

Database Relationships and Queries

Microsoft Access 2013

Instructor: Don Bremer

Workshop Outline for Database Relationships and Queries Microsoft Access 2013

Objectives	3
Tables	3
Relationships	4
Normalization	5
Creating Relationships	8
Editing Relationships	9
Deleting Relationships	9
Relationships Decisions	10
Queries	10
Creating Extended Price – Use Design View	10
Date Ranges – Use Design View	12
Parameterized Query – Use Design View	13
Customers Without Orders – Use Wizard View	13
How much of each product	14
Return only the top X of a query	14
Crosstab Queries	15
Finding Duplicate Records in a query	16
Finding Orphan Records	19
SQL View of Queries	21
Query Decisions	22
Union Queries	22
Pass Through Query	22
Build a Database	23
Create the tables and relationships	23
Create queries over the order data	25
Glossary	29
Microsoft Access 2013 Shortcuts	30

Objectives

Learn relational database terms and concepts and then build a relational database from scratch.

- Create tables and learn database terminology
- Define relationships between tables
- Learn the basics of data model normalization
- Understand some rules of thumb for building a relational database
- Write queries using the design view, SQL view, and query wizard
- Build a relational database from scratch

Tables

A database is built on relationships between tables. Tables store the data in an organized, defined way. Each table holds a unique set of data.

Field Name	Data Type	Description
OrderID	AutoNumber	Unique identifier for this order
CustomerID	Number	Unique identifier for customer placing this order
CarrierID	Number	Unique identifier of shipping company
OrderDate	Date/Time	Date this order received
Gift	Yes/No	Is this a gift?
OrderMethod	Number	Check or charge? (No longer used)
ShipLastName	Text	Last name of person to whom order shipped
ShipFirstName	Text	First name of person to whom order shipped
ShipStreet	Text	Address to which order shipped
ShipCity	Text	City to which order shipped
ShipStateOrProvince	Text	2-letter state abbreviation or complete province name
ShipPostalCode	Text	Postal code to which order shipped
ShipCountry	Text	Country to which order shipped
AccountNumber	Text	Credit card account to which order charged
ExpirationDate	Date/Time	Credit card expiration date
CreditCard	Number	Credit card type

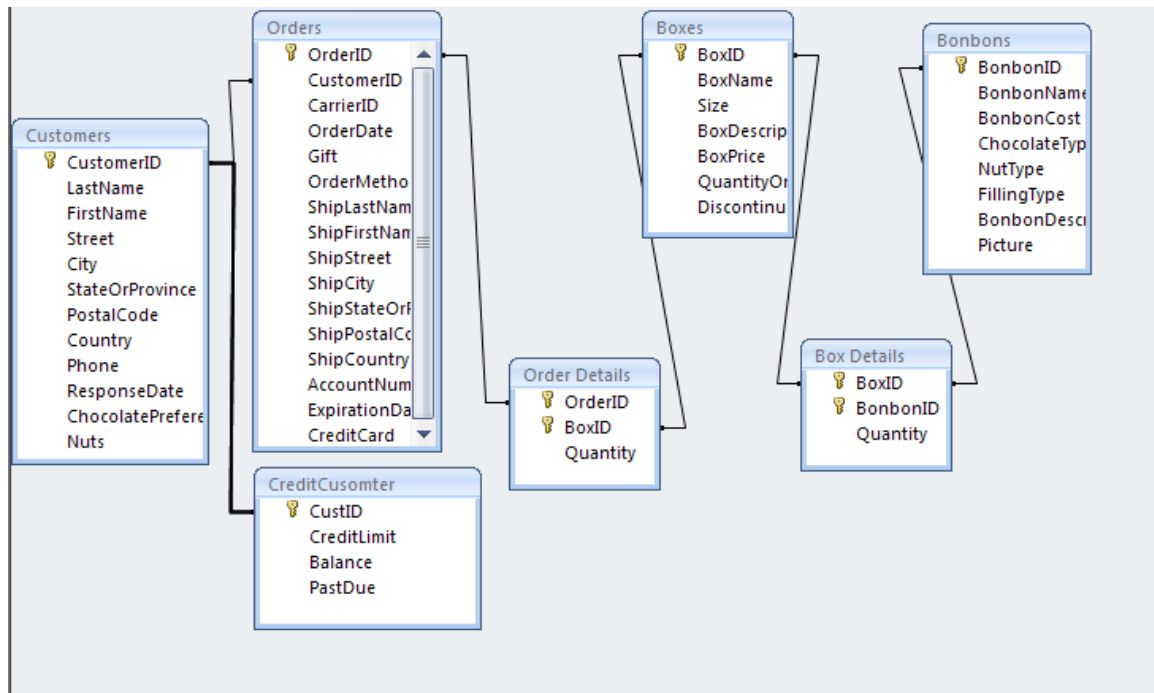
Field Properties	
General	Lookup
Field Size	Long Integer
New Values	Increment
Format	
Caption	Order ID
Indexed	Yes (No Duplicates)
Smart Tags	
Text Align	General

- Every table consists of a list of fields also called columns or attributes.

- Data within the table are called records or rows.
- Data Type determines whether the field is a text or number or date. Keep the data type as small as needed.
- Tables have a unique primary key. Every table has a unique index over the primary key.
- A composite key is where the primary key consists of multiple fields.
- Foreign Key is where the primary key of another table is also found on the current table as a foreign key. Example: Customer ID is the primary key on the customer table. Customer ID is a foreign key on the Orders Table. The sender of the foreign key (Customer) is called the parent table. The receiver (Orders) of the foreign key is called the child table. Think of the primary key flying from 1 to infinity. “To infinity and beyond!”
- There should be a relationship between two tables for every foreign key. Example: Customer to Order relationship.

Relationships

How Tables are Related: Understanding Relationships. Relationships allow one to correlate unique information in one table with related information from another table that contains a unique set of data.



Relationships link two tables by specifying a common field that appears in both tables. Three types of relationships:

- One-to-One

In a one-to-one relationship, each record in Table A can have only one matching record in Table B. Each record in Table B can have only one matching record in Table A. You

might use a one-to-one relationship to divide a table with many fields, to isolate part of a table for security reasons, or to store optional information that applies only to a subset of the main table. For example, you might want to create a table to store customer shipping data or customer credit data separately.

- **One-to-Many (Most Common)**

One record in Table A can be related to many records in Table B in a one-to-many relationship. For example: each customer can have many orders, but each order belongs to only one customer.

- **Many-to-Many** (these are handled by building intersecting tables or junction tables)
A record in Table A can be related to many records in Table B, and a record in Table B can be simultaneously related to many records in Table A. To establish a many-to-many relationship, you need to create a third (junction or intersection) table and add the primary key fields from each of the other two tables to this table. For example, if you have a Products table and an Orders table, an order can include more than one product, and a product can be included in more than one order. You must create an Orders Detail intersecting table that has a one-to-many relationship with the Products table and a one-to-many relationship with the Orders table.

Referential integrity: A foreign key is a field in one table whose values are required to match the primary key of another table. For example: An order can only be created for an existing customer.

Cascading delete: It is an option for relationships that enforce referential integrity between tables. When a record is deleted from the primary table – all related records from the secondary table will be deleted. For example: Deleting an order will automatically delete all order details.
--

Cascading update: When the primary key of the parent table is changed, the value of that field will also be changed in the related table. Cascading update is rarely used.
--

Join Type: Determines whether only matching data in two tables are included or whether all data in one table is included regardless of the data in the other table. Join 1 (inner join) requires a match in both tables. Join Types 2 and 3 (outer join) means that all rows from the primary table will be included even if there is no corresponding match from the secondary table.
--

Normalization

Data Normalization is the process of deciding which fields go in which table.

- Every table contains unique kinds of information.
- The orders table contains only information about orders. The customer table contains only information about customers.
- Every table uniquely identifies each row of data.
- A normalized database contains no duplicates and nothing missing.
- A meaningful report of data contains information from multiple related tables.

Levels of Normalization

- Unnormalized means that multiple kinds of data are in one table.
- 1st Normal Form. Separate repeating groups into a new table
- 2nd Normal Form. Separate fields not dependent on the entire primary key to a new table.
- 3rd Normal Form. Separate fields dependent on another non-key to a new table.
- “The key, the whole key, and nothing but the key.”

