The Entity/Relationship (E/R) Model & DB Design

Introduction to databases

CSC343, Fall 2015

Bogdan Simion

Thanks to Manos Papaggelis, Ryan Johnson, John Mylopoulos, Arnold Rosenbloom and Renee Miller for material in these slides
Announcements

• Extra tutorial for BCNF/3NF
  – TODAY 4-6pm, in BA1130!
  – More complex examples of decomposition/synthesis
  – Optional, no marks
  – Solutions will be available afterwards too, but try to make it and work through the examples with the TA

• Lecture prep marks (reminder):
  – We will consider best 7 prep marks you get, 1% each

• A2 marks, coming soon..
Overview

• Using the Entity/Relationship (ER) Model to model the real world

• From there, designing a database schema
  – Restructuring of an E/R model
  – Translating an E/R model into a logical model (DB Schema)
THE ENTITY/RELATIONSHIP (E/R) MODEL
Conceptualizing the real-world

• DB design begins with a boss or client who wants a database.

• We must map the entities and relationships of the world into the concepts of a database. This is called *modeling*.

• Sketching the key components is an efficient way to develop a design.
  – Sketch out (and debug) schema designs
  – Express as many constraints as possible
  – Convert to relational DB once the client is happy
Entity/Relationship Model

• Visual data model (diagram-based)
  – Quickly “chart out” a database design
  – Easier to “see” big picture
  – Comparable to class diagrams in UML

• Basic concept: *entities* and their *relationships*, along with the *attributes* describing them
<table>
<thead>
<tr>
<th>“thing” to be modeled</th>
<th>$E/R$</th>
<th>OO</th>
<th>RA</th>
</tr>
</thead>
<tbody>
<tr>
<td>“thing” to be modeled</td>
<td>$Entity$</td>
<td>Object</td>
<td>Tuple</td>
</tr>
<tr>
<td>set of similar “things”</td>
<td>$Entity$</td>
<td>Class</td>
<td>Relation</td>
</tr>
<tr>
<td>relationship</td>
<td>$Relationship$</td>
<td>Object?</td>
<td>Tuple?</td>
</tr>
<tr>
<td>set of similar relationships</td>
<td>$Relationship set$</td>
<td>Class?</td>
<td>Relation?</td>
</tr>
<tr>
<td>property of a “thing” or of a relationship</td>
<td>$Attribute$</td>
<td>Field</td>
<td>Attribute</td>
</tr>
</tbody>
</table>
Entity Sets

- An **entity set** represents a category of objects that have properties in common and an autonomous existence (e.g., City, Department, Employee, Sale)
- An **entity** is an instance of an entity set (e.g., Stockholm is a City; Peterson is an Employee)
Relationship Sets

- A relationship set is an association between 2+ entity sets (e.g., Residence is a relationship set between entity sets City and Employee)
- A relationship is an instance of a n-ary relationship set (e.g., the pair <Johanssen, Stockholm> is an instance of relationship Residence)
Example of Instances for Exam

A student can’t take more than one exam for a particular course
Recursive Relationships

• Recursive relationships relate an entity to itself

• Note in the second example that the relationship is not symmetric
  - In this case, it is necessary to indicate the two roles that the entity plays in the relationship
Ternary Relationships
Attributes

• Describe elementary properties of entities or relationships (e.g., Surname, Salary and Age are attributes of Employee)
• May be single-valued, or multi-valued
Composite Attributes

- **composite attributes** are grouped attributes of the same entity or relationship that have closely connected meaning or uses.
Example Schema with Attributes
Cardinalities

- Each entity set participates in a relationship set with a minimum (min) and a maximum (max) cardinality.
- Cardinalities constrain how entity instances participate in relationship instances.
- Graphical representation in E/R Diagrams: pairs of (min, max) values for each entity set.

An entity might not participate in any relationship.
Cardinalities (cont.)

