Database Systems

Session 4 – Main Theme

Practical Relational Database Design

Dr. Jean-Claude Franchitti

New York University
Computer Science Department
Courant Institute of Mathematical Sciences

Agenda

1. Session Overview
2. ER and EER to Relational Mapping
3. Database Design Methodology and UML
4. Mapping Relational Design to ER/EER Case Study
5. Summary and Conclusion

Presentation material partially based on textbook slides
by Ramez Elmasri and Shamkant Navathe
Slides copyright © 2011
Session Agenda

- Session Overview
- ER and EER to Relational Mapping
- Database Design Methodology and UML
- Mapping Relational Design to ER/EER Case Study
- Summary & Conclusion

What is the class about?

- Course description and syllabus:
  - [http://www.nyu.edu/classes/jcf/CSCI-GA.2433-001](http://www.nyu.edu/classes/jcf/CSCI-GA.2433-001)

- Textbooks:
    - Ramez Elmasri and Shamkant Navathe
    - Addison Wesley
Icons / Metaphors

- Information
- Common Realization
- Knowledge/Competency Pattern
- Governance
- Alignment
- Solution Approach

Agenda

1. Session Overview
2. ER and EER to Relational Mapping
3. Database Design Methodology and UML
4. Mapping Relational Design to ER/EER Case Study
5. Summary and Conclusion
Sets, Relations, and Tables

- In this unit, we learn the semantics of specifying a relational database, later we will learn the syntax of SQL for doing this
- The basic “datatype”, or “variable” of a relational database is a relation
- In this unit, such a variable will be a set
- Later, we will extend this, and such a variable will be a multiset
- In SQL, such a variable is called a table
- We may use the term table for a relation in this unit too
We will not use axiomatic set theory

A *set* is a “bag” of elements, some/all of which could be sets themselves and a binary relationship “is element of” denoted by $\in$, such as $2 \in \{2, 5, 3, 7\}$, $\{2,8\} \in \{2, \{2, 8\}, 5, 3, 7\}$,

You cannot specify
- How many times an element appears in a set (if you could, this would be a *multiset*)
- In which position an element appears (if you could, this would be a *sequence*)

Therefore, as sets: $\{2, 5, 3, 7\} = \{2, 7, 5, 3, 5, 3, 3\}$

Note: in many places you will read: “an element can appear in a set only once”
This is not quite right. And it is important not to assume this, as we will see in the next unit

Two sets $A$ and $B$ are equal iff (if and only if) they have the same elements

In other words, for every $x$: $x$ is an element of $A$ iff (if and only if) $x$ is an element of $B$

“More mathematically,”
\[ \forall x \{ x \in A \iff x \in B \} \] if and only if $A = B$

Therefore, as sets: $\{2, 5, 3, 7\} = \{2, 7, 5, 3, 5, 3, 3\}$
This reiterates what we have said previously
Relation

- Consider a table, with a fixed number of columns where elements of each column are drawn from some specific domain.
- The columns are labeled and the labels are distinct.
- We will consider such a table to be *a set of rows* (another word for "row": *tuple*).
- Here is an example of a table $S$ of two columns $A$ and $B$.

<table>
<thead>
<tr>
<th>$S$</th>
<th>$A$</th>
<th>$B$</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>a</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>b</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>c</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>d</td>
<td>3</td>
<td></td>
</tr>
</tbody>
</table>

- A *relation* is such a table.
- We will also write $S(A,B)$ for table $S$ with columns $A$ and $B$.

Relational Schema

- What we saw was an *instance* (current value for a relation with the defined columns and domains).
- To specify this relation in general (not the specific instance) we need to talk about a *relational schema*.
- A relational schema defines a constant number of relations, one or more.
Here is an informal, but complete, description of what is a relational schema of one relation:

1. We give it a name (we had S).
2. We chose the number of columns (we had 2) and give them distinct names (we had A and B).
3. We decide on the domains of elements in the columns (we had letters for A and integers for B).
4. We decide on constraints, if any, on the permitted values (we had that any two rows that are equal on A must be equal on B).

Let’s verify:
- A: all lower case letters in English
- B: all positive integers less than 100
- S(A,B) satisfies the condition that any two tuples that are equal on A must also be equal on B.

Our example was an instance of this relational schema:

<table>
<thead>
<tr>
<th>S</th>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>a</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>b</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>c</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>d</td>
<td>3</td>
<td></td>
</tr>
</tbody>
</table>
- Since relations are sets of tuples, the following two relations are equal (are really one relation written in two different ways)
  (This is a different example, not an instance of the previous relational schema)

<table>
<thead>
<tr>
<th>S</th>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>a</td>
<td>56</td>
<td></td>
</tr>
<tr>
<td>b</td>
<td>2</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>S</th>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>56</td>
<td>a</td>
</tr>
<tr>
<td>a</td>
<td>2</td>
<td>a</td>
</tr>
<tr>
<td>b</td>
<td>2</td>
<td>a</td>
</tr>
<tr>
<td>a</td>
<td>56</td>
<td>a</td>
</tr>
</tbody>
</table>

- Since the positions in the tuple (1st, 2nd, etc.) are labeled with the column headings, the following two relations are equal (are really one relation written in two different ways)

<table>
<thead>
<tr>
<th>S</th>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>a</td>
<td>56</td>
<td></td>
</tr>
<tr>
<td>b</td>
<td>2</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>S</th>
<th>B</th>
<th>A</th>
</tr>
</thead>
<tbody>
<tr>
<td>56</td>
<td>a</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>a</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>b</td>
<td></td>
</tr>
<tr>
<td>56</td>
<td>a</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>a</td>
<td></td>
</tr>
<tr>
<td>56</td>
<td>a</td>
<td></td>
</tr>
</tbody>
</table>
To specify relations, it is enough to do what we have done above

As long as we understand what are the domains for the columns, the following are formally fully specified relations

- Relational (schema) \( P(\text{Name}, \text{SSN}, \text{DOB}, \text{Grade}) \) with some (not specified, but we should have done it) domains for attributes
- Relational (schema) \( Q(\text{Grade}, \text{Salary}) \) with some (not specified, but we should have done it) domains for attributes

<table>
<thead>
<tr>
<th>P</th>
<th>Name</th>
<th>SSN</th>
<th>DOB</th>
<th>Grade</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>121</td>
<td>2367</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>B</td>
<td>101</td>
<td>3498</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>C</td>
<td>106</td>
<td>2987</td>
<td>2</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Q</th>
<th>Grade</th>
<th>Salary</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>90</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>80</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>70</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>70</td>
<td></td>
</tr>
</tbody>
</table>

But we will do more. We will specify, as appropriate for the schema:

- **Primary keys**
- **Keys** (beyond primary)
- **Foreign keys** and what they reference (we will see soon what this means)
- Additional constraints

Some of the constraints involve more than one relation

The above most important structurally

Later, when we talk about SQL DDL, we will specify additional properties
Relations (5/8)

- Consider relation (schema) Person(FN, LN, Grade, YOB)
- Instance:

