Entity-Relationship Modelling

- Entities
- Attributes
- Relationships
- Mapping Cardinality
- Keys
- Reduction of an E-R Diagram to Tables
A “enterprise” can be modeled as a collection of:
- entities, and
- relationships among those entities.

An entity is an object that is distinguishable from other objects.
- A specific person, company, automobile, etc.

Entities have attributes:
- People have names and addresses
- An entity and its’ attributes are represented by a tuple
  
  (342-97-4873, Smith, Main, Orlando)

An entity set is a set of entities of the same type, i.e., that share the same properties and attributes.
- Set of all students, set of all companies, set of all automobiles
Entity Sets customer and loan

- **Entity sets:**
  - The set of all *customers* of the Bank
  - The set of all *loans* at the bank

- **Entities:**
  - Customers of the Bank – Bob Jones, Sue Smith, Mark Hayes, etc.
  - Loans at the Bank – L17, L15, L23, etc.

- **Attributes:**
  - Bob Jones has an ID# (321-12-3231), a street address (475 Main Street), a city of residence (Orlando), and a last name (Jones).
  - Loan L17 has an amount ($4537), a date when the loan was taken out (12/15/2009), and a loan number (L17).
Attributes

- The set of permitted values for an attribute is called the **domain** of that attribute.

- Attributes can be:
  - Simple (i.e., atomic) – height in inches, weight in ounces, last-name
  - Composite – name, address, date-of-birth
  - Single-valued – height in inches, date-of-birth, name (any of the above)
  - Multi-valued – phone-numbers, dependents, hobbies
  - Derived – “age” is derived, or rather, computed from “date-of-birth”
Keep in mind that modeling is NOT design!
- during modeling we are focused on what the relevant data is, and not whether or how it will be stored in the database.
- age vs. date-of-birth

This approach is:
- consistent with most text-books
- somewhat *inconsistent* with industry
A **relationship** is an association among two (or more) entities

- Hayes is a *depositor* for account A-102
- The relationship is denoted by a tuple (Hayes, A-102)

A **relationship set** is a set of relationships, all of the same type.
Relationships can be visualized graphically:

```
customer          borrower
  Jones           L-23
    Smith         L-15
    Hayes     L-14
    Jackson
    Curry       L-19
    Williams    L-17
    Adams

loan
```
An attribute can also be property of a relationship set.
Another example of a relationship set having attributes:
- Entities: *Student* and *Course*
- Relationship: *Has-Taken*

Where does the attribute *grade* go?
The **mapping cardinality** of a relationship set expresses the number of entities to which one entity can be associated via the relationship set.

- One to one – US residents & social security #’s
- One to many – academic advisors (assuming at most one major)
- Many to one – same as one-to-many
- Many to many – depositors
Mapping Cardinalities

One to one

Note: Some elements in A and B may not be mapped to any elements in the other set

One to many
Mapping Cardinalities

- Many to one
- Many to many

Note: Some elements in A and B may not be mapped to any elements in the other set.
In the banking enterprise, *access-date* could be an attribute of account instead of a relationship attribute if each account can have only one customer, i.e., if the relationship is one-to-many.
Several ER diagramming techniques have been proposed over the years:

- Chen’s notation - 1976
- IDEF1X (NIST) - 1993
- Crow’s feet (Barker, Palmer, Ellis, et al.) – 1981
- UML (Booch, Jacobson and Rumbaugh) – 1990’s
- Others…

The authors current version is somewhat UML-like, but previously used Chen’s notation.
- Rectangles - entity sets
- Diamonds - relationship sets
- Lines - connect entity sets to relationship sets.
- Underlined Attributes – primary key attributes
Notes:
- An ER diagram is typically accompanied by a document that defines all the terms.
- Much harder to do than it appears (e.g., what is an “orbit” for a satellite?)
- In many applications the terms are much more ambiguous (e.g., function designators)
Relationship Sets with Attributes

customer
  ID
  name
  street
  city

depositor

account
  number
  balance

access-date
Mapping cardinality is indicated by drawing a directed line (→), signifying “one,” or an undirected line (—), signifying “many,” between the relationship and the entity.

One-to-one relationship:
- A customer would be associated with at most one loan
- A loan would be associated with at most one customer
One-to-many relationship - a customer is associated with zero or more one loans, and a loan is associated with at most one customer.
Many-to-one relationship - a loan is associated with zero or more customers, and a customer is associated with at most one loan.
Many-to-many relationship - a customer is associated with zero or more loans, and a loan is associated with zero of more customers.
- Total participation - every entity in an entity set must participate in the relationship set; indicated by a double-line and a double-diamond.

