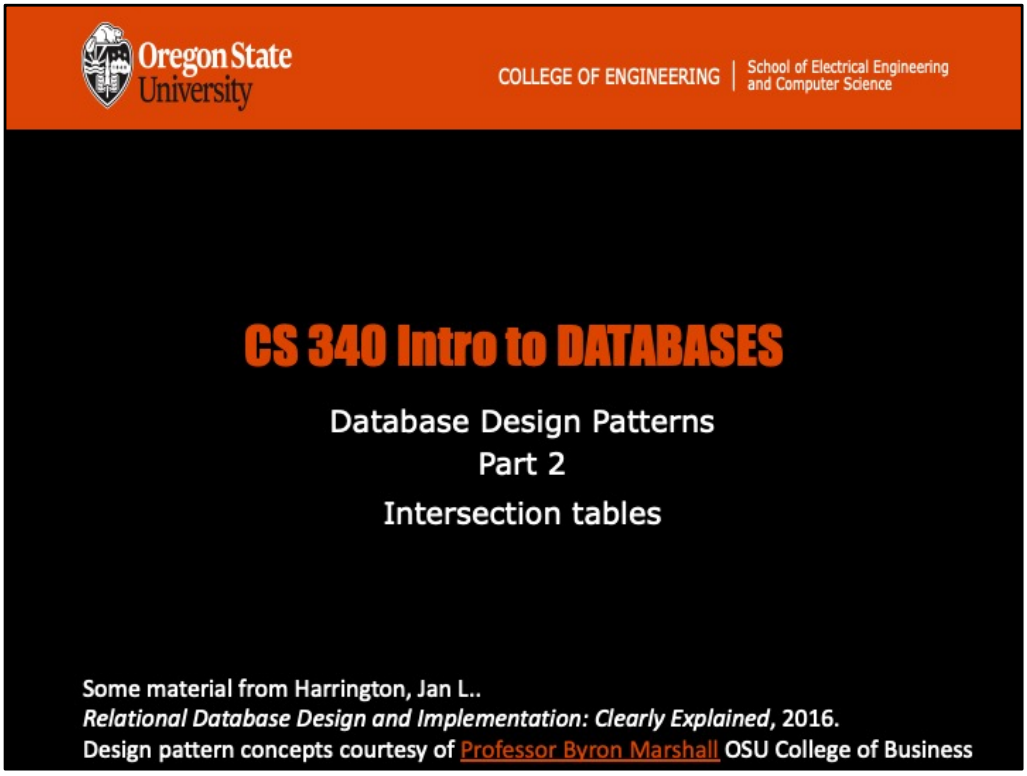
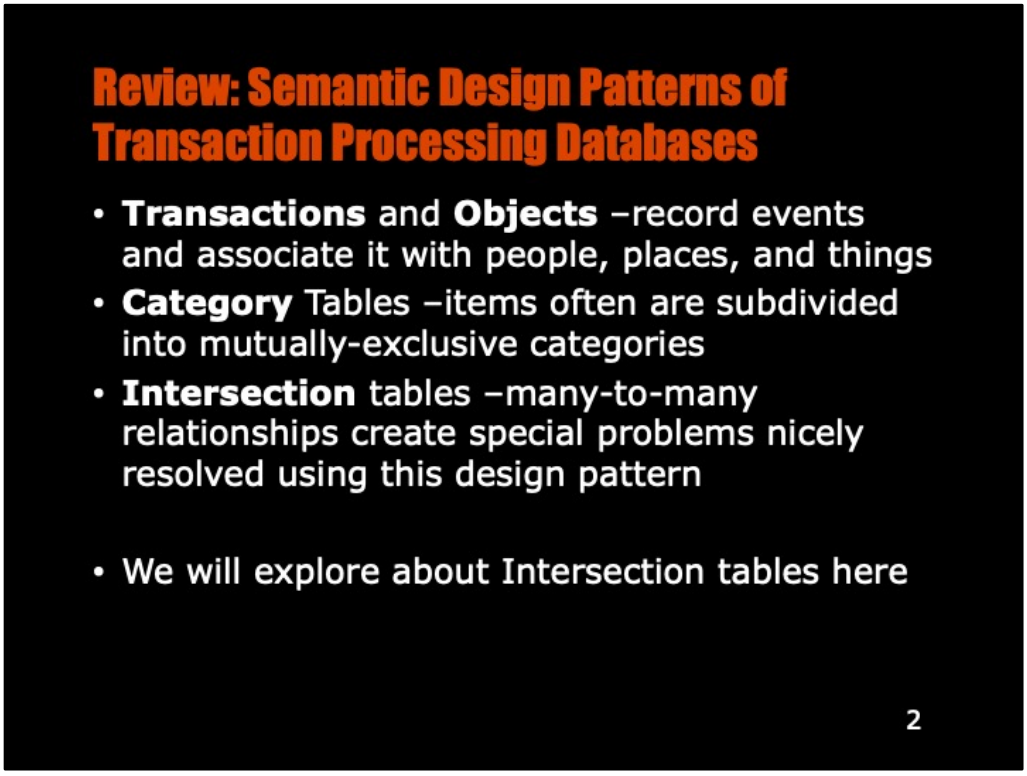