<table>
<thead>
<tr>
<th>Person</th>
<th>FN</th>
<th>LN</th>
<th>Grade</th>
<th>YOB</th>
</tr>
</thead>
<tbody>
<tr>
<td>John</td>
<td>Smith</td>
<td>8</td>
<td>1976</td>
<td></td>
</tr>
<tr>
<td>Lakshmi</td>
<td>Smith</td>
<td>9</td>
<td>1981</td>
<td></td>
</tr>
<tr>
<td>John</td>
<td>Smith</td>
<td>8</td>
<td>1976</td>
<td></td>
</tr>
<tr>
<td>John</td>
<td>Yao</td>
<td>9</td>
<td>1992</td>
<td></td>
</tr>
</tbody>
</table>

- We are told that any two tuples that are equal on both FN and LN are (completely) equal
  - We have some tuples appearing multiple times: this is just for clarifying that this permitted in the definition, we do not discuss here why we would have the same tuple more than one time (we will talk about this later)
- This is a property of every possible instance of Person in our application—we are told this
- Then (FN, LN) is a superkey of Person, and in fact a key, because neither FN nor LN by themselves are sufficient (we are told that too)

Relations (6/8)

- Consider relation (schema) Q(Grade, Salary)
- Example:

<table>
<thead>
<tr>
<th>Pay</th>
<th>Grade</th>
<th>Salary</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>128</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>139</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>147</td>
<td></td>
</tr>
</tbody>
</table>

- We are told that for any instance of Pay, any two tuples that are equal on Grade are (completely) equal
  - Of course, if each Grade appears in only one tuple, this is automatically true
- Then, similarly to before, Grade is a key
- What about Salary, is this a key also?
- No, because we are not told that any two tuples that are equal on Salary are equal on Grade in every instance of Pay
A set of columns in a relation is a **superkey** if and only any two tuples that are equal on the elements of these columns are (completely equal)

**A relation always has at least one superkey**

- The set of all the attributes is a superkey
- Because any two tuples that are equal on all attributes are completely equal
- A minimal superkey, is a key
- A relation always has at least one key (start with any superkey and remove unnecessary columns)
- There may be more than one key
- Exactly one key is chosen as **primary key**
- Other keys are just keys
- Sometimes they are called **candidate keys** (as they are candidates for the primary key, though not chosen)

We will underline the attributes of the chosen primary key

Returning to the City example:

\[ \text{City}(\text{Longitude}, \text{Latitude}, \text{Country}, \text{State}, \text{Name}, \text{Size}) \]

We can have

- \[ \text{City}(\text{Longitude}, \text{Latitude}, \text{Country}, \text{State}, \text{Name}, \text{Size}) \]
- This implies that Longitude, Latitude form a primary key
- We also have a candidate key: Country, State, Name

We can have

- \[ \text{City}(\text{Longitude}, \text{Latitude}, \text{Country}, \text{State}, \text{Name}, \text{Size}) \]
- This implies that Country, State, Name form a primary key
- We also have a candidate key: Longitude, Latitude
A relational database is basically a set of relations and is an instance of a relational schema.
COMPANY database example
- Assume that the mapping will create tables with simple single-valued attributes

Step 1: Mapping of Regular Entity Types
- For each regular entity type, create a relation \( R \) that includes all the simple attributes of \( E \)
- Called entity relations
  - Each tuple represents an entity instance
• Step 2: Mapping of Weak Entity Types
  • For each weak entity type, create a relation $R$ and include all simple attributes of the entity type as attributes of $R$
  • Include primary key attributes of owner as foreign key attributes of $R$
Step 3: Mapping of Binary 1:1 Relationship Types

- For each binary 1:1 relationship type
  - Identify relations that correspond to entity types participating in \( R \)

Possible approaches:
- Foreign key approach
- Merged relationship approach
- Crossreference or relationship relation approach

Step 4: Mapping of Binary 1:N Relationship Types

- For each regular binary 1:N relationship type
  - Identify relation that represents participating entity type at \( N \)-side of relationship type
  - Include primary key of other entity type as foreign key in \( S \)
  - Include simple attributes of 1:N relationship type as attributes of \( S \)
Alternative approach

- Use the relationship relation (cross-reference) option as in the third option for binary 1:1 relationships

Step 5: Mapping of Binary $M:N$ Relationship Types

- For each binary $M:N$ relationship type
  - Create a new relation $S$
  - Include primary key of participating entity types as foreign key attributes in $S$
  - Include any simple attributes of $M:N$ relationship type
Step 6: Mapping of Multivalued Attributes

- For each multivalued attribute
  - Create a new relation
  - Primary key of $R$ is the combination of $A$ and $K$
  - If the multivalued attribute is composite, include its simple components

Step 7: Mapping of $N$-ary Relationship Types

- For each $n$-ary relationship type $R$
  - Create a new relation $S$ to represent $R$
  - Include primary keys of participating entity types as foreign keys
  - Include any simple attributes as attributes
In a relational schema relationship, types are not represented explicitly
- Represented by having two attributes $A$ and $B$: one a primary key and the other a foreign key
Mapping EER Model Constructs to Relations

- Extending ER-to-relational mapping algorithm

Mapping of Specialization or Generalization (1/2)

- Step 8: Options for Mapping Specialization or Generalization (see textbook pages 294-295)
  - Option 8A: Multiple relations—superclass and subclasses
    - For any specialization (total or partial, disjoint or overlapping)
  - Option 8B: Multiple relations—subclass relations only
    - Subclasses are total
    - Specialization has disjointedness constraint
Mapping of Specialization or Generalization (2/2)

- **Option 8C: Single relation with one type attribute**
  - Type or discriminating attribute indicates subclass of tuple
  - Subclasses are disjoint
    - Potential for generating many NULL values if many specific attributes exist in the subclasses
- **Option 8D: Single relation with multiple type attributes**
  - Subclasses are overlapping
  - Will also work for a disjoint specialization

Mapping of Shared Subclasses (Multiple Inheritance)

- Apply any of the options discussed in step 8 to a shared subclass

![Diagram](image_url)

*Figure 9.6: Mapping the EER specialization lattice in Figure 8.8 using multiple options.*
Step 9: Mapping of Union Types (Categories)
- Defining superclasses have different keys
- Specify a new key attribute
  - Surrogate key

Sample Mapping of EER Categories to Relations
Summary

- Map conceptual schema design in the ER model to a relational database schema
  - Algorithm for ER-to-relational mapping
  - Illustrated by examples from the COMPANY database
- Include additional steps in the algorithm for mapping constructs from EER model into relational model

Agenda

1. Session Overview
2. ER and EER to Relational Mapping
3. Database Design Methodology and UML
4. Mapping Relational Design to ER/EER Case Study
5. Summary and Conclusion
Agenda

- The Role of Information Systems in Organizations
- The Database Design and Implementation Process
- Use of UML Diagrams as an Aid to Database Design Specification
- Rational Rose: A UML-Based Design Tool
- Automated Database Design Tools

Practical Database Design Methodology and Use of UML Diagrams

- Design methodology
  - Target database managed by some type of database management system
- Various design methodologies
- Large database
  - Several dozen gigabytes of data and a schema with more than 30 or 40 distinct entity types
Organizational context for using database systems
- Organizations have created the position of database administrator (DBA) and database administration departments
- Information technology (IT) and information resource management (IRM) departments
  - Key to successful business management

