University of Washington

iSchool Info 330

# Module 01 - Database Systems and Design

In this module, you will learn about databases; **why they exist, what types are common, and how they are created**.

## Outline

Here is a general outline of what we will be doing this module:

|  |
| --- |
| **Module 01: Database Systems and Design** |
| Session01 Lectures and Labs < 110 mins |
| Working with Data - 20 |
| Lab: How would you store data from an Invoice? - 30 |
| Relational Databases - 30 |
| Entity Relationship Diagrams - 20 |
| Session02 - Lab < 50 |
| Lab: Create an ERD from an Invoice - 50 |
| Session03 Lectures and Lab < 110 mins |
| Normalization Overview - 30 |
| Lab: How would you normalize a customer list? - 30 |
| Lab: How would you normalize a sales list? - 40 |

**Note**: Times are only estimates and may change without notice!

# Session01 < 110 mins

In this session, we explore **how** relational database **came to be**, what they are and **how** they are **documented**.

## Working with Data - 20

Working with data is the core of this course. Let's talk about what data is and how it is used.

### What Data Is

Data allows you to **store** information.

* Oral traditions
* Replaced by writing on stone
* Replaced by writing on paper
* Replaced by writing to electronic media
  + <https://www.google.com/search?q=paper+to+digital> (external link)

#### Question*:* How long have humans used each of these?

### Where It Is Used

We all work with data in our **daily lives**.

* School grades
* Social media entries
* News facts
* Money

#### Question*: What data do you commonly use*?

### How It Is Used

For data to be useful, data stored must be **turned back into information**. This is done with:

* Documents
* Tables
* Graphs and Charts

#### Question*: What is the difference between data and info*?

Common things you do with data

* Sorting
* Filtering
* Summarizing
* Formatting
* Visualizing

#### Question*: What do you do with data*?

### Why It Is Used

Take a moment to **scan** the following article.

When Less is More: **Migrating from Paper to Digital** Documents, By John Capurso.

* <http://www.visioneer.com/company/news/documents/lessismore-migratingfrompapertodigitaldocuments.pdf> (external link)
* What are the major point?
* When was it written?
* Are its points still valid?

## Lab 1: How would you store data from an Invoice? - 30

In this lab, you will review a typical invoice an ask yourself, "how would I turn this paper-based invoice into digital data?"

***Important****: I do not expect you to know how to do this! I want you to think about how you might do this, try out some of your ideas, and talk about it with others. Expect to not be as good at this as someone who has been trained in data.*

**Note**: This lab can be done individually or with a group of up to three people.

### Step 1: Investigate a Document

1. Open the typical invoice document.
2. Identify the different pieces of data.
3. Name for the different pieces of data.
4. Provide an example of a value for the different pieces of data.
5. Group the pieces data.

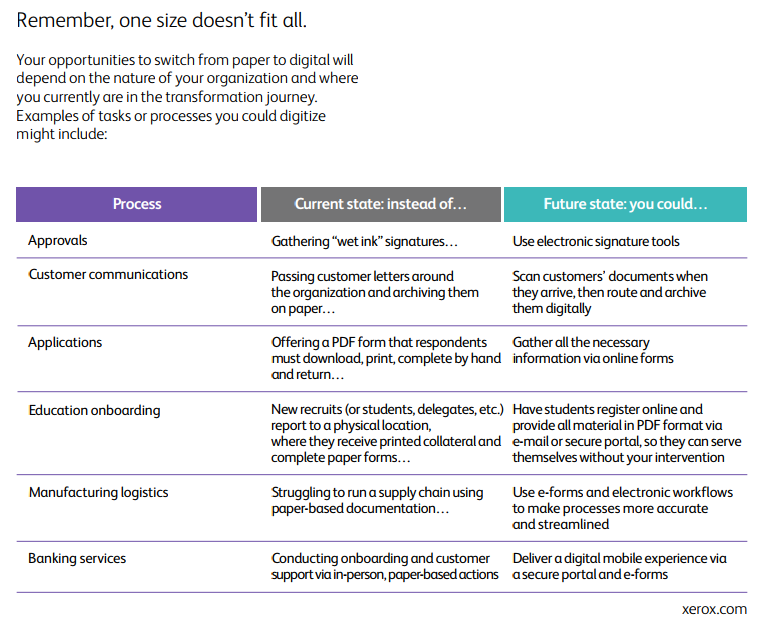
### Step 2: Review Your Work

Now, you will review your work with your instructor.

