MICROSOFT ACCESS

Designing a Multi-table Query

Contents

- Introduction
- Planning Our Query
  - Step 1: Pinpointing the question we want to ask
  - Step 2: Identifying the information we need
  - Step 3: Locating the tables containing the information we need
  - Step 4: Determining the criteria our query should search for
- Query Criteria Reference Guide
- Joining Tables in Queries
- Creating a Multi-table Query
  - To create a multi-table query
- Practice

Introduction

- Queries can be difficult to understand and build if you don't have a good idea of what you're trying to find and how to find it. A one-table query can be simple enough to make up as you go along, but to build anything more powerful you'll need to plan the query in advance.

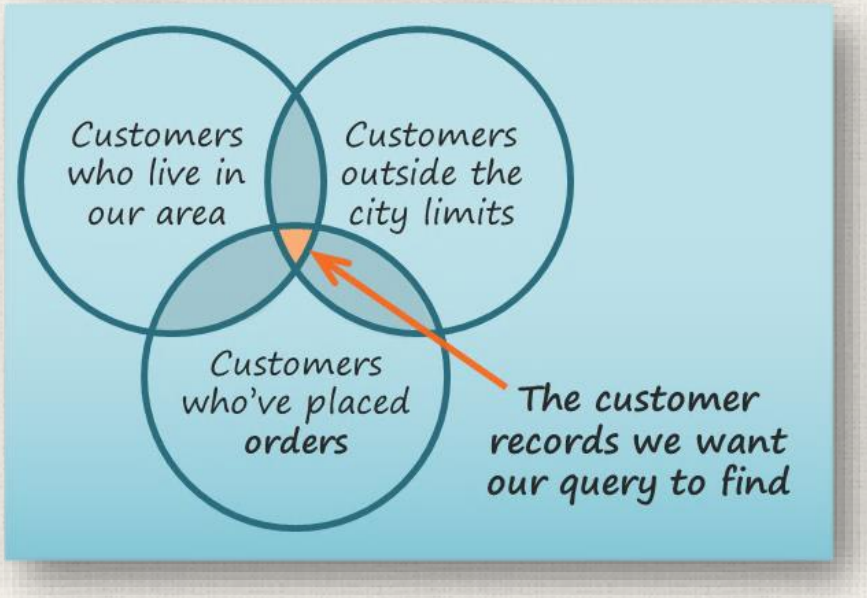
Planning a query

- When planning a query that uses more than one table, you should go through these four steps:
  1. **Pinpoint** exactly what you want to know. If you could ask your database any question, what would it be? Building a query is more complicated than just asking a question, but knowing precisely what question you want to answer is essential to building a useful query.
  2. **Identify** every type of information you want included in your query results. Which fields contain this information?
  3. **Locate** the fields you want to include in your query. Which tables are they contained in?
  4. **Determine** the criteria the information in each field needs to meet. Think about the question you asked in the first step. Which fields do you need to search for specific information? What information are you looking for? How will you search for it?
- This process might seem abstract at first, but as we go through the process of planning our own multi-table query you should start to understand how planning your queries can make building them a lot easier.

Planning our query

Step 1: Pinpointing the question we want to ask

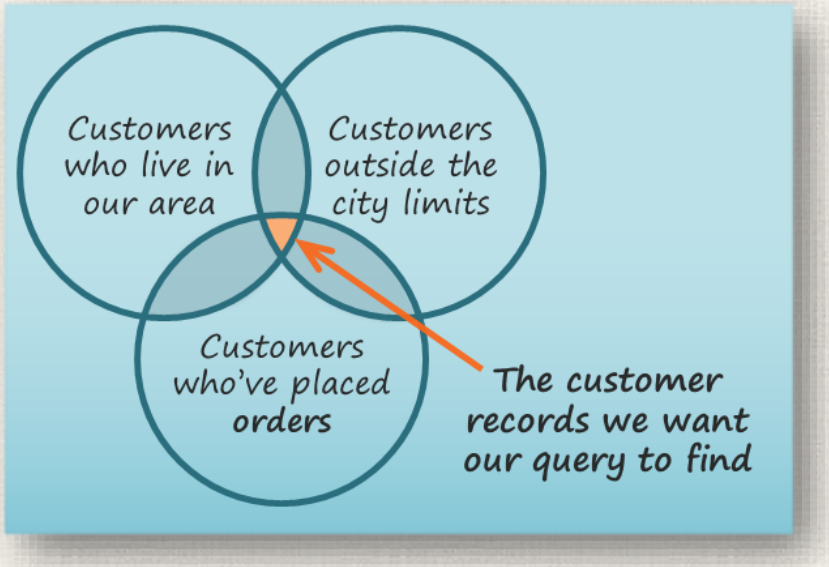
- Our bakery database contains many customers, some of whom have never placed an order but who are in our database because they signed up for our mailing list.
- Most of them live within the city limits, but others live out of town or even out of state. We want to get our out-of-town customers who've placed orders in the past to come back and give us another try, so we're going to mail them some coupons.



Planning our query

Step 1: Pinpointing the question we want to ask

- We don't actually want our list to include customers who live too far away; sending a coupon to someone who doesn't live in our area probably won't make that person come in. So we just want to find people who don't live in our city but who still live in our area.