Database systems are integral components in computer-based information systems
- Personal computers and database system-like software products
  - Utilized by users who previously belonged to the category of casual and occasional database users
- Personal databases gaining popularity
- Databases are distributed over multiple computer systems
  - Better local control and faster local processing
The Role of Information Systems in Organizations (3/3)

- Data dictionary systems or information repositories
  - Mini DBMSs
  - Manage meta-data
- High-performance transaction processing systems require around-the-clock nonstop operation
  - Performance is critical

The Information System Life Cycle (1/4)

- Information system (IS)
  - Resources involved in collection, management, use, and dissemination of information resources of organization
The Information System Life Cycle (2/4)

- **Macro life cycle**
  - Feasibility analysis
  - Requirements collection and analysis
  - Design
  - Implementation
  - Validation and acceptance testing
  - Requirements collection and analysis

The Information System Life Cycle (3/4)

- The database application system life cycle: **micro life cycle**
  - System definition
  - Database design
  - Database implementation
  - Loading or data conversion
The Information System Life Cycle (4/4)

- Application conversion
- Testing and validation
- Operation
- Monitoring and maintenance

The Database Design and Implementation Process (1/4)

- Design logical and physical structure of one or more databases
  - Accommodate the information needs of the users in an organization for a defined set of applications
- Goals of database design
  - Very hard to accomplish and measure
  - Often begins with informal and incomplete requirements
Main phases of the overall database design and implementation process:

1. Requirements collection and analysis
2. Conceptual database design
3. Choice of a DBMS
4. Data model mapping (also called logical database design)
5. Physical database design
6. Database system implementation and tuning
The Database Design and Implementation Process (3/4)

- Parallel activities
  - Data content, structure, and constraints of the database
  - Design of database applications
- Data-driven versus process-driven design
- Feedback loops among phases and within phases are common

The Database Design and Implementation Process (4/4)

- Heart of the database design process
  - Conceptual database design (Phase 2)
  - Data model mapping (Phase 4)
  - Physical database design (Phase 5)
  - Database system implementation and tuning (Phase 6)
Phase 1: Requirements Collection and Analysis (1/2)

- Activities
  - Identify application areas and user groups
  - Study and analyze documentation
  - Study current operating environment
  - Collect written responses from users

Phase 1: Requirements Collection and Analysis (2/2)

- Requirements specification techniques
  - Oriented analysis (OOA)
  - Data flow diagrams (DFDs)
  - Refinement of application goals
  - Computer-aided
Phase 2: Conceptual Database Design (1/3)

- Phase 2a: Conceptual Schema Design
  - Important to use a conceptual high-level data model
  - Approaches to conceptual schema design
    - Centralized (or one shot) schema design approach
    - View integration approach

Phase 2: Conceptual Database Design (2/3)

- Strategies for schema design
  - Top-down strategy
  - Bottom-up strategy
  - Inside-out strategy
  - Mixed strategy

- Schema (view) integration
  - Identify correspondences/conflicts among schemas:
    - Naming conflicts, type conflicts, domain (value set) conflicts, conflicts among constraints
    - Modify views to conform to one another
    - Merge of views and restructure
Phase 2: Conceptual Database Design (3/3)

- Strategies for the view integration process
  - Binary ladder integration
  - N-ary integration
  - Binary balanced strategy
  - Mixed strategy

- Phase 2b: Transaction Design
  - In parallel with Phase 2a
  - Specify transactions at a conceptual level
  - Identify input/output and functional behavior
  - Notation for specifying processes

Phase 3: Choice of a DBMS

- Costs to consider
  - Software acquisition cost
  - Maintenance cost
  - Hardware acquisition cost
  - Database creation and conversion cost
  - Personnel cost
  - Training cost
  - Operating cost

- Consider DBMS portability among different types of hardware
Phase 4: Data Model Mapping (Logical Database Design)

- Create a conceptual schema and external schemas
  - In data model of selected DBMS
- Stages
  - System-independent mapping
  - Tailoring schemas to a specific DBMS

Phase 5: Physical Database Design

- Choose specific file storage structures and access paths for the database files
  - Achieve good performance
- Criteria used to guide choice of physical database design options:
  - Response time
  - Space utilization
  - Transaction throughput
Phase 6: Database System Implementation and Tuning

- Typically responsibility of the DBA
  - Compose DDL
  - Load database
  - Convert data from earlier systems
- Database programs implemented by application programmers
- Most systems include monitoring utility to collect performance statistics

Use of UML Diagrams as an Aid to Database Design Specification

- Use UML as a design specification standard
- Unified Modeling Language (UML) approach
  - Combines commonly accepted concepts from many object-oriented (O-O) methods and methodologies
  - Includes use case diagrams, sequence diagrams, and statechart diagrams
**UML for Database Application Design**

- Advantages of UML
  - Resulting models can be used to design relational, object-oriented, or object-relational databases
  - Brings traditional database modelers, analysts, and designers together with software application developers

---

**Different Types of Diagrams in UML (1/4)**

- Structural diagrams
  - Class diagrams and package diagrams
  - Object diagrams
  - Component diagrams
  - Deployment diagrams
Different Types of Diagrams in UML (2/4)

- Behavioral diagrams
  - Use case diagrams
  - Sequence diagrams
  - Collaboration diagrams
  - Statechart diagrams
  - Activity diagrams

Use Case Diagram Notation

Figure 10.7
The use case diagram notation.
Different Types of Diagrams in UML (3/4)

![Sequence Diagram Notation](image)

**Figure 10.9**
The sequence diagram notation.

Different Types of Diagrams in UML (4/4)

![Statechart Diagram Notation](image)

**Figure 10.10**
The statechart diagram notation.

State consists of three parts:
- Name
- Activities
- Embedded machine
Activities and embedded machine are optional
Modeling and Design Example: UNIVERSITY Database

Figure 10.11
A sample statechart diagram for the UNIVERSITY database.

Sample Sequence Diagram

Figure 10.12
A sequence diagram for the UNIVERSITY database.
Rational Rose: A UML-Based Design Tool

- Rational Rose for database design
  - Modeling tool used in the industry to develop information systems
- Rational Rose data modeler
  - Visual modeling tool for designing databases
  - Provides capability to:
    - **Forward engineer** a database
    - **Reverse engineer** an existing implemented database into conceptual design
Data Modeling Using Rational Rose Data Modeler (1/4)

- Reverse engineering
  - Allows the user to create a conceptual data model based on an existing database schema specified in a DDL file
- Forward engineering and DDL generation
  - Create a data model directly from scratch in Rose
  - Generate DDL for a specific DBMS

Data Modeling Using Rational Rose Data Modeler (2/4)

- Conceptual design in UML notation
  - Build ER diagrams using class diagrams in Rational Rose
- Identifying relationships
  - Object in a child class cannot exist without a corresponding parent object
- Non-identifying relationships
  - Specify a regular association (relationship) between two independent classes
Data Modeling Using Rational Rose Data Modeler (3/4)

- Converting logical data model to object model and vice versa
  - Logical data model can be converted to an object model
  - Allows a deep understanding of relationships between conceptual and implementation models

Data Modeling Using Rational Rose Data Modeler (4/4)

- Synchronization between the conceptual design and the actual database
- Extensive domain support
  - Create a standard set of user-defined data types
- Easy communication among design teams
  - Application developer can access both the object and data models
Automated Database Design Tools (1/3)