- In principle, cardinalities are pairs of non-negative integers \((n, N)\) such that \(n \leq N\), where \(N\) means "any number"

- **minimum cardinality \(n\):**
  - If \(0\), entity participation in a relationship is **optional**
  - If \(1\), entity participation in a relationship is **mandatory**

- **maximum cardinality \(N\):**
  - If \(1\), each instance of the entity is associated **at most with a single instance** of the relationship
  - If \(N\), then each instance of the entity **may be associated with many instances** of the relationship. Why **may be**?
Cardinality Examples

ORDER (0,1) SALE (1,1) INVOICE

PERSON (1,1) RESIDENCE (0,N) CITY

TOURIST (1,N) RESERVATION (0,N) VOYAGE
Multiplicity of relationships

If entities E1 and E2 participate in relationship R with cardinalities (n1, N1) and (n2, N2), then \( \Rightarrow \) the multiplicity of R is N1-to-N2 (which is the same as saying N2-to-N1)

\[
1 \to 1 \quad \subseteq \quad N \to 1 \text{ OR } 1 \to N \quad \subseteq \quad N \to N
\]
Cardinalities of Attributes

- Describe min/max number of values an attribute can have
- When the cardinality of an attribute is (1, 1) it can be omitted (single-valued attributes)
- The value of an attribute, may also be null (“omittable”, e.g. 0:1 cardinality), or have several values (multi-valued attributes)
Cardinalities of Attributes (cont.)

- Multi-valued attributes often represent situations that can be modeled with additional entities. E.g., the ER schema of the previous slide can be revised into:

```
Person
    Surname
    License Number

Owns
    (0,N)

Car
    CarRegistration#
```
Keys in E/R

- **Keys** consist of minimal sets of attributes which uniquely identify instances of an entity set
  - socialInsurance# may be a key for Person
  - firstName, middleName, lastName, address may be a key for Person

- In most cases, a key is formed by one or more attributes of the entity itself (*internal* keys)

- Sometimes, an entity doesn’t have a key among its attributes. This is called a *weak entity*. Solution: the keys of related entities brought in to help with identification (becoming *foreign keys*).

- A key for a relationship consists of the keys of the entities it relates
Examples of Keys in E/R

- **internal, single-attribute**
  - AUTOMOBILE
    - Registration
    - Model
    - Colour

- **internal, multi-attribute**
  - PERSON
    - DateOf Birth
    - Surname
    - FirstName
    - Address

- **foreign, multi-attribute**
  - STUDENT
    - Registration
    - Year
    - Surname

- **Weak entity**
  - ENROLMENT
    - (1,1)
  - UNIVERSITY
    - (1,N)
    - Name
    - City
    - Address
Weak entity sets – another example

- Football players: `Player(name, number)`
- Is `name` a key?
- Is `number` a key?
- However, `number` together with the team `name`, related to the player by “PlaysFor” should be unique

![Diagram showing Weak Entity set with supporting relationship and connecting entities]

**Weak Entity (double rectangle)**

**Supporting relationship (double diamond)**
General Observations about Keys

• A key may consist of one or more attributes, provided that each of the attributes has (1,1) cardinality

• A foreign key can involve one or more entities, provided that:
  – each of those entities is member of a relationship and
  – the entity to be identified participates in the relationship with cardinality equal to (1,1)

• A foreign key may involve an entity that has itself a foreign key, as long as cycles are not generated

• Each entity set must have at least one (internal or foreign) key
Schema with Keys
Subclasses in E/R

- **Subclass = special case** → Inheritance
  - Inherits all attributes and relationships of the superclass
  - May have other separate attributes or relationships of its own
  - One-to-one relationship (isA) between classes
Multiple inheritance in E/R

- Allowed, but not usually necessary
  - Idea: Entity can “be” many classes (union)
- Usually not a good idea
  - Naming collisions, semantic clashes
  - Queries often work just as well
    - \( \text{SELECT A.}* \text{ FROM } \text{Actors A, Directors D WHERE A.SID = D.SID} \)
- Usable classes usually form a tree
Challenge: modeling the “real world”

• Life is arbitrarily complex
  – Directors who are also actors? Actors who play multiple roles in one movie? Animal actors?

• **Design choices**: Should a concept be modeled as an entity, an attribute, or a relationship?

