**Files — A Historical Look**

**Content Author:** Terri Washburn, Ph.D.

Nowadays most students are very familiar with the concept of “**files**” and rightfully so, as we use them in our everyday, highly technological lives. But not so long ago — perhaps 25-30 years — files were unwieldy, fragile and expensive ways to store the data that an organization needed to access in order to carry out its goals.

One of the earliest forms of **data storage** was that of the **computer tape**, a large circular plastic frame that enclosed magnetic tape. Each was approximately 12” across and pieces of data were stored on the tape sequentially, with the newest data being added to the end of the tape.

These were used by large **mainframes**, and hung by computer operators, whose job it was to load or unload the tapes as prompted by computer systems.

In this photo to the right, the tape reels are red and blue:

|  |  |  |
| --- | --- | --- |
|  | A close-up of a camera  Description automatically generated with medium confidence | A picture containing text  Description automatically generated |

I bring up the concept of magnetic tape reels because it puts into context just how far we have come in a couple short decades. When data was put onto a tape reel, it was done so **sequentially**, as mentioned above, meaning that each data field on the tape was written adjacent to another field, and the only way to get to a particular field was to read sequentially through the tape.

It was not unlike having to read the first 62 pages of a book to find a word on page 63. The tape reader would spin and the tape would feed through the reader, one inch at a time, until the entire reel was read.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | A close-up of a stethoscope on a white background  Description automatically generated with medium confidence |  | |  | | --- | | For example, imagine having to start reading with “John Adams” all the way through the tape, past every other record in the tape, to find “Alexander Zabot” at the very end of the tape. | |  |
|  | |  |  |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | | **John** | **Adams** | **123** | **Oak** | **Troy** | **MI48084** | **Alice** | **Bart** | **345 Elm...** | Alexander     Zabot | | | | |

**(To read proceed sequentially )**

The **spaces** on the tape were there deliberately, as each field on the tape was given so many spaces to occupy. First name might have 15 spaces, last name perhaps 20 spaces, address 15 spaces, and so on. Programmers had to **define** the **record and field lengths** within the file so that the computer programs knew where to find what field.

Sound complicated? You bet. It was. It was not something that could be changed easily, and this is — by the way — what led (in part) to the Year 2000 (“Y2K”) problem. Dates were a problem, especially years, which used to be stored as “YY” (such as “77” to mean 1977) simply to save space on the tape. Even when we moved to disk storage, the practice of storing a year as YY was continued. This meant that when we needed to add the century (ie, 20) to dates (as opposed to assuming it was 19) fields could not be made larger and new files had to be made (via Y2K conversion projects) to add the century, so that “77” became “1977” or "01" became "2001".

Keep in mind these reels of tape **were** the company’s data files. This was where it was kept — particularly files that needed to be moved from location to location. Some files were stored on what hard drive space was available, but as far as moving data from location to location, there were no networks, no wi-fi, no USB or disks, no cloud computing. There was little aside from magnetic tape — and of course, paper printouts.

We’ll be discussing all of these this week.

**Data Storage Considerations**

**Content Author:** Terri Washburn, Ph.D.

Up until now we have been concerned with the flow of data, the processing of data, the actors or entities involved with the data, and the display of data on the user interface(s). Now we begin to think about how we will **store** data, and how it will be **accessed** by the programs that are involved in our systems project. We will begin to consider **data storage design**.

Dennis, Wixom and Roth (2012) note that data storage design asks that you:

* Select the data storage format;
* Convert the logical data model created during analysis into a physical data model to reflect the implementation decision;
* Ensure that DFDs and ERDs are in alignment, and
* Design the selected data storage format to optimize its processing efficiency.

|  |  |
| --- | --- |
| There are two ways to store data that we will discuss, one of which was already mentioned in Lecture 1 this week: | **1. Data Files** |
| **2. Databases** |