Example of unnormalized database:

Orders table

Field Name	Data Type	Description
OrderID	Number	Orders Primary key
OrderDate	Date/Time	Date Order was placed
CustomerLastName	Text	Customer Last Name
CustomerFirstName	Text	Customer First Name
ShiptoStreet	Text	Customer Ship To Street Address
ShiptoCity	Text	Customer Ship to City Name
ShiptoState	Text	Customer Ship To State Code
ShiptoZip	Number	Customer Ship To Zip Code
Product1Desc	Text	Order line item 1 Product Description
Product1Price	Currency	Order line item 1 Product price
Product1Qty	Number	Order line item 1 Qty
Product1Weight	Number	Order line item 1 Weight
Product2Desc	Text	Order line item 2 Product Description
Product2Price	Currency	Order line item 2 Product price
Product2Qty	Number	Order line item 2 Qty
Product2Weight	Number	Order line item 2 Weight
Product3Desc	Text	Order line item 3 Product Description
Product3Price	Currency	Order line item 3 Product price
Product3Qty	Number	Order line item 3 Qty
Product3Weight	Number	Order line item 3 Weight
Product4Desc	Text	Order line item 4 Product Description
Product4Price	Currency	Order line item 4 Product price
Product4Qty	Number	Order line item 4 Qty
Product4Weight	Number	Order line item 4 Weight

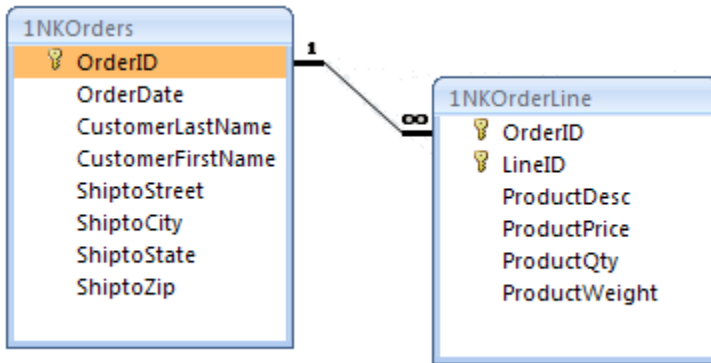
Field Properties	
General	Lookup
Field Size	Long Integer
Format	
Decimal Places	0
Input Mask	
Caption	
Default Value	0
Validation Rule	
Validation Text	
Required	Yes
Indexed	Yes (No Duplicates)
Smart Tags	
Text Align	General

Can you see the limitations of an unnormalized table?

- This kind of a table means that an order can contain only four items. If an order comes in for more items, it must be split into two orders.
- Also, if a customer gets married and changes their last name, the change must be made in every order for that customer instead of changing it only one time.

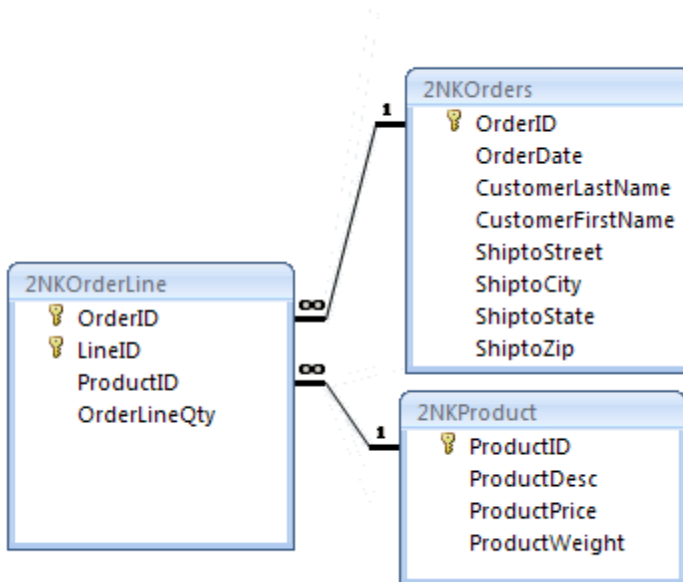
First Normal Form example: Separate repeating groups to another table.

- Move the Product description, price, qty, and weight into the Order Line table.
- Add a second Primary key field on LineID to the Order Line table.



Second Normal Form example: Separate fields not dependent on the entire primary key to a new table.

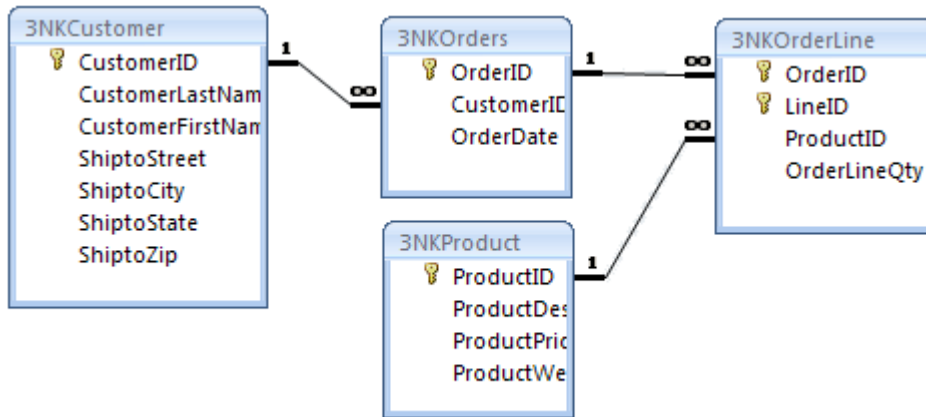
- Move the Product description, price and weight into the Product table because they depend just on the product, not on the specific order of a product.
- Add the Product ID to the OrderLine table as a foreign key.



Third Normal Form example: Separate fields dependent on another non-key to a new table.

- The ship to address information depends on the customer not the order.

- Move the customer last name, customer first name and the ship to address information to the customer table because the name and address of the customer is the same for all of that customer’s orders.



Normalization is a common practice when a business need grows from a spreadsheet solution to a relational database solution. The normalization process is a way to ensure that all of the data is accounted for with nothing duplicated in the most efficient way. Many people practice normalization without knowing the name. It is just the natural way of organizing data.

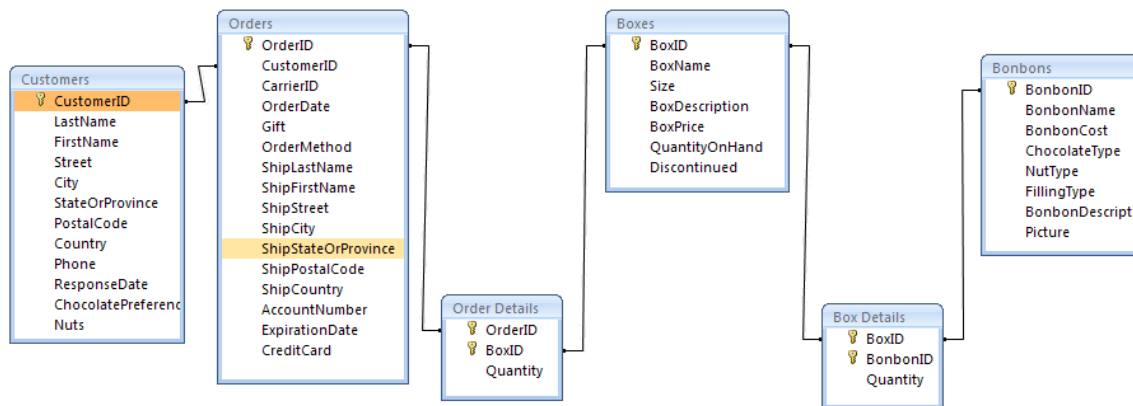
OPEN Database *practice1SWEETLIL*

Creating Relationships

- Show existing relationships
- Tools>Relationships or select the “Relationships” toolbar button
- Relationships>Show Table Hold the shift key down and select the last table in the list and click on “Add” or double click on a table.
- Click on close to put “Show Table” away
- Create the following relationships (in the table below)
 - Move the tables to a more easily accessible area if necessary
 - From the table below and on the next page, select the indicated field
 - While holding the mouse down, drag to the second table and release on the indicated field
 - Make necessary changes
 - Close Relationships and Click on OK to save changes

Tables	Fields	Type of relationship	Others
Customers to Orders	CustomerID to CustomerID	One to Many	“Enforce Referential Integrity”. Join Type 1. Do not Cascade update or delete.
Orders to Order	OrderID to OrderID	One to	“Enforce Referential

Tables	Fields	Type of relationship	Others
Details		Many	Integrity”. Join Type 1. Cascade Update and Cascade Delete Related Records.
Boxes to Box Details	BoxID to BoxID	One to Many	“Enforce Referential Integrity”. Join Type 1. Do not Cascade update or delete
Bonbons to Box Details	BonbonID to BonbonID	One to Many	“Enforce Referential Integrity” and “Cascade Delete Related Records” Join Type 1.
Boxes to Order Details	BoxID to Box ID	One to Many	“Enforce Referential Integrity”. Join Type 1. Do not Cascade update or delete



Editing Relationships

- Show existing relationships
- Tools>Relationships or select the “Relationships” toolbar button
- Edit a relationship by double left clicking.
- Or Edit a relationship by right clicking on it, then choose Edit Relationship.
- Or Edit a relationship by single left clicking on it then Relationships>Edit Relationship.
- In the Orders to Order Details relationship, Change the Cascade Update to not checked. Click OK to save.

