MIS2502: Data Analytics

Relational Data Modeling

Jing Gong

gong@temple.edu

http://community.mis.temple.edu/gong
Where we are...

**Transactional Database**
- Stores real-time transactional data

**Analytical Data Store**
- Stores historical transactional and summary data

- Data entry
- Data extraction
- Data analysis
What is a model?

Representation of something in the real world
Modeling a database

• A representation of the information to be captured

• Describes the data contained in the database

• Explains how the data interrelates
Why bother modeling?

• Creates a blueprint before you start building the database

• Gets the story straight: easy for non-technical people to understand

• Minimize having to go back and make changes in the implementation stage
Systems analysis and design in the context of database development

• **Systems Analysis** is the process of modeling the problem
  – Requirements-oriented
  – *What should we do?*

• **Systems Design** is the process of modeling a solution
  – Functionality-oriented
  – *How should we do it?*

This is where we define and understand the business scenario.

This is where we implement that scenario as a database.
Start with a problem statement

Design a database to track orders for a store. A customer places an order for a product. People can place an order for multiple products.

Record first name, last name, city, state, and zip code for customers. We also want to know the date an order was placed.

Finally, we want to track the name and price of products and the quantity of each product for each order.
The Entity Relationship Diagram (ERD)

- The primary way of modeling a relational database
- Three main diagrammatic elements

**Entity**
A uniquely identifiable thing (i.e., person, order)

**Relationship**
Describes how two entities relate to one another (i.e., makes)

**Attribute**
A characteristic of an entity or relationship (i.e., first name, order number)
Begin with Identifying the Entities

This is what your database is about.

1. List the nouns in the problem statement.
2. When nouns are synonyms for other nouns, choose the best one.
3. Make a note of nouns that describe other nouns. These will be your entities’ attributes.
4. Rule out the nouns that don’t relate to the process to be captured.

What’s left are your entities!
So here are the nouns...

Design a **database** to track orders for a **store**. A **customer** places an **order** for a **product**. **People** can place an order for multiple products.

Record **first name**, **last name**, **city**, **state**, and **zip code** for customers. We also want to know the **date** an order was placed.

Finally, we want to track the **name** and **price** of products and the **quantity** of each product for each order.
Here’s where it gets tricky...

**store** is not an entity because we are not tracking specific information about the store (i.e., store location)

BUT...if there were many stores and we wanted to track sales by store, then store would be an entity!

In this case, “store” is the context

But that isn’t part of the problem statement....
The ERD Based on the Problem Statement
The primary key

• Entities need to be uniquely identifiable
  – So you can tell them apart
  – They may not be explicitly part of the problem statement, but you need them!

• Use a primary key
  – One or more attributes that uniquely identifies an entity

  | Customer ID | Uniquely identifies a customer |
  | Order number | Uniquely identifies an order |

How about these as primary keys for Customer:
- First name and/or last name?
- Social security number?
Last component: Cardinality

- Defines the rules of the association between entities

This is a one-to-many (1:m) relationship:
One customer can have **many** orders.
A customer could have **no orders**.
One order can only belong to **one** customer.
An order has to belong to **at least one** customer.
Crows Feet Notation

There are other ways of denoting cardinality, but this one is pretty standard.

So called because this...

...looks something like this

There are also variations of the crows feet notion!
An order can be composed of many products. An order has to have at least one product. A product can be a part of many orders. A product has to be associated with at least one order.

Does it make sense for the maximum cardinality to be 1 for either entity?

Does it make sense for the minimum cardinality to be 0 (optional) for either entity?
Cardinality is defined by business rules

- What would the cardinality be in these situations?

Order contains Product
Course has Section
Employee has Office
Another example of attributes describing a m:m relationship

The grade and semester describes the combination of student and course

(i.e., Bob takes MIS2502 in Fall 2011 and receives a B; Sue takes MIS2502 in Fall 2012 and receives an A)
A scenario: The auto repair shop

Each transaction is associated with a repair, a car, and a mechanic.

Cars, repairs, and mechanics can all be part of multiple transactions.

Many transactions can make up an invoice.

A transaction can only belong to one invoice.

A car is described by a VIN, make, and model.

A mechanic is described by a name and SSN.

A repair is described by a repair id and a price.

A transaction occurs on a particular date and has a transaction ID.

An invoice has an invoice number and a billing name, city, state, and zip code.
Solution
Normalization

• Organizing data to minimize redundancy (repeated data)

• This is good for several reasons
  – The database takes up less space
  – Fewer inconsistencies in your data
  – Easier to search and navigate the data

• It’s easier to make changes to the data
  – The relationships take care of the rest
Normalizing your ER Model

If an entity has multiple sets of related attributes, split them up into separate entities

Don’t do this…

…do this ➔

Then you won’t have to repeat vendor information for each product.
Normalizing your ER Model

Each attribute should be **atomic** – you can’t (logically) break it up any further.

