Chapter 2 – Databases

2.1 What is Data?

Databases are used to retrieve and report meaningful information to organizations. This begs the question: what is information? A simple definition might be that it is a fact provided or learned about something or someone. This is a relatively superficial view on the concept though. It would be beneficial to consider a structured taxonomy of information and how it might be broken down.

2.1.1 Theoretical View of Data

One approach to this is to take a theoretical view of information. A well vetted theory is Stamper's (1973) Semiotic Framework. This theory breaks information down into its most atomic unit: signs and signals. This is purely at the physical dimension. For example, a pulsed radio wave containing intermittent bursts could be a signal. The same could be said of smoke signals. A mark in the form of a circle could be a sign. Both of these are forms of the empirical or data level of semiotics. These are items that can be measured and quantified independent of any intended meaning or structure.

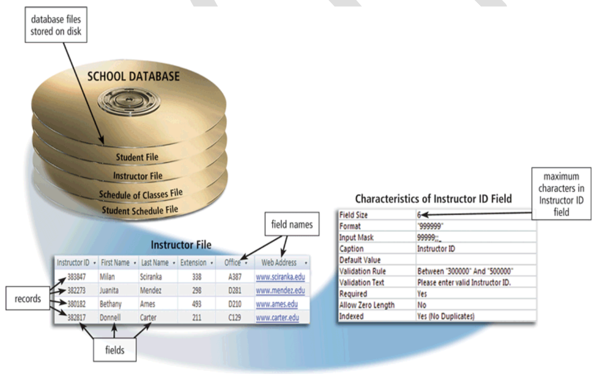
If one begins to apply meaning to these signs, they have moved up to a different semiotic level: semantics. Semantics is the study of meaning. For example, the pulsed radio waves might *mean* that a sequence of 1s (high pulse) and 0s (low pulse) are being sent through the air. The mark in the form of a circle might *mean* that an “o” sound is being recorded.

Bringing together different subunits of meaning requires a set of rules. This is yet another level up in the semiotic model: syntactic. Syntax regulates the structures by which meaning takes form. These are the formal structures that govern higher levels of meaning. In the example of the radio waves containing 1s and 0s, there could be a syntactical rule that states that groups of 8 of these 1s and 0s has a special meaning. 01000001 (or 65 in base 10) would mean that an “A” is being communicated. There are different levels of structure, from building abstractions of letters, words, sentences, etc.

As messages are coalesced from rules binding the meaning together, information becomes apparent. Information is the highest level of Semiotic Theory. Messages that contain meaning, structure, and information is where Semiotic Theory ultimately leads to. Contemporary researchers have proposed one additional layer at the top of the theory: knowledge. This demonstrates that social interpretation can have an effect on how information is understood. For example, the Chinese language uses graphical characters instead of a Latin alphabet. Because of this, that string of 1s and 0s would mean something very different for their knowledge base.

2.1.2 Practical View of Data

While Semiotic Theory provides a nuanced and detailed view of information, it might prove useful to consider how this might act as an analogy to information that is technically implemented in a database. In a database, data is organized in layers, much like the layers of Semiotic Theory. At the lowest level, a database stores characters. These might be characters, integers, floating numbers, dates, etc. These are combined to make up the data within fields. The group of characters “Michael” might be the data in the field “First_Name.” Fields are combined together to make up rows called records. A record might contain the fields First_Name, Last_Name, Customer_ID, Phone_No, and Email_Addr. Combining records together makes up a table (which is stored in a file).



Instructor File

Instructor ID	First Name	Last Name	Extension	Office	Web Address
383847	Milan	Sciranka	338	A387	www.sciranka.edu
382273	Juanita	Mendez	298	D281	www.mendez.edu
380182	Bethany	Ames	493	D210	www.ames.edu
382817	Donnell	Carter	211	C129	www.carter.edu

Characteristics of Instructor ID Field

Field Size	6
Format	"999999"
Input Mask	99999;;_
Caption	Instructor ID
Default Value	
Validation Rule	Between "300000" And "500000"
Validation Text	Please enter valid Instructor ID.
Required	Yes
Allow Zero Length	No
Indexed	Yes (No Duplicates)

The analogy works well as the levels in Semiotic Theory can be used to describe the layers of data in a database. The atomic data units in a database are the Semiotic equivalent of signs. They do not contain any inherent meaning. Grouping these units together into meaningful fields is analogous to the semantic level of Semiotic Theory. These fields are grouped together in an organized, rule based methodology into records. You can only have one record per line of a table. Each record must be distinct. Each record must have the same number of fields. These are the semiotic equivalent of syntax or rules. Finally, you group the records together into a table or file. It is within these tables that you finally have information, the last layer of Semiotic Theory.