- If participation in a relationship is optional for some entities then that entity set is said to have *partial participation* in the relationship.
Another version of the notation uses specific cardinality limits:
A **super key** of an entity set is a set of one or more attributes that uniquely identify each entity in the entity set.

A **candidate key** of an entity set is a *minimal* super key

- customer-id is candidate key of *customer*
- account-number is candidate key of *account*

Although several candidate keys may exist, one of the candidate keys is selected to be the **primary key**.

- All others are referred to as **secondary keys**
- Book says selection of the primary key is arbitrary, but this is not true.

Later we will also discuss **foreign keys** and **search keys**.
Other Examples of Keys

- **Student = (SS#, Name, Date-of-Birth, Status)**
  - SS# - Super key and candidate key
  - SS#, Name, DOB - Super key, but not candidate key
  - SS#, Status - Super key, but not candidate key
  - Name - Neither
Other Examples of Keys

- Payment = (Payment#, Loan#, Date-Made, ID#)
  - ID# - Super key, and candidate key
  - Payment#, Loan# - Super key, and candidate key
  - ID#, Date-Made - Super key, but not candidate key
  - Date-Made - Neither

- ID# is frequently referred to as a **pseudo-key**.
A primary key of an entity set is specified in an ER diagram by underlining the key attributes.
Much like an entity set, a relationship set can also have a super key.

The combination of primary keys of the participating entity sets forms a super key of a relationship set.

- \((\text{customer-id}, \text{account-number})\) is the super key of \text{depositor}

The mapping cardinality of a relationship set will determine, in part, what the candidate keys of the relationship set are.

If the relationship is one-to-one, then using just one of the primary keys of the participating entity sets is, in fact, minimal, and hence forms a candidate key for the relationship set.
Frequently there are many ways to model a given situation.

- Use of an entity vs. an attribute:
  - Is *Telephone-Number* an attribute or an entity?

- Use of an entity vs. a relationship:
  - Is *Loan* an entity or a relationship?

- Placement of relationship attributes
Construct an ER diagram for a car insurance company whose customers own one or more cars each. Each car has associated with it zero to any number of recorded accidents.
Let's Try an Example!

From the 6th edition…

Construct an ER diagram for a car insurance company whose customers own one or more cars each. Each car has associated with it zero to any number of recorded accidents. Each insurance policy covers one or more cars, and has one or more premium payments associated with it. Each payment is for a particular period of time, and has an associated due date, and the date when the payment was received.
Examples and Exercises

- Look for other ER diagramming exercises in the book, at the end of the chapter, or online.

- Google image search “ER diagram example”

- One example:
Strong vs. Weak Entity Sets

- Recall the loan and payment entity sets:

  ![Diagram of loan and payment entity sets]

- Now consider the following: (notice the ambiguity)

  ![Diagram showing ambiguity between loan and payment]
For most entity sets, a primary key can be specified in terms of its immediate attributes.

Such an entity set is referred to as a **strong entity set**.

For some entities, however, it is helpful to specify its’ primary key, at least in part, in terms of some other entities’ attributes.

Such an entity set is referred to as a **weak entity set**.

A weak entity set is typically associated with an **identifying entity set** (which is usually strong) via a total, one-to-many relationship.
Strong vs. Weak Entity Sets

- Expressed as strong & weak entity sets:

  loan
  \[ \begin{array}{c}
  \text{ID} \\
  \text{amount} \\
  \end{array} \]
  \hspace{1cm}
  loan-payment
  \hspace{1cm}
  payment
  \[ \begin{array}{c}
  \text{number} \\
  \text{date} \\
  \text{amount} \\
  \end{array} \]

- The relationship between the strong and weak entity sets is specified with a double-diamond.
Strong vs. Weak Entity Sets

Note how this resolves the ambiguity:
In such a case, the (weak) entity typically has a subset of attributes, called a **discriminator (or partial key)**, that distinguishes among all entities of the weak entity set associated with one identifying entity.

The discriminator is underlined with a dashed line.

A primary key for the weak entity set consists of two parts:
- The primary key of the associated identifying entity set
- The weak entity set’s discriminator

Primary key for payment is \((\text{loan-number, payment-number})\)
In the university enterprise, a *course* is a strong entity and a *section* can be modeled as a weak entity.

