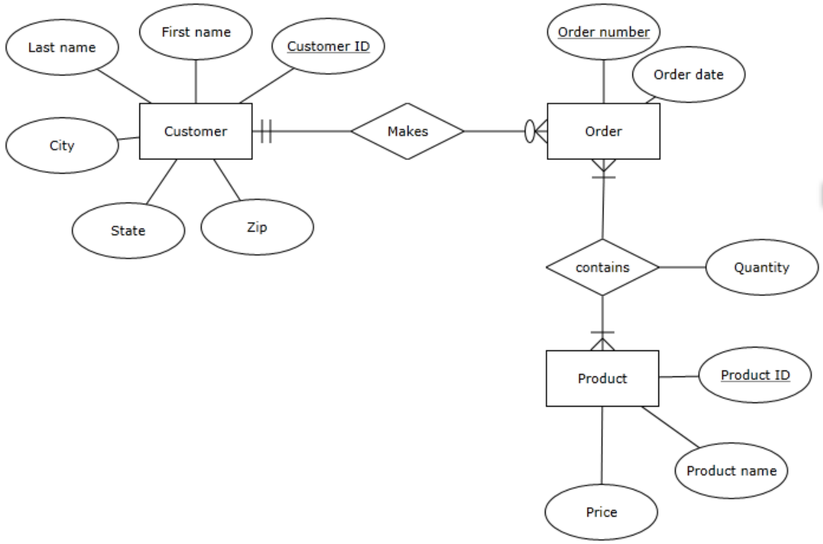
MIS2502:
Data Analytics
Relational Data Modeling (2)

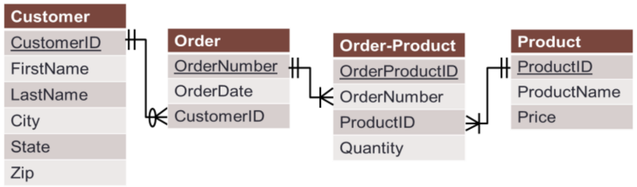
Alvin Zuyin Zheng

zheng@temple.edu

http://community.mis.temple.edu/zuyinzheng/

Let Move From Model to Implementation…




Customer
CustomerID
FirstName
LastName
City
State
Zip

Order
OrderNumber
OrderDate
CustomerID

Order-Product
OrderProductID
OrderNumber
ProductID
Quantity

Product
ProductID
ProductName
Price

Implementing the ERD

- As a database schema
  - A map of the tables and fields in the database
  - This is what is implemented in the database management system
  - Part of the “design” process

- A schema actually looks a lot like the ERD
  - Entities become tables
  - Attributes become fields
  - Relationships ***can*** become additional tables

