The Entity-Relationship Model (ER Model) - Part 2

By Michael Hahsler

Based on slides for CS145 Introduction to Databases (Stanford)
What you will learn about in this section

1. Relationships: multiplicity, multi-way

2. Design considerations

3. Conversion to SQL

4. Advanced concepts
Multiplicity of ER Relationships

Using Chen’s Notation

One-to-one:

Many-to-one:

One-to-many:

Many-to-many:
How to read a relationship in both directions:
1. A product is made by a one company
2. A company makes many product
No specified cardinality often means N:M, or we do not want to decide, yet.
Multi-way Relationships

How do we model “A person buys a product in a store?”
Q: What do the 1s and the N mean?

Product \( \xrightarrow{N} \) Purchase \( \xrightarrow{1} \) Store

Person \( \xrightarrow{1} \) Purchase
Better: many to many to many relationship
Conversion of Multi-way Relationship to New Entity + Binary Relationships?

Multi-way Relationship

- Product
- Purchase
- Store
- Person

→

Entity + Binary

- Purchase
- ID
- date
- N
- N
- 1
- N
- N
- 1
- 1
- 1

- Product
- ProductOf
- N
- 1
- Store
- StoreOf
- 1
- Person
- BuyerOf
- 1

Multiple purchases per (product, store, person) possible here!
3. Design Principles

What’s wrong with these examples?

- Product 1 Purchase N Person
- Country President Person
Design Principles: What’s Wrong?
Design Principles:
What’s Wrong? - Fixed
Examples: Entity vs. Attribute

Should address be an attribute?

Or an entity?
Examples: Entity vs. Attribute

Should address be an attribute?

- How do we handle employees with more than two addresses?

- How do we handle addresses where internal structure of the address (e.g. zip code, state) is useful?
Examples: Entity vs. Attribute

Use an entity

In general, when we want to record several values, we choose a separate entity.
From ER Diagrams to Relational Schema

• Key concept:

Both *Entity sets* and *Relationships* become relations (tables in RDBMS)
From ER Diagrams to Relational Schema

- An entity set becomes a relation (multiset of tuples / table)
  - Each tuple is one entity
  - Each tuple is composed of the entity’s attributes, and has the same primary key

<table>
<thead>
<tr>
<th>Product</th>
<th>name</th>
<th>price</th>
<th>category</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gizmo1</td>
<td>99.99</td>
<td>Camera</td>
<td></td>
</tr>
<tr>
<td>Gizmo2</td>
<td>19.99</td>
<td>Edible</td>
<td></td>
</tr>
</tbody>
</table>
From ER Diagrams to Relational Schema

```sql
CREATE TABLE Product(
    name     CHAR(50) PRIMARY KEY,
    price    DOUBLE,
    category VARCHAR(30)
)
```
From ER Diagrams to Relational Schema (N:M)

- A relation between entity sets $A_1, \ldots, A_N$ also becomes a multiset of tuples / a table
  - Each row/tuple is one relation, i.e. one unique combination of entities $(a_1, \ldots, a_N)$
  - Each row/tuple is
    - composed of the union of the entity sets’ attributes
    - has the entities’ primary keys as foreign keys
    - has the union of the entity sets’ keys as primary key

<table>
<thead>
<tr>
<th></th>
<th>name</th>
<th>firstname</th>
<th>lastname</th>
<th>date</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gizmo1</td>
<td>Bob</td>
<td>Joe</td>
<td></td>
<td>01/01/15</td>
</tr>
<tr>
<td>Gizmo2</td>
<td>Joe</td>
<td>Bob</td>
<td></td>
<td>01/03/15</td>
</tr>
<tr>
<td>Gizmo1</td>
<td>JoeBob</td>
<td>Smith</td>
<td></td>
<td>01/05/15</td>
</tr>
</tbody>
</table>
CREATE TABLE Purchased(
    name CHAR(50),
    firstname CHAR(50),
    lastname CHAR(50),
    date DATE,
    PRIMARY KEY (name, firstname, lastname),
    FOREIGN KEY (name)
        REFERENCES Product,
    FOREIGN KEY (firstname, lastname)
        REFERENCES Person
)

<table>
<thead>
<tr>
<th>Purchased</th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>firstname</td>
<td>lastname</td>
<td>date</td>
</tr>
<tr>
<td>Gizmo1</td>
<td>Bob</td>
<td>Joe</td>
<td>01/01/15</td>
</tr>
<tr>
<td>Gizmo2</td>
<td>Joe</td>
<td>Bob</td>
<td>01/03/15</td>
</tr>
<tr>
<td>Gizmo1</td>
<td>JoeBob</td>
<td>Smith</td>
<td>01/05/15</td>
</tr>
</tbody>
</table>
From ER Diagrams to Relational Schema (1:N)