The difference between the two is both subtle and not-so-subtle. We will discuss each in turn in the next two lectures, but for now, think of them this way:

|  |  |
| --- | --- |
| Graphical user interface  Description automatically generated with medium confidence | A data file can be one collection of related information (or data) that can **stand alone**. If you have a contacts list on your phone, it is a file: it has everything in it you need to make the phone calls.  A data file is a separate, discrete data storage repository just for that information you need for calls (likely, names and area codes and phone number(s) for each entry in the file). Each entry in the file would be called a **record**, in data-file lingo. |
| Engineering drawing  Description automatically generated | A **database**, on the other hand, is also data that is related and stored electronically **but** databases are optimized for processing by grouping together collections of data that are **related** to one another in some way. Each collection of data is known as a **table**, and each entry in the table is known as an **instance**. (Note that "file" and "table" are sometimes used interchangeably and 'record' and 'instance' are also used interchangeably although they are not - technically speaking - exactly the same. But in computer lexicon we sometimes use terms loosely, so be aware that a file and table store off records or instances.)  So to continue our previous example, when thinking about a database, if your phone contacts list is one file (or table) in a database, an address list might be another file (or table); or perhaps a birthday table about your contacts (who are probably your friends and family for the most part). |

Confused? Let's move on and try to clarify things, if so. If not, good for you, and continue along to learn more!

**Data Files**

**Content Author:** Terri Washburn, Ph.D.

**Data files** are repositories that hold data for computer processing. We all use data files when we open up MS Word or other word processing application on our laptop and create and store off data (or information). When you type up an assignment for school and save it in MS Word, you have created a **file**. The file contains **data** (the characters and numbers and images you utilized) as well as (hopefully) information, in that you arranged or organized the characters, numbers and images to mean something. Data files can also be image files (.jpg or .png format, for example) or sound files (.wav) and so on. **Data files generally contain records or bits of records that are related to the other content in that file.** In an image file, the pixels would be the pieces of data that when pulled together create an image. In a Word file, it would be the letters, characters, and so on. Or a file might be a paper report file, generated by a computer.

Because this is an IT (Information Technology) course, we naturally are referring to electronic data files, not paper files. You are familiar with many types of data files, no doubt:

Graphical user interface, application

Description automatically generated

Keep in mind that data files can be both **input** and **output** in nature. What is generated from one computer application (say an output email) may well become an input in another application (another email application). An Excel file can be read into SPSS (statistical processing software), for example. So, of course, can an SPSS data file, or .sav file.

Graphical user interface, text, application

Description automatically generated

The previous paragraph is at the heart of computer processing. Output from one source is read into another source as new information (or printed or displayed) to be further processed or utilized. It may be used electronically, or it may be an end-user report or display screen.

If you were to "dump" a data file (the term for printing it out or displaying it so that you can look at it by eye), it would likely be unintelligible. Data by itself often can't be interpreted. Viewing three records of one data file may look something like this:

A screen shot of a game

Description automatically generated with low confidence

Of course at a glance we cannot make sense of it. We may try and we may actually come up with good guesses at times. XF112 looks like an identification number. The other numbers beside it (2456654) could be just about anything, as could 01, SA, and 35. But once we **define the record** it puts it into context for us, thus making **information** for us.

**Let's assume you were told the fields for the above record were, respectively:**

* Part number
* Units sold in 2014
* Division within a world-wide region
* World-wide region of the company
* Increase (in percentages, to one decimal point) over the previous year

**Now we have information! Now we can say that:**

|  |  |
| --- | --- |
| Text, application, chat or text message  Description automatically generated | Diagram  Description automatically generated |

We only know that because it has been put into context (defined) for us. Again, the above **data** was stored in **data fields** in **records** within a **data file** ... and became **information** once we defined it and put it into context. Note that **XF112 2456654 01 SA 35** takes up much less space than did the paragraph in the blue thought bubble, above. Data storage used to be expensive and only the least amount of space necessary was used. We could, for example, have stored PART\_NUMBER=XF 112 UNITS\_SOLD=2456654... and so on right in the record, but as long as we **defined** the data field, there was no need to be repetitive - so we only stored the data value of each record, instead.

There are other ways of getting information from a data file, aside from putting it in context or defining it. One other way is to *create* it from the data in the file: adding numbers, performing a calculation, or joining words together, ultimately generating information or *new information*.

**A Historical Look**

|  |  |
| --- | --- |
| A group of people in an office  Description automatically generated *Image source: https://tinyurl.com/y73txjgy* | The historical, early days of IT were full of disconnected data files with data fields that were repetitive in nature and ill-planned. They were added to the IT department as needed, without any real plan behind using technology to support strategy. Over time, it became apparent that disparate data files were not sufficient or adequate for most organizations. Data files were fragmented views of the organizations and what organizations needed were cohesive perspectives of all the data: planned, carefully thought-out, architected (in the IT sense of the word) - what organizations needed were **databases**.  Let's move on, then, to discuss **databases**. |

# Data Files versus Databases

**Content Author:** Terri Washburn, Ph.D.

In the previous lecture we discussed **data files**, which are repositories (also known as data stores) that contain **records** of information that are related to the other information in that file. In early computing days, all pieces of data were stored in files (versus databases, which we will get to in a few paragraphs). At those times **functional areas** of an organization each maintained their own files.

