Entity Relationship Models

Data Design Concepts

Data design is critical in computer systems. In particular the design of the persistent data or database effects all other parts of the system: people, procedures, software, and even the choice of hardware. If a system was an animal then the database is the skeleton of the animal. Luckily there are well established ways of designing good databases.

You have already studied the discovery and documentation of data as it flows through an enterprise [a4.html], you have designed data flows [a5.html] [c1.html], and studied the design of data elements [d2.html] and the input and output of data [d1.html] to/from a user. You have looked at combining elements into logical data groups and records. The next two topics about about organizing many kinds of of records into a workable and reliable database. Modern data storage is always called a database even if it consists of a single file of student names and addresses. All the data stores in a DFD are conceptually in one database.

( conceptual models): Conceptual models are a picture of the stakeholder's world not the computer data. The purpose of stored data is to represent a part of the current reality. The data base should be a mirror of the world outside the computer. So a wise way to design data is to start with creating a picture of the stakeholder's world. We call this a conceptual model. It also turns up as the domain model in object oriented program design. We use Entity-Relationship Diagrams (ERDs) to draw a picture of it. Notice the importance of the stakeholder's ideas. These form an excellent entry into database design. Later we will talk about another entry into database design: samples of existing and planned data. There is a process that extracts an ideal database design from samples of data.
The entity relationship diagram gathers all types of records -- from all over the system -- into one place and works out how they are related. It then maps directly into the structure of most DBMSs.

An ERD shows a single database -- or a part of a larger database. There is no official way in an ERD to indicate that data is spread out over several data bases or file systems -- largely because it is a tool for integrating data onto one system. Even when you distribute data over several servers you need to first draw a single data base that ignores the deployment. It is (in my experience) rare -- probably due to some strange non-functional requirement -- that you distribute data over several "systems". And example would be the Internets DNS system -- millions of servers with their own data but linked into a single data base. ERD are not used to describe the placement of data in a system. We use an UML deployment diagram to show how we distribute various parts of a system onto hardware. We use them to show that some data is held on one system and the rest of it elsewhere. We saw these in [a3.html] earlier in this course.

Inside a data base there is a lot of computer science: CS330 Data structures, CS350 File Structures, etc.. We have a required BA course on Data Bases: CSE572. These notes here are just an introduction to a fascinating and lucrative field.

| The Principle of Locality | Data is processed most quickly in the order in which it is stored. Example: Telephone directories. It is designed to make it easy to find Joe Coyote's address in a phone book. However, it is not easy to find Joe Coyote's neighbor's name using a phone book. Data bases are first designed as Logical data bases and then fitted to a particular Physical design to meet performance requirements. The logical model is (nowadays) always a normalized entity-relationship model. |
| DBMS | :=Data Base Management Systems. Examples with links to information later. |
| DBA | :=Data Base Administrator. The guru who maintains the structure and integrity of the data base. |
| guru | :="a person who has arcane knowledge and expertise", roughly synonymous with ruhi, wizard, and sensei. |

**DBMS Components**

A DataBase Management System (DBMS) supports three kinds of interface: User + DBA + API. The user interface is special purpose software that makes it easy for the user to read, write, update, ... data. The DBA has a more technical interface that includes the creation and deletion of new types of data. The API is programmers to write programs that read and write data in the data base. An example is the Java DataBase Connectivity package -- JDBC. These days DBA's and programers typically use the Structured Query Language -- SQL to define and access data in the data base. You would be wise to get some experience of SQL before you go hunting for a job! The DBA can define special views and applications that let Users access the right data. Programs can use the DBMS API (Application Programmer Interface) to meet stakeholder needs.

| Schema | :="a description of all the records, links, reports, triggers, ... making up the data base". This is also called a data dictionary. |
| Physical data | -- the underlying coding and data structures. |

**Web-based database design**

Many modern web pages are just an elegant and easily used front-end to a complex data base. The connection between the web and the data base is called middle-ware. Sometimes the web server, the middle-ware, and the data base are all different computers. This is called a multi-tier architecture.

Security is a vital concern when you hook up the web to a data base (or anything else). Security Rule 1: never store unencrypted passwords in a data base.

**Data Design Terminology**
**Entity**

An entity is something that we need to keep track of in the real world, also used for the data we store about a "real" entity. This can be confusing. Examples -- *me* and *this class* are entities in CSUSB and we have our data stored in CMS (and some other systems no doubt).

