HOUR 9

Creating Relationships

NEW TERM A *relationship* exists between two tables when one or more key fields from one table are matched to one or more key fields in another table. The fields in both tables usually have the same name, data type, and size. Relationships are a necessary by-product of the data normalization process. *Data normalization*, introduced in Hour 7, "Designing Databases," is the process of eliminating duplicate information from a system by splitting information into several tables, each containing a unique value (that is, a primary key). Although data normalization brings many benefits, it means you need to relate an application's tables to each other so that users can view the data in the system as a single entity. After you define relationships between tables, you can build queries, forms, reports, and data access pages that combine information from multiple tables. In this way, you can reap all the benefits of data normalization while ensuring that a system provides users with all the information they need. Referential integrity is another very important topic covered in this hour. *Referential integrity* consists of a series of rules that the Jet Engine applies to ensure that Jet properly maintains relationships between tables. In this hour you'll learn about the following:

- Relational database design principles
- The types of relationships available
- Establishing relationships
- Establishing referential integrity

Introduction to Relational Database Design

Many people believe Access is such a simple product to use that database design is something they don't need to worry about. I couldn't disagree more! Just as a poorly planned vacation will generally not be very much fun, a database with poorly designed tables and relationships will fail to meet the needs of its users.

The History of Relational Database Design

NEW TERM Dr. E. F. Codd first introduced formal relational database design in 1969 while he was at IBM. *Relational theory*, which is based on set theory and predicate logic, applies to both databases and database applications. Codd developed 12 rules that determine how well an application and its data adhere to the relational model. Since Codd first conceived these 12 rules, the number of rules has expanded into the hundreds.

You should be happy to learn that, although Microsoft Access is not a perfect application development environment, it measures up quite well as a relational database system.

Goals of Relational Database Design

The number-one goal of relational database design is to, as closely as possible, develop a database that models some real-world system. This involves breaking the real-world system into tables and fields and determining how the tables relate to each other. Although on the surface this might appear to be a trivial task, it can be an extremely cumbersome process to translate a real-world system into tables and fields.

A properly designed database has many benefits. The processes of adding, editing, deleting, and retrieving table data are greatly facilitated in a properly designed database. In addition, reports are easy to build. Most importantly, the database is easy to modify and maintain.

Rules of Relational Database Design

To adhere to the relational model, you must follow certain rules. These rules determine what you store in a table and how you relate the tables. These are the rules:

- The rules of tables
- The rules of uniqueness and keys
- The rules of foreign keys and domains

The Rules of Tables

New Term Each table in a system must store data about a single entity. An *entity* usually represents a real-life object or event. Examples of objects are customers, employees, and inventory items. Examples of events include orders, appointments, and doctor visits.

The Rules of Uniqueness and Keys

New Term Tables are composed of rows and columns. To adhere to the relational model, each table must contain a unique identifier. Without a unique identifier, it is programmatically impossible to uniquely address a row. You guarantee uniqueness in a table by designating a *primary key*, which is a single column or a set of columns that uniquely identifies a row in a table.

New Term Each column or set of columns in a table that contains unique values is considered a *candidate key*. One candidate key becomes the *primary key*. The remaining candidate keys become *alternate keys*. A primary key made up of one column is considered a *simple key*. A primary key composed of multiple columns is considered a *composite key*.

It is generally a good idea to choose a primary key that is

- Minimal (has as few columns as possible)
- Stable (rarely changes)
- Simple (is familiar to the user)

Following these rules greatly improves the performance and maintainability of a database application, particularly if it deals with large volumes of data.

New Term Consider the example of an employee table. An employee table is generally composed of employee-related fields such as Social Security number, first name, last name, hire date, salary, and so on. The combination of the first name and the last name fields could be considered a primary key. This might work until the company hires two employees who have the same name. Although the first and last names could be combined with additional fields (for example, hire date) to constitute uniqueness, that would violate the rule of keeping the primary key minimal. Furthermore, an employee might get married, and her last name might change. This violates the rule of keeping a primary key stable. Therefore, using a name as the primary key violates the principle of stability. The Social Security number might be a valid choice for primary key, but a foreign employee might not have a Social Security number. This is a case in which a derived, rather than a natural, primary key is appropriate. A *derived key* is an artificial key that you create. A *natural key* is one that is already part of the database.

In examples such as this, I suggest adding `EmployeeID` as an `AutoNumber` field. Although the field would violate the rule of simplicity (because an employee number is meaningless to the user), it is both small and stable. Because it is numeric, it is also efficient to process. In fact, I use `AutoNumber` fields as primary keys for most of the tables that I build.

The Rules of Foreign Keys and Domains

A *foreign key* in one table is the field that relates to the primary key in a second table. For example, the `CustomerID` field may be the primary key in a `Customers` table and the foreign key in an `Orders` table.

NEW TERM A *domain* is a pool of values from which columns are drawn. A simple example of a domain is the specific data range of employee hire dates. In the case of the `Orders` table, the domain of the `CustomerID` column is the range of values for the `CustomerID` in the `Customers` table.