Don’t do this...

![Diagram of a poorly normalized model](Image)

...do this

![Diagram of a well-normalized model](Image)

This way you can search or sort by last name OR first name, and by city, state, or zip code.
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
Our Order Database schema

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

- Order-Product has a 1:n 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:n Relationship

Customer Table

<table>
<thead>
<tr>
<th>CustomerID</th>
<th>FirstName</th>
<th>LastName</th>
<th>City</th>
<th>State</th>
<th>Zip</th>
</tr>
</thead>
<tbody>
<tr>
<td>1001</td>
<td>Greg</td>
<td>House</td>
<td>Princeton</td>
<td>NJ</td>
<td>09120</td>
</tr>
<tr>
<td>1002</td>
<td>Lisa</td>
<td>Cuddy</td>
<td>Plainsboro</td>
<td>NJ</td>
<td>09123</td>
</tr>
<tr>
<td>1003</td>
<td>James</td>
<td>Wilson</td>
<td>Pittsgrove</td>
<td>NJ</td>
<td>09121</td>
</tr>
<tr>
<td>1004</td>
<td>Eric</td>
<td>Foreman</td>
<td>Warminster</td>
<td>PA</td>
<td>19111</td>
</tr>
</tbody>
</table>

Order Table

<table>
<thead>
<tr>
<th>Order Number</th>
<th>OrderDate</th>
<th>Customer ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>101</td>
<td>3-2-2011</td>
<td>1001</td>
</tr>
<tr>
<td>102</td>
<td>3-3-2011</td>
<td>1002</td>
</tr>
<tr>
<td>103</td>
<td>3-4-2011</td>
<td>1001</td>
</tr>
<tr>
<td>104</td>
<td>3-6-2011</td>
<td>1004</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>CustomerID</th>
<th>FirstName</th>
<th>LastName</th>
<th>City</th>
<th>State</th>
<th>Zip</th>
</tr>
</thead>
<tbody>
<tr>
<td>1001</td>
<td>Greg</td>
<td>House</td>
<td>Princeton</td>
<td>NJ</td>
<td>09120</td>
</tr>
<tr>
<td>1002</td>
<td>Lisa</td>
<td>Cuddy</td>
<td>Plainsboro</td>
<td>NJ</td>
<td>09123</td>
</tr>
<tr>
<td>1003</td>
<td>James</td>
<td>Wilson</td>
<td>Pittsgrove</td>
<td>NJ</td>
<td>09121</td>
</tr>
<tr>
<td>1004</td>
<td>Eric</td>
<td>Foreman</td>
<td>Warminster</td>
<td>PA</td>
<td>19111</td>
</tr>
</tbody>
</table>

### Order Table

<table>
<thead>
<tr>
<th>Order Number</th>
<th>OrderDate</th>
<th>Customer ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>101</td>
<td>3-2-2011</td>
<td>1001</td>
</tr>
<tr>
<td>102</td>
<td>3-3-2011</td>
<td>1002</td>
</tr>
<tr>
<td>103</td>
<td>3-4-2011</td>
<td>1001</td>
</tr>
<tr>
<td>104</td>
<td>3-6-2011</td>
<td>1004</td>
</tr>
</tbody>
</table>

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):

We now know which order belonged to which customer

– This is called a join
Now the many:many relationship

**Order Table**

<table>
<thead>
<tr>
<th>Order Number</th>
<th>OrderDate</th>
<th>Customer ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>101</td>
<td>3-2-2011</td>
<td>1001</td>
</tr>
<tr>
<td>102</td>
<td>3-3-2011</td>
<td>1002</td>
</tr>
<tr>
<td>103</td>
<td>3-4-2011</td>
<td>1001</td>
</tr>
<tr>
<td>104</td>
<td>3-6-2011</td>
<td>1004</td>
</tr>
</tbody>
</table>

**Product Table**

<table>
<thead>
<tr>
<th>ProductID</th>
<th>ProductName</th>
<th>Price</th>
</tr>
</thead>
<tbody>
<tr>
<td>2251</td>
<td>Cheerios</td>
<td>3.99</td>
</tr>
<tr>
<td>2282</td>
<td>Bananas</td>
<td>1.29</td>
</tr>
<tr>
<td>2505</td>
<td>Eggo Waffles</td>
<td>2.99</td>
</tr>
</tbody>
</table>