**NOTE: Unlike assignments, labs do not need to be turned in to Canvas!**

### How It Is Created

Having a plan on how to create data is important, but it **takes practice and experience** to become good at doing so. There are lots of articles on the internet about this and they can help you **avoid common mistakes and dead ends**. While you don't need to look all of this up now, you should become familiar with the process for the day when it becomes important. Here is an example of one of these article images on the Web (Figure 1).

****

**Figure 1: Excerpt from paper-to-digital-transformation.pdf workbook, by Xerox.com**

### Traditional File Processing

"The principal feature of file-processing applications is to allow **each application/designer to create files that are independent** of other file systems within the organization. Each **application** or project is **responsible for designing the contents**, capturing the original data, and maintaining the files throughout the application's lifespan.

This **leads to non-standardization of file contents, formats, design, and security measures**, and makes **access** (e.g., summary of organization-wide information) **very difficult**." (Greg Hey, UW)

## Relational Databases - 30

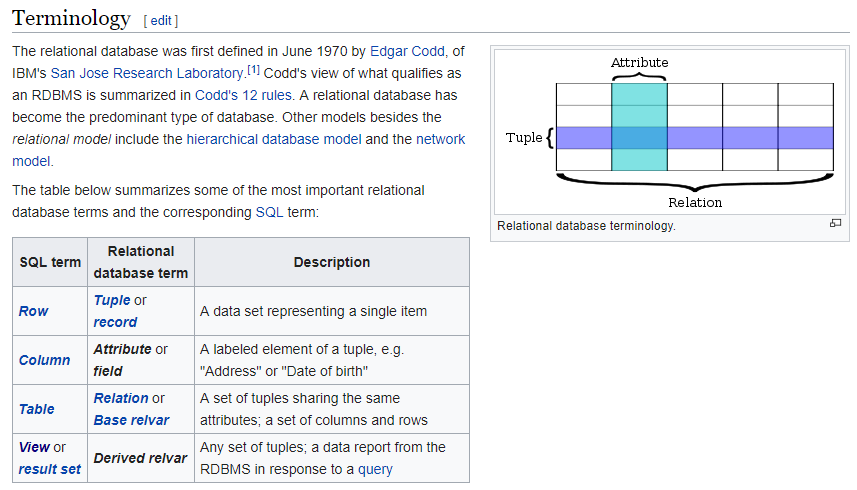
Relational databases are a **common way to store common digital data**. This is because it provides ways to name, group, and locate data once it is stored.

Relational databases have been around for many years, but **in the 1970's their designs became of formalized**.

"A relational database is **a digital database based on the relational model of data**, as proposed **by E. F. Codd in 1970**.[1] A software system used to maintain relational databases is a relational database management system (RDBMS). **Virtually all relational database systems use SQL** (Structured Query Language) for querying and maintaining the database." (<https://en.wikipedia.org/wiki/Relational_database>, 2017)

### Terms

A **relational database is made of relations** (tables). These tables are made of **columns and rows**. You will find as you research this subject that the **names for things vary**, depending on time, location, and technical level of an article.

****

**Figure 2: an excerpt from** [**https://en.wikipedia.org/wiki/Relational\_database**](https://en.wikipedia.org/wiki/Relational_database)**, Terminology, 2017)**

**Some terms**, like relvar (relational variable) you will **almost never hear**, but **other** such as tuple (row) **you may hear from time to time**.

### Design

The design of a **relational** database involves both **dividing data into individual "atomic" values**.

"Bob Smith 206.444.1111, Sue Jones 206.999.1234"

Becomes...

"Bob", "Smith" ,"206.444.1111"

"Sue", "Jones","206.999.1234"

Is then named...

"FirstName " : "Bob" "LastName" : "Smith" "PhoneNumber" : "206.444.1111"

"FirstName " : "Sue" "LastName" : "Jones" "PhoneNumber" : "206.999.1234"

Grouped into collections...

{ "FirstName " : "Bob" , "LastName" : "Smith" , "PhoneNumber" : "206.444.1111" }

{ "FirstName " : "Sue", "LastName" : "Jones", "PhoneNumber" : "206.999.1234" }

These collections can be considered tuples (rows) and each row needs to be a unique combination of values.