2.2 Historical Context

In the first chapter, an *Information System* was defined as a social system that has been technically implemented. This is important when considering that databases provide a way to store data in a modern sense. Organizations have stored their data for millennia before the advent of computers, networks, or databases. From scrolls, to scraps of paper, to filing systems, to “flat file” computerized systems, organizations have always found a way to store their records.

2.2.1 A Thought Experiment with Filing Cabinets

Before computers became commonplace in organizations, the most common method that was used to store data was with paper filing systems. In order to organize their paper files, most organizations utilized filing cabinets. As a thought experiment, it might be interesting to determine how much data a filing cabinet actually stores. A 4-drawer filing cabinet can store about 32,000 sheets of paper. A piece of paper can hold about 6000 characters. Therefore, a filing cabinet can hold about 192,000,000 characters worth of data. Considering the fact that computers use 1 byte of space to store information about 1 character, this is equivalent to 192 million bytes of storage. A million bytes is typically denoted by the term Megabyte (MB). Therefore, a filing cabinet can store 192MB of data.

To take this a step further, a typical laptop computer in 2014 will offer a 1 Terabyte (TB) hard drive in storage. A TB is a trillion bytes and is equivalent to 1,000,000 MB. So if a 4 drawer filing cabinet can store 192 MB and a modern hard drive can store 1TB, it would take approximately 5,200 filing cabinets to store the information a single modern hard drive can. This is calculated by dividing the size of a modern hard drive by the storage available in a filing cabinet. This would look like this: FilingCabinets = HardDrive / FilingCabinetStorage. In this example, ~5200 = 1,000,000,000,000 / 192,000,000.

What is important about this thought experiment is that it shows the exponential increase in available storage available to organizations over the last 50 years. Considering the fact that organizations live and die by the information they have available to them, this has radically changed the way organizations do business. It has allowed for game changing innovations to proliferate through various industries including Just-in-Time inventory, real time inventory tracking, customer tracking, targeting marketing, accounting information systems, and others. Without massive data storage capabilities, these would not be possible. For the organizations that deploy these technologies, they put themselves at a competitive advantage.

2.2.2 Islands of Information

Islands of Information are clusters of data that are utilized by a distinct branch within an organization. For example, the shipping department might need to have customer information readily available for quick shipping. To facilitate this, the department manager would maintain a set of customer information located physically in his or her department. The billing department might also need quick access to customer information for quick reference in order to effectively settle bills. The organization has now set up an ad hoc system of multiple, redundant sets of data. While it may facilitate the day to day operations of the individual departments, the organization has opened up a potential can of worms. If a customer moves, are they expected to notify each department? If not, is there an internal procedure in place to handle this situation?

The more complex an organization is, the more likely is going to encounter errors and problems with islands of information. Even if a process is in place to handle changing data between different departments, this process is reliant on human input. Humans are prone to error and thus the system is doomed to problems.

As computers first made their way into organizations in the 1950s and 1960s, their application was seen as a replacement to current methods of data storage. Bulky mainframes replaced the tons of filing cabinets that clogged large organizations. This method of data storage is now known as “flat file” storage. They are retroactively referred to as flat files because they are flat and reside as stand-alone units of data. They do not relate to each other or to anything else in the system. The problem with this was that this implementation did nothing to alleviate the “islands of information” problem that become inherent in large and complex organizations.

There are three major events that can happen to data: add new data, modify existing data, and delete data. Even with the new computer flat files of the 1950s and 1960s, organizations found themselves having to create separate processes to add data to different files every time a single type of data was changed. For example, when a customer changed their address, the organization would have to arrange a paper process to forward the change to all affected departments. The same went for when a new customer was added or when a customer was deleted from the system. It made for a terribly inefficient and flawed data storage methodology.