For example,

* The marketing department kept marketing files,
* The sales department kept their own sales files,
* The human resources department had HR files, and
* ... so on

Diagram

Description automatically generated

The data fields in each file were usually related to other fields in the same file. A Parts File held data on parts; an Orders File held data on orders. Every department kept its own files, and their work processes were based upon those files. This meant each department controlled its own data, since they had control over the collection and storage of that data (which ultimately became information for their departments). This sounds like an advantage for the organization, doesn't it?

It really wasn't. It was an advantage for **that department** but **not for the organization**. Stop and think about this. Why was it an advantage for a department but not the organization as a whole?

[Click to see answer](javascript:setContent('ans1',%20'link1');)

It also meant that the departments or organizational units were not always aware of what data had been collected and stored. Thus, organizational strategy and planning was short sighted. Although perhaps this doesn't **sound** like an earth-shattering problem, in reality - and in certain contexts - it definitely could be. In healthcare organizations, for example, you want all your patient information to be consistent among the different organizational units. You don't want different information in the lab files, the doctor files, the medical records files, or surgical files, for reasons I am sure you can well imagine.

|  |  |
| --- | --- |
| Frankly, **data files** belonging to functional organizational units was (and still is) an unorganized and sloppy way of keeping track of data. What is a better way?  Let's look at this image for an idea.  Do you notice that the customer **file** is centrally located and the **departments**interact with the file, and each department doesn't simply have its own customer file? There is only **one** place for the customer's name and address - not multiple places. Each department accesses customer information via the same file. | Diagram  Description automatically generated |

### For consideration!

Why doesn't "human resources" have an arrow going to customer file? For the simple reason that the Human Resource department may not need to use the customer file for any reason! Databases can control access (by need) easily.

This is the simplicity, the beauty, and the genius of databases. We will discuss continue our discussion of databases in the next lecture. Click to continue.

**Databases – Part 1**

**Content Author:** Terri Washburn, Ph.D.

***A database is a “collection of information that is organized so that it can easily be accessed,  
managed, and updated (Rouse, 2014)”***

Databases are an organization’s way of arranging data so that it is optimized for processing with a minimal amount of redundancy or duplication. When databases were first introduced they were a paradigmatic shift in the way of each functional department’s way of thinking. Instead of keeping each department’s separate and distinct (yet possibly duplicated) data files, databases **centralized** the storage of data and made the functional departments access a single storage point: the **database**. A way of looking at it is that the department's “circle,” the database repository, represented in the center of this image below.

|  |  |
| --- | --- |
| A picture containing white  Description automatically generated | **Why was the concept of a database a paradigmatic shift?**  Primarily because functional units had to give up ownership of files and collaborate with other departments in the design of a single, unified data repository that serves the entire organization. Many felt that turning their files over to the IT department to manage was in a sense ‘losing control’ of their information.  Note that the data is central to all the functional units or departments in the organization, as represented by the circles of people. Each department uses computers to access the central data – or database. Further, each department may only be able to access **part** of the database – its own data.  Icon  Description automatically generated |

**A database is not an actual *thing***. It is actually a collection of file definitions, storage areas, and the arrangement of the data that comprise the concept we call a database. You can’t pick up a database and carry it around, for example. You ***may*** be able to pick up the computer(s) on which the database is housed, and you may certainly carry a **database schema** with you and reference it. Computer applications can write to databases and read from them. Databases can be backed up (saved) to an external drive or storage area. You can use the data / information stored in the database. But really, a database is an abstract concept.

A database is comprised of **tables** (which are files) of information that have been optimized for processing. What makes it different from a group of separate files, such as those discussed in lecture the previous lecture, is that there is a concerted effort to eliminate redundant and duplicated fields among the tables. Here is an image of a database **schema**. Tables (often casually referred to as “records”) contain data that is related to other data in the table.

