

Lesson 7: Declaring relationships

7.1 Introduction: The advantage of “normalization”

A common mistake made by inexperienced database designers (or those who have more experience with spreadsheets than databases) is to ignore the recommendation to model the problem in terms of entities and relationships and to put all the information they need into a single, large table. [Figure 7.1](#) shows such a table containing information about customers, regions, and salespeople.

The advantage of the single-table approach is that it requires less thought during the initial stages of application development. The

disadvantages are too numerous to mention, but some of the most important can be illustrated using the small amount of data shown in [Figure 7.1](#):

1. **Insertion anomaly** — Assume that we want to create a new sales region called “On-line”. The sales rep for the new region would be responsible for any Internet-based selling initiatives. However, a new region cannot be added to the table unless a valid customer ID is specified. Since the On-line region does not have any customers at this point, the only way to add a new region is to introduce a “dummy” customer.

FIGURE 7.1: The “monolithic” approach to database design.

The diagram shows a table with columns: Customer name, RegionCode, Region, RepID, EMP_FNAME, and EMP_LNAME. The table contains six rows of data. Annotations with arrows point to specific parts of the table: one points to the 'Central' region rows, another to the 'Central' regionCode, and a third to the 'RepID' column.

	Customer name	RegionCode	Region	RepID	EMP_FNAME	EMP_LNAME
▶	Sam's Stock Pot	C	Central	9	Ben	Sidhu
	Gadgets "R" Us	C	Central	9	Ben	Sidhu
	Loonie Mart #107	K	Key	9	Ben	Sidhu
	Rosch Dry Goods Inc	E	East	3	Jocelyn	Scorer
	The Chef's Assistant	N	North	4	Bill	Williams
*						

The table combines information about customers and regions

The “Central” region contains more than one customer.

Sales reps are assigned to regions, not customers.



- Deletion anomaly** – Assume that THE CHEF'S ASSISTANT goes out of business and you delete its record from the table. Since THE CHEF'S ASSISTANT is the only customer assigned to the North sales region, deleting the customer also wipes out the only record you have that Bill Williams is the sales rep for the North region.
- Modification anomaly** – Suppose that Sabine Villeneuve takes over responsibility for the Central region. Since each region can have multiple customers, you have to make the change for all customers assigned to the Central region. Not only is this extra work, it creates the potential for inconsistent data. What happens, for example, if you forget to change some of the records? Is the sales rep for the Central region Sabine Villeneuve or Ben Sidhu?

7.1.1 Normalized table design

The anomalies identified above can be avoided by splitting the table in [Figure 7.1](#) into three separate tables:

- **Customers** – information about customers only,
- **Regions** – information about regions only, and
- **Employees** – information about employees only.

Once the separate tables are created, what is needed is a means of linking to the tables together. As you saw in [Lesson 6](#), linking tables in a relational database is accomplished through the use of foreign keys. For example, we use **Customers.RegionCode** to create a link to the appropriate record in the **Regions** table. Once we know a customer has a **RegionCode = “C”**, we can switch to the **Regions** table, find the correct record, and determine everything we need to know about the region. Since other customers in the Central region point to the same record in the regions table, information about the region (such as its name, the sales rep who is responsible for the region, and so on) is stored in exactly one place.

A database schema in which every entity has its own table and redundancy is minimized is said to be **normalized**. The redundancy that does exist in a normalized table structure is minimized by using very short fields (e.g., **RegionCode**) to link the tables.



The science of normalization is a bit more complex than this quick summary. However, a common-sense understanding of the concept is all we require for the purposes of this project.



7.1.2 Making relationships explicit

In [Lesson 6](#), you created the foreign key infrastructure for several one-to-many relationships. In this lesson, you are going to *declare* the relationships between tables explicitly and tell ACCESS to enforce **referential integrity**. Briefly, referential integrity is a tool that helps minimize the amount of garbage that gets entered into your database. A more formal definition of this important database concept can be found in subsequent sections.

7.2 Learning objectives

- understand how to create relationships in ACCESS
- edit and delete relationships
- create a relationship for a concatenated key
- understand referential integrity and its relationship to business rules

7.3 Exercises

Once you are 100-percent happy with your table structure, and before you populate the tables or go on to create queries and forms for your application, you should use the **relationships window** to formally declare the relationships between your tables.