Deleting Relationships

- Show existing relationships
- Tools>Relationships or select the “Relationships” toolbar button
- Right Click on the Bonbons to Box Details relationship. Delete. Confirm.

- Recreate the Bonbons to Box Details relationship.

Relationships Decisions

When to enforce referential integrity

- Referential integrity is usually enforced.
- Referential integrity is not always used for optional foreign key data. Example: Creating a Price Quote with the Order table when customer Id is not yet created.

When to cascade delete

- Cascade delete within a logical group like orders to order details. If an order is deleted, all order details will be automatically deleted.
- Do not set cascade delete if it crosses logical data boundaries. Do not specify Cascade delete for the customers to orders relationship. This means that before deleting a customer, all orders must be deleted.

When to cascade update

- Generally not needed because design should prevent primary key updates. Cascade update if a foreign key changes and the change should be reflected in the related tables. Example: product ID changes and the sales history of the product should be changed to reflect the new product.

How to set the join type

- The join type between tables is usually set to 1. (Inner Join).
- Join types 2 and 3 are called outer joins. An outer join means that every row of the parent table will be displayed regardless of whether there is data in the child table.

Queries

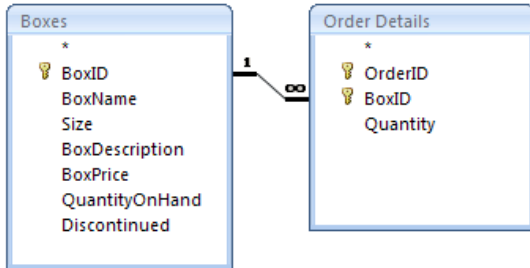
Queries are used to retrieve information, combine information from multiple tables, or return the data with computations.

OPEN Database *practice2SWEETLIL*

Creating Extended Price – Use Design View

- Create New Query from Design View
- Choose the following 2 tables: *Boxes* and *Order Detail* (hold Control key to select non-adjacent tables). Add and then close.
- From *Order Detail* choose the following fields:
 - OrderID
 - BoxID
 - Quantity
- From *Boxes* choose the following fields:
 - BoxName
 - Size
 - BoxPrice
- Save as “**Extended Price**”

- In blank column next to BoxPrice, click in field text box, then click on “Build” from toolbar button
- In the dialog box, type: Extended Price: ([Quantity]*[BoxPrice])
- Click OK.
- Save and Run Query



Field:	OrderID	BoxID	Quantity	BoxName	Size	BoxPrice	Extended Price: ([Quantity]*[BoxPrice])
Table:	Order Details	Boxes	Order Details	Boxes	Boxes	Boxes	
Sort:							
Show:	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
Criteria:							
or:							

Looking for something that is nothing

Perhaps we want to look for customers that have no phone numbers. We will send these people postcards saying that if they give us their phone number, we will give them 20% off the next purchase. But, how do we look for something that is *not* there. That is using the **is Null** criteria.

Field:	LastName	FirstName	Street	City	StateOrProvince	PostalCode	Country	Phone
Table:	Customers	Customers	Customers	Customers	Customers	Customers	Customers	Customers
Sort:								
Show:	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>
Criteria:								Is Null
or:								

Well, that’s wonderful! But what happens if we want to call anyone who *has* something in there. Just like in English, we can negate this using **Not**. So, the new query will look like this:

Field:	LastName	FirstName	Street	City	StateOrProvince	PostalCode	Country	Phone
Table:	Customers	Customers	Customers	Customers	Customers	Customers	Customers	Customers
Sort:								
Show:	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
Criteria:								Is Not Null
or:								

Finding Matches for a Checkbox

Remember that some of our Customers were “Nuts” and we checked the checkbox to show that? How do we find all of these positive results?

To find a positive result in a checkbox, we would simply use “True”

The screenshot shows a query design grid with the following fields: FirstName, LastName, Street, and Nuts. All fields are selected (checked). The criteria for the Nuts field is set to 'True'.

Field:	FirstName	LastName	Street	Nuts
Table:	Customers	Customers	Customers	Customers
Sort:				
Show:	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
Criteria:				True
or:				

First Nam	Last Name	Street	Nuts
Dale	Carter	14 S. Elm Dr.	Yes
Walt	Jefferson	23 Tsawassen Blvd.	Yes
M.	Morales	1001 West Pender	Yes
Roberta	MacIntyre	72233 Bayou Dr.	Yes
Frank	Silverman	897 N. Northern Ave.	Yes
Carol	James	8997 Cobol Ave.	Yes
Ali	Zahoor	597 West End Ave.	Yes
Hope	Bingsley	2743 Bering St.	Yes
R.A.	Hiatt	2226 Shattuck Ave.	Yes
Leslie	Gilman	11 Eastern Ave.	Yes
V. J.	Bernstein	507 20th Ave. E.	Yes
Philomae	Carruthers	342 Lake Crescent S.W	Yes
Henry	Malone	9 Royal Rd.	Yes
Barry	Kurtz	135 Cleveland Ave.	Yes
Janice	Leverling	12-A N. 126th St.	Yes
Bill	Moran	82 Main St.	Yes
Nina	Valerio	14 N. Paterson St.	Yes
Joseph	Albert	1041, rue Brunel	Yes

Likewise, we can find people who don't match by using the “False” Criteria.

Date Ranges – Use Design View

- Create New Query from Design View
- Choose the following 4 tables: *Customers, Orders, Order Details & Boxes*
 - Add the following to build the “Sales by Customers” query:
 - 1st column: Field: *BoxName* Table: *Boxes*
 - 2nd column: Field: *Country* Table: *Customers*
 - 3rd column: Field: type in: Customer: *[LastName]&”, ”&[FirstName]*
 - 4th column: Field: type in: Number Of Boxes: *Quantity* Tables: *Orders Detail*
 - 5th column: Field: *OrderDate* Table: *Orders*
 - Criteria: type in: *Between #12/6/98# And #12/12/98#*
 - Or: Build
 - Save as “**Sales by Customer**”

Field:	BoxName	Country	Customer: [LastName] & ", " & [FirstName]	Number of Boxes : Quantity	OrderDate
Table:	Boxes	Customers		Order Details	Orders
Sort:					
Show:	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
Criteria:					Between #12/6/1998# And #12/12/1998#
or:					