- The discriminator of *section* would be *sec-id*, *semester* and *year*.
Specialization and Generalization - Inheritance

- person
  - ID
  - name
  - address

- employee
  - salary
  - credit-rating

- officer
  - office-number

- teller
  - station-number
  - hours-worked

- secretary
  - hours-worked

- customer
  - credit-rating
Specialization

- Inheritance relationships also referred to as a *superclass-subclass* relationships.

- Lower-level entity sets:
  - Have attributes that do not apply to the higher-level entity set.
  - Participate in relationships that do not apply to the higher-level entity set, e.g., airline employees, pilots, crew, agents, etc., but only pilots are certified to fly certain aircraft types.

- Lower-level entity sets are said to *inherit* all the attributes and relationships from the higher-level entity sets to which they are linked.
Specialization and Generalization - Inheritance

- **person**
  - ID
  - name
  - address

- **employee**
  - salary

- **customer**
  - credit-rating

- **TQM-team**
  - name
  - mission
  - budget

- **assigned-to**

- **officer**
  - office-number

- **teller**
  - station-number
  - hours-worked

- **secretary**
  - hours-worked
Specialization vs. Generalization

- Top-down design process: we designate sub-groupings within an entity set that are distinctive from other entities in the set.

- Bottom-up design process: combine a number of entity sets that share the same features into a higher-level entity set.

- The terms specialization and generalization are used interchangeably, for the obvious reasons.
A specialization/generalization relationship can be:
- disjoint vs. overlapping – notated in any number of ways.
- total vs. partial – notated as before.

Multiple specializations of an entity set are possible:
- permanent-employee vs. temporary-employee
- In addition to officer vs. secretary vs. teller

Each particular employee would be a member of:
- one of permanent-employee or temporary-employee, and
- one of officer, secretary, or teller

Multiple inheritance
E-R Diagram for a University
E-R Diagram for a Banking Enterprise (Chen’s Notation)
E-R Diagram for the Banking Enterprise

account

account-number
balance

depositor

branch

branch-name
Branch-city
assets

customer

customer-name
customer-street
customer-city

borrower

loan

loan-number
amount

account-branch

loan-branch
Alternative E-R Notations

Entity set E with attributes A1, A2, A3 and primary key A1

<table>
<thead>
<tr>
<th>E</th>
</tr>
</thead>
<tbody>
<tr>
<td>A1</td>
</tr>
<tr>
<td>A2</td>
</tr>
<tr>
<td>A3</td>
</tr>
</tbody>
</table>

Many to Many Relationship

One to One Relationship

Many to One Relationship
ER Diagramming Practice

- Practice developing ER diagrams:
  - see exercises at the end of the chapter on ER diagrams
  - use your imagination!

- Possible enterprises to model:
  - airline or airport
  - hospital
  - Insurance company
  - library
  - retailer – clothing, food, equipment
  - your favorite government agency
Reduction of an E-R Schema to Tables

- Converting an E-R diagram to a relational database:
  - Each entity set is converted to its’ own table.
  - Each relationship can be (but may not be) converted to its’ own table.

- Each table has a number of columns, which generally corresponding to the attributes in the corresponding entity or relationship set.

- The resulting tables can be modified in a variety of ways to support performance, space, or other requirements.
A strong entity set reduces to a table with the same attributes.

<table>
<thead>
<tr>
<th>customer-id</th>
<th>customer-name</th>
<th>customer-street</th>
<th>customer-city</th>
</tr>
</thead>
<tbody>
<tr>
<td>019-28-3746</td>
<td>Smith</td>
<td>North</td>
<td>Rye</td>
</tr>
<tr>
<td>182-73-6091</td>
<td>Turner</td>
<td>Putnam</td>
<td>Stamford</td>
</tr>
<tr>
<td>192-83-7465</td>
<td>Johnson</td>
<td>Alma</td>
<td>Palo Alto</td>
</tr>
<tr>
<td>244-66-8800</td>
<td>Curry</td>
<td>North</td>
<td>Rye</td>
</tr>
<tr>
<td>321-12-3123</td>
<td>Jones</td>
<td>Main</td>
<td>Harrison</td>
</tr>
<tr>
<td>335-57-7991</td>
<td>Adams</td>
<td>Spring</td>
<td>Pittsfield</td>
</tr>
<tr>
<td>336-66-9999</td>
<td>Lindsay</td>
<td>Park</td>
<td>Pittsfield</td>
</tr>
<tr>
<td>677-89-9011</td>
<td>Hayes</td>
<td>Main</td>
<td>Harrison</td>
</tr>
<tr>
<td>963-96-3963</td>
<td>Williams</td>
<td>Nassau</td>
<td>Princeton</td>
</tr>
</tbody>
</table>
Composite and Multi-valued Attributes

- Composite attributes are broken up.