Some of this material comes from the course textbook chapter 11, which you can read for free from the OSU Library.

Harrington, Jan L.. *Relational Database Design and Implementation : Clearly Explained*, Elsevier Science & Technology, 2016. *ProQuest Ebook Central*, http://ebookcentral.proquest.com/lib/osu/detail.action?docID=4509772

Design pattern concepts courtesy of Byron Marshall | College of Business | Oregon State University



Recall we discussed the value of knowing the semantic patterns of an entity. We observe that although there are many different types of entities most of them tend to fall into a small set of common patterns that are used in database design.

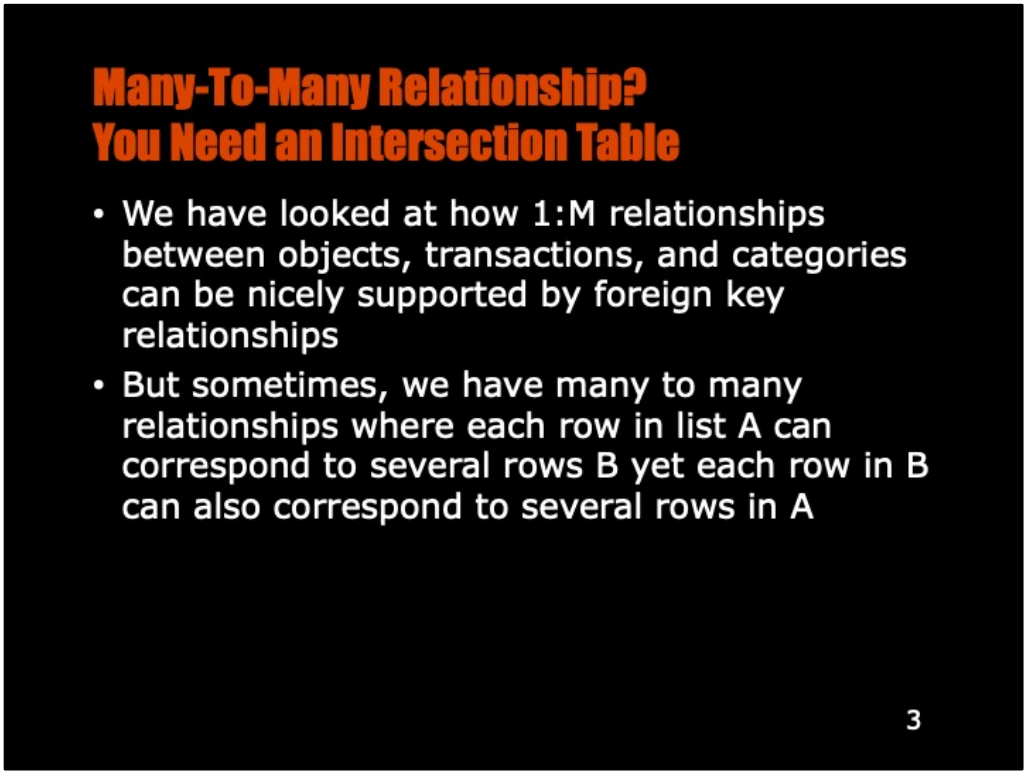
An ***Object*** table holds a list of a particular kind of thing: these things exists over time and can participate in many transactions. Examples include Vendors, Projects, Employees, and Pizza toppings