- In short, the question we want our query to answer is this: **Which customers live in our area, are outside the city limits, and have placed an order at our bakery?**

Planning our query

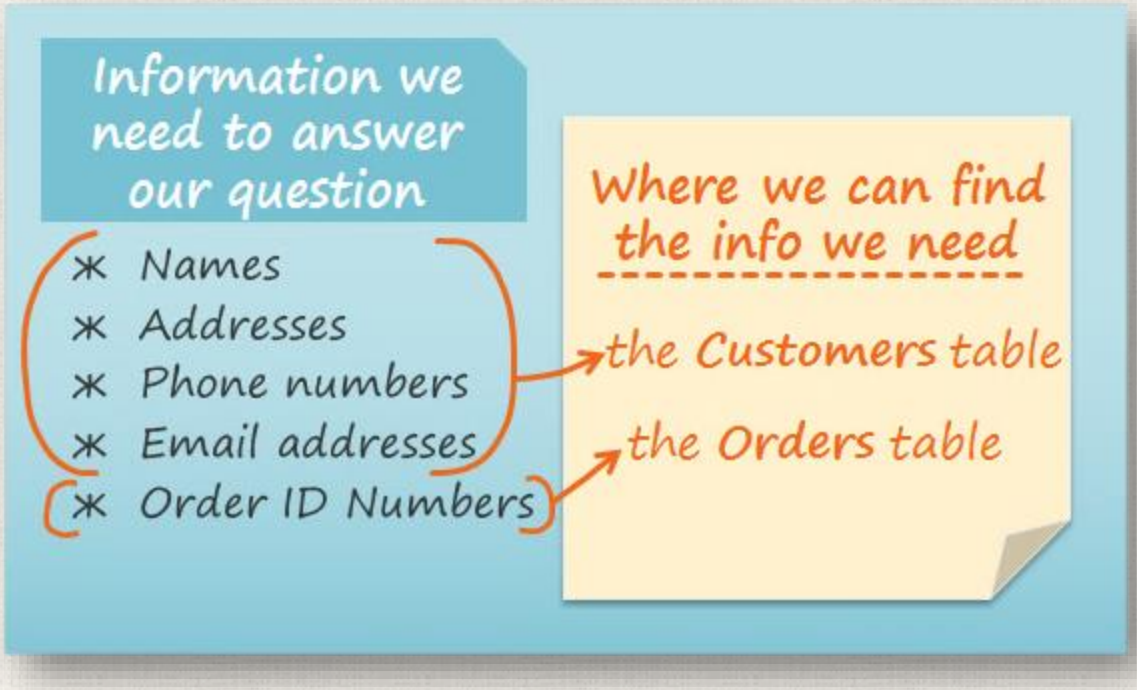
Step 2: Identifying the information we need

- What information might we want to see in a list about these customers? Obviously, we'll need the **customers' names** and their **contact information**—their **addresses**, **phone numbers**, and **email addresses**. But how are we going to know if they've placed orders? Each record of an order identifies the customer who placed that order. If we include the **order ID numbers**, we should be able to narrow our list down to only customers who have previously placed orders.

Planning our query

Step 3: Locating the tables containing the information we need

- In order to write a query, you need to be familiar with the different tables in your database. From working extensively with our own database, we know that the customer information we need is located in fields in the **Customers** table. Our **Order ID numbers** are in a field in the **Orders** table. We only need to include these two tables to find all of the information we need.



Planning our query

Step 4: Determining the criteria our query should search for

- When you set criteria for a field in a query, you are basically applying a filter to it that tells the query to retrieve only information that matches your criteria. Review the list of fields we are including in this query. How and where can we set criteria that will best help us answer our question?

Planning our query

Step 4: Determining the criteria our query should search for

- We don't want customers who live in our town, Raleigh, so we want a criteria that will return all records **except for** those with **Raleigh** in the city field.
- We don't want customers who live too far away, either. All of the phone numbers in the area start with the 919 area code, so we'll also include a criteria that will only return records whose entries from the **phone number field** begin with **919.** This should guarantee that we'll only send coupons to customers who live close enough to actually come back and use them.
- We won't set a criteria for the order ID field or any other fields because we want to see **all** of the orders made by people who meet the two criteria we just set.

Planning our query

Step 4: Determining the criteria our query should search for

Criteria the query should use to find customer records:

- *No one living in our town, **Raleigh***
  - *In the **City** field, type* **Not in ("Raleigh")**
- *Only customers with phone numbers that start with "**919**"*

  (So we only get customers who live nearby)
  - *In the **Phone Number** field, type* **Like ("919*")**

Query criteria reference guide

Simple criteria for all data types:

	Criteria Name	Write it like...	Function
1	Equals	"x"	Searches for values equal to x
2	Does Not Equal	Not in ("x")	Searches for all values except those equal to x
3	Null	Is Null	Searches for empty fields
4	Not Null	Is Not Null	Searches for non-empty fields

Query criteria reference guide

Simple criteria for text:

	Criteria Name	Write it like...	Function
1	Contains	Like "*x*"	Searches for all values that contain x
2	Does Not Contain	Not like "*x*"	Searches for all values except those that contain x
3	Begins With	Like "x*"	Searches for all values beginning with x
4	Ends With	Like "*x"	Searches for all values ending with x
5	Comes After	>= "x"	Searches for all values that come after x in alphabetical order
6	Comes Before	<= "x"	Searches for all values that come before x in alphabetical order

Query criteria reference guide

Simple criteria for numbers:

	Criteria Name	Write it like...	Function
1	Between	Between "x" and "y"	Searches for values in the range between x and y
2	Less Than	< x	Searches for all values smaller than x
3	Less Than or Equal To	<= x	Searches for all values smaller than or equal to x
4	Greater Than	> x	Searches for all values larger than x
5	Greater Than or Equal To	>= x	Searches for all values larger than or equal to x