Finally, rows are then collected into relations (tables) so that data can easily be located and viewed:

Persons: [

{ "FirstName " : "Bob" , "LastName" : "Smith" , "PhoneNumber" : "206.444.1111" }

{ "FirstName " : "Sue", "LastName" : "Jones", "PhoneNumber" : "206.999.1234" }

]

In a relational database management system (RDMS) software tables are presented like this:

|  |  |  |
| --- | --- | --- |
| FirstName | LastName | PhoneNumber |
| Bob | Smith | 206.444.1111 |
| Sue | Jones | 206.999.1234 |

#### Question: Besides storing data in tables, what other data structures do you know of?

### Constraints

Tables have constraints placed on their columns and rows to **keep the set of rows consistent** (entity integrity), **column** **values consistent** (domain integrity), and **connections between rows consistent** (referential integrity).

There are several constraints in RDMS software. The ones we cover in this course are:

* Primary Key
* Foreign Key
* Unique
* Check
* Null
* Default

#### Question: Can you think of an example of some data you know of and a useful constraint?

### Relational Operations

To access data you need to program your computer to **find what you want**, tr**ansform it into something that makes sense**, and **present** it as **output**.

While **you can do** **this with a language like Python** and some text files, this work has already been done for you when **you use RDMS software like Microsoft's SQL Server**.

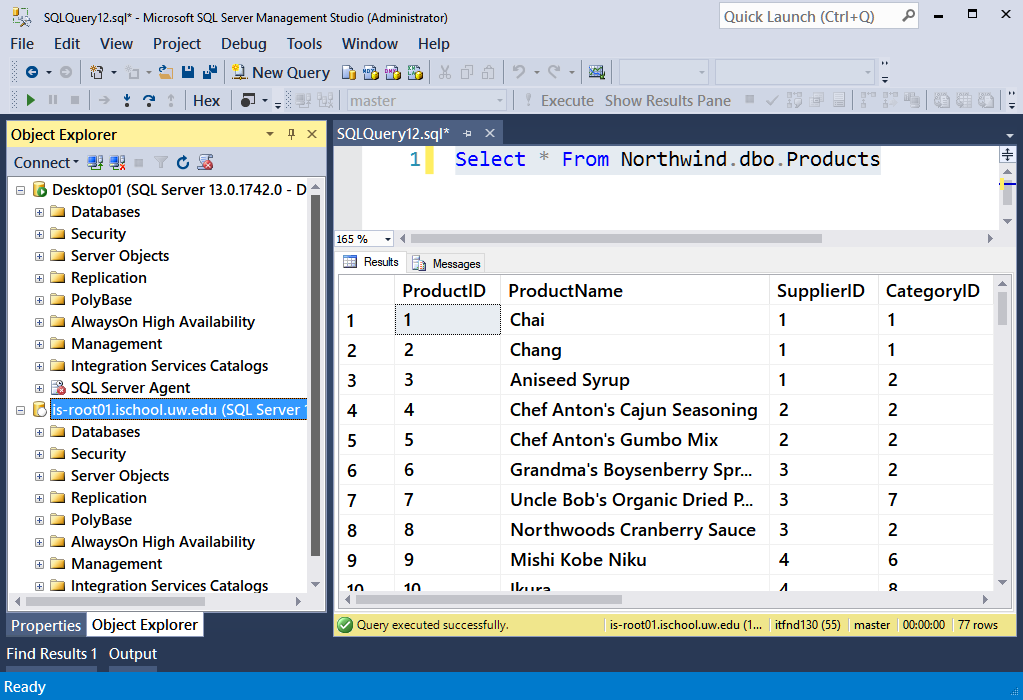
On other systems, like in some Big Data solutions, you may need to **program it yourself** using languages like **Java or Python.**

Over the years, the Structure Query Language (SQL) evolved to perform typical data operations in consistent manner and although some have tried to **create their own non-SQL languages**, **SQL is still the most common language you will use while working with data**.

***Note****: This course will teach you the basics of SQL programming and even a bit beyond the basics!*

### A Tour of SQL Server

Your instructor will now demonstrate how Microsoft's SQL Server RDMS software works.

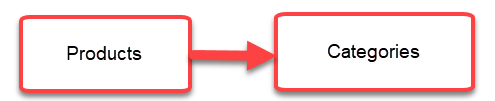
****

**Figure 3: SQL Server Management Studio's User Interface (UI)**

# Entity Relationship Diagrams -20

Database can have **many** tables and these **tables can be related**. These tables (entities) and their relationship are often **shown** using Entity Relationship Diagrams (ERD).

