PDM9L7 DATABASES 2

USING RELATED TABLES IN MS ACCESS

LEARNING OUTCOMES

By the end of this practical students should be able to:

★ Create a multi-entity database.
★ Define relationships between tables.
★ Execute queries using data from several tables.
★ Use calculated fields in queries.
★ Import data into a table.

TRANSFERABLE SKILLS

The ability to analyse a problem in terms of related entities.

Remember:

Attendance: Record your attendance each week. Go to the groups (V: drive) folder for PDM9L7 and click on the register icon.

Database Test: There is a test on the database material shortly (session 8, this workshop is session 6).

Checkpoints: There is a checkpoint in every worksheet. You must attain at least 4C in the checkpoints to get a grade for PDM9L7. If you don’t finish a checkpoint in class, finish it off in your own time and show us the results next workshop.

In your own time: The more you practise, the more proficient and confident you will become. Do the worksheet again during the week to consolidate your learning.

Saving: Don’t forget: if you plan to work on your documents elsewhere in a previous version of the Office Suite, then you must save them in an appropriate format. Please ask if you don’t know what this means.

Your first database (mark 2)

Our one table database was useful up to a point but had some serious limitations. For example, what if the film had more than one writer (lots of films do)? How can we store information about actors, directors, executive producers, etc?

Consider the way we have been storing the names of several actors in a field called Cast. Although possible, it is quite difficult to look for individual actors and quite impossible to
find out anything about actors apart from their names. If we tried to store biographical data about a particular actor, where would we put it?

We could add extra fields to our Movies table to store the name of each actor separately (Cast1, Cast2, Cast3, etc.) but how many fields would we need: three? ten? twenty-five? It depends on the film! Even if we do this, it still does not solve the problem about storing actor data. Fortunately, there is a solution.

Incidentally, this would be a good time to go back and read the introduction to databases back at the beginning of last week's worksheet. Some of the concepts may make a little more sense now.

Remember we talked about entities? Entities are what we call the fundamental things that we build our database from. An entity will be represented by a table in a database. In our database, examples of possible entities are Movies, Actors, Directors and Writers. Each movie is an example (or occurrence) of the Movies entity and each actor is an example of the Actors entity. Putting it in another way, each movie is a row in the Movies table and each actor is a row in the Actors table. Deciding which things should be entities in our database is not an exact science. There is no single correct database structure. It depends on the context. I said above that Writers could be an entity and certainly would be in a full-scale database about films. However, if we decide that we will not record any details about writers other than their names and, if we assume that each movie has only one writer, then keeping Writer as a field in the Movies table will be OK. We’ll do just that for simplicity.

Relationships

The various entities in our database are related. We could say that, in general, each movie will feature several actors, while each actor will appear in several movies. There are exceptions; some actors will never get a movie role and some movies are animations! In general, we could say that each director will direct several movies, while each movie will be directed by only one director. Some films have more than one director but we’ll ignore that possibility.

When we are designing our database the types of relationships between entities are important. If we assume that each film will only have one director but that each director will direct one or more films, then we describe the relationship between directors and movies as a one-to-many relationship. We can display the relationship in a diagram.

The arrowheads describe the multiplicities of the relationship. These relationships are straightforward to represent in a database, as we’ll see.

The relationship between actor and movie is described as a many-to-many relationship. These are less straightforward to represent but are central to the implementation of working databases. We will see how to handle these relationships in our database later.

One-to-many relationships

» Start up Access and create a new blank database called movies2. Save it in the same folder as your original movies database from last week.

» Open your movies database from last week. You can reuse bits of it. (You may get a security warning when opening your database. Select ‘options’ from the Security Warning bar if it appeared underneath the ribbon. Click ‘Enable this content’ and hit OK.)
» Arrange things so that you can see both database windows on the screen.

» **Copy** the Movies table from movies to movies2. You can do this by dragging it from one database window (the movies window) to the other (the movies2 window).

» Just call the table Movies again.

» Close the original movies database window (we’re finished with that database now). The new database will use the copy of the Movies table as a starting point.

» Create a new table in Design View called Directors.

» Set it up as follows.

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Data Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>DirectorID</td>
<td>AutoNumber</td>
</tr>
<tr>
<td>Name</td>
<td>Text</td>
</tr>
<tr>
<td>Born</td>
<td>Number</td>
</tr>
<tr>
<td>Died</td>
<td>Number</td>
</tr>
<tr>
<td>Birthplace</td>
<td>Text</td>
</tr>
</tbody>
</table>

» Make sure DirectorID is the Primary key.