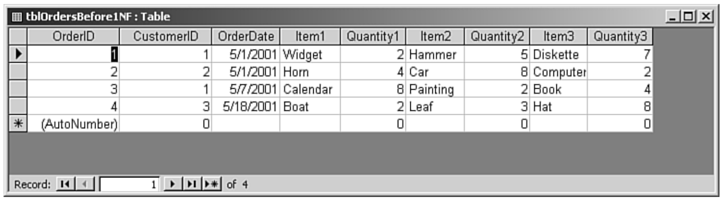
Normalization and Normal Forms

NEW TERM Some of the most difficult decisions that you face as a developer are what tables to create and what fields to place in each table, as well as how to relate the tables that you create. As you learned in Hour 7, *Normalization* is the process of applying a series of rules to ensure that a database achieves optimal structure. *Normal forms* are a progression of these rules. Each successive normal form achieves a better database design than the previous form. Although there are several levels of normal forms, it is generally sufficient to apply only the first three levels of normal forms. The following sections describe the first three levels of normal forms.

First Normal Form

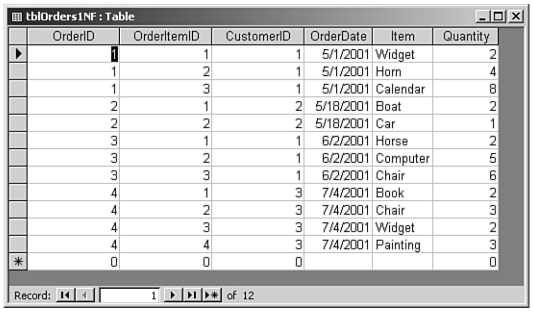
NEW TERM To achieve first normal form, all columns in a table must be *atomic*. This means, for example, that you cannot store first name and last name in the same field. The reason for this rule is that data becomes very difficult to manipulate and retrieve if you store multiple values in a single field. Let's use the full name as an example. It would be impossible to sort by first name or last name independently if you stored both values in the same field. Furthermore, you or the user would have to perform extra work to extract just the first name or just the last name from the field.

Another requirement for first normal form is that the table must not contain repeating values. An example of repeating values is a scenario in which `Item1`, `Quantity1`, `Item2`, `Quantity2`, `Item3`, and `Quantity3` fields are all found within the `Orders` table (see Figure 9.1). This design introduces several problems. What if the user wants to add a fourth item to the order? Furthermore, finding the total ordered for a product requires searching several columns. In fact, all numeric and statistical calculations on the table are extremely cumbersome. Repeating groups make it difficult to summarize and manipulate table data. The alternative, shown in Figure 9.2, achieves first normal form. Notice that each item ordered is located in a separate row. All fields are atomic, and the table contains no repeating groups.



OrderID	CustomerID	OrderDate	Item1	Quantity1	Item2	Quantity2	Item3	Quantity3
1	1	5/1/2001	Widget	2	Hammer	5	Diskette	7
2	2	5/1/2001	Horn	4	Car	8	Computer	2
3	1	5/7/2001	Calendar	8	Painting	2	Book	4
4	3	5/18/2001	Boat	2	Leaf	3	Hat	8
(AutoNumber)	0			0		0		0

FIGURE 9.1
A table that contains repeating groups.



OrderID	OrderItemID	CustomerID	OrderDate	Item	Quantity
1	1	1	5/1/2001	Widget	2
1	2	1	5/1/2001	Horn	4
1	3	1	5/1/2001	Calendar	8
2	1	2	5/18/2001	Boat	2
2	2	2	5/18/2001	Car	1
3	1	1	6/2/2001	Horse	2
3	2	1	6/2/2001	Computer	5
3	3	1	6/2/2001	Chair	6
4	1	3	7/4/2001	Book	2
4	2	3	7/4/2001	Chair	3
4	3	3	7/4/2001	Widget	2
4	4	3	7/4/2001	Painting	3
0	0	0			0

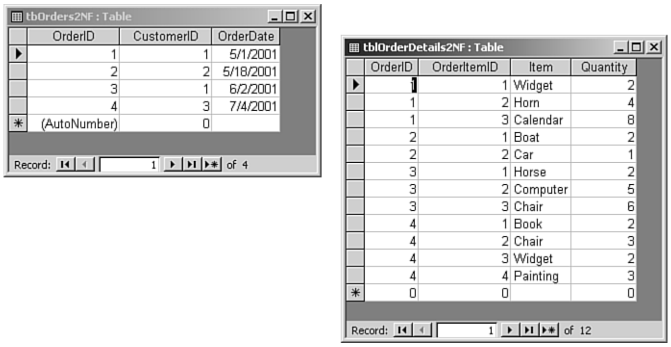
FIGURE 9.2
A table that achieves first normal form.

Second Normal Form

NEW TERM For a table to achieve second normal form, all non-key columns must be fully dependent on the primary key. In other words, each table must store data about only one subject. For example, the table shown in Figure 9.2 includes information about the order (`OrderID`, `CustomerID`, and `OrderDate`) and information about the items the customer is ordering (`Item` and `Quantity`). To achieve second normal form, you must break this data into two tables—an order table and an order detail table. The process of breaking the data into two tables is called *decomposition*. Decomposition is considered to be *non-loss* decomposition because no data is lost during the decomposition process. After you separate the data into two tables, you can easily bring the data back together by joining the two tables via a query. Figure 9.3 shows the data separated into two tables. These two tables achieve second normal form because the fields in each table pertain to the primary key of the table.