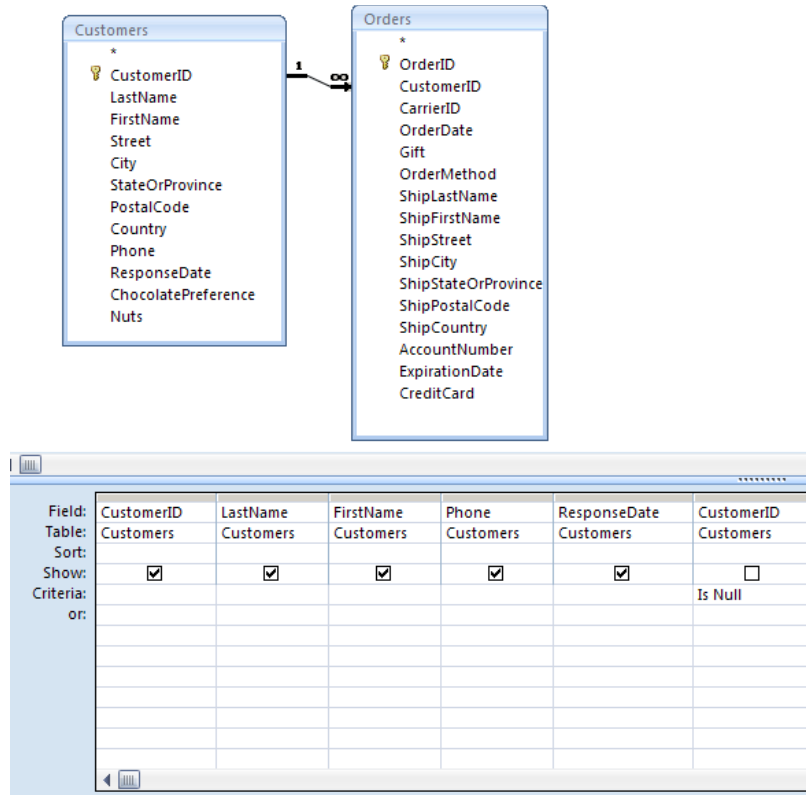
Parameterized Query – Use Design View

- Modify Sales by Customer to have date values as parameters
 - Save as “Sales by Customer Date Parm”
 - In the Criteria field for Order Date, replace the criteria with Between [Begin date:] And [End date:] The [] means Field. If the field name does not exist, the query will prompt for the values.
 - Save the Query and Run it by clicking on the Red exclamation point. Or, click Open in the Query Object View. Set the Begin Date to 12/6/98. Set the End Date to 12/12/98.

Field:	BoxName	Country	Customer: [LastName] & ", " & [FirstName]	Number of Boxes : Quantity	OrderDate
Table:	Boxes	Customers		Order Details	Orders
Sort:					
Show:	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
Criteria:					Between [Begin Date:] And [The End:]
or:					

Customers Without Orders – Use Wizard View

- Create New Query from Unmatched Query Wizard
- Select Customers to search for unmatched data.
- Select Orders as the related table
- Pick the fields CustomerID and CustomerID from both tables
- Pick the following fields to see in the query results:
 - CustomerID
 - LastName
 - FirstName
 - Phone
 - ResponseDate
- Name the query “Customers Without Orders”. View the Results

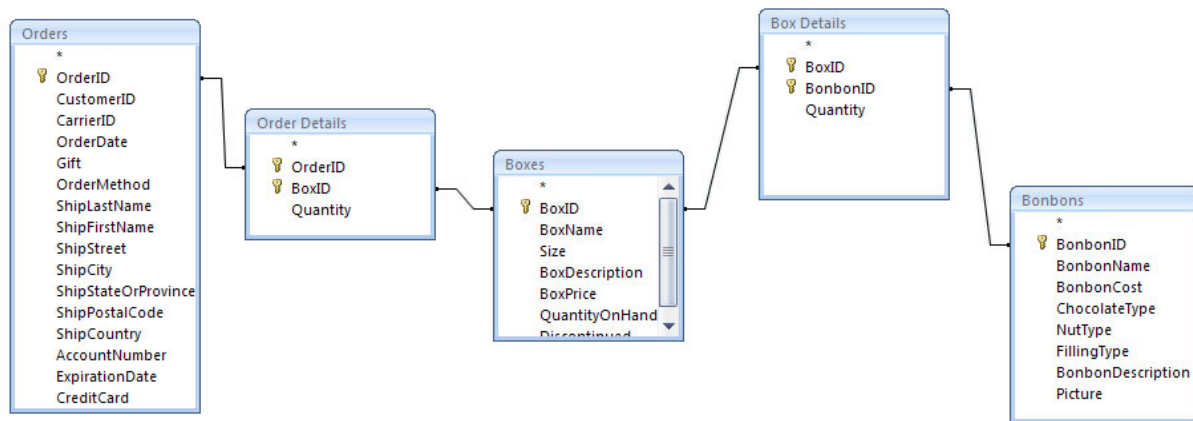


How much of each product

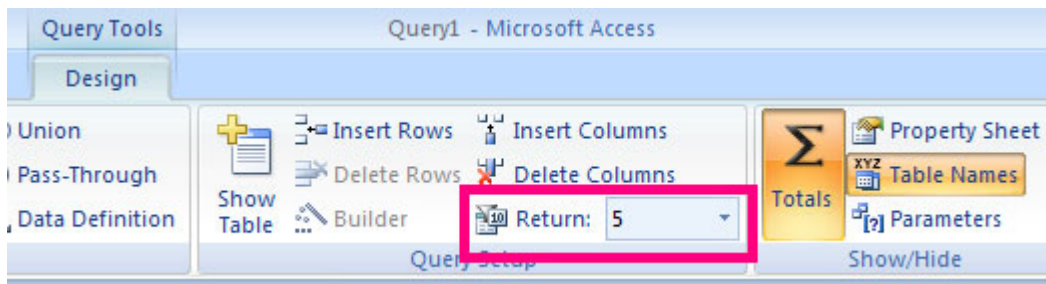
- Create New Query from Design View
- Choose the following 5 tables: *Orders, Order Details, Boxes, Box Details & Bonbons*
- Add the following fields:
 - BonbonName from Bonbons
 - Quantity from Box Details
 - OrderID from Orders
- Click on the totals button from the ribbon
 - Group by BonbonName
 - Sum by Quantity
 - Count by OrderID

Return only the top X of a query

- With the above query, we can get over 35 items. Let’s just get the top 5.
- Copy the Quantity Field and place the copy to the far right
- Sort Descending
- Don’t show that field
- Under design, Return 5 records



Field:	Quantity	BonbonName	Quantity	OrderID				
Table:	Box Details	Bonbons	Box Details	Orders				
Total:	Sum	Group By	Sum	Count				
Sort:	Descending							
Show:	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Criteria:								
or:								



Crosstab Queries

Crosstab Queries are a lot like Pivot tables in Excel. This might be handy in finding out how many of each bonbon went to what countries. To create one requires that we create another query before:

- Create a Query in Design View
- Add the Tables:
 - Customers
 - Orders
 - Order Details
 - Boxes
 - Box Details
 - Bonbons

Select only three fields:

- Country in Customers
- BonbonName in Bonbons

- BonbonID in Bonbons

Save this query as BonbonsByCountry

To Create the Crosstab query:

- Create a Query from Wizard
- Select Crosstab Query Wizard
- Select the Queries Button on items to view and select “BonbonsByCountry”

Crosstab Query Wizard

What number do you want calculated for each column and row intersection?

For example, you could calculate the sum of the field Order Amount for each employee (column) by country and region (row).

Do you want to summarize each row?

Yes, include row sums.

Fields:

BonbonID

Functions:

Count
First
Last
Max
Min

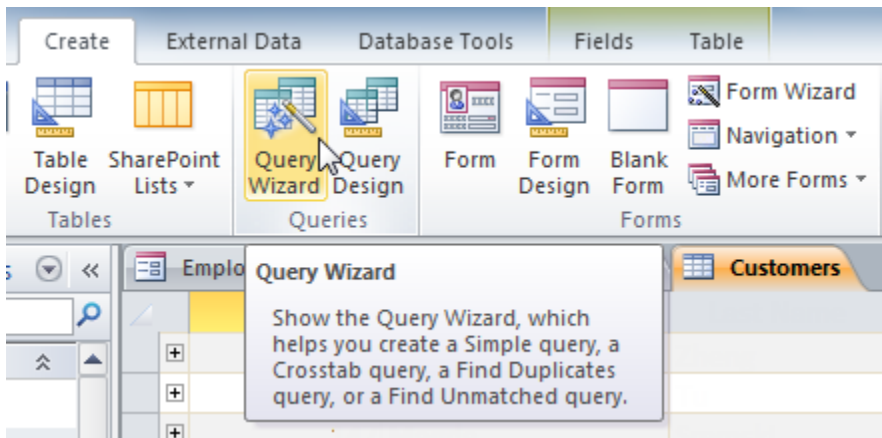
Sample:

BonbonName	Country1	Country2	Country3
BonbonName1	Count(BonbonID)		
BonbonName2			
BonbonName3			
BonbonName4			