» Switch to Datasheet View and enter the following data. (Note: you cannot manually enter the DirectorID as it is an autonumber, just start typing in the name and Access will auto generate the number for you.)

<table>
<thead>
<tr>
<th>DirectorID</th>
<th>Name</th>
<th>Born</th>
<th>Died</th>
<th>Birthplace</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Alfred Hitchcock</td>
<td>1899</td>
<td>1980</td>
<td>UK</td>
</tr>
<tr>
<td>2</td>
<td>Billy Wilder</td>
<td>1906</td>
<td>2002</td>
<td>Poland</td>
</tr>
<tr>
<td>3</td>
<td>Carol Reed</td>
<td>1906</td>
<td>1976</td>
<td>UK</td>
</tr>
<tr>
<td>4</td>
<td>Howard Hawks</td>
<td>1896</td>
<td>1977</td>
<td>USA</td>
</tr>
<tr>
<td>5</td>
<td>John Huston</td>
<td>1906</td>
<td>1987</td>
<td>USA</td>
</tr>
<tr>
<td>6</td>
<td>Orson Welles</td>
<td>1915</td>
<td>1985</td>
<td>USA</td>
</tr>
<tr>
<td>7</td>
<td>Robert Siodmak</td>
<td>1900</td>
<td>1973</td>
<td>Germany</td>
</tr>
</tbody>
</table>

» Save the table.

» Open the Movies table in Design View

» Change the name of the Director Field to DirectorID.

» What if we want to have two films with the same name? Let’s add a MovieID like we have DirectorID above. Add a new row above Title (right click on Title and select Insert Rows). Call it MovieID and make it an AutoNumber. Make this the table’s primary key.

» Switch back to Datasheet View and you will see that the MovieID column has been filled in automatically for you.

» Replace the directors’ names in the Movies table with their DirectorID numbers from the Directors table. Take care here; it’s easy to make a mistake. Check your table against this listing.

<table>
<thead>
<tr>
<th>MovieID</th>
<th>Title</th>
<th>DirectorID</th>
<th>Writer</th>
<th>Year</th>
<th>Runtime</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Sunset Blvd.</td>
<td>2</td>
<td>Charles Brackett</td>
<td>1950</td>
<td>110</td>
</tr>
<tr>
<td>2</td>
<td>Double Indemnity</td>
<td>2</td>
<td>James Cain</td>
<td>1944</td>
<td>107</td>
</tr>
<tr>
<td>3</td>
<td>The Third Man</td>
<td>3</td>
<td>Graham Greene</td>
<td>1949</td>
<td>104</td>
</tr>
<tr>
<td>4</td>
<td>The Maltese Falcon</td>
<td>5</td>
<td>Dashiell Hammett</td>
<td>1941</td>
<td>101</td>
</tr>
</tbody>
</table>
Change the data type of the DirectorID field in the Movies table to Number. NB We do not want this field to have an AutoNumber type because its value references the DirectorID field in the Directors table. (Access may complain that there may be some data lost as we have reduced the allowed size of the information in that cell. Just hit OK).

Creating a relationship or join between two tables will allow us to create queries which can involve both tables (e.g. Find the birthplaces of all directors who directed films made in 1946). For this to be possible, the two tables must have (usually) one field in common. This field will normally be the primary key of one table, known as the Primary Table. In this example, the DirectorID (primary key) field in the Directors table (primary table) forms the relationship with the DirectorID Field in the Movies table (related table). The DirectorID field in the Movies table is known as a Foreign Key. Incidentally, we do not need to give the join field the same name in both tables.

Essentially, what we are saying is that any Director that appears in the Movie table should be one of the Directors listed in the Directors table. A well-designed database will enforce this kind of requirement. It makes sense really. Why would we want a database to contain a movie whose director is not listed as a director?

Save and close both tables. (To close right click on the table tab and select Close.)

Press the Relationships button in Database Tools. The next step is to choose the tables between which we wish to form the relationship.

In the Show Table dialog click Directors and press Add.

Now click Movies and press Add.

Close the Show Table dialog.

To form the relationship click and drag the DirectorID field in the Directors table on top of the DirectorID field in the Movies table.

At this point, the Edit Relationships dialog will appear confirming the tables that you are intending to join, and allowing you to set up some properties for the join.
» Check the boxes Enforce Referential Integrity, Cascade Update Related Fields but leave the Cascade Delete Related Records unchecked.

You can gloss over this referential integrity bit if you want but if you are curious...