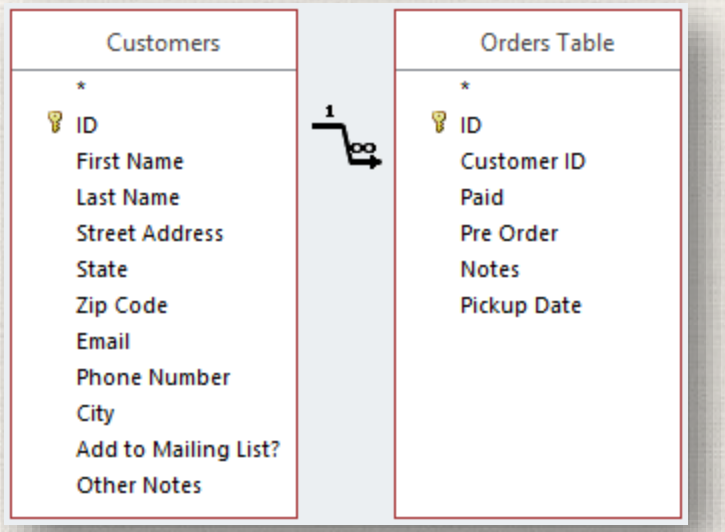
Query criteria reference guide

Simple criteria for dates:

	Criteria Name	Write it like...	Function
1	Between	Between "#mm/dd/yyyy#" and "#mm/dd/yyyy#"	Searches for dates that fall between two dates
2	Before	<#mm/dd/yyyy#	Searches for dates before a certain date
3	After	>#mm/dd/yyyy#	Searches for dates after a certain date
4	Today	=Date()	Searches for all records containing today's date
5	Days Before Today	<=Date()-x	Searches for all records containing dates x or more days in the past

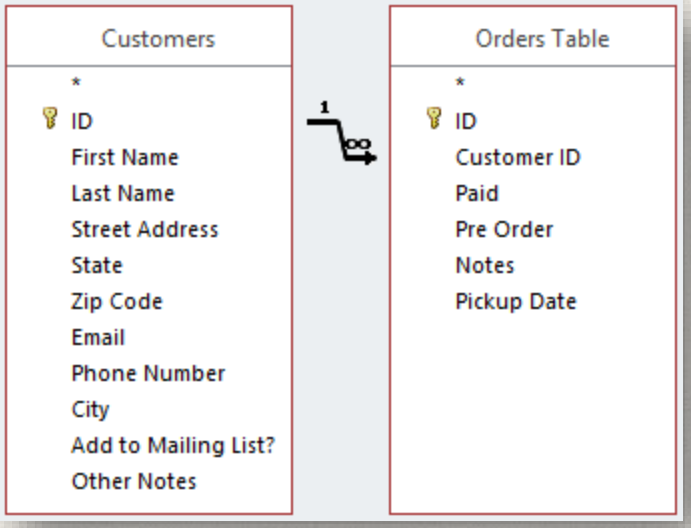
Joining tables in queries

- The final thing you need to consider when designing a query is the way you link—or **join**—the tables you're working with. When you add two tables to an Access query, this is what you'll see in the **Object Relationship pane**:



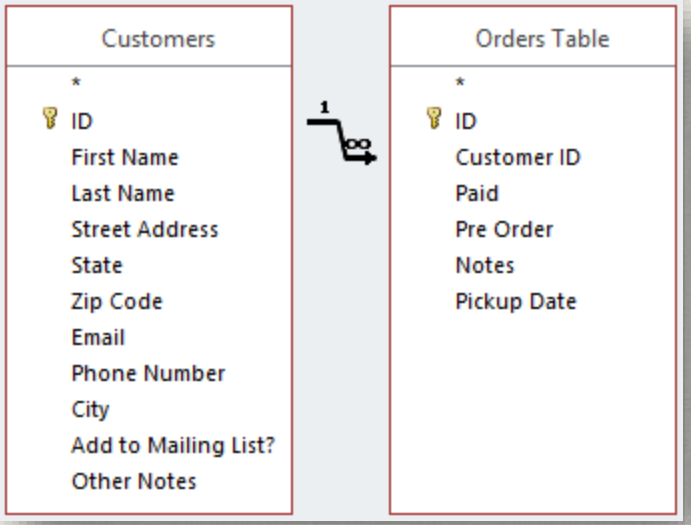
Joining tables in queries

- The line connecting the two tables is called the **join line**. See how the join line is actually an arrow? This is because it indicates the order in which the query looks at data from the two tables.
- In the image at the left, the arrow is pointing from **left** to **right**, which means the query will look at data in the **left** table first, then look at only the data in the **right** table that **relates** to the records it's already seen in the left table.



