Northeastern University
College *of* Computer and Information Science

CS1100: Data, Databases, and Queries

QUERY CONSTRUCTION

Microsoft Access Tutorial: Data, Databases, and Queries

LAYOUT OF THE *ORDERS* DATABASE

The *Orders* Database

- We will be using a database that contains data for orders placed by customers for our examples.
- This database stores the following information:
  - For each order know what was ordered, how much of that item was ordered, and at what price.
  - For each order know who placed that order.
  - For each customer (called a contact) store where they live.
  - For each product track its description and price.

A Sample Order

Order ***O0001***

Customer Contact
Contact ID: C0004
Name: Colon, Nicholas
Address: 9020 N.W. 75 Street
Coral Springs, FL 33065

Order Date: 4/15/1999

Product ID	*Product Name*	*Quantity*	*UnitPrice*	*ExtendedPrice*
P0013	DVD Disks	1	$ 23.00	$ 23.00
P0014	HD Floppy Disks	4	$ 9.99	$ 39.96
P0027	Norton Anti-Virus	1	$ 115.95	$ 115.95

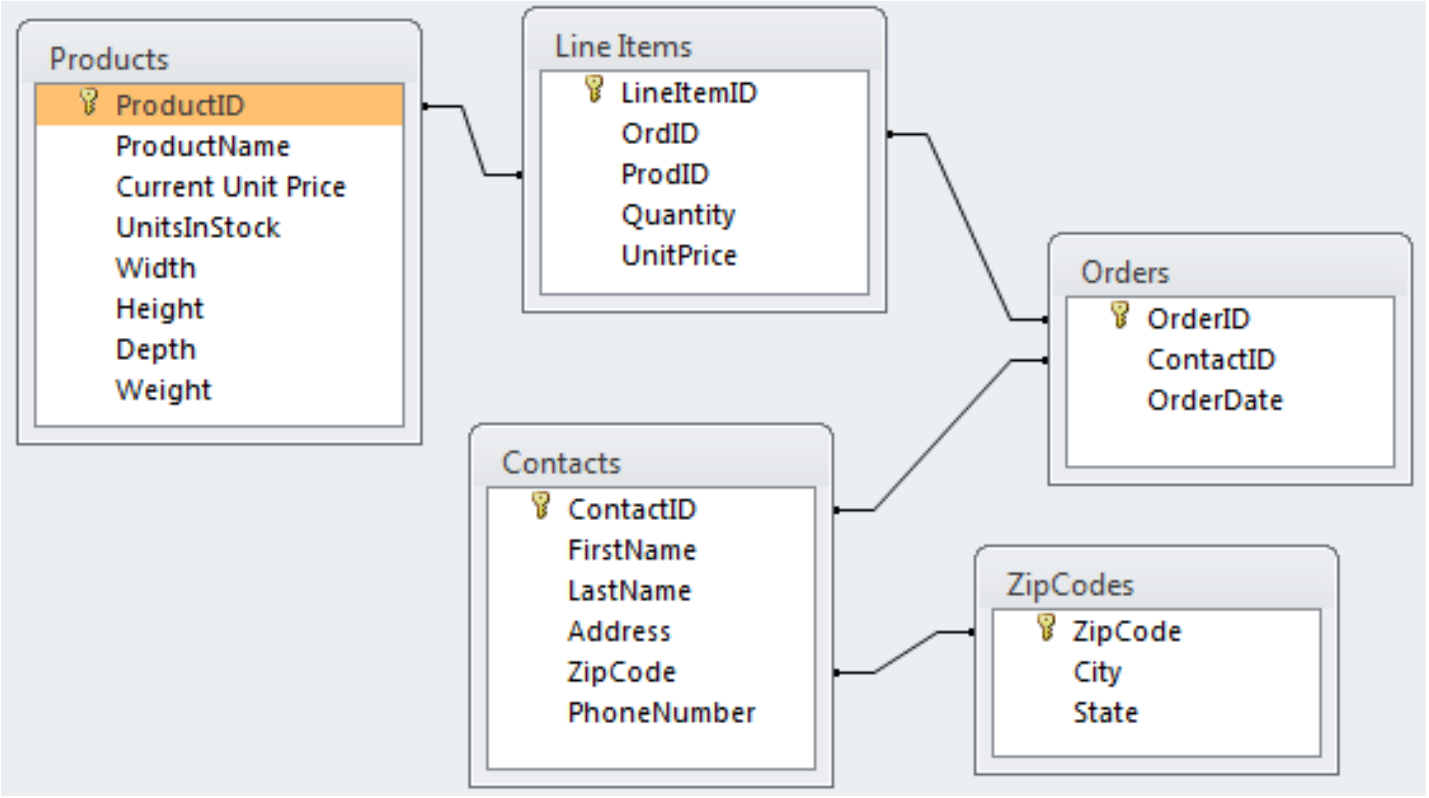
Order Total: *$ 178.91*

Tables, Rows, and Identifiers

- Microsoft Access is a *relational database* which means that it stores data in tables
- Each table contains rows; one row for each record, *i.e.*, a contact, order, product, etc.
- Each row in a table has a unique identifier, *e.g.*, OrderID, ProductID, ContactID, etc.

The Database Layout

- These are all of the tables in the database:



Where Does The Data Come From?

Order ***O0001*** ← Orders.OrderID

Customer Contact — Contacts

Contact ID: C0004
Name: Colon, Nicholas
Address: 9020 N.W. 75 Street
Coral Springs, FL 33065 ← ZipCodes

Order Date: 4/15/1999 ← Orders.OrderDate

ExtendedPrice = Quantity * UnitPrice

Product ID	*Product Name*	*Quantity*	*UnitPrice*	*ExtendedPrice*
P0013	DVD Disks	1	$ 23.00	$ 23.00
P0014	HD Floppy Disks	4	$ 9.99	$ 39.96
P0027	Norton Anti-Virus	1	$ 115.95	$ 115.95