Enforcing referential integrity means that, because we now have more than one table, and because the tables are related to each other, changes to one of the tables can affect the relationship between them and by enforcing referential integrity, Access will warn us if we are trying to do something stupid. Checking this box, on its own, means that you cannot change the value of an entry in the primary key field of the primary table. This is a Good Thing because doing so would break the relationship between the tables.

Checking the Cascade Update Related Fields box means that you can change the value of an entry in the primary key field of the primary table but that doing so will cause the corresponding records in the related table to be updated as well. In other words, your update will cascade through to related tables. This is Almost Certainly a Good Thing.

Checking the Cascade Delete Related Records box means that if you delete a record in the primary table any related records in other tables will also be deleted. This can be Quite a Useful Thing if you are sure that you really want to do it but a Very Bad Thing if you do it by accident. You could lose a lot of data very easily. It is better not to have this option permanently turned on.

You can, at any time, edit or delete a relationship by right clicking on its join line in the Relationships Window and choosing the Edit Relationship option.

» OK, press the Create button to confirm the relationship.
There should now be a line joining the two fields. The one-to-many relationship should be indicated by a number 1 at the Directors end of the line and an infinity symbol (∞) at the Movies end.

» Go back to the Directors table in Datasheet View.

There should now be a column of plus buttons (+) to the left of the primary key field. Clicking any one of these will reveal the sub datasheet associated with that record.

» Try it! The records displayed are the ones related to the chosen value of the primary key. To close the sub datasheet, just press the button again. This time it has a minus sign on it.

» Right-click on the row marker for the Alfred Hitchcock row in the Directors table and choose Delete Record.

Access should tell you that it couldn’t be deleted because the Movies table contains related records. This is because you did not check the Cascade Delete Related Records box earlier. If you did check it, then you are going to have a bit of typing to do! Try the undo button first!

If you really wanted to delete Alfred Hitchcock from the database, without breaking the relationship, you would edit the relationship and switch on Cascade Delete Related Records to allow Access to cascade the delete operation through to any related tables.

Lookup Fields

When you enter a new record in the Movies table, it would be useful not to have to look up the DirectorID in the Directors table. This can be arranged. We can convert the DirectorID field in the Movies table into a drop-down list containing all the entries from the DirectorID field in the Directors table.

Firstly we need to break the relationship between the two tables.
Often, when a database is in the construction phase, relationships need to be made and broken to facilitate design changes in the database structure. As the database becomes more complex, this process becomes trickier so designing the database structure carefully in the first place is a Very Good Idea.

- Close the Movies and Directors.
- In the Relationships Window, right-click on the relationship line connecting the two table icons and choose the Delete option. You will have to be precise about positioning the mouse pointer over the line otherwise you will get the wrong menu.
- Access will ask you if you are sure that you want to permanently delete the relationship. Press Yes.

- SAVE everything first.
- Open the Movies table in Design View.
- Select the Data Type field for the DirectorID Field and choose Lookup Wizard.
- Select I want the lookup column to look up the values in a table or query box (it should be selected anyway) and press Next.
- The next step should indicate that we are going to use the Directors table to provide the values for the lookup column (there is no other table yet), so just press Next again.
- The lookup wizard now wants to know which column in the Director table will be used to provide the values for the lookup. Choose the DirectorID and Name fields by moving them into the Selected Fields list. This will ensure that our drop-down list will contain enough information to allow us to choose the director. Press Next again.
- Now the lookup wizard wants to know how to present your values. Choose Name and Ascending order. Press Next again.
- Uncheck the Hide key column box. Adjust the width of the DirectorID column to be as small as possible (while still allowing you to read the column header).
- Press Next again.
- Make sure the DirectorID field is selected from the list of available fields to use as a unique identifier for the row and press Next again.
- Leave the column name as it is (DirectorID) and press Finish.
- Save the table when prompted. Notice that the DirectorID field has retained its Number data type, but …
- Click on the Lookup tab in the Field Properties section in the bottom half of the Movies table window by way of confirmation that the lookup field has been set.

If you are interested, look at the settings. The Display Control (screen widget) is now a Combo Box (i.e. a drop-down list) instead of a simple Text box and the Row Source (i.e. the data for the drop-down list) comes from an SQL query.

- Switch back to Datasheet View.
- Add a new record for the movie titled Shadow of a Doubt.
- Click in the DirectorID field. A drop-down list appears allowing you to select the director from the available options.
Choose Alfred Hitchcock. His DirectorID (1) should appear in the field.