**Relations**

Relationships (relations) link entities together in the real world. Example -- "Dick is married to Tricia". Example -- "Dick teaches this course". These are reflected by special tables in a database. A relation is a set of facts all with the same template:

___ is taking class ___ with teacher ___ at time ___

Filling in the blanks generates facts. The blanks in a template gives us the columns in a table in a database! They are the attributes of an entity that represents a relationship. A binary relation would have a sentence with two blanks, a ternary with 3, and so on. The number of columns in the data base table equals the number of blanks in the template.

**Entity type**

A collection of entities that all have similar data. Sometimes we deliberately use the word "entity" to refer to an "entity type". Examples -- *I* have entity type "Faculty", and *this class* is a "Class".

**Record Tuple or Row**

A collection of named data elements that should reflect some facts about a real world entity. Example -- "My record showing I teach this class"

A *Record* is also called a *row* because it is displayed as a row in a table of similar records.

1. A *Record* is also called a *row* because it is displayed as a row in a table of similar records.
2. Records are also called *Tuples* because of the underlying discrete mathematical model of an "n-tuple". In mathematics (1,2,3) is a 3-tuple and (3,4,5,6) is a 4-tuple and so a tuple is a list of data items.
3. Records are also called "Logical Data Groups" because a record is a collection of data elements that go together.

**Table or File**

A set of similar records -- all reflecting the same type of entity and (ideally) having the same structure.

4. Example: the CSE Schedule for the next quarter.
5. Many tables describe the attributes or properties of a set of entities.
6. Other tables relate different entities. For example this table is part of a "Teaches" table:

<table>
<thead>
<tr>
<th>Teacher</th>
<th>Course</th>
</tr>
</thead>
<tbody>
<tr>
<td>RBOTTING</td>
<td>CSE372</td>
</tr>
<tr>
<td>TYU</td>
<td>CSE310</td>
</tr>
<tr>
<td>KVOIGT</td>
<td>CSE201</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

(Close Table)

7. Most tables/files describe both the attributes of an entity and the other entities it is related to.
8. Some keep track of real relationships between entities. The above table relates the entity Teacher and the Entity Course.
9. Some keep track of complicated properties between many entities/values. Each row represents a sentence that is true:

**CSE372 section 01 is taught in JBH146 at 4-6pm TuTh by RBOTTING**

<table>
<thead>
<tr>
<th>Course</th>
<th>Section</th>
<th>Room</th>
<th>Time</th>
<th>Teacher</th>
</tr>
</thead>
<tbody>
<tr>
<td>CSE372</td>
<td>01</td>
<td>JBH146</td>
<td>4-6pm TuTh</td>
<td>RBOTTING</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

(Close Table)

**Field or Column or Attribute**

A *field* is named data elements with a given name in a column of a table or file. The word *field* goes back to the days of punched cards. A field (or column) is also called an attribute. All attributes in a column or field must have the same type of data element. In the above table there is a column for Section, Room, Time, and Teacher. In a normalized table each column is either an item of data of a particular type or the identifier or *Key* of a row in another table.

Diagrams have special ways to show the fields/columns/attributes in a table/file/entity. You need to know them all since all are in use.
Key

Keys are fields or attributes that uniquely identify records in files/tables uniquely. They are used to connect the different tables in a database together. I will show how these are documented in an ERD when we cover normalization. There are many types of key: primary, candidate, foreign, secondary. They are the heart of normalization and the most popular type of database: the relational database. Relational data bases and normalization are in [d4.html] and will be covered later.

Cardinality and Multiplicity

The data base community talk about Cardinality and the UML community talk about multiplicity. In both cases the question is: For each X, how many Y's are there?

The common answers are: one, many, zero or one, and one or more.

Example each section in CSUSB is given a unique number but a course has many sections. Some relations relate a single entity to a single entity. We say these are one-to-one relations. An example is the relation between an SSN and a resident in the United States. Similarly some relations are said to be "one-to-many". An example is: a teacher may teach many classes and each class is taught by one teacher. Finally you can have relations link many entities to many entities. For example, a student takes many classes and a class has many students. It turns out that these differences are vital for getting a database that works. There are 3 popular ways of indicating the number of entities -- the cardinality or multiplicity.

Summary of the theory of conceptual data bases

Data reflects the world. Records reflect entities. Tables reflect properties of entities and relationships between entities.