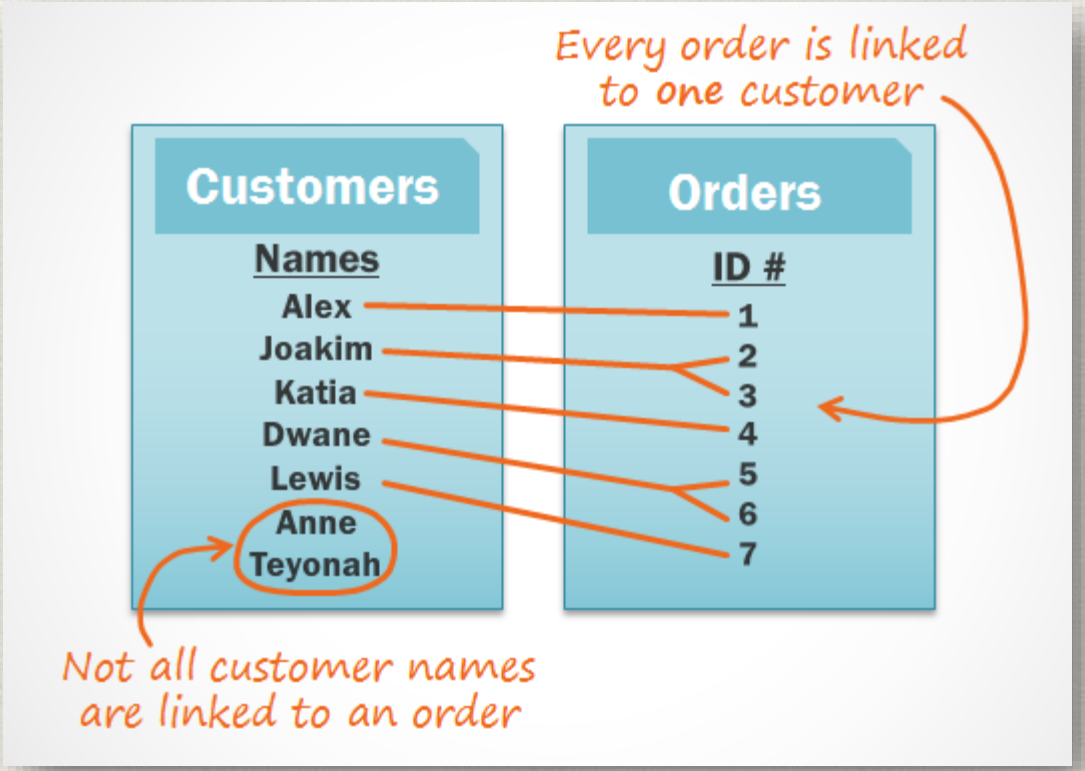
Joining tables in queries

- Your tables won't always be joined this way.
- Sometimes Access will join them **right** to **left**. In either case, you might need to **change the direction** of the join to make sure your query includes the correct information.
- The join direction can affect **which information** your query **retrieves**.



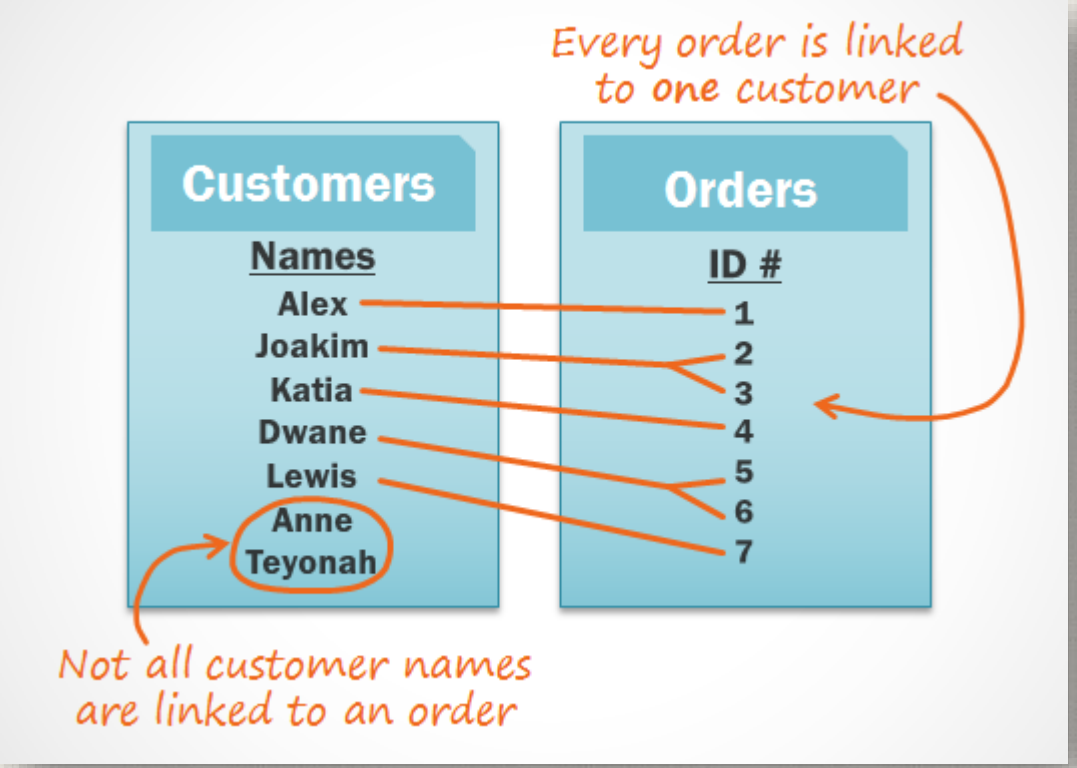
Joining tables in queries

- To understand what this means, consider the query we're designing. For our query, we need to see customers who have placed orders, so we've included the **Customers** table and the **Orders** table. Let's take a look at some of the data contained in these tables.