Third Normal Form

To attain third normal form, a table must meet all the requirements for first and second normal forms, and all non-key columns must be mutually independent. This means that you must eliminate any calculations, and you must break out the data into lookup tables. Lookup tables include tables such as inventory tables, course tables, state tables, and any other table where you look up a set of values from which you select the entry that you store in the foreign key field. For example, from the `Customer` table, you look up within the set of states in the state table to select the state associated with the customer.



tbOrders2NF : Table

OrderID	CustomerID	OrderDate
1	1	5/1/2001
2	2	5/18/2001
3	1	6/2/2001
4	3	7/4/2001
(AutoNumber)	0	

Record: 1 of 4

tblOrderDetails2NF : Table

OrderID	OrderItemID	Item	Quantity
1	1	Widget	2
1	2	Horn	4
1	3	Calendar	8
2	1	Boat	2
2	2	Car	1
3	1	Horse	2
3	2	Computer	5
3	3	Chair	6
4	1	Book	2
4	2	Chair	3
4	3	Widget	2
4	4	Painting	3
0	0		0

Record: 1 of 12

FIGURE 9.3
Tables that achieve second normal form.

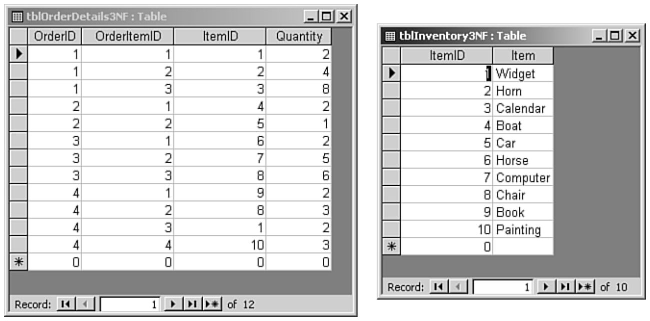
An example of a calculation stored in a table is the product of price multiplied by quantity. Instead of storing the result of this calculation in the table, you would generate the calculation in a query or in the control source of a control on a form or a report.

The example in Figure 9.3 does not achieve third normal form because the description of the inventory items is stored in the `Order Details` table. If the description changes, all rows with that inventory item need to be modified. The `Order Details` table, shown in Figure 9.4, shows the item descriptions broken into an `Inventory` table. This design achieves third normal form. We have moved the description of the inventory items to the `Inventory` table, and `ItemID` is stored in the `Order Details` table. All fields are mutually independent. You can modify the description of an inventory item in one place.

Denormalization: Purposely Violating the Rules

NEW TERM Although a developer's goal is normalization, there are many times when it makes sense to deviate from normal forms. This process is called *denormalization*. The primary reason for applying denormalization is to enhance performance.

An example of when denormalization might be the preferred tact could involve an open invoices table and a summarized accounting table. It might be impractical to calculate summarized accounting information for a customer when you need it. Instead, you can maintain the summary calculations in a summarized accounting table so that you can easily retrieve them as needed. Although the upside of this scenario is improved performance, the downside is that you must update the summary table whenever you make changes to the open invoices. This imposes a definite trade-off between performance and maintainability. You must decide whether the trade-off is worth it.



	OrderID	OrderItemID	ItemID	Quantity
▶	1	1	1	2
	1	2	2	4
	1	3	3	8
	2	1	4	2
	2	2	5	1
	3	1	6	2
	3	2	7	5
	3	3	8	6
	4	1	9	2
	4	2	8	3
	4	3	1	2
	4	4	10	3
*	0	0	0	0

Record: 1 of 12

	ItemID	Item
▶	1	Widget
	2	Horn
	3	Calendar
	4	Boat
	5	Car
	6	Horse
	7	Computer
	8	Chair
	9	Book
	10	Painting
*	0	

Record: 1 of 10

FIGURE 9.4
A table (on the right) that achieves third normal form.

If you decide to denormalize, you should document your decision. You should make sure that you make the necessary application adjustments to ensure that you properly maintain the denormalized fields. Finally, you need to test to ensure that the denormalization process actually improves performance.

Integrity Rules

Although integrity rules are not part of normal forms, they are definitely part of the database design process. Integrity rules are broken into two categories: overall integrity rules and database-specific integrity rules.

Overall Integrity Rules

NEW TERM The two types of overall integrity rules are referential integrity rules and entity integrity rules. *Referential integrity rules* dictate that a database does not contain any orphan foreign key values. This means that

- Child rows cannot be added for parent rows that do not exist. In other words, an order cannot be added for a nonexistent customer.
- A primary key value cannot be modified if the value is used as a foreign key in a child table. This means that a `CustomerID` in the `Customers` table cannot be changed if the `Orders` table contains rows with that `CustomerID`.
- A parent row cannot be deleted if child rows have that foreign key value. For example, a customer cannot be deleted if the customer has orders in the `Orders` table.

Entity integrity dictates that the primary key value cannot be `Null`. This rule applies not only to single-column primary keys, but also to multicolumn primary keys. In fact, in a multicolumn primary key, no field in the primary key can be `Null`. This makes sense because if any part of the primary key can be `Null`, the primary key can no longer act as a unique identifier for the row. Fortunately, the Jet Engine does not allow a field in a primary key to be `Null`.

