path to the shared folder and the file name in the text box (`\\OAKLEAF0\Shared\Nwind\pubs.dsn` for this example). Alternatively, click the Browse button to open the Save As dialog, type **pubs** in the File Name text box, and then select the server share from Network Neighborhood in the Save In navigation list. Access automatically adds the .dsn extension for you. Alternatively, specify a server share mapped to a local drive letter (see Figure 26.24). Click Save to close the Save As dialog and return to the Create New Data Source dialog.

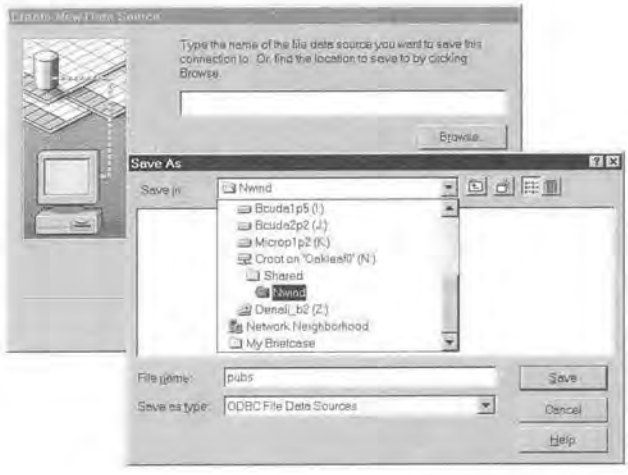


FIG. 26.24 Specifying the file name and share location for the pubs.dsn file data source.

The path and file name you specified appears in the Create New Data Source dialog (see Figure 26.25).

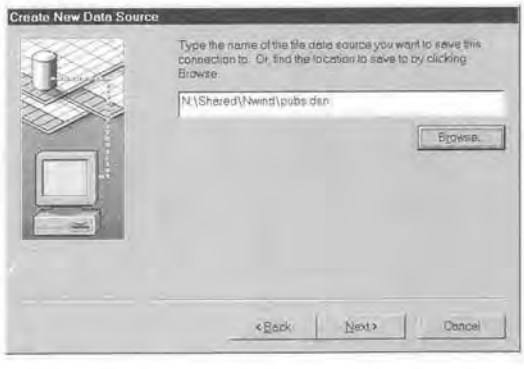


FIG. 26.25 The share path and file name used to test the new shared file data source.

Oracle Ex. 1009, p. 946
Oracle v. DeBellis
IPR2024-00221

7. Click Next to confirm your entries (see Figure 26.26). Click Finish to save and test the pubs.dsn file.

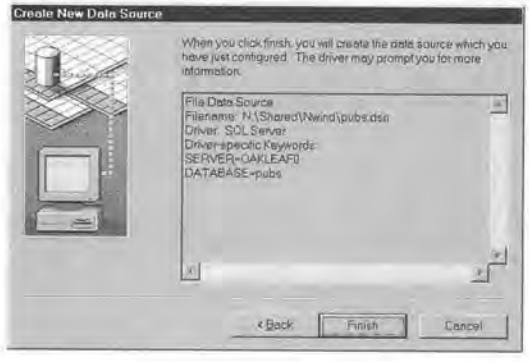


FIG. 26.26 Confirming the entries and location of the pubs.dsn file data source.

8. In the SQL Server Login dialog, verify that the server name you specified appears in the Server combo list, type your SQL Server login ID and password, and click the Options button to expand the dialog. Verify that the database you specified appears in the Database text box. The Application Name and Workstation ID values are added automatically (see Figure 26.27). Click OK to close the SQL Server Login dialog and test the connection. You receive an error message if the connection fails.



FIG. 26.27 Testing SQL Server login with the new file data source.

Figure 26.28 shows Notepad displaying the content of the pubs.dsn file data source. The workstation ID (`WSID`) entry specifies the computer name on which the file data source was created, but any computer can share the pubs.dsn file.

Oracle Ex. 1009, p. 947
Oracle v. DeBellis
IPR2024-00221

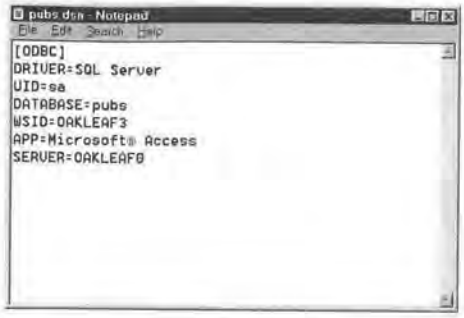


FIG. 26.28 The pubs.dsn file data source displayed in Windows Notepad.

Using Databases Connected by ODBC

After you add the SQL Server database as a user, system, or file data source, you can link, import, or export tables in the SQL Server database to your Access database, depending on the permissions granted by SQL Server to your SQL Server account. If you have full permissions for the database, you can use tables attached from the client/server database in the same manner that you use attached Access, dBASE, FoxPro, or Paradox tables. You don't specify primary key indexes to be used with client/server tables because primary key indexes are opened automatically when you open the table for which indexes have been created, just as with Access tables.

◀◀ See "Linking and Importing External Tables," p. 206



Pubs is a demonstration database that contains tables for a fictional book distributor. Pubs' tables include information on imaginary book publishers, titles, and authors. To attach the tables in the pubs SQL Server database to a new Access database (pubs.mdb in this example), follow these steps:

1. Launch Access and create a new database named pubs.mdb if you didn't create pubs.mdb earlier in this chapter. Choose File, Get External Data, Link Tables to display the Link dialog.
2. Open the Files of Type drop-down list. Select the ODBC Databases() entry in the list, as shown in preceding Figure 26.17, to open the Select Data Source dialog.
3. Click the Machine Data Sources tab to display a list of user and system data sources. Select the data source name—in this case, pubs—in the list (see Figure 26.29). Alternatively, click the File Data Source tab and type in or browse to the file data source you created in the preceding section. Click OK to open the SQL Server Login dialog.

Oracle Ex. 1009, p. 948
Oracle v. DeBellis
IPR2024-00221

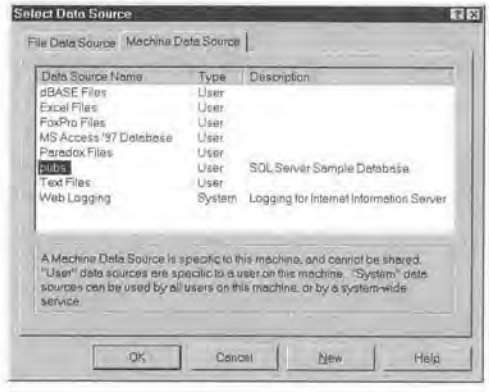


FIG. 26.29 Selecting the SQL Server pubs user data source.

4. Type your login identification and password in the Login ID and Password text boxes, respectively. The default login ID, sa (system administrator) with no password is used in this example (see Figure 26.30).



FIG. 26.30 Logging into the pubs SQL Server sample database as the system administrator with no password.

5. Click OK to log into SQL Server. After you connect to the pubs database with the ODBC API, the Link Tables dialog appears, as shown in Figure 26.31. The names of tables are prefixed with dbo, a SQL Server abbreviation for database owner.



FIG. 26.31 The list of tables in the pubs database of SQL Server 6.5 available for linking to an Access database.

Oracle Ex. 1009, p. 949
Oracle v. DeBellis
IPR2024-00221

6. Click the Select All button and mark the Save Password check box. Click OK to attach all of the tables to pubs.mdb.

   Selecting the Save Password check box eliminates the need to reenter your login ID and password each time you attach another table. If the database administrator adds additional security provisions to a database (by adding a special MSysConf table to the SQL Server database), this check box is disabled.

7. To make the tables updatable by Access, a unique index is required on one or more fields to identify each record uniquely in the table. If the SQL Server table does not have a primary-key index, Access 97 opens the Select Unique Record Identifier dialog and offers to create a local "pseudo-index" on the table. As an example, the discounttype and stor_id fields of the dbo.discounts table create a composite primary key (see Figure. 26.32). Select the field(s) for the primary key and click OK (or click Cancel to omit the pseudo-index). Do not add a pseudo-index for the dbo.royaltysched or dbo.titleview tables. Add a pseudo-index on the stor_id, ord_num, and title_id fields of the dbo.sales table.

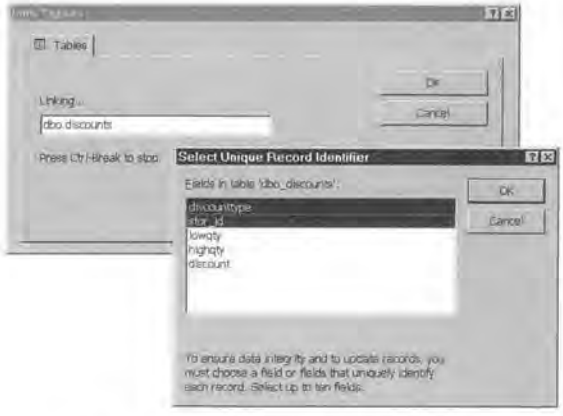


FIG. 26.32 Creating a pseud-oindex in the Select Unique Record Identifier dialog.

8. When the linking process is complete, the linked tables appear in the Tables page of the Database window (see Figure 26.33). The globe to the left of the table name indicates that the table is attached by ODBC. An underscore substitutes for the period between dbo and the table name because periods are not allowed within Access table names. (Periods are separators between objects and properties or methods in Access 97.)

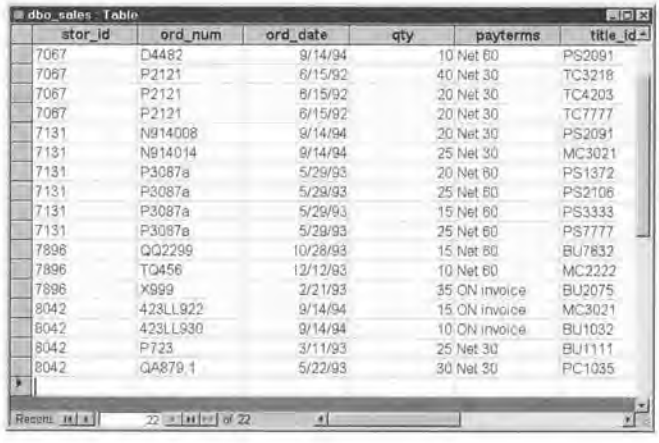
Double-click the dbo_*tablename* item you want to examine. To verify that you can update or append dbo_sales records, for which you created an Access pseudo-index, click the Last Record button to check for a tentative append record at the bottom of the table (see Figure 26.34).



Oracle Ex. 1009, p. 950
Oracle v. DeBellis
IPR2024-00221



FIG. 26.33 The 11 tables and one view attached from the pubs database.


dbo_sales : Table

stor_id	ord_num	ord_date	qty	payterms	title_id
7067	D4482	9/14/94	10	Net 60	PS2091
7067	P2121	6/15/92	40	Net 30	TC3218
7067	P2121	6/15/92	20	Net 30	TC4203
7067	P2121	6/15/92	20	Net 30	TC7777
7131	N914008	9/14/94	20	Net 30	PS2091
7131	N914014	9/14/94	25	Net 30	MC3021
7131	P3087a	5/29/93	20	Net 60	PS1372
7131	P3087a	5/29/93	25	Net 60	PS2106
7131	P3087a	5/29/93	15	Net 60	PS3333
7131	P3087a	5/29/93	25	Net 60	PS7777
7896	QQ2299	10/28/93	15	Net 60	BU7832
7896	TQ456	12/12/93	10	Net 60	MC2222
7896	X999	2/21/93	35	ON invoice	BU2075
8042	423LL922	9/14/94	15	ON invoice	MC3021
8042	423LL930	9/14/94	10	ON invoice	BU1032
8042	P723	3/11/93	25	Net 30	BU1111
8042	QA879.1	5/22/93	30	Net 30	PC1035

Record: 22 of 22

FIG. 26.34 The dbo_sales table displaying the tentative append record enabled by specifying a local Access pseudo-index for the table.

To create a query that joins the tables and displays the author name, book title, and book publisher, follow these steps:



1. Create a new query in Design view and add the dbo_authors, dbo_titleauthor, dbo_titles, and dbo_publishers tables to your query. Click the Close button of the Add Tables dialog.

   Key fields of SQL Server tables are indicated in bold type, the same way Access emphasizes key fields of its tables. Access creates and joins automatically based on the key fields and identically named foreign-key fields of the SQL Server tables.

2. Click and drag the au_lname field from the dbo_authors table to the field row of the first column of the query.

3. Repeat step 2 for the title field of the dbo_titles table and the pub_name field of the dbo_publishers table. Your Query Design window looks like the one shown in Figure 26.35.

Oracle Ex. 1009, p. 951
Oracle v. DeBellis
IPR2024-00221

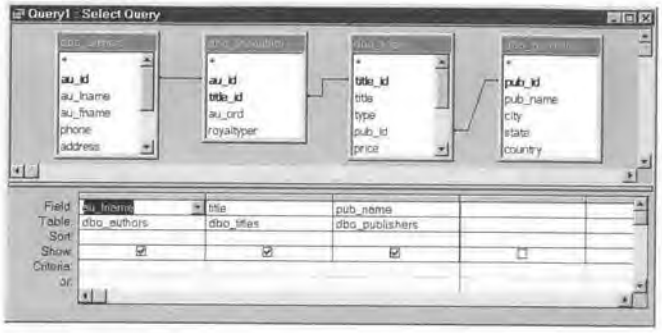


FIG. 26.35 The Query Design window for the example based on tables attached from the pubs database.

4. Click the Run button on the toolbar to display your query result set, as shown in Figure 26.36.

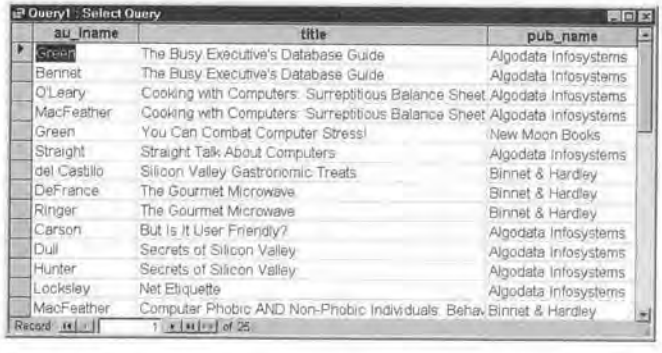


FIG. 26.36 The query result set from the query design shown in Figure 26.35.

5. Close and save your query for use in the next section.

To remove a linked table from the Northwind Traders database, click the Tables tab in the Database Window, and select the name of the linked table to delete—in this case, dbo_tableview. Press Delete. A message box appears, asking you to confirm that you want to delete the link to the table (see Figure 26.37). Click Yes because the SQL Server view is not needed in the sections that follow.



FIG. 26.37 The message box that confirms the deletion of a link to a table.

Oracle Ex. 1009, p. 952
Oracle v. DeBellis
IPR2024-00221

Note

You can eliminate the need to add the dbo_ prefix to a table name by renaming the attachments to the table name without the dbo_ prefix. Close any open attached tables, select the table, and then choose Edit, Rename to enter the new name. Alternatively, right-click the table name, then choose Rename from the popup menu.

The preceding example, using the pubs database, is typical of the procedure that you use to attach tables from any client/server database for which an ODBC driver is available.

Troubleshooting

A `Cannot connect to server` *or similar error message occurs when I attempt to attach to SQL Server.*

There are a variety of problems that can lead to your inability to connect to SQL Server. The most common cause is the lack of (or an outdated version of) the Dbnmpntw.dll library on your computer. Dbnmpntw.dll is the 32-bit named pipes library that is required to connect to SQL server. Choose Start, Find, Files or Folders, and verify that Dbnmpntw.dll is installed in your \Windows \System folder. (The size of the version of Dbnmpntw.dll that was current when this edition was written is 22K. The date should correspond to the dates of other files installed by Access or other applications that install Dbnmpntw.dll.) If you have additional copies of Dbnmpntw.dll with earlier dates in folders other than \Windows\System, delete these unneeded copies.

Using Access 97's SQL Pass-Through Queries

SQL pass-through queries allow you to write queries in the dialect of SQL used by the server RDBMS. SQL Server, as an example, lets you write stored procedures that you can execute by name, instead of by sending individual SQL statements to the server. Executing a stored procedure query is faster than executing the query through the Jet database engine because of reduced network traffic and faster execution of the query by the server. (Stored procedure queries are precompiled by SQL Server.) You also need to use SQL pass-through if you want to take advantage of special Transact-SQL reserved words not included in Jet SQL. In many cases, SQL pass-through queries execute faster than queries that use Jet's built-in query optimizer.

Executing a Simple Pass-Through Query

Access 1.x required that you use Access Basic to declare the functions of MSASP110.DLL, a SQL pass-through library supplied by Microsoft. Beginning with version 2.0, Access provides built-in SQL pass-through capability, similar to that offered by Visual Basic 3.0 and later. You can convert the query you created in the preceding section to a SQL pass-through query by following these steps:

1. Open the saved query you created in the previous section in Design view; then

Oracle Ex. 1009, p. 953
Oracle v. DeBellis
IPR2024-00221

choose Query, SQL-Specific, Pass-through.



2. The SQL window opens with the SQL statement behind your query. The Query Design button is disabled when you convert a conventional Access query to the SQL pass-through type.

3. Table names with the dbo_ prefix aren't acceptable to SQL Server, so replace all of the underscores (_) with periods (.), as shown in Figure 26.38. Alternatively, you can delete the dbo_ prefix from each table name.

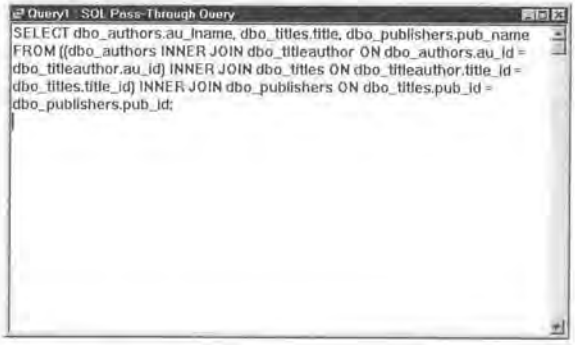


FIG. 26.38 The SQL window for a SQL pass-through query with the dbo. prefix for SQL Server table names.

4. Choose File, Save to save your pass-through query. (If you don't save the query, it may attempt to execute against the attached tables.)

5. Click the Run button on the toolbar to execute your pass-through query. When the Select Data Source dialog appears, click the appropriate tab to specify your machine or file data source. When you click OK, you must enter your SQL Server login ID and password, then click OK again. You receive the same query result set as that from the conventional query executed through the Jet database engine against the linked tables.



Note

Query result sets returned by SQL pass-through queries are `Recordset` objects of the `Snapshot` type, which are not updatable. Other query result sets against databases linked by ODBC are `Recordset` objects of the `Dynaset` type but may or may not be updatable, depending on the query design and whether you set a local pseudo-index on the table.

When you save a SQL pass-through query, the ODBC symbol appears next to the name of the pass-through query in the Queries list of the Database window (see Figure 26.39). SQL pass-through queries are especially useful for creating high-performance client/server front-ends with Access VBA.

Oracle Ex. 1009, p. 954
Oracle v. DeBellis
IPR2024-00221



FIG. 26.39 The Queries page of the Database window with a saved SQL pass-through query.

Setting the Connect String Property for SQL Pass-Through Queries

You can use the ODBC Connect Str(ing) property of the SQL pass-through QueryDef object to avoid the need to respond to the Select Data Source dialog and, if desired, bypass the SQL Server Login dialog. When you create a SQL pass-through query, Access automatically sets the value of the ODBC Connect Str property to "ODBC;". To specify the machine data source, the SQL Server login ID, and password in the ODBC Connect Str property value, follow these steps:

1. Right-click the title bar of the SQL window and choose Properties from the popup menu to open the Query Properties sheet with default values for all properties (see Figure 26.40).

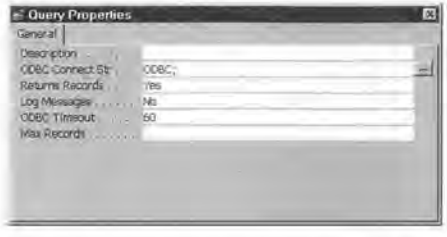


FIG. 26.40 The Query Properties sheet with default values for SQL pass-through queries.



2. Position the caret in the ODBC Connect Str text box and click the builder button to open the Select Data Source dialog. Click the Machine Data Source tab and select the data source for the query (see Figure 26.41).

3. Click OK to open the SQL Server Login dialog. Type your Login ID and Password (see Figure 26.42), then click OK. If you want to include your password in the ODBC connect string, click Yes when the Connection String Builder message box appears (see Figure 26.43).

Oracle Ex. 1009, p. 955
Oracle v. DeBellis
IPR2024-00221

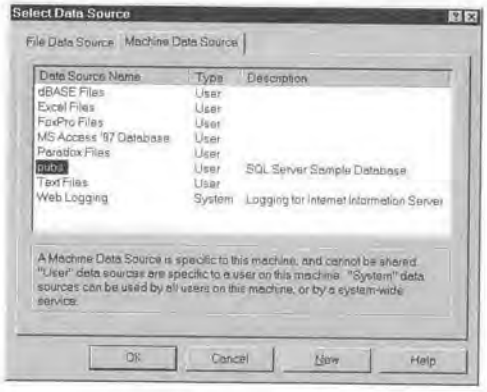


FIG. 26.41 Using the Select Data Source dialog to add the data source (DSN=) and database (DATABASE=) entries to the ODBC Connect Str property.



FIG. 26.42 Using the SQL Server Login dialog to add the Login ID (UID=) and Password (PWD=) entries to the ODBC Connect Str property value.



FIG. 26.43 Confirming inclusion of your password in the ODBC Connect Str property value.

Caution

Including your password, which appears in plain text, in the ODBC Connect Str property value is a breach of security. Any user with the retail version of Access and Read Design permission for the database can open the query and read your SQL Server password. The security breach is less serious with the runtime version of Access because users can't open objects in Design mode.

4. The final ODBC Connect Str value appears as shown in Figure 26.44. Close the Query Properties sheet, and choose File, Save to save your query. Then click the Run button to execute your SQL pass-through query.



Oracle Ex. 1009, p. 956
Oracle v. DeBellis
IPR2024-00221



FIG. 26.44 The Query Properties sheet with the final ODBC Connect Str value, including an empty password for the system administrator (sa) account.

Logging ODBC Queries

You can trace the execution of ODBC operations by selecting the appropriate option on the Tracing page of the ODBC Data Source Administrator, as shown in Figure 26.16 in the earlier section "Using the ODBC Administrator to Create a SQL Server Data Source." The ODBC API function call for SQL pass-through queries is `SQLExecDirect()`. Figure 26.45 shows WordPad displaying the `SQLExecDirect()` function call in the Sql.log file created by the preceding SQL pass-through query.

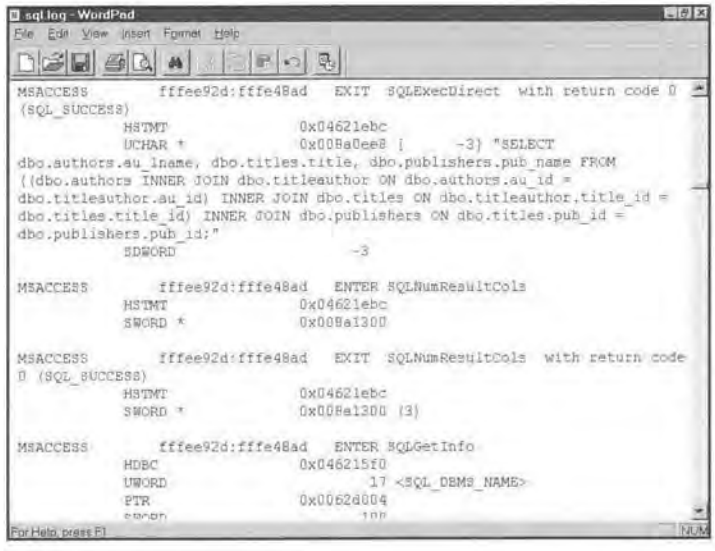


FIG. 26.45 A few of the ODBC API function calls that result from executing a SQL pass-through query.

Oracle Ex. 1009, p. 957
Oracle v. DeBellis
IPR2024-00221

Note

SQL pass-through queries bypass the query parser of the Jet database engine, but Access SQL and ANSI SQL-92 `INNER JOIN` statements are not accepted by version 4.x of SQL Server. (SQL Server 4.x uses `WHERE` clauses to create joins.) The Microsoft ODBC driver for SQL Server 4.x traps the `INNER JOIN` statement and converts (or *escapes*, in ODBC terminology) the Access/ANSI SQL statements to an equivalent SQL `WHERE` clause. SQL Server 6.5 accepts `INNER JOIN` syntax. As an example, you can copy the SQL pass-through statement from the Query window (without the trailing semicolon), paste the query into ISQL/w, SQL Server 6.5's query tool, and then execute the query directly against SQL Server 6.5.

Exporting Access Tables to a Client/Server Database

Creating the definition of tables in a client/server database can be a very lengthy process, especially if you need to write SQL DDL statements in ISQL/w to create the tables. SQL Server's SQL Enterprise Manager simplifies the process of creating new client/server tables. It's even easier, however, to export your existing Access tables to a database in SQL Server. When you export Access tables to SQL Server, you ensure that the SQL Server table's field data types correspond to the data types of your Access table.

Using Access's Built-In Export Feature

To export a table from Northwind.mdb to the pubs database, follow these steps:



1. Open Northwind.mdb. In the Tables page of the Database window, select the table you want to export to the SQL data source.
2. Choose File, Save As/Export, then click OK when the Save As dialog with the To an External File or Database option selected appears to display the Export dialog. Select ODBC Databases( ) in the Save As Type drop-down list. The Export dialog appears to confirm your selection (see Figure 26.46). You can change the SQL Server table name, if you want, but you don't need to add the dbo. prefix for SQL Server tables. Click OK to display the Select Data Source dialog.



FIG. 26.46 Confirming the export of a table to SQL Server.

Oracle Ex. 1009, p. 958
Oracle v. DeBellis
IPR2024-00221

3. Double-click the data source for the database to which you want to export the table—pubs, in this case—to display the SQL Server Login dialog.
4. Enter your user ID and password, then click OK to export the table to the server database.

The status bar shows the progress of exporting the table to the server. Exporting an Access table with a large number of records is not as fast as importing the content of the table to SQL Server from an ASCII text file with a bulk copy program (BCP). However, exporting the table is much faster when the table contains fewer than about 10,000 records.

Figure 26.47 shows SQL Server 6.5's SQL Enterprise Manager displaying the structure of the Orders table of Northwind.mdb exported to the pubs database. When you export a table containing an Access AutoNumber field, the field data type is converted to SQL Server's int data type, which is equivalent to the Access Long Integer field data type. The export process does not add SQL Server's identity attribute, the equivalent of Access's AutoNumber, Increment attribute, to the field.

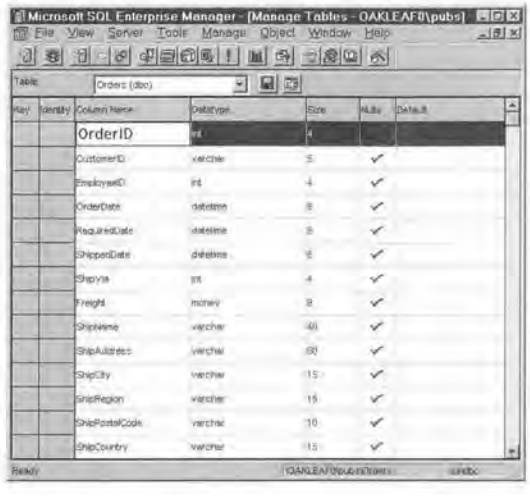


FIG. 26.47 SQL Enterprise Manager displaying the structure of the exported Orders table.