• **Limitations of the ER Model**: A lot of data semantics can be captured but some cannot

• **Key to successful model**: **parsimony**
  – As complex as necessary, but no more
  – Choose to represent only “relevant” things
EXAMPLE
We wish to create a database for a company that runs training courses. For this, we must store data about trainees and instructors. For each course participant (about 5,000 in all), identified by a code, we want to store her social security number, surname, age, sex, place of birth, employer’s name, address and telephone number, previous employers (and periods employed), the courses attended (there are about 200 courses) and the final assessment for each course. We need also to represent the seminars that each participant is attending at present and, for each day, the places and times the classes are held.

Each course has a code and a title and any course can be given any number of times. Each time a particular course is given, we will call it an ‘edition’ of the course. For each edition, we represent the start date, the end date, and the number of participants. If a trainee is self-employed, we need to know her area of expertise, and, if appropriate, her title. For somebody who works for a company, we store the level and position held. For each instructor (about 300), we will show the surname, age, place of birth, the edition of the course taught, those taught in the past and the courses that the tutor is qualified to teach. All the instructors’ telephone numbers are also stored. An instructor can be permanently employed by the training company or freelance.
From real world to E/R Model

We wish to create a database for a company that runs training courses. For this, we must store data about the trainees and the instructors. For each course participant (about 5,000), identified by a code, we want to store her social security number, surname, age, sex, place of birth, employer’s name, address and telephone number, previous employers (and periods employed), the courses attended (there are about 200 courses) and the final assessment for each course. We need also to represent the seminars that each participant is attending at present and, for each day, the places and times the classes are held.

Each course has a code and a title and any course can be given any number of times. Each time a particular course is given, we will call it an ‘edition’ of the course. For each edition, we represent the start date, the end date, and the number of participants. If a trainee is self-employed, we need to know her area of expertise, and, if appropriate, her title. For somebody who works for a company, we store the level and position held. For each instructor (about 300), we will show the surname, age, place of birth, the edition of the course taught, those taught in the past and the courses that the tutor is qualified to teach. All the instructors’ telephone numbers are also stored. An instructor can be permanently employed by the training company or freelance.
## Glossary

<table>
<thead>
<tr>
<th>Term</th>
<th>Description</th>
<th>Synonym</th>
<th>Links</th>
</tr>
</thead>
<tbody>
<tr>
<td>Trainee</td>
<td>Participant in a course. Can be an employee or self-employed.</td>
<td>Participant</td>
<td>Course, Company</td>
</tr>
<tr>
<td>Instructor</td>
<td>Course tutor. Can be freelance.</td>
<td>Tutor</td>
<td>Course</td>
</tr>
<tr>
<td>Course</td>
<td>Course offered. Can have various editions.</td>
<td>Seminar</td>
<td>Instructor, Trainee</td>
</tr>
<tr>
<td>Company</td>
<td>Company by which a trainee is employed or has been employed.</td>
<td></td>
<td>Trainee</td>
</tr>
</tbody>
</table>
We wish to create a database for a company that runs training courses. For this, we must store data about trainees and instructors. For each course participant (about 5,000), identified by a code, we want to store her social security number, surname, age, sex, place of birth, employer’s name, address and telephone number, previous employers (and periods employed), courses attended (there are about 200 courses) and the final assessment for each course. We need also to represent seminars that each participant is attending at present and, for each day, the places and times the classes are held.

Each course has a code and a title and any course can be given any number of times. Each time a particular course is given, we will call it an ‘edition’ of the course. For each edition, we represent the start date, the end date, and the number of participants. If a trainee is self-employed, we need to know her area of expertise, and, if appropriate, her title. For somebody who works for a company, we store the level and position held. For each instructor (about 300), we will show the surname, age, place of birth, the edition of the course taught, those taught in the past and the courses that the tutor is qualified to teach. All the instructors’ telephone numbers are also stored. An instructor can be permanently employed by the training company or freelance.
... the E/R model result
FROM E/R MODEL TO DATABASE SCHEMA
Designing a Database Schema

The “real world”

The E/R Model (Conceptual Model)

Part
(1,1)
supplies
Date
(1,N)
orders
(1,N)
Supplier
Customer

The Relational Schema

Part (Name, Description, Part#)
Supplier (Name, Addr)
Customer (Name, Addr)
Supplies (Name, Part#, Date)
Orders (Name, Part#)
(Relational) Database Design