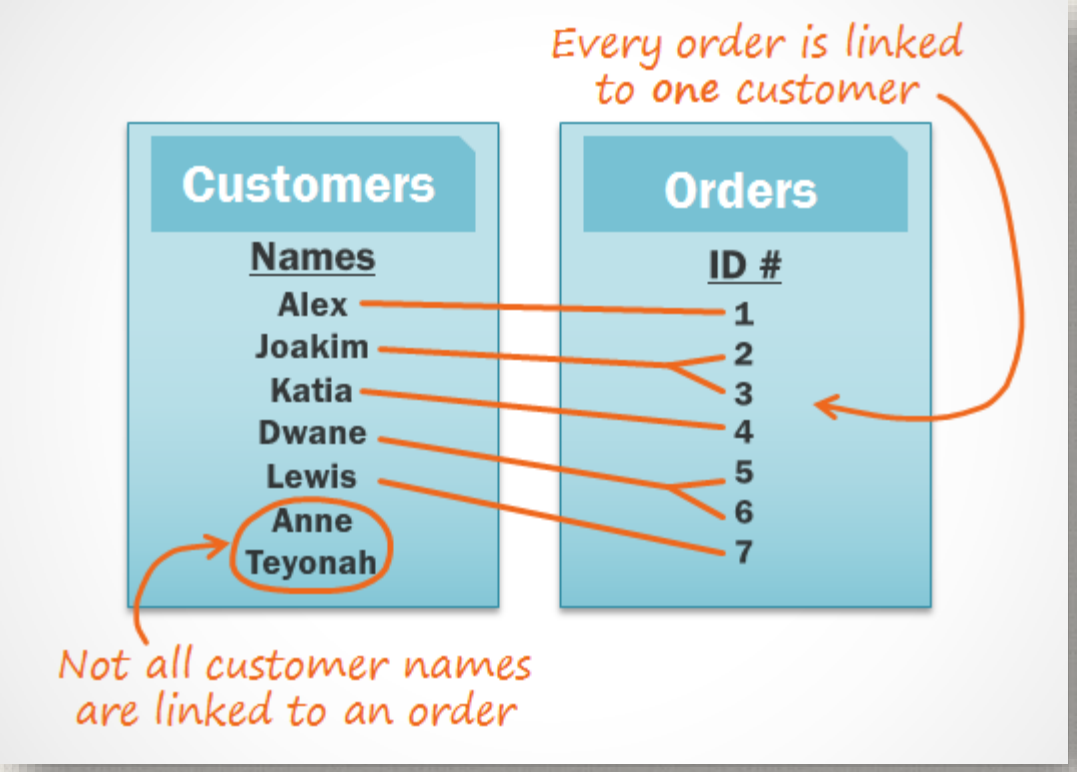
Joining tables in queries

- What do you notice when you look at these lists?
- First of all, every single order in the **Orders** table is linked to someone in the **Customers** table—the customer who placed that order.



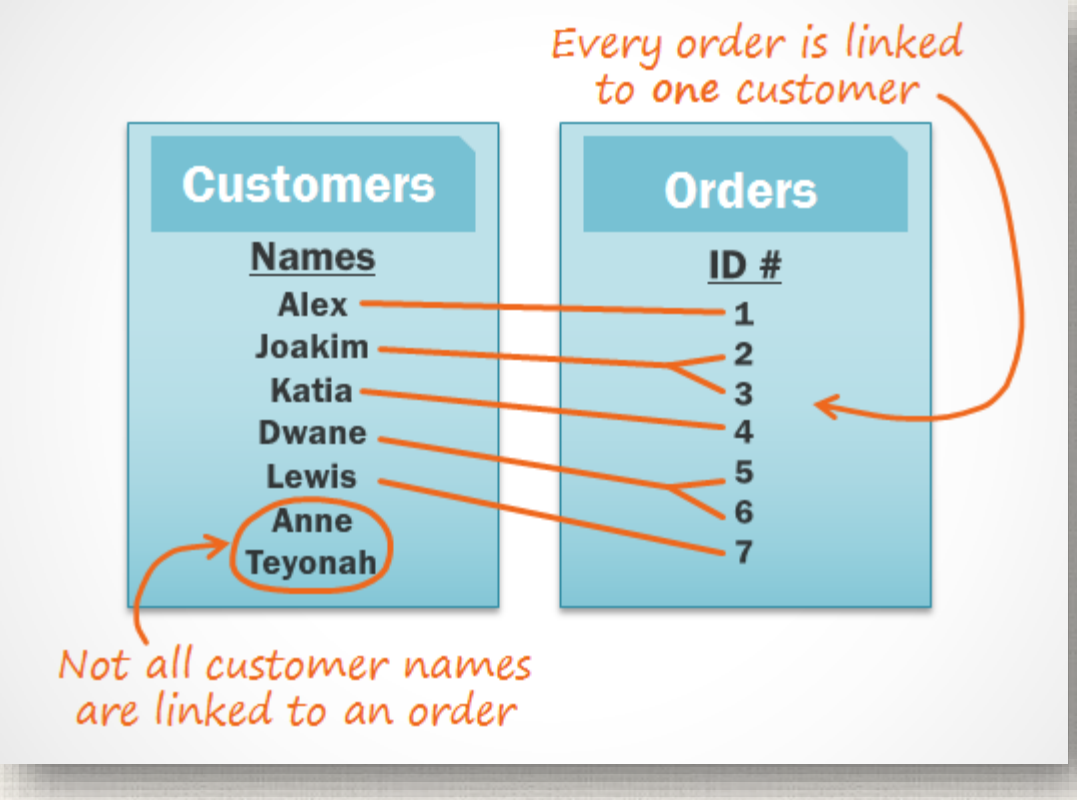
Joining tables in queries

- However, when you look at the Customers table, you'll see that the customers who've placed multiple orders are linked to more than one order, and those who've never placed an order are linked to no orders.



Joining tables in queries

- As you can see, even when two tables are linked it's possible to have records in one table that have no relationship to any record in the other table.

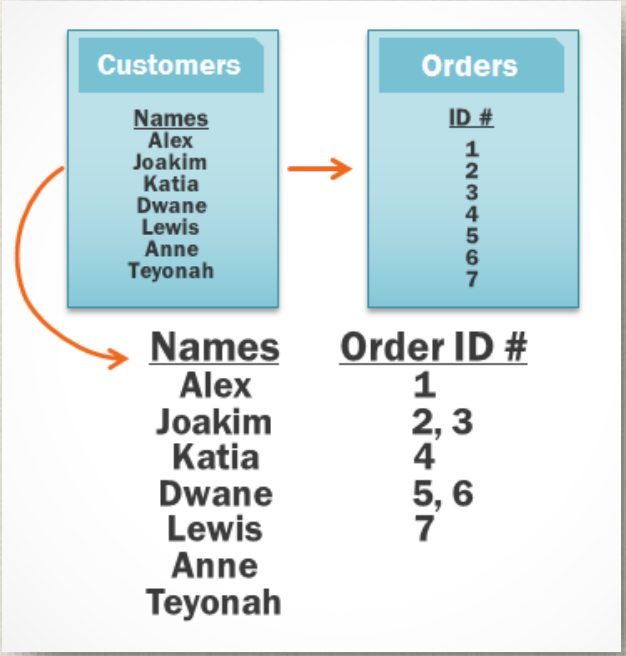


Joining tables in queries

- So what happens when Access tries to run our query with the current join, **left to right**? It pulls every record from the table to the left: our Customers table.