A Conceptual Data Base Design Process

1. Start with your notes and recordings of interviews, your DFDs, and the data dictionaries you already have.
2. List the kinds of facts that you want to record about the world outside the computer.
   1. "Dick Botting is 6ft 3in high and teaches CSE372"
   2. "RBOTTING teaches CSE372"
   3. "RBOTTING teaches CSE489"
   4. "CSE372 has 20 students"
   5. "Student Jo Coyote is taking CSE372"
   6. ...
3. When the facts contain words like "and" or "or" split them up into simpler facts: Example: "Dick Botting is 6ft 3in high".
   "Dick Botting teaches CSE372".
4. Look for facts that are Too Much Information! -- for example "Dick Botting is 6ft 3in high".
5. Look for nouns that are used by the stakeholders, these are often entities. But what type of entity are they? Are they aliases -- alternative names of entities you've already spotted? Do they pass the "Point at it" test -- is there a way, in the real world to identify a unique entity with this name?
6. Look for data that acts as an identifier. Look for names that end "_ID", "_Number", etc. When the user has gone to the trouble of assigning an identifier to something then it is probably an entity in that user's world. For example our courses have numbers so we can expect there to be an entity called "Course" in a system for this campus. Example -- This class has the id "CSE372". These are an excellent source for "candidate Keys". They are nearly always the kind of thing you will need as an entity in your data base.
Work out the best name for the types of entity. Good names come from the user's world.

Look for facts that assert simple properties of entities -- especially measurements, counts, and strings. These are likely to be attributes/elements/fields/columns. Again give attributes a good name.

Example: “Dick Botting is 6ft 3in high”. Attribute name: height.
Example: “CSE372 has 24 students”. Attribute: classSize.

Now look for facts that relate two or more entities -- typically by using their names and IDs. Example: “Dick Botting teaches CSE372” You’ll find these in samples of data and not in the statements that people make to you when you interview them. These become relationships. Relationships a stored as tables/files/entities in your data base.

What is an Entity

First, it should be something that exists (or is imagined) outside the system you are designing.

Second, there must be many occurrences of the entity. So, we don't store the name of CSUSB in a CSUSB data base: there is only one CSUSB. However in the Chancellor's office CSUSB will be an occurrence of the entity Campus.

Third, the occurrences should be identifiable and countable. Thus Butter is not an entity, but CaseOfButter would be. On current and/or planned systems is there an ID or number(#)? How many occurrences are there currently? Record this “frequency” because it will determine the speed of the resulting programs.

Fourth, it may be something that has data stored in the current system. It is nice if it is something that will need to have data stored in the future system. So, in a system about students and faculty, Student and Faculty are likely to be entities. Ask: If we lost the data, how much damage would this do to the enterprise? For example, CSUSB doesn't get hurt much if it doesn't know the color of your hair. Notice: you can extract a lot of structure from existing and proposed input and output formats.

Fifth, the occurrence should have interesting life cycles that last longer than a single program, function, or application. Ask: when do these entities get created and when are they destroyed?

Finally, most attributes are not entities. For example, Address may be an entity but NumberOnStreet is not. Things that are elementary data -- numbers, measurements, character strings, etc don't make good entities. They will end up as attributes/fields/columns of an entity/class/table. Be careful to distinguish attributes from identifiers and keys which point to entities. Also do not confuse the identifier/key with the type of entity it points at: The ISBN is an identifier for “Book” but is not itself an entity. Book is an entity with attribute ISBN.

How do I find entities

Talk to stakeholders and see what is important to them. How much would they pay to have data on this kind of entity on file?

Look at the stores and flows in your DFDs.

Start from some sample documents and look for the identifiers embedded in them. For example, a class roster identifies the Section and has the identifiers for a number of Students. Hence we can assume that "Student" and "Section" are entities.

How do I find relationships

Write down sentences that describe important facts: facts that your system will need to keep track of:

Student "J Coyote" has enrolled in section "01" of class "CSE372".

Given that "Student" and "Section" are entities in your ERD then you have uncovered the relationship: "Enrollment" between them. You can also see another relationship in the above sentence: Each "Section" is for a particular "Course". Exercise: sketch a quick informal ERD showing: Course, Section, Enrollment, and Student.

ERDs -- Entity Relationship Diagrams

It helps to have a diagram showing the Entity types and the Relations that hold between them. They were introduced in [a4.html Modeling Entities and Relationships in the UML] previously.

An ERD shows types of entities + relationships between them.

There are ERD notations: Chen vs UML

[Chen and Unified notations]

Not all relationships have only two entities involved in them and there are different notations and philosophies.
Both ERD notations can include attributes. Some people call the resulting diagrams **ERA diagrams**.