- Many CASE (computer-aided software engineering) tools for database design
- Combination of the following facilities
  - Diagramming
  - Model mapping
  - Design normalization

Automated Database Design Tools (2/3)

- Characteristics that a good design tool should possess:
  - Easy-to-use interface
  - Analytical components
  - Heuristic components
  - Trade-off analysis
  - Display of design results
  - Design verification
Variety of products available
Some use expert system technology

<table>
<thead>
<tr>
<th>Company/Technologies</th>
<th>Tool/Features</th>
<th>Functionality</th>
</tr>
</thead>
<tbody>
<tr>
<td>Embarcadero Technologies</td>
<td>ER/Studio, DBArtisan</td>
<td>Database modeling, application development</td>
</tr>
<tr>
<td>Oracle</td>
<td>Developer 2000 and Designer 2000</td>
<td>Database modeling, application development</td>
</tr>
<tr>
<td>Persistence Inc.</td>
<td>PowerTier</td>
<td>Mapping from O-O to relational model</td>
</tr>
<tr>
<td>Platinum Technology (Computer Associates)</td>
<td>Platinum ModelBuilder, Effort, BPWin, AllFusion Component Modeler</td>
<td>Data, process, and business component modeling</td>
</tr>
<tr>
<td>Popkin Software</td>
<td>Tellogic System Architect</td>
<td>Data modeling, object modeling, process modeling, structured analysis/design</td>
</tr>
<tr>
<td>Rational (IBM)</td>
<td>Rational Rose, XDE Developer Plus</td>
<td>Modeling in UML and application generation in C++ and Java</td>
</tr>
<tr>
<td>Resolution Ltd.</td>
<td>XCase</td>
<td>Conceptual modeling up to code maintenance</td>
</tr>
<tr>
<td>Sybase</td>
<td>Enterprise Application Suite</td>
<td>Data modeling, business logic modeling</td>
</tr>
<tr>
<td>Visio</td>
<td>Visio Enterprise</td>
<td>Data modeling, design and reengineering, Visual Basic and Visual C++</td>
</tr>
</tbody>
</table>

Summary

Six phases of the design process
- Commonly include conceptual design, logical design (data model mapping), physical design

UML diagrams
- Aid specification of database models and design

Rational Rose and the Rose Data Modeler
- Provide support for the conceptual design and logical design phases of database design
A Case Study

- Implementing an ER diagram as a relational schema (relational database)
- General implementation of strong entities
- Handling attributes of different types
- General implementation of relationships
- Possible special implementation of binary many-to-one relationships
- Implementation of ISA
- Implementation of weak entities Foreign keys
- Primary key / foreign key constraints inducing many-to-one relationships between tables
- Concept of referential integrity
- Crow’s feet notation: ends of lines
- Crow’s feet notation: pattern of lines
We are now ready to convert ER diagrams into relational databases

**Generally, but not always**
- An entity set is converted into a table
- A relationship is converted into a table

We will first go through a simple example
Then, we will go through our large example, studied previously
Then, we look at some additional points of interest
Finally, we summarize the process, so we are sure we understand it
More About The Example

- The given ER diagram is clear, other than
  - Discovered, which is the continent in which a particular species was first discovered
- Each child is a “dependant” of only one employee in our database
  - If both parents are employees, the child is “assigned” to one of them
- We are given additional information about the application
  - Values of attributes in a primary key must not be missing (this is a general rule, not only for this example)
  - Other than attributes in a primary key, other attributes unless stated otherwise may be missing
  - The value of Name is known for every Employee
- To build up our intuition, let’s look at some specific instance of our application

Country

- There are four countries, listing for them: Cname, Population (the latter only when known):
  - US
  - IN, 1150
  - CN, 1330
  - RU
- We create a table for Country “in the most obvious way,” by creating a column for each attribute (underlying the attributes of the primary key) and this works:

<table>
<thead>
<tr>
<th>Country</th>
<th>Cname</th>
<th>Population</th>
</tr>
</thead>
<tbody>
<tr>
<td>US</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IN</td>
<td></td>
<td>1150</td>
</tr>
<tr>
<td>CN</td>
<td></td>
<td>1330</td>
</tr>
<tr>
<td>RU</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
There are five animals, listing for them: Species, Discovered (note, that even though not required, Discovered happens to be known for every Species):

- Horse, Asia
- Wolf, Asia
- Cat, Africa
- Yak, Asia
- Zebra, Africa

We create a table for Animal as before, and this works:

<table>
<thead>
<tr>
<th>Animal</th>
<th>Species</th>
<th>Discovered</th>
</tr>
</thead>
<tbody>
<tr>
<td>Horse</td>
<td>Asia</td>
<td></td>
</tr>
<tr>
<td>Wolf</td>
<td>Asia</td>
<td></td>
</tr>
<tr>
<td>Cat</td>
<td>Africa</td>
<td></td>
</tr>
<tr>
<td>Yak</td>
<td>Asia</td>
<td></td>
</tr>
<tr>
<td>Zebra</td>
<td>Africa</td>
<td></td>
</tr>
</tbody>
</table>

There are five employees, listing for them: ID#, Name, (name of) Child (note there may be any number of Child values for an Employee, zero or more):

- 1, Alice, Erica, Frank
- 2, Bob, Bob, Frank
- 4, Carol
- 5, David
- 6, Bob, Frank

We create a table for Employee in the most obvious way, and this does not work:

<table>
<thead>
<tr>
<th>Employee</th>
<th>ID#</th>
<th>Name</th>
<th>Child</th>
<th>Child</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Alice</td>
<td>Erica</td>
<td>Frank</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>Bob</td>
<td>Bob</td>
<td>Frank</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>Carol</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>David</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>Bob</td>
<td></td>
<td>Frank</td>
<td></td>
</tr>
</tbody>
</table>
Employee

- Child is a multivalued attribute so, the number of columns labeled “Child” is, in principle, unbounded
- A table *must have* a fixed number of columns
  - It must be an instance in/of a relational schema
- If we are ready to store up to 25 children for an employee and create a table with 25 columns for children, perhaps tomorrow we get an employee with 26 children, who will not “fit”
- We *replace our attempted single table* for Employee *by two tables*
  - One for all the attributes of Employee other than the multivalued one (Child)
  - One for pairs of the form (primary key of Employee, Child)
- Note that both tables have a fixed number of columns, no matter how many children an employee has

Employee And Child

- Replace (incorrect)

<table>
<thead>
<tr>
<th>Employee</th>
<th>ID#</th>
<th>Name</th>
<th>Child</th>
<th>Child</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Alice</td>
<td>Erica</td>
<td>Frank</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>Bob</td>
<td>Bob</td>
<td>Frank</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>Carol</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>David</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>Bob</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Employee</th>
<th>ID#</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Alice</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>Bob</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>Carol</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>David</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>Bob</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Child</th>
<th>ID#</th>
<th>Child</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Erica</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>Frank</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>Bob</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>Frank</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>Frank</td>
<td></td>
</tr>
</tbody>
</table>
Employee And Child