Fill in the rest of the record:

Writer: Gordon McDonell
Cast: Teresa Wright, Joseph Cotten
Year: 1943
Runtime: 108

If we had chosen to hide the key field when we arranged the lookup, we would only see the list of directors' names and indeed Access would display the names in the DirectorID field even although the column actually contains the DirectorID number. This is just Access trying to be helpful but, for the moment, it's probably less confusing to do it the way we have. It's your choice however.

Save the table and close it.

Go back to the Relationships window. Notice that the relationship between Director in the Movie table and Name in the Directors table has been re-established with a thin line connecting the two. The one-to-many multiplicity is not indicated, however.

Right-click on the line and choose Edit Relationship.

As before, check the boxes Enforce Referential Integrity, Cascade Update Related Fields but leave the Cascade Delete Related Records unchecked. The relationship should now be back to the way it was before it was deleted.

Open the Directors table and add a new director:

<table>
<thead>
<tr>
<th>Name</th>
<th>Born</th>
<th>Died</th>
<th>Birthplace</th>
</tr>
</thead>
<tbody>
<tr>
<td>Raoul Walsh</td>
<td>1887</td>
<td>1980</td>
<td>USA</td>
</tr>
</tbody>
</table>

Save the Directors table.

Open the Movies table again and add the record below. When you click on the Director ID field, the drop-down list should now contain the new director. When a table contains a Lookup field, the lookup is performed afresh every time the table is opened meaning that your lookup list is always up to date. This type of Lookup list is called a Dynamic Lookup List. It is possible to create Static Lookup Lists by typing values into the list directly if you know that the list of values is not going to change over time.

<table>
<thead>
<tr>
<th>Title</th>
<th>DirectorID</th>
<th>Writer</th>
<th>Cast</th>
<th>Year</th>
<th>Runtime</th>
</tr>
</thead>
<tbody>
<tr>
<td>White Heat</td>
<td>8</td>
<td>Virginia Kellogg</td>
<td>James Cagney, Virginia Mayo</td>
<td>1949</td>
<td>114</td>
</tr>
</tbody>
</table>

Save the Movies table.
Many-to-many relationships
We said that each actor can appear in several films and that each film will feature several actors. This is a classic many-to-many relationship. How do we deal with this situation? Well, guess what, we are going to need more tables! It should be fairly obvious, I hope, that we will need an **Actors** table so let’s create it. It will be similar in structure to the **Directors** table. The contents are listed in the table below but to save you a lot of typing, let’s look at another Access feature, namely, **importing data from an external source**.

One of the issues involved when building a database is where do you get the data from? You may have to create it yourself by transcribing it from a notebook or other handwritten source based on years of diligent research. On the other hand, data sources may already exist, which can be harvested electronically.

In this case we’ll get Access to import the data for the Actors table from a text file, which has been prepared by some nice, thoughtful person (me).

We can either create the table first, or we can get Access to do this for us. For this example we shall make the table before importing.