**ERD Patterns**

Knowing about patterns lets you make good models quickly. Forgetting them may slow you down a bit as you reinvent the wheel. A **design pattern** is a common way of resolving a set of conflicting forces. Patterns don't claim to be perfect, but they are one a way of solving a set of design issues. They are common solutions.

Here are some classic analysis patterns that I've noticed over the years. Use this list to help discover ERD models:

**Sentences lead to Relations:** Sentences often describe relationships between entities: Jo Doe is married to Jane Roe. Student Joe Coyote is enrolled in CSE372. Joe Coyote is working as a Librarian on the "FIS" project team in CS372.

1. A simple relationship between only two entities can be shown as simple link. RJBOTTING teaches CSE372.

2. But many-to-many and n-many relationships are best treated as entities. For example: An Enrollment links a Section to a Student....

3. A sentence can relate entities of the same type. For example a Marriage relates two People. The different occurrences play different roles. For example in a traditional a Marriage has one Person who is playing the role of *husband* and a second person playing the role of *wife*.
4. Some sentences describe an entity in terms of its attributes. When an entity has a property that is a value of a simple data type: a number, Boolean, character, string, or one of a number of enumerated values, then it is best shown as an attribute. For example a Person may have attributes: gender, age, height, ...

5. Notice: an entity can have attributes and be related to other entities. A Relationship can also have attributes, and be related to other Entities. However be careful not to record the same fact in multiple ways.

**Master and Detail Pattern** Example: A Sale has a number of Items. An Order has a number of order items.

**Factoring attributes** -- When two entities have a many-to-one relation between them and the entity at the many end has attributes, check to see if they can be moved to the other end. You can do this if all the entities that map into one entity all have the same value for an attribute. It can be shared by putting it in the other end of the association. A common example is in the [Master and Detail Pattern] where the same information is recorded in many details when it should only occur once in the master entity.

**Pattern -- Actor plays a Role for a Time** A Person can be a Student in a Quarter. A Person can Work on campus in a Quarter. They can be a Teacher in another Quarter. Combinations of roles are possible at one time.

**Events Time and Place Pattern** Event of a particular Type takes place at a Time and a Place. Example: CSE372 meets 4-6pm in JBH146 during winter quarter 2007.

**Description and Instance Pattern** Example: A product describes a sales Item. Also good for entities that are grouped by shared attribute values. Example: A Course has many Sections.

**Part and Whole Pattern** Building has many rooms. A Week has many days. Dog has 4 legs. Project has team members.

General and Particular Pattern -- A Student and a Teacher are special types of Person. In an ERD the special types are shown with a 0..1 multiplicity in the General. In general, you can have a whole hierarchy of special types. There are ways to document whether the special types overlap or not (UML Generalization). Avoid this detail in simple ERDs.

The Composite Pattern This a Gang of Four "Object-Oriented Programming" pattern that also turns up in some ERDs. A General entity has two types: Composite and Atomic. The Composite has many Parts and each Part is (in turn) a General entity.

The Enumeration pattern It is very common to define an entity type not by its relationships and attributes, but by listing its possible values. For example: Gender, Race, Boolean, ... Either document the values elsewhere or use the <<enumeration>> stereotype in the UML for the second of these. Do not confuse listing the attributes of an entity: Person has age, gender, eye color, ... with listing the possible values of an enumerated data type: Gender = {male, female, unknown}. Another example: student status = (freshman, sophomore, junior, senior, graduate).

By the way it is a common error to forget the "<<enumeration>>" and so turn the possible values of a simple entity into the attributes of a complex and erroneous one.

Any time you find yourself drawing the same diagram for the same reason -- you’ve found a pattern.

Check list of ERD Errors

1. Entities are outside the system and so you never have an entity in an ERD that represents the system.
2. Plural entity names. Examples: People, Students, Adults, Children. Exception when the entity is actually a grouping, set, collection, or list of other entities. Example: Class of Students. In the ERD show a box called "Class" associated with "Student".

3. Showing activities, actors, processes, etc. in an ERD. Entities and relations don't do things. You only show an actor if you need to store data about it. If you need to record an event or an activity that takes place (outside the system) then this will often be encoded in an attribute in an entity rather than a separate entity named after the event or activity.

4. Showing composition, aggregation, arrows, and operations: these are part of an object-oriented model of software. Keep ERDs simple.