A ***Transaction*** table holds a list of a particular kind of event which we want to record. An example is Invoices. Details are recorded once for this transaction. The data is used to record sales. Once the transaction has been completed, this data is history and its not typical to change those details. In a college course management schema, Enrollments would be a transaction –once grades are recorded they are rarely changed. In an accounts payable system Checks would be a transaction, the details of the check we wrote would not usually be subject to change.

A **Category** Table design pattern assigns each record in one table into one of several mutually exclusive categories. Oftentimes these categories are useful to the organization. For instance being able to track which employees were full time, part time or contractors is useful to the HR department so the DB designers may use an EmployeeType entity to categorize Employees. The accounts payable department may want to use different Invoice terms (e.g. Due on Receipt, Net 30, or 10% Net 10). So the

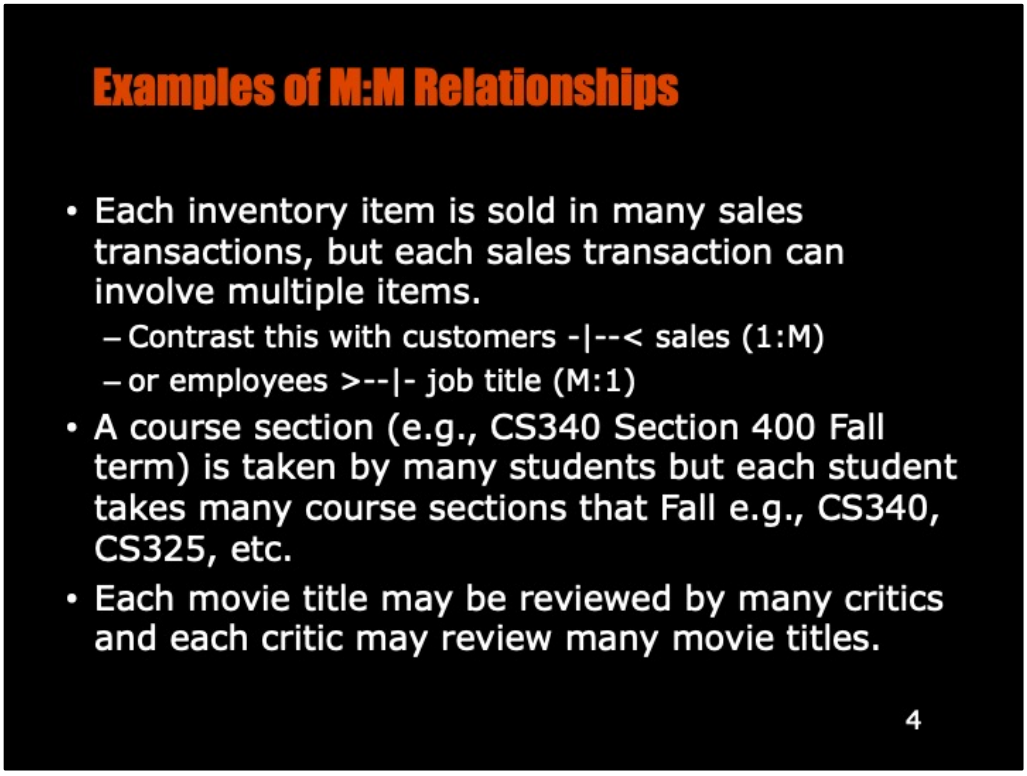
DB designers may use a TermsCode entity to categorize Invoices. In general category tables should not have a lot of data rows. Think about a report that we might generate for the HR department or Accounting department. If we are using a pie chart or a bar chart for example to display percentages of categorization, how many categories make sense? Business Intelligence report designers have found that most users do not want to look at a pie chart with more than 8 slices (source https://datachant.com/2019/10/17/bad-practices-power-bi-pie-charts/). Another example is often time entities are categorized by geographic area. For example state-by-state voting participation might use a United States state map with different colors on each state to indicate frequency. In this example a Votes entity could be categorized by a States entity which had fifty rows of data (one for each state). But trying to categorize with hundreds or even thousands of rows is not recommended.