At its simplest, these are just images showing how groups of data are related.

****

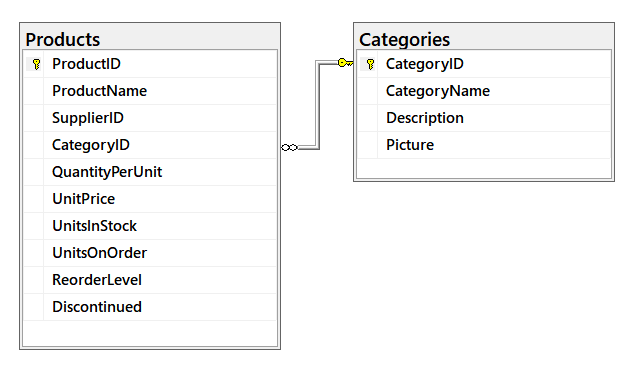
**Figure 4: A simple ERD**

Over the years many people have come up with **organized systems of drawing** these out using standard symbols.

****

**Figure 5: An ERD using standard notation**

Some companies have included ERD features in their RDMS software. For example, SQL Server's Management Studio (SSMS) can create basic ERDs that show table relationships and structures.

****

**Figure 6: An example of a products and categories table in a database ERD using SQL Server**

**Important things to look for** in reading an ERD are:

* **What** Entities are **related** to each other?
* **How** are they **related** to each other?
* What **kind of relation** do they have?

There is also third-party software you can use to create ERDs, but **you do not need them for this course**!

<https://www.smartdraw.com/entity-relationship-diagram/> (external site)

<https://www.draw.io/> (external site)

# Session02 - Lab < 50

Session 02, is a hand-on lab monitored by your TA. You are tasked to perform the following activities during this lab session.

## Lab 2: Create an ERD from an Invoice - 50

In this lab, you will review a typical invoice an ask yourself, "how would I create an ERD using paper based invoice data?"

***Important****:* ***I do not expect you to know how to do this****! I* ***want you to think about how you might do this****, try out some of your ideas, and talk about it with others. Expect to not be as good at this as someone who has been trained in data.*

**Note**: This lab can be done individually or with a group of up to three people.

### Step 1: Review the data

As we did in lab one, review structure of two paper invoices. This time the invoices have data.

* Open the typical invoice document.
* Identify the different pieces of data.
* Name for the different pieces of data.
* Provide an example of a value for the different pieces of data.
* Group the pieces data

### Step 2: Identifying the Entities

Use the groups you found to create named entities.

### Step 3: Create an ERD

Use ERD techniques to create an ERD diagram of these entities and their relationships. Use the "Chen Style" as described in this article, **but do not buy any software!**

* <https://www.smartdraw.com/entity-relationship-diagram/> (external site)

**NOTE:** You can use MS Paint, PowerPoint, or similar software that you already have, to create your ERD.

### Step 4: Review Your Work

Now, you will review your work with your instructor.

**NOTE: Unlike assignments, labs do not need to be turned in to Canvas!**

# Session03 < 110 mins

## Normalization Overview - 30

"Database normalization, or simply normalization, is the **process of organizing the columns** (attributes) and **tables** (relations) of a relational database to **reduce data redundancy** and **improve data integrity**. Normalization is also the process of simplifying the design of a database so that it achieves the optimal structure. It was first proposed by Edgar F. Codd, as an integral part of a relational model." ( <https://en.wikipedia.org/wiki/Database_normalization>, 2017)

#### Normal Forms

There are several rules that make up normalization. These **rules are described as "Normal Forms."** While there are a number of these, the **first three** are considered the most important.

* First normal form (1NF)
  + Table format, atomic values, non-repeating data, primary key identified
* Second normal form (2NF)
  + 1NF and no partial dependencies
* Third normal form (3NF)
  + 2NF and no transitive dependencies

#### First Normal Form

"A **relation** is in first normal form if and only if **the**[**domain**](https://en.wikipedia.org/wiki/Data_domain)**of each**[**attribute**](https://en.wikipedia.org/wiki/Column_(database))**contains only**[**atomic**](https://en.wikipedia.org/wiki/First_normal_form#Atomicity)**(indivisible) values**, and the value of each attribute contains **only a single value from that domain**" (<https://en.wikipedia.org/wiki/First_normal_form>, 2017)

* Relation = Table
* Attribute = Column
* Domain = Set of possible allowed values

##### Table Format

To start the process of organizing data into the first normal form you need a table with columns and rows of data.