- The primary key of the table Employee is ID#
- The primary key of the table Child is the pair: ID#,Child
- One attribute is not sufficient to get a primary key for Child
- It is clear from the example how to handle any number of multivalued attributes an entity has
  » Create a “main” table with all the attributes other than multivalued
    Its primary key is the original primary key of the entity set
  » Create a table for each multivalued attribute consisting a primary key for the main table and that multivalued attribute
    Its primary key is the primary key of the entity combined with the multivalued attribute

Foreign Key

- Let us return to our example
- Note that any value of ID# that appears in Child must also appear in Employee
  » Because a child must be a dependant of an existing employee
- This is an instance of a foreign key
- ID# in Child is a foreign key referencing Employee
  » This means that ID# appearing in Child must appear in some row "under" columns (here only one) of primary key in Employee
  » Note that ID# is not a key of Child, so a foreign key in a table does not have to be a key of that table

<table>
<thead>
<tr>
<th>Employee</th>
<th>ID#</th>
<th>Name</th>
<th>Child</th>
<th>ID#</th>
<th>Child</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td>Alice</td>
<td>1</td>
<td></td>
<td>Erica</td>
</tr>
<tr>
<td>2</td>
<td></td>
<td>Bob</td>
<td>1</td>
<td></td>
<td>Frank</td>
</tr>
<tr>
<td>4</td>
<td></td>
<td>Carol</td>
<td>2</td>
<td></td>
<td>Bob</td>
</tr>
<tr>
<td>5</td>
<td></td>
<td>David</td>
<td>2</td>
<td></td>
<td>Frank</td>
</tr>
<tr>
<td>6</td>
<td></td>
<td>Bob</td>
<td>6</td>
<td></td>
<td>Frank</td>
</tr>
</tbody>
</table>
Foreign Key Induces A Many-To-One Relationship Between Tables

- **Note:**
  - Every row of Child has a single value of a primary key of Employee, so every row of Child “maps” to a single row of Employee.
  - Every row of Employee has zero or more rows of Child mapped into it.
    - In other words, no constraint.

Likes (1/3)

- Likes needs to specify which employees like which animals.
- Such specification can be done using the primary keys of the entities.
  - We do not need other attributes such as Name or Discovered.
- The table for likes contains some tuples:
  - 1 likes Horse
  - 1 likes Cat
  - 2 likes Cat
  - 6 likes Yak

<table>
<thead>
<tr>
<th>Likes</th>
<th>ID#</th>
<th>Species</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Horse</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>Cat</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>Cat</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>Yak</td>
<td></td>
</tr>
</tbody>
</table>
We could phrase the above somewhat differently
Likes needs to specify which employees (as rows in table Employee) like which animals (as rows in table Animal)
Such a specification can done using the primary keys of the tables Employee and Animal
The table for Likes contains some tuples:
- 1 likes Horse
- 1 likes Cat
- 2 likes Cat
- 6 likes Yak

Note that there are foreign key constraints
- ID# appearing in Likes is a foreign key referencing Employee
- Species appearing in Likes is a foreign key referencing Animal
And two many-to-one mappings are induced

This is true whenever we build a table for a relationship
- Likes was a relationship in the ER diagram
Born needs to specify which employees were born in which countries (for whom this information is known).

Such specification can be done using the primary keys of the entities/tables.

The relation Born contains some tuples:

- 1, US
- 2, IN
- 5, IN
- 6, CN

Note that there are foreign key constraints:

- ID# appearing in Born is a foreign key referencing Employee.
- Cname appearing in Born is a foreign key referencing Country.

And two many-to-one mappings are induced:

- One of them happens to be one-to-one as an employee can be born in only one country.
- This follows from the fact that in the ER diagram Born was a many-to-one relationship.
- Compare with Likes, where an employee can like more than one animal and an animal can be liked by more than one employee.
Let us focus on ID# in Employee
No two different tuples in Born can have the same ID#
Therefore ID# serves as a primary key, and we do not need Cname as part of the primary key

Using Visio
Visio can be used to designing/specifying relational databases
You can look at a tutorial, to get familiar with the mechanics of Visio
This is greatly oversimplified, but a good start
http://www.youtube.com/watch?v=1BYt3wmkgXE but foreign keys are not explained
http://www.youtube.com/watch?v=55TpWp4TmMw&NR=1
http://www.youtube.com/watch?v=r0x8ZMyPoj4&NR=1 but this third part
• Is misleading in the context of relational databases, due to the handling of many-to-many relationships and
• The use of the second page, all the pages in a single Visio drawing refer to a single ER diagram, so each ER diagram needs its own Visio drawing/file
Specifying A Relational Implementation

- It is possible to use Visio or ErWin to specify our relational implementation
  - Visio has an “enterprise” version to generate database specifications from the diagram to SQL DDL
- We will just focus on the first task
- The second can be done automatically so we do not need to look at it here

Specifying A Relational Implementation (more on Visio)

- A drawing in Visio is not an Entity Relationship Diagram tool despite such terminology in Visio
  - In fact, this is good, as it produces a relational schema, which is what we actually need, but this is a lower-level construct than ER diagrams
- It focuses on tables and the implicit many-to-one binary relationships induced by foreign key constraints
- Table
  - A rectangle with three vertical subrectangles: name, list of attributes in the primary key, list of attributes not in the primary key
  - Required attributes are in bold
  - Attributes in the primary key and foreign keys are labeled as such
- Relationship
  - A many-to-one binary (or perhaps one-to-one, which is a special case) relationship induced by a foreign key constraint is explicitly drawn by means of a segment with an arrow head
  - We will have alternative notations later
Relational Implementation For The Example

<table>
<thead>
<tr>
<th>Country</th>
<th>CName</th>
<th>Population</th>
<th>ID#</th>
<th>CName</th>
</tr>
</thead>
<tbody>
<tr>
<td>US</td>
<td>US</td>
<td>1150</td>
<td>1</td>
<td>US</td>
</tr>
<tr>
<td>IN</td>
<td>IN</td>
<td>1330</td>
<td>2</td>
<td>IN</td>
</tr>
<tr>
<td>CN</td>
<td>CN</td>
<td>1330</td>
<td>5</td>
<td>CN</td>
</tr>
<tr>
<td>RU</td>
<td>RU</td>
<td>1234</td>
<td>6</td>
<td>RU</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Child</th>
<th>ID#</th>
<th>Child</th>
<th>Employee</th>
<th>ID#</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>Erica</td>
<td>1</td>
<td>Alice</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>Frank</td>
<td>2</td>
<td>Bob</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>Bob</td>
<td>4</td>
<td>Carol</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>5</td>
<td>Frank</td>
<td>5</td>
<td>David</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>6</td>
<td>Frank</td>
<td>6</td>
<td>Bob</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Animal</th>
<th>Species</th>
<th>Discovered</th>
<th>ID#</th>
<th>Species</th>
</tr>
</thead>
<tbody>
<tr>
<td>Horse</td>
<td>Asia</td>
<td>1</td>
<td>Horse</td>
<td></td>
</tr>
<tr>
<td>Wolf</td>
<td>Asia</td>
<td>1</td>
<td>Cat</td>
<td></td>
</tr>
<tr>
<td>Cat</td>
<td>Africa</td>
<td>2</td>
<td>Cat</td>
<td></td>
</tr>
<tr>
<td>Yak</td>
<td>Asia</td>
<td>6</td>
<td>Yak</td>
<td></td>
</tr>
<tr>
<td>Zebra</td>
<td>Africa</td>
<td>6</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Cardinality Constraints