**First, create the Actors table, in Design View, as follows:**

<table>
<thead>
<tr>
<th>ActorID</th>
<th>Name</th>
<th>Born</th>
<th>Died</th>
<th>Birthplace</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>William Holden</td>
<td>1918</td>
<td>1981</td>
<td>USA</td>
</tr>
<tr>
<td>2</td>
<td>Gloria Swanson</td>
<td>1897</td>
<td>1983</td>
<td>USA</td>
</tr>
<tr>
<td>3</td>
<td>Fred MacMurray</td>
<td>1908</td>
<td>1991</td>
<td>USA</td>
</tr>
<tr>
<td>4</td>
<td>Barbara Stanwyck</td>
<td>1907</td>
<td>1990</td>
<td>USA</td>
</tr>
<tr>
<td>5</td>
<td>Joseph Cotten</td>
<td>1905</td>
<td>1994</td>
<td>USA</td>
</tr>
<tr>
<td>6</td>
<td>Alida Valli</td>
<td>1921</td>
<td>2006</td>
<td>Italy</td>
</tr>
<tr>
<td>7</td>
<td>Orson Welles</td>
<td>1915</td>
<td>1985</td>
<td>USA</td>
</tr>
<tr>
<td>8</td>
<td>Humphrey Bogart</td>
<td>1899</td>
<td>1957</td>
<td>USA</td>
</tr>
<tr>
<td>9</td>
<td>Mary Astor</td>
<td>1906</td>
<td>1987</td>
<td>USA</td>
</tr>
<tr>
<td>10</td>
<td>Charleton Heston</td>
<td>1924</td>
<td>2008</td>
<td>USA</td>
</tr>
<tr>
<td>11</td>
<td>Janet Leigh</td>
<td>1927</td>
<td>2004</td>
<td>USA</td>
</tr>
<tr>
<td>12</td>
<td>Farley Granger</td>
<td>1925</td>
<td></td>
<td>USA</td>
</tr>
<tr>
<td>13</td>
<td>Ruth Roman</td>
<td>1922</td>
<td>1999</td>
<td>USA</td>
</tr>
<tr>
<td>14</td>
<td>Robert Walker</td>
<td>1918</td>
<td>1951</td>
<td>USA</td>
</tr>
<tr>
<td>15</td>
<td>Cary Grant</td>
<td>1904</td>
<td>1986</td>
<td>England</td>
</tr>
<tr>
<td>16</td>
<td>Ingrid Bergman</td>
<td>1915</td>
<td>1982</td>
<td>Sweden</td>
</tr>
<tr>
<td>17</td>
<td>Claude Rains</td>
<td>1889</td>
<td>1967</td>
<td>England</td>
</tr>
<tr>
<td>18</td>
<td>Lauren Bacall</td>
<td>1924</td>
<td></td>
<td>USA</td>
</tr>
<tr>
<td>19</td>
<td>Burt Lancaster</td>
<td>1913</td>
<td>1994</td>
<td>USA</td>
</tr>
<tr>
<td>20</td>
<td>Ava Gardner</td>
<td>1922</td>
<td>1990</td>
<td>USA</td>
</tr>
<tr>
<td>21</td>
<td>Teresa Wright</td>
<td>1918</td>
<td>2005</td>
<td>USA</td>
</tr>
<tr>
<td>22</td>
<td>James Cagney</td>
<td>1899</td>
<td>1986</td>
<td>USA</td>
</tr>
<tr>
<td>23</td>
<td>Virginia Mayo</td>
<td>1920</td>
<td>2005</td>
<td>USA</td>
</tr>
</tbody>
</table>

Note that there is no ActorID AutoNumber field (yet). We’ll have to add that in after we have imported the data from the text file. Since Access **generates** the AutoNumbers, we cannot store them in a file, can we?

**Save the table as Actors and make sure there is no primary key. We’ll add one after we have imported the data. Close the table.**

**Select External Data from the ribbon. There are two main subsections; Import and Export. Select Text File from the Import section.**
» Press the Browse button and navigate to the Groups folder for PDM9L7 (remember the V: \ drive in the My Computer window?). Open the Databases folder and find the file Actors.txt.

» Select Append a copy of the records to the table and chose the Actors table you just created.

» Hit OK.

» The Import Text Wizard should detect that the file contains data, which is delimited by tab characters in between the field data. What Access will do, unless we tell it otherwise, is to assume that the tab stops in the text file are field separators (delimiters) and will place each item in a separate field along the current row. At the end of the row, Access will assume the start of a new record in the table. Clever Access! This is exactly what we want!

» Press Next. Access shows you how it intends to split the data in the table based on where the tab stops appear in the file. As you can see, it is possible to use other characters as delimiters. The main thing is to use a character that does not occur within the data itself. If you select Space, for example, you will see what I mean.

» Make sure that Tab is selected and press Next.

» Finally, press Finish. Access will ask you if you would like to save the import steps that you went through. Leave unticked and hit close.

» Open the Actors table. Success?

» Switch back to Design View and insert a row above the Name field.

» Use this row to create the field ActorID with data type AutoNumber.

» Make this field the primary key of the table.

» Save the table.
This is a foolproof way of making sure that each row in the table can be uniquely identified. The Name field could have been suitable as a primary key but we can’t be sure that two actors might not have the same name (although one of them should get a new agent!).

We cannot directly join tables that have a many-to-many relationship with each other. Instead, we must introduce another table in the middle (called a junction table). This is like the Book-Borrower problem we looked at, at the beginning of this module. In the Movies-Actors case, we need a record for each performance (one actor appearing in one film). Each of the two tables (Movies and Actors) will have a one-to-many relationship with the junction table, which we can call Performances. This models the reality of the situation because each actor will be associated with many performances (in different movies) and each movie will be associated with many performances (by different actors). Just as each row in the Actors table represents a single actor and each row in the Movies table represents a single movie, so each row in the Performances table represents a single performance.

We can say that Performances is an entity in our database. Entities like Performances, which depend on other entities for their existence, are called associative or intersection entities (also sometimes, weak entities). Entities such as Actors, Movies and Directors, which have independent existence (they are “what the database is all about”) are called independent entities (also sometimes, strong entities).