- A multi-valued attribute $M$ of entity $E$ is represented by a new table with the following attributes:
  - The primary key of $E$
  - An attribute corresponding to multi-valued attribute $M$
Example:

<table>
<thead>
<tr>
<th>employee</th>
</tr>
</thead>
<tbody>
<tr>
<td>id#</td>
</tr>
<tr>
<td>name</td>
</tr>
<tr>
<td>first</td>
</tr>
<tr>
<td>middle</td>
</tr>
<tr>
<td>last</td>
</tr>
<tr>
<td>phone-number</td>
</tr>
<tr>
<td>{ dependent }</td>
</tr>
</tbody>
</table>

Tables:

employee (id#, first-name, middle-name, last-name, phone-number)
dependent (id#, dname)
A weak entity set becomes a table that includes a column for the primary key of the identifying strong entity set.

<table>
<thead>
<tr>
<th>loan-number</th>
<th>payment-number</th>
<th>payment-date</th>
<th>payment-amount</th>
</tr>
</thead>
<tbody>
<tr>
<td>L-11</td>
<td>53</td>
<td>7 June 2001</td>
<td>125</td>
</tr>
<tr>
<td>L-14</td>
<td>69</td>
<td>28 May 2001</td>
<td>500</td>
</tr>
<tr>
<td>L-15</td>
<td>22</td>
<td>23 May 2001</td>
<td>300</td>
</tr>
<tr>
<td>L-16</td>
<td>58</td>
<td>18 June 2001</td>
<td>135</td>
</tr>
<tr>
<td>L-17</td>
<td>5</td>
<td>10 May 2001</td>
<td>50</td>
</tr>
<tr>
<td>L-17</td>
<td>6</td>
<td>7 June 2001</td>
<td>50</td>
</tr>
<tr>
<td>L-17</td>
<td>7</td>
<td>17 June 2001</td>
<td>100</td>
</tr>
<tr>
<td>L-23</td>
<td>11</td>
<td>17 May 2001</td>
<td>75</td>
</tr>
<tr>
<td>L-93</td>
<td>103</td>
<td>3 June 2001</td>
<td>900</td>
</tr>
<tr>
<td>L-93</td>
<td>104</td>
<td>13 June 2001</td>
<td>200</td>
</tr>
</tbody>
</table>
A many-to-many relationship set is represented as a table with columns for the primary keys of the two participating entity sets, and any descriptive attributes of the relationship set.

Example: the borrower relationship set:

<table>
<thead>
<tr>
<th>customer-id</th>
<th>loan-number</th>
</tr>
</thead>
<tbody>
<tr>
<td>019-28-3746</td>
<td>L-11</td>
</tr>
<tr>
<td>019-28-3746</td>
<td>L-23</td>
</tr>
<tr>
<td>244-66-8800</td>
<td>L-93</td>
</tr>
<tr>
<td>321-12-3123</td>
<td>L-17</td>
</tr>
<tr>
<td>335-57-7991</td>
<td>L-16</td>
</tr>
<tr>
<td>555-55-5555</td>
<td>L-14</td>
</tr>
<tr>
<td>677-89-9011</td>
<td>L-15</td>
</tr>
<tr>
<td>963-96-3963</td>
<td>L-17</td>
</tr>
</tbody>
</table>
A many-to-one relationship set can be represented just like a many-to-many relationship.

Technically this is not necessary, and in some cases it does not result in a good design.

Example:

Relationship Set (total, many-to-one from account to branch)
The preceding could be converted to 3 tables directly, or as follows:

account (account-number, balance, branch-name)
branch (branch-name, branch-city, assets)

Since the above relationship is total, this makes sense.
- by the way, eliminating an unnecessary table is frequently considered…cool…

On the other hand, suppose:
- the relationship is partial
- lots of accounts
- most accounts don’t have branches

Consider a query that looks up account #’s for a given branch-name.
- In this case, 3 tables are potentially better (why?).
For one-to-one relationship sets, the extra attribute can be added to either of the tables corresponding to the two entity sets.