Note

When you export an Access table to SQL Server, the indexes, validation rules, default values, and other properties of the table are not exported. You need to use SQL Object Manager or a similar tool to add indexes and other properties to the new server table. Alternatively, you can use SQL Enterprise Manager's Manage Tables feature to add the indexes and other properties. The Access 97 Upsizing Wizard, the subject of the next section, automatically adds Access indexes, validation rules, and other properties to the SQL Server database.

Oracle Ex. 1009, p. 959
Oracle v. DeBellis
IPR2024-00221

Using the Access 97 Upsizing Wizard

The Access 97 Upsizing Wizard, which is almost identical to the Access 95 Upsizing Wizard, makes it quick and easy to upsize an entire Access application to the use of tables attached from SQL Server 6.5. You can download the Access 97 Upsizing Wizard at no charge from **http://www.microsoft.com/msaccess/**. Once you install the Wizard library (Wzcs97.mde) with the Access Add-In Manager, you can upsize an Access 97 database to SQL Server 4.21, 6.0, or 6.5.

> **Note**
>
> The Access 97 Upsizing Wizard includes the Upsize to SQL Server and SQL Server Browser add-ins. These add-ins work only with Microsoft SQL Server because they rely on Transact-SQL syntax for many operations and specify SQL Server-specific field data types. You cannot use the Upsizing Wizard or the SQL Server Browser with other client/server RDBMSs, such as Oracle or Informix. The SQL Server Browser add-in is the subject of the section that follows.

To test the Access 97 Upsizing Wizard with a copy of the Northwind Traders sample database, follow these steps:

1. Create a SQL Server 6.x database, nwind for this example, with 5M of available space. If the SQL Server DBA creates the database for you, make sure you have full permissions (database owner) for the database. Make sure to set the database's Truncate Log on Checkpoint option to avoid creating a large log file.
2. Create a user or system data source, Nwind, specifying the newly created database (nwind). Make sure that the Convert OEM to ANSI Characters check box is cleared. Also verify that the Don't Trace options is selected in the Tracing Page of the ODBC Data Source Administrator.

   > **Caution**
   >
   > If you specify the All the Time or One Time Only tracing options, operation of the Upsizing Wizard slows dramatically because tracing writes a very large number of entries to Sql.log. If you experience an extraordinary amount of disk activity during the wizard's operation, it's likely that you have specified one of these two tracing options. Tracing the operation of the Upsizing Wizard is likely to result in a Sql.log file of 30M or greater.

3. Launch Access, if necessary, and compact Northwind.mdb to a new database, Nwind.mdb. Upsizing modifies the database, so don't use the original version of Northwind.mdb for this example.
4. Choose Tools, Add-Ins, Upsize To SQL Server to launch the Upsizing Wizard.
5. In the first wizard dialog, select the Use Existing Database option (see Figure 26.48). Click Next to open the Select Data Source dialog.

Oracle Ex. 1009, p. 960
Oracle v. DeBellis
IPR2024-00221



FIG. 26.48 Specifying an existing SQL Server database for the upsizing process.

6. Select your new data source (Nwind) in the Machine Data Sources page of the Select Data Source dialog (see Figure 26.49). Click OK to log into SQL Server.

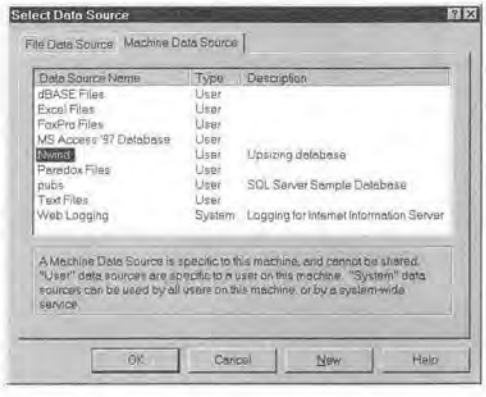


FIG. 26.49 Selecting the user data source for the upsizing operation.

7. After you log into the nwind database, the wizard displays an Available Tables list of the database. Click the >> button to move all of the entries to the Export to SQL Server list (see Figure 26.50). Click Next to continue.

8. Accept the default values offered by the wizard for table attributes and data options. Mark the Attach Newly Created SQL Server Tables and Save Password and User ID with Linked Table check boxes (see Figure 26.51). Click Next to open the last Upsizing Wizard dialog.

Oracle Ex. 1009, p. 961
Oracle v. DeBellis
IPR2024-00221



FIG. 26.50 Specifying all tables to be upsized to the SQL Server database.

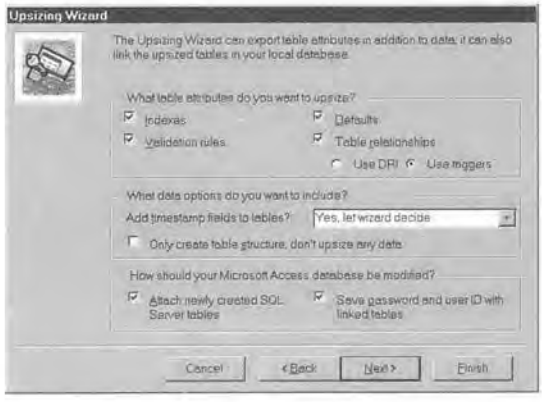


FIG. 26.51 Setting options for the upsizing the tables and modifying the existing database.

9. If you want to create an Access report to document the upsizing process, mark the Create Upsizing Report check box (see Figure 26.52). Click Finish to begin the upsizing operation.

10. The Upsizing Wizard displays a progress bar with messages describing the status of the upsizing process (see Figure 26.53). Click OK when the message box advises that upsizing is complete. If you specified an Upsizing report in step 9, the report opens in Preview mode.

Oracle Ex. 1009, p. 962
Oracle v. DeBellis
IPR2024-00221

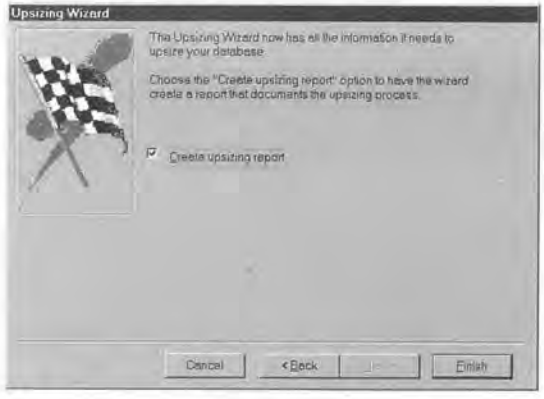


FIG. 26.52 The final dialog of the Upsizing Wizard.



FIG. 26.53 The dialog that displays the progress of the upsizing operation.

The Upsizing Wizard renames the existing tables of your database to *TableName*_local, then renames the table links from *TableName*_remote to *TableName*, taking the place of the original tables. Figure 26.54 shows the appearance of the Database window after upsizing is complete. Once you verify that there are no problems with your application, you can delete the *TableName*_local tables and compact the database to save disk space.



FIG. 26.54 The Database window with the new linked tables and the original tables renamed to *TableName*_local.

The Upsizing Wizard adds SQL Server timestamp fields to tables with OLE Object or Memo fields (Categories and Employees), as well as to tables with composite primary keys (Order Details). Figure 26.55 shows the Categories table in Design view with an

Oracle Ex. 1009, p. 963
Oracle v. DeBellis
IPR2024-00221

added upsize_ts timestamp (Binary) field. Timestamp data is used by SQL Server to resolve multiple, simultaneous edits to a single record. Unlike the table export operation, described in the preceding section, the Upsizing Wizard adds the SQL Server 6.5 primary key and identity attributes to primary key table fields of the Access AutoNumber field data type. These attributes appear as a primary key icon and check mark in the SQL Enterprise Manager's Manage Tables window's Key and Identity columns (see Figure 26.56).

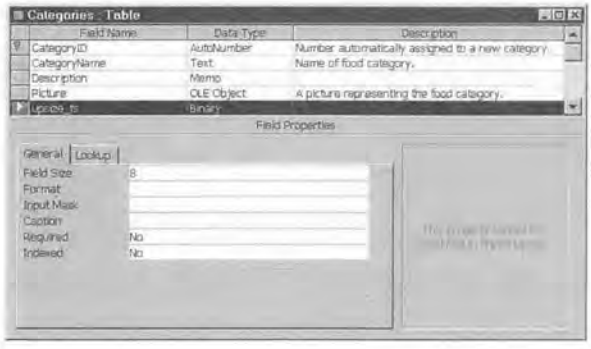


FIG. 26.55 The link to the exported Categories table in Design view.

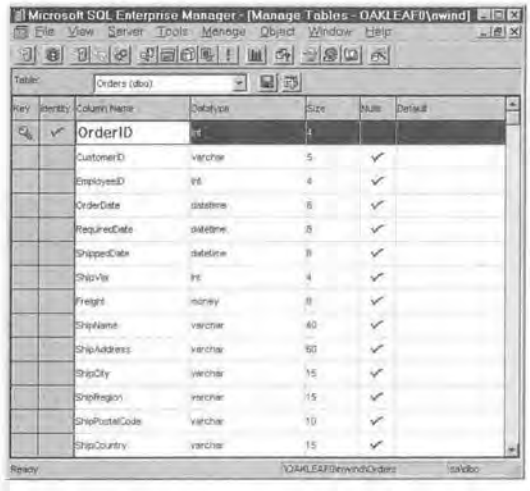


FIG. 26.56 The exported Orders table with SQL Server primary key and identity attributes set on the OrderID column.

Using the SQL Server Browser Add-In

Access 97's Upsizing Wizard library includes the new SQL Server Browser utility that you can use to view, add, and/or modify the properties of SQL Server tables, views, default values, rules, and stored procedures. To give the SQL Server Browser a test drive with the nwind database, follow these steps:

Oracle Ex. 1009, p. 964
Oracle v. DeBellis
IPR2024-00221

1. Choose Tools, Add-Ins, SQL Server Browser to open the Select ODBC Data Source dialog.
2. Select the Nwind data source and click OK. Log on to SQL Server to display the main Browser window for the nwind database (see Figure 26.57).



FIG. 26.57 The main window of the SQL Server Browser add-in displaying the tables in the nwind database.

3. To display the records of a table in a datasheet view, double-click the table item in the list, or select the table and click the Open button.
4. Select the Orders table and click the Design button to display the field and table properties (see Figure 26.58). SQL Server field data types, such as `int`, `datetime`, and `varchar` appear in the Data Type column. Name of SQL Server triggers that the Upsizing Wizard creates to enforce referential integrity appear in the Table Properties pane.

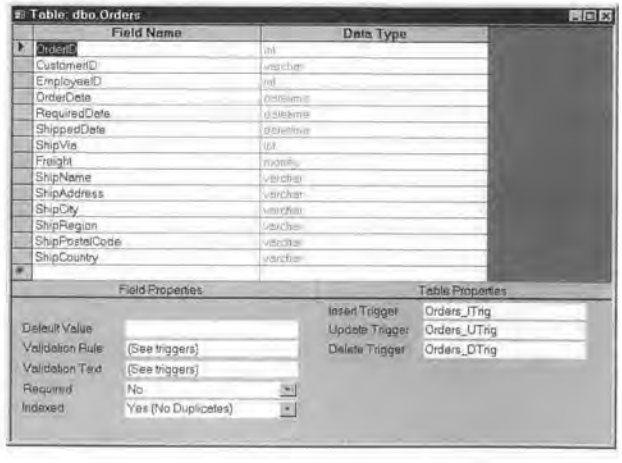


FIG. 26.58 The SQL Server Orders table in the SQL Server Browser's Table Design window.

Oracle Ex. 1009, p. 965
Oracle v. DeBellis
IPR2024-00221

Note

Triggers are the most common method of enforcing referential integrity of SQL Server databases. *Triggers* are a set of stored procedures (similar to an Access `QueryDef` object) that execute immediately prior to performing an `INSERT`, `UPDATE`, or `DELETE` operation. SQL Server 6.x lets you replace triggers with SQL-92's declarative referential integrity (DRI) syntax. If you specified the DRI option in step 8 of the preceding section, the Upsizing Wizard doesn't create triggers.

5. Select one of the triggers in the Table Properties pane to activate the builder button. Click the Builder button to open the Triggers window (see Figure 26.59).



If you're experienced with writing SQL Server triggers, you can edit the trigger code and click Save to update the trigger in the nwind database. You also can delete the trigger by clicking the Drop Trigger button.



FIG. 26.59 The SQL Server Browser's Triggers window displaying the *INSERT* trigger for the Order table.

6. Click Close to return to the Orders table design window, then close the table design window to return to SQL Server Browser's main window.

Note

The nwind table created by the Upsizing Wizard doesn't include SQL Server views or stored procedures. If you created the pubs data source in the section "Using the ODBC Data Source Administrator to Create a SQL Server Data Source" earlier in this chapter, you can display the view (titleview) and several stored procedures included in the pubs sample database. To open another SQL Server database, close SQL Server Browser's main window and restart the Browser Add-In.

Oracle Ex. 1009, p. 966
Oracle v. DeBellis
IPR2024-00221

Chapter 27

Replicating Access Databases

Laptop and notebook computers now comprise a substantial percentage of PCs running Windows, and the market share of portable PCs is expected to increase during the late 1990s. Microsoft designed Windows 95 specifically to accommodate the needs of mobile computing by adding features such as hot-swapping of PC Card (formerly PCMCIA) adapters and hot-docking to connect portable PCs to stationary keyboards and displays (called *docking stations*). The growing number of home PCs that are used part-time for business purposes also fits the mobile computing pattern. A large percentage of portable and home PC users connect to office networks by modem using Windows 95's Dial-Up Networking client and Windows NT Server's Remote Access Services (RAS) or the Windows 95 Dial-Up Networking Server included with Microsoft Plus!. Using RAS or Dial-Up Networking lets mobile computer users update the current versions of files on the server or their office PC.



If you don't have a RAS or Dial-Up Networking connection to a server or your office PC, Windows 95 provides a Briefcase feature to synchronize the contents of multiple copies of a file. This chapter explains the principles of Briefcase synchronization and how Access 97 synchronizes updates to table data using database replicas stored in Windows 95 Briefcases. Briefcase synchronization was new in Access 95, and enabled users to replicate entire databases. Access 97's new features extend Briefcase replication to include the ability to replicate selected portions of a database. Access 97's Briefcase replication lets you create database replicas on a local disk drive, on a disk drive connected to your computer through a local-area network, or to FTP locations reached through the Internet or an intranet.

Note

The My Briefcase icon appears on the Windows 95 or Windows NT 4.0 desktop only if you installed Windows by using the Portable option, or if you installed Windows using the Custom option and then specified that Briefcase be included. Microsoft Office 97 Professional Edition

(continues)

Oracle Ex. 1009, p. 967
Oracle v. DeBellis
IPR2024-00221

(continued)

also includes Briefcase replication. To install Briefcase replication with Microsoft Office, you must choose a Custom installation and specifically include Briefcase replication as part of your installation. If you didn't install Briefcase and want to try this chapter's examples of Briefcase synchronization, launch Control Panel's Add/Remove Programs feature and add the Briefcase files from your Microsoft Office 97 CD-ROM.

Understanding the Principles of Briefcase Replication

To synchronize updates to Access 97 databases, you must create a *replica set*. An Access 97 replica set consists of one *design-master* replica, to which you can make design changes, and one or more replicas that do not support Design mode. One (and only one) design-master and all other replicas comprise an Access *replica set*. The design-master replica is the .mdb file from which all Briefcase replicas are created. In most cases, the .mdb file designated the design-master replica is the original application or, more often, the original data .mdb file stored on your local fixed disk or shared from a file server. You can designate one of the Briefcase replicas as the design-master, if desired.

OLE (Object Linking and Embedding) 2+ provides a set of functions, called *OLE interfaces*, that support operations such as embedding or linking objects within container documents. The chapters in Part V, "Integrating Access with Other Office 97 Applications," describe the basic features of OLE 2.1 as they apply to Access 97. Windows 95's implementation of OLE 2.1 includes an additional OLE 2.1 interface, not discussed in Part V, that adds the following Briefcase functions:

- *Binding a reconciliation handler to an application.* A *reconciliation handler* is a set of methods, specific to the type of document created by the application, to reconcile accumulated changes to replicas of a particular document file made at different times by multiple users. *Binding* is a Windows programming term that means attaching a process to an application when the application needs to use the process.



◀◀ See "The Registry Editor," p. 683

- *Tracking the contents of the Briefcase.* Access replica sets are identified by a unique number, called a *Globally Unique ID* (*GUID*, also called a *Universally Unique ID*, or *UUID*), created by Windows 95. Each member of a replica set uses the same GUID. GUIDs are automatically generated, 128-bit hexadecimal numbers that have an almost infinitesimal probability of duplication. (Access 97 provides the GUID data type for creating and manipulating GUIDs.) If you have explored the Windows 95 Registry with the Registry Editor (RegEdit.exe), you probably have seen a large number of entries with 32-character GUIDs. (The OLE Class ID, CLSID, entries in the Registry are GUIDs assigned to each OLE object class and interface by the

Oracle Ex. 1009, p. 968
Oracle v. DeBellis
IPR2024-00221

author of the object.) Only classes of documents that are supported by an OLE 2.1 application with a reconciliation handler are candidates for Briefcase synchronization.

- *Reconciling the contents of the Briefcase.* Reconciling Briefcase documents updates the original design-master file with the content of the replica file(s) in Briefcase(s) with accumulated changes. The updating process is relatively simple for a conventional OLE document, such as a Microsoft Word 97 or Excel 97 file that has built-in revision handling features, but it is a much more involved procedure for Access .mdb files, which may include a wide variety of different objects.

All participants in the Briefcase replication process must use Windows 95 or Windows NT 4.0. Prior versions of Windows NT Workstation and Server (version 3.55 and earlier) do not support Briefcase replication.

Creating and Updating a Briefcase Replica

To create a design-master and Briefcase replica of a document, follow these basic steps:

1. Drag the database's icon from Windows 95's Explorer (or any folder window) into Briefcase's window or onto its icon. (Access 97 also offers the Tools, Replication, Create Replica command to create a design-master replica.)
2. Create additional Briefcases containing replicas by copying and pasting My Briefcase to the desktop, and then renaming the copy. Other database users update the Briefcase replica(s).
3. Open My Briefcase, and choose Briefcase, Update All. All replicas of the same replica set in all Briefcases on the desktop update the design-master replica.

Figure 27.1 illustrates the steps involved in Briefcase replication of Access 97 databases.

Once the replica is in the Briefcase of the host computer, mobile users can update the Briefcase replica on the host or on the guest (mobile) computer by the following four methods:

- *Network connection.* If you can connect your guest computer to the host computer's Briefcase, you can update the Briefcase replica directly. If the replica is opened in shared mode (the default), multiple users can update the Briefcase replica simultaneously. Unless there is a particular reason to use the Briefcase replica, networked users should connect directly to and update the design-master (original) .mdb file.
- *Direct cable connection.* You can use a serial cable (called a *null-modem cable*) between the serial (COM) ports of host and guest computers to make the host Briefcase accessible to the guest computer. If you have an ECP-enabled parallel port, you can use an ECP (Extended Communication Protocol) cable to connect the parallel (LPT) ports of the host and guest PCs. A parallel ECP connection is much faster than a serial connection. You can use a Universal Cable Module (UCM) cable to connect different types of PC parallel ports.

Oracle Ex. 1009, p. 969
Oracle v. DeBellis
IPR2024-00221

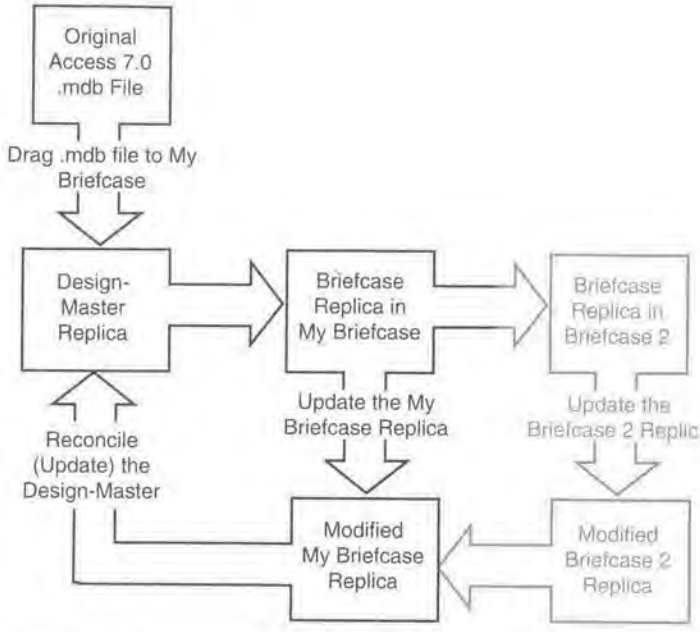


FIG. 27.1 Creating an Access replica set and updating the design-master replica with updates made to Briefcase replicas.

- *Dial-up modem connection.* If you've installed on the host computer the Dial-Up Networking Server component included on the Microsoft Windows 95 Plus! Pack, your guest computer can connect to the Briefcase on the host computer and update the Briefcase replica.
- *Disk.* You can move the Briefcase replica on the host computer to a disk, and then move the Briefcase replica from the disk to a Briefcase on the desktop of the guest PC. After you make changes on the guest PC, you reverse the process and move the updated files from the disk to the host PC's Briefcase. Most Briefcase users are likely to employ the disk method, so the example in this chapter uses disks for synchronizing changes to Access tables.

> **Note**
>
> To use the network, direct cable, or dial-up modem connection method, you need to have the File and Printer Sharing services for either the Microsoft or Novell networks installed.

Creating the Contact Management Application for Replication

Contact management is one of the most common applications for Briefcase replication. Salespersons take their laptops on sales calls and record important information relating to sales contacts. To use the Database Wizard to create a new Contact Management database, ContactManager.mdb, follow these steps:

Oracle Ex. 1009, p. 970
Oracle v. DeBellis
IPR2024-00221

1. Launch Access and select Database Wizard from the opening window or, if Access is open, choose File, New Database to open the New dialog (see Figure 27.2); then click the Database tab.

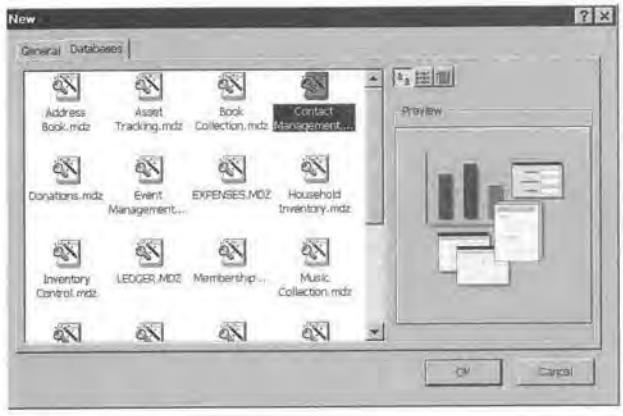


FIG. 27.2 The New dialog displaying icons for databases you can create with the Database Wizard.

2. Double-click the Contact Management icon in the Database list to create a new database based on the Contact Management template. The File New Database dialog opens.

3. Navigate to the directory for the new database and give the file a name, such as **ContactManager.mdb** (ContactManagement1.mdb is the default file name), as shown in Figure 27.3. The location of the file is not important; in this example, the new database is located in C:\My Documents. Click Create to have the Database Wizard generate the new database.



FIG. 27.3 Specifying a name and location for the new Contact Management database.

Oracle Ex. 1009, p. 971
Oracle v. DeBellis
IPR2024-00221

4. When the first Database Wizard screen appears, click the Next button to display the tables and fields dialog. Mark the Yes, Include Sample Data check box to add the sample data to the tables (see Figure 27.4). Accept the remainder of the dialog's defaults and click the Next button.

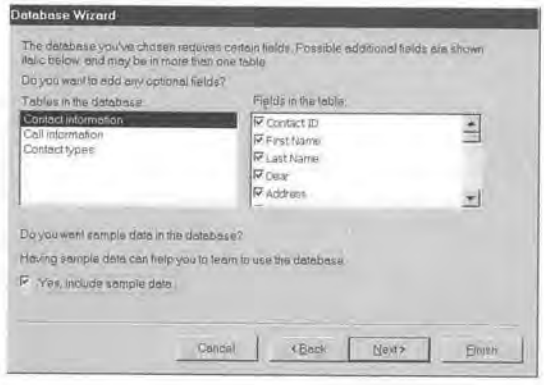


FIG. 27.4 Specifying addition of sample data to the Contact Management database.

5. In the first style selection dialog, select the style you like for forms and click the Next button. This example uses the International style.

6. In the second style selection dialog, select the report style you want and click the Finish button to accept the remainder of the Database Wizard's defaults. This example uses the Corporate style.

Access 97 displays a progress dialog as it creates the new database; after a few minutes of intense disk activity, the Database Wizard completes the task of building the ContactManager.mdb database, and the Contact Management Switchboard appears (see Figure 27.5).



FIG. 27.5 The Switchboard form of the Contact Manager database.

Oracle Ex. 1009, p. 972
Oracle v. DeBellis
IPR2024-00221

Replicating only the tables of the database is more efficient than replicating the entire database, so follow these steps to split the tables of ContactManager.mdb into ContactData.mdb and create the links to the tables:

1. With ContactManager.mdb open and the Main Switchboard form closed, choose Tools, Add-Ins, Database Splitter to open the Database Splitter dialog.
2. Click the Split Database button to open the Database Splitter's Save As dialog. Maneuver to the same folder that contains ContactManager.mdb, if necessary, and type **ContactData.mdb** in the File Name text box (see Figure 27.6).



FIG. 27.6. Naming the "back-end" database to contain the Contact Manager tables.

3. Click the Split button to complete the splitting and linking process. The Tables page of the Database window appears as shown in Figure 27.7.



FIG. 27.7 The tables linked from ContactData.mdb to ContactManager.mdb.

4. Choose Tools, Add-Ins, Linked Table Manager to display the links from ContactManager.mdb's application objects to the tables of ContactData.mdb (see Figure 27.8). Close the Linked Table Manager.

Caution

Do not password-protect Access .mdb files that you intend to use with Briefcase replication. Providing a password requires opening Access. All reconciliation operations operate behind the scenes (using the Jet 3.5 database engine directly) and do not launch Access. You receive an error message if you attempt to reconcile updates to a password-protected replica.

Oracle Ex. 1009, p. 973
Oracle v. DeBellis
IPR2024-00221

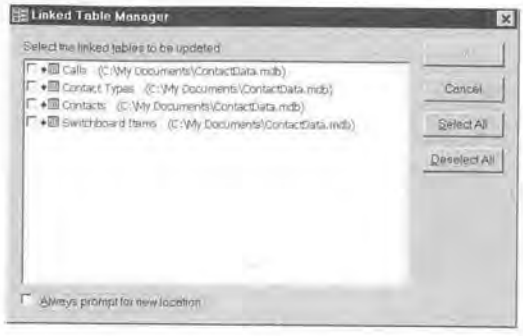


FIG. 27.8 Verifying the links between the application and data .mdb file for the Contact Management application.

Creating a Replica Set

To create a design-master and a Briefcase replica (a replica set) from ContactData.mdb, follow these steps:

1. Close Access and launch Explorer. Open the Explorer folder that contains ContactData.mdb.
2. Move Explorer's window, if necessary, to expose the My Briefcase desktop icon. Drag ContactData.mdb's icon to the desktop and drop the icon on the My Briefcase icon. (If My Briefcase is open, you can drop the icon in My Briefcase's window.) My Briefcase begins to create a replica set of ContactData.mdb.
3. After a brief period of disk activity, the message shown in Figure 27.9 appears. Click Yes to continue.

