Module_2 - Relationship Diagram for Data Analysis & SQL

2.1 Introduction and Background

ERD stands for **Entity Relationship Diagram**. An ER diagram illustrates the logic within a database and how individual components relate to one another (Visual Paradigm). Similar to a story board for film, an ER diagram lays out the wireframe of internal database structures to aid in understanding, improving efficiency, and debugging logical errors. The key pieces of an ER diagram are the entities, attributes, and the relationships between them. For example, within the complex systems of Amazon, they would have an entity for customer listing attributes such as name, address, payment, and email. Once an order is placed and ready for shipment, the shipping database would access the customer database to retrieve information to complete the order and ship out the package. An ER diagram would show how the shipping department can access many customers within the database, and the information it would require to complete shipment.

In the 1960s and 1970s, there was a rise in data modeling needs and published works on different models. Peter Chen is attributed with inventing and promoting the entity-relationship model stemming from a paper he published in 1976, "The Entity-Relationship Model – Toward a Unified View of Data" (Kempe, 2013). As databases and businesses grew more complex, a unified modeling system became more important so that structures could be modeled across industries consistently. After the original ERD was developed by Peter Chen, many of his colleagues adapted and made small alterations to improve and simplify the overall model. The key components are still present; the look has been simplified to accommodate more complex systems. Below we see examples of both the original and newer ERD models.

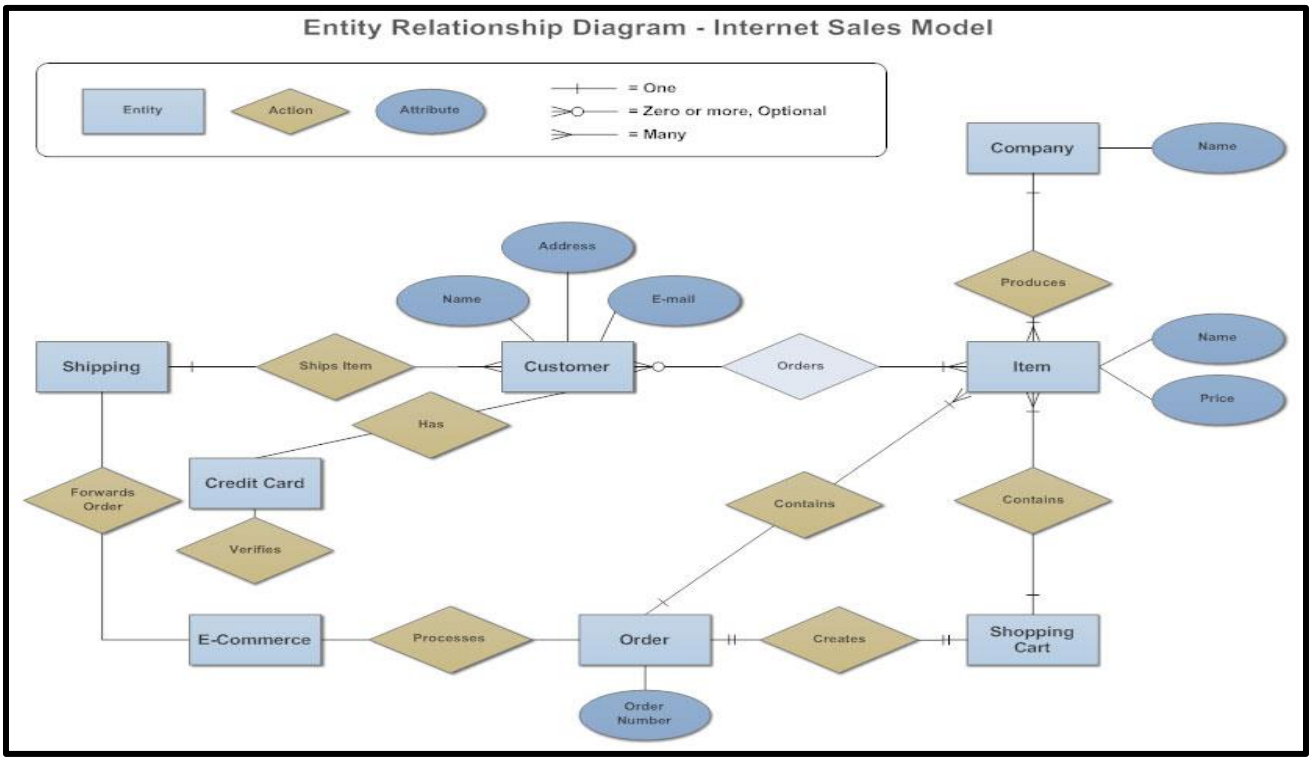

Figure 1 – Original Style ERD (SmartDraw)

(1) This is an example of an older ERD version depicting e-commerce Company.
We can tell because there are action diamonds between entities, and attributes are each listed separately in individual ovals.