An **Intersection Table** supports a M:M relationship between two other tables and has at least two foreign key. We sort of glossed over intersection tables previously so in this module we will go into more detail. We will also look at one type of intersection pattern as an example, the **Header-Detail pattern.**



Previous modules focused on how foreign keys support 1:M relationships that commonly occur in organizational data. Foreign keys nicely support relationships where each item in an object list corresponds to a number of recorded transactions. Assignment of items in a list into mutually exclusive categories also illustrates how foreign keys and referential integrity rules effectively validate data stored in an RDBMS. Hint: If you don't understand this paragraph, you have not yet mastered the vocabulary.

Many-to-many relationships also are tracked in most organizational databases – but effectively implementing them in a database requires two foreign keys and an additional table.

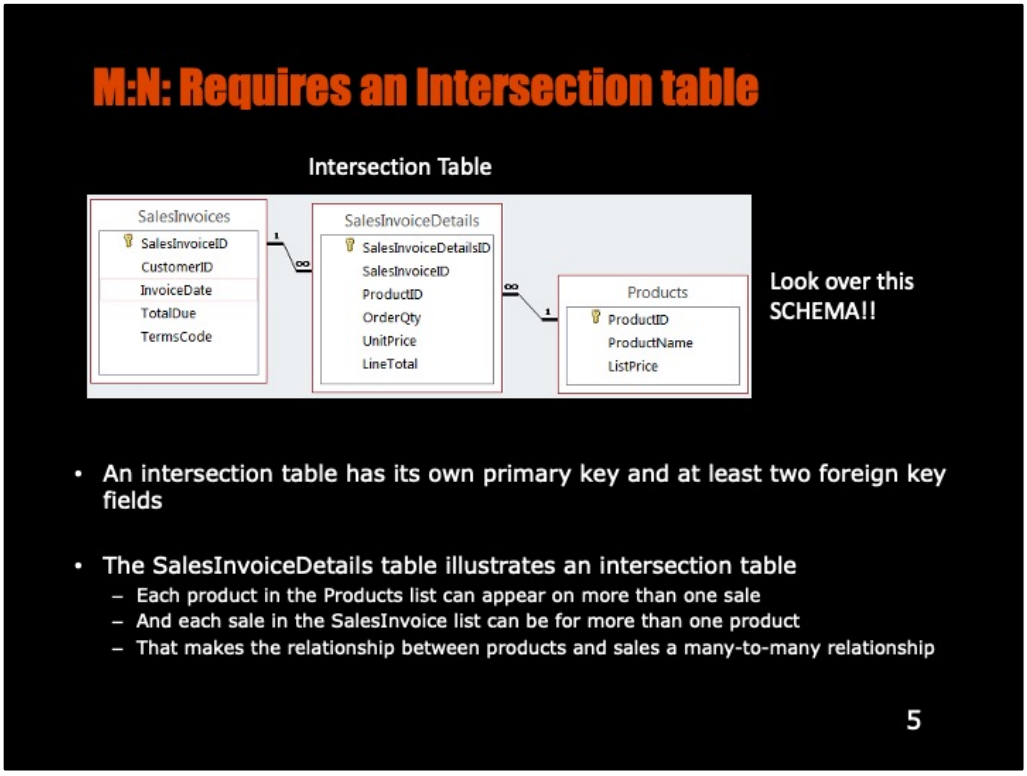


Example M:M relationships

- Each inventory item is sold in many sales transactions, and each sales transaction can involve multiple items.
    - Contrast this with customers+--<sales (1:M)
    - or employees>--+ job title (M:1)

- A course section (e.g., CS340 Section 400 Fall term) is taken by many students but each student takes many course sections that Fall e.g., CS340, CS325, etc.