Organizations needed something new. A single centralized place to store all of the organization’s data was one thing that was obvious. The physical “islands of information” was too problematic. They also needed a way to relate the data to each other. It wouldn’t do much good if the data had to be repeated in a centralized flat file system. For example, if a customer placed 5 orders over the course of a year, how would this information be stored? In a centralized flat file system, a separate record containing the full customer information would have to be recorded. If the data could be related to each other, the system could simply tie together the single customer record with each of the five orders.

2.3 A Closer Look at Databases

2.3.1 What are Databases?

A database is a structured set of data held in a computer, especially one that is accessible in various ways. A relational database stores data in tables that consist of rows and columns. Each row has a *primary key* (PK) that allows each row to be uniquely identified. A PK is required as all rows must be unique and identifiable. Columns are fields in the table and are referred to as *Attributes*. A relationship is a link within the data and is implemented with *Foreign Keys* (FKs). An FK is an attribute in a table that ties directly to the PK of another table.

The chapter previously described one of the major drawbacks of using flat files to store data in organizations: the lack of ability to tie pieces of data together. Not being able to create relational data leads to redundancy and errors. Databases do allow this tying together of tables by way of FKs. In Microsoft Access, you first include the field (or “attribute”) in the table being related when designing the table. You would then open up the Relationship View and create the relationship between the two tables.

Determining how to go about deciding which table shares its PK to the table with the FK is typically done during the design of the database. In a single relationship, one would determine which table (or entity) could exist independently of the other one. For example, if there is a relationship between a “customer” entity and an “order” entity, the designer would ask if a customer could exist without an order and if an order can exist without a customer. Since a customer can exist independent of an order and an order cannot exist independent of a customer, the customers PK (say for example customer_ID) would be included in the order table as an FK. This process would be repeated for all relationships in the database.

A single entity does not necessarily only have to take the role of parent or child if it has multiple relationships. It could be a parent in one relationship and child in another. For example, in the customer example, there could also be a relationship between the customer and salesperson. If an organization has a policy that all customers are assigned one salesperson, then a salesperson could exist with no customers but a customer could not exist without a salesperson. The PK of the salesperson would be placed in the customer table as an FK. Now, the customer is the parent table in the customer-order relationship and the child table in the salesperson-customer relationship.

One important aspect of building relationships is to maintain what is called *referential integrity*, or ensuring that all FKs point to valid PKs. If a record from a parent table is deleted and that parent has one or more children in a child table, which is problematic. In this situation, records in a child table would exist that are now inaccessible. For example, if customer #234 has ordered something in the past, there is now a record in the order table that has an FK of 234. If at some point, customer #234 is deleted, there would now be an order record that has no associated customer.

The way a database handles this is in one of two ways. One is to simply not allow a user to delete a record that has associated records in another table. The other option is to conduct a cascading delete. In a cascading delete, the database will first delete all associated records first and then come back and delete the parent record.

2.3.2 The Language of Databases

In an introductory class, it is common to use Microsoft Access to learn how to create and manipulate databases. In Access, tables are created graphically and are relatively easy to learn. As described prior, Access also allows the developer to create relationships graphically. What “graphically” refers to in this context is that the developer clicks icons and is visually led through the process of database creation.

Most Database Management Systems (DBMS) do not offer this and the databases must be created and maintained using a command line. The language that is used is called Structured Query Language (SQL). This can be pronounced as either “Sequel” or “S-Q-L.” SQL is a query language that allows users to manage, update, and retrieve data. The full details of SQL are outside of the scope of a general MIS introductory class but it is still important to understand its function and have a general knowledge of how it works.

In the example of Access, tables are created graphically. In a DBMS that uses a command line, SQL is used. To create a table, one would use the “CREATE TABLE” command. An example might look like this: CREATE TABLE customer (customer_ID int, customer_Lname varchar (25), customer_Fname varchar (25)). In this example, the table customer is created with three attributes: customer_ID, customer_Lname, and customer_Fname. The word after each of the field names is the datatype of that field. The first field, customer_ID is an integer. The second two fields, customer_Lname and customer_Fname are alphanumeric character strings that are up to 25 characters long.