The Performances table must contain both the primary keys from the Movies and the Actors table as well as any additional information about the performance, e.g. the name of the character played.

Let’s let Access create the Performances table for us.

Choose External Data ➔ Text File as before.

Browse to Groups/PDM9L7/Databases folder and select Performances.txt.

Leave the Import the source data into a new table in the current database checked. Hit OK.

Leave Delimited checked. Hit Next.

Leave Tab checked. Hit Next.

Rename Field1 as MovieID. Hit Next.

Select No Primary Key for just now. Hit Next.

Access will call the new table Performances. Hit Finish. Select both the rows by clicking in the row markers while holding down the Control key.

Switch into Design View.

Rename Field2 as ActorID and Field3 as Character.

Notice Access chose the data type for us which it judged from the data it imported.

We could create a primary key field (e.g. PerformanceID) as an AutoNumber just as we did for the other tables but, this time, we’ll do something different. We will assume that any possible row in this table will be uniquely identified by the combination of MovieID and ActorID, in other words each actor only plays one character in any movie, so we can decide that the primary key of the Performances table will consist of the two fields together. This is called a composite key.

Select both the MovieID and the ActorID rows by holding down the ctrl-key on the keyboard. With both rows selected, press the Primary Key button. (Under the Design tab)
Save the Performances table.

**Something to think about:** There are movies in which one actor plays more than one character. Indeed there are movies in which the same character is played by more than one actor. We won’t concern ourselves with these cases but, if our table did contain such entries, it would mean that the combination of MovieID and ActorID would be insufficient to guarantee a unique entry in the table. How do you think we could make our database deal with these possibilities?

Open the Performances table in Datasheet view and check that everything looks OK. Row 1 should contain:

<table>
<thead>
<tr>
<th>MovieID</th>
<th>ActorID</th>
<th>Character</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>Joe Gillis</td>
</tr>
</tbody>
</table>

This is a record of the fact that the movie Sunset Blvd (MovieID = 1) features the actor William Holden (ActorID = 1) playing the character Joe Gillis. It is worth mentioning that, had we decided to use the **Title** field from the **Movies** table and the **Name** field from the **Actors** table as primary keys of their respective tables (instead of using AutoNumber ID fields), then the Performances table would have consisted of rows like:

<table>
<thead>
<tr>
<th>Movie</th>
<th>Actor</th>
<th>Character</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sunset Blvd</td>
<td>William Holden</td>
<td>Joe Gillis</td>
</tr>
</tbody>
</table>

Both approaches would have worked well in the present database but the approach we have used is arguably better and is certainly required in the construction of serious databases. It is not always necessary, however, to construct an **artificial** ID-type field. Many entities used in databases would have their own unique identifiers, for example:

- ★ Student Registration Numbers in a University database.
- ★ Bank Account Numbers in a Banking database.
- ★ Part Numbers in a manufacturing company database.

**Modelling the many-to-many relationship**

We need to form the relationships between Movies, Performances and Actors.

Open the **Relationships** Window. (**Database Tools** → **Relationships**). At the moment all we have is the relationship between the Movies and Directors tables.

Right click in the middle of the screen and select **Show Table**.

Add the **Actors** and **Performances** table to the layout.

The other two tables should already be there. If you accidentally add the same table twice (it is possible to form a relationship between a table and itself), Access will rename the second copy of the table with a name like **Actors_1, Actors_2**, etc. To remove the unwanted copy of the table, right-click on it and choose **Hide Table**.

Close the **Show Table** window.
» Close any of your tables you have open.

» Form the relationship between MovieID in the Movies table and MovieID in the Performances table.

» Form the relationship between ActorID in the Actors table and ActorID in the Performances table. Make the same choices as before regarding referential integrity and cascading updates. You may need to drag the table icons around a bit to prevent the lines from crossing. Notice that the only relationship between Movies and Actors is through the Performances table.

Open the Movies table in Design View.

» Delete the Cast Field (we simply don’t need it any more!). Right-click on the row marker and choose Delete Rows.

Multi-table queries

Our database now contains four tables and three relationships. Now it will be possible to construct queries involving one, two, three or four tables. In addition, the fields in each table are concerned only with the entity represented by that table, except for the foreign key fields, which are the things that link the table to other tables (hence the term foreign key). This satisfactory arrangement means that data is not duplicated unnecessarily and allows Access to …well… access, the data efficiently. We say that the tables are normalised.