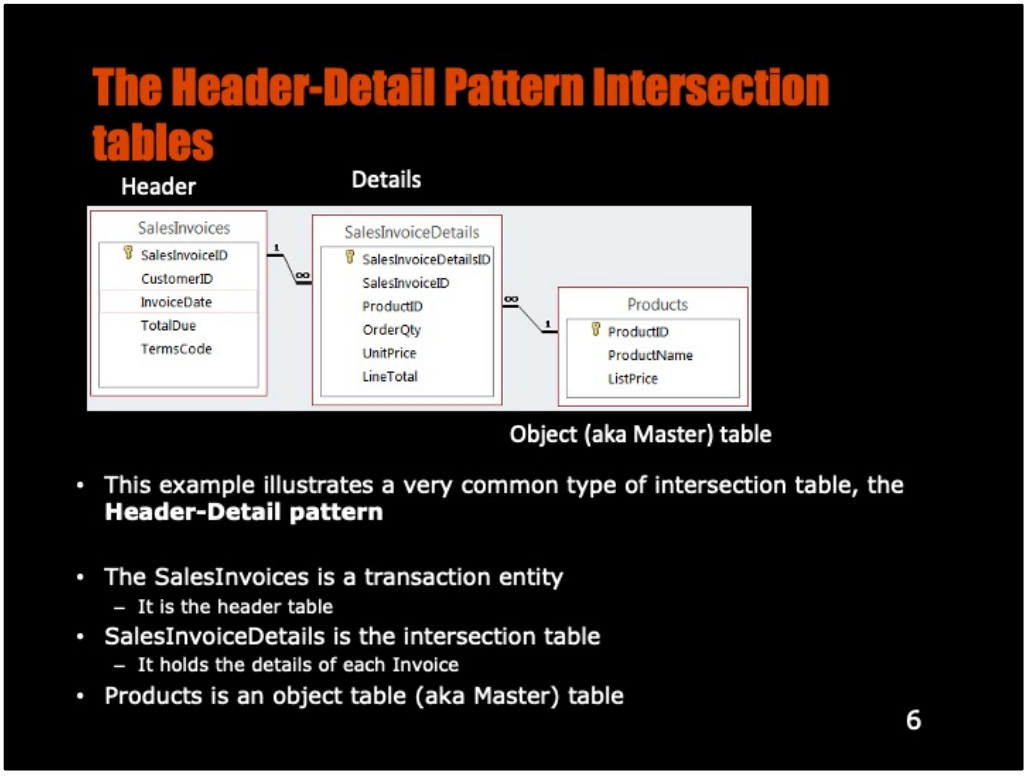
- Each movie title is reviewed by many critics and each critic may review many movie titles



Many-to-many relationships are tracked in organizational databases – but effectively implementing them in a database requires two foreign keys and an additional table.

In this example, the SalesInvoiceDetails table in the Adventure Bikes Sales database (see image) illustrates an intersection table. Each product in the Products list can appear on more than one salesInvoice and each salesInvoice in the SalesInvoice list can be for more than one product. That makes the relationship between products and sales a many-to-many relationship.

While not all M:N relationships are supported by a database design, important ones should be.
In this course, an intersection table has its own primary key, at least two foreign key fields and facilitates a M:N relationship.