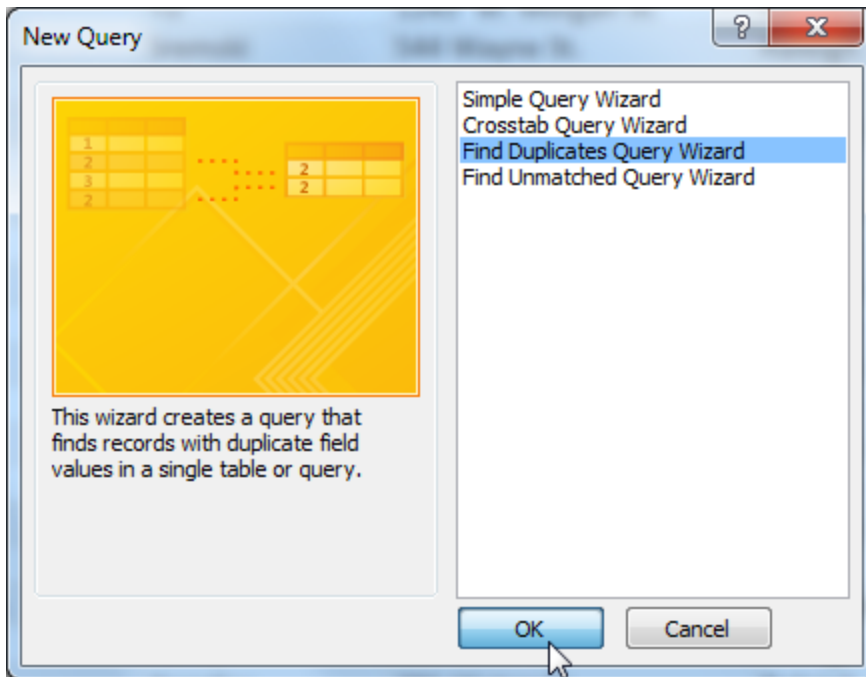
Cancel < Back Next > Finish

Finding Duplicate Records in a query

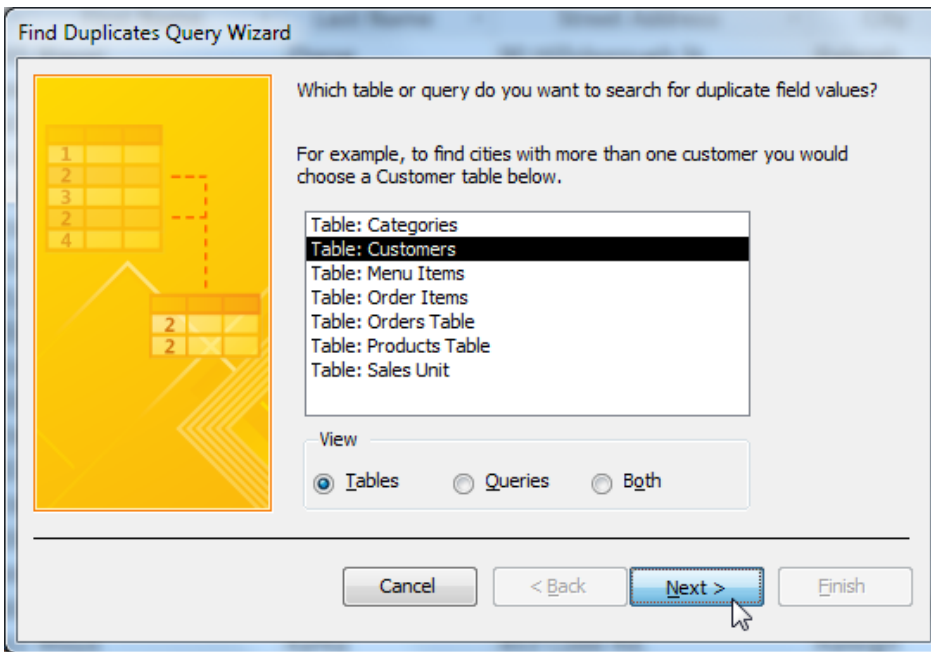
Everything that has been shown so far has worked on *one* record at a time. There is no way to look at multiple records for something like duplicates. For that, we are going to use the query wizard.




The New Query dialog box will appear. Select the Find Duplicates Query from the list of queries and click OK.

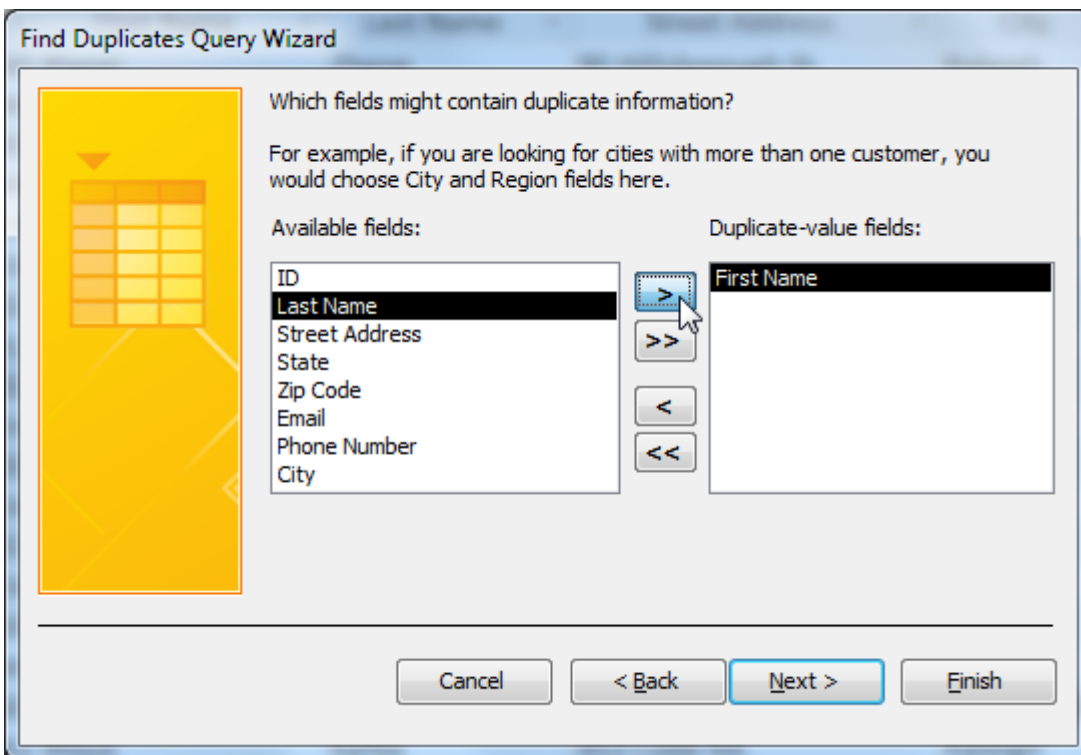


Select the table you want to search for duplicate records and click Next. We're searching for duplicate customer records, so we'll select the Customers table. We may have to create some duplicates. Copy some records and paste them back into the table.



Choose the fields you wish to search for duplicate information by selecting them, then clicking the right arrow button . Only select fields that should not be identical in non-duplicate

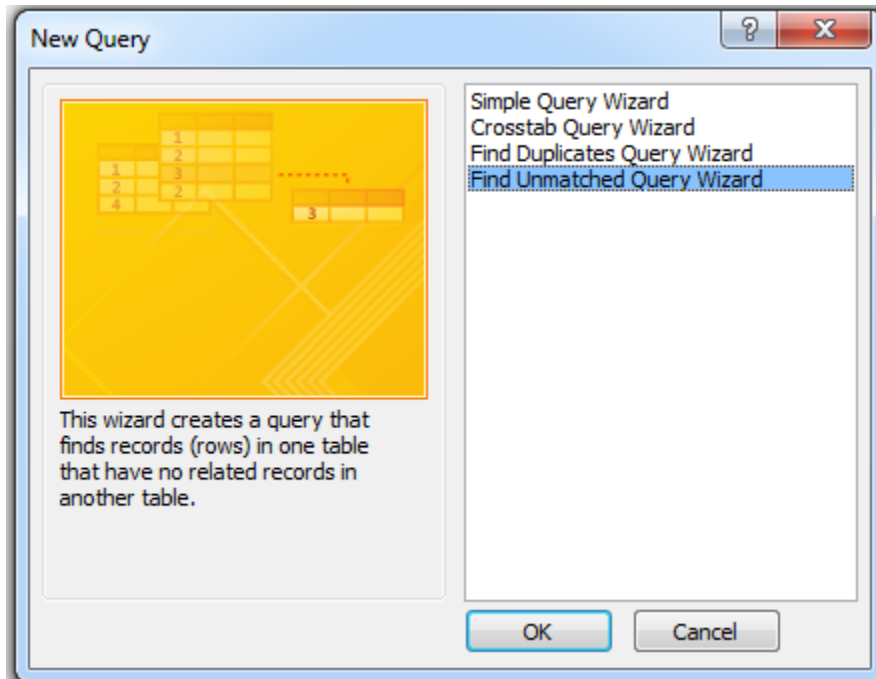
records.