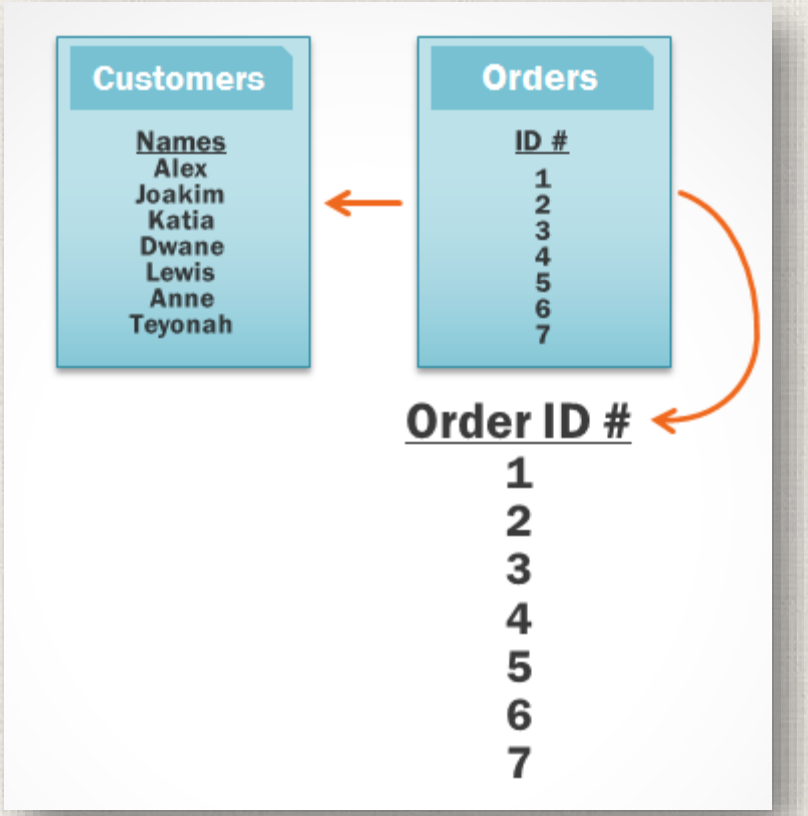
Joining tables in queries

- It then retrieves every record from the **right** table that has a relationship with a record Access has already taken from the left table.
- Because our join began with the **Customers** table, our query will include records for **all** of our customers, including those who've never placed orders. This is more information than we need. We **only** want to see records for **customers who have placed orders**.



Joining tables in queries

- Fortunately, we can fix this problem by changing the direction of the join line. If we join the tables from **right to left** instead, Access will first retrieve the orders from the **right** table, our **Orders** table:



Joining tables in queries

- Then Access will look at the left table and retrieve **only** the records of customers who are linked to an order on the right.

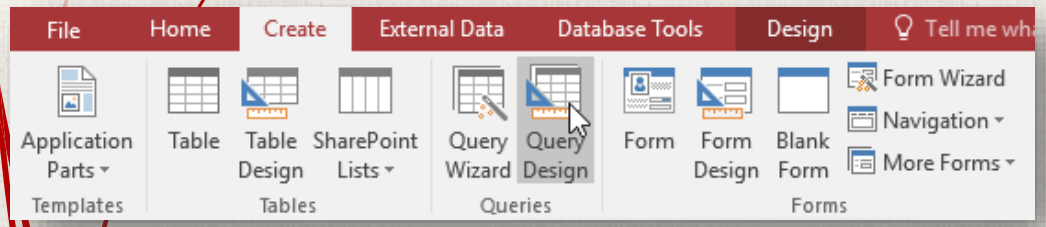
- We now have exactly the information we want: **all** of the customers who have placed an order, and **only** those customers. As you can see, we had to join our tables in the **correct direction** to obtain the information we wanted.
- Now that we understand which join direction we need to use, we're ready to build our query!

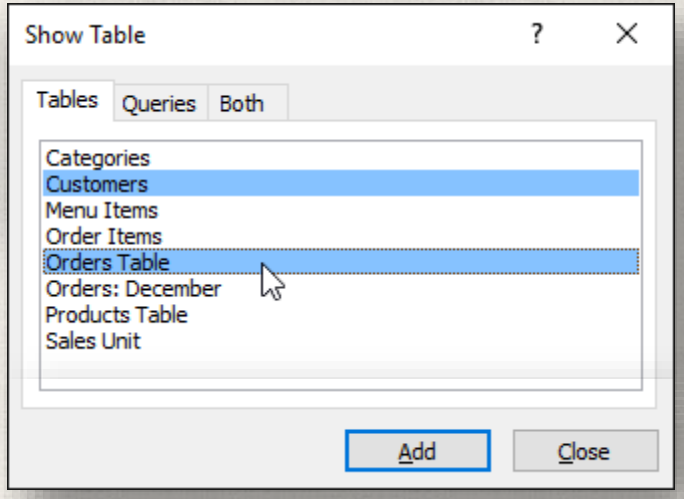
Joining tables in queries

- In our query, we needed to use the **right-to-left** join, but the correct join direction for the tables in your queries will depend on **what** information you want to see and **where** that information is stored. When you add tables to a query, Access will automatically join the tables for you, but it often doesn't join them in the correct direction. This is why it's important to **always review the joins** between your tables before you build a query.