This example also illustrates a very common type of intersection table: the **Header-Detail pattern**

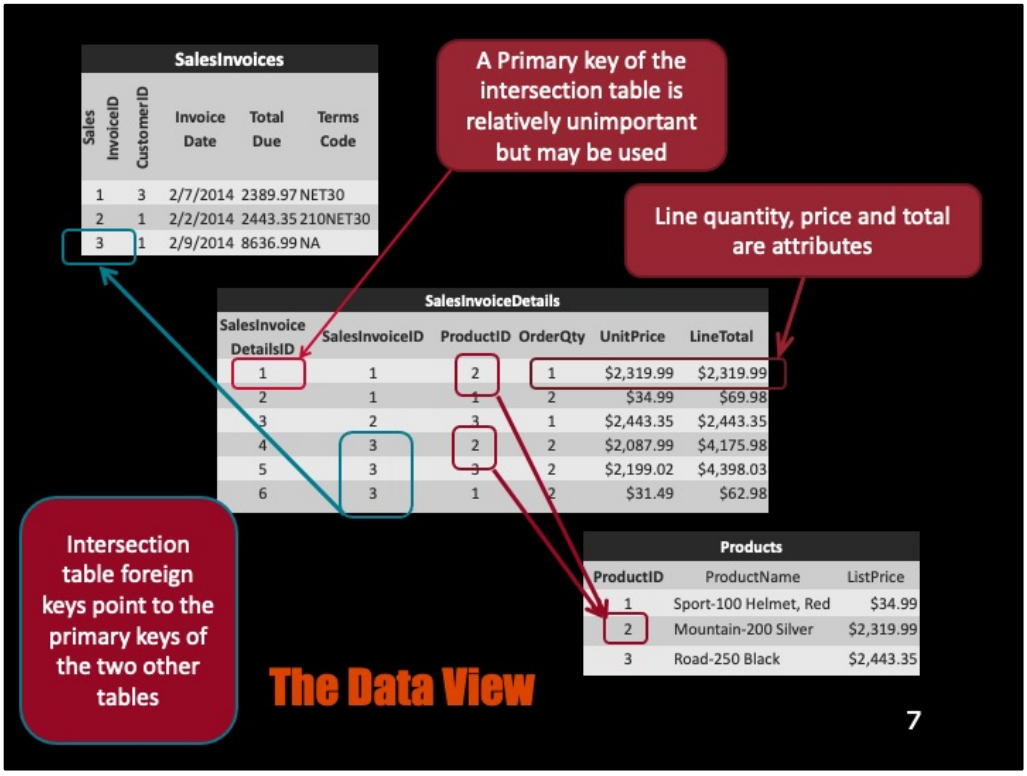
The SalesInvoices is a transaction entity

It is the header table

SalesInvoiceDetails is the intersection table

It holds the details of each Invoice

Products is an object table (aka Master) table



Here is the data view of the previous example

- In this example, each row of the intersection table SalesInvoiceDetails has a primary key
- How do we track which rows correspond to an invoice? By using the SalesInvoiceID FK. For example the items on invoice 3 are listed on SalesInvoiceDetails rows 4, 5 and 6
- How do we know which Products appear on each invoice? By using the ProductID FK. For instance the Mountain-200 Silver (ProductID # 2) appears on Invoice 1 and 3, which is on SalesInvoiceDetails rows 1 & 4
- The quantity, price and line total (quantity x price) are recorded as attributes of each line
- Why record the Price in both the Products and SalesInvoiceDetails tables? Because prices change!

Novice database designers sometimes struggle with the rational of why an intersection table is needed.

Do you see why an invoice number in products won't get the job done?