Database-Specific Integrity Rules

Database-specific integrity rules are not applicable to all databases, but are, instead, dictated by business rules that apply to a specific application. Database-specific rules are as important as overall integrity rules. They ensure that the user enters only valid data into a database. An example of a database-specific integrity rule is requiring the delivery date for an order to fall after the order date.

The Types of Relationships

Three types of relationships can exist between tables in a database: one-to-many, one-to-one, and many-to-many. Setting up the proper type of relationship between two tables in a database is imperative. The right type of relationship between two tables ensures

- Data integrity
- Optimal performance
- Ease of use in designing system objects

The reasons behind these benefits are covered throughout this hour. Before you can understand the benefits of relationships, though, you must understand the types of relationships available.

One-to-Many Relationships

NEW TERM A one-to-many relationship is by far the most common type of relationship. In a *one-to-many relationship*, a record in one table can have many related records in another table. A common example is a relationship set up between a `Customers` table and an `Orders` table. For each customer in the `Customers` table, you want to have more than one order in the `Orders` table. On the other hand, each order in the `Orders` table can belong to only one customer. The `Customers` table is on the "one" side of the relationship, and the `Orders` table is on the "many" side. In order for you to implement this relationship, the field joining the two tables on the "one" side of the relationship must be unique.

In the `Customers` and `Orders` tables example, the `CustomerID` field that joins the two tables must be unique within the `Customers` table. If more than one customer in the `Customers` table has the same customer ID, it is not clear which customer belongs to an order in the `Orders` table. For this reason, the field that joins the two tables on the "one" side of the one-to-many relationship must be a primary key or have a unique index. In almost all cases, the field relating the two tables is the primary key of the table on the "one" side of the relationship. The field relating the two tables on the "many" side of the relationship is the foreign key.

One-to-One Relationships

New Term In a *one-to-one relationship*, each record in the table on the "one" side of the relationship can have only one matching record in the table on the "many" side of the relationship. This relationship is not common and is used only in special circumstances. Usually, if you have set up a one-to-one relationship, you should have combined the fields from both tables into one table. The following are the most common reasons to create a one-to-one relationship:

- The number of fields required for a table exceeds the number of fields allowed in an Access table.
- Certain fields that are included in a table need to be much more secure than other fields in the same table.
- Several fields in a table are required for only a subset of records in the table.

The maximum number of fields allowed in an Access table is 255. There are very few reasons a table should ever have more than 255 fields. In fact, before you even get close to 255 fields, you should take a close look at the design of the system. On the rare occasion when having more than 255 fields is appropriate, you can simulate a single table by moving some of the fields to a second table and creating a one-to-one relationship between the two tables.

New Term The second reason to separate into two tables data that logically would belong in the same table involves security. An example is a table that contains employee information. Certain information, such as employee name, address, city, state, zip code, home phone, and office extension, might need to be accessed by many users of the system. Other fields, including the hire date, salary, birth date, and salary level, might be highly confidential. Field-level security is not available in Access. You can simulate field-level security by using a special attribute of queries called *Run with Owner's Permissions*. The alternative to this method is to place the fields that all users can access in one table and the highly confidential fields in another. You give only a special Admin user (that is, a user with special security privileges—not one actually named Admin) access to the table that contains the confidential fields.

The last situation in which you would want to define one-to-one relationships is when you will use certain fields in a table for only a relatively small subset of records. An example is an `Employees` table and a `Vesting` table. Certain fields are required only for employees who are vested. If only a small percentage of a company's employees are vested, it is not efficient, in terms of performance or disk space, to place all the fields containing information about vesting in the `Employees` table. This is especially true if the vesting information requires a large number of fields. By breaking the information into two tables and creating a one-to-one relationship between the tables, you can reduce disk-space requirements and improve performance. This improvement is particularly pronounced if the `Employees` table is large.

Many-to-Many Relationships

NEW TERM In a *many-to-many relationship*, records in two tables have matching records. You cannot directly define a many-to-many relationship in Access; you must develop this type of relationship by adding a table called a *junction table*. You relate the junction table to each of the two tables in one-to-many relationships. For example, with an `Orders` table and a `Products` table, each order will probably contain multiple products, and each product is likely to be found on many different orders. The solution is to create a third table, called `OrderDetails`. You relate the `OrderDetails` table to the `Orders` table in a one-to-many relationship based on the `OrderID` field. You relate it to the `Products` table in a one-to-many relationship based on the `ProductID` field.

Establishing Relationships in Access

You use the Relationships window to establish relationships between Access tables, as shown in Figure 9.5. To open the Relationships window, you click Relationships on the toolbar with the Database window active or you choose Tools|Relationships. If you have not established any relationships, the Show Table dialog box appears. The Show Table dialog box allows you to add tables to the Relationships window.

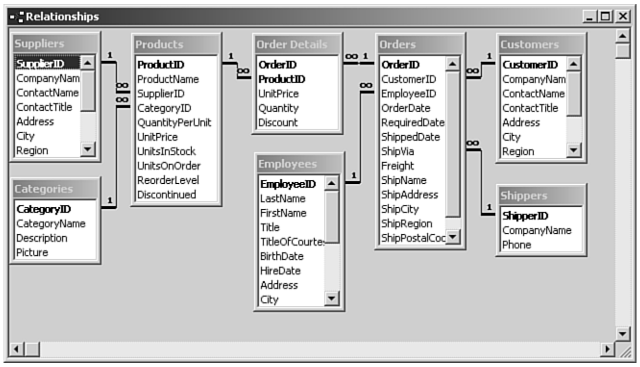


FIGURE 9.5
The Relationships window, which enables you to view, add, modify, and remove relationships between tables.