Creating a multi-table query

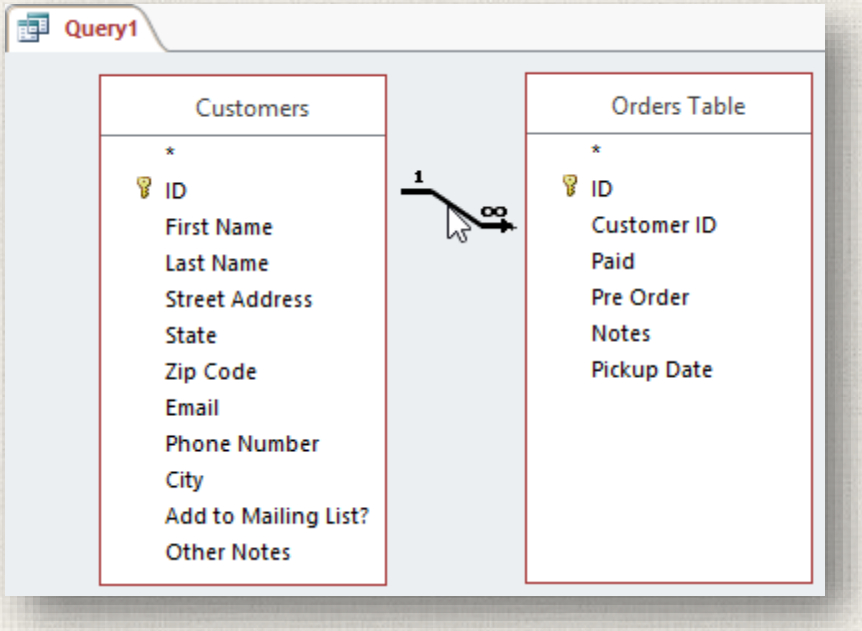
1. Select the **Query Design** command from the **Create** tab on the Ribbon.
2. In the dialog box that appears, select each table you want to include in your query and click **Add**. You can press and hold the **Ctrl** key on your keyboard to select more than one table. When we planned our query, we decided we needed information from the **Customers** and **Orders** tables, so we'll add these.





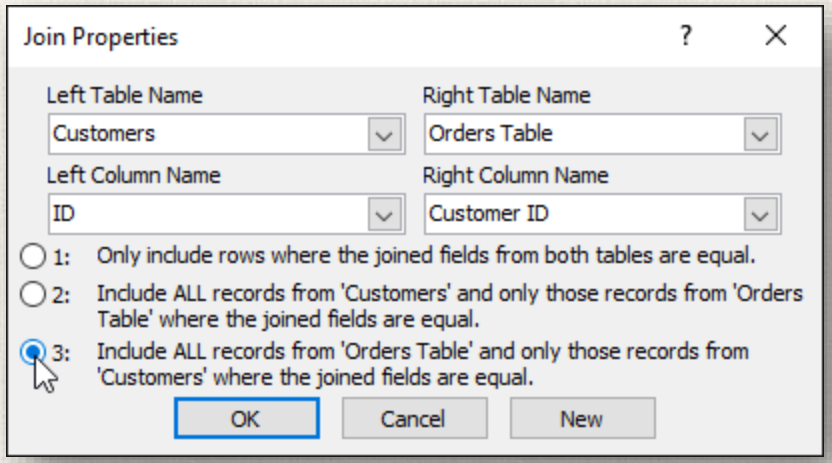
Creating a multi-table query

3. After you have added all of the tables you want, click **Close**.
4. The tables will appear in the **Object Relationship pane**, linked by a **join line**. Double-click the thin section of the join line between two tables to edit its **join direction**.