Although primary keys and foreign keys are all you need to implement relationships in a relational database, the DBMS does not “know” about the relationships until you declare them. In ACCESS, you declare relationships by dragging and dropping within the relationships window.

7.3.1 Creating a relationship

Recall that the relationships window in ACCESS is similar to an entity-relationship diagram (see [Figure 7.2](#)). In this section, you are going to use the relationships window to specify a one-to-many relationship between the **Regions** and **Customers** tables.



Make sure the database window is in the foreground and select **Tools** → **Relationships** from the main menu.

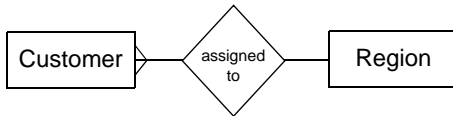


Remember, you can always bring the database window to the foreground by selecting **Window** → *<database name>* from the main menu.

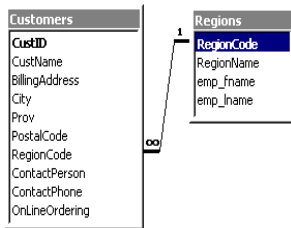
At this point, the relationships window should be empty.




FIGURE 7.2: Comparison of an entity-relationship diagram and the relationship feature in ACCESS.



"Each customer is assigned to one and only one region. Each region may be assigned any number of customers, including zero."



- To add tables, select **Relationships** → **Show Table** from the menu or press the show table icon () on the toolbar.

As shown in [Figure 7.3](#), a dialog box pops up that allows you to select one or more tables and add them to the relationships window.

- While holding down the **Ctrl** key, click on the **Customers** and **Regions** tables.
- Press the **Add** button and close the dialog box.



The dialog box in [Figure 7.3](#) is known as a **modal dialog box**. If a dialog is modal, it does not permit you to do anything else within the program until the dialog box is closed (by pressing **OK**, **Cancel**, **Close**, or whatever choices are offered).

Once your tables have been added to the relationships window, you tell ACCESS which primary key/foreign key fields are used link the tables together. The general procedure is to drag the primary key from the "one" side of the relationship on to the foreign key on the "many" side of the relationship.



Although it should not matter, the sequence described above (drag from the "one" side to the "many" side) appears to be critical in some situations.

- If necessary, resize the field list for the **Customers** table so that the **RegionCode** field is visible.
- Select the **RegionCode** field from the **Regions** table (primary key on the "one"



FIGURE 7.3: Select the tables to add to the relationships window.

1 Select **Tools** → **Relationships** from the main menu or press the **Show Table** icon on the toolbar to bring up the Show Table dialog.

2 Hold down the **Ctrl** key and click on the tables to multi-select.

3 Use the **Add** button to add the tables to the relationship window.

! The Show Table dialog is modal—it must be closed before you can continue working in ACCESS.

side) and drag it onto the **RegionCode** field in the **Customers** table (the foreign key on the “many” side), as shown in [Figure 7.4](#).

At this point, the relationship properties box should pop up, as shown in [Figure 7.5](#).

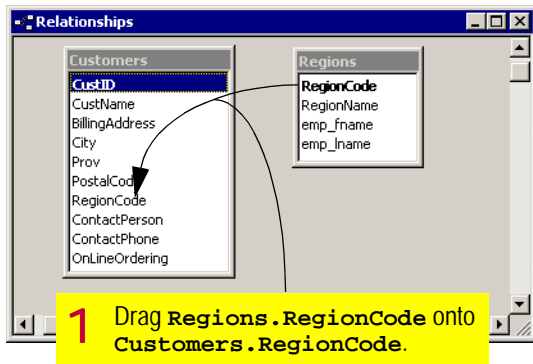
➔ Ensure that the correct field(s) are used for the relationship.



For single-field relationships, you only have to make changes in the properties dialog if you have made a mistake dragging and dropping. For multiple-field relationships, however, you must always specify the related fields manually.



FIGURE 7.4: Drag the primary key on the “one” side of the relationship on to the foreign key on the “many” side of the relationship.