##### Atomic Values

Each column should contain values that **cannot be divided further without losing their meaning** (atomic).

A person's name is a good example. Consider data about a customer whose name is "**Bob Smith**." This name can be separated into a first name "**Bob**" and a last name "**Smith**" and still maintain its information and meaning. While we could divide it further into "B", "o", "b", "S","m","i","t","h" it loses its meaning as describing attributes of a given subject, in this case a customer's data.

##### Non-Repeating Data

This mean that you should not have repeating data within the same column. If you have two people with the name of Bob, that is OK, as long is it is not the same data (a different Bob).

This on the other hand the following data is not OK, because the first four columns have repeating values for the same data.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| FirstName | LastName | AreaCode | PhoneNumber | ProductName |
| Bob | Smith | 206 | 999-1212 | ProdA |
| Bob | Smith | 206 | 999-1212 | ProdB |

##### Primary Key Identified

Each table should have a Primary way to identify one row from another. This is done by choosing one column or a set of columns whose values are unique (a determinant). A determinant is any column or combination of column values that determines other values within the same row.

For example, we could use a phone number to determine a persons name.

|  |  |  |  |
| --- | --- | --- | --- |
| FirstName | LastName | AreaCode | PhoneNumber |
| Bob | Smith | 206 | 999-1212 |

However, if that person changes their phone number it is out of our control! For this reason we usually want to use a completely artifact number instead. One that we control and is not tied to any information outside of our database.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ID | FirstName | LastName | AreaCode | PhoneNumber |
| 1 | Bob | Smith | 206 | 999-1212 |

#### Second Normal Form

If you do use a combination, make sure that the data is **not dependent on only part of the combination**. In the following example, the OrderDate is repeating because it is only related to the order's ID and not the individual line item's ID.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| OrderID | OrderLineItemID | OrderDate | ProductID | Quantity |
| 100 | 1 | 1/10/2020 | 1 | 5 |
| 100 | 2 | 1/10/2020 | 3 | 12 |
| 101 | 1 | 1/11/2020 | 1 | 7 |

To fix this we make a new table and move the OrderDate data into it. We will be able to cross reference all the data by referring to the OrderID in the new table to the OrderID in original table.

|  |  |
| --- | --- |
| **OrderID** | **OrderDate** |
| 100 | 1/10/2020 |
| 101 | 1/11/2020 |

|  |  |  |  |
| --- | --- | --- | --- |
| **OrderID** | **OrderLineItemID** | **ProductID** | **Quantity** |
| 100 | 1 | 1 | 5 |
| 100 | 2 | 3 | 12 |
| 101 | 1 | 1 | 7 |

#### Third Normal Form

If you find that **any** other **column** in the row is **determined by the something other** than the chosen **Primary Key**, then it is **violating** the 3rd normal form.

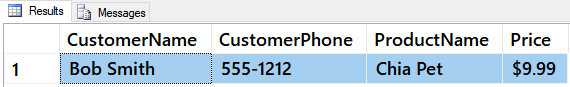
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| OrderID | OrderLineItemID | ProductID | ProductName | Quantity |
| 100 | 1 | 1 | ProdA | 5 |
| 100 | 2 | 3 | ProdC | 12 |
| 101 | 1 | 1 | ProdA | 7 |

### Simple Rules

Instead of only considering the 1NF, 2NF, and 3NF using the academic jargon you can also discuss it using **practical examples**! Here are a few more examples.

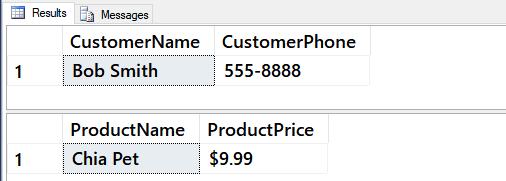
#### All Table should be about only one Subject or Event

Each table should store only data for a **single subject (or event**), like customers or products unless the table is being used to link other tables together.



##### Figure 7: A table with data from two subjects

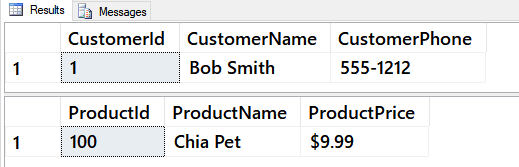
To normalize this data, you would break the table into two tables each about a separate subject.