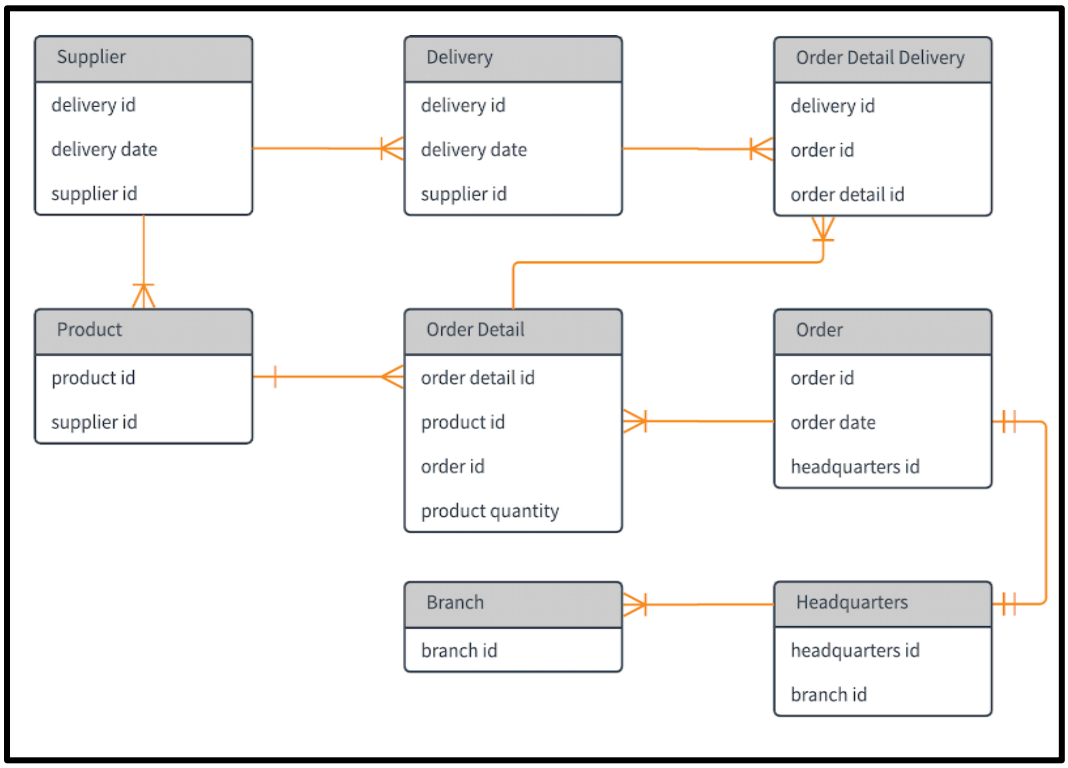

Figure 2 – New Style ERD (Lucidchart)

(2) In this newer ERD e-commerce example, we see attributes listed within each entity, and no actions are present. If depicted in the older model, we would see 21 attribute ovals and a minimum of 8 actions. This would create visual clutter, making the ERD harder to examine, detracting from its usefulness.

2.2 Understanding the importance of data modeling

Keeping a well-documented model of a database is important for overall business efficiency. ER diagrams are easy to use and understand, allowing for simple cross-departmental understanding. As new changes or expansions are proposed, an understanding of current workflow and capabilities is key to managing these projects. Also, as new team members join, an ERD allows for quick and efficient training. New and current team members can revisit this often as a reference while working on individual pieces to understand the overall effect.

While the foundation of an ERD is to show the relationship between entities, they can also be helpful in other instances. Designing and debugging databases can be very complex tasks, but having ER diagrams can simplify the process and reduce the chance of error. In designing an entirely new system, an ER diagram can aid in planning the logic and requirements of a database before any part of it is built. This can save money and hours, and it ultimately cuts down on redundancies and time spent on unneeded builds. Visualizing the overall structure allows programmers to locate any errors in logic or flaws in the overall design before creation. Once a system is already created, an ERD is still useful in debugging as errors occur. Because developers can see how all entities and attributes are linked, it is easier to trace down where any error stems from. In fixing the issue, the ERD also helps in seeing if any side effects may occur (Visual Paradigm).

ER diagrams are especially useful in organizations with bigger and more complex database systems that have many different entities, data points, and relationships between them. A good example of this can be found in any major hospital. All hospitals have doctors and patients, but the larger ones deal with doctors across different specialized departments and all the components that interact. When a

patient makes a yearly checkup appointment online, their appropriate insurance is checked and logged into their profile. When they arrive, the doctor is able to access all previous medical records and current medication, even if prescribed by a different doctor. Any samples taken are processed by the laboratory, who have editing access to the patient profile in order to update the results.

The information is then posted to the account, and both the doctor and patient are notified. Patients have view-only capabilities, while doctors can edit and add on prescriptions as needed. If a prescription is added, the in-house pharmacy is able to view and notify patients when filled. While this process is going on, the billing department is able to see activities in all other departments as it relates to the individual case, viewing insurance information to forward and process bills. They are able to communicate between both the insurance provider and the patient to fulfill total costs.

In this example, the process is not linear, and each department has different levels of access and editing capabilities. Not included in the example, but still present in the situation, are all of the back-end human resources for the hospital as an organization with employees. This becomes increasingly complex as more patients, doctors, specializations, and privacy concerns are added into the mix. In health care professions where certain situations can literally be life or death, efficiency is of the utmost importance. ER diagrams are essential to keeping hospital databases well-organized and free from error. While the figure below depicts a simplified and basic interaction with a doctor's office, we can already see many entities and complex relationships between them. A full-service hospital ER diagram would be much larger.