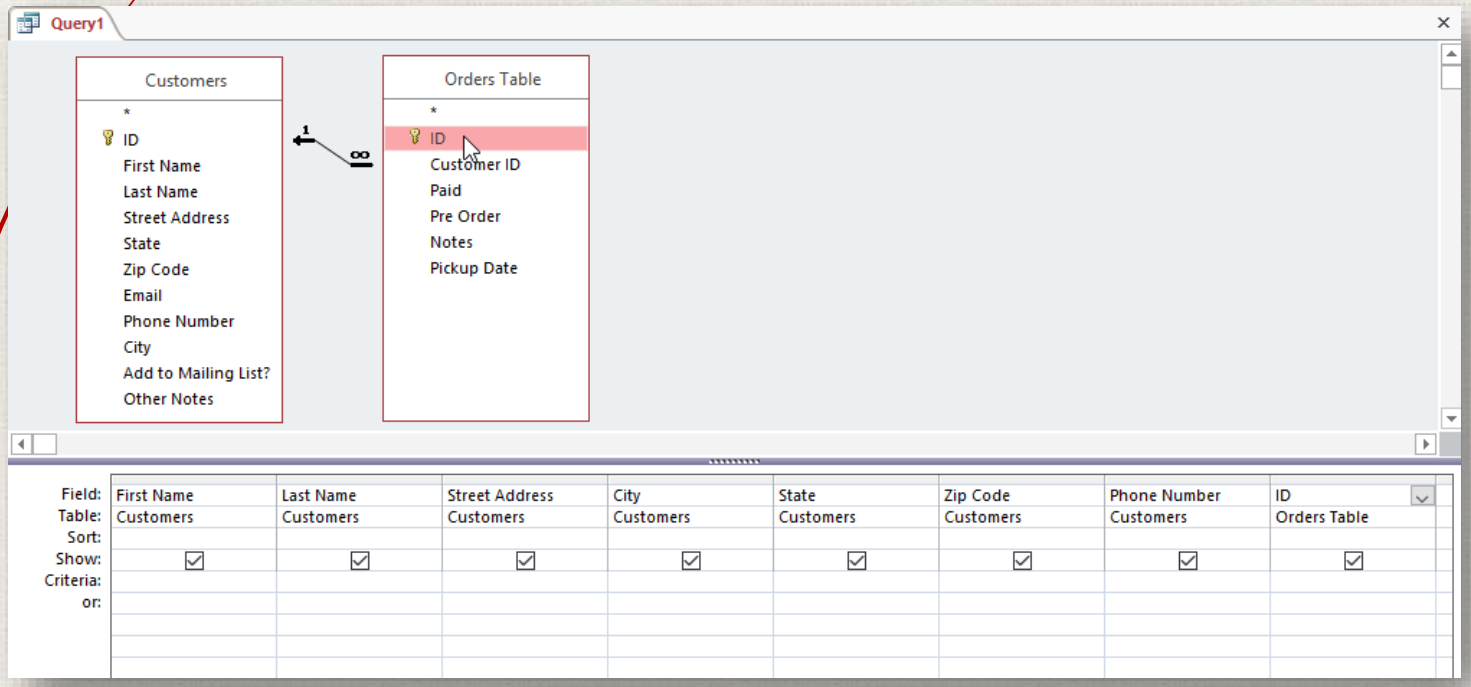
Creating a multi-table query

5. The **Join Properties** dialog box will appear. Select an option to choose the direction of your join. In our example, we'll choose option 3 because we want a right-to-left join.



Creating a multi-table query

6. In the table windows, double-click the **field names** you want to include in your query. They will be added to the **design grid** in the bottom part of the screen. In our example, we'll include most of the fields from the **Customers** table: **First Name**, **Last Name**, **Street Address**, **City**, **State**, **Zip Code**, and **Phone Number**. We'll also include the **ID number** from the **Orders** table.

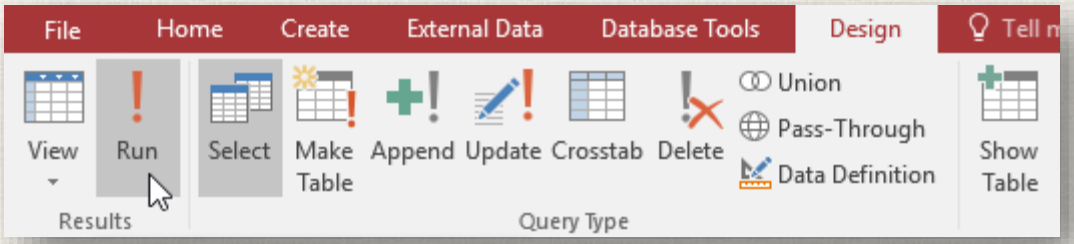


Creating a multi-table query

7. Set field **criteria** by entering the desired criteria in the criteria row of each field. We want to set two criteria: **Not in ("Raleigh")** in the **City** field, and **Like ("919*")** in the **Phone Number** field. This will find customers who do not live in Raleigh but who do live in the 919 area code.

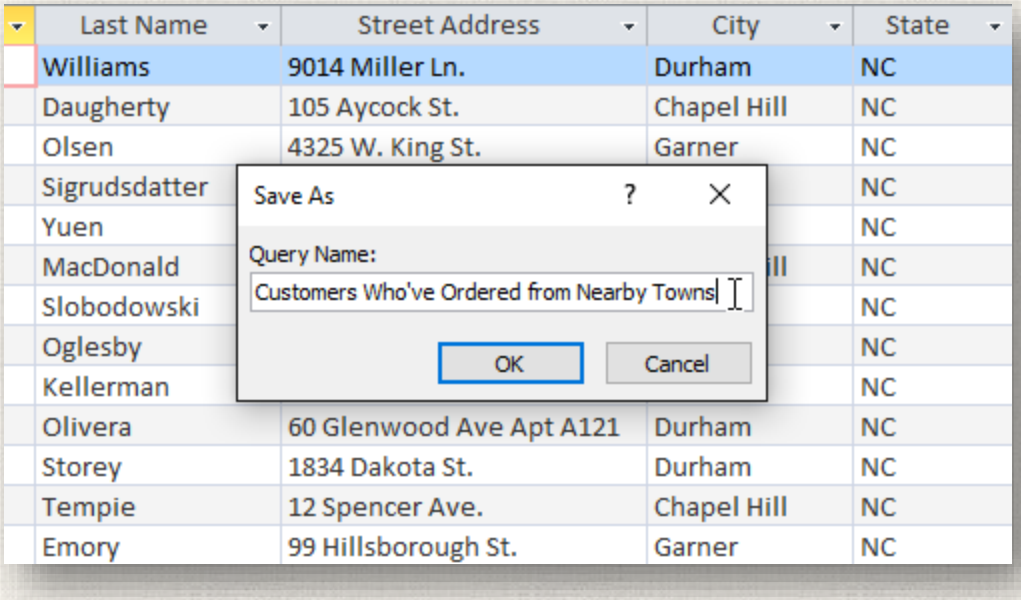
Field:	City	State	Zip Code	Phone Number	ID
Table:	Customers	Customers	Customers	Customers	Orders Table
Sort:					
Show:	☑	☑	☑	☑	☑
Criteria:	Not In ("Raleigh")			Like ("919*")	
or:					

8. After you have set your criteria, **run** the query by clicking the **Run** command on the **Design** tab.



Creating a multi-table query

9. The query results will be displayed in the query's **Datasheet view**, which looks like a table. If you want, **save** your query by clicking the **Save** command in the Quick Access Toolbar. When prompted to name it, type the desired name, then click **OK**.



Practice

- Open **practice database**.
- **Create** a new query.
- Select the **Customers** and **Orders** tables to include in your query.
- Change the **join direction** to **right to left**.
- Add the **First Name**, **Last Name**, and **Zip Code** fields from the **Customers** table to your query.
- Add the **Paid** field from the **Orders Table** to your query.
- Set the following **criteria**: In the **Zip Code** field, type **27609** to return only records with a zip code of **27609**. In the **Paid** field, type **Yes** to return only customers who have paid.
- **Run** the query. If you entered the query correctly, your results will include 20 records of customers who live in the zip code 27609 and have paid for an order. If not, click the **View** drop-down arrow on the Ribbon to return to Design view and check your work.
- **Save** the query with the name **Paying Customers in 27609**.

THE END