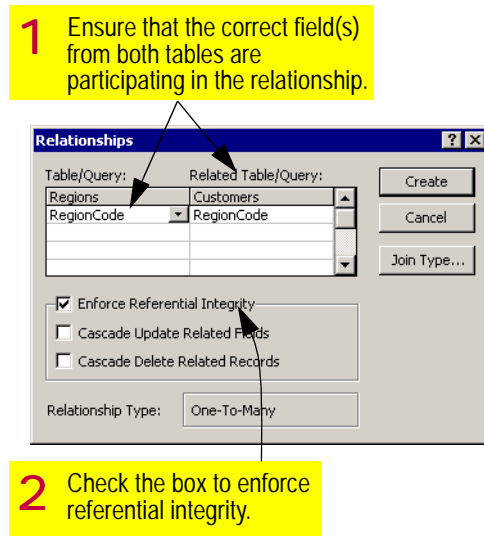
- Ensure referential integrity is enforced (see [Section 7.4.3](#) in the discussion for more information on referential integrity).

When you are done, your relationship should look like the one in the bottom half of [Figure 7.2](#).

7.3.2 Editing a relationship

To edit or delete an existing relationship, you simply right-click on the relationship line and select from the resulting **context menu**.

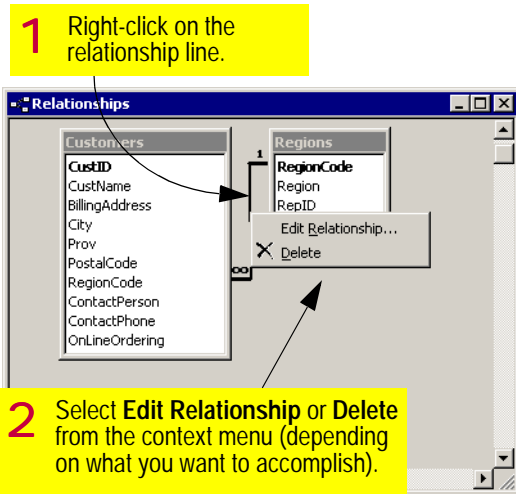
FIGURE 7.5: Set the properties of the relationship between *Regions* and *Customers*.



- Right click the relationship line. The context menu, shown in [Figure 7.6](#), should appear.
- Select **Edit Relationship** to get the relationships property dialog.



FIGURE 7.6: To edit a relationship, use the context menu.



You cannot modify a table's relationships if the table is open. Because of this, it is good practice to close all windows except the database window before opening the relationship window. If you forget to do this, simply close the relationship property sheet, close the table in question, and try again.

7.4 Discussion

7.4.1 Creating a relationship using a concatenated key

The foreign key linking the **Customers** table with the **Regions** table is a single field: **RegionCode**. However, recall that if the primary key on the "one" side of the relationship is concatenated, the foreign key will also be concatenated. For example, the foreign key currently used to link the **Regions** table with the **Employees** table consists of two fields: **emp_fname** and **emp_lname**.

Since you do not have access to the **Employees** table at this point (it is coming in [Lesson 8](#)), you cannot practice creating relationships for concatenated key yet. However, the process is identical to that described in [Section 7.3.1](#) with a couple of minor changes:

1. When selecting the fields on the "one" side (recall [Figure 7.4](#)), use the **Ctrl** key to multi-select all the field in the primary key.
2. When verifying the linking fields (recall [Figure 7.5](#)), you may have to set the linkages for the fields manually.



We will revisit the procedure for declaring concatenated foreign keys in [Section 8.3.4.1](#) once we have the **Employees** table in place.



7.4.2 Populating tables on the “many” side

By enforcing referential integrity when you declare relationships, ACCESS prevents you from creating meaningless links when you enter data into the tables. Specifically, ACCESS prevents you from entering a value in the foreign key on the “many” side that does not exist in the primary key on the “one” side. As a consequence, enforcing referential integrity affects the order in which you can populate tables.

For example, you cannot add a complete **Customer** record until the **Regions** table is populated. This is because values for **RegionCode** in the **Customers** table must correspond to valid values of **RegionCode** in the **Regions** table. If no valid regions are defined, **RegionCode** in the **Customers** table must be set to NULL for all customers.



NULL is a special value in database terminology that means “the absence of a value”. Thus, when you are asked to set a field’s value to NULL, it does not mean you type in the letters N-U-L-L. Nor does it mean you enter a blank space or a zero. NULL means empty.