2.3 Naming and Definitions

Now that we have covered what an ER diagram is, let's go into more detail regarding the finer points of data modeling. ER diagrams are used to visually represent data relationships in many settings, but they are most commonly used in business environments. These data representations allow many

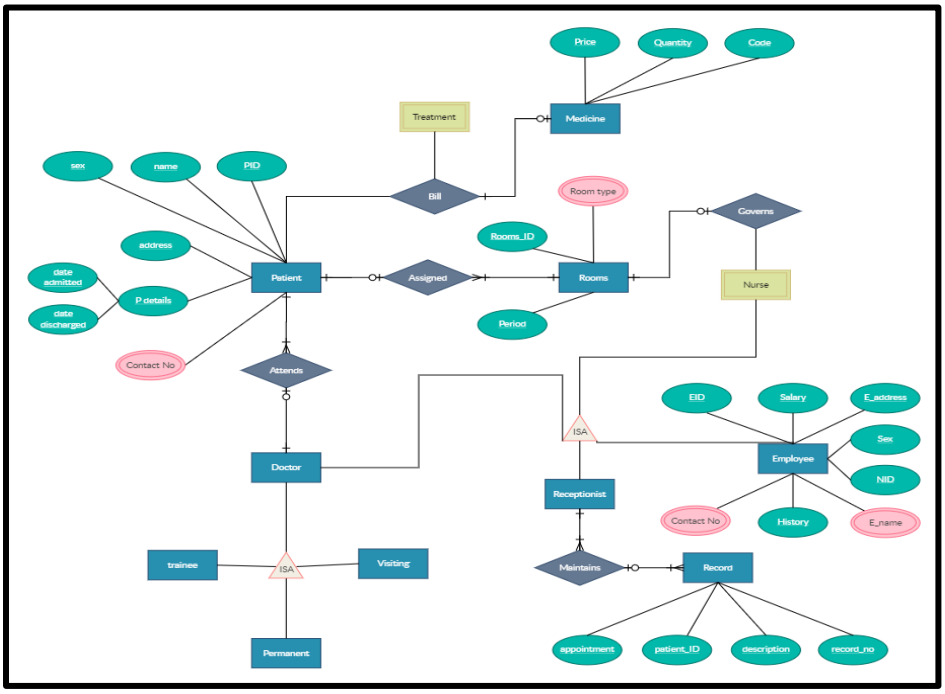

Figure 3 – Basic Hospital ERD (Createrly, 2008)

companies to better use their resources and make well-informed decisions. These diagrams should be easily representable and understandable by a wide majority of employees, as they represent data and/or inner functions of the company. Therefore, it is obvious that such diagrams must be easily readable and functional to allow quick decision-making.

Speaking of readability, in these ER diagrams, naming values and objects is half the battle. Something that shares this need is programming, as creating variables and methods with explanatory yet short names helps to increase readability of the code. Understanding this, writing ER diagrams brings many similar concepts, so writing an ER diagram is nearly the same as writing the code that drives it. The value of self-explanatory variable names cannot be understated, as they help keep the company and its employees informed. However, explaining such concepts without an example is needlessly abstract, so let's start by making a few tables and bringing them up to snuff.

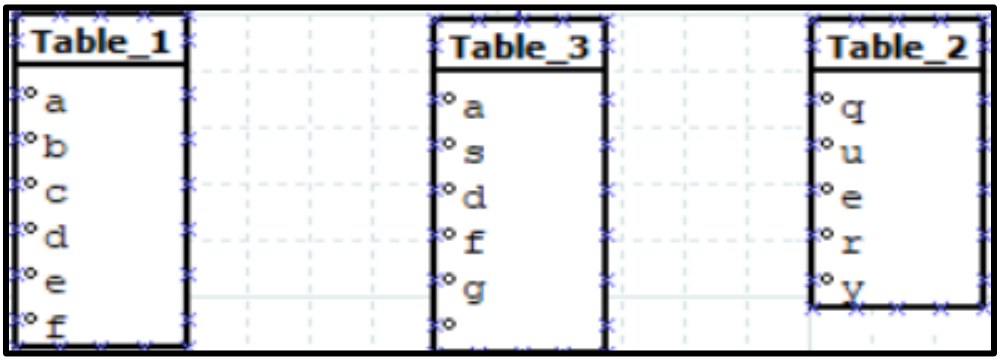

Figure 4 – Entity Relation Tables (Bartz, 2020)

Now with these tables, we can start to fix problems, but first let's start with some baselines. Firstly, we can see an empty value in table 3, and with no relations, table names, or variable names these tables do not convey any meaningful information. Choosing an example set of data, we can display the relationship between a Student, a College, and their Professors.