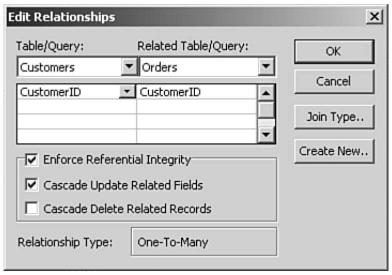
By looking at the Relationships window, you can see the types of relationships for each table. All the one-to-many and one-to-one relationships defined in a database are represented with join lines. If you enforce referential integrity between the tables involved in a one-to-many relationship, the join line between the tables appears with the number *1* on the "one" side of the relationship and with a link symbol (∞) on the "many" side of the relationship. A one-to-one relationship appears with a *1* on each end of the join line.

Establishing a Relationship Between Two Tables

To establish a relationship between two tables, you follow these steps:

1. Open the Relationships window.
2. If this is the first time that you've opened the Relationships window of a particular database, the Show Table dialog box appears. Select each table you want to relate and click Add.
3. If you have already established relationships in the current database, the Relationships window appears. If the tables you want to include in the relationship do not appear, click the Show Table button on the toolbar or choose Relationships| Show Table. To add the desired tables to the Relationships window, select a table and then click Add. Repeat this process for each table you want to add. To select multiple tables at once, press Shift while clicking to select contiguous tables or press Ctrl while clicking to select noncontiguous tables; then click Add. Click Close when you are finished.
4. Click and drag the field from one table to the matching field in the other table. The Edit Relationships dialog box appears.
5. Determine whether you want to establish referential integrity and whether you want to cascade update related fields or cascade delete related records by enabling the appropriate check boxes (see Figure 9.6). These topics are covered later in this hour, in the section "Establishing Referential Integrity."

FIGURE 9.6
The Edit Relationships dialog box, which enables you to view and modify the relationships between the tables in a database.


Edit Relationships
Table/Query: Customers
Related Table/Query: Orders
CustomerID | CustomerID
[✓] Enforce Referential Integrity
[✓] Cascade Update Related Fields
[ ] Cascade Delete Related Records
Relationship Type: One-To-Many

6. Click OK. The dialog box closes, and you return to the Relationships window.

Following Guidelines for Establishing Relationships

You must remember a few important things when establishing relationships. If you are not aware of these important gotchas, you could find yourself in some pretty hairy situations:

- It is important to understand the correlation between the Relationships window and the actual relationships established within a database. The Relationships window lets you view and modify the existing relationships. When you establish relationships, Access creates the relationship the moment you click OK. You can delete the tables from the Relationships window (by selecting them and pressing Delete), but the relationships still exist. (The "Modifying an Existing Relationship" section of this hour covers the process of permanently removing relationships.) The Relationships window provides a visual blueprint of the relationships that are established. If you modify the layout of the window by moving around tables, adding tables to the window, or removing tables from the window, Access prompts you to save the layout after you close the Relationships window. Access is not asking whether you want to save the relationships you have established; it is simply asking whether you want to save the visual layout of the window.
- When you're adding tables to the Relationships window by using the Show Tables dialog box, it is easy to accidentally add a table to the window many times. This is because the tables you are adding can hide behind the Show Tables dialog box, or they can appear below the portion of the Relationships window that you are viewing. If this occurs, you see multiple occurrences of the same table when you close the Show Tables dialog box. Access gives each occurrence of the table a different alias, and you must remove the extra occurrences.
- You can add queries to the Relationships window by using the Show Tables dialog box. Although this method is rarely used, it might be useful if you regularly include the same queries within other queries and want to permanently establish relationships between them.
- If you remove tables from the Relationships window (remember that this does not delete the relationships) and you want to once again show all relationships that exist in the database, you can click Show All Relationships on the toolbar or choose Relationships|Show All. All existing relationships are then shown.
- To delete a relationship, you can click the join line and press Delete.

Modifying an Existing Relationship

Modifying an existing relationship is easy. Access gives you the capability to delete an existing relationship or to simply modify the nature of the relationship.

To permanently remove a relationship between two tables, you follow these steps:

1. With the Database window active, click Relationships on the toolbar.
2. Click the line joining the two tables whose relationship you want to delete.
3. Press Delete. Access prompts you to verify your actions. Click Yes.

You often need to modify the nature of a relationship rather than remove it. To modify a relationship, you follow these steps:

1. With the Database window active, click Relationships on the toolbar.
2. Double-click the line joining the two tables whose relationship you want to modify.
3. Make the required changes.
4. Click OK. All the normal rules regarding the establishment of relationships apply.

