Entity Relationship modeling, while good, eventually needed improvements.

Extended Entity Relationship Model (EERM) added some newer and/or more advanced features.

Extended Entity Relationship Diagram (EERD) is the accompanying diagram style.
Let's design a table for professors of a University.
Critical Thinking

Now what if we need to account for... administrators? police? grounds crew? janitors?
A better way?

- If we try to include too many related-but-different entities in a single table we can be left with a lot of null values.
- Additionally, what if certain employees have specific relationships with other entities/tables?
- The solution....
Types

- Types and sub-types are very similar to how we think about object-oriented programming.
- There is a large type at the top (the parent) with related-but-different sub-types (children) below.
  - Ex. An Employee table would be the parent of the Professor, Administrator, etc. tables.
- The "top" type is called the **entity supertype** and children of it are **entity subtypes**.
- Just like other trees, a subtype may be the supertype of its own (sub)tree.
Comparing to OOP

- The parent/super's job is to hold all the common characteristics of its children, who hold the attributes unique to themselves
  - Ex. Parent → SS#, phone #, home address etc.
    Professor → dept., office #, etc.
    Administrator → dept., list of supervisees, etc.

- Because the parent holds all the common attributes, types do exhibit *inheritance* much like OOP

- Note that *relationships* are also inherited
  - For all children of Employee have a 1:M relationship with Dependents for instance
Specialization Hierarchy

- An ERD-like diagram that illustrates the parent-child relationships
- These are going to be "is-a" style relationships, again borrowed from OOP theory
Why no crow's feet here?
Another Example

**FIGURE 6.3** The EMPLOYEE-PILOT supertype-subtype relationship

**Table Name: EMPLOYEE**

<table>
<thead>
<tr>
<th>EMP_NUM</th>
<th>EMP_LNAME</th>
<th>EMP_FNAME</th>
<th>EMP_INITIAL</th>
<th>EMP_HIRE_DATE</th>
<th>EMP_TYPE</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>Kolmycz</td>
<td>Xavier</td>
<td>T</td>
<td>15-Mar-86</td>
<td></td>
</tr>
<tr>
<td>101</td>
<td>Lewis</td>
<td>Marcos</td>
<td>P</td>
<td>25-Apr-89</td>
<td></td>
</tr>
<tr>
<td>102</td>
<td>Vandam</td>
<td>Jean</td>
<td>A</td>
<td>20-Dec-93</td>
<td></td>
</tr>
<tr>
<td>103</td>
<td>Jones</td>
<td>Victoria</td>
<td>R</td>
<td>28-Aug-03</td>
<td></td>
</tr>
<tr>
<td>104</td>
<td>Lange</td>
<td>Edith</td>
<td>P</td>
<td>20-Oct-97</td>
<td></td>
</tr>
<tr>
<td>105</td>
<td>Williams</td>
<td>Gabriel</td>
<td>U</td>
<td>08-Nov-97</td>
<td></td>
</tr>
<tr>
<td>106</td>
<td>Duzak</td>
<td>Mario</td>
<td>P</td>
<td>05-Jan-04</td>
<td></td>
</tr>
<tr>
<td>107</td>
<td>Diante</td>
<td>Venite</td>
<td>L</td>
<td>02-Jul-97</td>
<td></td>
</tr>
<tr>
<td>108</td>
<td>Wiesenbach</td>
<td>Joni</td>
<td>M</td>
<td>18-Nov-95</td>
<td></td>
</tr>
<tr>
<td>109</td>
<td>Travis</td>
<td>Brett</td>
<td>T</td>
<td>14-Apr-01</td>
<td></td>
</tr>
<tr>
<td>110</td>
<td>Genkazi</td>
<td>Stan</td>
<td>A</td>
<td>01-Dec-03</td>
<td></td>
</tr>
</tbody>
</table>

**Table Name: PILOT**

<table>
<thead>
<tr>
<th>EMP_NUM</th>
<th>PIL_LICENSE</th>
<th>PIL_RATINGS</th>
<th>PIL_MED_TYPE</th>
</tr>
</thead>
<tbody>
<tr>
<td>101</td>
<td>ATP</td>
<td>SEL/MEL/Instr/CFII</td>
<td>1</td>
</tr>
<tr>
<td>104</td>
<td>ATP</td>
<td>SEL/MEL/Instr</td>
<td>1</td>
</tr>
<tr>
<td>105</td>
<td>COM</td>
<td>SEL/MEL/Instr/CFI</td>
<td>2</td>
</tr>
<tr>
<td>106</td>
<td>COM</td>
<td>SEL/MEL/Instr</td>
<td>2</td>
</tr>
<tr>
<td>109</td>
<td>COM</td>
<td>SEL/MEL/SES/Instr/CFII</td>
<td>1</td>
</tr>
</tbody>
</table>

Make note of this column!
Implementation Details

- All super-subtype relationships are of and thus implemented as 1:1
- The PK of a subtype is the PK of the parent
  - Which means the subtype's PK is also a...?
  - And its relationship to the parent is what type?
- The attribute and relationship inheritance are only exist at the design level
Okay, if the inheritance isn't enforced by the system, how do we know which subtype an entry belongs to?
A **subtype discriminator** is an attribute in the supertype table that denotes the child type

- Could come in many different forms: a number, a letter, a code, etc.