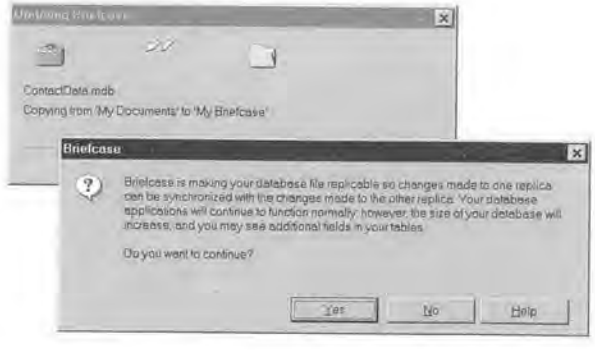


FIG. 27.9 The message requesting confirmation for creating the Access replica set.

4. A second message appears offering you the option to make a backup copy of your replicated database (see Figure 27.10). Click No to continue because you don't need a backup copy of this example database. (Ordinarily, you create a backup copy of your database at this point.)

Oracle Ex. 1009, p. 974
Oracle v. DeBellis
IPR2024-00221

5. A third message appears that lets you select whether to make the Original Copy or the Briefcase Copy the design-master set (see Figure 27.11). Unless you plan to do off-site development work on your database, select the default (Original Copy) option. Ordinarily, you don't want users making changes to the design of your tables, especially if the tables are attached to an Access application. Click OK to complete the replication process.



FIG. 27.10 The message offering to create a backup copy of your design-master replica database.

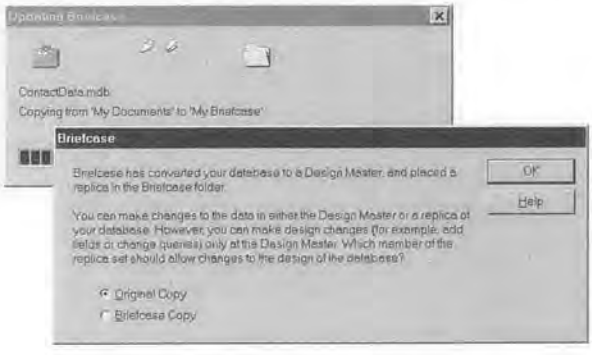


FIG. 27.11 The dialog to select whether the original or the Briefcase copy is the design-master replica.

6. Launch My Briefcase, if necessary. Your replica copy appears in My Briefcase's file list, as shown in Figure 27.12.

7. Right-click ContactData.mdb's icon and choose Properties from the popup menu to open the Properties window. Click the Update Status tab to display the synchronization status. The original and replica of ContactData.mdb are Unmodified and the status is Up-to-Date because you haven't yet made changes to either replica (see Figure 25.13). Click Close to return to Briefcase.

Oracle Ex. 1009, p. 975
Oracle v. DeBellis
IPR2024-00221



FIG. 27.12 The Briefcase copy of ContactData.mdb in My Briefcase's file list.

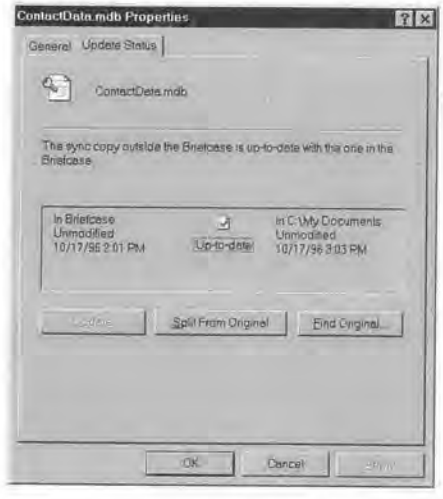


FIG. 27.13 The Update Status properties page of the of ContactData.mdb Briefcase replica.

8. To view the tables of the replica, double-click ContactData.mdb's Briefcase icon. Access launches with the replica copy of ContactData.mdb active. `Replica` appears in the title bar of the Database window, indicating you're working with a replica of the original database; replicated tables in the database are indicated with a special icon (see Figure 27.14).

9. Choose Tools, Options and click the View tab. Mark the Hidden Objects and System Objects check boxes of the Show frame and click OK to close the Options sheet. Note that many new hidden tables for replication management, such as MSysExchangeLog, MSysReplicas, and MSysTombstones, have been added to ContactData.mdb (see Figure 27.14).

Oracle Ex. 1009, p. 976
Oracle v. DeBellis
IPR2024-00221

10. Choose Tools, Options again, and clear the Hidden Objects and System Objects check boxes. Then click OK to close the Options sheet. Close the Briefcase copy of ContactData.mdb.

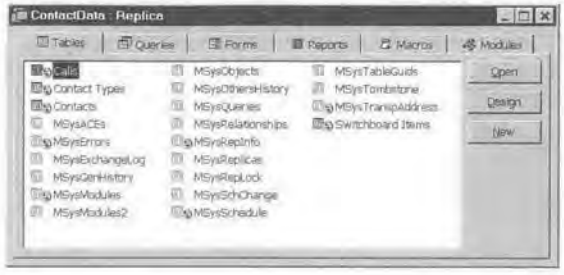


FIG. 27.14 The Tables page of the Database window displaying hidden and system tables, including hidden and system tables for replication management.

Synchronizing the Design-Master and Briefcase Replicas

Modifying data or options in either the Briefcase replica or the design-master sets the status of the file to Modified. If either of the replicas has been modified since the last reconciliation, either the Briefcase or the design-master replica is a candidate for updating. Follow these steps to verify the preceding statements:

1. Open Briefcase, if necessary. Right-click ContactData.mdb's icon, and select Properties from the popup menu. Click the Update Status tab of the Properties window. The status of the Briefcase copy is Modified, as shown in Figure 27.15, although you didn't change any data in the preceding steps. Changing the View options in preceding steps 9 and 10 set the "Dirty" flag for the Briefcase copy, which determines if the copy's status is set to Modified.

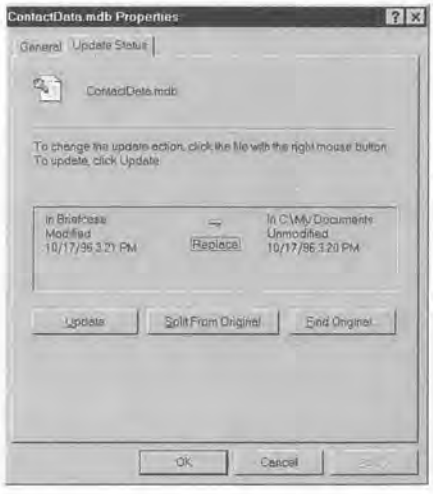


FIG. 27.15 The Update Status properties page of the modified Briefcase replica.

Oracle Ex. 1009, p. 977
Oracle v. DeBellis
IPR2024-00221

2. Click Update to update the design-master replica.
3. Open the original (design-master) copy of ContactData.mdb. The title bar of the Database window displays `Design Master`, and the replicated tables are marked with a special icon, as shown in Figure 27.16.



FIG. 27.16 The Tables page of the Database window for the design-master replica of ContactData.mdb.

4. Open the Calls table of ContactData.mdb and modify an entry, such as changing the date in the Call Date field of a record.
5. Open Briefcase, if necessary, and check the Update status using the procedure described in step 1. The original version now shows modified status and the Update button is enabled (see Figure 27.17). Click Update to propagate the change from the original (design-master replica) file to the Briefcase replica.

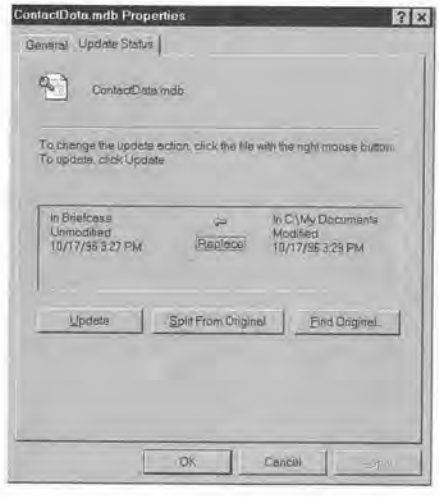


FIG. 27.17 Updating the Briefcase replica with changes made to the design-master replica.

Oracle Ex. 1009, p. 978
Oracle v. DeBellis
IPR2024-00221

Emulating Replica Set Reconciliation with Disks

As noted earlier in this chapter, most Access 97 users will use disks to replicate Briefcases and reconcile replica sets. You can emulate floppy disk Briefcase update operations on a single computer by following these steps:

1. Create a copy of your application .mdb file. For example, copy ContactManager.mdb and name the copy **ContactRemote.mdb**.
2. Right-click the My Briefcase icon and choose Copy from the popup menu. Right-click the desktop and choose Paste to add a copy of My Briefcase icon to the desktop. (The copy of My Briefcase is a backup.)
3. Insert a blank formatted disk in your A: drive. Open Explorer and drag the My Briefcase icon to Explorer and drop the icon on the A: drive icon. This step *moves* My Briefcase from the desktop to the A: drive.

> **Note**
>
> *Moving* the Briefcase is an important step in understanding the concept of a Windows Briefcase. You keep replicas of all the office files you want to update while on the road in My Briefcase. You take My Briefcase with you when you leave; a real briefcase cannot be two places at once, and neither can a briefcase's abstraction as a Windows desktop object. When you return, you move My Briefcase from the disk to the desktop, as described later in this procedure. If you share your Briefcase on a network, the real-world object corresponding to the Briefcase is an in-basket.

4. Now you need to change the links to ContactData.mdb to the copy of ContactData.mdb in the \My Briefcase folder on the disk to emulate disk operations with your guest computer. Open ContactRemote.mdb, and then choose Tools, Add-Ins, Linked Table Manager to open the Linked Table Manager's dialog.
5. Click the Select All button and mark the Always Prompt for New Location check box (see Figure 27.18). Click the OK button to open the Select New Location of Calls dialog.



FIG. 27.18 Selecting all links for refreshing by the Linked Table Manager Add-In.

Oracle Ex. 1009, p. 979
Oracle v. DeBellis
IPR2024-00221

6. Open the Look In drop-down list and select the A:\My Briefcase folder, which contains the replica of ContactData.mdb, as shown in Figure 27.19.



FIG. 27.19 Selecting A:\My Briefcase as the location of the ContactData.mdb file for linking.

7. Select ContactData.mdb and click the Open button to close the dialog and refresh the table links. The Linked Table Manager displays a message dialog indicating the successful refreshing of table links. Click OK to clear the message dialog; the Linked Table Manager's dialog appears as shown in Figure 27.20. Click Close to close the dialog.

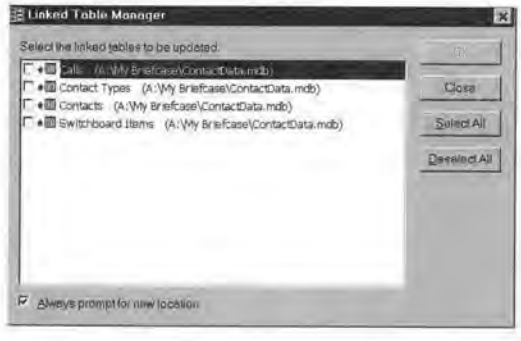


FIG. 27.20 Tables in ContactRemote.mdb linked to the My Briefcase replica of ContactData.mdb on the disk.

8. Open the Contacts form and change an entry, such as giving Nancy Davolio a promotion from Sales Representative to **Sales Director** (see Figure 27.21), and then close ContactRemote.mdb.

Oracle Ex. 1009, p. 980
Oracle v. DeBellis
IPR2024-00221



FIG. 27.21 Changing an entry in the Contacts form of the ContactRemote.mdb copy of the Contact Management application.

Operations using .mdb files stored on floppy disk are very slow; you need patience to perform steps 7 and 8. In real-world applications, you move the Briefcase from the disk to the desktop of your guest PC and attach the files from My Briefcase (or whatever you've named your Briefcase) to the desktop. When returning to the office, you move the Briefcase back to the disk.

9. Open Explorer and select the A: drive, as shown in Figure 27.22. Briefcase folders in Explorer are indicated by a small Briefcase icon. Drag the small My Briefcase icon in Explorer's window back to your desktop.

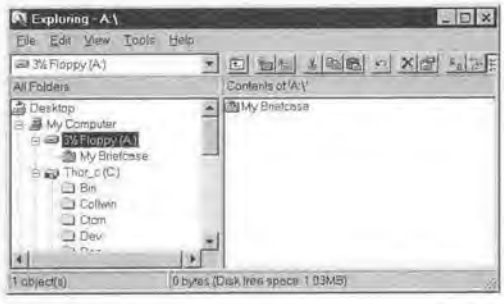


FIG. 27.22 Explorer displaying the My Briefcase folder on the floppy disk.

10. Open My Briefcase from the desktop and choose View, Details to display the location of the design-master copy to synchronize, the status of the replica set, and other file details, as illustrated in Figure 27.23.

Oracle Ex. 1009, p. 981
Oracle v. DeBellis
IPR2024-00221

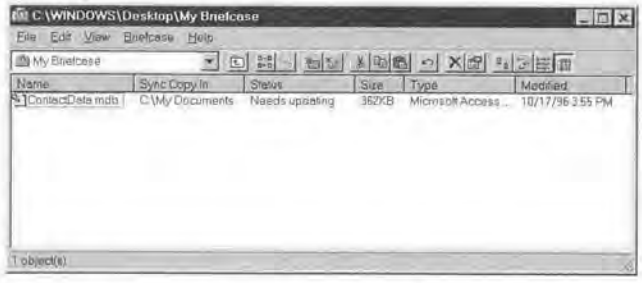


FIG. 27.23 My Briefcase's windows displaying details of the modified Briefcase replica.

11. Choose Briefcase, Update All to update the design-master .mdb. The Update My Briefcase dialog appears, as shown in Figure 27.24. Click Update to make the changes to the design-master copy of ContactData.mdb.

The Update All menu choice updates the design-master .mdb from all Briefcases on the desktop that are synchronized with the .mdb, not just the open Briefcase.



FIG. 27.24 The Update My Briefcase dialog that confirms reconciliation of the design-master replica with the updated Briefcase replica.

12. Reopen My Briefcase to check the status of your Briefcase replica, which now is Up-to-Date (see Figure 27.25).

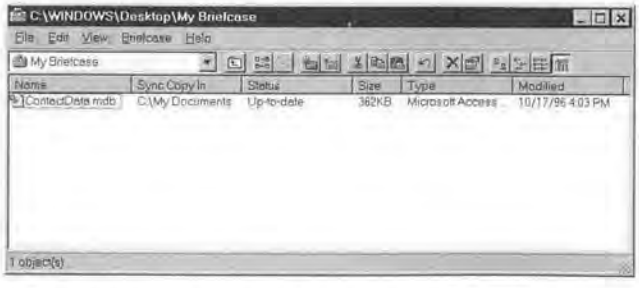


FIG. 27.25 My Briefcase's window displaying the status of an up-to-date replica set.

Oracle Ex. 1009, p. 982
Oracle v. DeBellis
IPR2024-00221

13. Close My Briefcase and delete the copy of My Briefcase on the desktop.

Tip

If you create multiple Briefcases to accommodate several mobile users who update Briefcase replicas using disks, try to organize the process so that you obtain the disks from all users at the same time. This allows you to use a single Update All operation to reconcile all changes for a given period simultaneously.

Replicating Part of a Database

Frequently, you might want to replicate only a portion of the data contained in a database, instead of replicating the entire contents of a database. For this reason, Microsoft has created the Partial Replication Wizard. By using the Partial Replication Wizard, you can create replica databases that have the same tables and structure as other databases in the replica set, but contain only a specific subset of the information stored in the database.

Partial database replicas provide three basic advantages:

- *Data restriction.* A partial replica enables you to restrict the data included in a particular replica database to only those records relevant to the particular use for which you prepare the replica. Using the Contact Management database you created in the preceding sections of this chapter as an example, there may be no reason or need for a salesperson to be able to view any other salesperson's contact information. In some cases, it might even be undesirable to have salespeople able to access the contact information of other salespersons. You could therefore use the Partial Replication Wizard to create replica databases for each salesperson, including only the contact information that belongs to that specific salesperson's contacts. Each salesperson would then use a partial replica containing only their own contact information, which could be synchronized to a full database for perusal by authorized managers.
- *Smaller size.* Because the partial replica contains less data (although it has the same structure), partial replicas tend to be smaller than the full database. Depending on how you choose the data included in the partial replica, a partial replica may be substantially smaller than the full database. This increases the likelihood that your partial replica will fit on a floppy disk, or on the typically smaller hard disks of laptop and notebook computers. The smaller size of a partial replica also means that you won't waste disk space storing data that isn't needed by a particular user, and that copying the Briefcase folder containing the partial replica will be faster.
- *Faster synchronization.* Because the partial replica contains fewer records than the full database, fewer data items have to be compared or exchanged in order to reconcile the two versions of the database. As a result, Briefcase updating operations with partial replicas tend to be faster than updating two full copies of the database.

Oracle Ex. 1009, p. 983
Oracle v. DeBellis
IPR2024-00221

Obtaining and Installing the Partial Replica Wizard

The Partial Replica Wizard isn't included in Microsoft Office 97 Professional Edition, nor is it included with the Access 97 stand-alone product. Instead, you must obtain the Partial Replica Wizard directly from Microsoft, by downloading it from its Web site.


On the Web

The following URL is the home page for Access 97 that provides links to pages from which you can download Microsoft add-ins for Access 97:

http://www.microsoft.com/msaccess

After downloading Partial.exe, follow these steps:

1. At a DOS prompt, type **Partial** and then press Enter; alternatively, display the Partial.exe file in the Windows Explorer and double-click it. Partial.exe is a self-extracting archive file, and will unpack itself into a Readme.txt file and a file named Wzprtl80.mde.
2. Copy Wzprtl80.mde into the folder in which Access 97 is installed, usually C:\Program Files\Microsoft Office\Office.
3. Start Access and open any database file. Choose Tools, Add-Ins, Add-In Manager. Access opens the Add-In Manager dialog.
4. Select the Partial Replica Wizard in the Available Add-Ins list, and then click Install. An X appears at the left of the Partial Replica Wizard entry to indicate that it is installed (see Figure 27.26).
5. Click Close to close the Add-In Manager dialog.

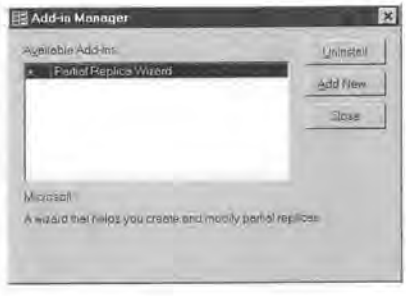


FIG. 27.26 Installing the Partial Replica Wizard using the Add-In Manager.

Using the Partial Replica Wizard

To use the Partial Replica Wizard, follow these steps:

1. Open any full replica of the database. It need not be the Design Master, but it must be a replica that contains the full data set.

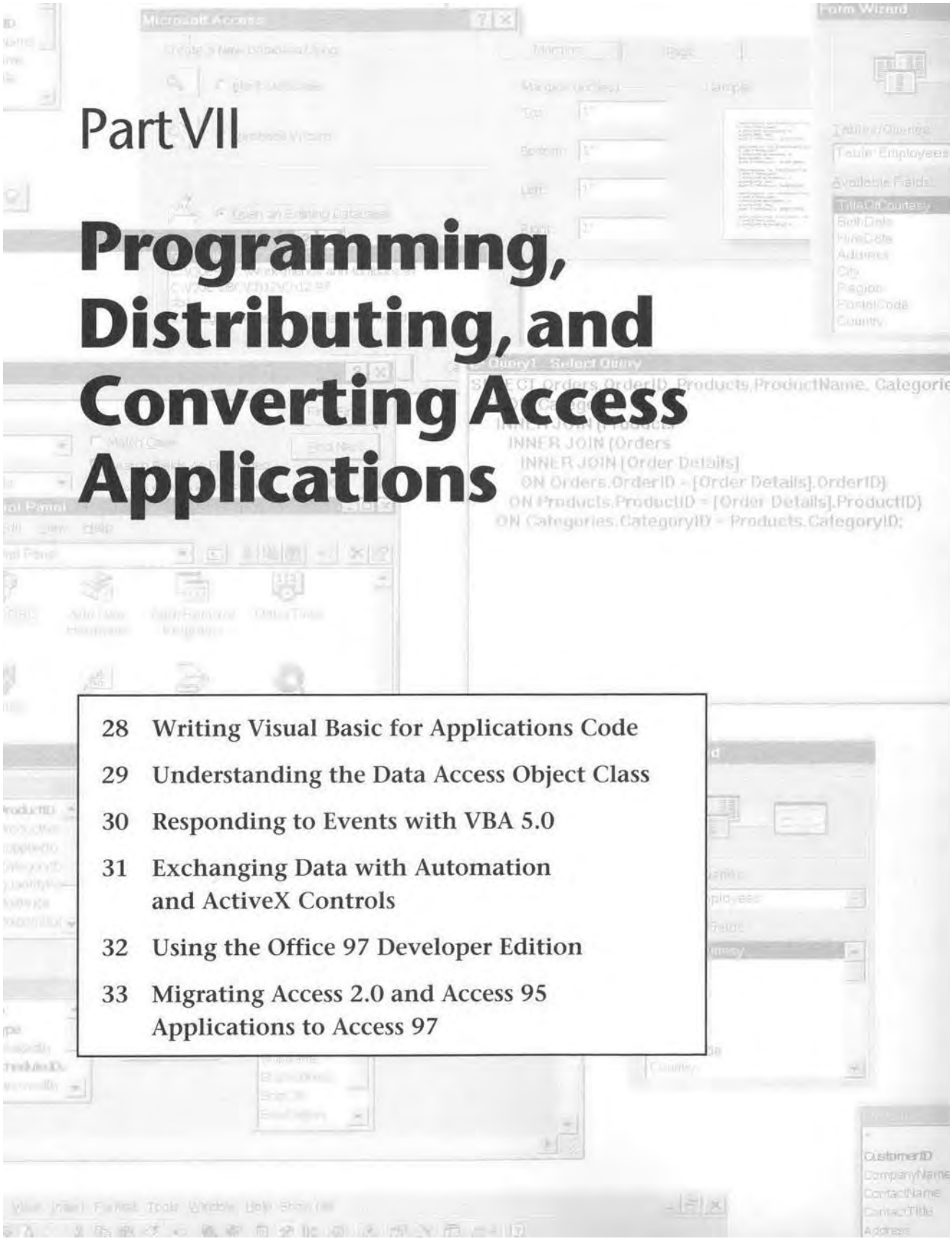
Oracle Ex. 1009, p. 984
Oracle v. DeBellis
IPR2024-00221

2. Select Tools, Add-Ins, Partial Replica Wizard. Access starts the Partial Replica Wizard. Follow the on-screen instructions to complete the partial replica.

Note

The Partial Replica Wizard was not available at the time this book was in preparation. A working version is promised for the release of retail Access 97. Make sure to check the Readme.txt file that comes with the Partial Replica Wizard for last-minute installation and operating instructions.

Oracle Ex. 1009, p. 985
Oracle v. DeBellis
IPR2024-00221



Part VII

Programming, Distributing, and Converting Access Applications

28 Writing Visual Basic for Applications Code

29 Understanding the Data Access Object Class

30 Responding to Events with VBA 5.0

31 Exchanging Data with Automation and ActiveX Controls

32 Using the Office 97 Developer Edition

33 Migrating Access 2.0 and Access 95 Applications to Access 97

Oracle Ex. 1009, p. 986
Oracle v. DeBellis
IPR2024-00221

Chapter 28

Writing Visual Basic for Applications Code

Most Access applications you create require you to write little or no Access Visual Basic for Applications (VBA) code. A few commercial Access applications rely primarily on macros rather than Access VBA code for automating applications. Starting with Access 95, however, Microsoft recommended that Access developers use VBA code instead of macros, with the clear implication that macros may not be supported in future versions of Access. (Access 97 does support macro operations, but the Microsoft documentation states that it does so primarily for backward compatibility.) Short VBA procedures using the `DoCmd` object usually are sufficient to provide the methods needed by simple applications to run queries, display forms, and print reports. The built-in functions of Access allow you to perform complex calculations in queries. You may want or need to use Access VBA code for any of the following reasons:

- To create user-defined functions (UDFs) that substitute for complex expressions you use repeatedly to validate data, compute values for text boxes, and perform other duties. Creating a UDF that you refer to by a short name minimizes potential typing errors and allows you to document the way your expression works.
- To write expressions that include more complex decision structures than allowed by the standard `IIf()` function (in an **`If...Then...Else...End If`** structure, for example), or to write expressions that need loops for repetitive operations.
- To perform transaction processing actions with the Access VBA equivalents of SQL `COMMIT` and `ROLLBACK` statements.
- To manipulate other applications' objects with Automation code. Using Automation is the primary subject of Chapter 31, "Exchanging Data with Automation and ActiveX Controls."
- To open more than one database in an application where attaching a table or using the SQL `IN` statement is not sufficient for your application.

Oracle Ex. 1009, p. 987
Oracle v. DeBellis
IPR2024-00221

- To provide hard-copy documentation for your application. If you include actions in Access VBA code, you can print the Access VBA code to improve the documentation for your application.
- To provide graceful error-handling if something goes wrong in your application. With Access VBA code, you can closely control how your application responds to errors such as missing forms, missing data, incorrect values entered by a user, and other problems.

This chapter describes Access VBA, introduces you to Access VBA modules and procedures, shows you how to use the Module window to enter and test Access VBA code, and helps you start writing user-defined functions. The chapter also includes examples of Access VBA programs.

Introducing Access VBA

Several years ago, Bill Gates, the founder and chairman of Microsoft Corporation, stated that all Microsoft applications using macros would share a common macro language built on BASIC. BASIC is the acronym for Beginner's All-Purpose Symbolic Instruction Code, originally developed at Dartmouth College. Gates' choice of BASIC is not surprising when you consider that Microsoft was built on the foundation of his BASIC interpreter that ran in 8K on the early predecessors of the PC. Gates reiterated his desire for a common macro language in an article that appeared in *One-to-One with Microsoft* in late 1991.

Before Access 1.0 was released, the results of Gates' edict were observed in only one Microsoft product, Word for Windows. If you have created Microsoft Word macros or just made minor changes to macros you have recorded, you will find that Access VBA is similar to WordBasic. With the arrival of Excel 5.0, Visual Basic, Applications Edition (more commonly known as Visual Basic for Applications, or VBA), became the *lingua franca* for programming Microsoft's productivity applications.

Access 95 joined Excel 5+, Project 4+, and Visual Basic 4.0 in implementing VBA as Microsoft's "common macro language." With the advent of Microsoft Office 97, Microsoft Word and Excel now share a common VBA programming environment, and even PowerPoint uses VBA to program presentations. Access 97 does not yet contain the same VBA programming environment used by Excel 97 and Word 97, but otherwise Access 97 fully supports VBA.

Note

Each application that uses VBA automatically adds references for its built-in application objects to the language. Thus, each application has its own "flavor" of VBA. As an example, the Access `DoCmd` object used to execute actions in code is specific to the Access "flavor" of VBA. For this reason, this book uses the term *Access VBA* to describe VBA with the standard set of Access object references added to the root language.

Oracle Ex. 1009, p. 988
Oracle v. DeBellis
IPR2024-00221

VBA is a real programming language, not a "macro language;" Access has its own macro language. You can expect Access VBA to replace Access macros gradually—Microsoft recommends that you develop applications in Access by using Access VBA instead of macros. You create the preferred equivalent of macros with Access VBA functions and subprocedures. Although you can execute Access VBA subprocedures directly from an open code module, you more typically execute Access VBA subprocedures by specifying the name of the subprocedure as the value of a specific event. (Chapter 30, "Responding to Events with VBA 5.0," explains how to use VBA subprocedures as event-handlers.) You typically execute Access VBA functions by using them in calculated controls in forms and reports, in the Validation Rule property of a field or table, or by calling the function from within a VBA subprocedure.