5. Not preparing a precise definition of when a relation holds. Is it true now, or in the past (a record of an event), or in the future (a contract)?
Later I transcribed it to a napkin using the UML. And even later, I used Visio to get this ERD:

Normalize ERDs so that all relations are binary and many to one

The details for normalizing a set of sample data are presented later. But much normalization is worth can be done directly on a ERD/UML class diagram. Here is a picture summarizing ways to normalize a ERD:
How to quantify UML ERDs

ERDs are actually diagrams of discrete mathematical models. The boxes are finite sets and the many-to-one relations are functions mapping between the sets. As a result you can do calculations on normalized ERDs.

You can attach numbers to an ERD. This is like attaching times and durations to an activity diagram in the CPM. More importantly you can calculate missing numbers, and check the logic of the model you constructed.

Introduction to Quantification

Example 1.
Net

1. You have 20 cats.
2. Each cat has 4 legs (on average).
3. There are 80 legs.

(End of Net)

Example 2
Net

1. You need to keep track of 1,000 patients.
2. Each patients record takes up 500 bytes.
3. How much storage do you need?

(End of Net)

Example 3
Net

1. The college has 200 teachers and 4,000 students.
2. How many students per faculty on average?
Example 4

Net

1. The college has 200 teachers and 4,000 students.
2. A class is taught by one teacher.
3. Students enroll in an average of 5 classes.
4. On average, a teacher teaches 3 classes.
5. What is the average class size?

I use five special tagged values to quantify the above data bases.

Table

<table>
<thead>
<tr>
<th>tag</th>
<th>applied to</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>L</td>
<td>attribute</td>
<td>length. Number of bytes needed to store attribute</td>
</tr>
<tr>
<td>L</td>
<td>class</td>
<td>length. Number of bytes in one record in class/entity</td>
</tr>
<tr>
<td>N</td>
<td>class</td>
<td>expected number of entities/rows in table</td>
</tr>
<tr>
<td>V</td>
<td>class</td>
<td>volume in bytes, KB, MB</td>
</tr>
<tr>
<td>average</td>
<td>role</td>
<td>average number of objects/rows per associated entity</td>
</tr>
<tr>
<td>document</td>
<td>any</td>
<td>where to find detailed documentation</td>
</tr>
</tbody>
</table>

Procedure -- Quantifying an ERD

1. Start by normalizing the ERD
2. Write 1 at the 1 end and the average number at the many end as a UML constraint: \{average=\ldots\}
3. Write the number of occurrences of each entity by or in the entity box as a constraint: \{N=\ldots\}
4. Check the constraints.

Constraints on Quantities

Why check the constraints?
- Applying these constraints will expose problems in data models.
- They let you figure out how much disk space, back up, etc you will need.
- They can help you figure out why the performance of a system is lousy and help redesign it to work faster.

Some of the constraints are based on the mathematical principle that if you put 20 pigeons in 10 holes than on average you’ve got 2 pigeons in each hole.

(Pigeon hole principle): If there are \(n\) A’s and \(m\) B’s and each B has one A then the average number of A’s per B is \(m/n\).

Table

<table>
<thead>
<tr>
<th>Context</th>
<th>Constraint</th>
<th>Note</th>
</tr>
</thead>
<tbody>
<tr>
<td>class</td>
<td>(L = \Sigma <a href="a.L">a:attribute</a>)</td>
<td>Length of entity is sum of its entity lengths.</td>
</tr>
<tr>
<td>class</td>
<td>(V = L \times N)</td>
<td>Volume is Length times Number of occurrences</td>
</tr>
<tr>
<td>association</td>
<td>All are one-to-many or one-to-optional, no 1-1 and no many-to-many</td>
<td></td>
</tr>
<tr>
<td>association</td>
<td>If class with N=N1 has a one-to-average A role in class with N=N2 then N2=N1*A.</td>
<td></td>
</tr>
</tbody>
</table>
association
If class N=N\textsubscript{1} has a one-to-(\text{min}..\text{max}) role in class with N=N\textsubscript{2} then \text{min} \leq N2/N1 \leq \text{max}.

Questions

Online Resources

An Example of ERDs in the Context of a Project
The following has too many different ERDs: [toc.html] But you can see how ERDs can be used in a project.

More on Patterns
For more see [Gammaetal94] and PATTERN in my bibliography.