To allow for some more interesting queries, we’ll add a few more records to the database.

» Add a new record to the Movies table.

<table>
<thead>
<tr>
<th>MovieID</th>
<th>Title</th>
<th>DirectorID</th>
<th>Writer</th>
<th>Year</th>
<th>Runtime</th>
</tr>
</thead>
<tbody>
<tr>
<td>12</td>
<td>The Magnificent Ambersons</td>
<td>6 (Orson Welles)</td>
<td>Orson Welles</td>
<td>1942</td>
<td>88</td>
</tr>
</tbody>
</table>

It was a well-known feature of Alfred Hitchcock’s movies that he always appeared, usually very briefly, on screen so we’ll credit him as an actor in each of the movies he directed.

» Add a new record to the Actors table.

<table>
<thead>
<tr>
<th>ActorID</th>
<th>Name</th>
<th>Born</th>
<th>Died</th>
<th>Birthplace</th>
</tr>
</thead>
<tbody>
<tr>
<td>24</td>
<td>Alfred Hitchcock</td>
<td>1899</td>
<td>1980</td>
<td>UK</td>
</tr>
</tbody>
</table>

Hitchcock’s DirectorID in the Directors table should be 1 and the MovieIDs of the movies that he directed should be 6, 7 and 10.
It is possible that your numbers might be different if you have deleted rows from the tables at any time. Access AutoNumber fields will not reuse numbers for rows that have been deleted. To recreate your AutoNumber Field without any missing numbers, you would have to delete the AutoNumber Field, which in turn would mean deleting any relationships involving the table. Then you would need to recreate the AutoNumber Field and re-establish the table relationships. This is actually straightforward for our present database but would be a nightmare for a large database.

Just make sure the numbers you enter in the following steps are the correct ones for your database. Assuming the numbers are the same as mine, add four new records to the Performances table. The first three record Hitchcock’s appearances and the fourth is for Joseph Cotten’s role in The Magnificent Ambersons.

<table>
<thead>
<tr>
<th>MovieID</th>
<th>ActorID</th>
<th>Character</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>24</td>
<td>Not credited</td>
</tr>
<tr>
<td>7</td>
<td>24</td>
<td>Not credited</td>
</tr>
<tr>
<td>10</td>
<td>24</td>
<td>Not credited</td>
</tr>
<tr>
<td>12</td>
<td>5</td>
<td>Eugene</td>
</tr>
</tbody>
</table>

Two-table query

We’ve already seen examples of single table queries in our first version of the database. Remember our first query, which was to list the titles and the directors of all movies made between 1946 and 1951. Well now, the data we need is divided between the Movies table and the Directors table. No problem…

One of the features of a database management system (DBMS) is that it will preserve data integrity. In principle, many users could be using the database at the same time, which is fine if all they are doing is querying the existing data. However, if one user is in the middle of adding new data to a table and a second user wants to add new data to the same table, the DBMS will not allow this to happen until the first user has completed her changes and has finished using the table. The table is said to be locked for updates. Access may not let you use a table to make a query if you already have the table open because it has locked it for use by any other process.

Close any open tables.

In the Create tab, click on Query Design.

In the Show Table dialog box, add the tables you want to use in your query (Movies and Directors).

Press Close to remove the dialog box.

You should see the familiar Query Design window, which should contain the icons for the chosen tables and the relationship link between them in the top half of the window. The equally familiar filter/sort grid should be visible in the bottom half of the window.

Click in the Field box in the leftmost column.
The drop-down list will contain every table and field combination available. (You can, if you wish, select the table first by clicking in the Table box, in which case you will only see the field names for the chosen table in the Field box.)

» Choose Movies.Title.
» In the next column, choose Directors.Name.
» In the third column, choose Movies.Year.

In the Year column, uncheck the Show box and, in the Criteria column, type in “ >= 1946 And <= 1951” (without the quotes).

» Switch to Datasheet View or press the Run button to see the results of the query. Hopefully, there should be no nasty surprises.

» Save the query as Movie and Director for Year 46 to 51.

Access has used the relationship between the tables to find only the combinations of movie titles and director names for records, in which the DirectorID Field is the same in both tables. If the relationship did not exist, Access would have combined every movie with every director and the result would have been nonsensical. (If you really want to prove this, try deleting the relationship between the tables and constructing the query again. If you are prepared to take my word for it, don’t bother!)

Three-table queries

Find the movie titles and character names for characters played by actors who were not born in the USA! This query will involve the tables Movies, Performances and Actors so create a new query and add the three tables.