LineItems

Order Total: *$ 178.91*

Total Order Amount

Microsoft Access Tutorial: Data, Databases, and Queries

RETRIEVING DATA WITH QUERIES

Queries

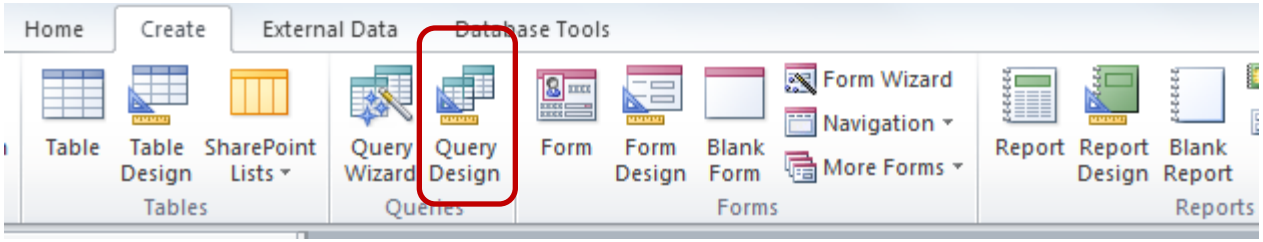
- Data is retrieved through queries.
- Queries are formulated in a specialized language called SQL (pronounced *SEQUEL*).
- Microsoft Access makes it easy to create SQL queries through a simple drag-and-drop interface called the *Query Builder*.
- Queries are eventually integrated into reports, forms, programs, or executed by themselves.

Queries on Tables

- Queries retrieve data from one or more tables.
- You can specify which
  - rows to include in the result through filters (WHERE clause in SQL terminology)
  - columns to include in the results
- The result of a query is a table that can be used in other queries (as subqueries).

Creating Queries in Access

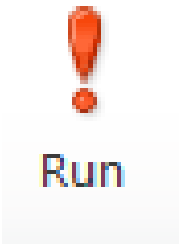
- To create a query:
  - Select the **Create** tab
  - Pick **Query Design**



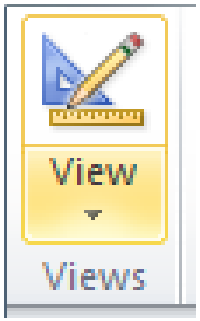
  - Select the tables to include in the query or simply close the dialog and drag the needed tables into the query designer

Running a Query

- To run a query, click on:

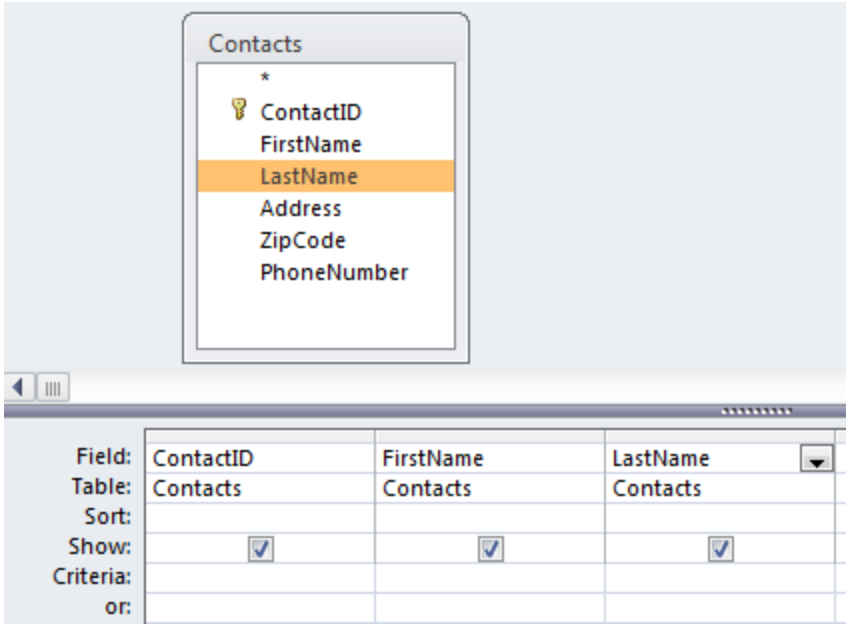


- To return to the query design, click on:



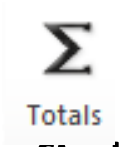
Example: Simple Query

- Find the contact id, first name, and last name for each contact.

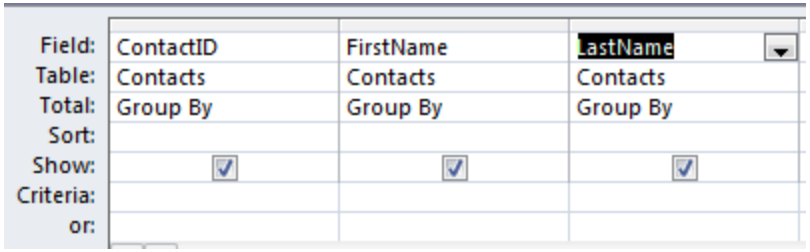


ContactID	FirstName	LastName
1	Benjamin	Lee
2	Eleanor	Milgrom
3	Neil	Goodman
4	Nicholas	Colon
5	Michael	Ware
6	Jeffrey	Muddell
7	Ashley	Geoghegan
8	Serena	Sherard
9	Luis	Couto
10	Derek	Anderson

Removing Duplicates

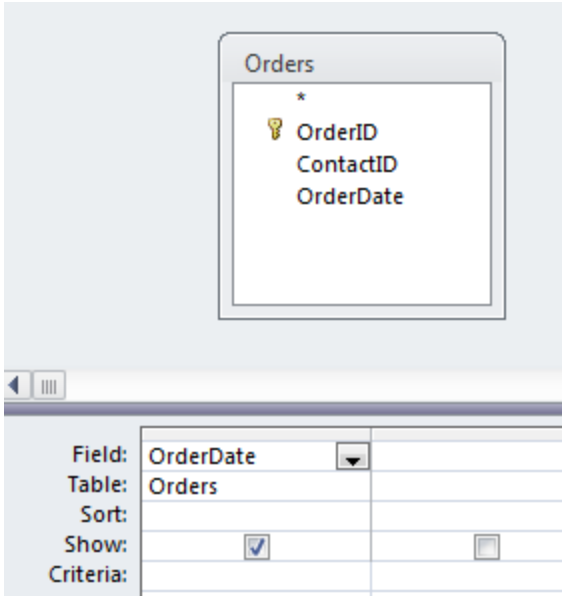
- Queries often result in duplicate rows.
- These are removed by “grouping rows” with the same value as a single row.
- To do a Group By, follow these steps:
  - Select the function button 

 in the ribbon
  - Select “Group By” for each field



Example: Group By

- Find the dates on which orders were placed.
- Here's the result ***without*** a Group By:

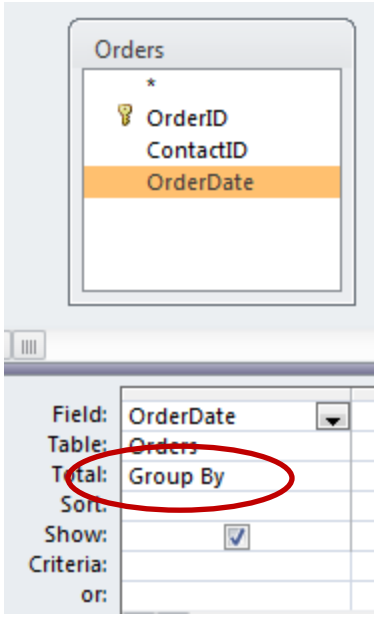


Order Date
4/15/1999
4/18/1999
4/18/1999
4/18/1999
4/20/1999
4/21/1999
4/21/1999
4/22/1999
4/22/1999
4/22/1999
4/24/1999
4/24/1999
4/24/1999

Note the duplicate rows

Example: Group By

- Here's the same query ***with*** a Group By:



Order Date
2/4/1998
3/15/1999
4/15/1999
4/18/1999
4/20/1999
4/21/1999
4/22/1999
4/24/1999
4/25/1999

Note that the duplicate rows have been "collapsed" into groups and only the group is displayed

Duplicates with Group By

- Group By collapses all rows that contain the same data across all columns.

FirstName	LastName	ZipCode	OrderID
Benjamin	Lee	45501	O0005
Benjamin	Lee	45501	O0006
Benjamin	Lee	45501	O0009
Benjamin	Lee	45501	O0011
Eleanor	Milgrom	33063	O0007
Eleanor	Milgrom	33063	O0008
Eleanor	Milgrom	33063	O0010
Eleanor	Milgrom	33063	O0036
Neil	Goodman	33065	O0002
Nicholas	Colon	33065	O0001
Nicholas	Colon	33065	O0013

- Contacts are appearing multiple times in this example as the rows are not exactly the same.

Joining Tables

- A “join” is a query operation in which rows are selected that have a common value for some row.
  - Two columns (each from a different table) can be related if their values represent the same things
- To find contacts that actually placed an order[1], the *ContactID* column value must be the same in the *Contacts* and the *Orders* table.
  - This essentially finds all contacts who placed at least one order.

[1] There may be contacts in the Contacts table that are not linked to any order, *i.e.*, they never placed an order.

Table Joins

- In this example, the ContactID in the Contact table and the ContactID in the Orders table represent the same thing so they can be related.

Contacts Table

ContactID	FirstName	LastName	Addr
1	Peter	Levoy	17 Halp
2	Jane	Wu	68 Trus
3	Jim	Wobek	5 Main

Orders Table

OrderID	ContactID	OrderDate
1	1	4/1/2011
2	1	4/19/2011
3	2	5/13/2011
4	1	6/1/2011
5	2	9/1/2011
6	2	9/5/2011

Steps for Joining Tables

1. For every possible combination, take a row from the first table and a row from the second table
2. Remove all the rows that do not have equal values in the related columns
3. Merge the related columns into one column

Step 1 – all combinations

ContactID	FirstName	LastName	OrderID	ContactID	OrderDate	erDate
1	Peter	Levoy	1	1	4/1/2011	4/1/2011
2	Jane	Wu	1	1	4/1/2011	4/19/2011
3	Jim	Wobek	1	1	4/1/2011	5/13/2011
1	Peter	Levoy	2	1	4/19/2011	6/1/2011
2	Jane	Wu	2	1	4/19/2011	9/1/2011
3	Jim	Wobek	2	1	4/19/2011	9/5/2011
1	Peter	Levoy	3	2	5/13/2011	
2	Jane	Wu	3	2	5/13/2011	
3	Jim	Wobek	3	2	5/13/2011	
1	Peter	Levoy	4	1	6/1/2011	
2	Jane	Wu	4	1	6/1/2011	
3	Jim	Wobek	4	1	6/1/2011	
1	Peter	Levoy	5	2	9/1/2011	
2	Jane	Wu	5	2	9/1/2011	
3	Jim	Wobek	5	2	9/1/2011	
1	Peter	Levoy	6	2	9/5/2011	
2	Jane	Wu	6	2	9/5/2011	
3	Jim	Wobek	6	2	9/5/2011	

Step 2 – remove rows with unequal values in related columns

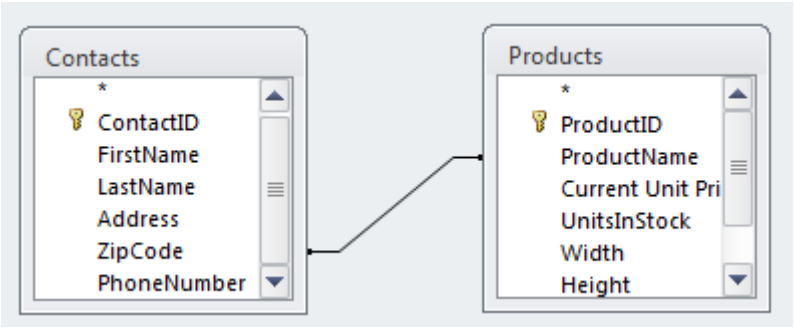
ContactID	FirstName	LastName	OrderID	ContactID	OrderDate
1	Peter	Levoy	1	1	4/1/2011
2	Jane	Wu	1	1	4/1/2011
3	Jim	Wobek	1	1	4/1/2011
1	Peter	Levoy	2	1	4/19/2011
2	Jane	Wu	2	1	4/19/2011
3	Jim	Wobek	2	1	4/19/2011
1	Peter	Levoy	3	2	5/13/2011
2	Jane	Wu	3	2	5/13/2011
3	Jim	Wobek	3	2	5/13/2011
1	Peter	Levoy	4	1	6/1/2011
2	Jane	Wu	4	1	6/1/2011
3	Jim	Wobek	4	1	6/1/2011
1	Peter	Levoy	5	2	9/1/2011
2	Jane	Wu	5	2	9/1/2011
3	Jim	Wobek	5	2	9/1/2011
1	Peter	Levoy	6	2	9/5/2011
2	Jane	Wu	6	2	9/5/2011
3	Jim	Wobek	6	2	9/5/2011