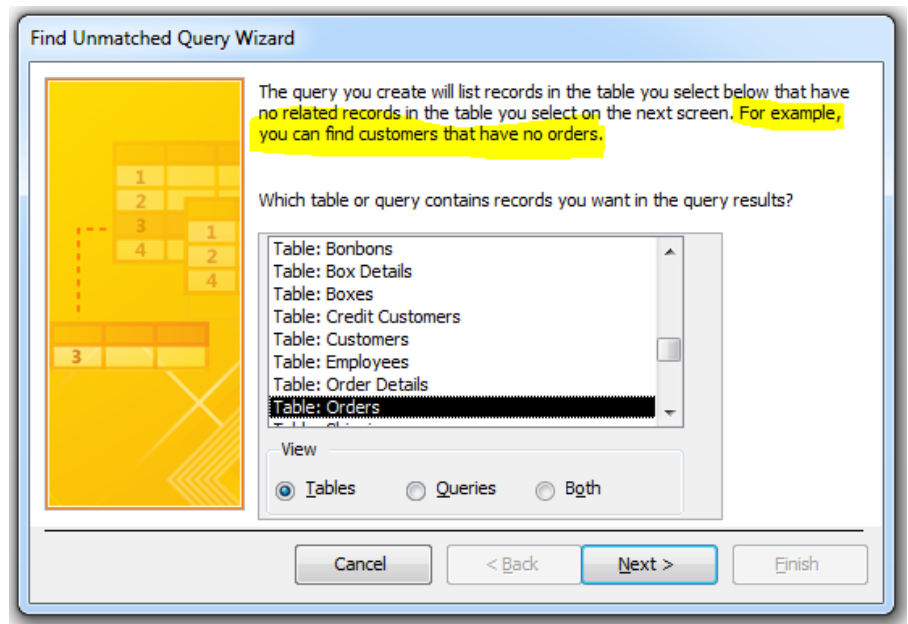
Select any additional fields you would like to see when the query runs and then select Next. The final screen will ask what you want to call the query. It will default to something like: Find duplicates for Customers. You can keep it as this, or select a better name for it.

Finding Orphan Records

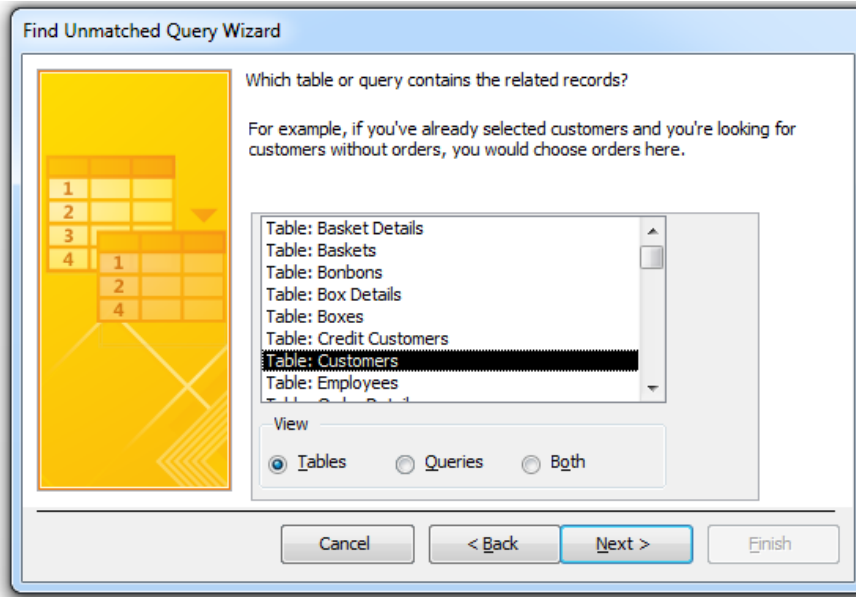
Orphan Records are records in a table that should have a matching record in another table. Take for example Customers and Orders. Every Order *should* have a corresponding Customer in the customer table. But, sometimes, glitches occur and you have Orders with no Customer. To find these, we can use the Query Wizard:



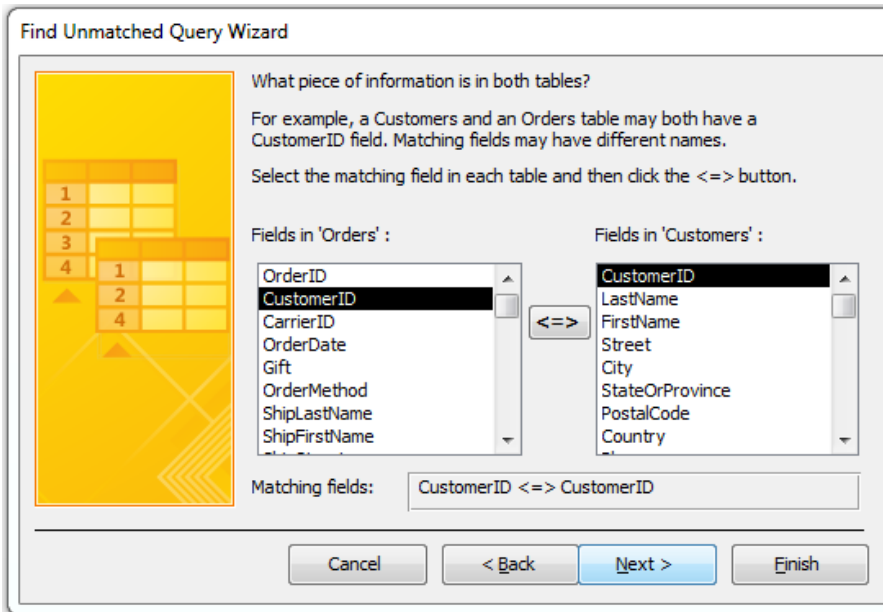
Select “Find Unmatched Query Wizard” and click the OK button. The wizard will walk you through the correct way to set up the table relations:



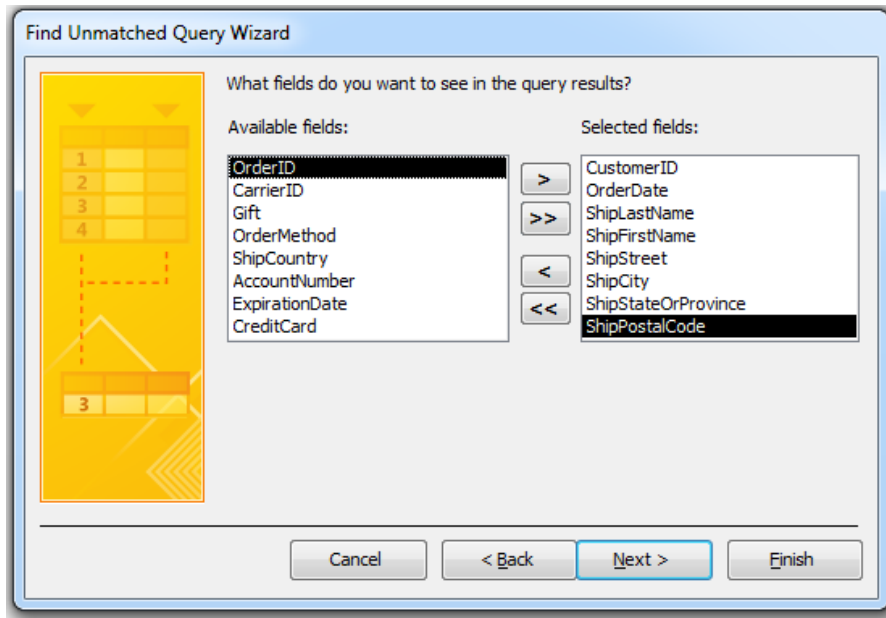
Then, it will ask which table will have the unrelated records:



Since these table don't *have* to be related (they could be two queries), Access will then look for a common field between the two data sets. It can find it if we named the fields the same; we may need to give it some help if we named the fields differently.