» Set up the query fields to be Movies.Title, Actors.Name, Performances.Character and Actors.Birthplace.

» In the Criteria box of the Actors.Birthplace field, type Not "USA” (or <> “USA” The <> sign means ‘not equal to’. The quotes are advisable but not always necessary.)

» Run the query.

This query is essentially the same as asking for a list of all the actors who were not born in the USA, but through the relationships defined, we can access data in the other tables and also decide not to display the fields to which the criteria were applied.

Note that the column headings in the query are taken from the field names in the various tables. It would be useful to be able to provide column names more appropriate to the query bearing in mind that the same field names are used in more than one table already, e.g. Name, Born, Died etc.
» Switch back to Design View and position the cursor at the start of the word **Title** in the first **Field** box.

» Type in the words Movie Title followed by a colon (:).

» Change the other column headings to something more readable. The part you put before the colon is the new heading you’re giving to the column.

» Save the query as **Not born in the USA**. (Catchy title!)

**Another three-table query**

To find all the films in which Humphrey Bogart appeared, we need a three-table query involving **Actors**, **Performances** and **Movies**.

![Diagram of three tables: Actors, Movies, Performances]

We could search for the exact name…

![Field: Name, Table: Actors, Sort: Name, Show: Name, Criteria: Name]

…or use the **Like** criteria operator, as before…

![Field: Name, Table: Actors, Sort: Name, Show: Name, Criteria: Name]

Notice that we are only viewing fields from the **Actors** and **Movies** tables although the records of which actors appeared in which movies is actually contained in the **Performances** table. This kind of ‘complexity’ for very simple queries is the price we have to pay for storing our data efficiently and accessing it speedily.

» Save the query as **Like bog**.

**Rewind**

It’s worth rewinding a few steps here to understand what is actually going on when we form relationships between two tables. This is core to the understanding of relational databases.

» Close any open tables and queries.

» Open the Relationships window.
Break the relationship between Directors and Movies by right-clicking on the relationship line and choosing Delete. We'll recreate the relationship again in a minute.

Create a new query in Design View.

Add the tables Movies and Directors. There should be no line joining the tables. If there is, right-click on it and delete it.

Select the Movies.Title and Directors.Name fields for the query.

Run the query.

Because we have chosen to form a query using one field from each of two tables, which are not related, the only thing that Access can do is to combine every value of the Movies.Title field with every value of the Directors.Name field producing a table containing 8 x 12 (=96) rows, 8 being the number of directors and 12 the number of movies in the database. This is called the Cartesian Product of the two fields and, more importantly, is completely meaningless as a query result.

Close the query without saving it.

In the Relationships window, recreate the relationship between Movies and Directors exactly as it was before.

Recreate the last query exactly as it was before and run it. Now we should get a table containing each movie title with the movie’s director.

Without the relationship, Access has no idea that the DirectorID field in the Movies table refers to the same set of values as the DirectorID field in the Directors table. With the relationship in place, the query becomes:

“Select the Title field from the Movies table and the Name field from the Directors table but only show the combinations in which the DirectorID field from the Movies table is equal to the DirectorID field from the Directors table.”

Save this query as Who directed what?

Four-table query (ooOOoo!) Find the names of directors of movies featuring Joseph Cotten and show the movie titles and the names of the characters he played.

Start a new query.

Add all four tables.

Drag them around so that the relationship lines are untangled. I suggest, from left to right, Directors, Movies, Performances and Actors.
Add the fields **Directors.Name**, **Movies.Title**, **Performances.Character** and **Actors.Name** to the query. A quick way of doing this is to double-click on the field name inside the table icon.

Set the Criteria box in the **Actors.Name** field to be **“Joseph Cotten”** (note the spelling).

Run the Query. Go back to Design View and uncheck the Show box in the **Actors.Name** field. We don’t need to see it.

Save the query as **Directors, Movies and Characters for Joseph Cotten**.

### Try It Yourself

Consider the following four queries.

1. Which movies were directed by a director born in the UK?
2. Who directed really short movies (with runtime of less than 100 minutes)?
3. Which movies featured Ingrid Bergman?
4. What were the movies, directors and character names for the actors born before 1900?

### Checkpoint

Demonstrate your queries to the tutor. Remember you will get credit for reaching the checkpoint.

The database test is coming up in 2 weeks. The test will be open book so make sure you have your worksheets with you. You should make sure you complete and understand all the Databases worksheets before coming to the test to give yourself the best chance of achieving a good grade.