Once a table is created, a developer may want to add some data into it. To do that, they would use the “INSERT INTO” command. An example might look like this: INSERT INTO customer VALUES (234, ‘Smith’, ‘John’). In this example, the customer John Smith with customer ID 234 is being added to the customer table.

One of the most commonly used commands in SQL is the SELECT command. When the SELECT command is run, the database is being queried for data. It might be clear why the name of the language is Structured *Query* Language since querying with the SELECT command is the most common command run in the language. When querying data, a user or developer must specify three things: what they want, where they want it from, and under what conditions.

An example of a SELECT command might look like this: SELECT customer_Fname FROM customer WHERE customer_ID = 234. In this example, customer_Fname is what the user wants. They want it from the customer table. Finally, they only want customer first names when the customer ID equals 234. The resultant table from this query would contain the value "John."

2.4 Implementing Databases

A database is implemented with a Database Management System (DBMS). A DBMS is a piece of software that controls the organization, storage and retrieval of data (fields, records and files) in a database. It also controls the security and integrity of the database. The major providers of DBMSs are IBM (with their DB2 DBMS), Oracle, Microsoft (with SQL Server) and the open source MySQL and SQLite DBMSs. Not including mobile platforms, MySQL is the most widely used open source DBMS and is used in many platforms including high profile websites such as Google, Facebook, Twitter, Flickr, and YouTube. It is also the only major DBMS listed that is free. Oracle, SQLServer, and DB2 have substantial licensing costs associated with their use.

Note that Access is not included on the list of major DBMSs. Though it is a DBMS in the sense of performing the core tasks of a DBMS (controlling the organization, storage and retrieval of data), there are several key missing elements. One is that Access allows a developer to create a database that violates referential integrity. As described before, referential integrity refers to making sure that every

FK has a corresponding PK. It is true that Access has the ability to enforce referential integrity but the fact that it is possible to create a database without that is troubling.

The other issue with Access is that it is not designed to easily be used by multiple users at the same time. This is known as multiuser access. In organizations, databases are typically accessed by many people (usually at the same time!) so multiuser access is critical. In order to facilitate this, a DBMS must be capable of locking and unlocking specific records seamlessly. Consider a situation where two users are trying to update a customer record at the same time. The DBMS has to decide who gets to do it and when they can access that record. DBMSs like MySQL and Oracle do this automatically and without any developer input. With Access, developers often have to jury rig a solution with layers of software on top of Access.

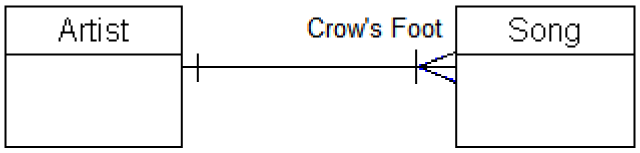
This is not to say that Access is not a useful tool. Given its graphical interface, it makes for a much better learning experience with introductory students. The learning curve is much shallower than in a command line, multi-user environment. The core concepts of table creation, data entry, and querying can be learned much easier. Access is also useful in limited professional environments such as smaller companies or organizations that have a limited need to share the database.

2.5 Designing Databases

Databases are designed with a graphical tool called the Entity Relationship Diagram (ERD). The ERD allows a database designer to think about the nature of the entities and the relationships between the entities without being bogged down by the technical aspects of creating the actual database. A good design is paramount to creating a working database. The information that is used when building an ERD comes from the business requirements for the organization. There are three major concepts to consider when creating an ERD.

The first is the entity. An entity is denoted by a rectangle in an ERD. An entity is anything real or abstract about which we want to store data. Entity types fall into five classes: roles, events, locations, tangible things or concepts. Some examples might include employee, payment, campus, or book. Specific examples of an entity are called instances. For example, the employee John Jones, Mary Smith's payment, etc. Instances are not part of the ERD, however. They come to play when the database is created and records are entered. The entity is more of a general placeholder.