The Rules

1. Create a table for every entity

2. Create table fields for every entity's attributes

3. Implement relationships between the tables

1:many relationships

- Primary key field of "1" table put into "many" table as foreign key field

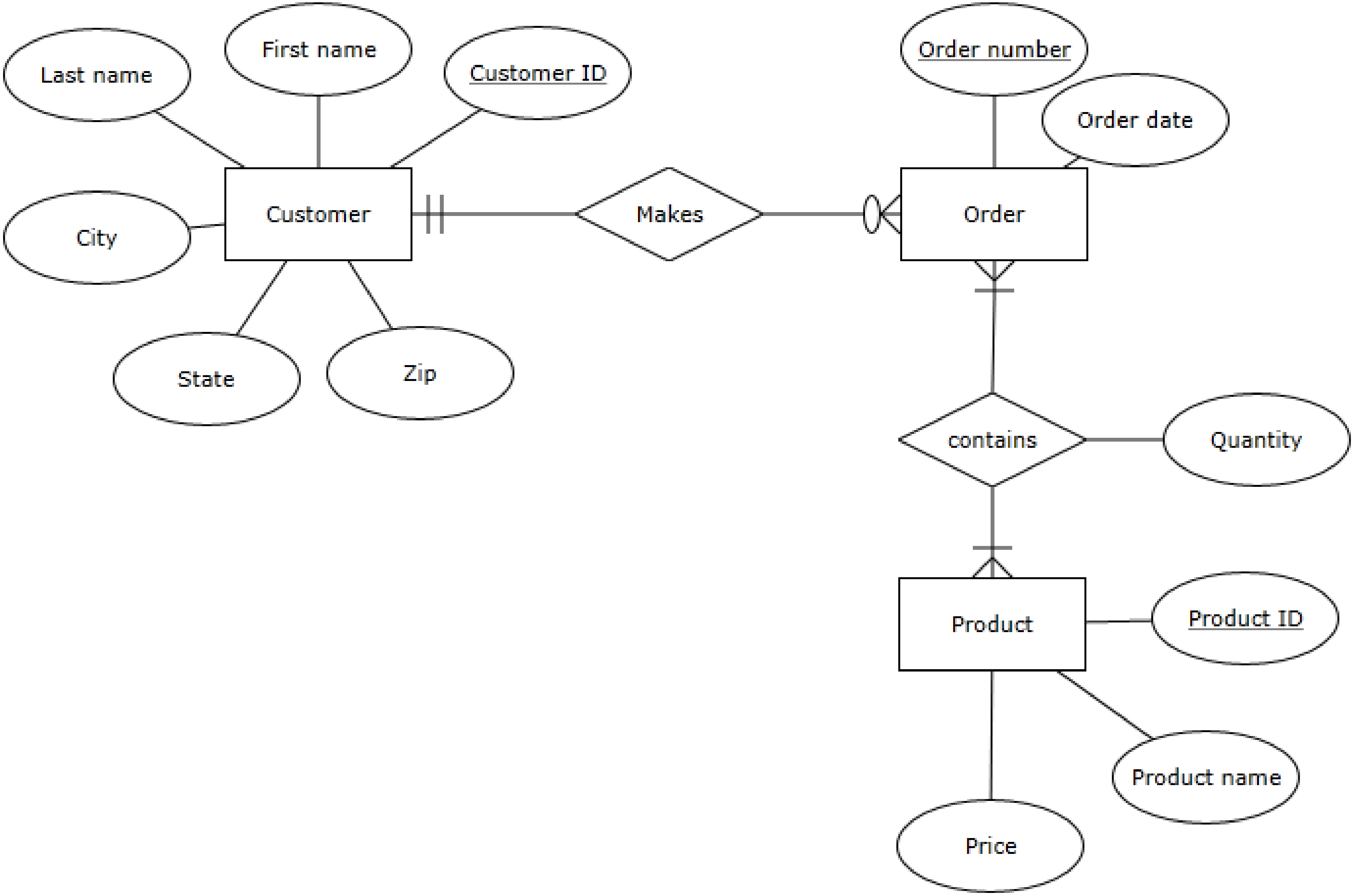
many:many relationships

- Create new table
- 1:many relationships with original tables

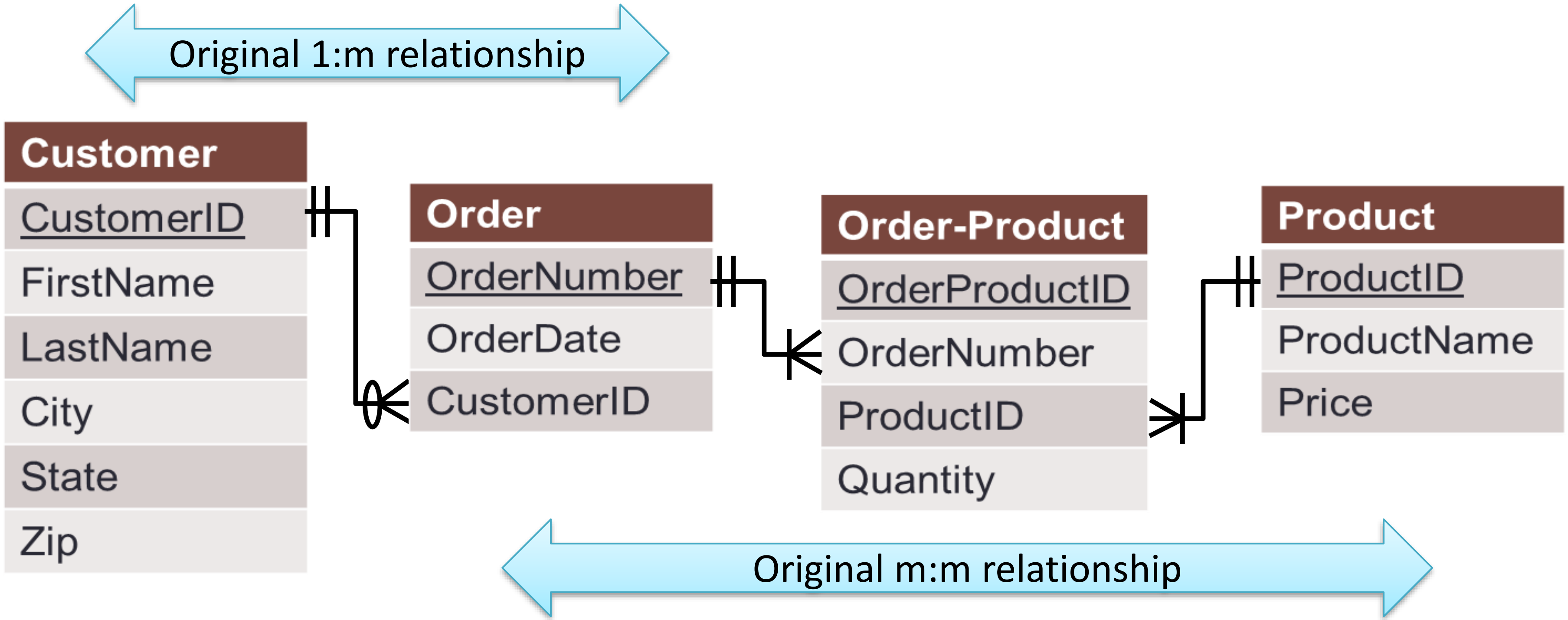
1:1 relationships

- Primary key field of one table put into other table as foreign key field

The ERD Based on the Problem Statement



Our Order Database schema



Order-Product is a decomposed many-to-many relationship

- Order-Product has a 1:m relationship with Order and Product
- Now an order can have multiple products, and a product can be associated with multiple orders