Access VBA Compared with xBase, PAL, and Visual VBA

Programmers experienced with dBASE or other DOS xBase dialects, such as Clipper and FoxBase, will find that Access VBA uses many of the same xBase keywords or minor variations of familiar xBase keywords. Borland's PAL is related to xBase, so PAL and ObjectPAL programmers will also find the translation of PAL to Access VBA keywords straightforward. A large number of DOS-based RDBMS applications running on PCs remain, many of which are likely to be converted to 32-bit Access applications running under Windows 95 or Windows NT. Therefore, a comparison of Access VBA and DOS database applications continues to be warranted.

User-defined functions are almost identical in structure in Access VBA and the DOS versions of xBase and PAL. This similarity in code structure ends, however, with user-defined functions. Writing code for Windows applications in general and for Access in particular requires an entirely different code structure. Applications for DOS RDBMSs use top-down programming techniques, which means that a main program calls (executes) other subprograms or procedures that perform specific actions. The user of the application makes choices from a menu that determine which subprograms or procedures are invoked. Menus that let the application remain idle are enclosed within `DO WHILE...ENDDO` loops while the user decides what menu command to choose next. Your code is responsible for all actions that occur while the application is running.

The situation with Windows and Access differs from that of top-down programming. Access itself is the main program, and Windows is responsible for many functions that you must code in DOS RDBMS languages. This fact is a blessing for new programmers because Windows and Access simplify the development of complex applications. However, contributions from Access and Windows are a curse for top-down DOS programmers, because Access and Windows require an entirely new approach to writing RDBMS code.

The guiding principles for DOS RDBMS developers who write Access VBA applications are the following:

- Don't try to write a great deal of Access VBA code during the development stage of your first applications. Create all forms, reports, and queries for your application

Oracle Ex. 1009, p. 989
Oracle v. DeBellis
IPR2024-00221

first, and then write the minimum VBA code required to carry out the pre-programmed actions. Most applications do not need any code other than an occasional user-defined function and a few event-handling procedures.

- Use command buttons and associated event actions to substitute for the traditional menu commands of DOS RDBMSs. In many applications, you create a menu that gives the user only the File, Exit command. All other user-initiated choices in the application are handled by control objects on forms.
- Concentrate on using event procedures to respond to user- and application-initiated events, such as opening forms. Study the event-oriented properties of form and control objects and the `DoCmd` actions that are available for responding to the event properties. Learn the full capabilities of each `DoCmd` action that is native to Access before you write your own actions in Access VBA. Event properties combined with the appropriate `DoCmd` actions usually can substitute for about 95 percent of the code you write for DOS RDBMSs.
- After your application is up and running, consider writing Access VBA code to add the nuances that distinguish professionally written database applications.

Where You Use Access VBA Code

You probably will first use Access VBA to create functions that make complex calculations. Creating an Access VBA function to substitute for calculations with many sets of parentheses is relatively simple and a good introduction to writing Access VBA code. Writing expressions as Access VBA functions allows you to add comments to the code that make the purpose and construction of the code clear to others. Also, comments aid your memory when you decide to revise an application after using it for a few months.

Your next step is to create Access VBA event-handling subprocedures in form or report modules. You use event-handling subprocedures to perform operations in response to events such as clicking a command button on a form, updating a value in a field, or when a field gains or loses the focus. Access 97 gives you the choice of calling a function contained in an Access module or executing an event-handling subprocedure contained in a form or report module. Microsoft called Access VBA contained in forms or reports Code Behind Forms (CBF) in Access 2.0 and 95; event-handling code in Access 97 is contained in class modules. (Chapter 30, "Responding to Events with VBA 5.0," describes class module programming in more detail.)

Typographic and Naming Conventions Used for Access VBA

This book uses a special set of typographic conventions for references to Access VBA keywords and object variable names in Access VBA examples:

- Monospace type is used for all Access VBA code in the examples, as in `lngItemCounter`.
- Bold monospace type is used for all VBA reserved words and type-declaration symbols, as in **`Dim`** and **`%`**. Standard function names in Access VBA, as described in Chapter 9, "Understanding Operators and Expressions in Access," also are set in bold

Oracle Ex. 1009, p. 990
Oracle v. DeBellis
IPR2024-00221

type so that reserved words, standard function names, and reserved symbols stand out from variable and function names and values you assign to variables. Keywords incorporated by reference in Access, such as `DoCmd`, are not set in bold.

- Italic monospace type indicates a replaceable item, as in **`Dim`** *`DataItem`* **`As String`**. *`DataItem`* is replaced by a name that you supply.
- Bold-italic monospace type indicates a replaceable reserved word, such as a data type, as in **`Dim`** *`DataItem`* **`As`** ***`DataType`***; ***`DataType`*** is replaced by a VBA-reserved word corresponding to the desired VBA data type.
- Names of variables that refer to Access objects, such as forms or reports, use a three-letter prefix derived from the object name, as in `frm`*`FormName`* and `rpt`*`ReportName`*. The prefixes for object variables are listed in Appendix B, "Naming Conventions for Access Objects and Variables."
- Names of other variables are preceded by a one-letter or three-letter data type identifier, such as `var`*`VariantVariable`* and `int`*`IntegerVariable`*. Boolean variables (flags) that return only **`True`** or **`False`** values use the `f` prefix, as in `fIsLoaded`.
- Optional elements are included within square brackets, as in `[`*`OptionItem`*`]`. Square brackets also enclose object names that contain spaces or special punctuation symbols.
- An ellipsis (. . .) substitutes for code that is not shown in syntax and code examples, as in **`If...Then...Else...End If`**.

Modules, Functions, and Subprocedures

A *module* is a container for Access VBA code, just as a form is a container for control objects. Access 97 provides the following three types of modules:



- *Access Modules*. You create an Access module to contain your Access VBA code the same way that you create any other new database object: click the Module tab in the Database window and then click the New button. Alternatively, you can click the New Object button on the toolbar and choose New Module from the drop-down menu. Figure 28.1 shows the `IsLoaded()` function of the Utility Functions module of Northwind.mdb. Access modules are also called *standard modules*.

Tip

Using Access modules to contain all of your Access VBA code speeds the opening of forms and reports but slows the initial opening of your application.





- *Form Modules*. Form modules contain code in class modules to respond to events triggered by forms or controls on forms. Essentially, when you add code to a form object, you create a new class of object in the database. The event-handling procedures you create for the form are its new methods, hence the term *class module* for the code module associated with a particular form. You open a form module by clicking the Code button of the toolbar in Form Design view. Alternatively, choose

Oracle Ex. 1009, p. 991
Oracle v. DeBellis
IPR2024-00221

View, Code. Either of these methods opens a module that Access automatically names Form_*FormName*, where *FormName* is the name of the selected form. (Access 2.0 named the modules Form.*FormName*.) Prior to Access 97, forms automatically included a module as part of the form. Forms in Access 97, however, have a new property—Has Module. If this property is set to Yes, then the form has an attached module; otherwise, it does not. Leave the Has Module property set to No unless you really do need to add code to the form.

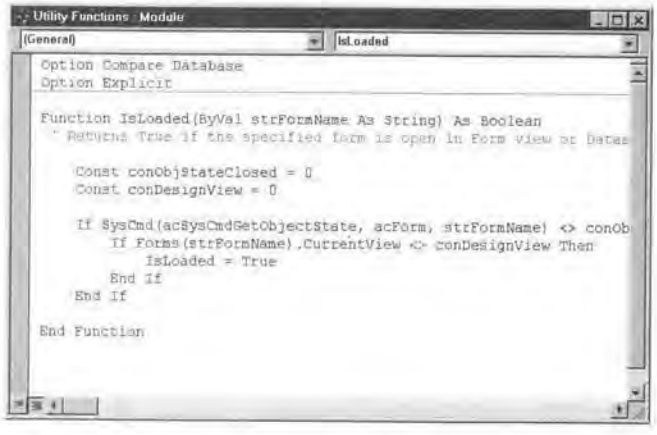


FIG. 28.1 The Access module window for the `IsLoaded()` function.

Another method of opening a form module is to click the ellipsis button for one of the event properties for a form or a control object on a form. Selecting Code Builder from the Choose Builder dialog displays the Form_*FormName* module with a procedure stub, **`Private Sub`** *`ObjectName_EventName`*`()...`**`End Sub`**, written for you. Access 97 adds the VBA **`Private`** prefix by default. Figure 28.2 shows the Access VBA code for the `CustomerID_AfterUpdate` procedure of Northwind.mdb's Orders form.

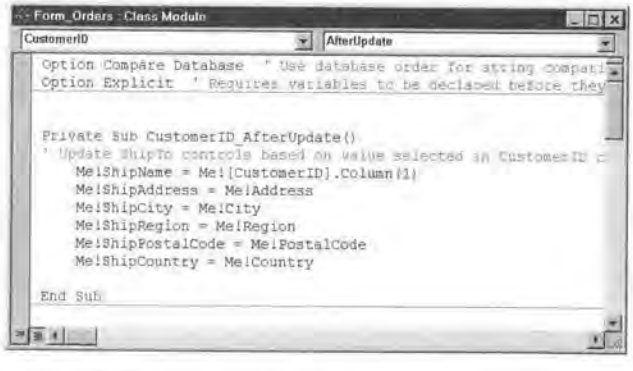


FIG. 28.2 A typical event-handling procedure in a class module.

Oracle Ex. 1009, p. 992
Oracle v. DeBellis
IPR2024-00221

- *Report Modules*. Report modules contain code for responding to events triggered by reports, sections of reports, or group headers and footers. (Control objects on reports do not trigger events.) You open a report's class module the same way you open a form's class module. Report modules are named Report_*ReportName* automatically. Like forms, whether an Access 97 report has a class module depends on the setting of the Has Module property.

A module consists of a *Declarations section* and usually one or more *procedures* or *functions*. As the name suggests, the Declarations section of a module is used to declare items (usually variables and constants, the subjects of following sections) used by the procedures and functions contained in the module. You can use a module without functions or procedures to declare **Public** variables and constants that can be used by any function or procedure in any module. Similarly, you use the **Public** prefix for functions and subprocedures to allow their use by code in any module. (**Public** replaces Access 2.0's **Global** keyword, but **Global** continues to work in Access 97.)

Note

The Module windows of Figures 28.1 and 28.2 show the Declarations section and a function or procedure separated by a horizontal line. The ability to display the Declarations sections and all procedures in a single scrollable window, called Full Module view, is a feature shared by the Module windows of all flavors of VBA. If you see only the Declarations section when you open a module, click the Full Module View button at the bottom left of the module window (shown in its "down" position in both Figures 28.1 and 28.2). Alternatively, choose Tools, Options to display the Options dialog. Click the Module tab and mark the Full Module View and Procedure Separator check boxes.

Procedures are typically defined as subprograms referred to by name in another program. Referring to a procedure by name *calls* or *invokes* the procedure; the code in the procedure executes, and then the sequence of execution returns to the program that called the procedure. Another name for a procedure is *subroutine*. Procedures can call other procedures, in which case the called procedures are called *subprocedures*. Procedures are defined by beginning (**Sub**) and end (**End Sub**) reserved words with a **Public**, **Private**, or **Static** prefix, as in the following example:

```
Private Sub ProcName
    [Start of procedure code]
    ...
    [End of procedure code]
End Sub
```

Note

You can refer to the procedure name to invoke the procedure, but Access VBA provides a keyword, **Call**, that explicitly invokes a procedure. Prefixing the procedure name with **Call** is a good programming practice because this keyword identifies the name that follows as the name of a procedure rather than a variable.

Oracle Ex. 1009, p. 993
Oracle v. DeBellis
IPR2024-00221

Functions are a class of procedures that return values to their names, as explained in Chapter 9, "Understanding Operators and Expressions in Access." C programmers would argue that procedures are a class of functions, called *void functions*, that do not return values. Regardless of how you view the difference between functions and subprocedures, keep the following points in mind:

- Access macros require that you write Access VBA functions (not subprocedures) to act in place of macro actions when using the `RunCode` macro action.
- The only way you can call a subprocedure in an Access VBA module is from an Access VBA function or another procedure. You cannot directly execute a procedure in an Access module from any Access database object.
- Unlike Access modules, form and report class modules use subprocedures (not functions) to respond to events. Using form- and report-level subprocedures for event-handling code mimics Visual VBA's approach for events triggered by forms and controls on forms.
- Function names in Access modules are global in scope with respect to Access modules unless they are declared **`Private`**. Thus, you cannot have duplicate **`Public`** function names in any Access module in your application. However, form and report class modules can have a function with the same name as a **`Public`** function in a standard module because form and report function and procedure names have form- or report-level scope. A function in a form module with the same name as a function in an Access module takes priority over the Access module version. Therefore, if you include the `IsLoaded()` function in a form module and call the `IsLoaded()` function from a procedure in the form module, the `IsLoaded()` function in the form module executes.
- To execute an Access VBA function in Access VBA code, you must use the function in an expression, such as

  ```
  intReturnValue = nilFunctionName([Arguments])
  ```

  even when the function returns no value.

Functions are created within a structure similar to procedures, as in the following example:

```
Private Function FuncName([Arguments])
    [Start of function code]
    ...
    [End of function code]
End Function
```

You cannot use **`Call`** to execute a function; you must refer to the function by name. Function calls are identified by the parentheses that follow the function name—even if the function requires no arguments.

You can add as many individual procedures and functions to a module as you want. If you write a substantial amount of Access VBA code, you should take one of the following two approaches:

Oracle Ex. 1009, p. 994
Oracle v. DeBellis
IPR2024-00221

- Use form and report modules to contain the code that responds to the events on forms. Adding a class module to a form or report creates a new class of object within your database. Other forms and reports created from the originals will inherit the class module of the form or report and the VBA code that it contains. Procedures common to several forms, however, should be incorporated in Access modules. As mentioned in the tip earlier in this chapter, adding substantial amounts of code to form modules slows the opening of the form.
- Create separate Access modules for code that is associated with a particular class of object, such as forms. A form that requires an appreciable amount of code usually deserves its own module. When you take the Access module approach, your application opens more slowly, but forms open more rapidly. Users are likely to favor applications that respond more quickly after they are launched.

Access VBA introduces another class of procedure called property procedures that use the `{Property Let | Property Get | Property Set}...End Property` structure to create custom properties for Access objects, such as forms or controls. Property procedures are used by VBA developers primarily for creating custom wizards. A discussion of property procedures is beyond the scope of this book.

References to VBA and Access Modules

Access 97 uses references to make objects available for use in modules. Access 97's default references are Visual Basic for Applications, Microsoft Access 97, Microsoft DAO 3.5 Object Library, and the currently open database. To view the default references, open a module, then choose Tools, References to open the References dialog (see Figure 28.3). Current references, except to the open database, are indicated by a mark in the adjacent check box.

Access 97 implements VBA by establishing references to collections of objects exposed by VBA's object library, Vbaen32.olb. Vbaen32.olb is the English (en) version of VBA, 32-bit version (32). Similarly, Access exposes its application objects through an object library, Msacc8.olb, and the Jet database engine exposes data access objects (DAOs) through Dao2535.olb (the Jet 3.5 engine is compatible with DAO 2.5 and 3.5, hence the 2535 in the object library's name). Each application that supports VBA has a similar, but not identical, set of default references.



Referenced objects appear in the Project/Library drop-down list of the Object Browser. To view the Object Browser, open a module and click the Object Browser button on the toolbar (or choose View, Object Browser). <All Libraries> is the default selection in the Project/Library list. Figure 28.4 shows a few of the references to `Report` objects in Northwind.mdb and the Utility Functions module in the Classes list. Only objects that can act as VBA containers appear in the Classes list.

When you select a function or subprocedure name in a module, the function or subprocedure name and arguments, if any, appear in the window at the bottom of the Object Browser dialog. You can obtain help on Access, VBA, and other objects by clicking the help (?) button, which is disabled for user-defined functions and the event-handling

Oracle Ex. 1009, p. 995
Oracle v. DeBellis
IPR2024-00221

subprocedures you write. Chapter 29, "Understanding the Data Access Object Class," describes object classes and the use of the Object Browser in detail.

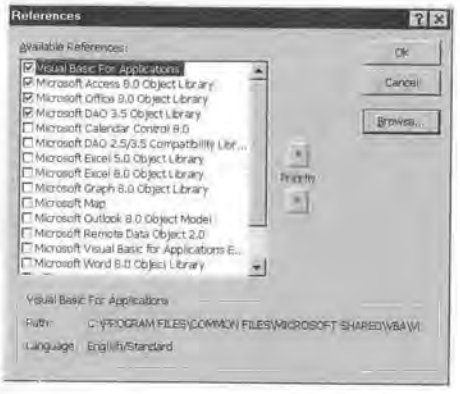


FIG. 28.3 The four default object references for Access 97.

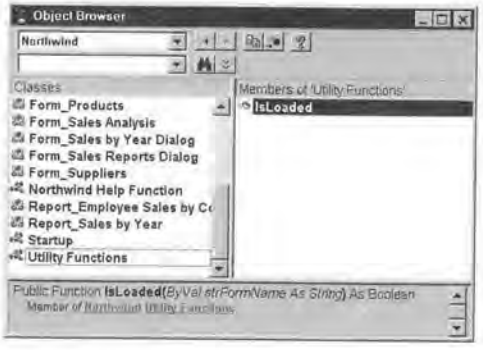


FIG. 28.4 The Object Browser displaying the calling syntax for the `IsLoaded()` function.

Data Types and Database Objects in Access VBA

When you create Access VBA tables, all data types that you use to assign field data types and sizes (except for OLE and Memo field data types) have data type counterparts in Access VBA. With the exception of the **`Variant`** and **`Currency`** data types, Access VBA data types are represented in most other dialects of BASIC, such as Microsoft QuickBASIC and the QBasic interpreter supplied with MS-DOS 5 and later.

Traditional BASIC dialects use a punctuation symbol called the *type-declaration character*, such as **`$`** for the **`String`** data type, to designate the data type. The Access VBA data types, the type-declaration characters, the corresponding field data types, and the ranges of values are shown in the VBA Type, Symbol, Field Type, Minimum Value, and Maximum Value columns, respectively, of Table 28.1. The Field Types **`Byte`**, **`Integer`**, **`Long Integer`**, **`Single`**, and **`Double`** correspond to the Field Size property of the Number data type in

Oracle Ex. 1009, p. 996
Oracle v. DeBellis
IPR2024-00221

tables, queries, forms, and reports. Access VBA adds the **Byte** and **Boolean** data types to support the 8-bit Byte and 16-bit Yes/No field data types.

Table 28.1 Access VBA and Corresponding Field Data Types

VBA Type	Symbol	Field Type	Minimum Value	Maximum Value
Byte	None	Byte	0	255
Integer	%	Integer	-32,768	32,767
Boolean	None	Yes/No	**True**	**False**
Long	&	Long Integer, AutoNumber	-2,147,483,648	2,147,483,647
Single	!	Single	-3.402823E38 1.401298E-45	-1.401298E-5 3.402823E38
Double	#	Double	-1.79769313486232E308 4.94065645841247E-324	4.9406564841247E-324 1.79769313486232E308
Currency	@	Currency	-922,337,203,685, 477.5808	922,337,203,685, 477.5807
String	$	Text or Memo	0 characters	Approximately 2 billion characters
Date	None	Date/Time	January 1, 100	December 31, 9999
Variant	None	Any	January 1, 100 (date)	December 31, 9999 (date)
			Same as **Double** (numbers)	Same as **Double** (numbers)
			Same as **String** (text)	Same as **String** (text)

Tip

All data returned from fields of tables or queries is of the **Variant** data type by default. If you assign the field value to a conventional data type, such as **Integer**, the data type is said to be *coerced*.

You can dispense with the type-declaration character if you explicitly declare your variables with the **Dim...As *DataType*** statement, discussed later in this section. If you do not explicitly declare the variables' data type or use a symbol to define an implicit data type, Access VBA variables default to the **Variant** data type.

The # sign is also used to enclose values specified as dates, as in `varNewYears = #1/1/97#`. In this case, bold type is not used for the enclosing # signs because these symbols are not intended for the purpose of the # reserved symbol that indicates the **Double** data type.

Database objects—such as databases, tables, and queries—and application objects (forms and reports), all of which you used in prior chapters, also have corresponding object data types in Access VBA. Here, Access VBA departs from other BASIC languages with the exception of Visual Basic. The most commonly used object data types of Access VBA and the object library that includes the objects are listed in Table 28.2.

Oracle Ex. 1009, p. 997
Oracle v. DeBellis
IPR2024-00221

Table 28.2 The Most Common Database Object Data Types Supported by Access

VBAObject Data	Library	Corresponding Database Object Type
`Database`	DAO 3.5	Databases opened by the Jet database engine
`Form`	Access 97	Forms, including subforms
`Report`	Access 97	Reports, including subreports
`Control`	Access 97	Controls on forms and reports
`QueryDef`	DAO 3.5	Query definitions (SQL statement equivalents)
`TableDef`	DAO 3.5	Table definitions (structure, indexes, and other table properties)
`Recordset`	DAO 3.5	A virtual representation of a table or the result set of a query

Note

The `Table`, `Dynaset`, and `Snapshot` object data types of Access 1.x have been replaced by the all-encompassing `Recordset` data type of Access 2+. You distinguish the type of `Recordset` object (`Table`, `Dynaset`, or `Snapshot`) by the `Recordset` object's `Type` property. The `Table`, `Dynaset`, and `Snapshot` object types are supported by Access 97 for backward compatibility with Access 1.x code. However, it is a good practice to use the new `Recordset` data type because no guarantee exists that the obsolete Access 1.x object data types will be supported in future Access versions. DAO data types and object collections are discussed in Chapter 29, "Understanding the Data Access Object Class."

Variables and Naming Conventions

Variables are named placeholders for values of a specified data type that change when your Access VBA code is executed. You give variables names, as you name fields, but the names of variables cannot include spaces or any other punctuation except the underscore character (_). The other restriction is that a variable cannot use an Access VBA keyword by itself as a name; keywords are called *reserved words* for this reason. The same rules apply to giving names to functions and procedures. Variable names in Access VBA typically employ a combination of upper- and lowercase letters to make them more readable.

Implicit Variables. You can create variables by assigning a value to a variable name, as in the following example:

```
NewVar = 1234
```

A statement of this type *declares* a variable, which means to create a new variable with a name you choose. The statement in the example creates a new implicit variable, `NewVar`, of the **`Variant`** data type with a value of 1234. (Thus, `NewVar` would be more appropriately named `varNewVar`.) When you do not specify a data type for an implicit variable by appending one of the type-declaration characters to the variable name, the **`Variant`** data type is assigned by default. The following statement creates a variable of the Integer data type:

Oracle Ex. 1009, p. 998
Oracle v. DeBellis
IPR2024-00221

```
NewVar% = 1234
```

Declaring variables of the **Integer** or **Long** (integer) type when decimal fractions are not required speeds the operation of your code. Access takes longer to compute values for **Variant**, **Double**, and **Single** variables.

Explicit Variables. It is a better programming practice to declare your variables and assign those variables a data type before you give variables a value. Programming languages such as Pascal, C, C++, and Delphi require you to declare variables before you use them. The most common method of declaring variables is by using the **Dim...As *Datatype*** structure where **As** specifies the data type. This method declares explicit variables. An example follows:

```
Dim intNewVar As Integer
```

If you do not add the **As Integer** keywords, `intNewVar` is assigned the **Variant** data type by default.

You can require that all variables be explicitly declared prior to their use by adding the statement **Option Explicit** in the Declarations section of a module. The advantage of using **Option Explicit** is that Access detects misspelled variable names and displays an error message when misspellings are encountered. If you do not use **Option Explicit** and you misspell a variable name, Access creates a new implicit variable with the misspelled name. The resulting errors in your code's operation can be difficult to diagnose. Access automatically adds an Option Explicit statement to the Declarations section of each module if you set the Require Variable Declaration option on in the Module sheet of the Options dialog (see Figure 28.5).

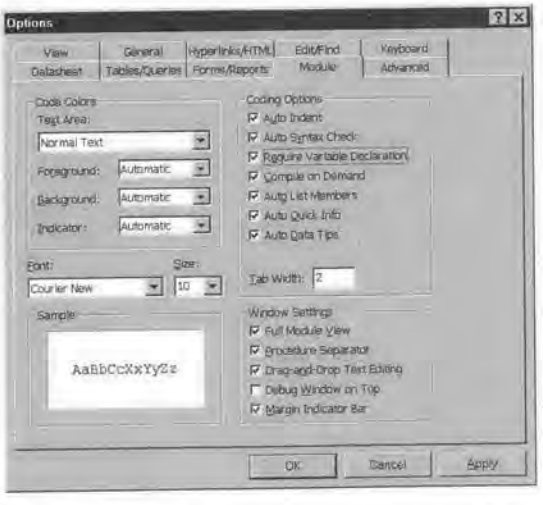


FIG. 28.5 Setting the Require Variable Declaration option in the Module sheet of the Options dialog.

Oracle Ex. 1009, p. 999
Oracle v. DeBellis
IPR2024-00221

Scope and Duration of Variables. Variables have a property called *scope*, which determines when the variables appear and disappear in your Access VBA code. Variables appear the first time you declare them and then disappear and reappear on the basis of the scope you assign to them. When a variable appears, it is said to be *visible*—meaning that you can assign the variable a value, change its value, and use it in expressions. Otherwise, the variable is *invisible*; if you use a variable's name while it is invisible, you instead create a new variable with the same name.

The following lists the four scope levels in Access VBA:

- *Local (procedure-level) scope.* The variable is visible only during the time when the procedure in which the variable is declared is executed. Variables that you declare, with or without using `Dim...As Datatype` in a procedure or function, are local in scope.
- *Form-level and report-level scope.* The variable is visible only when the form or report in which it is declared is open. You declare form-level and report-level variables in the Declarations section of form and report modules with `Private...As Datatype`. (`Dim...As Datatype` also works.)
- *Module-level scope.* The variable is visible to all procedures and functions contained in the module in which the variable was declared. (Modules open when you open the database.) You declare variables with module scope in the Declarations section of the module with the same syntax as form- and report-level variables.
- *Global or public scope.* The variable is visible to all procedures and functions within all modules. You declare variables with global scope in the Declarations section of a module using `Public...As Datatype`.

The scope and visibility of variables declared in two different Access modules of the same database, both with two procedures, are illustrated by the diagram in Figure 28.6. In each procedure, variables declared with different scopes are used to assign values to variables declared within the procedure. Invalid assignment statements are shown crossed out in the figure. These assignment statements are invalid because the variable used to assign the value to the variable declared in the procedure is not visible in the procedure.

Variables also have a property called *duration*, or *lifetime*. The duration of a variable is your code's execution time between the first appearance of the variable (its declaration) and its disappearance. Each time a procedure or function is called, local variables declared with the `Dim...As Datatype` statement are set to default values with 0 for numeric data types and the empty string ("") for string variables. The duration of these local variables is usually equal to the lifetime of the function or procedure—from the time the function or procedure is called until the `End Function` or `End Sub` statement is executed.