The last question is what we would like to see on the query result. Since we are looking for Orders without Customers, perhaps who the previous Customer was and who it was shipped to may help us find out why this order is in limbo.



After that, we just name the query and View the Results!

SQL View of Queries

Look at the SQL View of other queries.

- Go to Design View
- View menu > SQL View
- Look at the SQL View of the Customers Without Orders Query
- SELECT AS
FROM
INNER JOIN ON
WHERE
GROUP BY
ORDER BY
HAVING
- Change the “INNER JOIN” to “LEFT JOIN”. What happens?
- Change the “INNER JOIN” to “RIGHT JOIN”. What happens?

**SELECT Customers.CustomerID, Customers.LastName, Customers.FirstName,
Customers.Phone, Customers.ResponseDate
FROM Customers LEFT JOIN Orders ON Customers.CustomerID = Orders.CustomerID
WHERE (((Customers.CustomerID) Is Null));**

- Look at the SQL View of the Extended Price Query.
 - Change the “LEFT JOIN” to “INNER JOIN”. What happens?
 - Change the “LEFT JOIN” to “RIGHT JOIN”. What happens?

```
SELECT Customers.CustomerID, Customers.LastName, Customers.FirstName,  
Customers.Phone, Customers.ResponseDate  
FROM Customers INNER JOIN Orders ON Customers.CustomerID = Orders.CustomerID  
WHERE (((Customers.CustomerID) Is Null));
```

Query Decisions

- Only use the tables that you need.
- Only use the fields that you need.
- Clear the show box for fields you do not need displayed.
- Leave the sort as default unless there is a need for sorted data.
- Experiment with inner joins versus outer joins.
- Test criteria built on a one-to-many join on both sides of the join.
- When grouping records by the values in a joined field, specify **Group By** for the field that's in the same table as the field you're totaling. If the sum is extended price, group by on orderdetail.orderid not orders.orderid.
- Group by as few fields as needed.

Union Queries

A union query combines the result sets of several similar select queries. For example, suppose that you have one table that stores information about customers, and another table that stores information about the Employees. You'd like to send them all a Holiday Card!

You could create a select query for each table to retrieve only those fields that contain contact information, but the information that is returned would still be in two, separate places. To combine the results of two or more select queries into one result set, you can use a union query

```
SELECT FirstName, LastName, City  
FROM Customers  
UNION ALL  
SELECT FirstName, LastName, DepartmentName As City  
FROM Employees;
```

The name and number of output fields must be the same for both select queries. If the names are not the same, you can use the "AS" operator to change the name and make sure they are the same.

Pass Though Query

SQL pass-through queries are used to send commands directly to an ODBC database server. By using an SQL pass-through query, you work directly with the server tables instead of having the Microsoft Jet database engine process the data.

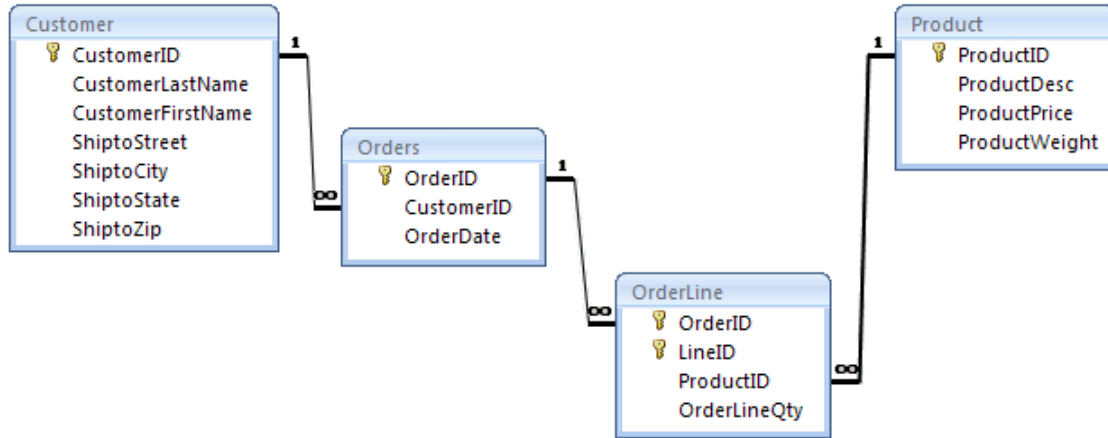
And, that's all I'm going to say about that!

Build a Database

Close the existing database and Create a new database.

Create the tables and relationships

- File > Close
- File > New. Then Choose Blank Database. Call it practicefinal.



- Create the Orders database in the normalization example into 3rd normal form.
- Create the tables with the correct primary keys, foreign keys, and non-key fields.

Table	Design			
Primary Key	Field name	Data type	Length	Notes
Table name:	Customer			
Primary key	CustomerID	Number		Required
	CustomerLastName	Text	30	
	CustomerFirstName	Text	30	
	ShiptoStreet	Text	30	
	ShiptoCity	Text	30	
	ShiptoState	Text	2	
	ShiptoZip	Text	10	
Table name:	Orders			
Primary key	OrderID	Number		Required
	CustomerID	Number		Foreign key to Customer
	OrderDate	Date/Time		
Table	Product			

Table	Design			
Primary Key	Field name	Data type	Length	Notes
Name:				
Primary key	ProductID	Number		Required
	ProductDesc	Text	30	
	ProductPrice	Currency		
	ProductWeight	Number	Integer	
Table Name:	OrderLine			
Primary key	OrderID	Number		Required. Foreign key to Orders. Cascade Delete.
Primary key	LineID	Number		Required
	ProductID	Number		Foreign key to Product.
	OrderLineQty	Number	Integer	

- Create the correct relationships between tables.
- Define the correct referential integrity on all tables. Define cascading delete on Order to OrderLine.
- Enter a row into the OrderLine table. What happens? Why?
 - OrderID = 1, LineID = 1, ProductID = 1, Quantity = 4
- Enter a few rows of data in each table to test for each unique key.
 - What does the error on customer Mel Torme mean?
 - What does the error on Order 6 for customer 5555 mean?
- Enter data into the tables as defined below:

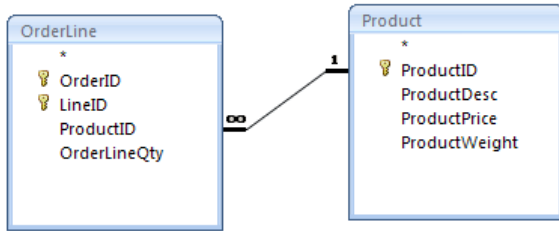
Customer	Data					
CustomerID	CustomerLast Name	Customer FirstName	ShiptoStreet	ShiptoCity	ShiptoState	ShiptoZip
1	Dillon	Robert	514 3 rd St E	Duluth	MN	55802
2	Bopper	Big	1205 London Rd	Duluth	MN	55805
3	Dion	Celine	1200 Tower Ave	Superior	WI	54880
2 (Why is this Error?)	Torme	Mel	11 E Superior St	Duluth	MN	55802
4	Torme	Mel	11 E Superior St	Duluth	MN	55802
Product	Data					
ProductID	ProductDesc	ProductPrice	ProductWeight			
1	Hooked on Bach	13.95	1			
2	Brewhouse	7.95	1			

	Sampler					
3	Star Wars soundtrack	13.95	2			
Orders	Data					
OrderID	CustomerID	OrderDate				
1	3	1/3/2002				
2	3	1/29/2002				
3	3	3/31/2002				
4	1	1/17/2002				
5	2	4/29/2002				
6	5555	4/29/2002		Error		
OrderLine	Data					
OrderID	LineID	ProductID	OrderLineQty			
1	1	2	1			
1	2	3	4			
2	1	3	3			
3	1	2	1			
4	1	2	2			
4	2	3	4			
5	1	3	1			

- Test the deletion of data from the tables
 - Delete Customer = 1 from the Customer table. Customer 1 has an order. What happens? Why?
 - Delete Order = 4 from the Order table. Order 4 has OrderLine. Look in the OrderLine table for Order 4. What happens? Why?
 - Delete Customer = 1 from the Customer table again. Now customer 1 does not have any orders. What happens? Why?
- What order do you have to enter data into tables? Why?