**Figure 8: Two tables with one subject each**

#### Use artificial Primary Key values

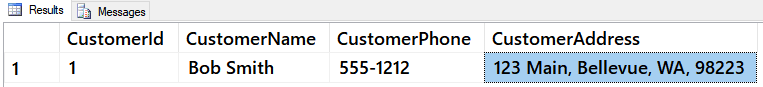
As mentioned, each table needs a way to distinguish between rows. It is considered a best practice to create your own **artificial identifier**.



**Figure 9: Two tables with an artificial primary key identifier**

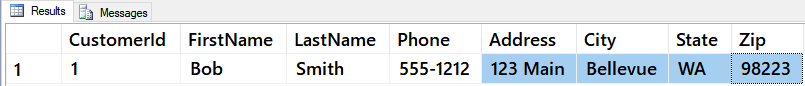
#### No Multi-PART Fields

Normalization Rule 1 states that every column in table must be **atomic** (single value). Consider how this table violates this rule because the customer's address can be divided further into individual values.

****

**Figure 10: A table with a multi-part address field**

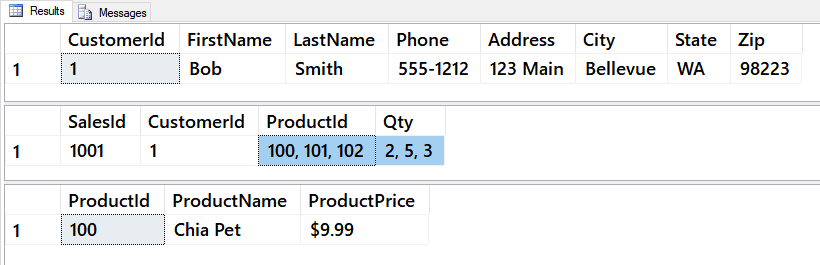
To fix this, you **break up multi-part fields** (non-atomic fields containing more than one piece of data) into other columns.

****

**Figure 11: A table with the address divided into atomic values**

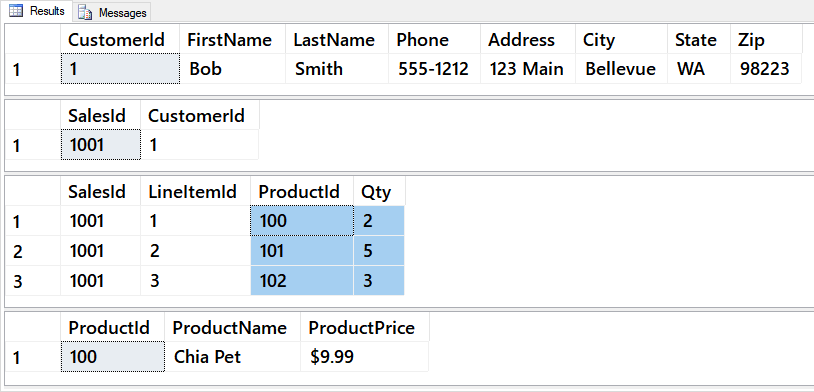
#### No Multi-VALUED Fields

While the address was a single piece of information about one customer that could be divide into separate pieces of data, this next example is **multiple pieces of information** about many products incorrectly **stored as a single piece of data**.

****

**Figure 12: A table storing multiple pieces of information as a single data value**

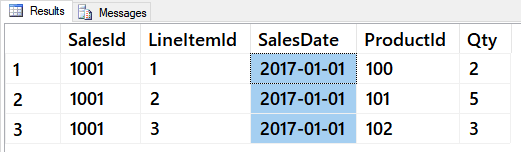
To fix this, we **break up multi-value fields into** more columns, like before, but this time we add these new columns to a **separate table**!

****

**Figure 13: A table storing multiple pieces of information as individual rows of values**

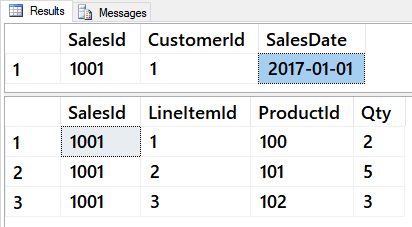
#### All columns must depend on composite primary key when there is one

If a Table has a **composite Primary Key** then **all columns must depend on the combination** of both key columns. The following is an example of a table violating this, second normal form, rule. Note how the values for date repeat in the table. This is because it is not about a given SalesID AND a LineItemID, but only about the SalesID!