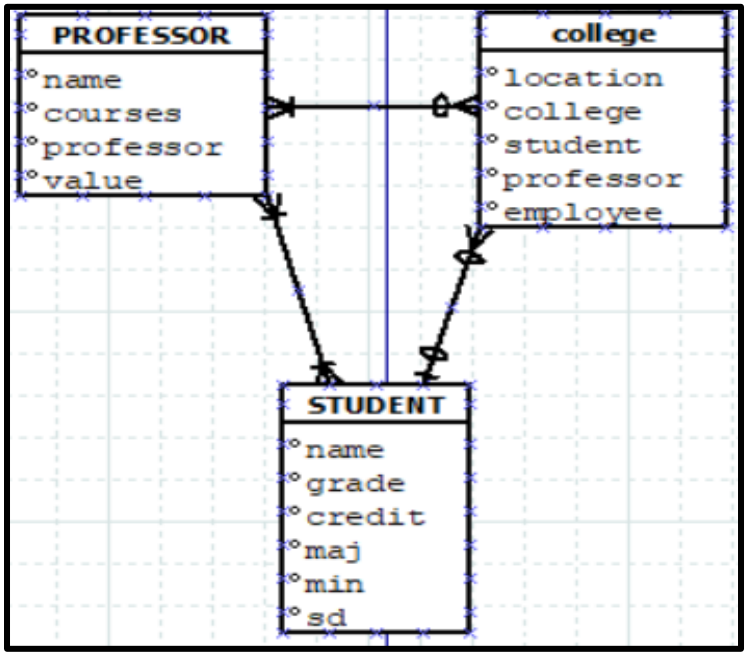

Figure 5 – College, Professor, Student ER Diagram (Bartz, 2020)

Now this table is still far from ideal. Tables share variables, and these names are terrible, and would require a large amount of explanation to make sense. Let's start fixing it step by step. Our first objective will be getting the data to a more readable state. The variable names currently in the table are atrociously low on elaboration or are so general that anything could fit where they are. Other variables are even repeated, so if they were used as a unique identifier or a primary key the program would give irrelevant or hindering information. This would lead to many problems, both in coding and in viewing. Let's make some edits to the table to better represent the variables that are in there right now.

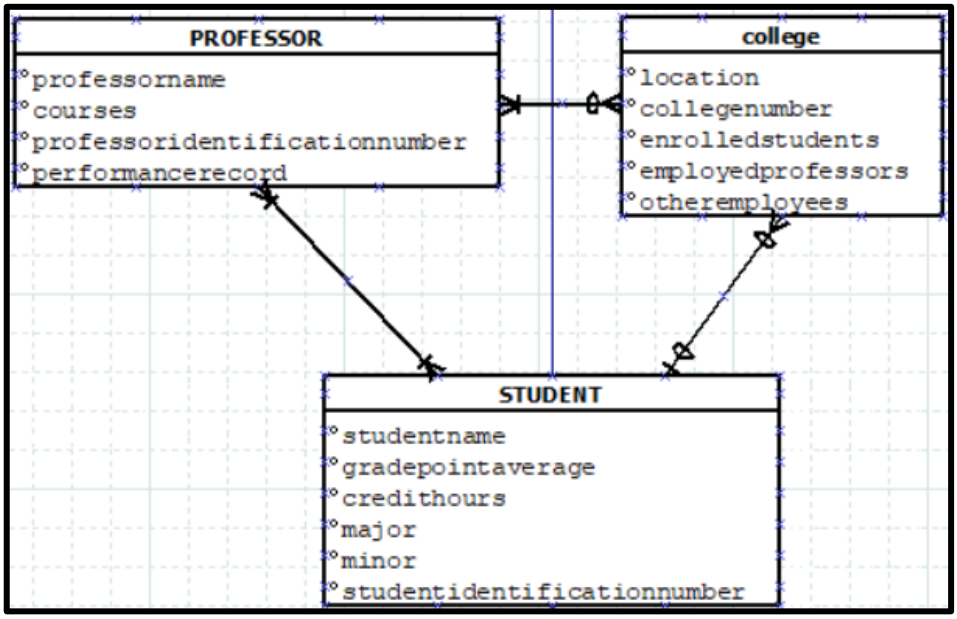

Figure 6 – ER Diagram with Corrected Attribute Names (Bartz, 2020)

Now that the variables have been clarified, we can begin fixing some finer details like the relations between each table. The student and college relation is not indicative of the general format. A student at college will only have one college, and a college will have far more than one student enrolled. Additionally, professors that teach generally tend to stay at one college, so changing that correlation will better fit the represented data. Fixing this will better clarify the tables' reliance on each other, as well as opening more relation options between them.

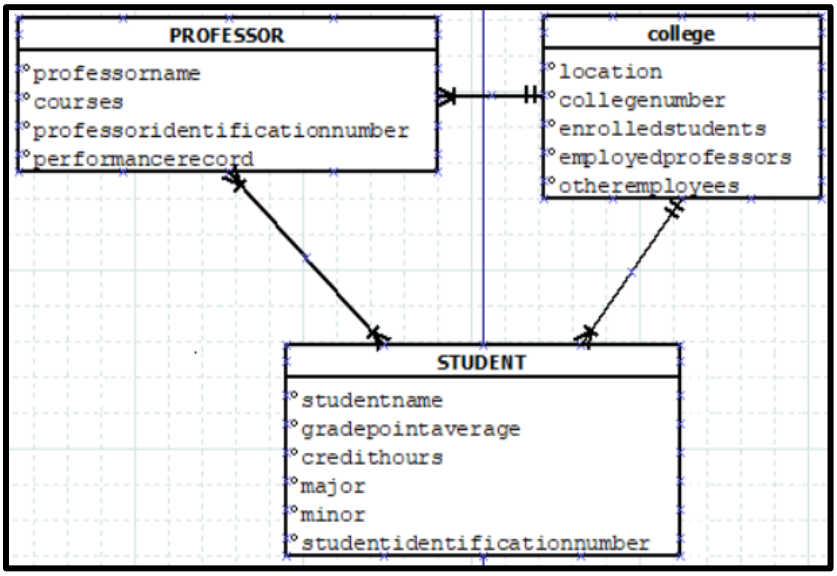

Figure 7 – ER Diagram With Corrected Relation Types (Bartz, 2020)

Now with the inter-table relations fixed, our focus can shift to the formatting of variables and table names. Some variables have far too long a name to be practically used or understood. Be careful when fixing these names, however, as cutting off too much or using unorthodox acronyms can lead to the same confusion regarding the variables. Also, no one wants to read super long variable names, and doing so in a conference or meeting puts an unnecessarily large amount of time spent explaining what a simple diagram should be.