- A 1:N relationship can be implemented without an extra table.
- Add the primary key of the “1 side” to the table for the “N side” entity.
CREATE TABLE Address(
    ID       CHAR(50),
    Number  CHAR(50),
    Street   CHAR(50),
    ZIPCode       CHAR(10),
    PRIMARY KEY (ID),
    FOREIGN KEY (CustID)
        REFERENCES Customer,
    REFERENCES Customer
)
From ER Diagram to Relational Schema

How do we represent this as a relational schema?
Alternative Notations

Chen
- Person \( \rightarrow \) Birthplace \( \rightarrow \) Location

IDEF1X
- Person \( \rightarrow \) Location

Bachman
- Person \( \rightarrow \) Born in \( \rightarrow \) Birthplace of \( \rightarrow \) Location

Martin / IE / Crow’s Foot
- Person \( \rightarrow \) Born in \( \rightarrow \) Birthplace of \( \rightarrow \) Location

Min-Max / ISO
- Person \( \rightarrow \) (1,1) Born in \( \rightarrow \) Birthplace of (0,N) \( \rightarrow \) Location

UML
- \( <<\text{Entity}>> \) Person \( \rightarrow \) \( <<\text{Relationship}>> \) Born in \( \rightarrow \) 0..N \( \rightarrow \) Birthplace of \( \rightarrow \) 1 \( \rightarrow \) \( <<\text{Entity}>> \) Location
Add Multiplicity to your ER diagram

Also make sure to add **(new concepts underlined):**

A player can only belong to one team, a play can only be in one game, a pass/run..?

Multiple players **Tackle** a single person in a play

Players can achieve a **Personal Record** linked to a specific Game and Play

Players have a **weight** which changes in on vs. off-season
3. Advanced ER Concepts
What you will learn about in this section

1. Subclasses
2. Constraints
3. Weak entity sets
4. Normal Forms
Modeling Subclasses

• Some objects in a class may be special, i.e. worthy of their own class

• Define a new class?
  • But what if we want to maintain connection to current class?

• Better: define a *subclass*
  • Ex:

```
Products
  /   \
Software products  Educational products
```

*We can define subclasses in ER!*
Modeling Subclasses

Child subclasses contain all the attributes of all of their parent classes plus the new attributes shown attached to them in the ER diagram.
Understanding Subclasses

• Think in terms of records; ex:

  • Product
  • SoftwareProduct
  • EducationalProduct

Child subclasses contain all the attributes of all of their parent classes plus the new attributes shown attached to them in the ER diagram.
Think like tables…

<table>
<thead>
<tr>
<th>Product</th>
<th>name</th>
<th>price</th>
<th>category</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gizmo</td>
<td>99</td>
<td>gadget</td>
<td></td>
</tr>
<tr>
<td>Camera</td>
<td>49</td>
<td>photo</td>
<td></td>
</tr>
<tr>
<td>Toy</td>
<td>39</td>
<td>gadget</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Software Product</th>
<th>name</th>
<th>platforms</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gizmo</td>
<td>unix</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Educational Product</th>
<th>name</th>
<th>ageGroup</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gizmo</td>
<td>todler</td>
<td></td>
</tr>
<tr>
<td>Toy</td>
<td>retired</td>
<td></td>
</tr>
</tbody>
</table>
IsA Review

• If we declare $A \text{ IsA } B$ then every $A$ is a $B$

• We use IsA to
  • Add descriptive attributes to a subclass
  • To identify entities that participate in a relationship

• No need for multiple inheritance
Modeling UnionTypes With Subclasses

Suppose each piece of furniture is owned either by a person, or by a company. *How do we represent this?*
Modeling Union Types with Subclasses

Say: each piece of furniture is owned either by a person, or by a company

**Solution 1.** Acceptable, but imperfect (What’s wrong?)
Modeling Union Types with Subclasses

Solution 2: better (though more laborious)

What is happening here?
Constraints in ER Diagrams

• Finding constraints is part of the ER modeling process. Commonly used constraints are:

• **Keys**: Implicit constraints on uniqueness of entities
  • *Ex: An SSN uniquely identifies a person*

• **Single-value constraints:**
  • *Ex: a person can have only one father*

• **Referential integrity constraints**: Referenced entities must exist
  • *Ex: if you work for a company, it must exist in the database*

• **Other constraints:**
  • *Ex: peoples’ ages are between 0 and 150*
Participation Constraints: Partial v. Total

Are there products made by no company? Companies that don’t make a product?

Double line indicates \textit{total participation} (i.e. here: all products are made by a company)
Single Value Constraints

N makes 1

vs.

makes
Referential Integrity

Express that each product is made by exactly one company:

Total Participation + Multiplicity

Each product made by at least one company.

Each product made by at most one company.
Keys in ER Diagrams

Underline keys:

Note: no formal way to specify *multiple* keys in ER diagrams...
Weak Entity Sets

Entity sets are *weak* when their key comes from other classes to which they are related.

(E.g., Berkeley has a football team too, sort of)
Weak Entity Sets

Entity sets are **weak** when their key comes from other classes to which they are related.