- In the EERD it is marked by the relationship connector on each child (look back at the specialization hierarchy where it asks about why no crow's feet)
### Subtype Categorization

- **Disjoint or non-overlapping** subtypes → subtypes contain a *unique* subset of parent entity instances
  - An entity instance in the supertype may only be a member of one subtype
  - Ex. A student may only be an undergrad or a grad

- **Overlapping** subtypes → subtypes contain a *nonunique* subset of parent entity instances
  - Entity instances in the supertype can be of any number of subtypes
  - Ex. A baseball player can fill any number of positions as needed
Specialization hierarchy with overlapping subtypes

**PERSON**
- PK: P_ID
- P_LNAME
- P_FNAME

**EMPLOYEE**
- PK,FK1: P_ID
- EMP_HIRE_DATE

**STUDENT**
- PK,FK1: P_ID
- STU_MAJOR

**ADMINISTRATOR**
- PK,FK1: P_ID
- ADM_TITLE

**PROFESSOR**
- PK,FK1: P_ID
- PROF_RANK

**GRADUATE**
- PK,FK1: P_ID
- GRAD_THESIS

**UNDERGRAD**
- PK,FK1: P_ID
- UND_HOURS
Critical Thinking

- If we have disjoint subtypes how is/are subtype discriminator(s) implemented for them in a table?
- What about the case of optional subtypes?
Completeness

- Completeness serves as a constraint on the super-subtype relationship: it denotes whether an entity in the supertype \textit{must} be a member of at least one subtype.
- Cases where it isn't required have \textit{partial completeness}.
- Cases where it is required have \textit{total completeness}. 
<table>
<thead>
<tr>
<th>TYPE</th>
<th>DISJOINT CONSTRAINT</th>
<th>OVERLAPPING CONSTRAINT</th>
</tr>
</thead>
<tbody>
<tr>
<td>Partial</td>
<td>Supertype has optional subtypes. Subtype discriminator can be null. Subtype sets are unique.</td>
<td>Supertype has optional subtypes. Subtype discriminators can be null. Subtype sets are not unique.</td>
</tr>
<tr>
<td>Total</td>
<td>Every supertype occurrence is a member of a (at least one) subtype. Subtype discriminator cannot be null. Subtype sets are unique.</td>
<td>Every supertype occurrence is a member of a (at least one) subtype. Subtype discriminators cannot be null. Subtype sets are not unique.</td>
</tr>
</tbody>
</table>
How do I determine typing?

- As with anything there's choices
  - **Specialization** → top-down approach, identify the supertype and then figure out the subtypes
  - **Generalization** → bottom-up approach, have all the (to be sub-) types available and figure out the appropriate supertype(s)
Entity Clustering

- For large databases the ERD can get very large and cluttered and possibly unreadable.
- Clusters of nearly self-contained entities and relationships may be abstracted out of an ERD and be represented by a virtual entities.
- Clustering should only simplify, not abstract so much that the overall design is no longer clear.
- See Fig. 6.5
  - Offering is serving as a placeholder for the Course-Class relationships and tables.
  - Location is taking the place of Room and Building.
Identifying Keys

- PK's job is to ensure *entity integrity*, or that each row is uniquely identified.
- PKs and FKs are used to implement relationships between entities.
- Picking the right PK has significant impact on efficiency and effectiveness of the database.
Identifying Keys

- A key should *identify* not *describe*
- Often keys will be of the *natural* sort, meaning something from the real world is a PK
  - Ex. social security #, credit card #, etc.
  - Be careful, sometimes they are not a *good* key and something else should be used!
- Keys and their relationships will often be "behind the scenes" knowledge rather than something users deal with
  - Ex. product UPCs, employee ID, etc.
Desirable Characteristics

- Unique Values
- Nonintelligent
- No change over time
- (Preferably) single-attribute
- (Preferably) numeric
- Security Compliant
Composite Keys

- Used mostly in two types of entities:
  - Composite entities because of the M:N relationship
  - Weak entities where the weak entity has a strong relationship with the parent entity
    - Real-world objects that are existence dependent on another object, like Dependent-Employee
    - Real-world object that is represented by two separate entities in a strong relationship, like Line-Invoice
Occasionally a natural key or combination key will not work
- No natural key available or it may not be a good one
- Combination key may be too long and complex, especially if it is used as an FK in another table

In this case use a unique & sequential identifier
- Make sure that it is guaranteed unique and is not null!
Look to the Book!

Case Studies: pg 206
Data Modeling Checklist: pg 212
Homework

- Review Questions: none required, but these would be good to go over for your own studying
- Problems: 1, 2, 4