Oracle Ex. 1009, p. 1000
Oracle v. DeBellis
IPR2024-00221

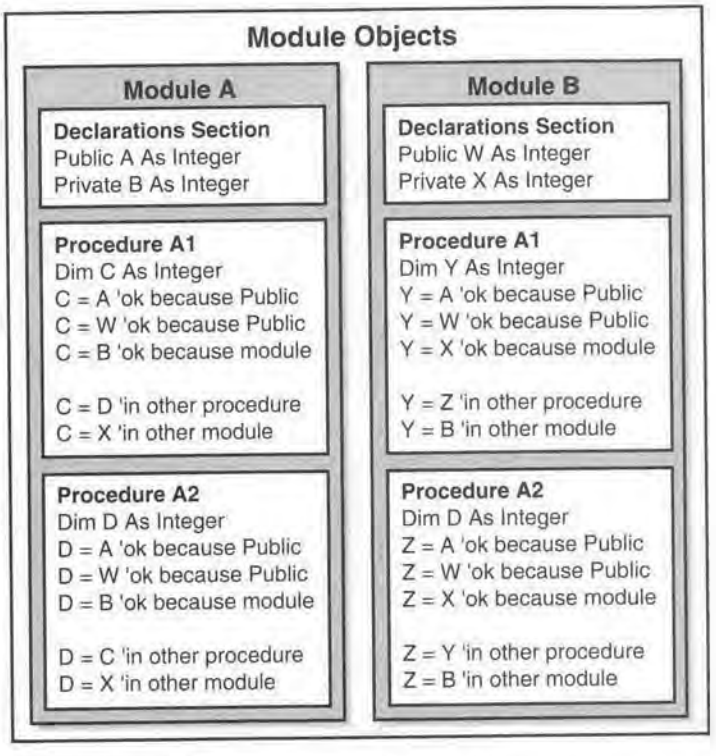


FIG. 28.6 Valid and invalid assignment statements for variables of different scopes.

To preserve the values of local variables between occurrences (called *instances*) of a procedure or function, you substitute the reserved word **`Static`** for **`Dim`**. Static variables have a duration of your Access application, but their scope is determined by where you declare them. Static variables are useful when you want to count the number of occurrences of an event. You can make all variables in a function or procedure static variables by preceding **`Function`** or **`Sub`** with the **`Static`** keyword.

Note

Minimize the number of local variables that you declare **`Static`**. Local variables do not consume memory when they are not visible. This characteristic of local variables is especially important in the case of arrays, discussed in the "Access VBA Arrays" section that follows shortly, because arrays are often very large.

User-Defined Data Types. You can create your own data type that consists of one or more Access data types. User-defined data types are discussed in this section pertaining to variables because you need to know what a variable is before you can declare a user-defined data type. You declare a user-defined data type between the **`Type...End Type`** keywords, as in the following example:

Oracle Ex. 1009, p. 1001
Oracle v. DeBellis
IPR2024-00221

```
Type DupRec
    Field1 As Long
    Field2 As String * 20
    Field3 As Single
    Field4 As Double
End Type
```

User-defined data types are particularly useful when you create a variable to hold the values of one or more records of a table that uses fields of different data types. The `String * 20` statement defines `Field2` of the user-defined data type as a fixed-length string of 20 characters, usually corresponding to the Size property of the Text field data type. String variables in user-defined data types are always specified with a fixed length. You must declare your user-defined data type (called a *record* or *structure* in other programming languages) in the Declarations section of a module.

You must explicitly declare variables to be of the user-defined type with the **Dim**, **Private**, **Public**, or **Static** keywords because no reserved symbol exists to declare a user-defined data type, as in `Dim usrCurrentRec As tagDupRec`. To assign a value to a field of a variable with a user-defined data type, you specify the name of the variable and the field name, separating the names with a period, as in `usrCurrentRec.lngField1 = 2048`.

Access VBA Arrays. Arrays are variables that consist of a collection of values, called *elements* of the array, of a single data type in a regularly ordered structure. Implicitly declared arrays are not allowed in Access VBA (or in Visual VBA). You declare an array with the **Dim** statement, adding the number of elements in parentheses to the variable name for the array, as in the following example:

```
Dim astrNewArray(20) As String
```

This statement creates an array of 21 elements, each of which is a conventional, variable-length string variable. You create 21 elements because the first element of an array is the `0` (zero) element unless you specify otherwise by adding the **To** modifier, as in the following example:

```
Dim astrNewArray(1 To 20) As String
```

The preceding statement creates an array with 20 elements.

You can create multidimensional arrays by adding more values separated by commas. The statement

```
Dim alngNewArray(9, 9, 9) As Long
```

creates a three-dimensional array of 10 elements per dimension. This array, when visible, occupies 4,000 bytes of memory (10×10×10×4 bytes/long integer).

You can create a *dynamic array* by declaring the array using **Dim** without specifying the number of elements and then using the **ReDim** reserved word to determine the number of elements the array contains. You can **ReDim** an array as many times as you want. Each time you do so, the values stored in the array are reinitialized to their default values, determined by the data type, unless you follow **ReDim** with the reserved word, **Preserve**. The following sample statements create a dynamic array:

Oracle Ex. 1009, p. 1002
Oracle v. DeBellis
IPR2024-00221

```
Dim alngNewArray() As Long              'In Declarations sections
ReDim Preserve alngNewArray(9, 9, 9)    'In procedure, preserves prior values
ReDim alngNewArray(9, 9, 9)             'In procedure, reinitializes all
```

Dynamic arrays are useful when you don't know how many elements an array requires when you declare it. You can **ReDim** a dynamic array to zero elements when you no longer need the values it contains; this tactic allows you to recover the memory that the array consumes while it is visible. Alternatively, you can use the **Erase** reserved word followed by a dynamic array's name to remove all of the array's elements from memory. (**Erase** used on an array with fixed dimensions merely reinitializes the array to its condition before you assigned any values to it.) Arrays declared with **Dim** may have up to 60 dimensions. You can use the **ReDim** statement only to alter the size of the last dimension in a multidimensional array.

Scope, duration rules, and keywords apply to arrays in the same way in which they apply to conventional variables. You can declare dynamic arrays with global and module-level scope by adding the **Public** or **Private** statement to the Declarations section of a module and then using the **ReDim** statement by itself in a procedure. If you declare an array with **Static**, rather than **Dim**, the array retains its values between instances of a procedure.

> **Note**
>
> Do not use the **Option Base** keywords to change the default initial element of arrays from 0 to 1. **Option Base** is included in Access VBA for compatibility with other BASIC dialects. Many arrays you create from Access VBA objects must begin with element 0. If you are concerned about the memory occupied by an unused zeroth element of an array, use the **Dim** *ArrayName*(1 **To** N) **As** ***DataType*** declaration. In most cases, you can disregard the zeroth element.

Named Database Objects as Variables in Access VBA Code. Properties of database objects you create with Access can be treated as variables and assigned values within Access VBA code. For example, you can assign a new value to the text box that contains the address information for a customer by name. Use the following statement:

```
Forms!Customers!Address = "123 Elm St."
```

The collection name `Forms` defines the type of object. The exclamation point (called the *bang* symbol by programmers) separates the name of the form and the name of the control object. The ! symbol is analogous to the \ path separator that you use when you are dealing with folder and file names. If the name of the form or the control object contains a space or other punctuation, you need to enclose the name within square brackets, as in the following statement:

```
Forms!Customers![Contact Name] = "Joe Hill"
```

Alternatively, you can use the **Set** keyword to create your own named variable for the control object. This procedure is convenient when you need to refer to the control object several times. It is more convenient to type `txtContact` rather than the full "path" to the control object—in this case, a text box:

```
Dim txtContact As Control
```

Oracle Ex. 1009, p. 1003
Oracle v. DeBellis
IPR2024-00221

```
Set txtContact = Forms!Customers![Contact Name]
txtContact = "Joe Hill"
```

You can assign any database object to a variable name by declaring the variable as the object type and using the **Set** statement to assign the object to the variable. You do not create a copy of the object in memory when you assign it a variable name; the variable refers to the object in memory. Referring to an object in memory is often called *pointing* to an object; many languages have a pointer data type that holds the address of the location in memory where the variable is stored. Access VBA does not support pointers. Chapter 29, "Understanding the Data Access Object Class," deals with creating variables that point to the Access 97 database objects supplied by the Jet 3.5 DAO.

Object Properties and the *With...End With* Structure. Access VBA introduces the **With...End With** structure that offers a shorthand method of setting the values of object properties, such as the dimensions and other characteristics of a form. The **With...End With** structure also lets you set the values of fields of a user-defined data type without repeating the variable name in each instance. To use the **With...End With** structure to set object property values, you must first declare and set an object variable, as in the following example:

```
Dim frmFormName As Form
Set frmFormName = Forms!FormName
With frmFormName
    .Top = 1000
    .Left = 1000
    .Width = 5000
    .Height = 4000
End With
```

When using the **With...End With** structure with user-defined data types, you don't use the **Set** statement. Names of properties or fields within the structure are preceded by periods.

Variable Naming Conventions. In the event that you need to write large amounts of Access VBA code, you probably will employ a large number of variable names and data types. On forms and reports, many different types of named control objects can be used, each of which you can assign to a variable name with the **Set** keyword in your Access VBA code. As your code grows in size, remembering the data types of all variables becomes difficult.

Stan Leszynski, a Seattle database consultant, has created a set of variable-naming conventions for Access VBA. The *Leszynski Naming Conventions for Microsoft Access* employs *Hungarian notation*, which is used primarily in the C and C++ languages. "Hungarian" refers to the nationality of the method's inventor, Charles Simonyi (who was involved in the development of Access), and the fact that only Hungarians are likely to be able to pronounce some of the abbreviations correctly, such as "lpsz." Leszynski adapted Hungarian notation for use in the Access VBA environment.

Hungarian notation uses a set of codes for the data type. You prefix the variable name with the code in lowercase letters. For example, the prefix code for a text box is `txt`, so

Oracle Ex. 1009, p. 1004
Oracle v. DeBellis
IPR2024-00221

the variable name for the text box in the preceding example is `txtContact`. Strings in C code are identified by the prefix `lpsz`, an abbreviation for long pointer to a string that is zero-terminated (with `Chr$(0)` or ASCII `Null`). Access VBA does not support the pointer data type (but it does use pointers for the location of string variables), and it does not use zero-terminated strings. Therefore, `str` is a more appropriate prefix for Access VBA strings.

The data type identifier of a user-defined data type is called a *created tag*. In Hungarian notation, created tags are capitalized as in the following:

```
Type REC
    lngField1 As Long
    strField2 As String
    sngField3 As Single
    dblField4 As Double
End Type
```

A variable of type `REC` is declared with the lowercase `rec` prefix, such as in the following:

```
Dim recCurRecord As REC
```

Created tags should be short but should not duplicate one of the standard data-type prefix codes.

Using standard data type prefixes makes your code easier to read and understand. Many of the code examples in this book use the new proposed naming conventions. The complete list of the Leszynski notation for Access VBA objects and variables appears in Appendix B, "Naming Conventions for Access Objects and Variables."

Symbolic Constants

Symbolic constants are named placeholders for values of a specified data type that do not change when your Access VBA code is executed. You precede the name of a symbolic constant with the keyword, **`Const`**, as in `Const sngPI As Single = 3.1416`. You declare symbolic constants in the Declarations section of a module or within a function or procedure. Precede **`Const`** with the **`Public`** keyword if you want to create a global constant that is visible to all modules, as in `Public Const gsngPI = 3.1416`. Public constants can be declared only in the Declarations section of an Access VBA module.

Typically, symbolic constants take names in all capital letters to distinguish them from variables. Underscores often are used to make the names of symbolic constants more readable, as in `sngVALUE_OF_PI`. This naming convention no longer applies to system-defined constants in Access 97 or Access 95. System-defined intrinsic constants are the subject of the next section, "Access System-Defined Constants."

You do not need to specify a data type for constants because Access VBA chooses the data type that stores the data most efficiently. Access VBA can make this choice because it knows the value of the data when it "compiles" your code. It is a good programming practice, however, to add **`As Datatype`** when declaring constants.

Oracle Ex. 1009, p. 1005
Oracle v. DeBellis
IPR2024-00221

> **Note**
>
> Access is an interpreted language, so the term *compile* in an Access VBA context is a misnomer. When you "compile" the Access VBA source that you write in code-editing windows, Access creates a tokenized version of the code (called *pseudo-code*, or *p-code*). This subject is discussed in the section "The Access VBA Compiler" later in the chapter.

Access System-Defined Constants. Access VBA includes seven system-defined constants—**`True`**, **`False`**, `Yes`, `No`, `On`, `Off`, and **`Null`**—that are created by the VBA and Access type libraries when launched. Of these seven, you can use **`True`**, **`False`**, and **`Null`**, which are declared by the VBA library, in Access VBA code. The remaining four are declared by the Access type library and are valid for use with all database objects except modules. When the system-defined constants **`True`**, **`False`**, and **`Null`** are used in Access VBA code examples in this book, they appear in bold, monospace type.

Access Intrinsic Constants. Access VBA provides a number of predeclared, intrinsic, symbolic constants that are primarily for use as arguments of `DoCmd.Action` statements. These statements let you execute standard database actions in Access VBA (such as opening forms, printing reports, applying sorts or filters, and so on). Access 97 intrinsic constants carry the prefix `ac`, as in `acExportMerge`. You can display the list of Access intrinsic constants in the Object Browser by selecting Access in the Project/Library list and then selecting Constants in the Classes list.

When you select a constant in the Members of list, its numeric value appears at the bottom of the Object Browser window (see Figure 28.7). A good programming practice is to use constants rather than their numeric values when applicable. You may not use any of these intrinsic constants' names as names for constants or variables that you define.

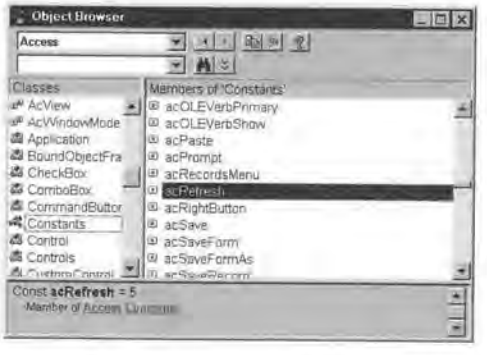


FIG. 28.7 Displaying Access 97 intrinsic constants in the Object Browser.

Oracle Ex. 1009, p. 1006
Oracle v. DeBellis
IPR2024-00221

Note

In Access 2.0 and earlier versions, intrinsic constants were classified as action constants (prefixed with A_), database constants (prefixed with DB_), and the Variant data type constants (prefixed with V_). In Access 97, each object library contributes its own set of intrinsic constants, prefixed with db for Jet 3.5 DAO constants and with vb for VBA constants. If you convert an Access 2.0 or earlier application to Access 97, the prior constant names are valid. These prior constants appear when you select `OldConstants` in the Classes list.

Access VBA Named and Optional Arguments

Procedures often have one or more arguments that pass values from the calling statement to the called procedure. Traditionally, you must pass all of the values required by the procedure in your calling statement. As an example, if a procedure accepts four arguments, *Arg1...Arg4*, your calling statement must provide values for `Arg1...Arg4`, as in the following example:

```
Sub CallingProc()
    ...
    Call CalledProc(100000, 200000, 300000, 400000)
    ...
End Sub

Sub CalledProc(Arg1 As Long, Arg2 As Long, Arg3 As Long, _
               Arg4 As Long)
    [Subprocedure code]
End Sub
```

Access 97 lets you declare the arguments of the subprocedure to be **`Optional`**, eliminating the need to pass every parameter to the procedure. You use *named arguments* to pass values to specific arguments, as in the following example:

```
Sub CallingProc()
    ...
    Call CalledProc(Arg2:=200000, Arg3:=300000)
    ...
End Sub

Sub CalledProc(Optional Arg1, Optional Arg2, Optional Arg3, _
               Optional Arg4)
    [Subprocedure code]
End Sub
```

The `:=` operator specifies that the preceding element is the name of an argument; named arguments need not be entered in the order that the arguments appear in the called procedure. However, if you want to omit an argument or arguments, the corresponding argument name(s) of the called procedure must be preceded by the keyword **`Optional`**. Optional arguments must be of the **`Variant`** data type, and missing arguments return **`Null`** values to subprocedure code. If you omit the **`As Datatype`** modifier of an argument in the called procedure, the argument assumes the default **`Variant`** data type.

Oracle Ex. 1009, p. 1007
Oracle v. DeBellis
IPR2024-00221

Controlling Program Flow

Useful procedures must be able to make decisions based on the values of variables and then take specified actions based on those decisions. Blocks of code, for example, may need to be repeated until a specified condition occurs. Statements used to make decisions and repeat blocks of code are the fundamental elements that control program flow in Access VBA and all other programming languages.

All programming languages require methods of executing different algorithms based on the results of one or more comparison operations. You can control the flow of any program in any programming language with just three types of statements: conditional execution (**`If...Then...End If`**), repetition (**`Do While...Loop`** and related structures), and termination (**`End...`**). The additional flow control statements in Access VBA and other programming languages make writing code more straightforward.

The code examples used in the following sections do not include **`Dim`** statements, so Hungarian notation is not used here. Data type identification symbols are used for brevity to indicate the data types of variables.

Branching and Labels

If you have written DOS batch files or WordPerfect macros, you are probably acquainted with branching and labels. Both the DOS batch language and the WordPerfect macro language include the **`GoTo`** *`Label`* command. DOS defines any word that begins a line with a colon as a label; WordPerfect requires the use of the keyword `LABEL` and then the label name.

When BASIC was first developed, the only method of controlling program flow was through its `GOTO` *`LineNumber`* and `GOSUB` *`LineNumber`* statements. Every line in the program required a number that could be used as a substitute for a label. `GOTO` *`LineNumber`* caused the interpreter to skip to the designated line and continue executing the program from that point. `GOSUB` *`LineNumber`* caused the program to follow that same branch, but when the BASIC interpreter that executed the code encountered a `RETURN` statement, program execution jumped back to the line following the `GOSUB` statement and continued executing at that point.



◀◀ See "Assignment and Comparison Operators," p. 290

Skipping Blocks of Code with *GoTo*. Procedural BASIC introduced named labels that replaced the line numbers for `GOTO` and `GOSUB` statements. Access VBA's **`GoTo`** *`Label`* statement causes your code to branch to the location named *`Label`*`:` and continue from that point. Note the colon following *`Label`*`:`, which identifies the single word you assigned as a label. However, the colon is not required after the label name following the **`GoTo`**. In fact, if you add the colon, you get a `Label not found` error message.

A label name must begin in the far-left column (1) of your code. This positioning often interferes with the orderly indenting of your code (explained in the next section), which is just one more reason, in addition to those following, for not using `GoTo`.

Oracle Ex. 1009, p. 1008
Oracle v. DeBellis
IPR2024-00221

Avoiding Spaghetti Code by Not Using *GoTo*. The sequence of statements in code that uses multiple **GoTo** statements is very difficult to follow. It is almost impossible to understand the flow of a large program written in line-numbered BASIC because of the jumps here and there in the code. Programs with multiple **GoTo** statements are derisively said to contain "spaghetti code."

The **GoTo** statement is required for only one purpose in Access VBA: to handle errors with the **On Error GoTo** *Label* statement. Although Access VBA supports BASIC's ON...GOTO and ON...GOSUB statements, using those statements is not considered good programming practice. You can eliminate all **GoTo** statements in form and report modules by using Access's Error event and DAO 3.5's new Errors collection. The Error event is described in the "Handling Run-time Errors" section later in this chapter, and the Errors collection is explained in Chapter 29, "Understanding the Data Access Object Class."

Conditional Statements

A *conditional statement* executes the statements between its occurrence and the terminating statement if the result of the relational operator is true. Statements that consist of or require more than one statement for completion are called *structured statements*, *control structures*, or just *structures*.

The *If...Then...End If* Structure. The syntax of the primary conditional statement of procedural BASIC is as follows:

```
If Condition1% [= True] Then
    Statements to be executed if Condition1 is true
 [Else[If Condition2%[ = True] Then]]
     Optional statements to be executed if Condition1%
     is false [and Condition2% is true]
End If
```

The **= True** elements of the preceding conditional statement are optional and typically not included when you write actual code. **If** *Condition1%* **Then** and **If** *Condition1%* **= True Then** produce the same result when *Condition1%* is **True**.

You can add a second condition with the **ElseIf** statement. The **ElseIf** condition must be true to execute the statements that are executed if *Condition1%* is false. Note that no space is used between **Else** and **If**. An **If...End If** structure that incorporates the **ElseIf** statement is the simplified equivalent of the following:

```
If Condition1% Then
    Statements to be executed if Expression1 is true
Else
    If Condition2% Then
        Statements to be executed if Condition1% is
        false and Condition2% is true]
    End If
End If
```

A statement is executed based on the evaluation of the immediately preceding expression. Expressions that include **If...End If** or other flow-control structures within other **If...End If** structures are said to be *nested*, as in the preceding example. The number, or

Oracle Ex. 1009, p. 1009
Oracle v. DeBellis
IPR2024-00221

depth, of **If...End If** structures that can be nested within one another is unlimited.

Note that the code between the individual keywords that make up the flow-control structure is indented. Indentation makes code within structures easier to read. You usually use the Tab key to create indentation.

To evaluate whether a character is a letter and to determine its case, you can use the following code:

```
If Asc(Char$) > 63 And Asc(Char$) < 91 Then
    CharType$ = "Uppercase Letter"
ElseIf Asc(Char$) > 96 And Asc(Char$) < 123 Then
    CharType$ = "Lowercase Letter"
End If
```

Note

You have seen a single-line version of the **If...End If** statement, IIf(), in the "Assignment and Comparison Operators" section of Chapter 9, "Understanding Operators and Expressions in Access." The single-line (also called *inline*) version in Access VBA, **If** *Condition%* **Then** ..., does not require the terminating **End If** statement. Although acceptable for simple statements, the use of the single-line version is questionable (not necessarily poor) programming practice, so this book avoids it.

You use the **If...End If** structure more often than any other flow control statement.

The *Select Case...End Select* Construct. When you must choose among many alternatives, **If...End If** structures can become very complex and deeply nested. The **Select Case...End Select** construct was added to procedural BASIC to overcome this complexity. In addition to testing whether an expression evaluates to true or false, **Select Case** can evaluate variables to determine whether those variables fall within specified ranges. The generalized syntax is in the following example:

```
Select Case VarName
    Case Expression1[, Expressions, ...]
        (Statements executed if the value of VarName
         = Expression1 or Expressions)
    [Case Expression2 To Expression3
        (Statements executed if the value of VarName
         is in the range of Expression2 to Expression3)]
    [Case Is RelationalExpression
        (Statements executed if the value of
         VarName = RelationalExpression)]
    [Case Else
        (Statements executed if none of the
         above cases is met)]
End Select
```

Select Case evaluates *VarName*, which can be a string, a numeric variable, or an expression. It then tests each **Case** expression in sequence. **Case** expressions can take one of the following four forms:

Oracle Ex. 1009, p. 1010
Oracle v. DeBellis
IPR2024-00221

- A single value or list of values to which to compare the value of *VarName*. Successive members of the list are separated from their predecessors by commas.
- A range of values separated by the keyword **To**. The value of the first member of the range limits must be less than the value of the second. Each string is compared by the ASCII value of its first character.
- The keyword **Is** followed by a relational operator, such as <>, <, <=, =, >=, or >, and a variable or literal value.
- The keyword **Else**. Expressions following **Case Else** are executed if no prior **Case** condition is satisfied.

The **Case** statements are tested in sequence, and the code associated with the first matching **Case** condition is executed. If no match is found and the **Case Else** statement is present, the code following the statement is executed. Program execution then continues at the line of code following the **End Select** terminating statement.

If *VarName* is a numeric type, all expressions with which it is to be compared by **Case** are forced to the same data type.

The following example is of **Select Case** using a numeric variable, Sales#:

```
Select Case Sales#
    Case 10000 To 49999.99
        Class% = 1
    Case 50000 To 100000
        Class% = 2
    Case Is < 10000
        Class% = 0
    Case Else
        Class% = 3
End Select
```

Note that because Sales# is a double-precision real number, all of the comparison literals also are treated as double-precision (not the default, single-precision) real numbers for the purpose of comparison.

A more complex example that evaluates a single character follows:

```
Select Case Char$
    Case "A" To "Z"
        CharType$ = "Upper Case"
    Case "a" To "z"
        CharType$ = "Lower Case"
    Case "0" To "9"
        CharType$ = "Number"
    Case "!", "?", ".", ",", ";"
        CharType$ = "Punctuation"
    Case ""
        CharType$ = "Empty String"
    Case < 32
        CharType$ = "Special Character"
    Case Else
        CharType$ = "Unknown Character"
End Select
```

Oracle Ex. 1009, p. 1011
Oracle v. DeBellis
IPR2024-00221

This example demonstrates that **Select Case**, when used with strings, evaluates the ASCII value of the first character of the string—either as the variable being tested or the expressions following **Case** statements. Thus, **Case** < 32 is a valid test, although Char**$** is a string variable.

Repetitive Operations: Looping

In many instances, you must repeat an operation until a given condition is satisfied, whereupon the repetitions terminate. You may want to examine each character in a word, sentence, or document, or you may want to assign values to an array with many elements. Loops are used for these and many other purposes.

Using the *For...Next* Statement. Access VBA's **For...Next** statement allows you to repeat a block of code for a specified number of times, as shown in the following example:

```
For Counter% = StartValue% To EndValue% [Step Increment%]
    Statements to be executed
    [Conditional statement
    Exit For
    End of conditional statement]
Next [Counter%]
```

The block of statements between the **For** and **Next** keywords is executed (*EndValue%* - *StartValue%* + 1) / *Increment%* times. As an example, if *StartValue%* = 5, *EndValue%* = 10, and *Increment%* = 1, the execution of the statement block is repeated six times. You need not add the keyword **Step** in this case—the default increment is 1. Although **Integer** data types are shown, **Long** (integers) may be used. The use of real numbers (**Single** or **Double** data types) as values for counters and increments is possible but uncommon.

The dividend of the previous expression must always be a positive number if the execution of the internal statement block is to occur. If *EndValue%* is less than *StartValue%*, *Increment%* must be negative; otherwise, the **For...Next** statement is ignored by Access VBA.

The optional **Exit For** statement is provided so that you can prematurely terminate the loop using a surrounding **If...Then...End If** conditional statement. Changing the value of the counter variable within the loop itself to terminate its operation is discouraged as a dangerous programming practice. You might make a change that would cause an infinite loop.

> **Note**
>
> If you use a numeric variable for *Increment%* in a **For...Next** loop and the value of the variable becomes **0**, the loop repeats indefinitely and locks up your computer, requiring you to reboot the application. (You can also sometimes halt an infinite loop by pressing the Esc key to interrupt code execution.) Make sure that your code traps any condition that could result in *Increment%* becoming **0**.

Oracle Ex. 1009, p. 1012
Oracle v. DeBellis
IPR2024-00221

The repetition of *Counter%* following the **Next** statement is optional but considered good programming practice—especially if you are using nested **For...Next** loops. Adding *Counter%* keeps you informed of which loop you are counting. If you try to use the same variable name for a counter of a nested **For...Next** loop, you receive an error message.

When the value of *Counter%* exceeds *EndValue%* or the **Exit For** statement is executed, execution proceeds to the line of code following **Next**.

Using *For...Next* Loops to Assign Values to Array Elements. One of the most common applications of the **For...Next** loop is to assign successive values to the elements of an array. If you have declared a 26-element array named `Alphabet$()`, the following example assigns the capital letters *A* through *Z* to its elements:

```
For Letter% = 1 To 26
    Alphabet$(Letter%) = Chr$(Letter% + 63)
Next Letter%
```

The preceding example assigns 26 of the array's 27 elements if you used **Dim** `Alphabet$(26)` rather than **Dim** `Alphabet$(1 To 26)`. 63 is added to `Letter%` because the ASCII value of the letter *A* is 64, and the initial value of `Letter%` is 1.

Understanding the *Do While...Loop* and *Do Until...Loop*. A more general form of the loop structure is **Do While...Loop**, which uses the following syntax:

```
Do While Condition% [= True]
    Statements to be executed
    [Conditional statement
    Exit Do
    End of conditional statement]
Loop
```

This loop structure executes the intervening statements only if *Condition%* equals **True** (**Not False**, a value other than **0**) and continues to do so until *Condition%* becomes **False** (**0**) or the optional **Exit Do** statement is executed.