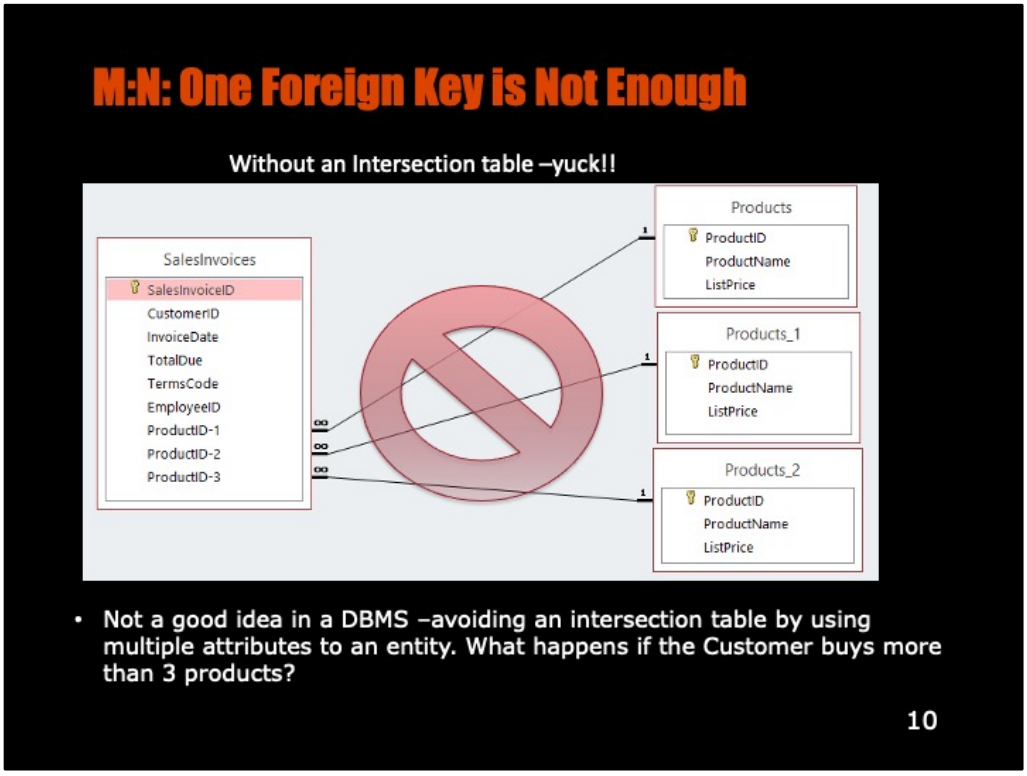
In this example, how many products can ever be on an invoice? Just one (and only one). So what happens if a customer wants to but multiple products? Well with this design we would have to create multiple invoices. That's not going to work.

OK lets try it the other way… Do you see why a product number in invoices doesn't work?

In this example, how many times can that one product (e.g. Mountain-200 Silver) ever be sold? Just once (and only once).

So what happens if we wanted to sell the same type of product a second or third time? The only workable solution would be to create duplicate product (e.g. Mountain-200 Silver-1 and Mountain-200 Silver-2) but again each of those products can only appear on a single invoice.

So that's not going to work.



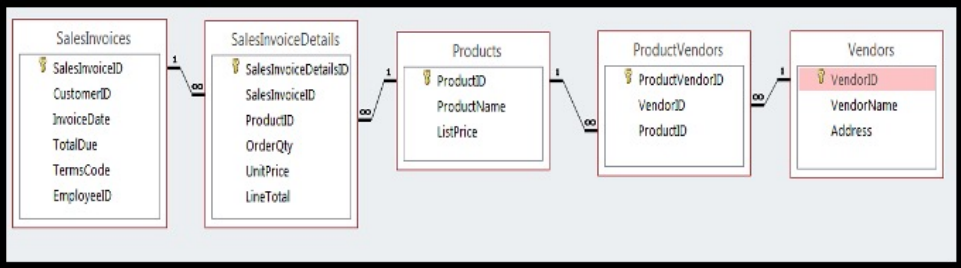
A final method that novice DB designers use sometimes to avoid an intersection table is by duplicating the attribute multiple times as in this example.

How many Products can we add to this invoice? Just three at most (ProductID-1, ProductID-2 & ProductID-2). That clearly won't work for a DB to track sales of products!

However, there are times when this approach can work. For example in a Customers entity it is common to see attributes of email-1 and email-2. The DB designer wants to allow for the possibility of a second email, and because it is not necessary for the DB to track more than two emails, there is no need for an intersection table.