In programming, such variables would never be used, mainly because no one wants to type out *student identification nnumber*, and then find out they misspelled it. Additionally, such values are memory inefficient, wasting space that could be used to make processes run faster. So, now to fix the table, look for shorter variable names or acronyms to replace the names already there, but remember, going for shortness over functionality could lead to a similarly cryptic graph.

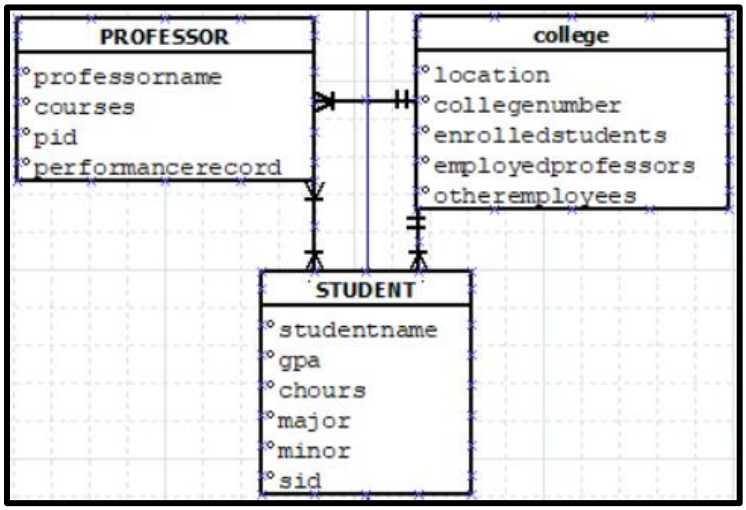

Figure 8 – ER Diagram With Corrected Relation Formatting (Bartz, 2020)

Now that these variable names are much shorter, let's format them more professionally. Capitalization is an easy way to better present data in a professional way. Capitalizing acronyms and one-word variables is one more layer to add to the credibility of the relation. Additionally, we can format variables in a better way than just a long string of characters. There are many naming conventions common in programming that can be applied outside of code to ER diagrams, but for this example, we will use the Camel case convention. Camel case uses capital letters after the first word to increase readability of longer variable names (ex: numapplepie to numApplePie).

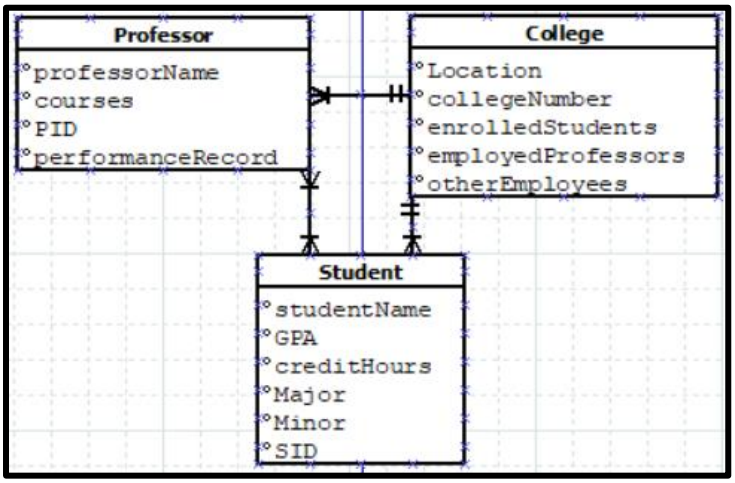

Figure 9 – ER Diagram With Corrected Attribute Format (Bartz, 2020)

This table has come a long way, and now that it is properly formatted, let's focus on details like the primary key for each table. For formatting, the primary key should go at the top of the table variables to better represent its role. Adding to the layers of complexity, let's examine some of the variables and their use. What about student names? Attempting to separate the name down into 2-3 parts is a tactic that is required, as most universities and colleges order role via last name.

So let's add some clarification to that name. Remember if we apply some convention to the visual, it should carry through, so if we edit studentName, we must edit the professorName variable as well. Another topic to cover is sub tables, that is tables that elaborate on certain variables within another linked table. In the next example, we will add the dependent variable to the Professor table as well as a sub table. One other topic to cover is useless data. Look at the college table and see that there are two employee variables. That's a waste of both space and data, so they can be combined into one variable.