From the preceding syntax, the previous **For...Next** array assignment example can be duplicated by the following structure:

```
Letter% = 1
Do While Letter% <= 27
    Alphabet$(Letter%) = Chr$(Letter% + 63)
    Letter% = Letter% + 1
Loop
```

Another example of a **Do** loop is the **Do Until...Loop** structure, which loops as long as the condition is not satisfied, as in the following example:

```
Do Until Condition% <> True
    Statements to be executed
    [Conditional statement
    Exit Do
    End of conditional statement]
Loop
```

Oracle Ex. 1009, p. 1013
Oracle v. DeBellis
IPR2024-00221

Access VBA also supports the **While...Wend** loop, which is identical to the **Do While...Loop** structure, but you cannot use the **Exit Do** statement within it. The **While...Wend** structure is provided for compatibility with earlier versions of BASIC and should be abandoned in favor of **Do {While|Until}...Loop** in Access VBA.

Making Sure Statements in a Loop Occur at Least Once. You may have observed that the statements within a **Do While...Loop** structure are never executed if *Condition%* is **Not True** when the structure is encountered in your application. You also can use a structure in which the conditional statement that causes loop termination is associated with the **Loop** statement. The syntax of this format is in the following example:

```
Do
    Statements to be executed
    [Conditional statement then
    Exit Do
    End of conditional statement]
Loop While Condition%[ = True]
```

A similar structure is available for **Do Until...Loop**:

```
Do
    Statements to be executed
    [Conditional statement
    Exit Do
    End of conditional statement]
Loop Until Condition%[ = False]
```

These structures ensure that the loop executes at least once *before* the condition is tested.

Avoiding Infinite Loops. You have already received warnings about infinite loops in the descriptions of each of the loop structures. If you create a *tight loop* (one with few statements between Do and Loop) that never terminates, executing the code causes Access VBA to appear to freeze. You cannot terminate operation; and you must use Ctrl+Alt+Delete to reboot, then relaunch Access VBA and correct the code.

These apparent infinite loops are often created intentionally with code such as the following:

```
Temp% = True
ExitLoop% = False
Do While Temp% [= True]
    Call TestProc(ExitLoop%)
    If ExitLoop%[ = True] Then
        Exit Do
    End If
Loop
```

The preceding example repeatedly calls the procedure TestProc, whose statements are executed, until ExitLoop% is set **True** by code in TestProc. You must make sure one way or another that ExitLoop% eventually becomes **True**, no matter what happens when the code in TestProc is executed.

Oracle Ex. 1009, p. 1014
Oracle v. DeBellis
IPR2024-00221

Note

To avoid locking up Access VBA with tight infinite loops, include the **DoEvents** command in loops during the testing stage, as in the following example:

```
Do While Temp%[ = True]
    Call TestProc(ExitLoop%)
    If ExitLoop% Then
        Exit Do
    End If
    DoEvents
Loop
```

DoEvents tests the Windows environment to determine whether any other event messages, such as a mouse click, are pending. If so, **DoEvents** allows the messages to be processed and then continues at the next line of code. You can then remove **DoEvents** to speed up the loop after your testing verifies that infinite looping cannot occur. The **DoEvents** statement should not be required in production 32-bit VBA code because threaded applications, such as Access 97, take advantage of preemptive multitasking in Windows 95 and Windows NT. However, not all 32-bit Windows applications are threaded, so **DoEvents** is likely to remain in use for some time.

Handling Run-Time Errors

No matter how thoroughly you test and debug your code, run-time errors appear eventually. Run-time errors are errors that occur when Access executes your code. Use the **On Error GoTo** instruction to control what happens in your application when a run-time error occurs. **On Error** is not a very sophisticated instruction, but it is your only choice for error processing in Access modules. You can branch to a label or ignore the error. The general syntax of **On Error...** follows:

```
On Error GoTo LabelName
On Error Resume Next
On Error GoTo 0
```

On Error GoTo *LabelName* branches to the portion of your code with the label *LabelName*:. *LabelName* must be a label; it cannot be the name of a procedure. The code following *LabelName*, however, can (and often does) include a procedure call to an error-handling procedure, such as ErrorProc, as in the following:

```
On Error GoTo ErrHandler
...
 [RepeatCode:
(Code using ErrProc to handle errors)]
...
GoTo SkipHandler
ErrHandler:
Call ErrorProc
[GoTo RepeatCode]
SkipHandler:
...
(Additional code)
```

Oracle Ex. 1009, p. 1015
Oracle v. DeBellis
IPR2024-00221

In this example, the `On Error GoTo` instruction causes program flow to branch to the `ErrHandler` label that executes the error-handling procedure `ErrorProc`. Ordinarily, the error-handler code is located at the end of the procedure. If you have more than one error handler or if the error handler is in the middle of a group of instructions, you need to bypass it if the preceding code is error-free. Use the `GoTo SkipHandler` statement that bypasses `ErrHandler:` instructions. To repeat the code that generated the error after `ErrorProc` has completed its job, add a label such as `RepeatCode:` at the beginning of the repeated code, and then branch to the code in the `ErrHandler:` code. Alternatively, you can add the keyword `Resume` at the end of your code to resume processing at the line that created the error.

`On Error Resume Next` disregards the error and continues processing the succeeding instructions.

After an `On Error GoTo` statement executes, it remains in effect for all succeeding errors until another `On Error GoTo` instruction is encountered or error processing is explicitly turned off with the `On Error GoTo 0` form of the statement.

If you do not trap errors with an `On Error GoTo` statement or if you have turned error trapping off with `On Error GoTo 0`, a dialog with the appropriate error message appears when a run-time error is encountered.

If you do not provide at least one error-handling routine in your Access VBA code for run-time applications, your application quits abruptly when the error occurs.

Detecting the Type of Error with the *Err* Function

The `Err` function (no arguments) returns an integer representing the code of the last error or returns 0 if no error occurs. This function is ordinarily used within a `Select Case` structure to determine the action to take in the error handler based on the type of error incurred. Use the `Error$()` function to return the text name of the error number specified as its argument, as in the following example:

```
ErrorName$ = Error$(Err)
Select Case Err
    Case 58 To 76
        Call FileError 'procedure for handling file errors
    Case 281 To 297
        Call DDEError 'procedure for handling DDE errors
    Case 340 To 344
        Call ArrayError 'procedure for control array errors
End Select
Err = 0
```

Some of the error codes returned by the `Err` function are listed in the Error Codes topic of the Access help file. Choose Help, Contents and Index to display the Help Topics dialog. Then type **Error Codes** in the text box of the Index page, and press Enter. Double-click Error Codes in the Topics Found dialog to display a help window with an abbreviated list of code numbers and their descriptions. Click the underlined description text to display a window that describes each error code in detail.

Oracle Ex. 1009, p. 1016
Oracle v. DeBellis
IPR2024-00221

You can substitute the actual error-processing code for the **Call** instructions shown in the preceding example, but using individual procedures for error handling is the recommended approach. The **Err** *statement* is used to set the error code to a specific integer. This statement should be used to reset the error code to 0 after your error handler has completed its operation, as shown in the preceding example.

The **Error** statement is used to simulate an error so that you can test any error handlers you write. You can specify any of the valid integer error codes or create a user-defined error code by selecting an integer that is not included in the list. A user-defined error code returns `User-defined error` to **Error$**().

Using the *Error* Event in Form and Report Modules

Access includes an event, `Error`, that is triggered when an error occurs on a form or report. You can use an event-handling procedure in a form or report to process the error, or you can assign a generic error-handling function in an Access module to the `Error` event with an `=ErrorHandler()` entry in order to call the `ErrorHandler()` function.

When you invoke an error-handling function from the `Error` event, you need to use the **Err** function to detect the error that occurred and take corrective action, as described in the preceding section.

Exploring the Module Window

You write Access VBA functions and procedures in the Module window. To display a Module window, click the Module tab of the Database window, and then double-click the name of the module you want to display. To open a new Access VBA module, click the New button. A Module code-editing window appears, as shown in Figure 28.8. This figure shows part of the `IsItAReplica()` function of the Startup module that is included in Access 97's Northwind.mdb example database. You choose the function or procedure to display from the procedures drop-down list at the upper-right portion of the Module window. The Module window incorporates a text editor, similar to Windows Notepad, in which you type your Access VBA code. Access VBA color-codes keywords and comments.

The Access VBA code in Figure 28.8 demonstrates two principles of writing code in any language: adding comments that explain the purpose of the statements and using indentation to make your code more readable. Comments in Access VBA are preceded with an apostrophe (**'**); alternatively, use the prefix **Rem** (for "remark" in earlier versions of BASIC) to indicate that the text on the line is a comment. **Rem** must be the first statement on a line (unless preceded by a colon that separates statements), but the apostrophe prefix can be used anywhere in your code.

Comments that precede the code identify the procedure, explain its purpose, and indicate the function or procedure that calls the code. If the function returns a value, a description of the returned value and its data type are included.

Oracle Ex. 1009, p. 1017
Oracle v. DeBellis
IPR2024-00221

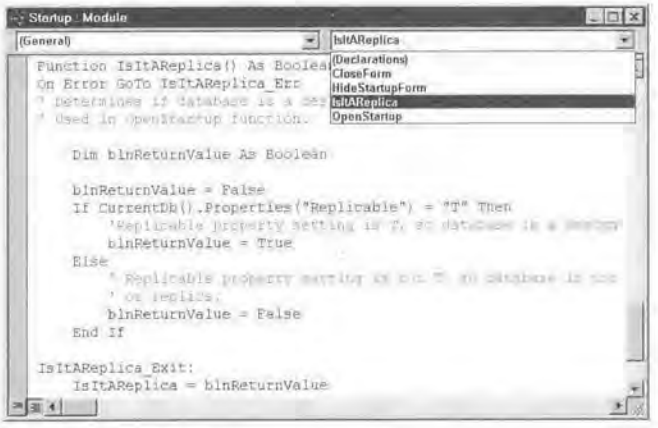


FIG. 28.8 Opening an Access VBA function in the Module window.

The Toolbar of the Module Window

Table 28.3 lists the purpose of each item in the toolbar of the Module window and the menu commands and key combinations that you can substitute for toolbar components. Buttons marked with an asterisk (*) in the Items column are new with Access 97 or 95. Buttons whose design or name has changed but perform the same function in Access 97 as in Access 2.0 are not marked with an asterisk.

Table 28.3 Elements of the Module Window's Toolbar

Button	Item	Alternate Method	Purpose
Module	Insert Module*	Insert, Module	Creates a new, empty module. Click the down arrow next to this button to create a new class module or to insert a new procedure or function.
	Save	File, Save	Saves changes to the current module.
	Print*	File, Print	Prints the content of the module as text.
	Cut	Edit, Cut	Cuts selected text from the module and places it on the Windows Clipboard.
	Copy	Edit, Copy	Copies selected text from the module and places it on the Windows Clipboard.
	Paste	Edit, Paste	Pastes the contents of the Window Clipboard into the module at the current position of the caret.
	Find*	Edit, Find	Similar to the Find feature used in Table or Form view; allows you to search for a specific word or phrase in a module.

Oracle Ex. 1009, p. 1018
Oracle v. DeBellis
IPR2024-00221

Button	Item	Alternate Method	Purpose
	Undo	Edit, Undo	Rescinds the last keyboard or mouse operation performed, if possible.
	Redo*	Edit, Redo	Rescinds the last undo operation, if possible.
	Go/Continue*	Run, Go/Continue or press F5	Starts the execution of the current procedure, or continues executing a procedure after the execution of a procedure has been halted by a break condition or after use of the Single Step or Procedure Step button.
	End*	Run, End	Terminates execution of a procedure.
	Reset	Run, Reset or press Shift+F5	Terminates execution of an Access VBA procedure and reinitializes all variables to their default values.
	Debug Window*	View, Debug Window	Opens the Debug Window.
	Object Browser	View, Object Browser	Opens the Object Browser dialog.
	Quick Watch*	Tools, Instant Watch or Shift+F9	Adds a watch expression to the Debug window that causes a break in execution when the expression returns **True**.
	Call Stack	View, Call Stack	Displays the Calls dialog that lists all procedures called prior to reaching a breakpoint in your code.
	Compile Loaded Modules	Debug, Compile Loaded Modules	Creates pseudo-code (p-code) from the text version of all Access modules and all form and report modules currently open.
	Toggle Breakpoint	Debug, Toggle Breakpoint or press F9	Toggles a breakpoint at the line of code in which the caret is located. Breakpoints are used to halt execution at a specific line. If a breakpoint is set, the Breakpoint button turns it off.
	Step Into	Debug, Step Into or press F8	Moves through an Access VBA procedure one statement (line) at a time.
	Step Over	Debug, Step Over or press Shift+F8	Moves through an Access VBA procedure one subprocedure at a time.
	Step Out*	Debug, Step Out or Ctrl+Shift+F8	Executes the remaining statements in a VBA procedure or function at full speed, returning to single-step mode at the point after the current procedure or function was called.
	Database Window	Window, 1	Opens the Database window.

(continues)

Oracle Ex. 1009, p. 1019
Oracle v. DeBellis
IPR2024-00221

Table 28.3 Continued

Button	Item	Alternate Method	Purpose
	Office Assistant	Help, Microsoft Access Help	Starts the Office Assistant for help.
N/A	Object List*	None	Displays a list of objects in form or report modules. Only (General) appears for Access modules.
N/A	Procedure List	None	Displays a function or procedure in a module. Select the procedure or event name from the drop-down list. Procedures are listed in alphabetical order by name.

Module Shortcut Keys

Additional shortcut keys and key combinations listed in Table 28.4 can help you as you write and edit Access VBA code. Only the most commonly used shortcut keys are listed in Table 28.4.

Table 28.4 Primary Key Combinations for Entering and Editing Access VBA Code

Key Combination	Purpose
F3	Finds next occurrence of a search string.
Shift+F3	Finds previous occurrence of a search string.
F9	Sets or clears a breakpoint on the current line.
Ctrl+Shift+F9	Clears all breakpoints.
Tab	Indents a single line of code by four (default value) characters.
Tab with selected text	Indents multiple lines of selected code by four (default) characters.
Shift+Tab	Outdents a single line of code by four characters.
Shift+Tab with selected text	Outdents multiple lines of selected code by four characters.
Ctrl+Y	Deletes the line on which the caret is located.

You can change the default indentation of four characters per tab stop by choosing Tools, Options. Click the Module tab and then enter the desired number of characters in the Tab Width text box.

Menu Commands in the Module Window

Menu commands to perform operations not included in Table 28.3 are listed in Table 28.5. Menu commands that are common to other database objects, such as File, Save, are not included in the table.

Oracle Ex. 1009, p. 1020
Oracle v. DeBellis
IPR2024-00221

Table 28.5 Module Window Menu Commands

Menu	Commands	Purpose
File	Save As Text	Displays the Save Text dialog to save the module's code to a text file in ASCII format.
	Replace	Replaces all or selected occurrences of the text entered in the Find text box with the text in the Replace text box.
View	Procedure Definition	If the caret is on a line that contains a procedure call, it finds and displays the called procedure.
Insert	File	Opens the Insert File dialog to allow insertion of code contained in a text file.
Debug	Run to Cursor	Continues execution until reaching the line above the position of the caret.
	Set Next Statement	Sets the next statement to be executed, by passing other code.
	Show Next Statement	Displays the next statement to be executed.
	Clear All Breakpoints	Clears all breakpoints set in the module. Active only when breakpoint(s) are set. (See "Adding a Breakpoint to the `IsLoaded()` Function" in this chapter.)
	Compile All Modules	Compiles all modules in the database.
	Add Watch	Opens the Add Watch dialog to add a watchpoint.
	Edit Watch	Opens the Edit Watch dialog to modify or delete a watchpoint.
Tools	ActiveX Controls	Opens the ActiveX Controls dialog to display a list of registered ActiveX Controls, and provides for the registration of new ActiveX Controls.
	References	Opens the References dialog for adding or deleting references to objects exposed by libraries.

The Access VBA Help System

Microsoft provides an extensive, multilevel Help system to help you learn and use Access VBA. The majority of the help topics for Access VBA are supplied by a generic VBA help file that's applicable to all flavors of VBA. If you place the caret on a keyword or select a keyword and then press the F1 key, for example, a help window for the keyword appears. If you click the "Example" hotspot under the name of the keyword and an example that is specific to Access VBA is available, the Topics Found dialog appears to let you choose between the generic VBA and Access VBA example (see Figure 28.9). Double-clicking the topic with the (`Microsoft Access`) suffix displays the example that is specific to Access VBA (see Figure 28.10).

If you press F1 when the caret is not located on a keyword, the Help system displays an `Ambiguous selection` topic. Choosing Help, Contents and Index and then clicking the Contents tab displays a list of "books" that comprise the entire Access 97 Help system. To obtain additional information on VBA programming, double-click the Microsoft Access and Visual Basic for Applications Reference book. Then double-click the topic you want, such as Keywords by Task. Expanding the Keywords by Task item displays a list of

Oracle Ex. 1009, p. 1021
Oracle v. DeBellis
IPR2024-00221

chapters (represented by a "?" icon) that are devoted to specific programming subjects (see Figure 28.11). When you double-click the chapter icon, a keyword summary for the topic appears (see Figure 28.12). Clicking an underlined keyword hotspot displays the corresponding help topic.

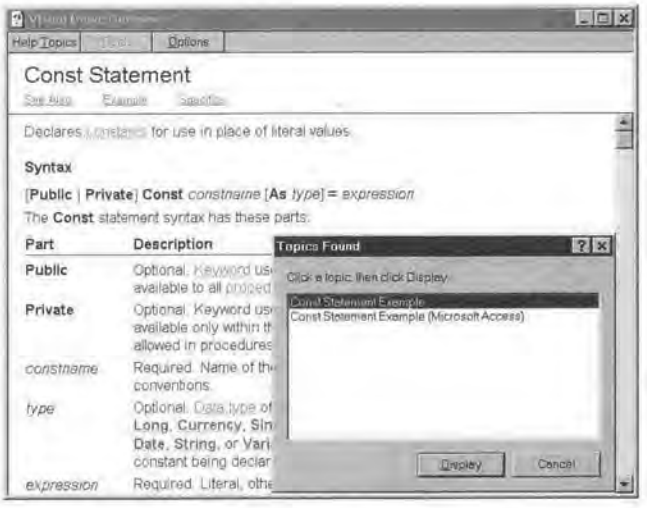


FIG. 28.9 A help window for the **Const** keyword with generic VBA and Access VBA examples available.

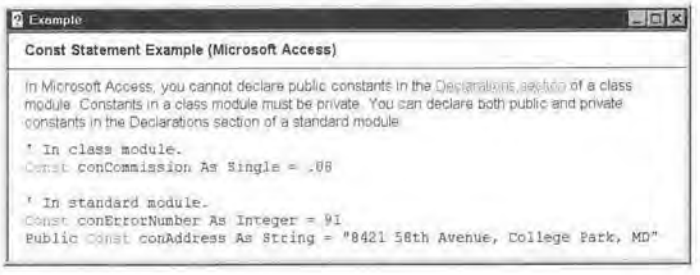


FIG. 28.10 The example that is specific to Access VBA for the **Const** keyword.

The Access VBA Compiler

Programming languages, such as Pascal and C, use compilers. *Compilers* are applications that convert the code statements—*source code*—you write into instructions that the computer can understand—*object code*. These languages use punctuation symbols to identify where statements begin and end; Pascal, for example, uses a semicolon to tell the compiler that a statement is complete. Separating code into individual statements and determining which words in a statement are keywords is called *parsing the code.*

Oracle Ex. 1009, p. 1022
Oracle v. DeBellis
IPR2024-00221



FIG. 28.11 Help topics for VBA tasks beginning with the letters A through E.

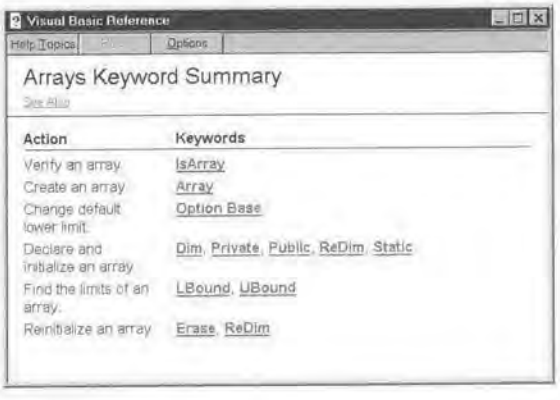


FIG. 28.12 The Keyword Summary help topic for tasks related to arrays.

After you compile Pascal or C code, you link the code with *libraries* to create an *executable file*. *Libraries* are additional object code that perform standard operations, such as mathematical calculations. Access libraries, such as Utility.mda, have a similar purpose. An executable file for a Windows application has the extension .exe; you run executable files as independent applications. Compiling and linking an application, especially a Windows application, can be a complex process. Microsoft Visual C++, Professional Edition, for example, requires more than 60M for a typical installation. (A minimum installation consumes about 10M.)

Traditional BASIC languages, such as QBasic as well as the native dBASE and Paradox programming languages, employ interpreters to execute code. An *interpreter* is an application that reads each line of code, translates the code into instructions for your computer,

Oracle Ex. 1009, p. 1023
Oracle v. DeBellis
IPR2024-00221

and then tells the computer to execute these instructions. The interpreter parses code line by line, beginning with the first non-blank character on a line and ending with the newline pair, carriage return (CR, `Chr$(13)`), and line feed (LF, `Chr$(10)`). A newline pair is created by pressing the Enter key. Compilers for the dBASE language, such as the Borland dBASE compiler and CA-Clipper, also use the newline pair to indicate the end of a statement.

You must execute interpreted code within the application in which the code was created. You run QBasic code, for example, from QBasic—not as a stand-alone application. An interpreter's advantage is that it can test the statements you enter for proper syntax as you write them. Compiled languages don't issue error messages until you compile the source code to object code. Another advantage of interpreters is that you don't need to go through the process of compiling and linking the source code every time you make a change in the code. Unfortunately, interpreters usually execute code more slowly than the computer executes a compiled (.exe) application.

Access VBA combines the features of both a compiler and an interpreter. VBA interprets the code you write when you terminate a line with the Enter key. If possible, the VBA interpreter corrects your syntax; otherwise, you receive a syntax error message that usually is accompanied by a suggestion for correcting the mistake. Each line of code, therefore, usually contains a syntactically correct statement. (In some cases, you must run the "compiler" to interpret the source code.) Most languages, such as xBase and Paradox, allow you to continue a statement on another line; xBase, including Clipper, uses the semicolon for this purpose. Access VBA offers a statement continuation character: a space followed by an underscore.

After you write the source code, the VBA interpreter converts this code into a cross between interpreted and object code known as pseudo-code, or p-code. Pseudo-code runs faster than conventional interpreted code. Access compiles to p-code the code you write or modify the first time that the code is used in an application. Access discovers most errors not caught during entry as the code compiles. You can choose Run, Compile All Modules to force the VBA interpreter to compile all of the code in Access modules and in form/report modules. Forcing compilation before running the code—especially if the form or macro that executes the code takes a long time to load—can save substantial time during development.

Examining the Utility Functions Module

One recommended way to learn a new programming language is to examine simple examples of code and analyze the statements used in the example.

The sections that follow show how to open a module, display a function in the Module window, add a breakpoint to the code, and then use the Debug window to execute the function.

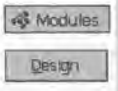
Oracle Ex. 1009, p. 1024
Oracle v. DeBellis
IPR2024-00221

Adding a Breakpoint to the *IsLoaded()* Function

When you examine the execution of Access VBA code written by others, and when you debug your own application, breakpoints are very useful. This section explains how to add a breakpoint to the `IsLoaded()` function so that the Suppliers form stops executing when the Suppliers macro calls the `IsLoaded()` function and Access displays the code in the Module window.

To add a breakpoint to the `IsLoaded()` function, follow these steps:

1. Display the Database window, click the Modules tab, select the Utility Functions module, and then click Design to make the Module window active.



2. Place the caret on the line that begins with `If SysCmd(acSysCmdGetObjectState,...)`.
3. Click the Breakpoint button of the toolbar, choose Run, Toggle Breakpoint, or press F9. The breakpoint you create is indicated by changing the display of the line to reverse red and by the placement of a red dot in the margin indicator at the left of the window (see Figure 28.13).



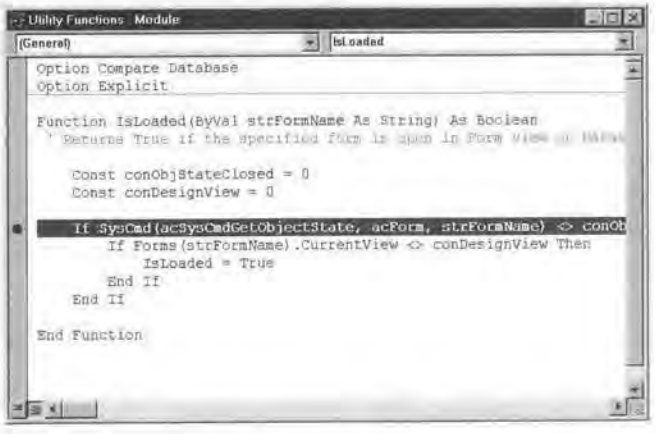


FIG. 28.13 The `IsLoaded()` function with a breakpoint set.

4. Close the Utility Functions module and open the Suppliers form to execute the `Form_Open` procedure attached to the `On Open` event of the form. When the Suppliers form's `Form_Open` procedure calls the `IsLoaded()` function, the execution of `IsLoaded()` begins with the `Const conObjStateClosed = 0` line and halts at the line with the breakpoint. When execution encounters a breakpoint, the module containing the breakpoint opens automatically. The line with the breakpoint is enclosed in a highlighted rectangle (see Figure 28.14).

Oracle Ex. 1009, p. 1025
Oracle v. DeBellis
IPR2024-00221

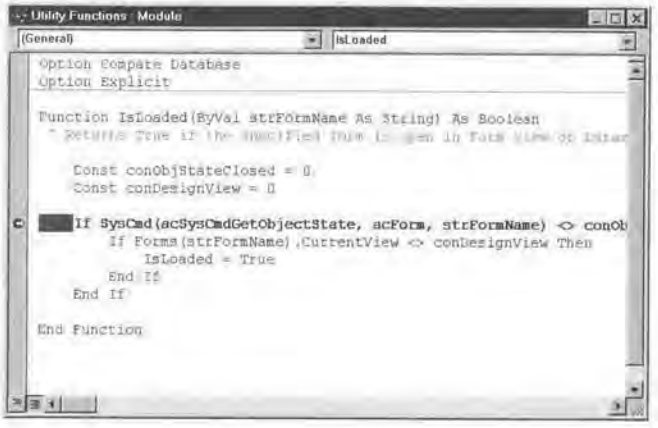


FIG. 28.14 The `IsLoaded()` function when the breakpoint is reached and the Utility Functions module opens.



5. Click the Go/Continue button to resume execution of the VBA code. Access displays the Suppliers form.

6. Close the Suppliers form to execute the `Form_Close` procedure that is attached to the form's `On Close` event. When the Suppliers form's `Form_Close` procedure calls the `IsLoaded()` function, execution occurs as described in step 4, and the `IsLoaded()` function again halts at the line with the breakpoint.

The element of the Forms collection to be tested is specified by `strFormName`. Access's `SysCmd()` function tests the `Forms` collection for the value of `strFormName` with the `SysCmd(acSysCmdGetObjectState, acForm, strFormName)` statement. If the specified form is open in either Form or Design view, `SysCmd()` returns **`True`**. The second test checks the value of the `CurrentView` property of the form to see whether the form is open in Form view. If the result of the `CurrentView` comparison is **`True`**, the `IsLoaded = True` line executes and the `IsLoaded()` function returns **`True`** to the calling procedure, in this case the Suppliers form's `Form_Open` procedure. (When `IsLoaded()` is called as the form is opened, it actually returns **`False`**, because the form isn't yet loaded.) The `IsLoaded()` function is called again by the Suppliers form's `Form_Close` procedure, which is attached to the `On Close` event. At this time, because the Suppliers form is loaded, the `IsLoaded()` function does return **`True`**.