Step 3 – merge related columns

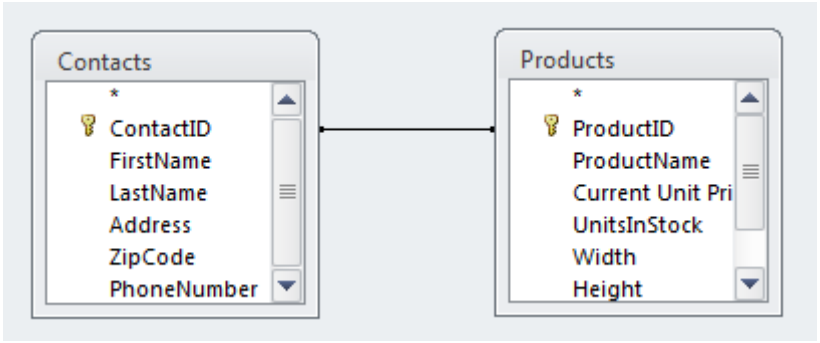
ContactID	FirstName	LastName	OrderID	OrderDate
1	Peter	Levoy	1	4/1/2011
1	Peter	Levoy	2	4/19/2011
1	Peter	Levoy	4	6/1/2011
2	Jane	Wu	3	5/13/2011
2	Jane	Wu	5	9/1/2011
2	Jane	Wu	6	9/5/2011

Bad Joins

- No Output: Attributes are never equal.

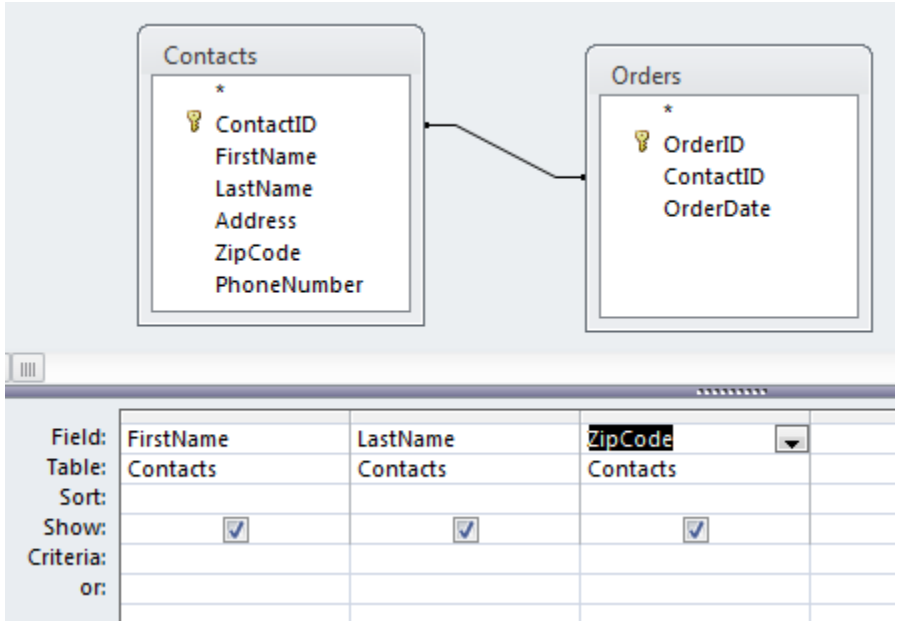


- Meaningless Output: That attributes are equal doesn't mean anything.



Example: Group By and Join

- Find the first name, last name, and zip code of all contacts that placed an order.
- Here's the result ***without*** a Group By:

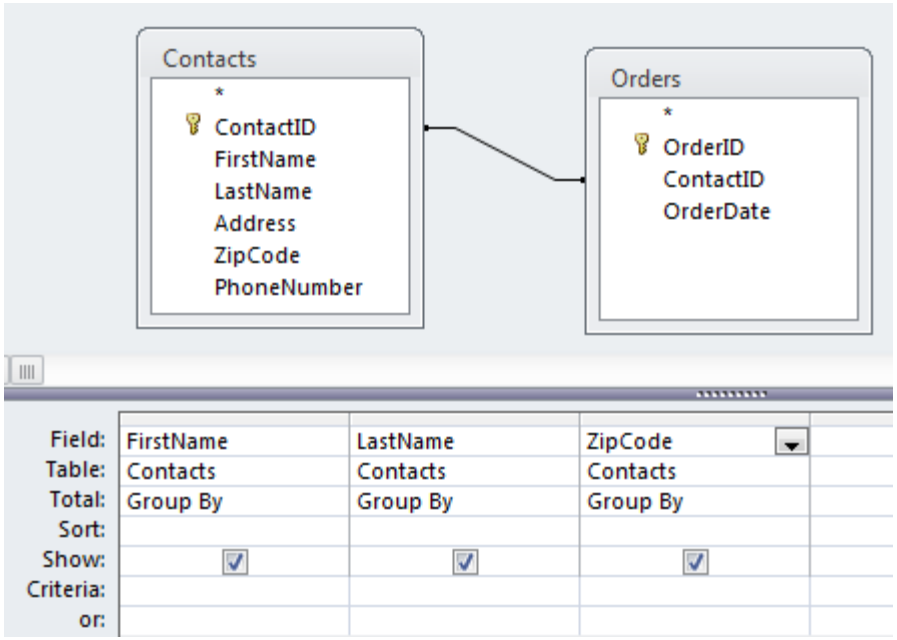


FirstName	LastName	ZipCode
Benjamin	Lee	45501
Benjamin	Lee	45501
Benjamin	Lee	45501
Benjamin	Lee	45501
Eleanor	Milgrom	33063
Eleanor	Milgrom	33063
Eleanor	Milgrom	33063
Eleanor	Milgrom	33063
Neil	Goodman	33065

Note the duplicate rows

Example: Group By and Join

- Find the first name, last name, and zip code of all contacts that placed an order.
- Here's the result ***with*** a Group By:



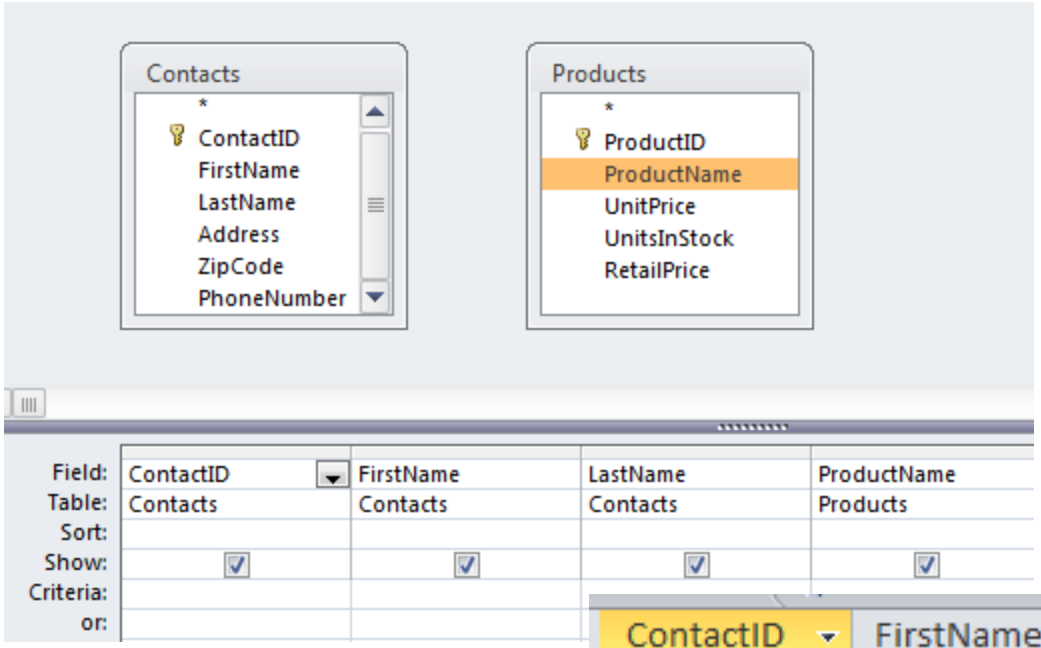
FirstName	LastName	ZipCode
Alan	Turing	33065
Ashley	Geoghegan	33070
Benjamin	Lee	45501
Derek	Anderson	33120
Eleanor	Milgrom	33063

All rows with the same first name, last name, and zip code have been collapsed into a single "group"

Unrelated Tables

- It is possible to join tables without relating columns in those tables.
- Such a join is called an “outer join” or a “Cartesian product.”
- The Cartesian product contains all possible combinations of rows.
- For new database users, the Cartesian product is almost always the wrong thing.

Who ordered which products?



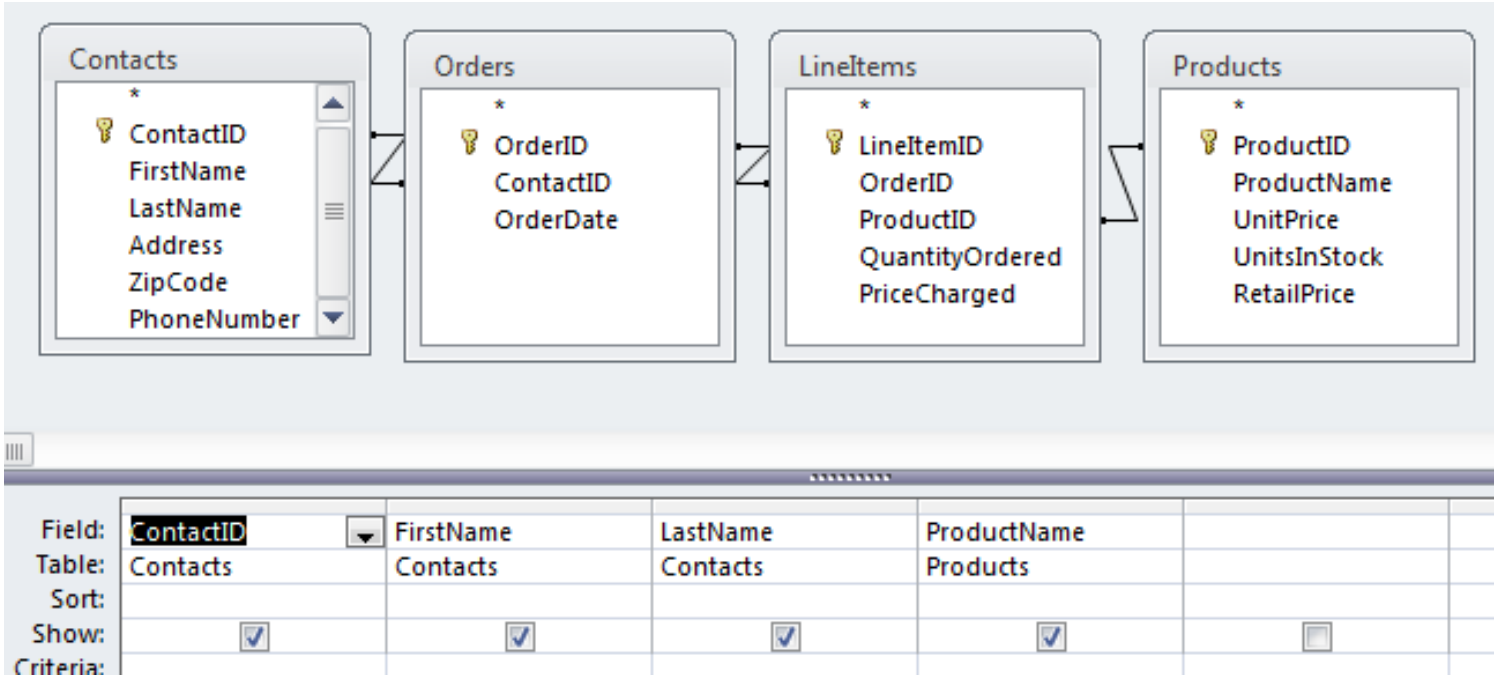
- Unrelated tables -- **wrong**

- Every contact is paired with every product.