- The statement that a relationship is many-to-one as opposed to be a "standard" many-to-many relationship is really a cardinality constraint.
- We will look at a relationships Likes between Person and Country and four cases of cardinality constraints on how many Countries a Person may like:
  - No constraint
  - At least one
  - At most one
  - Exactly one
- For the first two, Likes is many-to-many
- For the last two, Likes is many-to-one
- Intuitively, Likes is many to one if for every Person, when you see which Countries this Person Likes, you get 0 or 1
- If you always get 1, this is a total function, otherwise this is a partial function.
Specifying These Constraints (Revisited)

Arrow Notation Cannot Distinguish Some Cases
**Crow’s Feet: Improved Arrow Notation**

- **Note: different sides of the relationship are labeled in the two notations!**

![Diagram showing different cardinalities in Crow’s Feet notation](image)

**Crow’s Feet**

- In general, cardinalities of both sides of the relationship may need to be specified
- We did only one, because it is sufficient to understand the notation
- We now return to the relational implementation of our example
- Visio and ErWin can use the Crow’s Feet notation
Relational Implementation For The Example

<table>
<thead>
<tr>
<th>Country</th>
<th>Cname</th>
<th>Population</th>
</tr>
</thead>
<tbody>
<tr>
<td>US</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IN</td>
<td></td>
<td>1150</td>
</tr>
<tr>
<td>CN</td>
<td></td>
<td>1330</td>
</tr>
<tr>
<td>RU</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Born</th>
<th>ID#</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>5</td>
</tr>
<tr>
<td></td>
<td>6</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Child</th>
<th>ID#</th>
<th>Child</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Erica</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>Bob</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>Frank</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>Frank</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Employee</th>
<th>ID#</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Alice</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>Bob</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>Carol</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>David</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>Bob</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Animal</th>
<th>Species</th>
<th>Discovered</th>
</tr>
</thead>
<tbody>
<tr>
<td>Horse</td>
<td>Asia</td>
<td></td>
</tr>
<tr>
<td>Wolf</td>
<td>Asia</td>
<td></td>
</tr>
<tr>
<td>Cat</td>
<td>Africa</td>
<td></td>
</tr>
<tr>
<td>Yak</td>
<td>Asia</td>
<td></td>
</tr>
<tr>
<td>Zebra</td>
<td>Africa</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Likes</th>
<th>ID#</th>
<th>Species</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Horse</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>Cat</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>Cat</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>Yak</td>
<td></td>
</tr>
</tbody>
</table>

Ends Of Lines (1/2)

- For every Country: 0..* of Born
  - 0 or more Employees were born there
- For every Born: 1..1 of Country
  - Because each row in Born has exactly 1 value of Country
- For every Employee: 0..1 of Born
  - Because an Employee was born in at most 1 Country
- For every Born: 1..1 of Employee
  - Because each row in Born has exactly 1 value of Employee
- For every Employee: 0..* Child
  - 0 or more of Child for an Employee
- For every Child: 1..1 Employee
  - Because every Child is assigned to exactly one Employee
### Ends Of Lines (2/2)

- For every Employee: 0..* of Likes
  - Employee can Like 0 or more Species
- For every Likes: 1..1 of Employee
  - Because each row in Likes has exactly 1 value of Employee
- For every Animal: 0..1 of Likes
  - Because a Species can be Liked by 0 or more Employees
- For every Likes: 1..1 of Species
  - Because each row in Likes has exactly 1 value of Species

### Born Versus Likes (1/2)

<table>
<thead>
<tr>
<th>Country</th>
<th>Cname</th>
<th>Population</th>
</tr>
</thead>
<tbody>
<tr>
<td>US</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IN</td>
<td>1150</td>
<td></td>
</tr>
<tr>
<td>CN</td>
<td>1330</td>
<td></td>
</tr>
<tr>
<td>RU</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Child</th>
<th>ID#</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Erica</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>Frank</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>Bob</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>Frank</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>Frank</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Employee</th>
<th>ID#</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Alice</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>Bob</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>Carol</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>David</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>Bob</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Animal</th>
<th>Species</th>
<th>Discovered</th>
</tr>
</thead>
<tbody>
<tr>
<td>Horse</td>
<td>Asia</td>
<td>Asia</td>
</tr>
<tr>
<td>Wolf</td>
<td>Asia</td>
<td>Asia</td>
</tr>
<tr>
<td>Cat</td>
<td>Africa</td>
<td>Africa</td>
</tr>
<tr>
<td>Yak</td>
<td>Asia</td>
<td>Asia</td>
</tr>
<tr>
<td>Zebra</td>
<td>Africa</td>
<td>Africa</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Likes</th>
<th>ID#</th>
<th>Species</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Horse</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>Cat</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>Cat</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>Yak</td>
<td></td>
</tr>
</tbody>
</table>
Born Versus Likes (2/2)

- Note that the many-to-one relationships are not of the same type in both cases.
- The relationship between Likes and Employee indicates that when you start from a row of Employee you end up in between 0 and unbounded number of rows of Likes: no restriction.
  An employee can like any number of animals.
- The relationship between Born and Employee indicates that when you start from a row of Employee you end up in between 0 and 1 rows of Born.
  An employee can be born in at most one country and therefore from a row of Employee you end up in between 0 and 1 rows of Born: a restriction.

  *Born is really a (partial) one-to-one relationship*

Such relationships are considered “strange.”

---

Treating Born Differently From Likes

- The above discussion implies that for every row in Employee there is at most one “relevant” row of Born.
- Therefore, the “extra” information about an employee that is currently stored in Born can be added to Employee.
- Born can be removed from the design.
- This sounds very formal, but intuitively very clear as we can see from an alternative design.
Alternative For Born

- Replace

<table>
<thead>
<tr>
<th>Employee</th>
<th>ID#</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>Alice</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>Bob</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>Carol</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>David</td>
</tr>
<tr>
<td>6</td>
<td>6</td>
<td>Bob</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Born</th>
<th>ID#</th>
<th>Cname</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>US</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>IN</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>IN</td>
</tr>
<tr>
<td>6</td>
<td>6</td>
<td>CN</td>
</tr>
</tbody>
</table>

by

<table>
<thead>
<tr>
<th>Employee</th>
<th>ID#</th>
<th>Name</th>
<th>Cname</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>Alice</td>
<td>US</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>Bob</td>
<td>IN</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>Carol</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>David</td>
<td>IN</td>
</tr>
<tr>
<td>6</td>
<td>6</td>
<td>Bob</td>
<td>CN</td>
</tr>
</tbody>
</table>