The Customer and Order Tables:
The 1:m Relationship

Customer Table

CustomerID	FirstName	LastName	City	State	Zip
1001	Greg	House	Princeton	NJ	09120
1002	Lisa	Cuddy	Plainsboro	NJ	09123
1003	James	Wilson	Pittsgrove	NJ	09121
1004	Eric	Foreman	Warminster	PA	19111

Order Table

Order Number	OrderDate	Customer ID
101	3-2-2011	**1001**
102	3-3-2011	1002
103	3-4-2011	**1001**
104	3-6-2011	1004

Customer ID is a foreign key in the Order table. We can associate multiple orders with a single customer!

In the Order table, Order Number is unique; Customer ID is not!

The Customer and Order Tables: Normalization

Customer Table

CustomerID	FirstName	LastName	City	State	Zip
1001	Greg	House	Princeton	NJ	09120
1002	Lisa	Cuddy	Plainsboro	NJ	09123
1003	James	Wilson	Pittsgrove	NJ	09121
1004	Eric	Foreman	Warminster	PA	19111

Order Table

Order Number	OrderDate	Customer ID
101	3-2-2011	1001
102	3-3-2011	1002
103	3-4-2011	1001
104	3-6-2011	1004

No repeating orders or customers.

Every customer is unique.

Every order is unique.

This is an example of **normalization**..

To figure out who ordered what

Match the Customer IDs of the two tables, starting with the table with the foreign key (Order):

Order Table			Customer Table					
Order Number	**OrderDate**	**Customer ID**	**Customer ID**	**FirstName**	**LastName**	**City**	**State**	**Zip**
101	3-2-2011	**1001**	**1001**	Greg	House	Princeton	NJ	09120
102	3-3-2011	**1002**	**1002**	Lisa	Cuddy	Plainsboro	NJ	09123
103	3-4-2011	**1001**	**1001**	Greg	House	Princeton	NJ	09120
104	3-6-2011	**1004**	**1004**	Eric	Foreman	Warminster	PA	19111

We now know which order belonged to which customer

- This is called a **join**

Now the many:many relationship

Order Table

Order Number	OrderDate	Customer ID
101	3-2-2011	1001
102	3-3-2011	1002
103	3-4-2011	1001
104	3-6-2011	1004

Product Table

ProductID	ProductName	Price
2251	Cheerios	3.99
2282	Bananas	1.29
2505	Eggo Waffles	2.99

Order-Product Table

Order ProductID	Order number	Product ID	Quantity
1	101	2251	2
2	101	2282	3
3	101	2505	1
4	102	2251	5
5	102	2282	2
6	103	2505	3
7	104	2505	8

This table relates Order and Product to each other!

To figure out what each order contains

- Match the Product IDs and Order IDs of the tables, starting with the table with the **foreign keys** (Order-Product):

Order-Product Table				Order Table			Product Table		
Order ProductID	**Order Number**	**Product ID**	**Quantity**	**Order Number**	**Order Date**	**Customer ID**	**Product ID**	**Product Name**	**Price**
1	**101**	**2251**	2	**101**	3-2-2011	1001	**2251**	Cheerios	3.99
2	**101**	**2282**	3	**101**	3-2-2011	1001	**2282**	Bananas	1.29
3	**101**	**2505**	1	**101**	3-2-2011	1001	**2505**	Eggo Waffles	2.99
4	**102**	**2251**	5	**102**	3-3-2011	1002	**2251**	Cheerios	3.99
5	**102**	**2282**	2	**102**	3-3-2011	1002	**2282**	Bananas	1.29
6	**103**	**2505**	3	**103**	3-4-2011	1001	**2505**	Eggo Waffles	2.99
7	**104**	**2505**	8	**104**	3-6-2011	1004	**2505**	Eggo Waffles	2.99

So which customers ordered Eggo Waffles (by their Customer IDs)?

This is denormalized data

necessary for querying but bad for storage…

stomer	Product ID	Product Name	Price
01	2251	Cheerios	3.99
	2282	Bananas	1.29
001	2505	Eggo Waffles	2.99
	2251	Cheerios	3.99
02	2282	Bananas	1.29
	2505	Eggo Waffles	2.99
	2505	Eggo Waffles	2.99

er	Customer ID	First Name	Last Name	City	State	Zip
	1001	Greg	House	Princeton	NJ	09120
	1002	Lisa	Cuddy	Plainsboro	NJ	09123
	1001	Greg	House	Princeton	NJ	09120
	1004	Eric	Foreman	Warminster	PA	19111

The redundant data seems harmless, but:

What if the price of "Eggo Waffles" changes?

And what if Greg House changes his address?

And if there are 1,000,000 records?

Summary of Database Schema

- Draw the corresponding schema of an ERD
  - Identify tables based on entities and relationships
  - Implement primary key/foreign key relationships
  - Decompose many-to-many relationships in an ERD into one-to-many relationships in the schema

- Best practices for normalization

- Be able to match up (join) multiple tables