But be careful! Very early in my career as a programmer, I was told by my clients that a Manager would NEVER have more than five employees. So I created a Manager entity in the database without an intersection table with ten employee attributes (twice the max). Guess what happened –a year later there was a supervisor with 14 employees, and I had to re-engineer it, what a pain! I learned my lesson and now I always use an intersection table to facilitate M:N in a RDBMS.

Additional Example



- Adventure Bikes might order the same Product from different Vendors
  - Each Product could have many Vendors
  - And each Vendor may supply many Products
  - ProductVendors implements this M:N relationship
- There are two intersection tables here
  - But Products is NOT an intersection table

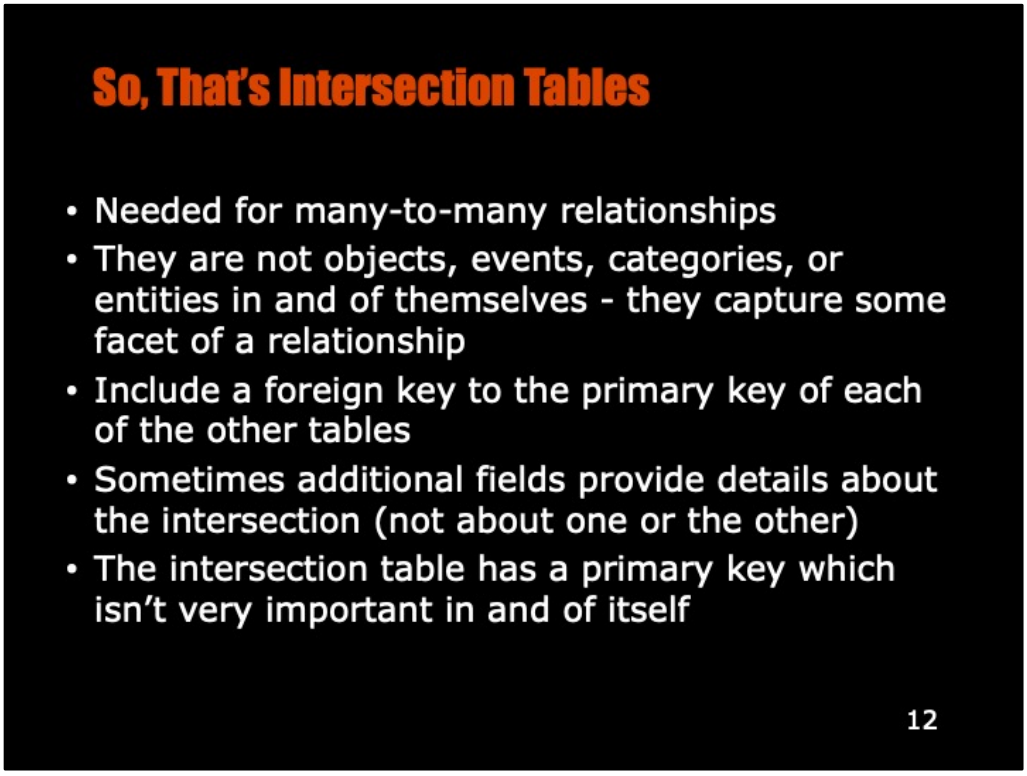
11

Here is one last example. Adventure Bikes might order the same Product from different Vendors

Each Product could have many Vendors
And each Vendor may supply many Products
ProductVendors implements this M:N relationship

There are two intersection tables here

But Products is NOT an intersection table. Think about the definition. It's simply a table between two tables.



Mastering the intersection table pattern is important to understand databases.

- They are needed for many-to-many relationships
- They are not objects, events, categories, or entities in and of themselves - they capture some facet of a relationship
- Include a foreign key to the primary key of each of the other tables
- Sometimes additional fields provide details about the intersection (not about one or the other)
- The intersection table may have a primary key which isn't very important in and of itself

Activity 5 in this module will give you the opportunity to create an Intersection table and add sample data using the MySQL CLI. You will use these skills in your course project as well.