CIS 315 - Reading Packet: "Entity-relationship modeling, part 2"

* NOTE: you are required to follow COURSE STANDARDS for ERD's, regardless of the different ERD notations used in different software and textbooks;

SOURCES:

* Kroenke, "Database Processing: Fundamentals, Design, and Implementation", 7th edition, Chapter 1, Prentice Hall, 1999.
* Connolly and Begg, "Database Systems: A Practical Approach to Design Implementation and Management", 3rd Edition, Addison-Wesley.
* Korth and Silberschatz, "Database System Concepts"
* Rob and Coronel, "Database Systems: Design, Implementation, and Management", 3rd Edition, International Thomson Publishing, 1997.
* Ricardo, "Databases Illuminated", Jones and Bartlett.

* Sunderraman, "Oracle 9i Programming: A Primer", Addison-Wesley.
* Ullman, "Principles of Database Systems", 2nd Edition, Computer Science Press.

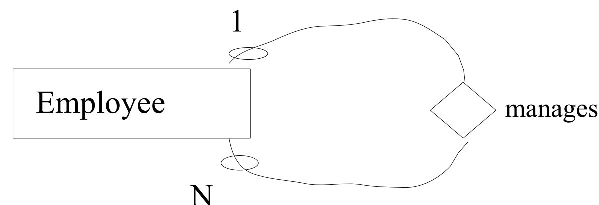
Note, too, that the recommended course text provides another approach for some of this material in Chapter 2, Section 2.2.

MODELLING, CONTINUED

In the last "lecture", we covered the fundamentals -- the most important basics -- of the Entity Relationship Model, and of ERD's depicting such models of scenarios. Here, we discuss a few more advanced constructs for such models.

Recursive Relationships

Note that a relationship class can be amongst instances of the same entity class -- that would be a recursive relationship, and the relationship line would begin and end on that entity class' rectangle! If so, the relationship class "line" simply begins and ends at the same entity class:



Some other examples of recursive relationships might be course-is-a-prerequisite-of-course, student-rooms-with-student, and employee-is-spouse-of-employee.

These really are handled similarly to any other relationship classes -- their relationship lines just happen to begin and end at the same entity class. Such relationships still need a diamond, a relationship line, a relationship name, maximum cardinalities, and minimum cardinalities.

Weak entities

In the **E-R model**, there's a definition for a **special** kind of entity class called a **weak entity** class. A **weak entity** is one whose presence is strongly dependent on the existence of another entity. Such entities are those that cannot exist in the database unless another type of entity also exists in the database. They may even derive their identity from that "parent" entity.

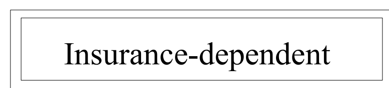
Be careful; it is easy to go overboard with this concept, and it is really meant to be for exceptional cases. Imagine an office scenario. Employees work on Projects, but neither Employee nor Project would be considered a weak entity class in this scenario; both are significant in their own right, although they are related to each other. You "care" about employee instances even if they haven't been assigned to work on a project yet; you "care" about project instances even if no employee has been assigned to work on them yet. But consider: perhaps this office allows Employees to designate Insurance-Dependents to receive health-care coverage. Insurance-Dependent may indeed be

an entity class; it may have its own attributes, such as the insurance-dependent's last name and first name, and perhaps date of birth, and perhaps whether he/she is a college student or not. But it is very dependent on Employee, under the scenario; the office may only allows insurance-dependents for current Employees. So, if an Employee leaves the office, any insurance-dependents of that employee must leave, too. That suggests that **Insurance-Dependent** is a weak entity class. An Insurance-Dependent entity depends on the existence of a corresponding Employee entity.

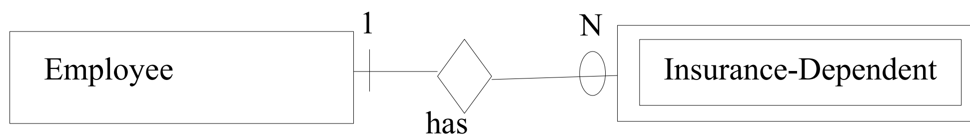
You could think of this as a kind of existence dependency, if you would like [Korth and Silberschatz]. If an entity is existence-dependent on another entity, then that entity may be a weak entity.