Other relationship attributes would be treated similarly.

Note that either of the above could introduce null values if the relationship is not total.
Note: This discussion assumes a 2-level inheritance hierarchy.

- Exercise: Generalize it to an arbitrarily deep hierarchy.

**Method 1:**

- Form a table for the higher level entity set.
- Form a table for each lower level entity set, including the primary key of the higher level entity set and local attributes.

<table>
<thead>
<tr>
<th>table</th>
<th>attributes</th>
</tr>
</thead>
<tbody>
<tr>
<td>person</td>
<td>name, street, city</td>
</tr>
<tr>
<td>customer</td>
<td>name, credit-rating</td>
</tr>
<tr>
<td>employee</td>
<td>name, salary</td>
</tr>
</tbody>
</table>

- One Drawback: getting information about specific entities requires accessing two tables
Method 2:

- Form a table for each entity set with all local and inherited attributes.

<table>
<thead>
<tr>
<th>table</th>
<th>table attributes</th>
</tr>
</thead>
<tbody>
<tr>
<td>person</td>
<td>name, street, city</td>
</tr>
<tr>
<td>customer</td>
<td>name, street, city, credit-rating</td>
</tr>
<tr>
<td>employee</td>
<td>name, street, city, salary</td>
</tr>
</tbody>
</table>

This method has obvious redundancies.

- Particularly bad for persons who are both customers and employees.

If specialization is total, the table for the generalized entity is redundant.

- Temptation is to delete the person table; still might be needed for foreign key constraints.
Method 3: (not presented in the book)
- Form one table for the higher level entity set
- This table has one column for every attribute in every subclass.

<table>
<thead>
<tr>
<th>table</th>
<th>attributes</th>
</tr>
</thead>
<tbody>
<tr>
<td>person</td>
<td>name, street, city, credit-rating, salary</td>
</tr>
</tbody>
</table>

- Optionally, include a type attribute that indicates which subclass the stored entity belongs to.

Obvious drawback - contains multiple nullable attributes.

Sometimes referred to as a “junk drawer”
E-R Diagram for the Banking Enterprise

- account
  - account-number
  - balance
- depositor
- customer
  - customer-name
  - customer-street
  - customer-city
- borrower
- loan
  - loan-number
  - amount
- branch
  - branch-name
  - Branch-city
  - assets
- account-branch
- loan-branch
The following relational schemes result:

- **branch** (*branch-name*, *branch-city*, *assets*)
- **customer** (*customer-name*, *customer-street*, *customer-city*)
- **account** (*account-number*, *branch-name*, *balance*)
- **loan** (*loan-number*, *branch-name*, *amount*)
- **depositor** (*customer-name*, *account-number*)
- **borrower** (*customer-name*, *loan-number*)
Classroom (building, room-number, capacity)
Department (dept-name, building, budget)
Course (course-id, title, dept-name, credits)
Instructor (ID, name, depart-name, salary)
Section (course-id, sec-id, semester, year, building, room-number, time-slot-id)
Teaches (ID, course-id, sec-id, semester, year)
Student (ID, name, dept-name, tot-cred)
Takes (ID, course-id, sec-id, semester, year, grade)
Advisor (s-ID, i-ID)
Time-slot (time-slot-id, day, start-time, end-time)
Prereq (course-id, prereq-id)
Schema Diagram for a University

- **Student**
  - ID
  - name
  - dept_name
  - tot_cred

- **Course**
  - course_id
  - title
  - dept_name
  - credits

- **Department**
  - dept_name
  - building
  - budget

- **Advisor**
  - s_id
  - i_id

- **Section**
  - course_id
  - sec_id
  - semester
  - year
  - building
  - room_no
  - time_slot_id

- **Time_slot**
  - time_slot_id
  - day
  - start_time
  - end_time

- **Classroom**
  - building
  - room_no
  - capacity

- **Teaches**
  - ID
  - course_id
  - sec_id
  - semester
  - year

- **Prereq**
  - course_id
  - prereq_id

- **Instructor**
  - ID
  - name
  - dept_name
  - salary
Note the advisor relationship is one-to-many, but in the schema it has been implemented as its' own table.

Similarly, time-slot is implemented slightly differently than described…
End of Chapter 6