There is a book of design patterns for enterprises:

1. Jim Arlow & Ila Neustadt
2. Enterprise Patterns and the MDA: Building better solutions with archetype patterns and UML
3. Addison Wesley 2003

Data Base Management Systems

Oracle [index.html] [Oracle \%28DBMS\%29] [http://www.oracjobs.com/], MSAccess [FX10048771033.aspx] [Microsoft\ Access\], MySQL [MySQL], Postgres [PostGres], etc.

SQL injection attacks to database-driven web services
What is the average security risk level of SQL injection attacks to database-driven web services? Are there standard security techniques that help to safeguard against this kind of attack, or are the security solutions variable depending on the structure of the database design and/or web interface design? You've got the usual suspect; extra quotation marks that fake a stupid script into executing the data as a command. See the Wikipedia article [SQL injection] which includes the usual ways to defend against the threat. I would also expect there are standard security techniques for them.

What is an API

API::= "Application Programmer Interface", a long list of functions that can be called in a program to manipulate the data. For a typical database the functions take a string as an argument that contains a SQL command. They return a code indicating if they worked or not. They may have other arguments that provide data to the database, or are set to the results of the command. There are three basic kinds of function:
1. Queries extract information from the data base
2. Updates change the data in the data base
3. Administration commands change the structure of the Data Base

This search [manual-lookup.php?pattern=SQL] lists the names of the functions in the PHP API.

Other Application Programmer Interfaces

1. JDBC::="Java Data Base Connectivity",
2. SQL::="Structured Query Language", [ SQL ]

Examples of working with SQL data bases: [372doc6.pdf] (Aaron Smith) and [372doc7.pdf] (Nieleh Wäcots). If you need to see an example of an SQL base API check out the PHP language that includes special functions [ ref.mysql.php ] that use SQL to drive a MySQL database.

Mapping ERDs to XML

Given all the attributes this is not difficult and there are tools that can help you: [xml_wrangling.html]
Review Questions

1. What is an ERD? Define the term and explain what an ERD looks like and what it means.
2. Redraw each of the UML diagrams on this page as diagrams using the Chen notation.
3. If you have a Chen ERD/ERA -- redraw it using the UML.
4. Draw a rough (no boxes) ERD of part of CSUSB. You have 20 seconds!
5. Look at my ERD Patterns and insert the asterisks I have left off.
6. Draw a rough ERD of a DFD. Use these entity names: Process, Store, External Entity, Data Flow. List any constraints that cannot be expressed in an ERD.
7. What fragment of an ERD is suggested by "CSE372 section 01 is taught in JBH146 at 4-6pm TuTh by RBOTTIN"?
   Note: several solutions are possible.
8. Draw an ERD describing Use Cases in a project. A project has many Use Cases. Each communicates with many Actors and many Actors communicate with a given Use Case. A Use Case has a name. An Actor has a name. Each use case has a prime actor that achieves some goal by using that use case.
9. We have 10,000 students and each student is on average enrolled in 5 sections. Each faculty, on average teaches 2.5 sections. How many faculty must we have?
10. Find any tally-roll or receipt that you have and list the identifiers on it. If each identifier in your tally-roll or receipt is the key to an Entity type derive a rough ERD for the system that produced the receipt.
11. What are the two properties of a entity type that are used to determine the total volume of data needed? What is the formula connecting these tagged values to the volume.
12. Draw a rough ERD of a Dungeon-type game: Rooms have walls. Most walls have two rooms: one on each side. Walls have doors. Doors can be open or closed. Rooms have artifacts. Characters are always in a room and a room can have any number of characters in it. Characters can carry artifacts. Types of artifact: weapon, clothes, jewel, wand, .... Rooms have monsters. (Multiple answers are possible).
13. Draw an ERD of: A formula is the sum of many terms. A term is a product of factors. A factor can be a constant, a variable, or another formula.

Abbreviations

1. TBA:="To Be Announced".
2. TBD:="To Be Done".

Links

- Notes -- Analysis ([a1.html] [a2.html] [a3.html] [a4.html] [a5.html]) Choices ([c1.html] [c2.html] [c3.html]) -- Data ([d1.html] [d2.html] [d3.html] [d4.html]) -- Rules ([r1.html] [r2.html] [r3.html])
- Projects ([project1.html] [project2.html] [project3.html] [project4.html] [project5.html] [projects.html])
- Field Trips ([F1.html] [F2.html] [F3.html])
- ([about.html] [index.html] [schedule.html] [syllabus.html] [readings.html] [glossary.html] [contact.html] [grading/])

End