Task: Establishing Relationships

▼ To Do

Relationships are an extremely important aspect of any database that you build. Let's practice the process of creating a brand-new database. We'll add tables and then establish relationships between them. To begin, you create a new database and add a table called `tblCustomers`, another called `tblOrders`, and a third called `tblOrderDetails`. The tables should have the following fields:

```
tblCustomers: CustomerID, CompanyName, Address, City, State, ZipCode

tblOrders: OrderID, CustomerID, OrderDate, ShipVIA

tblOrderDetails: OrderID, LineNumber, ItemID, Quantity, Price
```

After you've built the necessary tables, you're ready to establish the relationships between them. First you need to set some important properties of the fields that you just added. Then you'll be ready to establish the actual relationships between the tables. Here's how this works:

1. In the `tblCustomers` table, make the `CustomerID` field a `Text` field. Designate the `CustomerID` field as the primary key. Set the size of the field to `5`. Make all other fields `Text` fields and leave their default property values.
2. In the `tblOrders` table, set `OrderID` to the `AutoNumber` field type. Make `OrderID` the primary key field. Make the `CustomerID` field a `Text` field with the field size of `5`. Set the field type of the `OrderDate` field to `Date` and the field type of the `ShipVIA` field to `Number`, with a size of `Long Integer`.
3. In the `tblOrderDetails` table, set the field type of the `OrderID` field to `Number` and make sure that the size is `Long Integer`. Set the field type of the `LineNumber` field to

▼

▼ Number, with a size of Long Integer. Base the primary key of the table on the combination of the OrderID and LineNumber fields. Set the field type of the ItemID and Quantity fields to Number, with a size of Long Integer, and set the field type of the Price field to Currency.

4. Open the Relationships window. With the tblCustomers table in the Show Table dialog box selected, hold down the Shift key and click to select the tblOrders table. Click Add. All three tables appear in the Relationships window. Click Close. Click and drag from the CustomerID field in the tblCustomers table to the CustomerID field in the tblOrders table. After the Edit Relationships dialog box appears, click OK. Repeat the process, clicking and dragging the OrderID field from the tblOrders table to the OrderID field in the tblOrderDetails table. ▲

Establishing Referential Integrity

As you can see, establishing a relationship is quite easy. Establishing the right kind of relationship is a little more difficult. When you attempt to establish a relationship between two tables, Access makes some decisions based on a few predefined factors:

- Access establishes a one-to-many relationship if one of the related fields is a primary key or has a unique index.
- Access establishes a one-to-one relationship if both of the related fields are primary keys or have unique indexes.
- Access creates an indeterminate relationship if neither of the related fields is a primary key and neither has a unique index. You cannot establish referential integrity in this case.

As discussed earlier in this hour, *referential integrity* consists of a series of rules that the Jet Engine applies to ensure that Jet properly maintains the relationships between tables. At the most basic level, referential integrity rules prevent the creation of orphan records in the table on the "many" side of the one-to-many relationship. After you establish a relationship between a Customers table and an Orders table, for example, all orders in the Orders table must be related to a particular customer in the Customers table. Before you can establish referential integrity between two tables, the following conditions must be met:

- The matching field on the "one" side of the relationship must be a primary key field or must have a unique index.
- The matching fields must have the same data types. (For linking purposes, AutoNumber fields match Long Integer fields.) With the exception of Text fields, the

matching fields also must have the same size. `Number` fields on both sides of the relationship must have the same size (for example, `Long Integer`).

- Both tables must be part of the same Access database.
- Both tables must be stored in the proprietary Access file (`.MDB`) format. (They cannot be external tables from other sources.)
- The database that contains the two tables must be open.
- Existing data within the two tables cannot violate any referential integrity rules. All orders in the `Orders` table must relate to existing customers in the `Customers` table, for example.

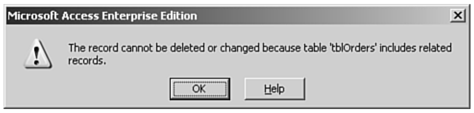
> Although `Text` fields involved in a relationship do not have to be the same size, it is prudent to make them the same size. Otherwise, you degrade performance as well as risk the chance of unpredictable results when you create queries based on the two tables.

After you establish referential integrity between two tables, the Jet Engine applies the following rules:

- You cannot enter in the foreign key of the related table a value that does not exist in the primary key of the primary table. For example, you cannot enter in the `CustomerID` field of the `Orders` table a value that does not exist in the `CustomerID` field of the `Customers` table.
- You cannot delete a record from the primary table if corresponding records exist in the related table. For example, you cannot delete a customer from the `Customers` table if related records (for example, records with the same value in the `CustomerID` field) exist in the `Orders` table.
- You cannot change the value of a primary key on the "one" side of a relationship if corresponding records exist in the related table. For example, you cannot change the value in the `CustomerID` field of the `Customers` table if corresponding orders exist in the `Orders` table.

If you attempt to violate any of these three rules and you have enforced referential integrity between the tables, Access displays an appropriate error message, as shown in Figure 9.7.

The Jet Engine's default behavior is to prohibit the deletion of parent records that have associated child records and to prohibit the change of a primary key value of a parent record when that parent has associated child records. You can override these restrictions by using the Cascade Update Related Fields and Cascade Delete Related Records check boxes that are available in the Relationships dialog box when you establish or modify a relationship.


Microsoft Access Enterprise Edition

The record cannot be deleted or changed because table 'tblOrders' includes related records.