****

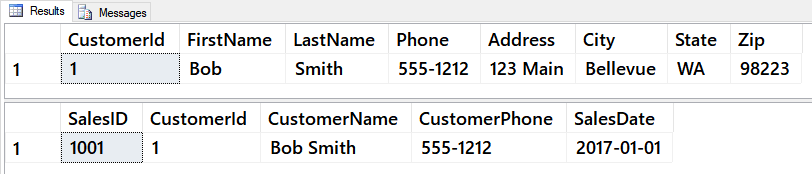
**Figure 14: A table storing repeating sales date information about a SalesID**

To fix this, we **remove the SalesDate column** from the "Sales Line Item" table and place it is the "Sales" table.

****

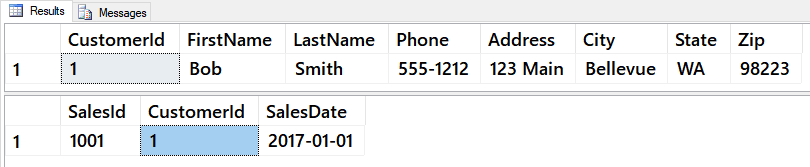
**Figure 15: The repeating sales data information moved to a single value in the Sales table**

Here is an example of a table **violating this rule in a different way**. Note how when a customer's name and phone number is added to a sales table, it is actually dependent on the CustomerID and not the SalesID (which would be it chosen Primary Key.)

****

**Figure 16: Customer data in the Sales table only relates to the CustomerId column**

The correct way for this to be modeled is for **all the non-dependent customer data to moved out** of the sales table.

****

**Figure 17: Customer data has been moved out of the Sales table which contains only the CustomerId column**

## Relationship between table rows of data

Data values can relate to each other with **three basic relationships; One-One, One-Many, or Many-Many**.

Data describes individual attributes or an object or event. Here are two sets of data describing Product information.

{ "ProductName " : "ProdA" } { "ProductListPrice" : "$9.99" }

Looking at this we might decide that each product object has one ProductName and one ProductListPrice attribute. **If you decide that this should be the rule**, then you can say that **ProductName and ProductListPrice have a one to one relationship**.

When **values are about the same object** they can be **grouped into sets (rows)** and these set **combined into an outer set (tables).**

**Products** [

{ "ProductName " : "ProdA", "ProductListPrice" : "$9.99" }

{ "ProductName " : "ProdB", "ProductListPrice" : "$1.00" }

]

**If we add more attributes**, we are careful that the relationship between attributes do not change.

**Products** [

{ "ProductName " : "ProdA", "ProductListPrice" : "$9.99" , "**CategoryID**" : "Cat1"}

{ "ProductName " : "ProdB", "ProductListPrice" : "$1.00" , "**CategoryID**" : "Cat1"}

]

If, however, we see that a relationship between attributes in one to many we need to rethink how we will store the data. **If we make a rule** that **products can belong to multiple categories** which means that the ProductName and CategoryID attributes **no longer have a one to one relationship**!

**Products** [

{ "ProductName " : "ProdA", "ProductListPrice" : "$9.99" , "**CategoryID**" : "Cat1", "**CategoryID**" : "Cat2"}

{ "ProductName " : "ProdB", "ProductListPrice" : "$1.00" , "CategoryID" : "Cat1"}

]

### How this looks in a relational database

When you are **working with a relational database** you can identify these three relationships by comparing the values in one row in a table with values in another row (usually in another table).

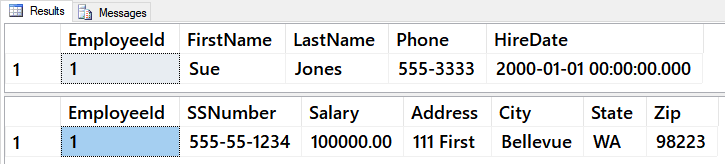
If sounds hard at first, but if you **watch for the patterns** **shown in the following examples** you will easily become good at spotting relationships.