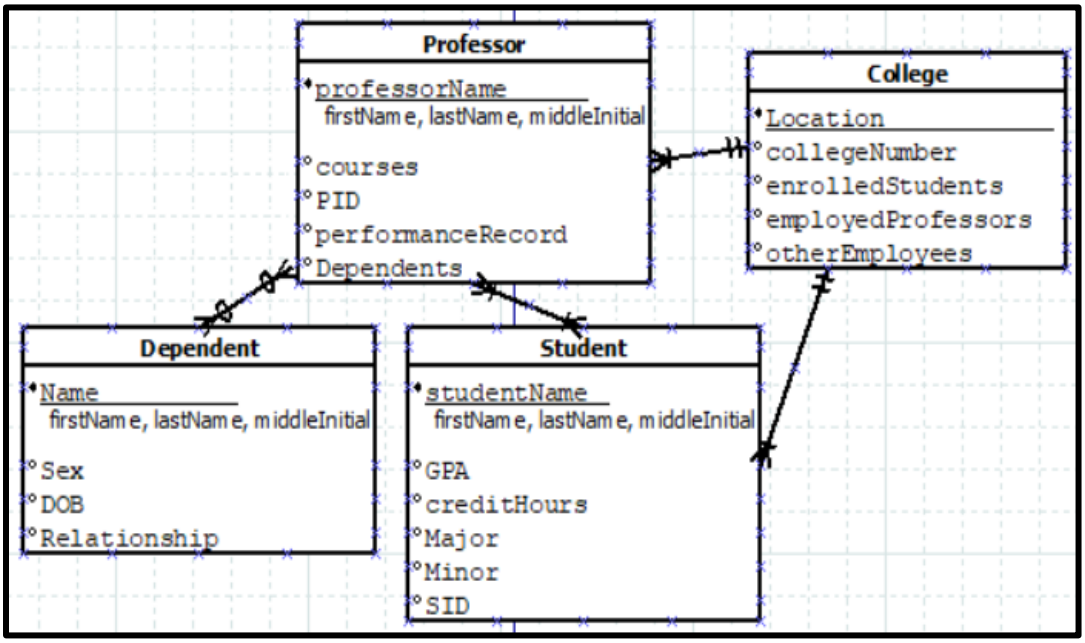

Figure 10 – ER Diagram with Corrected Keys (Bartz, 2020)

Now that all these edits have been applied, this table has been fixed from its confusing original state to one that could be shown to a conference room of businesspeople. All the changes were meant to improve both the appearance and readability of the data. Of course, some sacrifices were made to fit tables like these into this form. Ideally, an ER diagram should have a good bit of white space in between the tables to better separate them from each other. But stepping away from the massive amount of refining we had to do, these should not be applied after the table was written, but during the process. This allows for less overall time spent making the table, and more time available to improve or enhance the model or programming behind it. But let's step away from the complexity and formatting of these tables to better grasp the concept of modeling entities and their relationships.

When modeling new or different entities, it is best to approach the problem step by step. Start by splitting up the work into different categories: the initial entity, the attributes of that entity, and the entity's relationships to other values. Applying this logic to a new table, let's use a shipping company as an example. Firstly, let's identify the company's entities, which will be the tables necessary to understand its inner workings. Entities like orders and storage are very important, but there are far too many entities to model, so depending on what the meeting entails, tailoring the data to fit the occasion is a necessity. For instance, showing what customer data you get in a meeting discussing storage employee importance is unimportant to the current topic.

Speaking of customer data, the next step in making a comprehendible ER diagram revolves around getting the attributes and variables that each entity has. Remember to make this data easily recognizable by its name. That name should represent the data contained within, because while a

variable named Ren_and_Stimpy is very recognizable, if it contained an order number, that would be terribly confusing. Customer data might include things like their name, address, and so forth. Customer data should, however, not include extraneous details like their sex or skin color. While elaborating on what each entity has, be careful not to program personal details to be stored carelessly, as data leaks have become more and more common over the years. Keeping the data secure is important, but remember, the less you collect, the easier it is to encrypt or protect from outside influence.

2.4 Modeling

Finally, speaking of data, entities and variables might have relationships to other entities, so this should be the last phase of ER diagram assembly. Take, for example, the order and customer entities in the context of the company. The relationship is as such, for each customer, they may have multiple orders, while each order will only have one customer who made it. The storage facility may have multiple incoming orders or requests, but the order only references one facility.

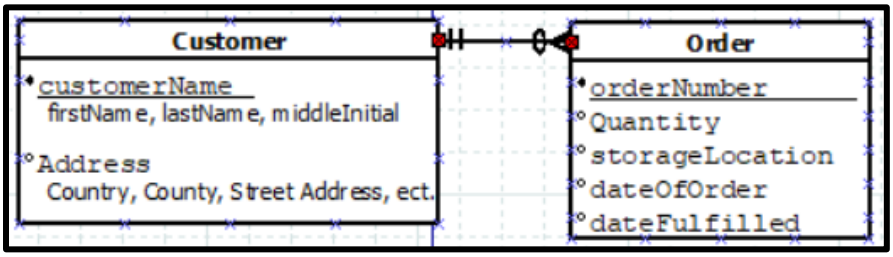

Figure 11 – Entity-to-Entity Relation (Bartz, 2020)

Thus we can see the importance of properly laying out and describing ER relations to better represent the data. However, let's address some smaller, more technical issues that may arise when creating a diagram. Given a diagram, sometimes concepts or variables put onto the board that others are not familiar with. Say you use some in-house acronyms at a large event, and then people ask questions about those topics. That is wasted time that could have been spent asking better questions that you prepared to answer. Remember that while you want to inform your audience, it is important to present that in an easily comprehendible way. Following some commonly used conventions, as well as using some advice outlined above, will allow you to work smarter, not harder.