Using the Debug Window

In the "Using the Debug Window" section of Chapter 9, "Understanding Operators and Expressions in Access," you learned to use the Debug window to display the results of computations and values returned by functions. The Debug window also is useful when you want to display the value of variables when the breakpoint is encountered. To display the value of local (procedure-level) variables, the execution of your code must be halted at a breakpoint beyond the point at which the variables are assigned their values.

Oracle Ex. 1009, p. 1026
Oracle v. DeBellis
IPR2024-00221

To open the Debug window and display the value of a variable while execution is halted at the breakpoint shown in Figure 28.14, and after completing step 6 of the preceding instructions, follow these steps:

1. Click the Debug Window button of the toolbar or choose View, Debug Window.



2. Click the Locals tab of the Debug window to display a split window. The upper portion of the Locals window displays the value of all constants and variables declared locally in the current function or procedure (see Figure 28.15).
3. Type **? strFormName** in the bottom window of the Locals page and then press Enter to display the value of the `strFormName` variable (see Figure 28.15). The `?` symbol is shorthand for the reserved word `Print`. Using the `?` symbol to print specific variable values or the results of test expressions in the Debug window can be a useful technique as you develop your code.
4. Click the Step Into button of the toolbar; choose Run, Single Step; or press F8 to continue the execution of the function, one line at a time. The **`End If`** statement is encountered. Press F8 again and execution moves to the **`End Function`** line. Press F8 again and execution proceeds to the **`Sub`** `Form_Close` procedure of the Suppliers form.



5. Press F5 twice to continue the execution of the code, closing the Suppliers form.
6. Choose Run, Clear All Breakpoints, or press Ctrl+Shift+F9 to toggle the breakpoint off.

When you write your own code, the Debug window is very handy for debugging purposes.

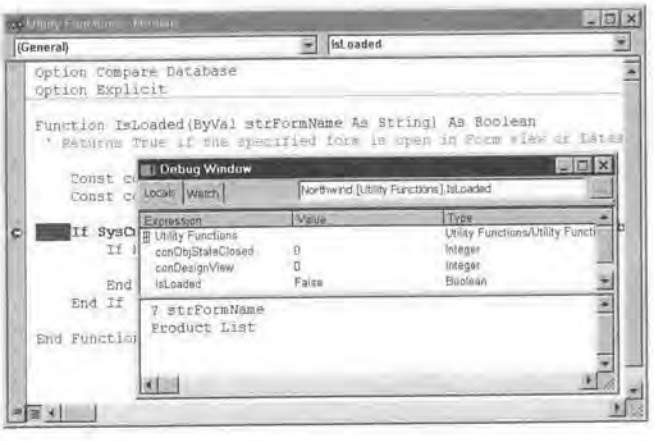


FIG. 28.15 Displaying a variable's value in the Debug window.

Oracle Ex. 1009, p. 1027
Oracle v. DeBellis
IPR2024-00221

Printing to the Debug Window with the *Debug* Object

When you need to view the values of several variables, you can use the **Debug** object to automate printing to the Debug window. If you add the **Debug** object to a function that tests the names of each open form, you can create a list in the Debug window of all the forms that are open.

To create a `WhatsLoaded()` function to list all open forms, follow these steps:

1. Load three or more forms by repeatedly opening the Database window, clicking the Form tab, and double-clicking a form icon. The Customers, Categories, Employees, and Main Switchboard forms are good choices because these forms load quickly.
2. Choose Module: Utility Functions from the Window menu and type **Sub WhatsLoaded()** below the End Function line of the `IsLoaded()` function. The Access VBA interpreter adds the **End Sub** statement for you automatically.
3. Type the following code between the **Sub**... and **End Sub** lines:

```
Dim intCtr As Integer
For intCtr = 0 To Forms.Count - 1
    Debug.Print Forms(intCtr).FormName
Next intCtr
```

The **For...Next** loop iterates the Forms collection. The **Debug.Print** statement prints the name of each open form in the Debug window.

> **Note**
>
> Access 97 contains a powerful new feature to help you write VBA code. The interpreter monitors each line of code as you type it in. When you type variable declarations, use built-in Access and VBA functions, or use object methods and properties in your code, the interpreter displays a popup window to help you select appropriate values. Figure 28.16 shows the popup list window that appears after you type the **As** keyword in the first **Dim** statement of the code you enter in step 3. For procedures, functions, and methods, the popup help window lists all of the arguments for the procedure, function, or method, so you don't have to remember all of the possible arguments. You can turn this feature on and off by choosing Tools, Options and then selecting or clearing the Auto Quick Info check box on the Module tab of the Options dialog.

4. Click the Debug Window button to open the Debug window. Type **WhatsLoaded** and press Enter. The name of each form is added to the Debug window by the `Debug.Print` statement (see Figure 28.17).
5. Close and don't save changes to the Utility Functions module. Then close the other forms that you opened for this example.

Oracle Ex. 1009, p. 1028
Oracle v. DeBellis
IPR2024-00221

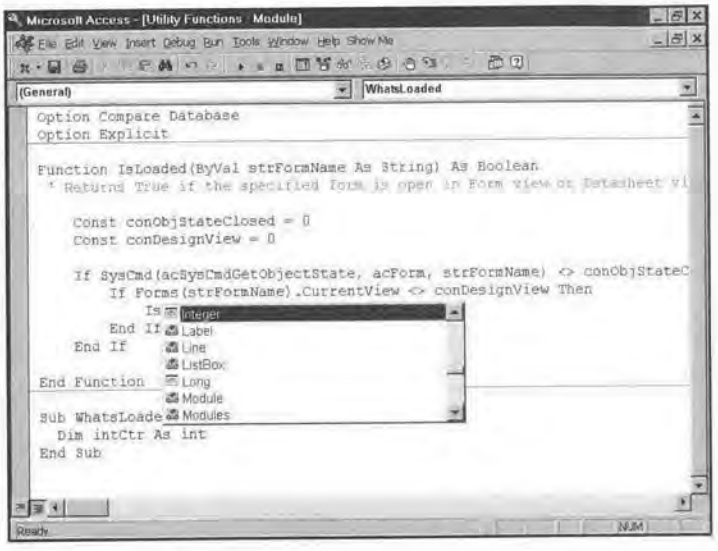


FIG. 28.16 The popup list of data types displayed as you enter the **Dim** statement.

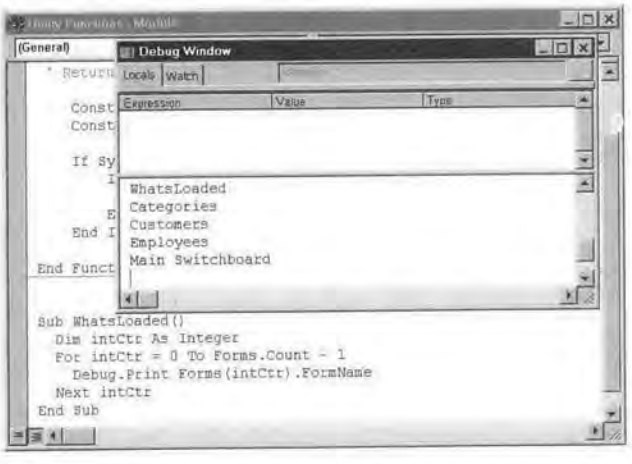


FIG. 28.17 The Debug object used to print values of variables to the Debug window.

The **Debug.Print** statement is particularly useful for displaying the values of variables that change when you execute a loop. When you have completed the testing of your procedure, you delete the **Debug** statements.

Using Text Comparison Options

Tests of text data in fields of tables, query result sets, and Recordset objects against **String** or **Variant** text data in modules depends on the value of the **Option** Compare... statement, which appears in the Declarations section of the Utility Functions module.

Oracle Ex. 1009, p. 1029
Oracle v. DeBellis
IPR2024-00221

To determine how text comparisons are made in the module, you can use any of the following statements:

- **Option** Compare Binary comparisons are case-sensitive. Lowercase letters are not equivalent to uppercase letters. To determine the sort order of characters, Access uses the character value assigned by the Windows ANSI character set.
- **Option** Compare Text comparisons are not case-sensitive. Lowercase letters are treated as the equivalent of uppercase letters. For most North American users, the sort order is the same as **Option** Compare Binary, ANSI. Unless you have a reason to specify a different comparison method, use **Option** Compare Text.
- **Option** Compare Database comparisons are case-sensitive, and the sort order is that specified for the database.

Option Compare Binary is the default if you do not include an **Option** Compare... statement in the Declarations section of the module. However, Access adds **Option** Compare Database to the Declarations section when you create a new module, overriding the default. Binary and Database are keywords in Access VBA, but these words do not have the same meaning when used in the **Option** Compare... statement. For compatibility with changes in possible future releases of Access, you should not use Compare or Text as names of variables.

Writing Your Own Functions and Procedures

You do not need to be an Access VBA expert to write a user-defined function. The information you learned about operators and expressions in Chapter 9, "Understanding Operators and Expressions in Access," as well as the introduction to Access VBA statements presented in this chapter, allow you to create UDFs that supplement Access VBA's repertoire of standard functions.

After you master user-defined functions, you can try your hand at writing procedures to be executed as the result of an event. Access VBA procedures offer flexibility in manipulating database and control objects but require more knowledge of programming techniques than is needed to create UDFs. This section describes how you write a simple user-defined function.

User-defined functions are required when you need a conditional statement that is more complex than that accommodated by the IIf() function. An example is a fixed set of quantity discounts that apply to all or a group of products. You can create a table of quantity discounts; however, a user-defined function that returns the discount is easier to implement and faster to execute.

For this example, the discounts are 50 percent for 1,000 or more of an item; 40 percent for 500 to 999; 30 percent for 100 to 499; 20 percent for 50 to 99; 10 percent for 10 to 49; and no discount for purchases of fewer than 10 items. Discount structures of this type lend themselves well to **Select Case...End Select** structures.

Oracle Ex. 1009, p. 1030
Oracle v. DeBellis
IPR2024-00221

To create the user-defined function `sngDiscount()` that returns a fractional percentage discount based on the value of the argument, `intQuantity`, follow these steps:

Modules

1. Click the Modules button of the Database window, and double-click the Utility Functions module.

2. Position the cursor on the line following the last `End Function` statement, and type **`Function sngDiscount(intQuantity As Integer) As Single`**. After you press Enter, Access adds the `End Function` statement. This process is quicker than choosing Insert, Procedure.

   The `sng` prefix of the function name indicates to users of the function that it will return a value of the **`Single`** data type. The **`As Integer`** modifier for the `intQuantity` argument specifies that the function treats `intQuantity` as an **`Integer`** data type. If you do not specify the data type, your function assigns the default **`Variant`** data type to `intQuantity`. The **`As Single`** keywords at the end of the line tell Access VBA that the function's result should be returned as a **`Single`** data type. Whereas the `sng` prefix gives users of the function a clue as to the data type returned by the function, the function will return a **`Variant`** data type unless you specify a data type.

Tip

You don't need to enter the `Is` keyword because Access checks the syntax of each statement you write when you press Enter. If you omit the `Is` keyword, Access adds it for you.

3. Position the caret at the beginning of the blank line and press Tab. To establish the discount schedule, enter the following lines of code:

```
Select Case intQuantity
    Case Is >= 1000
        sngDiscount = 0.5
    Case Is >= 500
        sngDiscount = 0.4
    Case Is >= 100
        sngDiscount = 0.3
    Case Is >= 50
        sngDiscount = 0.2
    Case Is >= 10
        sngDiscount = 0.1
    Case Else
        sngDiscount = 0
End Select
```

Use the Tab key to duplicate the indentation illustrated in the preceding example. **`Case`** statements are executed from the first statement to the last, so quantities are entered in descending sequence. If `sngQuantity` = 552, the criterion is not met for **`Case Is`** `>= 1000`, so the next **`Case`** statement is tested. **`Case Is`** `>= 500` is satisfied, so `sngDiscount` receives the value 0.4 and execution proceeds directly to the **`End Select`** statement.

Oracle Ex. 1009, p. 1031
Oracle v. DeBellis
IPR2024-00221



4. To make sure that the function is acceptable to Access VBA's compiler, click the Compile Loaded Modules button of the toolbar or choose Run, Compile Loaded Modules. Access converts the entries to a form that it can process. When you compile the code, Access indicates any errors that are not caught by line-by-line syntax checking.



5. You do not need to create a table or macro to test the function; the Debug window performs this task adequately for sample code. If the Debug window is not open, click the Debug Window button of the toolbar or choose View, Debug Window.

6. Test the `sngDiscount()` function by entering **? sngDiscount(*Quantity*)** in the Debug window, and then press Enter. Substitute numeric values for *Quantity* (see Figure 28.18).

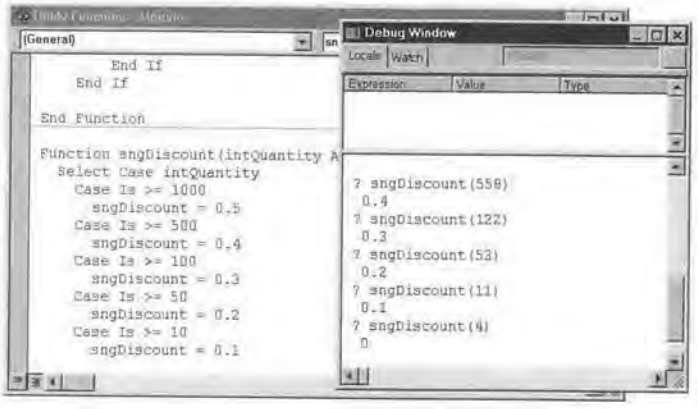


FIG. 28.18 Testing the `sngDiscount()` user-defined function.



7. To save the UDF, click the Save button of the toolbar or choose File, Save.

You can employ user-defined functions in expressions to compute values for calculated fields, to validate data entry, to construct queries, and to fulfill other purposes where an expression can be used.

Troubleshooting

An `Expected Shared or identifier` message box appears when entering a `Dim VariableName As DataType` statement.

You attempted to name a variable with a reserved word. For example, you receive the preceding error message if you attempt to create a **`Dim Option As Integer`** statement. **`Option`** is a VBA-reserved word. Change the name of the object, preferably using the data type tag prefix, as in `intOption`.

Oracle Ex. 1009, p. 1032
Oracle v. DeBellis
IPR2024-00221

Chapter 29

Understanding the Data Access Object Class



Access VBA lets you create and manipulate databases, tables, and queries using the Jet 3.5 database engine. Access 97 also offers a new alternative to the full-feature Jet 3.5 engine: ODBCDirect which uses the Remote Data Object (RDO) version 2.0 for connecting to client/server databases. This chapter introduces you to the many data access object (DAO) classes of Jet 3.5 and explains how you can manipulate the DAOs with Access VBA code. The chapter also describes how to use ODBCDirect, which is accessible only with VBA code.

No *Data Access Object* exists, per se; the term refers to all collections of Access 95 objects. The structure of the hierarchy of DAO classes was, to a major extent, designed to conform to the requirements and recommendations of the Object Linking and Embedding (OLE) 2.1 specification. Jet 3.5 is a very large in-process (OLE) Automation server; thus, much of this chapter uses OLE 2.1 terminology. Microsoft has adopted ActiveX as the name for its family of lightweight OLE components, primarily directed to Internet applications. Jet 3.5, to say the least, is a heavyweight; according to Microsoft, the size and resource consumption of Jet 3.5 is similar to that of SQL Server 6.5.

Note

Microsoft announced in the fall of 1996 that it plans to introduce a version of SQL Server that runs under Windows 95. The announcement caused several PC trade publications to run stories predicting the demise of Access when SQL Server becomes easily accessible to desktop (and even laptop) database users. The future Windows 95 version of SQL Server is intended as a high-performance substitute for the Jet database engine, not as a replacement for Access.

Understanding Objects and Object Collections

Access 97, like other members of the Office 97 productivity software suite, organizes programmable objects into a hierarchical structure of classes of

Oracle Ex. 1009, p. 1033
Oracle v. DeBellis
IPR2024-00221

objects. Two members of Access object classes that you have used extensively are `Forms` and `Reports`. You use these class names as identifiers, such as `Forms!FormName.PropertyName` and `Reports!ReportName!ControlName.PropertyName`, in expressions. In these expressions, `Forms` or `Reports` is the class name, and *FormName* or *ReportName* is the literal name of the member object of the class.

Groups of objects from the same class are called *object collections*. `Forms` is the collection of open `Form` objects, and `Reports` is the collection of open `Report` objects. Unopened objects do not appear in these collections. Collections are similar to arrays, except that collections consist of references (called *pointers*) to member objects. On the other hand, Access VBA arrays consist of elements with assigned values. Unlike arrays, collection members may appear or disappear without your intervention. As an example, the members of the `Forms` collection change as you open and close the `Form` objects of your application. Access 97 is also capable of creating user-defined collections with the `Dim colUserDefined As New Collection` statement. You then add user-defined objects to the collection by invoking the `AddItem` method of `Collection` objects.

Naming Standards for Object Collections

To conform to OLE 2.x standards for creating programmable objects, collections are named by the English-language plural of the class name of objects contained in the collection. As an example, Microsoft Graph 7 (MSGraph7) has `Axis` objects; the collection of these objects is called the `Axes` collection. Collections have relatively few properties; the most common is `Count`, the current number of members in the collection. You use the `Count` property in the `For intCtr = 0 To Forms.Count - 1...Next intCtr` statement to iterate (enumerate) the `Forms` collection. To *enumerate* a collection usually means to test all (or a particular set of) collection members for values of a specified property.

Figure 29.1 shows the hierarchy of the data access objects in Access 97. The uppermost member of the hierarchy is the `DBEngine` object, representing the Jet database engine, which contains the `Workspaces` and `Errors` collections. The `Workspaces` collection contains `Databases`, `Users`, and `Groups` collections, plus the new `Connections` collection. Exposing a collection makes the members of the collection accessible to Access VBA code. As an example, the `Errors` collection contains one or more `Error` objects when an error is encountered by Jet 3.5. Notice that the `Forms` and `Reports` collections do not appear in Figure 29.1. The `Forms` and `Reports` collections, like the `Scripts` (macros) and `Modules` collections, are not DAOs. The `Forms` and `Reports` collections are defined by Access, not by Jet, and have two distinct identities:

- `Forms` and `Reports` collections, which contain only members of the `Forms` and `Reports` `Containers` collection that are currently open in your application.
- `Documents` collections, which consist of all saved forms and reports, together with modules and macros are members of the `Containers` collection. The primary use for `Document` objects is to set security permissions with Access VBA. `Forms` and `Reports` collections have a much larger repertoire of properties than their corresponding `Document` objects. (The DAO defines the `Databases`, `Tables`, and `Relations` documents that refer to saved databases, tables and queries, and relations, respectively.)

Oracle Ex. 1009, p. 1034
Oracle v. DeBellis
IPR2024-00221

In the case of `Containers` and `Documents`, Access departs from the OLE 2.x rules for naming collections. `Documents` are collections that are members of the `Containers` collection; no "`Container`" objects are in the collection. The list in the Database window enumerates the `Name` property of members of the class in the `Documents` collection that you select by clicking a tab. When you refer to `Form` and `Report` objects in expressions, you use the `Forms` and `Reports` collections, not the member of the `Containers` collection with the same name.

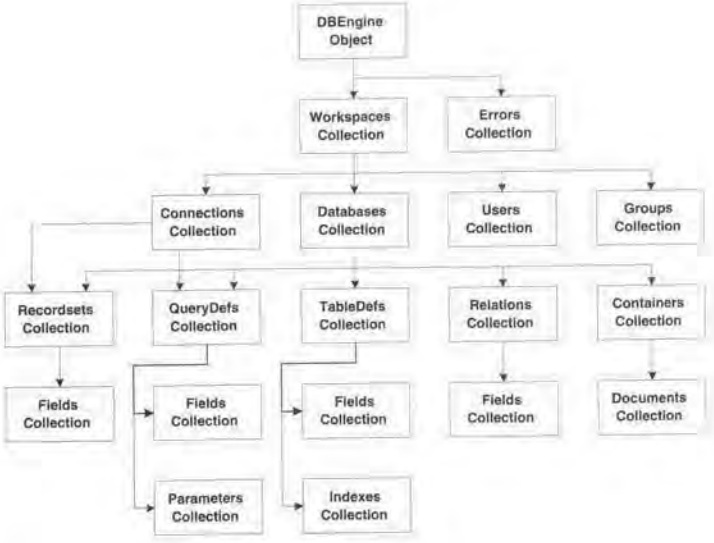


FIG. 29.1 The hierarchy of Access 97's data access objects.

The `Connections` collection, which is new to Access 97, contains the set of open ODBCDirect `Connection` objects. A `Connection` object is a single ODBC connection to a client/server RDBMS, such as SQL Server, and occupies a position in the DAO hierarchy that is equivalent to the position of the `Databases` collection. A `Connection` object is similar to the RDO's `rdoConnection` object, so both ODBCDirect and the RDO are discussed in the section "Using ODBCDirect and the RDO" near the end of this chapter.

Creating a Reference to the Data Access Object

VBA requires references to OLE objects that are exposed by components of OLE container applications. All flavors of VBA, including the Access, Excel, Project, and Visual Basic 5.0 versions, include references to application objects and to a common version of VBA. When you install Access 95, references are created automatically to the following four object libraries stored in various folders:

- Visual Basic for Applications (Vba332.dll, 32-bit VBA Version 5.0), which contains references to all of the commands and constants of VBA. The type library of this DLL is common to all members of Office 97 that support VBA. Vba332.dll is stored in the \Program Files\Common Files\Microsoft Shared\VBA folder.

Oracle Ex. 1009, p. 1035
Oracle v. DeBellis
IPR2024-00221

- Microsoft Access 97 Object Library (Msacc8.olb), which contains references to all Access-specific objects, Access-only VBA commands (for example, `DoCmd`), and constants. Msacc8.olb is stored in the \Program Files\Microsoft Office\Office folder.
- Microsoft DAO 3.5 Object Library (Dao350.dll), which contains references to all of the objects exposed by the Jet 3.5 database engine. Dao350.dll is stored in the \Program Files\Common Files\Microsoft Shared\DAO folder.
- Microsoft Office 8.0 Object Library (Mso97.dll), which contains references to objects and constants that Access shares with other members of Office 97, primarily `CommandBar` objects and constants. Like Msacc8.olb, Mso97.dll is stored in the \Program Files\Microsoft Office\Office folder.

References to object libraries (.olb), object type libraries (.tlb), and OLE dynamic link libraries (.dll) appear in the References dialog shown in Figure 29.2. To open the References dialog, open the Utility Functions module of Northwind.mdb in Design view, and choose Tools, References. The items that appear in the Available References list depend on the OLE-compliant applications and ActiveX controls installed on your computer. You cannot remove the references to or change the priority of Visual Basic for Applications or the Microsoft Access 8.0 Object Library. You can, however, click the check box for the Microsoft DAO 3.5 Object Library to remove references to the Jet 3.5 DAO. Removing the reference to Jet's DAO 3.5 doesn't unload the Jet database engine; most of Jet is required for Access 97 to run.

If you remove the DAO 3.5 reference, you cannot use data access objects in your VBA code. The references you establish with the References dialog apply only to VBA code, not to operations involving the Access user interface. You establish references to objects of other Automation servers and ActiveX controls to manipulate their Automation objects with VBA code, the subject of Chapter 31, "Exchanging Data with Automation and ActiveX Controls." All Automation client applications that use VBA as their application programming language have a References dialog similar to that of Access 97.

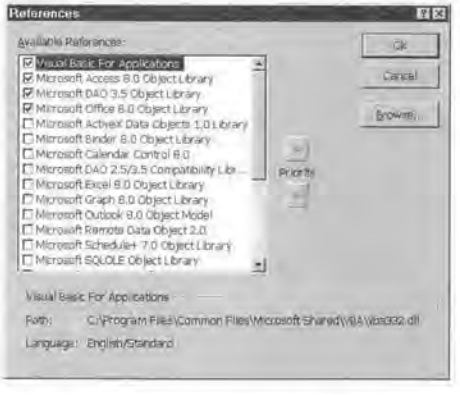


FIG. 29.2 Access 97's default VBA object references displayed by the References dialog.

Oracle Ex. 1009, p. 1036
Oracle v. DeBellis
IPR2024-00221

> **Note**
>
> The Available References list of the References dialog also includes a reference for the 32-bit version of the Microsoft DAO 2.5/3.5 Compatibility Library (Dao2535.tlb). You can substitute Dao2535.tlb for Dao350.dll to preserve compatibility with Access Basic DAO code in Access 1.x and 2.0 applications that you convert to Access 95. You cannot add references to both Dao350.dll and Dao2535.dll; you receive an error message if you attempt to do so. To assure compatibility of your VBA code with future versions of Access, it is advisable to use the Dao350.dll reference and change your VBA code to comply with Access 97 VBA syntax for data access objects.

Using the Object Browser

Access 97's upgraded Object Browser displays the objects exposed by OLE Automation servers to which references have been created, plus the properties and methods of the objects. To open Access 97's Object Browser dialog, press F2 and click the Object Browser button in the toolbar, or choose View, Object Browser. To choose the class of objects to view, open the upper drop-down list. You can select from the object classes supplied by the three references described in the preceding section, any additional references you have added, and objects in the current database (Northwind.mdb in this example). This chapter is devoted to data access objects, so select DAO.

The Object Browser is very useful for becoming acquainted with the names of objects exposed by the Jet 3.5 DAO and the properties and methods of data access objects. When you select an object or collection in the Classes list and a member function of the object or collection in the Members of '*ClassName*' list, the syntax for the method or property appears in a pane below the lists. Figure 29.3 shows the Object Browser displaying the syntax for the `OpenRecordset` method of the `Database` object. All members of Office 97 use the same Object Browser.

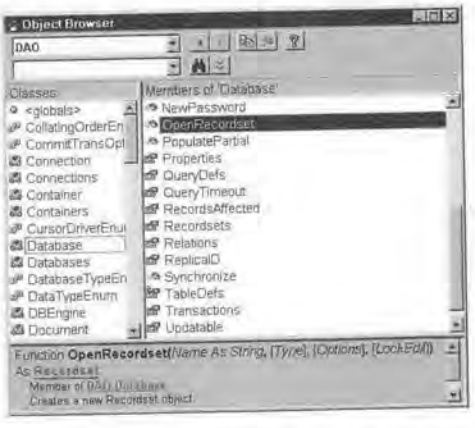


FIG. 29.3 Object Browser displaying the syntax for the `OpenRecordset` method of the `Database` object.

Oracle Ex. 1009, p. 1037
Oracle v. DeBellis
IPR2024-00221





Clicking the ? button of the toolbar displays the help topic for the method or property—in this case for the `OpenRecordset` method (see Figure 29.4). You can type a word for which to search in the lower combo list of the Object Browser, and then click the Search and Search List buttons to display a list of objects, methods, and properties where the word appears. Figure 29.5 shows part of the result for a search on "Recordset."

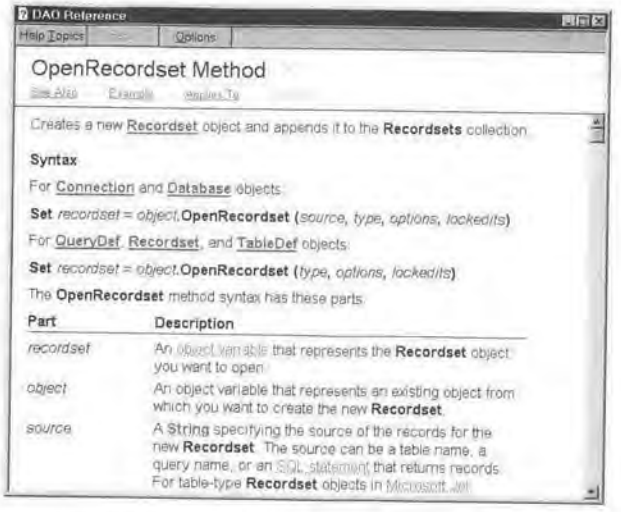


FIG. 29.4 The help topic for the `OpenRecordset` method opened by clicking the ? button of the Object Browser.