Diagram

Description automatically generated

A database schema is a visual rendition of a grouping of tables. It exists to show the fields, the attributes, and the interdependency among the tables. Schemas are used by data modelers to represent how the data will be stored - and accessed. The names in bold in the schema above are the names of each table. Beneath each are field names. The small key icon indicates a primary key (unique identifier) for that table. The lines show how the tables connect; i.e., you can read the **OrderDetails** file by using the OrderID – which happens to be a primary key in both the **Orders** and the **OrderDetails** file.

**For consideration!**

How is a table different from a file, you may wonder? For the purposes of this class, there is not much difference. I refer to “files” when I am talking about a stand-alone repository of data, such as Word file. I will say “table” when I’m referring to (file) components of a database. Each houses (holds) data in defined fields. The big difference is that tables in a database are systematically and carefully planned to be very “lean” – optimized so that there is no overlap of data being stored, and set up so that the tables can be linked to put together information in a way we need it., efficiently and effectively.

Let’s begin to discuss what I mean about keeping tables lean and being able to link them to get all the **information** from them that you need.

**We’ll go into more detail later, but here is a visual image to get you thinking about it. Below is an example of a file – not part of a database. Here is the accompanying scenario.**

Assume we have a customer who places an order for a part and wants it shipped to her home. The information surrounding this entire **event** or **transaction** could be saved off in a single data file like this, which I am naming “Order Table” because I will next integrate it into a database. The file (or table) is **defined** on the left and an example of the data that could be in **one record** is on the right:

|  |  |  |
| --- | --- | --- |
| Table  Description automatically generated | **The one record might look like this:** | Application  Description automatically generated with medium confidence |

When you review the above table and record, consider the amount of space that we use to store off the information – not to mention how often the information is duplicated. Also consider that the part number, for example, which is **C134** in the sample record above and which has a description of **Aquarium Filter 3-Pack**, is probably stored somewhere else in the organization.

* Where else might the part number and description be stored-? A parts file is one place. Perhaps in an inventory file.
* What about a customer order history file (or table – remember, I am using the terms rather loosely interchangeably at this point).
* And how about the customer’s name and address and phone number? Where else might they be stored?

When we look at a single **event** or transaction such as the one above, and all of the information that is captured and stored about the event, and (because we later may have to report upon it) we think about **why** we store it in such detail, it is easy to see why files can become large, unwieldy and very duplicative. Enter: the database. We are going to take the above single file and turn it into part of a database…in the next lecture.

# Databases - Part 2

**Content Author:** Terri Washburn, Ph.D.

Let's take a look at a file that an organization might have used before it began to utilize databases. Consider this sample orders file or table, below. Note that there is much information in the file: the order number, the customer's name and address, what the customer ordered, and more. And yes, everything in this file is related to the order in some way or another, so why is it important that we reconsider the use of this single file and instead move to storing the data in a database?

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  | | --- | | **Order Table** | | OT\_Order\_ID | | OT\_Cust\_ID | | OT\_Cust\_FirstName | | OT\_Cust\_LastName | | OT\_Cust\_Address1 | | OT\_Cust\_Address2 | | OT\_Cust\_City | | OT\_Cust\_StateOrProvince | | OT\_Cust\_Zip | | OT\_Cust\_Country | | OT\_Cust\_PhoneDay | | OT\_Cust\_PhoneEvenings | | OT\_Cust\_OrderDate | | OT\_Cust\_Part\_Ordered | | OT\_Cust\_Part\_Description | | OT\_Cust\_Part\_Cost | | OT\_Cust\_Order\_Tax | | OT\_Cust\_Order\_Shipping | | OT\_Cust\_Order\_Total | | OT\_Cust\_Ship\_To\_Addr1 | | OT\_Cust\_Ship\_To\_Addr1 | | OT\_Cust\_Ship\_To\_City | | OT\_Cust\_Ship\_To\_StateOrProv | | OT\_Cust\_Ship\_To\_Zip | | OT\_Cust\_Ship\_To\_Country | | Theoretically, there is nothing wrong with storing **all** that information in one Orders Table (or file). In fact, for years organizations operated that way. But if the customer's name and address are in the Orders Table, how will the Marketing Department or the Billing Department access it? Why would they look for customer data in the Orders Table?  Does the Orders department "own" the customer's data? Worth Considering: If the customer's name and address doesn't "belong" in an Orders Table, then where does it belong?  Doesn't the customer information belong in a customer file, or table?  Absolutely!!!  This is extremely important to systems analysis and design: **the data must be designed for optimal processing and retrieval**. This is the role of the systems analyst and database designers. Remember, we use databases to store related data, organized so as to be easily accessed and designed so that each table only contains what is needed **in that table**. This is called "normalizing" the data. This means each table (we will begin calling them tables now, not files, as we're talking about databases) - each table will only contain the information that is absolutely related to other data in that table and a field or two (usually a key field) that is needed to 'join' that table with another table.  What we will do now is examine the Orders Table (that which I called a file, above) and create a database from it. We are going to divide this large, unwieldy Orders Table into smaller tables - each of which will have a particular, discrete set of data in it that is completely related to other fields in that table. Now later, if we need all this information pulled together to create a report or a display screen, for example, we will use a **query language** (such as SQL) or other programming language to pull the pieces of data out of all the tables in the database and 'join' the tables to create our output. |