ContactID	FirstName	LastName	ProductName
1	Peter	Levoy	Intel i5 2.7Ghz
1	Peter	Levoy	1TB Seagate
1	Peter	Levoy	Windows 7
2	Jane	Wu	Intel i5 2.7Ghz
2	Jane	Wu	1TB Seagate
2	Jane	Wu	Windows 7
3	Jim	Wobek	Intel i5 2.7Ghz
3	Jim	Wobek	1TB Seagate
3	Jim	Wobek	Windows 7

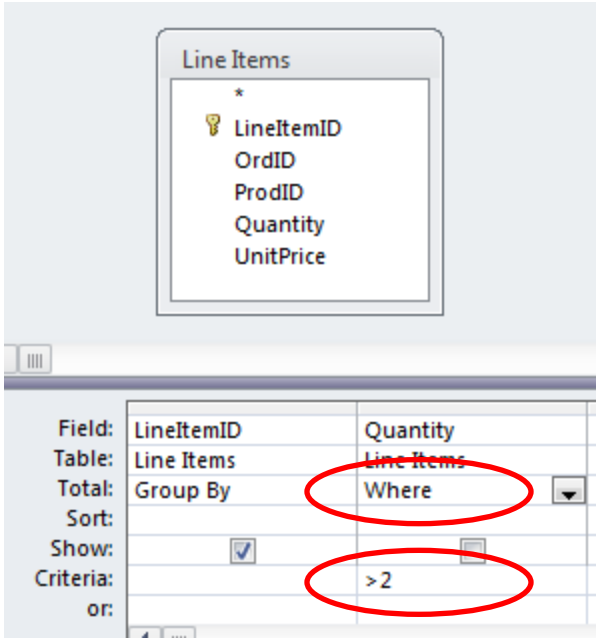
Who ordered which products?

- Correct:



Filtering

- Selecting rows that meet certain criteria is done through a WHERE clause.



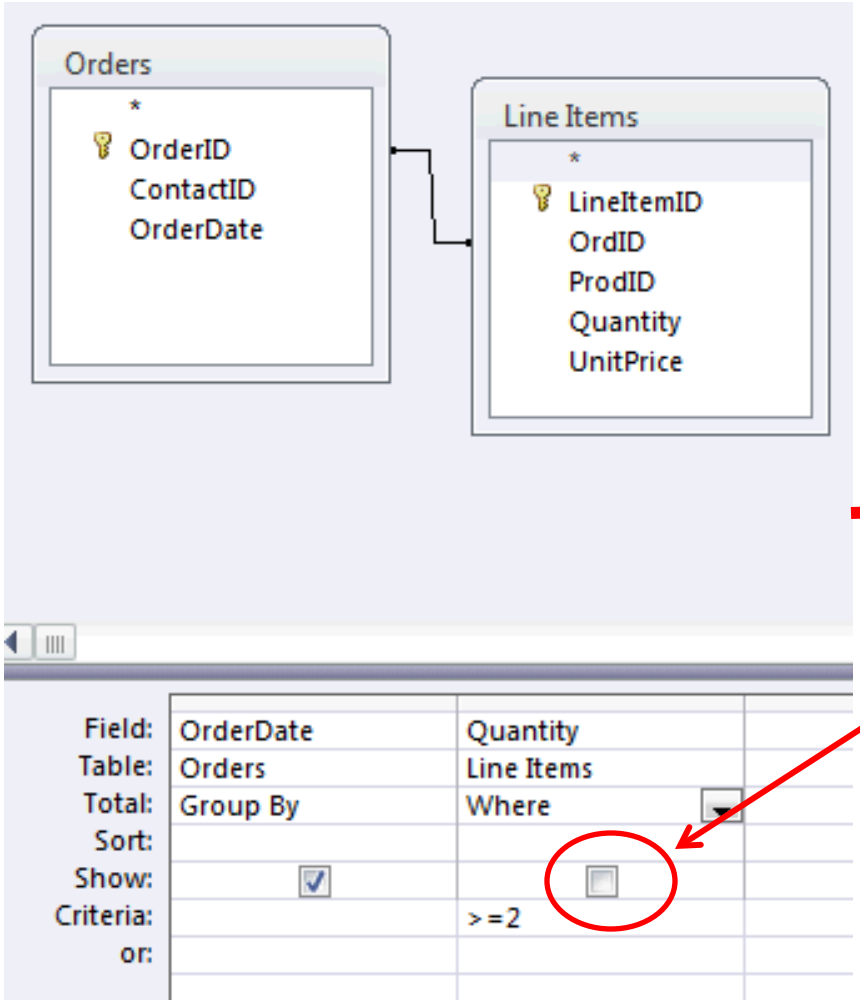
LineItemID
2
18
20
22
23
32
33
34

Lists all of the line items (ID only) that have a Quantity > 2.

Example

- List the OrderDates where the Quantity of an item ordered was at least 2. Each date should only appear once in the result.

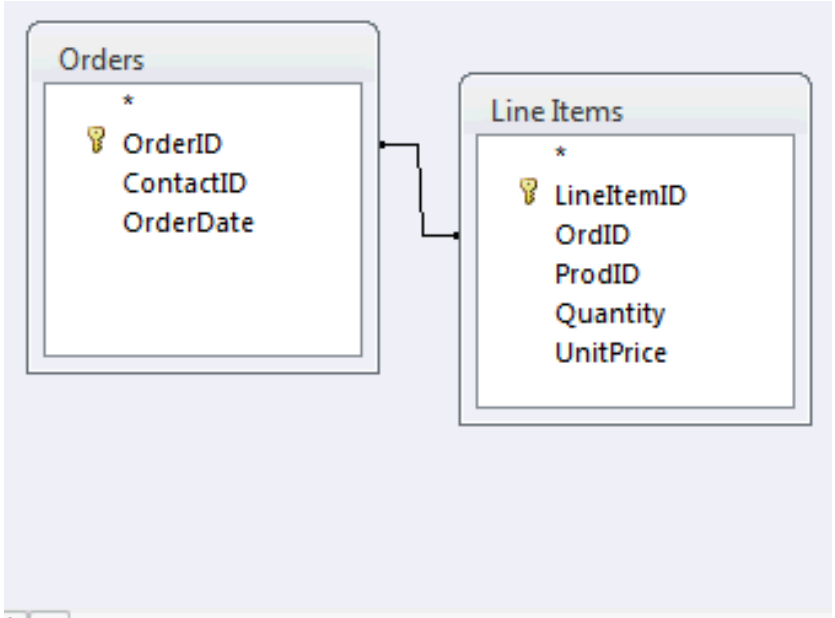
Correct: Use a WHERE Clause



Can't be shown, may be more than one quantity >=2 per date

Order Date
2/4/1998
3/15/1999
4/15/1999
4/18/1999
4/20/1999
4/21/1999
4/22/1999
4/24/1999
4/25/1999
4/26/1999
4/27/1999
5/1/1999
8/21/2001
12/15/2001
4/23/2002
9/2/2002
9/15/2002