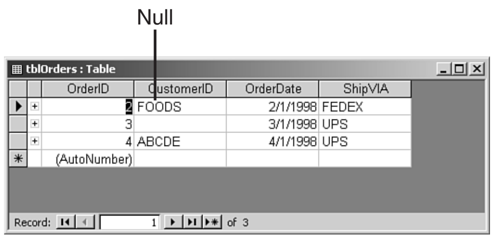
FIGURE 9.7
An error message that appears when you attempt to delete a customer who has orders.

The Cascade Update Related Fields Option

The Cascade Update Related Fields option is available only if you have established referential integrity between tables. When this option is selected, the user can change the primary key value of the record on the "one" side of the relationship. When the user attempts to modify the field joining the two tables on the "one" side of the relationship, the Jet Engine cascades the change down to the foreign key field on the "many" side of the relationship. This is useful if the primary key field is modifiable. For example, a purchase number on a purchase order master record might be updateable. If the user modifies the purchase order number of the parent record, you would want to cascade the change to the associated detail records in the purchase order detail table.

There is no need to select the Cascade Update Related Fields option when the related field on the "one" side of the relationship is an `AutoNumber` field. You can never modify an `AutoNumber` field. The Cascade Update Related Fields option has no effect on `AutoNumber` fields. In fact, this is why, in the preceding Task, you make the `CustomerID` a Text field. It provides an example that you can later use with a cascade update.

It is very easy to accidentally introduce a loophole into a system. If you create a one-to-many relationship between two tables but forget to set the `Required` property of the foreign key field to `Yes`, you allow the addition of orphan records. Figure 9.8 illustrates this point. In this example, I added an order to `tblOrders` without entering a customer ID. This record is an orphan record because no records in `tblCustomers` have `CustomerID` set to `Null`. To eliminate the problem, you set the `Required` property of the foreign key field to `Yes`.



tblOrders : Table

	OrderID	CustomerID	OrderDate	ShipVIA
+	2	FOODS	2/1/1998	FEDEX
+	3		3/1/1998	UPS
+	4	ABCDE	4/1/1998	UPS
*	(AutoNumber)			

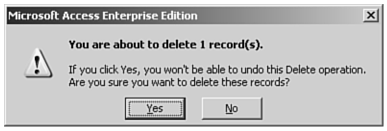
Record: 1 of 3

FIGURE 9.8
An orphan record with `Null` *in the foreign key field.*

The Cascade Delete Related Records Option

The Cascade Delete Related Records option is available only if you have established referential integrity between tables. When this option is selected, the user can delete a record on the "one" side of a one-to-many relationship, even if related records exist in the table on the "many" side of the relationship. A user can delete a customer even if the customer has existing orders, for example. The Jet Engine maintains referential integrity between the tables because it automatically deletes all related records in the child table.

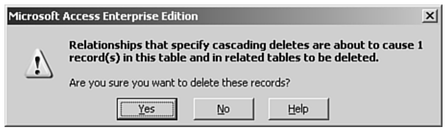
If you attempt to delete a record from the table on the "one" side of a one-to-many relationship and no related records exist in the table on the "many" side of the relationship, you get the usual warning message, as shown in Figure 9.9. On the other hand, if you attempt to delete a record from the table on the "one" side of a one-to-many relationship and related records exist in the child table, Access warns you that you are about to delete the record from the parent table as well as any related records in the child table (see Figure 9.10).


Microsoft Access Enterprise Edition

You are about to delete 1 record(s).

If you click Yes, you won't be able to undo this Delete operation.
Are you sure you want to delete these records?

FIGURE 9.9
A message that appears after the user attempts to delete a parent record that does not has related child records.


Microsoft Access Enterprise Edition

Relationships that specify cascading deletes are about to cause 1 record(s) in this table and in related tables to be deleted.

Are you sure you want to delete these records?

FIGURE 9.10
A message that appears after the user attempts to delete a parent record that has related child records.

The Cascade Delete Related Records option is not always appropriate. It is an excellent feature, but you should use it prudently. Although it is usually appropriate to cascade delete from an `Orders` table to an `Order Details` table, for example, it generally is not appropriate to cascade delete from a `Customers` table to an `Orders` table. This is because you generally do not want to delete all your order history from the `Orders` table if for some reason you want to delete a customer. Deleting the order history causes important information, such as the profit and loss history, to change. It is therefore appropriate to prohibit this type of deletion and handle the customer in some other way, such as marking him or her as inactive or archiving his or her data. On the other hand, if you delete an order because the customer cancelled it, you probably want to remove the corresponding order detail information as well. In this case, the Cascade Delete Related Records option is appropriate. You need to make the most prudent decision in each situation, based on business needs. You need to carefully consider the implications of each option before you make a decision.

Task: Working with Referential Integrity

▼ To Do

In this task you enforce referential integrity between the `tblCustomers` table and the `tblOrders` table you created earlier in this hour. It illustrates how enforcing referential integrity between the tables affects the process of adding and deleting records. You need to follow these steps:

1. Open the Relationships window. Double-click the join line between `tblCustomers` and `tblOrders`. Enable the Enforce Referential Integrity check box. Click OK. Repeat this process for the relationship between `tblOrders` and `tblOrderDetails`.
2. Go into `tblCustomer` and add a couple records. Take note of the customer IDs. Go into `tblOrders`. Add a couple records, taking care to assign customer IDs of customers that exist in the `tblCustomers` table. Now try to add an order for a customer whose customer ID does not exist in `tblCustomers`. You should get an error message.
3. Attempt to delete from `tblCustomers` a customer who does not have any orders. You should get a warning message, but Access should allow you to complete the process. Now try to delete a customer who does have orders. The Jet Engine should prohibit you from deleting the customer. Attempt to change the customer ID of a customer who has orders. You should not be able to do this.