Let's look again at the information in the table. We have information about:

* The order
* The customer
* The part that the customer ordered

Let's start with that much. And let me place here the Orders Table, or file, from above with the **order information** highlighted in blue.

|  |
| --- |
| **Order Table** |
| OT\_Order\_ID |
| OT\_Cust\_ID |
| OT\_Cust\_FirstName |
| OT\_Cust\_LastName |
| OT\_Cust\_Address1 |
| OT\_Cust\_Address2 |
| OT\_Cust\_City |
| OT\_Cust\_StateOrProvince |
| OT\_Cust\_Zip |
| OT\_Cust\_Country |
| OT\_Cust\_PhoneDay |
| OT\_Cust\_PhoneEvenings |
| OT\_Cust\_OrderDate |
| OT\_Cust\_Part\_Ordered |
| OT\_Cust\_Part\_Description |
| OT\_Cust\_Part\_Cost |
| OT\_Cust\_Order\_Tax |
| OT\_Cust\_Order\_Shipping |
| OT\_Cust\_Order\_Total |
| OT\_Cust\_Ship\_To\_Addr1 |
| OT\_Cust\_Ship\_To\_Addr1 |
| OT\_Cust\_Ship\_To\_City |
| OT\_Cust\_Ship\_To\_StateOrProv |
| OT\_Cust\_Ship\_To\_Zip |
| OT\_Cust\_Ship\_To\_Country |

These blue rows are the fields that absolutely must be in the order table so that we can fulfill the customer's order. What about information about the customer? Does it need to be in the order table?

Yes, of course, but not **all** that you see at left. What we need in the order table is the customer ID - and from that, we can extract all customer information (in pink, below) from a customer file.

**So let's split the order table into an order table and a customer table, to start.**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  | | --- | | **Order Table** | | OT\_Order\_ID | | OT\_Cust\_ID | | OT\_Cust\_OrderDate | | OT\_Cust\_Order\_Tax | | OT\_Cust\_Order\_Shipping | | OT\_Cust\_Order\_Total | | OT\_Cust\_Ship\_To\_Addr1 | | OT\_Cust\_Ship\_To\_Addr1 | | OT\_Cust\_Ship\_To\_City | | OT\_Cust\_Ship\_To\_StateOrProv | | OT\_Cust\_Ship\_To\_Zip | | OT\_Cust\_Ship\_To\_Country | | |  | | --- | | **Customer Table** | | CT\_Cust-ID | | OT\_Cust\_FirstName | | OT\_Cust\_LastName | | OT\_Cust\_Address1 | | OT\_Cust\_Address2 | | OT\_Cust\_City | | OT\_Cust\_StateOrProvince | | OT\_Cust\_Zip | | OT\_Cust\_Country | | OT\_Cust\_PhoneDay | | OT\_Cust\_PhoneEvenings | |

Here is the orders table, in blue above, and our new customer table is shown in pink, above. To split the tables, we simply defined two new tables, and placed the data fields from the one large table into the smaller, more discrete, distinct tables that each store data about one concept (the orders or the customer). Those fields will come out of the Orders Table.

|  |
| --- |
| Worth Noting! |
| Because the order table contains the customer ID, we can get all the customer information (address, etc.) by simply 'joining' the two tables, or by 'reading' the customer table using the field "OT\_Cust\_ID". Note that both tables have a field "Cust\_ID". Don't worry yet about the prefix 'OT' or 'CT'. Those identify the tables. |
| Icon  Description automatically generated |

What about the items (the "part") the customer ordered, or perhaps "parts" (more than one)? Let's create an **orders detail** table that will contain one instance of the table (i.e., one record of the file) **for each part** the customer may have ordered. It would look like this (below):

|  |
| --- |
|  |

|  |
| --- |
| **Order Details Table** |
| ODT\_Order\_Details\_ID |
| ODT\_Order\_ID |
| ODT\_Part\_ID |

Even though we only see one definition of the Order Details Table here, if the customer had ordered more than one part, there would be more than one **record** in the table for that order.  
  