Since you populated the **Regions** table in [Section 5.3.6](#), you are now in a position to populate the **Customers** table (including the **Customers.RegionCode** field).

One way to populate the **Customers** table is to go through the orders that have been faxed to you and copy the customer information from the order headers. To spare you this drudgery, however, you will learn how to take a spreadsheet containing this information and append it to your **Customers** table in [Lesson 8](#).

7.4.3 Referential integrity

An important feature of modern DBMS systems is that they support enforcement of referential integrity at the table level. What is referential integrity? Essentially, referential integrity means that every record on the “many” side of a relationship must have a corresponding record on the “one” side (or else be NULL-valued). Enforcing referential integrity means that you cannot, for instance, create a new record in the **OrderDetails** table without having a record with the same value of **OrderID** in the **Orders** table.

In addition, referential integrity prevents you from deleting records on the “one” side if related records exist on the “many” side. This eliminates the problem of “orphaned” records created when parent records are deleted.

Referential integrity is especially important in the context of transaction processing systems. Imagine that someone comes into your store, makes a large purchase, asks you to bill



customer number 123, and leaves. What if your order entry system allows you to create an order for customer 123 without first checking that such a customer exists? If you have no customer 123 record, where do you send the bill?

In systems that do not automatically enforce referential integrity, these checks have to be added to the application using a programming language. This is just one example of how table-level constraints can save you programming effort.

7.4.4 Numeric foreign keys

Unwanted referential integrity errors sometimes occur when you use a numeric (e.g., long integer) field for a foreign key. Recall from [Figure 5.2](#) that each field has a **Default** property that you can specify when creating the table. When a new record is added to the table, ACCESS sets the field to the default value.

The problem is that ACCESS automatically assumes that the default value for numeric fields is zero (the default is NULL for non-numeric fields). However, you often do not have a value of zero in the table on the “one” side of the relationship.

To illustrate, consider the foreign key used to link your **Customers** and **Orders** tables. If you have used an AutoNumber for the **CustID** field

in your **Customers** table, your customers will have **CustID** = 1, 2, However, when you add a new record to the **Orders** table, the value of the foreign key, **Orders.CustID**, will be set by ACCESS to its default value of zero. Since you do not have a record with **CustID** = 0 in the **Customers** table, a referential integrity error occurs unless you change the value of **Orders.CustID** to some other value.

There are two ways to get around this problem:

1. Set the **Default** property of any numeric foreign keys to NULL (i.e., leave the property blank) instead of zero when you are designing the table. In this way, “unknown” values will be NULL (which is perfectly legal), instead of zero.
2. Leave the default property set to zero, but create a record in the table on the “one” side with a primary key of zero. For example, create a dummy customer called “unknown” with a **CustID** = 0. In many ways, this approach is preferable to (1) above because NULL values are ambiguous. They can indicate “not applicable”, “unknown”, “incomplete”, and so on.



If the primary key on the “one” side of the relationship is an AutoNumber, you cannot add a **<primary key>** = 0 record to your table (AutoNumbers start at one). But there is no reason that you cannot use



a different number. For example, you designate `CustID = 1` the “unknown” customer and set the default property for `Orders.CustID` to one.

7.5 Application to the project

- ➔ Declare relationships for all the tables you have created so far. Make sure that referential integrity is enforced in every case.
- ➔ When the relationship properties window is open, press the “what’s this” button and click on the “Cascade Delete Related Records” check box, as shown in [Figure 7.7](#). This brings up the context-sensitive help box. Press the “more information” button at the bottom to read more about cascading deletes.
- ➔ For the relationship between `Orders` and `OrderDetails`, check the box to specify cascading deletes (see [Figure 7.5](#)).



Whenever you are dealing with a **weak entity** (an entity which depends on another entity for existence), it often makes sense to specify cascading deletions. In this case, the concept of an `OrderDetail` record without a

FIGURE 7.7: Use context-sensitive help to learn about cascading deletes

1 Press the “what’s this” button. The cursor changes to a question mark (indicating “context sensitive” help).

2 Click on the item you wish to learn more about. In this case, get help on **Cascade Delete Related Records**.

corresponding `Order` record is meaningless.