• Given a conceptual schema (ER, but could also be UML), generate a logical (relational) schema
• Not just a simple translation between models
• It is helpful to divide the design into two steps:
  – *Restructuring of the ER schema*, based on criteria for the optimization of the schema
  – *Translation into the logical model*, based on the features of the logical model (in our case, the relational model)
1. RESTRUCTURING AN E/R MODEL
Restructuring Overview

**Input:** E/R Schema  
**Output:** Restructured E/R Schema

Restructuring includes:
- Avoiding redundancies
- Choosing entity set vs attribute
- Limiting the use of weak entity sets
- Selection of keys
- Creating extra entity sets to replace attributes with cardinality greater than one
Example: redundancy

- Part
- Part Number
- Name
- Manf Name
- Manf Address
Example: redundancy

Diagram:
- Part
  - Part Number
  - Name
- Made By
  - (1,1)
- Manufacturer
  - Address
  - Name
- Manf Name
Example: no redundancy
Avoiding Redundancy

• Redundancy: saying the same thing in two (or more) different ways

• Wastes space and (more importantly) encourages inconsistency
  – Two representations of the same fact become inconsistent if we change one and forget to change the other

• Usually indicates a design flaw as well
  – Example: storing actor’s address with movie relation
    => Address at time of filming? Now? Hotel near studio?
Two types of redundancy

• Repeated information

<table>
<thead>
<tr>
<th>name</th>
<th>address</th>
<th>role</th>
</tr>
</thead>
<tbody>
<tr>
<td>Keanu Reeves</td>
<td>Los Angeles, CA</td>
<td>Neo</td>
</tr>
<tr>
<td>Keanu Reeves</td>
<td>Los Angeles, CA</td>
<td>Jack Traven</td>
</tr>
<tr>
<td>Keanu Reeves</td>
<td>Los Angeles, CA</td>
<td>Johnny Utah</td>
</tr>
<tr>
<td>Jennifer Lawrence</td>
<td>Hollywood, CA</td>
<td>Katniss</td>
</tr>
</tbody>
</table>

• Repeated design (same or similar attributes)
Entity Sets Versus Attributes

• An entity set should satisfy at least one of the following conditions:
  – It is more than the name of something; it has at least one non-key attribute.
    or
  – It is the “many” in a many-one or many-many relationship.

• Rules of thumb
  – A “thing” in its own right => Entity Set
  – A “detail” about some other “thing” => Attribute
  – A “detail” correlated among many “things” => Entity Set

Really this is just about avoiding redundancy
**E.S. vs. attributes: examples**

Domain fact change: *A part can have more than one manufacturer ...*

![Diagram](image)
E.S. vs. attributes: examples

Part (1,N) Made By (1,N) Manufacturer

Part Name
Part Number

(No address attribute)

Part Name
Part Number
Manf Name
Name
E.S. vs. attributes: examples

- Student
  - Name
  - Student number
- Mentored by
  - (0,1)
- Mentor
  - Name
  - Mentor email
  - Mentor name
E.S. vs. attributes: examples

Domain fact change: A mentor can have more than one mentee ...

Any additional potential problems with this second design though?
If no student has any mentor, all information about mentors is lost.
When to use weak entity sets?

- The usual reason is that there is no global authority capable of creating unique ID’s
- **Example:** it is unlikely that there could be an agreement to assign unique student numbers across all students in the world
Don’t Overuse Weak Entity Sets

• Beginner database designers often doubt that anything could be a key by itself
  – They make all entity sets weak, supported by all other entity sets to which they are linked

• It is usually better to create unique IDs
  – Social insurance number, automobile VIN, etc.
  – Useful for many reasons (next slide)
Selecting a Primary Key

• Every relation must have a primary key
• The criteria for this decision are as follows:
  – Attributes with null values cannot form primary keys
  – One/few attributes is preferable to many attributes
  – Internal keys preferable to external (foreign) ones (weak entities depend for their existence on other entities)
Keeping keys simple

Multi-attribute and/or string keys...