But, again, do not take this idea too far. You do not want to get to the point where you define any entity with a required relationship as weak. [Kroenke] A weak entity needs to be more dependent than that; it needs to logically depend on another entity, in a very fundamental way within the business rules of the scenario. An insurance-dependent must be related to an Employee to even be of interest within an office scenario; an apartment cannot exist without being in a building, within a property management scenario; an edition of a textbook cannot exist without the textbook existing, within a textbook publisher scenario; a version of a software application cannot exist without the software application existing, in a software company scenario. In contrast, in a university scenario, a Student exists without an Advisor, even if the university requires them to get one.

Once you have decided that an entity class is indeed a weak entity class, how do you depict that within an ERD? In this class, we will depict these in a fairly common way: as a **rectangle** with a **double-border**:



So, for example:



While we are looking at the above example, note the other following rule-of-thumb: a weak entity had better be involved in a mandatory relationship with the entity upon which it is dependent! (That is, there had BETTER be a hash or line on the Employee-end of the relationship line between Employee and Insurance-dependent, above -- there had BETTER be a minimum cardinality of 1 on the Employee end of that relationship!) Otherwise, how can we say that an Insurance-Dependent entity depends on an Employee entity, if it is not required to be related to at least one Employee entity?!

The recommended course text makes an interesting point on p. 16: notice that weak entity classes may not have any identifying attributes! (Note that this is not a hard-and-fast requirement, but it is a definite possibility.) It may be **id-dependent** on the entity class that it is dependent upon, deriving its identity from that. So, do not be concerned if this occurs with a weak entity class; just depict it appropriately within the model as a weak entity, and simply find yourself just not marking any of its attributes as identifying attributes.

Reminder: Remember that, in an ERD, relationships between entities are indicated **only** by relationship lines -- lists of attributes for each entity class include **no** such relationship-related information. So, you would **not** put the Employee's employee number, for example, into the attributes list for Insurance-dependent! The ERD's rectangles show what the entity classes are, the ERD's lines show the relationship classes between entity classes, and the ERD's lists of attributes for each entity should the important chracteristics of each entity **itself** (not anything related to, well, relationships). When these are depicted in this way, everything is in place to convert this E-R model into an appropriate set of relations during the design phase.

Supertypes and Subtypes

Supertype entities and **Subtype entities** are part of the so-called **extended** E-R model, added after Peter Chen's initial 1976 paper introducing the E-R model. Experience using the initial E-R model showed some situations that

were not handled well under that first version of the model, and so the E-R model was extended to allow better modeling of such situations.

Consider a scenario, say a small bank, that has Accounts -- but also Savings-accounts and Checking-accounts. Do you see that the relationship between these accounts is different than that between, say, a Customer and a Sale? Some entity classes may seem to contain seemingly-optional subsets of attributes, or, rather than being homogeneous, are more reasonably viewed as a collection of sub-groups. Sometimes such relationships are called **IS-A relationships** -- a Savings-Account **is an** Account, for example, whereas a Customer is not considered a Sale!

Supertype and subtype entity classes make such situations much more reasonable to model.

There are several ways to tell, as you are modeling, whether there are supertype/subtype entity classes lurking within your scenario, struggling to "get out": when an entity class has distinct sets of seemingly-"optional" attributes, for example. If you were just thinking about Accounts, for example, and started considering that some accounts have per-check-charges and max-number-of-checks-per-month, but others have interest-rates and minimum-balances, then that would be a sign that you really have an Accounts supertype and at least a couple of subtypes -- here, Checking-account and Savings-account subtypes. Another way to tell is if several entity classes seem to "share" some common attributes -- what if you had considered Savings-account and Checking-account only, and started noticing that both had Account-number, and date-opened, and current-balance, for example? That might suggest that the two are subtypes of a common supertype Account entity. Another indication may be relationships -- again, if you had only noticed Savings-account and Checking-account, but then noticed similar "owns" relationships between Customers and Savings-accounts and Customers and Checking-accounts, you might then notice that really there is a supertype Account, and Customer is really related to that Account.