Incorrect: Using a GROUP BY instead of WHERE



Can be shown, some dates appear more than once with several quantities >=2

Order Date	Quantity
2/4/1998	2
3/15/1999	2
4/15/1999	4
4/18/1999	2
4/20/1999	2
4/21/1999	3
4/21/1999	10
4/22/1999	2
4/22/1999	3
4/22/1999	4
4/24/1999	2
4/24/1999	4
4/24/1999	10
4/25/1999	2
4/26/1999	2
4/27/1999	25
5/1/1999	2
5/1/1999	3
5/1/1999	5
8/21/2001	12
12/15/2001	2
4/23/2002	12
9/2/2002	12
9/15/2002	2

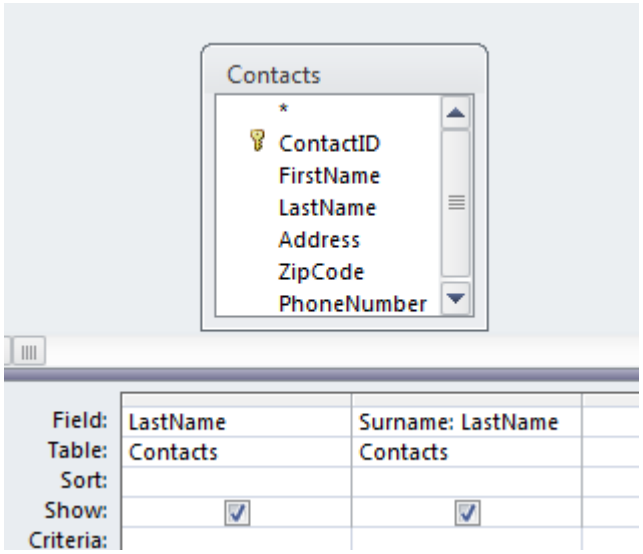
Selection Criteria

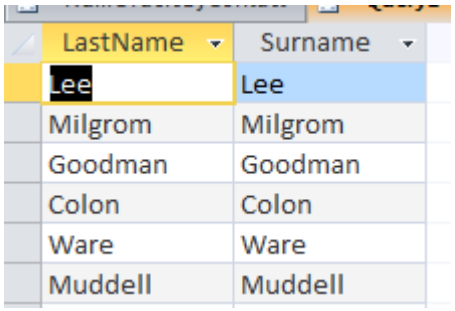
- Selection criteria are specified as an algebraic relationship, but queries are generally stated as a narrative, so we need to "translate".

Narrative	Algebraic Term
At least X	>= X
No more than X	< X
More than X	> X
No less than X	>= X
Less than X	< X
Up to X	< X
At most X	<= X

Renaming Attributes

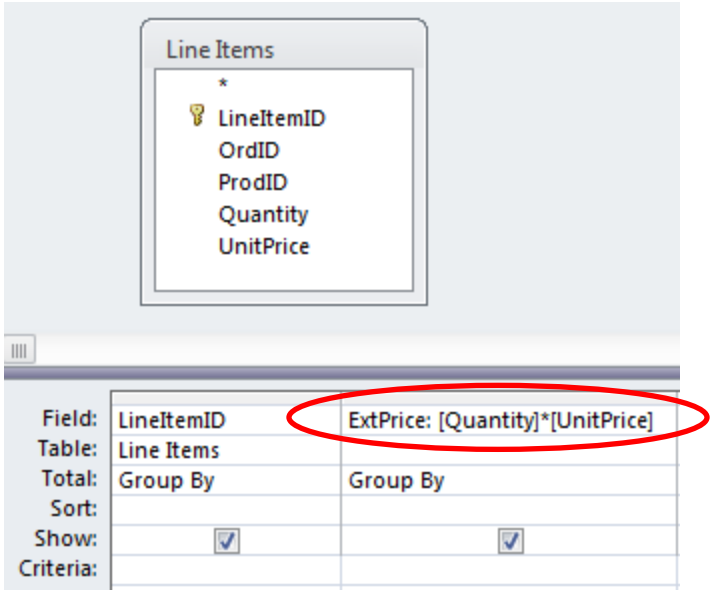
- In query design view, attributes can be renamed with a **:**
  - newname: oldname





Calculated Fields

- A query can also use calculated expressions.
- Example:
  - List the extended prices for each line item.



LineItemID	ExtPrice
1	$23.00
2	$39.96
3	$115.95

Note the user defined field name (ExtPrice) and the use of brackets for the field names, e.g., [Quantity].

Aggregate Functions

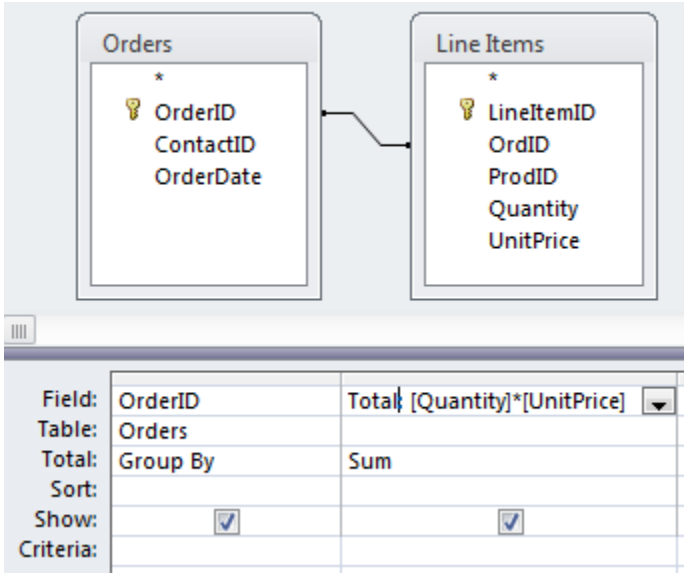
- Aggregate functions work on multiple rows.
- If used within a Group By, they work on the rows within each group.
- If used on an entire table or query, they work on all of the rows in the table or query.

Aggregate Functions

Aggregate Function	Description
COUNT	Counts the rows in a table or a group if used in a Group By
SUM	Adds the values of the summed field in a table or a group if used in a Group By
AVG	Averages the values of the field in a table or a group if used in a Group By
MIN	Finds the minimum value of the field in a table or a group if used in a Group By
MAX	Finds the maximum value of the field in a table or a group if used in a Group By
STDEV	Finds the standard deviation of the field in a table or a group if used in a Group By

Example: SUM

- Find the total of each order, *i.e.*, the sum of the extended prices for each line item within an order.