• ... are redundant
  – e.g. Movies(title, year, ...): 2 attributes for key, ~16 bytes
  – Use movieID key instead (4 bytes)
  => Integer movieID key saves 75% space and a lot of typing

• ... break encapsulation
  – e.g. Patient(firstName, lastName, phone, ...)
  – Security/privacy hole
  => Integer patientID prevents information leaks

• ... are brittle (nasty interaction of above two points)
  – Name or phone number change? Parent and child with same name?
  – Patient with no phone? Two movies with same title and year?
  => Internal ID always exists, immutable, unique

Also: computers are really good at integers...
Attributes with cardinality > 1

• The relational model doesn’t allow multi-valued attributes. We must convert these to entity sets.
2. TRANSLATING AN E/R MODEL INTO A DB SCHEMA
Translation into a Logical Schema

**Input:** E/R Schema

**Output:** Relational Schema

- **Goal:** Starting from an E/R schema, construct an equivalent relational schema
  - “equivalent” = a schema capable of representing the same information

- **A good translation should also:**
  - not allow redundancy
  - not invite unnecessary null values
The general idea

• Each **entity set** becomes a relation.
  Its attributes are
  – the attributes of the entity set.

• Each **relationship** becomes a relation.
  Its attributes are
  – the keys of the entity sets that it connects, plus
  – the attributes of the relationship itself.

• Intuitive example:

```
Student(sID, Name, campus)
Course(clID, dept, cNum, name)
Took(sID, clID, grade)
```
Many-to-Many Binary Relationships

Employee(\textbf{Number}, Surname, Salary)
Project(\textbf{Code}, Name, Budget)
Participation(\textbf{Number}, \textbf{Code}, StartDate)
Many-to-Many Recursive Relationships

Product(\textcolor{red}{Code}, \text{Name}, \text{Cost})
Composition(\textcolor{red}{Part}, \textcolor{red}{SubPart}, \text{Quantity})
Ternary Many-to-Many Relationships

Supplier(\textit{SupplierID}, \textit{SupplierName})
Product(\textit{Code}, \textit{Type})
Department(\textit{Name}, Telephone)
Supply(\textit{Supplier}, \textit{Product}, \textit{Department}, \textit{Quantity})
One-to-Many Relationships
with mandatory participation for one

Player(Surname, DateOfBirth, Position)
Team(Name, Town, TeamColours)
Contract(PlayerSurname, PlayerDateOfBirth, Team, Salary)

OR:
Player(Surname, DateOfBirth, Position, TeamName, Salary) – since it’s a 1:1, only one TeamName and Salary for each Player, right?
Team(Name, Town, TeamColours)

(Ronaldo, 1985, “Manch.United”, 10)
(Ronaldo, 1985, “Real Madrid”, 20)
One-to-One Relationships
with mandatory participation for both

Head(Number, Name, Salary, Department, StartDate) – 1:1 on the Head side => exactly one department per head
Department(Name, Telephone, Branch)

*Or:*

Head(Number, Name, Salary, StartDate)
Department(Name, Telephone, HeadNumber, Branch) – 1:1 on the Department side => exactly one head per each department
One-to-One Relationships
with optional participation for one

Employee(Number, Name, Salary)
Department(Name, Telephone, Branch, Head, StartDate) – 1:1 on the department side => exactly one employee as department head

Or, if both entities are optional

Employee(Number, Name, Salary)
Department(Name, Telephone, Branch)
Management(HeadNumber, Department, StartDate) – doesn’t have to include all heads, or all departments for that matter (both optional)
Summary of Types of Relationship

- many-to-many (binary or ternary)
- one-to-many
  - mandatory: (1,1) on the “one” side
  - optional: (0,1) on the “one” side
- one-to-one
  - both mandatory: (1,1) on both sides
  - one mandatory, one optional: 
    (1,1) on one side and (0,1) on other side
  - both optional: (0,1) on both sides
# Summary of Transformation Rules

<table>
<thead>
<tr>
<th>Type</th>
<th>Initial schema</th>
<th>Possible translation</th>
</tr>
</thead>
</table>
| **Binary many-to-many relationship**      | ![Diagram](image) | $E_1(A_{E11}, A_{E12})$  
$E_2(A_{E21}, A_{E22})$  
$R(A_{E11}, A_{E21}, A_R)$ |
| **Ternary many-to-many relationship**     | ![Diagram](image) | $E_1(A_{E11}, A_{E12})$  
$E_2(A_{E21}, A_{E22})$  
$E_3(A_{E31}, A_{E32})$  
$R(A_{E11}, A_{E21}, A_{E31}, A_R)$ |
| **One-to-many relationship with mandatory participation** | ![Diagram](image) | $E_1(A_{E11}, A_{E12}, A_{E21}, A_R)$  
$E_2(A_{E21}, A_{E22})$ |
### More Rules...