(This is not to say that you cannot have a relationship between a subtype entity class and another entity class - you can! But this should be the case when the relationship is exclusive to that subclass. For example, say that a University setting has a University-person supertype, with subtypes of Student and Instructor. Then it might be the Student entity class that has a registers relationship with a Section-registration entity class -- and it might be the Instructor entity class that has a teaches relationship with a Course-section entity class.)

So, you have a clue that there are supertype-subtype entity classes when you have some attributes common to several different "groups" and some particular to different "groups". Another clue may be if you have some relationships common to several different "groups" and some particular to different "groups".

Supertype and subtype entity classes are another aspect of ERD's that do not have one standard depiction; in this class, we will depict them as follows:

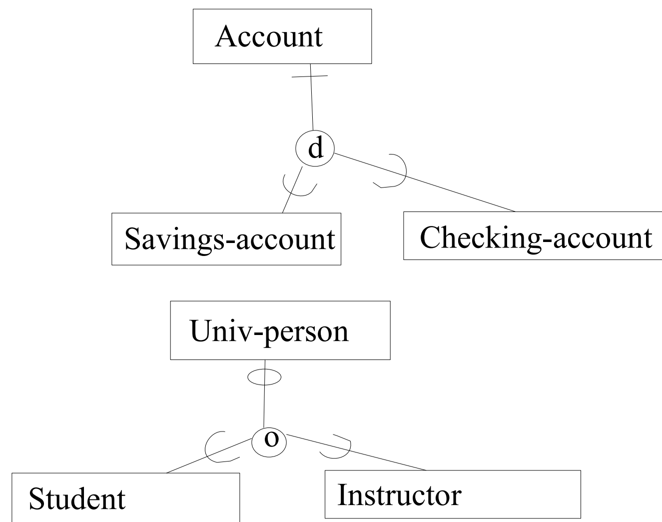
- Each supertype and subtypes entity classes will be written within a labeled rectangle, as is the case for any entity class,
- but since the relationship between a supertype and its subtypes is different, the relationship line will be accordingly different:
    - Lines are drawn from the supertype entity class and all subtype entity classes to a circle, (there is NO diamond on these lines),
    - and that circle is labeled with a **d** (for disjoint) or an **o** (for overlapping).
    - These reflect the different situations that sometimes a supertype instance must be exactly one of the subtypes -- that's **disjoint** -- and sometimes a supertype instance may be more than one of the subtypes -- that's **overlapping**.

(For example, an Account may be able to be a Savings-account or a Checking-account, but not both; but a University-person might be able to be both a Student and an Instructor (consider a Master's student permitted to teach course sections).)

- How can you tell the supertype from the subtypes?
    - Each line to a subtype has a u shape on it (with the points of the u "facing" the circle).
- And, finally, how can you tell if a supertype instance *has* to be one of the subtype instances?

- If it must be, you put a hash or line across the relationship line leading to the supertype entity class, near the supertype entity class rectangle;
- if it doesn't have to be, you put an oval across the relationship line leading to the supertype entity class, near the supertype entity rectangle.

So, here are two examples of supertype and subtype entity classes depicted in ERD's satisfying the class ERD notation:



(Note that you can certainly have more than two subtypes related to a given supertype... these two examples just happened to have two.)

Above, the ERD's depict a supertype entity call Account with two subtype entity classes Savings-account and Checking-account, and a supertype entity class Univ-person with two subtype entity classes Student and Instructor. In these particular scenarios, Savings-account and Checking-account are disjoint subtype entity classes: an Account entity instance may be one or the other, but not both. A Univ-person entity instance may be both a Student entity instance and an Instructor entity instance, since Student and Instructor are overlapping subtype entity classes. Finally, the hash near Account indicates that all Account entity instances must be either a Savings-account or a Checking-account; there are no Account entity instances that are neither a Savings-account nor a Checking-account. And the oval near Univ-person indicates that there can be Univ-person entity instances that neither a Student instance nor an Instructor instance (a University administrator, for example).