#### One to One Related Tables

One to One related tables are used for performance and security. Here is an example:

SELECT \* FROM Employees;

SELECT \* FROM EmployeeSensitiveData;

****

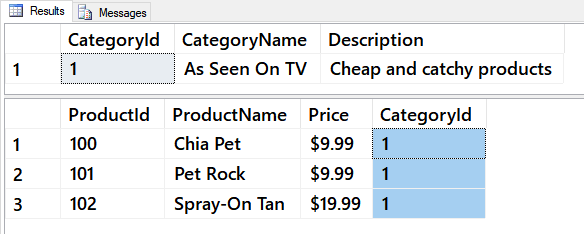
**Figure 18: Two rows of data in a one to one relationship**

#### One to Many Related Tables

One to Many related tables are very common in a relational database. Here is an example:

SELECT \* FROM Categories;

SELECT \* FROM Products;

****

**Figure 19: Two rows of data in a one to many relationship**

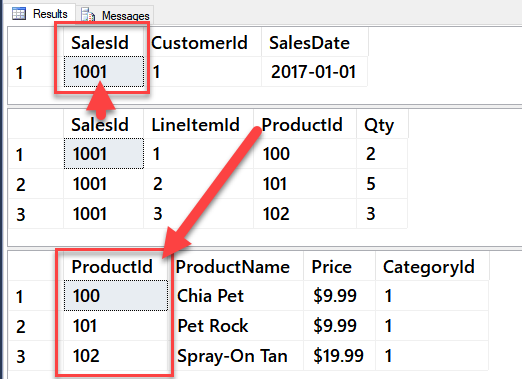
#### Many to Many Related Tables

Many to Many related tables need to be represented by a Junction (Bridge) Table. Consider how **one** sales event would have a **sales ID** and **one or more product IDs** that were sold. However, **also** consider that **one product ID** could relate to **many different sales ID's** for different sales events. Here is an example of a many to many relationship between **products and sales, joined by** a bridge table (also known as an **associated entity**!)

SELECT \* FROM Sales;

SELECT \* FROM SalesLineItems; -- Bridge Table

SELECT \* FROM Products;



**Figure 20: Data in a many to many relationship**

## Lab 3: How would you normalize a customer list? - 30

In this lab, you will consider how customer data could be normalized and turned into one or more database tables.

***Important****: I do not expect you to know how to do this! I want you to think about how you might do this, try out some of your ideas, and talk about it with others. Expect to not be as good at this as someone who has been trained in data.*

**Note**: This lab can be done individually or with a group of up to three people.

### Step 1: Review the data

As we did in lab one, review structure of the customer data provided in the customers.csv file.

1. Open the csv file.
2. Identify the different pieces of data.
3. Name for the different pieces of data.
4. Provide an example of a value for the different pieces of data.
5. Group the pieces data.

### Step 2: Identifying the Entities

Use the groups you found to create named entities.

### Step 3: Create an ERD

Use ERD techniques to create an ERD diagram of these entities and their relationships. Use the "Chen Style" as described in this article, **but do not buy any software!**

<https://www.smartdraw.com/entity-relationship-diagram/> (external site)

You can use MS Paint, PowerPoint, or similar software that you already have, to create your ERD.

### Step 4: Create an Relational Database table

Use your ERD diagram to create a SQL script for the relational database table or tables. You can use the code found in module's Module01Code-Normalization.sql file as a starter. Test your script on either your local SQL Server or on the UW SQL Server.

### Step 5: Review Your Work

Now, you will review your work with your instructor.

## Lab 4: How would you normalize a sales list? - 40

In this lab, you will consider how sales data could be normalized and turned into one or more database tables.

***Important****: I do not expect you to know how to do this! I want you to think about how you might do this, try out some of your ideas, and talk about it with others. Expect to not be as good at this as someone who has been trained in data.*

**Note**: This lab can be done individually or with a group of up to three people.

### Step 1: Review the data

As we did in lab one, review structure of the customer data provided in the customers.csv file.

1. Open the SimpleSalesData.htm file.
2. Identify the different pieces of data.
3. Name for the different pieces of data.
4. Provide an example of a value for the different pieces of data.
5. Group the pieces data.

### Step 2: Identifying the Entities

Use the groups you found to create named entities.

### Step 3: Create an ERD

Use ERD techniques to create an ERD diagram of these entities and their relationships. Use the "Chen Style" as described in this article, **but do not buy any software!**

<https://www.smartdraw.com/entity-relationship-diagram/> (external site)

You can use MS Paint, PowerPoint, or similar software that you already have, to create your ERD.

### Step 4: Create an Relational Database table

Use your ERD diagram to create a SQL script for the relational database table or tables. You can use the code found in module's Module01Code-Normalization.sql file as a starter. Test your script on either your local SQL Server or on the UW SQL Server.

### Step 5: Review Your Work

Now, you will review your work with your instructor.

**NOTE: Unlike assignments, labs do not need to be turned in to Canvas!**