- number is a *partial key*. (denote with dashed underline).
- University is called the *identifying owner*.
- Participation in affiliation must be total. Why?
ER Summary

• ER diagrams are a visual syntax that allows technical and non-technical people to talk
  • For conceptual design

• Basic constructs: entity, relationship, and attributes

• A good design is faithful to the constraints of the application, but not overzealous
Design Theory

• Design theory is about how to represent your data to avoid anomalies.

• It is a mostly mechanical process
  • Tools can carry out routine portions

• We have a notebook implementing all algorithms!
  • We’ll play with it in the activities!
Normal Forms

• 1\textsuperscript{st} Normal Form (1NF) = All tables are flat

• 2\textsuperscript{nd} Normal Form = disused

• Boyce-Codd Normal Form (BCNF)

• 3\textsuperscript{rd} Normal Form (3NF) DB designs based on \textit{functional dependencies}, intended to prevent data \textit{anomalies}

• 4\textsuperscript{th} and 5\textsuperscript{th} Normal Forms
1st Normal Form (1NF)

<table>
<thead>
<tr>
<th>Student</th>
<th>Courses</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mary</td>
<td>{CS145,CS229}</td>
</tr>
<tr>
<td>Joe</td>
<td>{CS145,CS106}</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

1NF: Entries have to be single values not a list!
Issues with 1NF

<table>
<thead>
<tr>
<th>Student</th>
<th>Course</th>
<th>Room</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mary</td>
<td>CS145</td>
<td>B01</td>
</tr>
<tr>
<td>Joe</td>
<td>CS145</td>
<td>B01</td>
</tr>
<tr>
<td>Sam</td>
<td>CS145</td>
<td>B01</td>
</tr>
<tr>
<td></td>
<td>..</td>
<td>..</td>
</tr>
<tr>
<td></td>
<td>..</td>
<td>..</td>
</tr>
</tbody>
</table>

If every course is in only one room, contains **redundant** information!
Issues with 1NF

<table>
<thead>
<tr>
<th>Student</th>
<th>Course</th>
<th>Room</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mary</td>
<td>CS145</td>
<td>B01</td>
</tr>
<tr>
<td>Joe</td>
<td>CS145</td>
<td>C12</td>
</tr>
<tr>
<td>Sam</td>
<td>CS145</td>
<td>B01</td>
</tr>
<tr>
<td>..</td>
<td>..</td>
<td>..</td>
</tr>
</tbody>
</table>

If we update the room number for one tuple, we get inconsistent data = an update anomaly.
Issues with 1NF

If everyone drops the class, we lose what room the class is in! = a delete anomaly
Issues with 1NF

Similarly, we can’t reserve a room without students = an *insert* anomaly

<table>
<thead>
<tr>
<th>Student</th>
<th>Course</th>
<th>Room</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mary</td>
<td>CS145</td>
<td>B01</td>
</tr>
<tr>
<td>Joe</td>
<td>CS145</td>
<td>B01</td>
</tr>
<tr>
<td>Sam</td>
<td>CS145</td>
<td>B01</td>
</tr>
</tbody>
</table>

...
### Issues with 1NF: Split Table

<table>
<thead>
<tr>
<th>Student</th>
<th>Course</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mary</td>
<td>CS145</td>
</tr>
<tr>
<td>Joe</td>
<td>CS145</td>
</tr>
<tr>
<td>Sam</td>
<td>CS145</td>
</tr>
<tr>
<td>..</td>
<td>..</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Course</th>
<th>Room</th>
</tr>
</thead>
<tbody>
<tr>
<td>CS145</td>
<td>B01</td>
</tr>
<tr>
<td>CS229</td>
<td>C12</td>
</tr>
</tbody>
</table>

Is this form better?
- Redundancy?
- Update anomaly?
- Delete anomaly?
- Insert anomaly?

How do we decide how to split the table?
ER Model (if done right) already creates 3NF

Corresponding tables:

**Enrollment**

<table>
<thead>
<tr>
<th>Student</th>
<th>Course</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mary</td>
<td>CS145</td>
</tr>
<tr>
<td>Joe</td>
<td>CS145</td>
</tr>
<tr>
<td>Sam</td>
<td>CS145</td>
</tr>
<tr>
<td>..</td>
<td>..</td>
</tr>
</tbody>
</table>

**Course**

<table>
<thead>
<tr>
<th>CourseNr</th>
<th>Room</th>
</tr>
</thead>
<tbody>
<tr>
<td>CS145</td>
<td>B01</td>
</tr>
<tr>
<td>CS229</td>
<td>C12</td>
</tr>
</tbody>
</table>
In short: The data depends on the key [1NF], the whole key [2NF] and nothing but the key [3NF]
Conclusion

• Databases should be in 3. NF to avoid anomalies.

• Some databases are poorly designed leading to bad data quality as a result of anomalies.

• Sometimes databases are on purpose not designed in 3. NF. E.g., for performance or other reasons. In this case, the application/user needs to prevent anomalies (which does not always work).

• For analytics purposes we often need a single table with all the data. Essentially this table is in 1. NF.