**Order-Product Table**

<table>
<thead>
<tr>
<th>Order ProductID</th>
<th>Order number</th>
<th>Product ID</th>
<th>Quantity</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>101</td>
<td>2251</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>101</td>
<td>2282</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>101</td>
<td>2505</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>102</td>
<td>2251</td>
<td>5</td>
</tr>
<tr>
<td>5</td>
<td>102</td>
<td>2282</td>
<td>2</td>
</tr>
<tr>
<td>6</td>
<td>103</td>
<td>2505</td>
<td>3</td>
</tr>
<tr>
<td>7</td>
<td>104</td>
<td>2505</td>
<td>8</td>
</tr>
</tbody>
</table>

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):

<table>
<thead>
<tr>
<th>Order ProductID</th>
<th>Order Number</th>
<th>Product ID</th>
<th>Quantity</th>
<th>Order Number</th>
<th>Order Date</th>
<th>Customer ID</th>
<th>Product ID</th>
<th>Product Name</th>
<th>Price</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>101</td>
<td>2251</td>
<td>2</td>
<td>101</td>
<td>3-2-2011</td>
<td>1001</td>
<td>2251</td>
<td>Cheerios</td>
<td>3.99</td>
</tr>
<tr>
<td>2</td>
<td>101</td>
<td>2282</td>
<td>3</td>
<td>101</td>
<td>3-2-2011</td>
<td>1001</td>
<td>2282</td>
<td>Bananas</td>
<td>1.29</td>
</tr>
<tr>
<td>3</td>
<td>101</td>
<td>2505</td>
<td>1</td>
<td>101</td>
<td>3-2-2011</td>
<td>1001</td>
<td>2505</td>
<td>Eggo Waffles</td>
<td>2.99</td>
</tr>
<tr>
<td>4</td>
<td>102</td>
<td>2251</td>
<td>5</td>
<td>102</td>
<td>3-3-2011</td>
<td>1002</td>
<td>2251</td>
<td>Cheerios</td>
<td>3.99</td>
</tr>
<tr>
<td>5</td>
<td>102</td>
<td>2282</td>
<td>2</td>
<td>102</td>
<td>3-3-2011</td>
<td>1002</td>
<td>2282</td>
<td>Bananas</td>
<td>1.29</td>
</tr>
<tr>
<td>6</td>
<td>103</td>
<td>2505</td>
<td>3</td>
<td>103</td>
<td>3-4-2011</td>
<td>1001</td>
<td>2505</td>
<td>Eggo Waffles</td>
<td>2.99</td>
</tr>
<tr>
<td>7</td>
<td>104</td>
<td>2505</td>
<td>8</td>
<td>104</td>
<td>3-6-2011</td>
<td>1004</td>
<td>2505</td>
<td>Eggo Waffles</td>
<td>2.99</td>
</tr>
</tbody>
</table>

So which customers ordered Eggo Waffles (by their Customer IDs)?
This is **denormalized data** necessary for querying but bad for storage...

<table>
<thead>
<tr>
<th>Customer ID</th>
<th>Product ID</th>
<th>Product Name</th>
<th>Price</th>
</tr>
</thead>
<tbody>
<tr>
<td>1001</td>
<td>2251</td>
<td>Cheerios</td>
<td>3.99</td>
</tr>
<tr>
<td></td>
<td>2282</td>
<td>Bananas</td>
<td>1.29</td>
</tr>
<tr>
<td>1001</td>
<td>2505</td>
<td>Eggo Waffles</td>
<td>2.99</td>
</tr>
<tr>
<td>1002</td>
<td>2251</td>
<td>Cheerios</td>
<td>3.99</td>
</tr>
<tr>
<td>1002</td>
<td>2282</td>
<td>Bananas</td>
<td>1.29</td>
</tr>
<tr>
<td></td>
<td>2505</td>
<td>Eggo Waffles</td>
<td>2.99</td>
</tr>
</tbody>
</table>

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?

<table>
<thead>
<tr>
<th>Customer ID</th>
<th>First Name</th>
<th>Last Name</th>
<th>City</th>
<th>State</th>
<th>Zip</th>
</tr>
</thead>
<tbody>
<tr>
<td>1001</td>
<td>Greg</td>
<td>House</td>
<td>Princeton</td>
<td>NJ</td>
<td>09120</td>
</tr>
<tr>
<td>1002</td>
<td>Lisa</td>
<td>Cuddy</td>
<td>Plainsboro</td>
<td>NJ</td>
<td>09123</td>
</tr>
<tr>
<td>1001</td>
<td>Greg</td>
<td>House</td>
<td>Princeton</td>
<td>NJ</td>
<td>09120</td>
</tr>
<tr>
<td>1004</td>
<td>Eric</td>
<td>Foreman</td>
<td>Warminster</td>
<td>PA</td>
<td>19111</td>
</tr>
</tbody>
</table>