The second major concept is the relationship. A data relationship is a natural association that exists between one or more entities. For example, employees process payments. A relationship is denoted by a line between entities in the ERD. Cardinality defines the number of occurrences of one entity for a single occurrence of the related entity. Cardinality is denoted with either a crows foot (see figure) for a many side relationship or a single line (see figure) for a one relationship. For example, an employee may process many payments but might not process any payments depending on the nature of her job. This was alluded to in the "What are Databases" section when the concept of parent-child relationships was described. In a single relationship, one would determine which table (or entity) could exist independently of the other one.

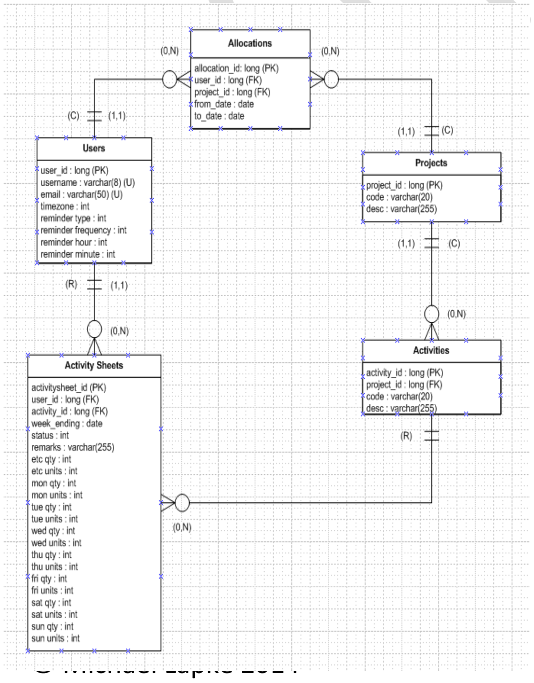


One-to-Many Cardinality

For example, if there is a relationship between a "customer" entity and an "order" entity, the designer would ask if a customer could exist without an order and if an order can exist without a customer. Since a customer can exist independent of an order and an order cannot exist independent of a customer, the customers PK (say for example customer_ID) would be included in the order table as an FK. This process would be repeated for all relationships in the database. Once a the parent/child relationship is established, cardinality is also determined. There is always *one* parent to *many* children.

There are three types of cardinality for a given relationship: one-to-many, one-to-one, and many-to-many. The ideal cardinality is one-to-many as described with the employee/payments relationship and the customer/order relationship. One-to-one relationships are not incorrect but are inefficient. If one instance of an entity will always be tied to one and only one instance of a related entity, why are they separate entities? In this situation, the designer should consider making this a single entity and make the attributes of the child table attributes in the parent table. The exception to this is when there is rarely data entered in the child table. A designer would save space in the database in this situation.

The third type, many-to-many, cannot exist in a relational database. If a relationship has this type of cardinality, you must create a bridge entity between the two entities that would have a many-to-many relationship. For example, if the entities student and course are related this would be a many to many relationship. When considering the parent/child aspect, you would determine that a student can exist without a course and a course can exist without a student. A student can take many courses and a course can be taken by many students. You should create a bridge entity (perhaps called class_schedule) between student and course. Now, a class schedule entry can only be tied to one student and one class. A student can take many classes in the class schedule and a course can be listed many times in the class schedule. You will have turned a many-to-many relationship into two one-to-many relationships.



The final major concept in ERDs is the attribute. A data attribute is a characteristic common to all or most instances of a particular entity. Synonyms include column, data element, field. Examples might include Name, address, Employee Number, and pay rate. Attributes for each entity should be written inside the box that makes up the entity.

An example of a full ERD is shown in the figure to the left. In this example, there are five entities and five relationships. The projects and activities entities have three attributes each. The allocations entity has five attributes. The users entity has eight attributes. The activity sheets entity has 22 attributes. The cardinality is represented by crow's feet for the many side and a single line for the one side. The PKs are denoted with the (PK) reference next to the appropriate attribute and the FKs are denoted with the (FK) reference.

2.6 Wrapping it up

Databases are essential to modern businesses. They facilitate the centralization of data and allow for data to relate to each other. This powerful tool helps organizations make the best use of their raw data and offers the most efficient way to build useful information that is critical in decision making. This can be at the operational level, managerial level, or executive/strategic level.