<table>
<thead>
<tr>
<th>Type</th>
<th>Initial schema</th>
<th>Possible translation</th>
</tr>
</thead>
</table>
| One-to-many relationship with optional participation | ![Diagram](image) | $E_1(A_{E11}, A_{E12})$
$E_2(A_{E21}, A_{E22})$
$R(A_{E11}, A_{E21}, A_R)$
Alternatively:
$E_1(A_{E11}, A_{E12}, A_{E21}^*, A_{E21}^*, A_R^*)$
$E_2(A_{E21}, A_{E22})$ |
| Relationship with external identifiers          | ![Diagram](image) | $E_1(A_{E12}, A_{E21}, A_{E11}, A_R)$
$E_2(A_{E21}, A_{E22})$ |

* Indicates that nulls are allowed
...Even More Rules...

<table>
<thead>
<tr>
<th>Type</th>
<th>Initial schema</th>
<th>Possible translation</th>
</tr>
</thead>
</table>
| One-to-one relationship with mandatory participation for both entities | ![Diagram](Image) | $E_1(A_{E11}, A_{E12}, A_{E21}, A_R)$  
$E_2(A_{E21}, A_{E22})$  
Alternatively:  
$E_2(A_{E21}, A_{E22})$  
$E_1(A_{E11}, A_{E12})$ |
| One-to-one relationship with optional participation for one entity       | ![Diagram](Image) | $E_1(A_{E11}, A_{E12}, A_{E21}, A_R)$  
$E_2(A_{E21}, A_{E22})$ |

*Note: The diagrams and formulas are placeholders for actual graphics and expressions.*
...and the Last One...

<table>
<thead>
<tr>
<th>Type</th>
<th>Initial schema</th>
<th>Possible translation</th>
</tr>
</thead>
<tbody>
<tr>
<td>One-to-one relationship with optional participation for both entities</td>
<td><img src="image" alt="Diagram" /></td>
<td>$E_1(A_{E11}, A_{E12})$</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$E_2(A_{E21}, A_{E22}, A^{<em>}_{E11}, A^{</em>}_{R})$</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Alternatively:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$E_1(A_{E11}, A_{E12}, A^{<em>}_{E21}, A^{</em>}_{R})$</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$E_2(A_{E21}, A_{E22})$</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Alternatively:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$E_1(A_{E11}, A_{E12})$</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$E_2(A_{E21}, A_{E22})$</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$R(A_{E11}, A_{E21}, A_{R})$</td>
</tr>
</tbody>
</table>
Will the schema be “good”?

• If we use this process, will the schema we get be a good one?
• The process should ensure that there is no redundancy.
• But only with respect to what the E/R diagram represents.
• Crucial thing we are missing: functional dependencies. (We only have keys, not other FDs.)
• So we still need FD theory.
Redundancy can be *desirable*

- We’ve talked a lot about avoiding redundancy.
- It’s a *disadvantage*: because of larger storage requirements, (but, usually at negligible cost) and the necessity to carry out additional operations in order to keep the derived data consistent.
- But it’s also an *advantage*: a reduction in the number of accesses necessary to obtain derived information.
- **Tradeoff**: The decision to maintain or eliminate a redundancy is made by comparing the *cost of operations* that involve the redundant information and the *storage needed*, in the case of presence or absence of redundancy.

*Performance analysis is required to decide about redundancy.*
Administrative info

• Next week
  – Course wrap-up
  – Exam review
  – No lecture on Thursday (Tuesday is the last day of classes)

• Office hours as usual: Thursday 2-4pm, BA4268
  – Help with exam prep, anything you might still have unclear, don’t hesitate to drop by

• Don’t forget the course evals:
  – http://www.uoft.me/course-evals