Alternative Relational Implementation For The Example

<table>
<thead>
<tr>
<th>Country</th>
<th>CName</th>
<th>Population</th>
</tr>
</thead>
<tbody>
<tr>
<td>US</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IN</td>
<td>1150</td>
<td></td>
</tr>
<tr>
<td>CN</td>
<td>1330</td>
<td></td>
</tr>
<tr>
<td>RU</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Child</th>
<th>ID#</th>
<th>Child</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>Erica</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>Frank</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>Bob</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>Frank</td>
</tr>
<tr>
<td>6</td>
<td>4</td>
<td>Frank</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Employee</th>
<th>ID#</th>
<th>Name</th>
<th>Cname</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>Alice</td>
<td>US</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>Bob</td>
<td>IN</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>Carol</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>David</td>
<td>IN</td>
</tr>
<tr>
<td>6</td>
<td>6</td>
<td>Bob</td>
<td>CN</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Likes</th>
<th>ID#</th>
<th>Species</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>Horse</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>Cat</td>
</tr>
<tr>
<td>2</td>
<td>6</td>
<td>Yak</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Animal</th>
<th>Species</th>
<th>Discovered</th>
</tr>
</thead>
<tbody>
<tr>
<td>Horse</td>
<td>Asia</td>
<td></td>
</tr>
<tr>
<td>Wolf</td>
<td>Asia</td>
<td></td>
</tr>
<tr>
<td>Cat</td>
<td>Africa</td>
<td></td>
</tr>
<tr>
<td>Yak</td>
<td>Asia</td>
<td></td>
</tr>
<tr>
<td>Zebra</td>
<td>Africa</td>
<td></td>
</tr>
</tbody>
</table>
Alternative Relational Implementation For The Example

<table>
<thead>
<tr>
<th>Animal</th>
<th>Species</th>
<th>Discovered</th>
</tr>
</thead>
<tbody>
<tr>
<td>Horse</td>
<td>Asia</td>
<td></td>
</tr>
<tr>
<td>Wolf</td>
<td>Asia</td>
<td></td>
</tr>
<tr>
<td>Cat</td>
<td>Africa</td>
<td></td>
</tr>
<tr>
<td>Yak</td>
<td>Asia</td>
<td></td>
</tr>
<tr>
<td>Zebra</td>
<td>Africa</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Likes</th>
<th>ID#</th>
<th>Species</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Horse</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>Cat</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>Cat</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>Yak</td>
<td></td>
</tr>
</tbody>
</table>

Pattern Of Lines (1/2)

- The line between Animal and Likes is **solid** because the primary key of the “many side”, Likes, includes the primary key of the “one side”, Animal, so it “cannot exist” without it.
- The line between Employee and Likes is **solid** because the primary key of the “many side”, Likes, includes the primary key of the “one side”, Employee, so it “cannot exist” without it.
- The line between Employee and Child is **solid** because the primary key of the “many side”, Child, includes the primary key of the “one side”, Employee, so it “cannot exist” without it.
- The line between Country and Employee is **dashed** because the primary key of the “many side”, Employee, does not include the primary key of the “one side”, Country, so it “can exist” without it.
This is not a question of the ends of lines “forcing” the pattern of lines

In the next slide, we see a slight modification of our example in which all lines have the same pair of endings

We required that for each Employee the Country of Birth is known

Nevertheless, as Cname is not part of the primary key of Country, the line is dashed

For technical reasons, the tables have slightly different names, but this has nothing to do with our point

Example
Which Implementation To Use For Born?

- We cannot give a general rule
- The first implementation uses more tables
- The second implementation may introduce NULLs (empty values), which we do not like
- For the purpose of the class we will always use the second version, to have better exercises
- So do this for all the homeworks and tests, when relevant

To Remember!

- Structurally, a relational database consists of
  1. A set of tables
  2. A set of many-to-one binary relationships between them, induced by foreign key constraints
     In other words; a set of functions (in general partial), each from a table into a table
- When designing a relational database, you must specify both (or you will produce a bad specification)
  » Technically, tables are enough, but this a very bad practice as you do not specify the relationships between tables
### Very Bad Diagram

- Tables are listed with attributes, specifying only which are in the primary key
  - Even the primary keys are not strictly required
- Foreign key constraints are not specified
  - So the DB system does not know what to enforce

```
<table>
<thead>
<tr>
<th>PK</th>
<th>Name</th>
<th>Population</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>PK</th>
<th>Country</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Name</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>PK</th>
<th>Name</th>
<th>CName</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>PK</th>
<th>IDF</th>
</tr>
</thead>
</table>
```

### Terrible Diagram

- Even primary keys are not specified

```
<table>
<thead>
<tr>
<th>PK</th>
<th>Name</th>
<th>CName</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>PK</th>
<th>Country</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>IDF</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>PK</th>
<th>IDF</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>PK</th>
<th>Species</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Student</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>PK</th>
<th>Species</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Employee</td>
</tr>
</tbody>
</table>
```

We now convert our big ER diagram into a relational database

- We specify
  - Attributes that must not be NULL
  - Primary keys
  - Keys (beyond primary)
  - Foreign keys and what they reference
  - Cardinality constraints
  - Some additional "stubs"

- We both give a narrative description, similar to actual SQL DDL (so we are learning about actual relational databases) and Visio/Erwin diagrams

- We should specify domains also, but we would not learn anything from this here, so we do not do it

- *We go bottom up, in the same order as the one we used in constructing the ER diagram*
Define Table Horse (Name NOT NULL, Primary Key (Name));

This represents the simplest possible relational database
  » One table with one attribute

<table>
<thead>
<tr>
<th>Horse</th>
<th>PK</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Person has some interesting attributes
- Multivalued attribute: we will create another table
- Derived attribute: we do not create a column for it, it will be computed as needed
- Composite attribute: we “flatten” it

Define Table Person (ID# NOT NULL, SS# NOT NULL, FN, LN NOT NULL, DOB NOT NULL, Primary Key (ID#), Candidate Key (SS#), Age (computed by procedure ...));

In SQL DDL, the keyword UNIQUE is used instead of Candidate Key, but “Candidate Key” is better for reminding us what this could be
- Age would likely not be stored but defined in some view
Define Table Child (ID# NOT NULL, ChildName NOT NULL, Primary Key (ID#, ChildName), Foreign Key (ID#) References Person);

This lists all pairs (ID# of person, a child’s name)

» We have chosen a more descriptive attribute name than the one in the ER diagram for children’s names

Note

» A person may have several children, each with a different name
» Two different persons may have children with the same name

Because of this, no single attribute can serve as primary key of Child
Person And Child (1/2)

- Note that some attributes are not bold, such as FN here
- This means that FN could be NULL (in this context, meaning empty)
- Note the induced many-to-one relationship
- We need to make sure we understand what the line ends indicate
  - A person may have 0 or more children (unbounded)
  - A child has exactly 1 person to whom it is attached
- We need to pay attention to such matters, though we are generally not going to be listing them here
  But you should look at all lines and understand the ends and the patterns (solid or dashed)

Person And Child (2/2)

[Diagram of Person and Child relationship]
Define Table Automobile (Model NOT NULL, Year NOT NULL, Weight NOT NULL, Primary Key (Model, Year));
- Define Table Likes (ID# NOT NULL, Model NOT NULL, Year NOT NULL, Primary Key (ID#,Model,Year), Foreign Key (ID#) References Person, Foreign Key (Model,Year) References Automobile);
Define Table Car (VIN NOT NULL, Color, Primary Key (VIN));
Type

- There is no need for a table for Type as Type is a binary many-to-one relationship.
- It is essentially “stored” in the “many” side, that is in Car.