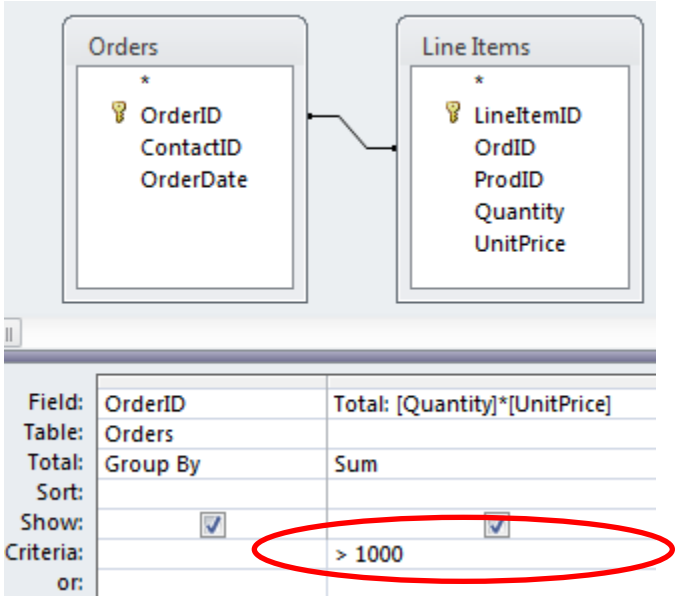


OrderID	Total
O0001	$178.91
O0002	$3,982.95
O0003	$4,183.95
O0004	$5,688.00

Note the user defined field name (*Total*) and the use of brackets for the field names, e.g., [*Quantity*]. The SUM applies to the values of the [*Quantity*]*[*UnitPrice*] within each group.

Example: SUM and Filter

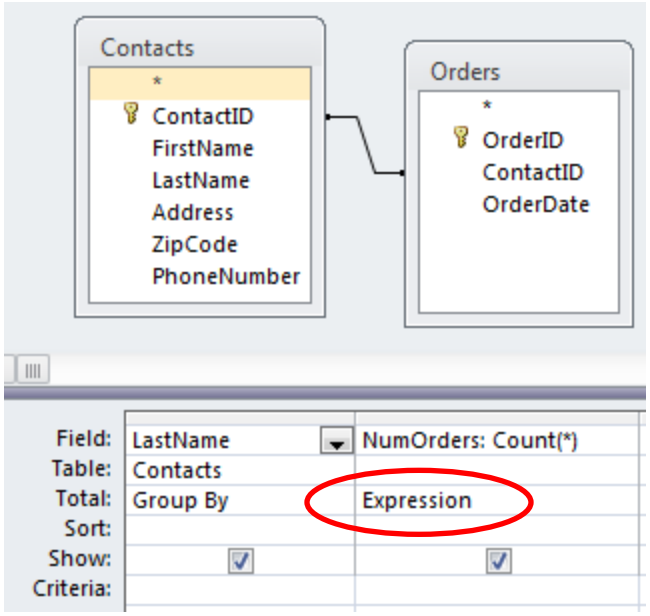
- Find all orders which have a total value of more than $1000.



OrderID	Total
O0002	$3,982.95
O0003	$4,183.95
O0004	$5,688.00
O0005	$5,055.90

Counting Items in a Group

- To count the rows that were collapsed into a group, use the COUNT(*) function.



LastName	NumOrders
Anderson	2
Center	2
Colon	2
Couto	1
Fudge	1
Geoghegan	2

*The Group By collects identical rows into a group. Only the group is displayed, **but** COUNT counts the number of rows that are in each group. So the above query tells us how many orders each contact placed.*

Expressions

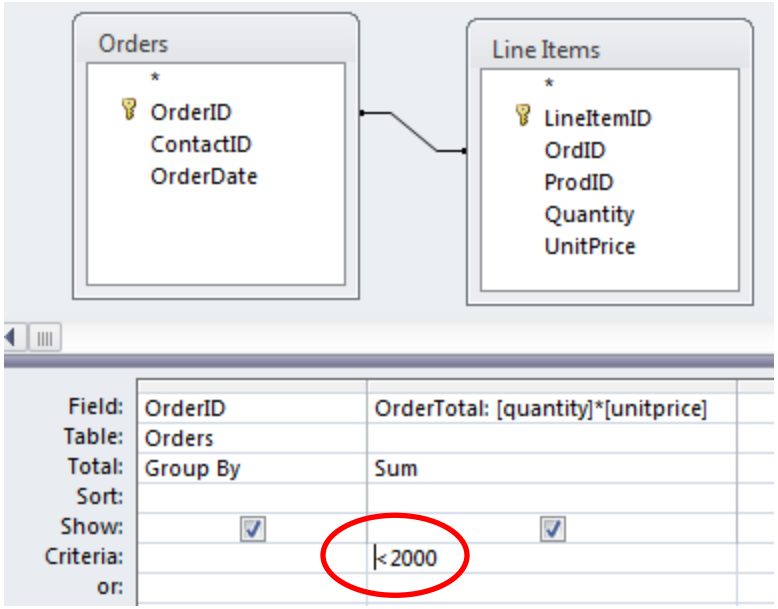
- Note the use of the “Expression” in the Total row for COUNT(*).
- Expression is often required when an aggregate function is used.
- Aggregate functions work on the items within a group of the Group By clause:
  - **COUNT** – counts items in a group
  - **SUM** – adds the items in a group (numbers only)

Query

- Which orders are for less than $2,000?

Query

- Which orders are for less than $2,000?



OrderID	OrderTotal
O0001	$178.91
O0006	$998.90
O0007	$209.80
O0009	$499.00
O0011	$739.80
O0014	$109.85
O0015	$249.90
O0016	$259.90
O0017	$79.90
O0018	$742.80
O0019	$249.75

Access Queries

TRY FOR YOURSELF…

Question 1

- How many orders were placed from each state?

Question 2

- Which contacts placed three or more orders?

Question 3

- How often was each product ordered?

Question 4

- Which products have a total sales volume of more than $10,000, i.e., for which products did the total amount sold across all orders exceed $10,000?

Summary

- Group By removes duplicate rows where the Group By values are the same
- Aggregate functions apply to groups or entire tables depending how they are used