2.5 Examples

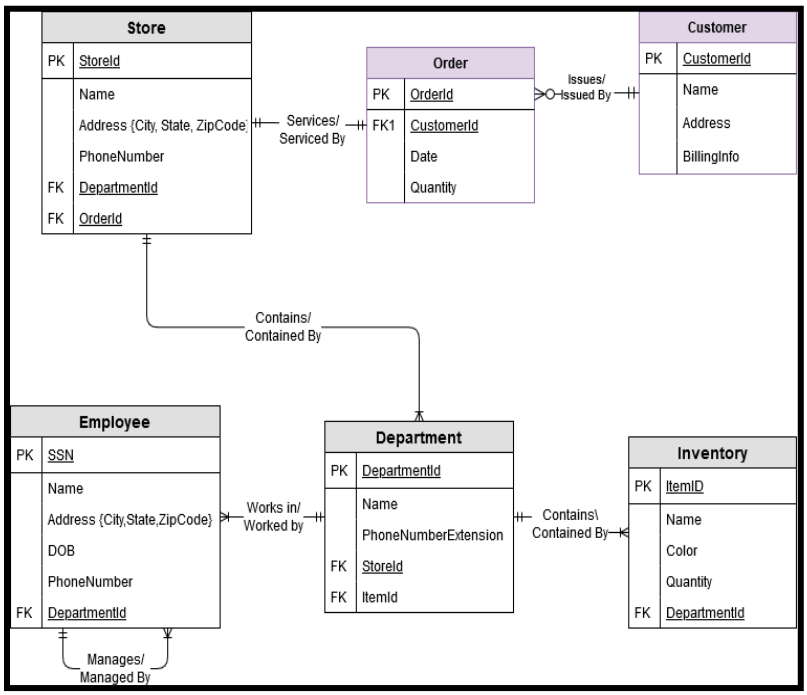

Figure 12 – Store ER Diagram (Bennett, 2020

2.6 Business Intelligence Systems and Data Warehouse

Business Intelligence (BI) Systems are any database system that has data, programs, and personnel and are specialized for the preparation of data for BI processing. A **Data Warehouse** is a system used for reporting and data analysis, and is considered a core component of business intelligence. This includes day-to-day transactions and other internal data or external data from sources by the **Extract, Transform, and load (ETL)** System.

Database SKUs (A **stock-keeping unit (SKU**) is a scannable bar code, most often seen printed on product labels in a retail store) where the **primary key** consists of one or more column will uniquely identify each row or record in the table. Some tables may have one or more **foreign keys** that are used to create **relationship** between tables logically link the tables together.

2.7 Introduction to Structure Query Language (SQL)

SQL is the universal query language of relational database management systems (DBMS) that is almost always behind user-friendly GUIs. In this section, we will briefly talk about SQL queries. We will visit SQL in more detail in later chapters.

SQL statements can also include a **SQL comment**, which is a block of text used to document the SQL statement but not executed as part of the statement. SQL comments are enclosed in the symbols **/* and */**, and any text between these symbols is ignored when the SQL statement is executed. Here is an example:

```
/* SQL-Quaery-Ch02 */
```

The **fundamental statement of SQL** query can apply to Microsoft Access, SQL Server, Oracle Database, and MySQL. The basic form of SQL queries uses **SQL SLECT – FROM – WHERE framework**. Here are some basic specifications:

- The **SQL SELECT** statement specifies which **columns** are to be listed in the query results.
- The **SQL FROM** statement specifies which **tables** are to be listed in the query results.
- The **SQL WHERE** statement specifies which **rows** are to be listed in the query results.

Reading Specified Columns from Single Table – to obtain values from the SKU_Data table, we write a **SQL SELECT** statement that contains all of the column names in the table:

/* SQL-Query-Ch_02 */

	SKU	SKU_Description	Department	Buyer
1	200201	Women's T-Shirts	Clothing	Tyra Perry
2	200202	Mans's T-Shirts	Clothing	Bradly Cooper
3	200203	Children T-Shirt	Clothing	Cora Mathis

```
SELECT    SKU, SKU_Description, Department, Buyer
FROM      SKU_DATA;
```

Below is a SQL queries from a single table, which obtains just the value of the Department and Buyer columns of the SKU_Data table:

Department	Buyer
Clothing	Tyra Perry
Clothing	Bradly Cooper
Clothing	Cora Mathis

```
SELECT    Department, Buyer
FROM      SKU_DATA;
```

The Catalog_SKU_2019 table shows DateOnWebSite. To see these items, we use the follow query:

/* SQL-Query-Ch_02 */

	SKU	SKU_Description	Department	Buyer	Date on WebSite
1	200201	Women's T-Shirts	Clothing	Tyra Perry	2020-10-10
2	200202	Mans's T-Shirts	Clothing	Bradly Cooper	2019-11-11
3	200203	Children T-Shirt	Clothing	Cora Mathis	2020-20-20

```
SELECT      *
FROM        CATALOG_SKU_2020
WHERE       DateOnWebSite = '10-Oct-2020';
```

2.8 Submitting SQL Statement to the DBMS

Using **SQL in Microsoft Access 2016**, we can execute SQL statements and run Query windows in **Design view**.

1. Click the **Create** command tab to display the Create command groups, as shown in Figure 2-13.
2. Click the **Query Design** button.
3. The Query1 tabbed document window is displayed in Design view, along with the Show Table dialog box, as shown in Figure 2-10.
4. Click the **Close** button on the Show Table dialog box. The Query1 document window now looks as shown in Figure 2-11, with the Query Tools contextual command tab and the Design command tab displayed. This window is used for creating and editing Microsoft Access queries in Design view and is used with Microsoft Access QBE (Query By Design).