Do these sound similar to superclasses and subclasses in object-oriented programming? Really, they should -- the ideas are very close. This kind of a hierarchy is sometimes called a **generalization hierarchy**, because a supertype could be considered as a generalization of its subtypes. And we mentioned that relationships such as these are sometimes called **IS-A** relationships, since, for example, a Savings-Account is an Account, and a Student is a Univ-Person.

Another important aspect of these entity classes: when you are writing out the attribute lists for these entity classes, you still follow the rule mentioned earlier: you only list attributes specifically for that entity class. Here, that means that attributes common to all subtypes of a supertype are listed **only** with the supertype entity class's attributes; the only attributes listed for subtype entity classes are those particular to that subtype. It is assumed that, in reality, a subtype entity **inherits** all of the attributes of its supertype entity class (and so there is no need to rewrite them for the subtype entity class's attributes). (Again, shades of object-oriented programming, where subclasses really do inherit all of the data fields and methods of its supertype!) And it is usually the case, here, that the subclass entity class' attributes will have no identifier attributes indicated -- such attributes are usually in the supertype entity class' attribute list instead. (A savings account is both an Account entity and a Savings-account entity, and it is identified by its identifier in the superclass entity.)

Here, then, might be attribute lists for the ERD's given earlier:

```
Account              Savings-account           Checking-account
```

```
-------              ---------------          -----------------
ACCT-NUM             interest-rate            per-check-charge
date-opened          minimum-balance          max-number-of-checks-per-month
current-balance

Univ-person          Student                  Instructor
-----------          -------                  ----------
UNIV-ID              gpa                      salary-per-course
Last-name
First-name
Campus-email
```

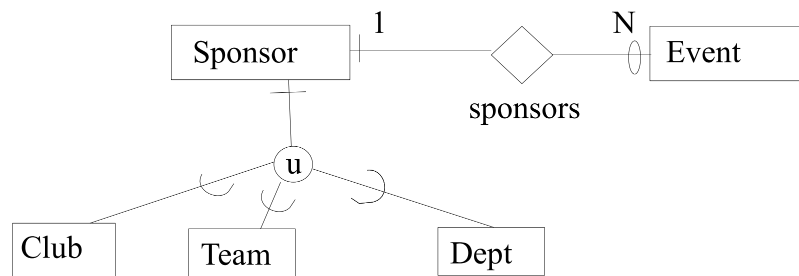
Note that it can happen that you could even have a subtype with **no** attributes in its attribute-list, for example in a case where a subtype entity class is part of a relationship class distinct to that subtype.

A Supertype/Subtypes variant: UNIONS

Ricardo (p. 384) mentions a potentially-useful supertype/subtypes variant: unions.

Sometimes you have entity classes that are really quite distinct -- they have separate identifying attributes, for example, and perhaps no overlap in terms of attributes -- that nevertheless share a relationship. Ricardo calls this a Union -- it would be depicted like a supertype/subtype in the ERD, except with a **U** on the circle. You'd notice here that the "subtype" entity classes feature identifying attributes, and that the "supertype" entity class has few or no attributes of its own, but participates in one or more relationships.

As an example, consider a university-based scenario in which there are entity classes for campus clubs, campus teams, campus departments, and campus events, and a business rule noting that campus events must be sponsored by either a club, a team, or a department. You have a situation where each event needs to be related to a club *or* a team *or* a department, which is awkward to model until you consider this Union approach -- but with that approach, you can recognize that a Sponsor entity class would make things much cleaner:




```
Sponsor     Club        Team        Dept             Event
-------     --------    --------    ---------        -----
            CLUB_NUM    TEAM_CODE   DEPT_CODE        EVENT_NUM
            Club_Name   Sport       Dept_title       Event_title
            Is_active   Season      Office_num       Event_date
```

Note about ternary and n-ary relationships

These are covered in the recommended course text in section 2.2, also, but I am skipping them here. I find it more useful to rewrite these as binary relationships with entity classes representing the embedded significant transactions or activities buried within such relationships --- having an enrollment entity class, or an assignment entity class, for example.