Create queries over the order data

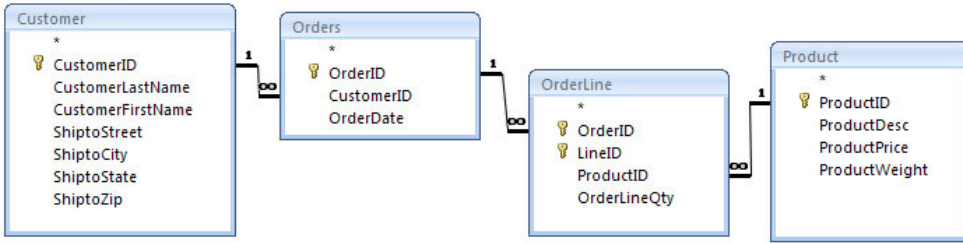
1. Create a query that computes extended price for each order line.
 - Choose the tables OrderLine and Product.
 - The query should include: OrderID, LineID, ProductID, ProductDesc, ProductPrice, OrderLineQty.
 - Include a new field ExtendedPrice: ([OrderLineQty]*[ProductPrice])
 - Save the Extended Price Query.



Field:	OrderID	ProductDesc	OrderLineQty	ProductPrice	ExtendedPrice: ([OrderLineQty]*[ProductPrice])
Table:	OrderLine	Product	OrderLine	Product	
Sort:					
Show:	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
Criteria:					
or:					

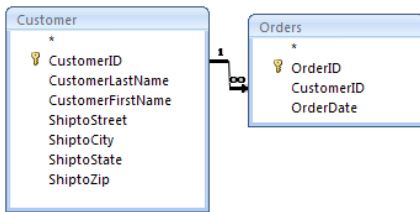
2. Create a query that computes the Order total.
 - Choose the tables Customer, Order, OrderLine, and Product.
 - Pick the following fields:
 - i. OrderID
 - ii. CustomerLastName
 - iii. CustomerFirstName
 - iv. OrderDate
 - v. ShiptoCity
 - vi. ShiptoZip
 - vii. LineID
 - viii. ProductWeight
 - ix. ProductPrice
 - x. OrderLineQty
 - xi. ExtendedPrice: ([OrderLineQty]*[ProductPrice])
 - Group the fields by Order
 - i. Sum the ExtendedPrice to compute OrderTotal.
 - ii. Count the LineID
 - iii. Avg the ProductPrice
 - iv. Sum the ProductWeight and the OrderLineQty
 - v. Group by all other fields
 - Sort the query by CustomerLastName, CustomerFirstName, OrderDate (descending)
 - Save the Order total query

Microsoft® Access 2013 – Database Relationships and Queries



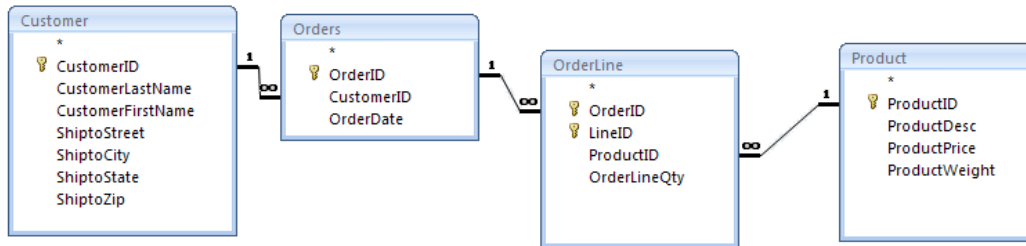
Field:	OrderID	CustomerLastNan	CustomerFirstNar	OrderDate	ShiptoCity	ShiptoZip	LineID	ProductWeight	ProductPrice	OrderLineQty	ExtendedPrice: {[Orde
Table:	OrderLine	Customer	Customer	Orders	Customer	Customer	OrderLine	Product	Product	OrderLine	
Total:	Count	Group By	Group By	Group By	Group By	Group By	Group By	Sum	Avg	Sum	Sum
Sort:		Descending	Descending	Descending							
Show:	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
Criteria:											
or:											

3. Create a query that shows customers without orders.
 - Select Customers to search for unmatched data.
 - Select Orders as the related table
 - Pick the fields CustomerID and CustomerID from both tables
 - Pick all the customer fields.
 - Sort the query by CustomerLastName, CustomerFirstName
 - Save the Customers without Orders query



Field:	CustomerID	CustomerLastNan	CustomerFirstNar	ShiptoStreet	ShiptoCity	ShiptoState	ShiptoZip	CustomerID
Table:	Customer	Customer	Customer	Customer	Customer	Customer	Customer	Orders
Sort:		Ascending	Ascending					
Show:	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
Criteria:								Is Null
or:								

4. Create a parameterized query that prompts the user for a date range
 - Copy the Order total query and name is Orders in Date Range
 - Add the criteria to order date for Between [Begin Date:] and [End Date:]
 - Save the query.



Field:	ShiptoCity	ShiptoZip	OrderDate	OrderID	LineID	OrderLineQty	ProductPrice	ProductWeight	ExtendedPrice: Sum([OrderLineQty]*[ProductPrice])
Table:	Customer	Customer	Orders	OrderLine	OrderLine	OrderLine	Product	Product	
Total:	Group By	Group By	Group By	Group By	Count	Sum	Avg	Sum	Expression
Sort:			Ascending						
Show:	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
Criteria:			Between [Begin Date:] And [End I						
or:									

5. Experiment with INNER JOIN, LEFT JOIN, RIGHT JOIN for each of the queries.
6. Experiment with changing the Join Type.
 - Change the Join type on Customer to Orders relationship to 2.
 - Write a query over Customers and Orders.
 - View the query in SQL View. Does anything change?

Glossary

Term	Definition
Cascading delete	When a row from a parent table is deleted – all dependent rows from the child tables are deleted too.
Cascading update	When the primary key of the parent table is updated, all dependent rows from child tables have their primary key updated to match.
Composite key	Multiple fields are part of the key. A composite primary key has multiple fields whose combination makes the row unique.
Data normalization	The process of determining what fields should belong in each table. The goal is to store each piece of data uniquely without any duplicates or gaps.
Data type	Determines whether the field contains text, numbers, dates
Fields	Fields are attributes of a table. They are also called columns like the columns of a spreadsheet.
Foreign key	Foreign keys are the glue of a relational database. A foreign key is where a primary key from another table matches a field on the current table.
Inner join	An inner join is where data from multiple tables are displayed only if there is an exact match in two tables between the foreign key of one table and the primary key of the other table.
Join type	Join type defines whether the relationship will have a default of an inner join or an outer join.
Junction	A junction is also called an intersecting table. It is used to resolve a many-to-many relationship between two tables by making the primary keys of both tables into a composite primary key of a new table.
Left outer join	A left outer join is where every row from the left table of a related pair is displayed even if there is no corresponding match in the related table on the right. The data from the right table is displayed as nulls if there is no match.
Primary key	The primary key of a table is a field that must uniquely identify each row of the table.
Records	Records are the data contents of a table. They are also referred to as rows like the rows of a spreadsheet.
Referential integrity	Referential integrity ensures that the data entered into related tables is consistent. The foreign key values must match the related primary key values. It also determines whether data can be related from tables if there is matching data in related tables.
Relationship	A relationship is where one table needs to refer to data in another table. Relationships can be one-to-one or one-to-many.
Right outer join	A right outer join is where every row from the right table of a related pair is displayed even if there is no corresponding match in the related table on the left. The data from the left table is displayed as nulls if there is no match.
SQL	Structured Query Language is an open standard for defining and using data from SQL compliant databases like Access, Oracle, or SQL Server. Common SQL commands allow for insert, update, delete, and selection of data.
Table	A table is the container of data. It is like a spreadsheet that contains columns and rows.

Microsoft Access 2013 Shortcuts

Activity	Shortcut Keys
Copy	CTRL+C
Edit the contents of a cell without erasing the entire cell	F2
Display the database window	F11
Find and replace	CTRL+F
Insert a carriage return in a memo or text field	CTRL+ENTER
Insert the current time	CTRL+:
Insert the data from the same field in the previous record	CTRL+'
Insert today's date	CTRL+;
Open a new database	CTRL+N
Open an existing database	CTRL+O
Paste	CTRL+V
Print	CTRL+P
Save	CTRL+S
Switch between the Visual Basic Editor and the previous active window	ALT+F11
Undo	CTRL+Z
Undo the changes you have made to the current field	ESC
Undo the changes you have made to the current record	ESC ESC (press ESC twice)