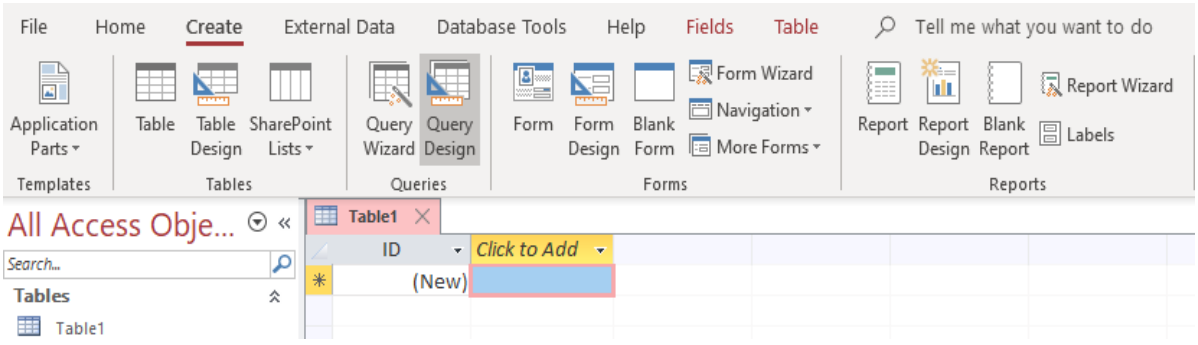

Figure 2-13- Microsoft Access 2016 Design view

2.9 Concise Summary

An Entity Relationship Diagram, or ER diagram, is a visual representation of a database and the interactions between its entities and attributes. The need for ER diagrams arose from the need to structure complex databases in the 1970s. Aspects of ER diagrams that are appealing include their not only being easy to understand but also easy to create and teach.

One of the earliest models was developed by Peter Chen, which included entities (rectangles), attributes (ovals) and their relationships (diamonds). Newer ER diagrams (using Unifying Modeling Language) simplify this by attaching attributes to their entities in a rectangle and by having relationships designated by lines with specific end-connection symbols that represent the following:

- One-to-one
- One-to-many
- Many-to-one
- Many-to-many

All four of the above relationships are also combined with a symbol for mandatory or optional cardinality.

There are a few naming conventions that have evolved since Chen's original model. Entity names are typically one word, bolded, and put in the upper rectangle (the case can be interchangeable, but most use either Pascal or Camel). When keys are introduced, primary keys are camel-case and underlined, and secondary keys are just camel-case; all keys must have the same name as their corresponding entities. Attributes are camel-case, unique, and as short as possible.

ER diagrams allow a prebuild for databases which saves time, effort, and confusion for programming teams. They are also a great tool when used to help diagnose errors in the database. These simple diagrams are crucial when creating a database as efficiently and quickly as possible.

2.10 Extended Resources

1. This explains ER diagrams in depth. They go over each of the main parts; entity, attribute, and relationship, and each of the variations of them.

https://beginnersbook.com/2015/04/e-r-model-in-dbms/

---

2. https://www.tutorialspoint.com/dbms/er_diagram_representation.htm

This site shows the old method of making ER diagrams and gives a good description of each of the types of connections between the entities.

---

3. This resource shows two pictures; one shows possible individual components of an ERD, and the other depicts an example of an ERD using all the components.

https://www.oreilly.com/library/view/database-systems-concepts/9788177585674/9788177585674_ch06lev1sec5.html

---

4. This video covers ER basics; what ER diagrams are, when to use them, and useful tips for creating them. The video shows an example of an ERD being drawn out in 'smart draw' to get a better understanding on how one is made from scratch

https://www.youtube.com/watch?v=dUJp0Yoq5eg

---

5. In this article they go over a brief history of ER diagrams. They discuss the different people involved with the creation and changes made to the models over time.

https://www.dataversity.net/a-short-history-of-the-er-diagram-and-information-modeling/

---

6. ER Diagram Template: Old version

https://www.lucidchart.com/pages/templates/er-diagram/hospital-er-diagram-template

---

7. ER Diagram Template: New version

https://www.lucidchart.com/pages/templates/er-diagram/database-er-diagram-template

---

2.11 References

SmartDraw. Entity Relationship Diagram. (n.d.). Retrieved from https://www.smartdraw.com/entity-relationship-diagram/#whatIsERD

Kempe, S. (2013, November 20). A Short History of the ER Diagram and Information Modeling. Retrieved from https://www.dataversity.net/a-short-history-of-the-er-diagram-and-information-modeling/#

Lucidchart. Template: Entity Relationship Diagram. (n.d.). Retrieved from https://lucidchart.zendesk.com/hc/en-us/articles/360000478183-Template-Entity-Relationship-Diagram

Visual Paradigm. What is Entity Relationship Diagram (ER)? (n.d.). Retrieved from https://www.visual-paradigm.com/guide/data-modeling/what-is-entity-relationship-diagram/

Creately 2008. R Diagram for Hospital Management System (Entity Relationship Diagram). (n.d.). Retrieved March 22, 2020, from https://creately.com/diagram/example/hwj0b7oy1/E-R Diagram for Hospital Management System