FIG. 29.5 Object Browser displaying part of the search result for "Recordset."

Oracle Ex. 1009, p. 1038
Oracle v. DeBellis
IPR2024-00221

Referring to Data Access Objects in VBA

Knowing the hierarchy of data access objects is critical to their manipulation by your Access VBA code. As an example, if you want to refer directly to a property value of the currently open database, you specify the full "path" through the hierarchy to your object and its properties or methods with an expression similar to the following:

```
strQDFName = DBEngine.Workspaces(0).Databases(0).QueryDefs(0).Name
```

The zeroth member of the `Workspaces` collection is the current `Workspace` object, also called the *current session* or *instance* of the Jet database engine. The current session comes into existence when you first launch Access. Similarly, `Databases(0)` specifies the database you open after launching Access, called the *current database*. `QueryDefs(0)` specifies the first `QueryDef` object of the `QueryDefs` collection. `Name` is a property of the `QueryDef` object. You can open additional `Workspace` objects and add one Database object or more to the newly created `Workspace` object with Access VBA code. Adding new members to DAO collections is described in the sections that follow.

> **Note**
>
> The currently open `Database` object, specified by `DBEngine.Workspaces(0).Databases(0)`, is the same object identified by the `CurrentDB()` function in all versions of Access. Access 2.0 introduced `Container(i)` syntax, as in
>
> ```
> Set dbCurrent = DBEngine.Workspaces(0).Databases(0)
> ```
>
> The preferred statement in Access 97 to refer to the database opened in Access is
>
> ```
> Set dbCurrent = CurrentDB()
> ```
>
> that points to the instance of the current database. Using the `CurrentDB()` function is advisable to assure compatibility with future versions of Access. (See the "Creating Object Variables" section further in this chapter.) The `CurrentDB()` function (method) is a member function of the Access `Application` object, not the DAO. Thus, `CurrentDB()` is not applicable when using Visual Basic 5.0, Excel VBA, Word VBA, or Project VBA.

A Database object can contain `TableDef`, `QueryDef`, `Recordset`, and `Relation` objects, as well as `Documents` collections in the `Containers` collection. A brief description of each of these objects, except the `Containers` collection, follows:

- `TableDef` objects represent the definition of saved tables. `TableDef` objects contain `Fields` and `Indexes` collections, which contain one member for each field and index of the table represented by the `TableDef` object. `TableDef` objects have several properties, such as `Name`, `RecordCount`, and `ValidationRule`, that apply to the table as a whole. The `Field` and `Index` objects have their own sets of properties.
- `QueryDef` objects represent the definition of saved queries and contain `Fields` (which might better have been named `Columns`) and `Parameters` collections. `QueryDef` objects have properties, such as `Name`, `SQL`, and `Updatable`.

Oracle Ex. 1009, p. 1039
Oracle v. DeBellis
IPR2024-00221

- `Recordset` objects, which have been mentioned throughout this book, represent virtual tables (images) that are stored in RAM. `Recordset` objects are said to be "created over" a table or a query result set. `Recordset` objects, which mimic the behavior of the underlying object, can be of the `Table`, `Dynaset`, or `Snapshot` type. As mentioned in earlier chapters of this book, the `Table`, `Dynaset`, and `Snapshot` object types appear in Access 97 for compatibility with Access 1.x code, but you must specify a reference to the Microsoft DAO 2.5/3.5 Compatibility Library to use these object types in VBA code. You should use Access 97's `Recordset` objects and the DAO 3.5 Object Library to assure compatibility with future versions of Access.
- `Relation` objects represent default relationships between fields of tables that, in most cases, you create in the Relationships window. `Relation` objects contain a `Fields` collection.

Note

If the size of a Recordset object exceeds the available RAM, the remainder of the object is placed in a temporary "spill file" in the \Windows\Temp folder.

Following are the three different methods of creating a reference to a member of a collection:

- The index method uses conventional array subscripts to specify a particular member of a collection by its position in the collection: *`CollectionName`*`(i)`, where `i` ranges from `0` to *`CollectionName`*`.Count - 1`. Using the index to the object is the fastest method of referring to members of collections. The problem with the index method is that the index of a particular member object is likely to change as you open (add) and close (remove) objects in the collection.
- The argument method uses a literal identifier, either a quoted string or a variable holding the value of a literal identifier: *`CollectionName`*`("`*`ObjectName`*`")` or *`CollectionName`*`(strObjectName)`. This method is preferred when you know the name of the member object.
- The literal method uses the bang operator to separate the literal name of the object from the collection name: *`CollectionName`*`!`*`ObjectName`*. If the literal name contains spaces or punctuation other than the underscore, enclose the literal in square brackets, as in *`CollectionName`*`![`*`Object Name`*`]`. This method is used primarily with `Form` and `Report` objects but is applicable to members of any collection. In this case, you can't use a variable in place of the member object's name. The literal method is the least flexible of the three.

Properties of and Methods Applicable to DAOs

Most objects have a Properties collection that you can use to enumerate the properties of the object. You can read (get) the value of properties and set the value of most properties of an object in Run mode. The Debug window is a handy tool to experiment with getting and setting the values of properties as well as invoking methods. The example statements in this section use the Debug window syntax to print the value of properties.

Oracle Ex. 1009, p. 1040
Oracle v. DeBellis
IPR2024-00221

The `? DBEngine.Workspaces(0).Properties.Count` statement returns 3, the number of `Workspace` object properties. You can obtain the name of a property with the *`CollectionName`*`(i).Properties(j).Name` statement, where `j = 0` to `CollectionName(i).Properties.Count - 1`. As an example, `? DBEngine.Workspaces(0).Properties(0)` returns the value of the `Name` property. `Name` is the zeroth (default) property of almost all DAOs. `? DBEngine.Workspaces(0).Properties(0)` returns `#DefaultWorkspace#`; `_Properties(1)` returns the value of `UserName`, `admin` for unsecure databases; and `_Properties(2)` returns the value of the `IsolateODBCTrans` property, `0 (False)`, as shown in Figure 29.6.

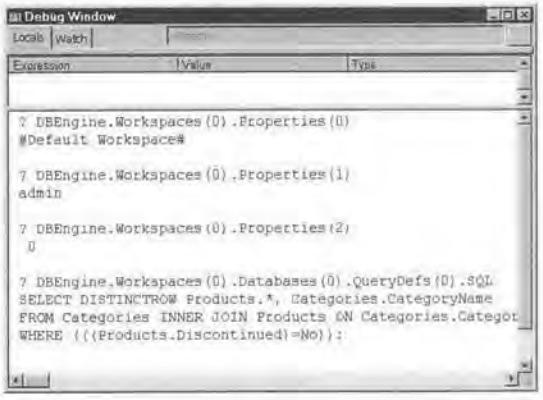


FIG. 29.6 Using the Debug window to print the values of properties of data access objects.

Note

In Access 2.0, the `Workspaces` object had four properties. `DBEngine.Workspaces(0).Properties(2)` held the `Password` property of the user. This property was write-only; you could set this value but not read it. The `Password` property has been removed from the `Workspaces` object of Access 97.

To read or set property values of objects lower than `Workspaces` in the hierarchy, you should declare an object variable and then assign the object variable the object you want to manipulate from its collection. Creating object variables is the subject of the next section, "Creating Object Variables." Declaring an object variable for objects deep in the hierarchy improves the performance of your application by eliminating the need of VBA to parse the upper object layers.

You can apply methods to members of collections, such as applying `Append` to add a new member to a collection. The `Properties` collection is interesting in this respect: Access 2.0 and later versions allow you to append user-defined properties to the Properties collection. Some properties of objects are called *Access-defined properties*; you must append the Access-defined property to the collection in order for Access to assign the property a value.

Oracle Ex. 1009, p. 1041
Oracle v. DeBellis
IPR2024-00221

Note

Publishing limitations preclude listing the properties and methods for each DAO. Use the Object Browser's ? button to display the online help topic for the object class (items only in the Classes/Modules list), as shown in Figure 29.7. You also can open the help topic for a specific member function (items in both the Classes/Modules and Methods/Properties lists), as illustrated by the preceding Figure 29.4. Using the Object Browser to open a help topic for a DAO is a much easier process than choosing Help, Microsoft Access Help Topics, selecting the Microsoft DAO book, and then attempting to find what you want in the book's table of contents.

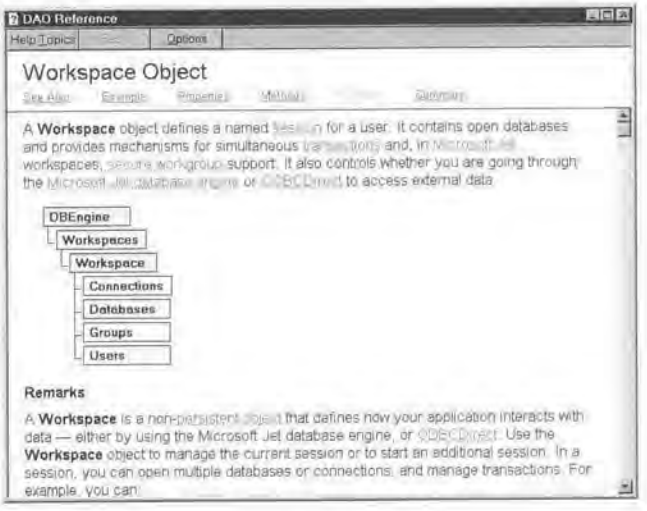


FIG. 29.7 The help topic for the `Workspace` object class.

Creating Object Variables

It's a good programming practice to declare object variables to represent a specified object lower than `Workspace` objects in the DAO hierarchy. All object variables must be declared explicitly. Object variables are declared with **`Private`** *`objName`* **`As`** *`ObjectType`* statements in the Declarations section of a module to create an object variable with module-level scope or in a procedure to create a local object variable. Use the **`Public`** reserved word to create a variable with global scope in a VBA code module. **`Private`** and **`Public`** replace the **`Dim`** and **`Global`** reserved words of Access Basic, although you can continue to use **`Dim`** and **`Global`**. The following code creates an object variable, `dbCurrent`, that refers to the current database:

```
Private dbCurrent As Database
Set dbCurrent = CurrentDB()
```

The **`Set`** reserved word is used to assign values to variables of object data types. The value of the variable is a reference (pointer) to a block of memory in which the object is located. Thus, `dbCurrent` is said to point to the current database.

Oracle Ex. 1009, p. 1042
Oracle v. DeBellis
IPR2024-00221

Once you've declared a Database variable, such as `dbCurrent`, you can use `dbCurrent` as a shorthand method of assigning pointers to the database objects. To refer to the `TableDefs` collection of the current database, which is a subclass of the `Database` object, use the following statements:

```
Private tdfCurrent As TableDef
Set tdfCurrent = dbCurrent.TableDefs("Categories")
```

As a general practice, you create a module-level variable of the Database type to point to the current database. If more than one module contains DAO manipulation code, substitute `Public dbCurrent As Database` for `Private dbCurrent As Database`. The subclass scope of your `dbCurrent` object depends on your use of the object subclasses in your code.

Using Object Variables in the Debug Window

You cannot declare variables explicitly in the Debug window; thus, you need to declare object variables in the Declarations section of a module and then enter `Set` statements in the Debug window to assign object pointers to the variables. Figure 29.8 provides examples of declaring object variables at the module level and of assigning pointer values to the variables in the Debug window. Database objects contain one `TableDef` object for each table listed in the Database window, and each `TableDef` object contains one `Field` object for each field of the table.

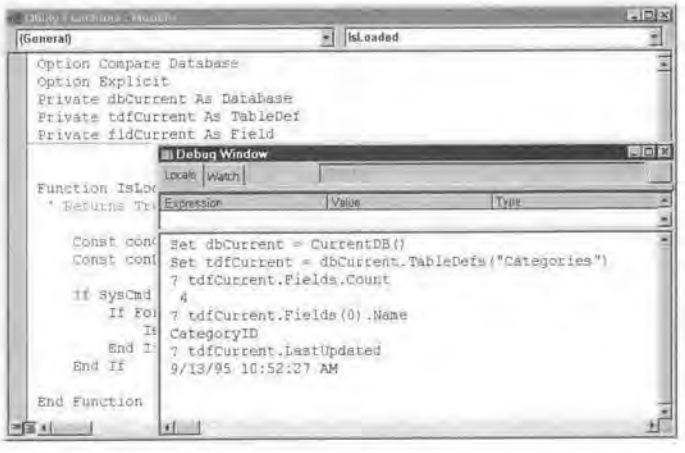


FIG. 29.8 Declaring and assigning values to object variables.

Creating New Data Access Objects

As mentioned in the section "Creating Object Variables" earlier in this chapter, Access 97 lets you create new DAOs with Access VBA code. The most common DAOs you create with Access VBA are `Recordset` and `QueryDef` objects. `QueryDef` objects, like `TableDef` objects, are called *persistent objects* because `QueryDef` and `TableDef` objects also are `Document` objects: `Query` and `Table` objects, respectively. `QueryDef` and `TableDef` objects are stored as

Oracle Ex. 1009, p. 1043
Oracle v. DeBellis
IPR2024-00221

elements of your .mdb file and appear in the Queries and Tables lists, respectively, of the Database window. `Recordset` objects are impersistent objects; they exist in memory only from the time you open them until they are closed. You can close a `Recordset` object by applying the `Close` method; `Recordset` objects are closed automatically when the variable that points to them goes out of scope. When you close a `Recordset` object, the memory that the object consumes is released to your application.

> **Note**
>
> Using Access VBA to create new `Workspace`, `Database`, `User`, `Group`, `TableDef`, and `Relation` objects is beyond the scope of this book. You can create these objects, with the exception of the `Workspace` object, much easier with Access's user interface. You also can create new `TableDef` objects with make-table queries or by using Jet SQL Data Definition Language statements.

Opening a New *Recordset* Object

You create a new `Recordset` object with the `OpenRecordset` method of `Database`, `TableDef`, and `QueryDef` objects. The general syntax of the `OpenRecordset` method has two forms:

```
Set rsdName = dbName.OpenRecordset(strSource[, intType[,_
  intOptions[, intLockEdits]]])
Set rsoName = objName.OpenRecordset([intType[, intOptions[, intLockEdits]]])
```

The first form, which is applicable only to `Recordset` objects created over `Database` objects, requires a value for the `strSource` argument. The value of the `strSource` argument can be the name of a `TableDef` object, or `QueryDef` object, or a Jet SQL statement. You also can use VBA's named arguments feature in the following statements:

```
Set rsdName = OpenRecordset(Name:=strSource[, Type:=intType[,_
  Options:=intOptions[, LockEdits:=intLockEdits]]])
Set rsoName = objName.OpenRecordset([Type:=intType[,_
  Options:=intOptions[, LockEdits:=intLockEdits]]])
```

Tables 29.1, 29.2, and 29.3 list values allowed for the `intType`, `intOptions`, and `intLockEdits` arguments for both of the preceding syntax examples. The values shown in the three tables are the names of predefined (intrinsic) global constants that are defined by the Jet 3.5 DAO. In earlier versions of Access, Access intrinsic global database constant names were uppercase and included underscore separators for readability, as in `DB_OPEN_TABLE`. Database intrinsic constants now are supplied by enumerations of the Jet 3.5 DAO and use the lowercase prefix `db` with intermediate capitalization to improve readability. A `MethodArgumentEnum` selection in the Class list of the Object Browser displays the names and values of applicable constants in the Members list (see Figure 29.9). In the case of the `intOptions` values listed in Table 29.2, you can combine the options you want with the VBA **`Or`** operator, as in `dbDenyWrite` **`Or`** `dbDenyRead`.

Oracle Ex. 1009, p. 1044
Oracle v. DeBellis
IPR2024-00221

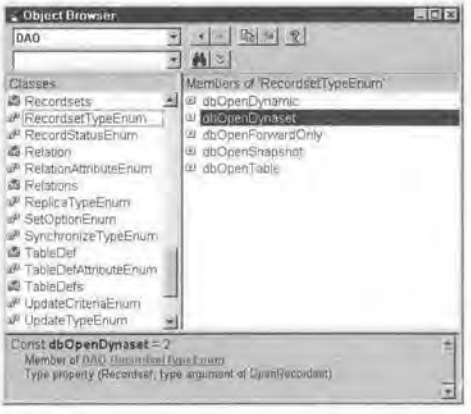


FIG. 29.9 Some of the predefined (intrinsic) constants supplied by the Jet 3.5 DAO.

Table 29.1 Values for the *intType* Argument of the *OpenRecordset* Method Comparing DAO 3.5 and Access 2.0 Intrinsic Constants

DAO 3.5	Access 2.0	Description of Type
`dbOpenTable`	`DB_OPEN_TABLE`	Table (default value for local `TableDef` source)
`dbOpenDynaset`	`DB_OPEN_DYNASET`	Dynaset (default value for `Database`, `QueryDef`, `Recordset`, or an attached `TableDef`)
`dbOpenSnapshot`	`DB_OPEN_SNAPSHOT`	Snapshot (not updatable)
`dbOpenForwardOnly`	Not defined	Similar to a Snapshot, but you can move only forward through the `Recordset` (default for ODBCDirect `Recordset` objects).
`dbOpenDynamic`	Not defined	Dynamic cursor (ODBCDirect `Recordset` objects only)

Table 29.2 Values for the *intOptions* Argument of the *OpenRecordset* Method Using DAO 3.5 and Access 2.0 Intrinsic Constants

DAO 3.5	Access 2.0	Purpose of Option
`dbDenyWrite`	`DB_DENYWRITE`	Prevents others from making changes to any records in the underlying table(s) while the `Recordset` is open.
`dbDenyRead`	`DB_DENYREAD`	Prevents others from reading any records in the underlying table while the `Recordset` is open. This option, which applies to `Table`-type `Recordset` objects only, should be used for administrative purposes only in a multiuser environment.

(continues)

Oracle Ex. 1009, p. 1045
Oracle v. DeBellis
IPR2024-00221

Table 29.2 Continued

DAO 3.5	Access 2.0	Purpose of Option
dbReadOnly	DB_READONLY	Does not allow updates to records in the table. Read-only access increases the speed of some operations.
dbAppendOnly	DB_APPENDONLY	Allows appending only new records. (Applies to Dynaset-type Recordset objects only.)
dbInconsistent	DB_INCONSISTENT	You can update the one side of a one-to-many relationship. (Applies to Dynaset-type Recordset objects only.)
dbConsistent	DB_CONSISTENT	You cannot update the one side of a one-to-many relationship, the default. (Applies to Dynaset-type Recordset objects only.)
dbForwardOnly	DB_FORWARDSCROLL	Creates a Recordset object of the forward-scrolling-only Snapshot type.
dbSQLPassThrough	Not defined	Specifies a SQL pass-through query against an ODBC database using Jet; returns a Snapshot-type Recordset.
dbSeeChanges	Not defined	Generates a runtime error when the user attempts to change data that another user is currently editing (Dynaset-type Recordset only).
dbRunAsync	Not defined	Executes the query asynchronously, allowing the query to be canceled during execution (ODBCDirect only).
dbExecDirect	Not defined	Runs a query with no parameters directly, instead of creating a prepared statement (ODBCDirect only).

Table 29.3 Values for the *intLockEdits* Argument of the *OpenRecordset* Method of DAO 3.5

DAO 3.5	Purpose of Option
dbReadOnly	Does not allow updates to the Recordset.
dbPessimistic	Uses pessimistic locking (locks the edited page during editing).
dbOptimistic	Uses optimistic locking (locks the edited page only when applying the edits).
dbOptimisticValue	Uses optimistic locking based on row values (ODBCDirect only).
dbOptimisticBatch	Uses optimistic locking for batch updates (ODBCDirect only).

Note

There are many limitations on the use of constant values for the *intOptions* and *intLockEdits* arguments of the OpenRecordset method. For instance, you receive an error if you use the dbReadOnly constant in both of these arguments. The "Remarks" section of the help topic for the OpenRecordset method explains the conflicts in detail.

Oracle Ex. 1009, p. 1046
Oracle v. DeBellis
IPR2024-00221

The following two sections show how to create two different types of Recordset objects.

A *Recordset* Object that Represents the Image of a Table. You can create the following two types of Recordset objects over a table, both of which create a virtual table in memory:

- A Recordset object of the Dynaset type created from the current Database object. To open a Dynaset-type Recordset object over the Orders table, add the **Private** rstCurrent **As** Recordset variable declaration statement to the Declarations section of the module. Then use the following statement to assign the pointer, assuming that you have previously executed the **Set** dbCurrent = CurrentDB() statement:

```
Set rsdCurrent = dbCurrent.OpenRecordset("Orders", dbOpenDynaset)
```

- A Recordset object of the Table type created from a TableDef object. To open a Table Recordset object over the Orders table, add the **Private** tdfCurrent **As** TableDef variable declaration statement to the Declarations section of the module. Then use the following statements to assign the pointers:

```
Set tdfCurrent = dbCurrent.TableDefs("Orders")
Set rstCurrent = tdfCurrent.OpenRecordset(dbOpenTable)
```

Figure 29.10 illustrates the creation of both of the preceding types of Recordset objects in the Debug window.

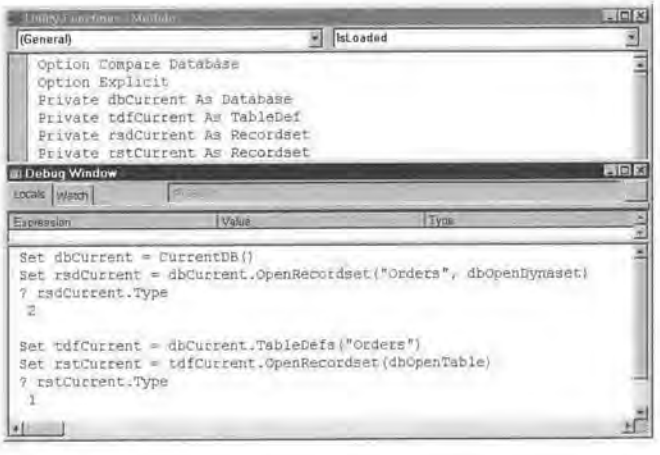


FIG. 29.10 Creating Recordset objects of the Dynaset and Table type in the Debug window.

Moving to a Specific Record in a *Recordset* Object. Opening a Recordset of the Table type lets you quickly find a record in a table with the Seek method. This method is applicable only to Table-type Recordset objects whose underlying table contains an index on the field in which the value you want to find is located. To apply the Seek method, you first must specify the index name. The following example sets the record pointer to the record in the Orders table of Northwind.mdb with a value of 10833 in the OrderID field, the primary key of the table:

Oracle Ex. 1009, p. 1047
Oracle v. DeBellis
IPR2024-00221

```
Set dbCurrent = CurrentDB()
Set tdfCurrent = dbCurrent.TableDefs("Orders")
Set rstCurrent = tdfCurrent.OpenRecordset()
rstCurrent.Index = "PrimaryKey"
rstCurrent.Seek "=", 10833
If rstCurrent.NoMatch Then
   'Record not found, display message box
Else
   'Record found, add code to process the record here
End If
```

Note

You can't enter conditional statements in the Debug window, but you can test the preceding code by typing the entries up to and including the Seek line and then executing **? rstCurrent.NoMatch** to determine whether the Seek method was successful in finding a matching record. The NoMatch property uses the **Boolean** data type. If False is returned, the record pointer rests on the matching record.

If the field in which you are Seeking the value is not the primary key field, you specify the field's name. The NoMatch property returns False if the value specified by the Seek expression is found; it returns True if a matching record is not found. Figure 29.11 shows typical entries to test the Seek method in the Debug window. Test the Indexes collection of the TableDef object to verify that the index you specify exists. The Seek method allows the use of a variety of other operators, such as < or >=, in place of the = operator.

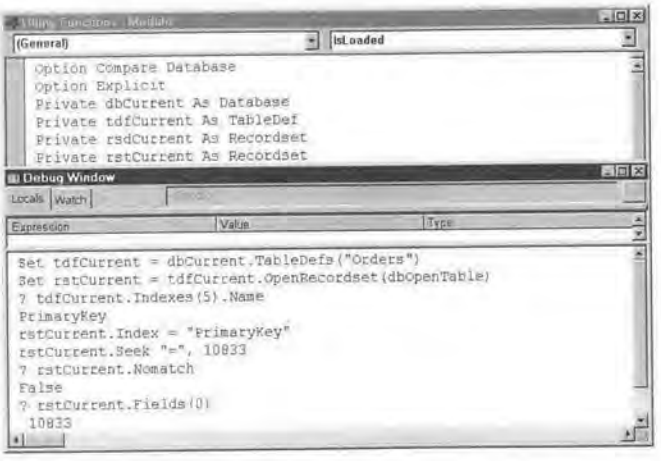


FIG. 29.11 Applying the Seek method to Recordset objects of the Table type.

Tip

You cannot apply the Find... methods to a forward-scrolling-only Recordset object of the Snapshot type. The only method applicable to such objects is MoveNext.

Oracle Ex. 1009, p. 1048
Oracle v. DeBellis
IPR2024-00221

You use the `Find...` methods to locate a specific record in `Recordset` objects of the `Dynaset` and `Snapshot` type. There are four `Find...` methods: `FindFirst`, `FindNext`, `FindLast`, and `FindPrevious`. Not surprisingly, each method does what its name indicates. The general syntax of the `Find...` methods is

```
rsdName.Find{First|Next|Last|Previous} strCriteria
```

The single `strCriteria` argument of the `Find...` methods must be a valid SQL `WHERE` clause without the `WHERE` reserved word. As with the `Seek` method, the `NoMatch` property returns `False` if a match occurs. Figure 29.12 shows examples of the use of the four `Find...` methods.

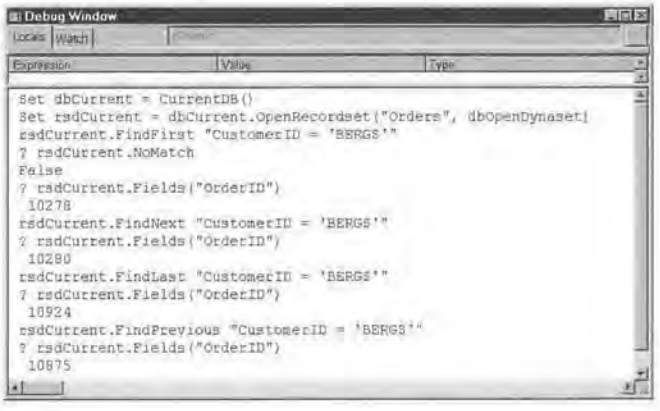


FIG. 29.12 Applying the `Find` methods to `Recordset` objects of the `Dynaset` or `Snapshot` type.

A *Recordset* that Emulates a *QueryDef* Object. You can substitute a valid Jet SQL statement for a table or query name as the value of the `strSource` property to create a `Recordset` object of the `Dynaset` or `Snapshot` type. You only can create such a `Recordset` over the `Database` object because more than one `TableDef` object may participate in the query. Such a `Recordset` is the equivalent of an "impersistent `QueryDef`" object (a `QueryDef` object that is not a `Document` object). Figure 29.13 shows you how to create a `Recordset` object using a SQL statement in the Debug window. You must add a **`Private`** `strSQL` **`As String`** statement to the Declarations section of the module if the **`Option`** `Explicit` statement has been executed. The entire `strSQL = _` statement is

```
strSQL = "SELECT * FROM Orders, [Order Details]
WHERE [Order Details].OrderID = Orders.OrderID
AND Orders.CustomerID = 'BERGS'"
```

If you need to manipulate a `Recordset`'s records only temporarily, creating the "impersistent `QueryDef`" `Recordset` is faster than creating a legitimate, persistent `QueryDef` object.

Oracle Ex. 1009, p. 1049
Oracle v. DeBellis
IPR2024-00221