**Confusing?** We'll discuss more in class!

And finally, in order to fill the customer order, we will use the order\_ID to read the order details table, and then use "ODT\_Part\_ID" to read the parts table, which we will create to look like this:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| |  | | --- | |  |  |  | | --- | | **Part Table** | | PT\_Part\_ID | | PT\_Part\_Description | | PT\_Part\_Cost | | PT\_Part\_Location\_Zip |   It's important to realize that there could be **many instances (i.e., many records)** of the orders detail table; in fact, there would be one instance (one record) for each **part** the customer ordered. |

Now let's put it all together in the next lecture. You may wish to print off this lecture while you read the next lecture, or open the next lecture in a different window so that you can tab back and forth between them. I realize this is a tough concept if you're new to I.T. and even more so if you are new to database design. We will cover it more in class - but do re-read this lecture for more clarification, too.

# Databases - Part 3

**Content Author:** Terri Washburn, Ph.D.

Let's continue our journey of defining and normalizing database tables from the larger Orders file in the previous lecture. I want you to note two things before we continue:

* 1. There are likely other ways to arrange this **database schema** - for that is what is called - depending upon the skill level and experience of the analyst or data modeler. For our purposes this **schema** will work fine for us. It is also not yet completely **normalized** - we will refine it in a bit. We are not pros at this! We are just getting started.
  2. We need to introduce the concept of **primary and foreign keys** before we go any further.

|  |
| --- |
| Primary Key A picture containing text  Description automatically generated  **A primary key is a field in the table that is unique for each record in (instance of) the table.**  This means that customer John Smith has a unique primary key, or field, whose value is different from that of Susan Brown, and Dan Martin, and so on. A social security number is a great example of a primary key. So is a telephone number or an email address. They are unique and can only belong to one entity.  In our customer table, we've created a "customer ID" that will be unique for each person in the table. We denote primary key by (PK) beside the field name.  **Click to read more about primary keys:** <http://www.techopedia.com/definition/5547/primary-key>  Image source: http://www.accessallinone.com/how-to-set-a-primary-key/ |

|  |
| --- |
| Foreign Key **A foreign key is a field in the table that is not necessarily unique in that table - but that is unique (that is, is a primary key) into another table.**  If you look at the Order Details Table below, you'll see a field ODT\_Part\_ID. This is a unique part number for some product. But many customers may have ordered part ODT\_Part\_ID = 1234, for example. This part ("1234") exists just one time in the Parts Table. It is unique (or a primary key) there. It is not unique in the Orders Detail Table so it is not the primary key in the Orders Detail Table. Why not? Because many people can order that part.  So... it is a primary key into the Part Table (where it has one record only) and it is a foreign key in the Orders Detail Table (where it may be in many records). We denote foreign key by (FK) beside the field name.  **Click to read more about foreign keys:**<http://www.techopedia.com/definition/7272/foreign-key> |

Calendar

Description automatically generated with medium confidence

**Now let's put all these tables together and talk through a scenario. I'll bullet the scenario for readability.**

|  |  |
| --- | --- |
| Scenario  * Customer Ronald Adams has customer ID 12345 * He placed an order (order ID = 1031) for two parts for his furnace on November 10, 2014. * The two Part IDs are 667 which is a 2"coupler ($4.50) and 712 ($8.00) which is a mesh tube. * Ronald wants them shipped to his home address at 123 Oak Street, Detroit, MI, 48226. * His billing address is the same as the shipping address. * Both parts are located in a warehouse in Warren, MI, with a zip code of 48088. * Michigan's sales tax is 6%. * Shipping is a flat $5.00 per order. | Graphical user interface, application  Description automatically generated |

Text

Description automatically generated Table

Description automatically generated

Can you begin to see how this all fits together? How the discrete (separate) tables with their related data can be connected (or joined with) other tables to get the big picture of Ron's order? Yet there is little duplication (redundancy) in any of the tables.