▲

Task: Working with Cascade Update Related Fields and Cascade Delete Related Records

To Do

With the Cascade Update Related Fields feature enabled, you are able to update the primary key value of a record that has associated child records. With the Cascade Delete Related Records feature enabled, you can delete a parent record that has associated child records. To see how to use Cascade Update Related Fields and Cascade Delete Related Records, follow these steps:

1. Modify the relationship between the `tblCustomers` and `tblOrders` tables you created earlier in this hour. Open the Relationships window. Double-click the join line between `tblCustomers` and `tblOrders`. Enable the Cascade Update Related Fields check box. Modify the relationship between `tblOrders` and `tblOrderDetails`. Enable the Cascade Delete Related Records check box. There is no need to enable Cascade Update Related Fields because the `OrderID` field in `tblOrders` is an `AutoNumber` field.
2. Attempt to delete a customer who has orders. The Jet Engine should prohibit you from doing this because Cascade Delete Related Records is not enabled. In `tblCustomers`, change the customer ID of a customer who has orders. The Jet Engine should allow this change. Take a look at `tblOrders`. The Jet Engine should have updated the customer ID of all corresponding records in the table to reflect the change in the parent record.
3. Add some order details to `tblOrderDetails`. Try to delete any order that has details within `tblOrderDetails`. You should receive a warning, but the Jet Engine should allow you to complete the process.

The Benefits of Relationships

The primary benefit of relationships is the data integrity they provide. Without the establishment of relationships, users are free to add records to child tables without regard to entering required parent information. After you establish referential integrity, you can enable Cascade Update Related Fields or Cascade Delete Related Records, as appropriate, which saves you quite a bit of code in maintaining the integrity of the data in the system. Most relational database management systems require that you write the code to delete related records when the user deletes a parent record or to update the foreign key in related records when the user modifies the primary key of the parent. When you enable the Cascade Update Related Fields and Cascade Delete Related Fields check boxes, you are sheltered from having to write a single line of code to perform these tasks when they are appropriate.

Access automatically carries relationships into your queries. This means that each time you build a new query, Access automatically establishes the relationships between the tables within the query, based on the relationships that are set up in the Relationships window. Furthermore, each time you build a form or report, Access uses relationships between the tables included on the form or report to assist with the design process. Whether you delete or update data by using a datasheet or a form, all referential integrity rules automatically apply, even if you establish the relationship after you build the form.

Summary

Relationships enable you to normalize a database. By using relationships, you can divide data into separate tables and then once again combine the data at runtime. This hour begins by explaining relational database design principles. It describes the types of relationships that you can define. It also covers the details of establishing and modifying relationships between tables and describes all the important aspects of establishing relationships.

The capability to easily establish and maintain referential integrity between tables is an important strength of Microsoft Access. This hour describes the referential integrity options and highlights when each option is appropriate. Finally, this hour summarizes the benefits of relationships.

Q&A

Q Name three benefits of relationships.

A The right type of relationship ensures data integrity, optimal performance, and ease of use in designing system objects.

Q Explain the concept of a foreign key.

A A foreign key in one table is the field that relates to the primary key in a second table. For example, whereas `CustomerID` is the primary key in the `Customers` table, it may be the foreign key in the `Orders` table.

Q Explain referential integrity.

A With referential integrity, a database cannot contain any orphan foreign key values. This means that you cannot add child rows for parents that don't exist, you cannot modify the parent key value if that parent has children (unless the Cascade Update Related Fields option is selected), and you cannot delete parents that have children (unless the Cascade Delete Related Fields option is selected).

Q Explain the Cascade Update Related Fields option and the Cascade Delete Related Fields option.

A With the Cascade Update Related Fields option enabled, if the user tries to update the primary key of a parent record that has children, Access updates the foreign key of each of the child rows. With the Cascade Delete Related Fields option enabled, if the user tries to delete a parent record that has children, Access attempts to delete all the associated children rows.

Workshop

The Workshop includes quiz questions that are designed to help you test your understanding of the material covered and activities to help put what you've learned to practice. You can find the answers to the questions in the section immediately following the quiz.

Quiz

1. Name three types of relationships.
2. Name the types of relationship that you cannot directly create in Access.
3. Name the three attributes that constitute a good primary key.
4. Denormalization slows down performance (True/False).
5. The Relationships window always reflects the relationships that are in place in a database (True/False).

Quiz Answers

1. One-to-many, one-to-one, and many-to-many.
2. Many-to-many.
3. Minimal, stable, and simple.
4. False. The primary reason for denormalization is generally to improve performance.
5. False. The Relationships window provides a visual layout of the relationships that are included within the window. This may or may not represent all the relationships in the database.

Activities

Build three related tables. Don’t forget to set their primary keys. Establish relationships between them. Set referential integrity. Attempt to add, remove, and modify data. Practice using the Cascade Update Related Fields and Cascade Delete Related Fields options. See how this affects the process of inserting, updating, and deleting data.