Now that instances of the tables have been created to show Ron's order, we can create an order invoice or a report - or whatever else we want to create - by "joining" the tables using the appropriate programming language

# Databases - Part 4

**Content Author:** Terri Washburn, Ph.D.

Let's revisit our scenario and **create an invoice** from the tables and the instances of the tables we created. For each bullet, refer to the table below. You must understand how these tables interact.

* Customer Ronald Adams has customer ID 12345
* He placed an order (order ID #1031) for two parts for his furnace on November 10, 2014.
* The two Part IDs are #667 which is a 2"coupler ($4.50) and #712 ($8.00) which is a mesh tube.
* Ronald wants them shipped to his home address at 123 Oak Street, Detroit, MI, 48226.
* His billing address is the same as the shipping address.
* Both parts are located in a warehouse in Warren, MI, with a zip code of 48088.
* Michigan's sales tax is 6%.
* Shipping is a flat $5.00 per order.

Let's mock up an invoice using colored arrows to show how the data tables relate to one another, and where the data come from. First, let me put a smaller image of all the files below:

Table

Description automatically generated

## Creating an Invoice from Our Database

Now let's consider how the data are connected (related) in these tables within this database, using colored arrows for clarity. You will be able to follow along without the arrows, but they may be of assistance to some of you:

* + Order #1031 (orange arrow) was placed by customer ID #12345 (red arrow) The Order Table contains information we'll need for our invoice (order date, tax, shipping info)

* + Note that "OT\_Cust\_ID " = 1234 in the Order Table, and it's a **foreign key (FK)**

* + We use the value in this **foreign key** ("1234") to read the Customer Table where **CT\_Cust\_ID=1234**

* + We locate all the information about customer #12345 (by the red bracket) for our invoice

* + Using "OT\_Order\_ID" as a **FK** into the order detail table, we find the parts (purple arrows)

* + There are two instances of the Order Detail Table, meaning this customer has two items (parts) in his order (part 667 and 712).

* + These are indicated by the blue double-headed arrow. Now we have parts information for our invoice.

We probably have enough information for an invoice now. The next link in our navigation will show you how the invoice might look.

And... The link following that one is an exercise (a challenge to you) to find another place you can normalize the above tables; that is, what other fields are redundant and can be eliminated? And how?

**From Database to Invoice**

**Content Author:** Terri Washburn, Ph.D.

Table

Description automatically generated

|  |  |
| --- | --- |
| Shape, arrow  Description automatically generated | **Spend some time reviewing the database and the related invoice to see how it all came together!** |

Table

Description automatically generated with medium confidence

**Ungraded Data Normalization Activity**

**Content Author:** Terri Washburn, Ph.D.

Though we finished our exercise in normalizing the orders table database in the previous few lectures, there remain yet a few things that we could do to **further normalize** the tables. Let's try to find one of them in this ungraded exercise. This is not a typical exercise; I just want you to "think along" with me!

Table

Description automatically generated

|  |  |
| --- | --- |
| What fields do you see duplicated in the Order Table and the Customer Table (aside from the keys)?  **Hint:** For purposes of this example, assume the customer's address and the ship-to address are the same. Now do you see duplicative data?  [Click to reveal:](javascript:showAnswer('ans1');) |  |

One more thing for you to consider and that is whether or not we are putting data into a couple of the tables that simply doesn't need to be there because it can be stored off - once - in its own file. Can you think of what I might mean?

|  |  |
| --- | --- |
| **Hint:** Look once again at the customer's name and address fields (in both Order and Customer files). What do we absolutely have to have in order to get the products to the customer? And what can we instead extrapolate from another table?  [Click to reveal another hint:](javascript:showAnswer('ans2');) |  |

So, one last modification to our order and customer tables, and then we're done with the week's material. Let's add a table for zip code, and remove city and state

Table

Description automatically generated

Note that I have placed a strike-through on each field (or attribute) that we **will not need** in these tables once we add a zip code table. A zip code table would need a primary key which would be: what? The zip code itself! It would need a city field and a state or province field, and a country field. It might look like this:

Table

Description automatically generated

Remember that this added bit of normalization makes zip code a **foreign key** in the customer and orders tables, as shown by the (FK) beside each field.