Car

- Define Table Car (VIN NOT NULL, Color, Model NOT NULL, Year NOT NULL, Weight NOT NULL, Primary Key (VIN), Foreign Key (Model, Year) References Automobile );
As Has is a binary many-to-one relationship, the attributed of this relationship, Date, is stored in the “many” side, Car

There is no need for a table for Has as Has is a binary many-to-one relationship

It is essentially “stored” in the “many” side, that is in Car

We can only specify that a Person has at least 1 Car with the notation we currently use

The CHECK condition is specified using appropriate SQL constraint syntax

This can actually be done in Visio/Erwin also
- Define Table Car (VIN NOT NULL, Color, Model NOT NULL, Year NOT NULL, Weight NOT NULL, ID#, Primary Key (VIN), Foreign Key (Model, Year) References Automobile Foreign Key (ID#) References Person);
We do not define a table for ISA

This/these relationship/s is/are “embedded” in Student and Professor

---

Define Table Student (ID# NOT NULL, Primary Key (ID#), Foreign Key (ID#) References Person, GPA (computed by procedure …) );

Note, how ISA, the class/subclass (set/subset) relations, is modeled by Visio/Erwin
Define Table Professor (ID# NOT NULL, Salary NOT NULL, Primary Key (ID#), Foreign Key (ID#) References Person);
Define Table Course (C# NOT NULL, Title NOT NULL, Description, Primary Key (C#));
- Define Table Prereq (First NOT NULL, Second NOT NULL, Primary Key (First,Second), Foreign Key (First) References Course, Foreign Key (Second) References Course);
This is our first example of a table modeling a recursive relationship, between an entity set and itself.

- We decide to name the table Prereq, as this is shorter than Prerequisite.
- Note that it is perfectly clear and acceptable to refer here to C# by new names: First and Second.
  - Similarly, to using ChildName in the Child table.
- We should add some constraint to indicate that this (directed graph) should be acyclic.
  - Maybe other conditions, based on numbering conventions specifying course levels.
Define Table Book (
Author NOT NULL,
Title NOT NULL,
Primary Key (Author, Title));
Define Table Required (ID# NOT NULL, C# NOT NULL, Author NOT NULL, Title NOT NULL, Primary Key (ID#, C#, Author, Title), Foreign Key (ID#) References Professor, Foreign Key (C#) References Course, Foreign Key (Author, Title) References Book);

Why is it **bad** to have Foreign Key (ID#) References Person, instead of Foreign Key (ID#) References Professor? Because only a Professor can Require a Book

This is our first example of a table modeling a relationship that is not binary

Relationship Required was ternary: it involved three entity sets

There is nothing unusual about handling it

We still have as foreign keys the primary keys of the “participating” entities
Define Table Section (C# NOT NULL, Year NOT NULL, Semester NOT NULL, Sec# NOT NULL, MaxSize, Primary Key (C#, Year, Semester, Sec#), Foreign Key (C#) References Course);

Note on the end of the edge between Course and Section, the Section end, on the drawing how the requirement of having at least one Section is modeled.
Section (2/2)

- Section is our first example of a weak entity

Offered

- We do not define a table for Offered
- Relationship Offered is implicit in the foreign key constraint
Section + Offered

Took (1/3)

- Define Table Took (ID# NOT NULL, C# NOT NULL, Year NOT NULL, Semester NOT NULL, Sec# NOT NULL, Grade, Primary Key (ID#, C#, Year, Semester, Sec#), Foreign Key (ID#) References Student, Foreign Key (C#, Year, Semester, Sec#) References Section);
- Note on the end of the edge between Section and Took, the Took end, on the drawing how the requirement of having between 3 and 50 students in a section is not fully modeled
- We can only show 1 or more using current notation
Because Took is a many-to-many relationship we store its attribute, Grade, in its table.

We cannot store Grade in any of the two Section or Student.
Define Table Taught (ID# NOT NULL, C# NOT NULL, Year NOT NULL, Semester NOT NULL, Sec# NOT NULL, Primary Key (ID#,C#,Year,Semester,Sec#), Foreign Key (ID#), References Professor, Foreign Key (C#,Year,Semester,Sec#) References Section);
Monitors

- This is our first example in which a table, Taught, that “came from” a relationship is treated as if it came from an entity and participates in a relationship with other tables
- Nothing special needs to be done to “convert” a table that models a relationship, to be also treated as a table modeling an entity
- In this case, Monitors is a binary many-to-one relationship, so we do not need to create a table for it, and it can be stored in the “many” side, Taught

Taught

- Define Table Taught (ID# NOT NULL, C# NOT NULL, Year NOT NULL, Semester NOT NULL, Sec# NOT NULL, Monitor Primary Key (ID#, C#, Year, Semester, Sec#), Foreign Key (ID#), References Professor, Foreign Key (C#, Year, Semester, Sec#) References Section Foreign Key (Monitor) References Professor);
Monitors

We Are Done
Additional Points

- We will discuss some additional, important, points
  - Elaboration on recursive relationships
  - Referential Integrity
  - Temporal databases

Recursive Relationships: Example (1/2)

- Assume now that a prerequisite course, “First” course, must be taken with at least some Grade to count as a prerequisite
- This to make an example a little “richer”
- Two cases:
  - A course may have any number of prerequisites
    - Prereq is many-to-many
  - A course may have at most one prerequisite
    - Prereq is many to one (Second is the many side, a single First could be a prerequisite for many Second courses)
### Recursive Relationships: Example (2/2)

- Nothing special, we handle the second case of Prereq by storing it in the “many” side of the relationship
- So there are two additional attributes in Course1
  - The prerequisite course, if any
  - The required grade, if any

### Referential Integrity: Example (1/3)

Assume that we have some professors in table Professor, with rows: 5,1 and 7,2
- There is a row in Taught 5,G22.2433,2009,Spring,001,7
- This means that 5 teaches a specific section and 7 monitors this assignment
A user accesses the database and attempts to delete row (or all rows like this, recall that duplicates are permitted) 5,1 from Professor.

What should happen, as there is a row in Taught referencing this row in Professor?

A user accesses the database and attempts to delete row 7,2 from Professor.

What should happen, as there is a row in Taught referencing this row in Professor?

---

Part of specification of foreign key in in Taught

An action on Professor can be denied, or can trigger an action on Taught

For example

- ON DELETE NO ACTION
  
  This means that the "needed" row in Professor cannot be deleted
  
  Of course, it is possible to delete the row from Taught and then from the Professor (if no other row in in any table in the database "needs" the row in Professor)

- ON DELETE CASCADE
  
  This means that if the a row is deleted from Professor, all the rows in Taught referring to it are deleted too

- ON DELETE SET NULL
  
  This means, that the value referring to no-longer-existing professor is replaced by NULL
  
  In our example, this is not possible for ID# as it is a part of the primary key of Taught, but is possible for Monitor
Referential Integrity: Another Example

- Part of specification of foreign key in Professor
- An action on Person can be denied, or can trigger an action on Professor
- For example
  - ON UPDATE CASCADE
    - This means that if the value of ID# in Person is changed, this value of ID# also propagates to Professor
- Could (and probably should) add to Taught and Required:
  - ON UPDATE CASCADE
    - In appropriate attributes, so that the change of ID# in Professor also propagates to them
    - In Taught in both ID# and Monitor
    - In Required in ID#
- Excellent mechanism for centralized maintenance

Temporal Databases

- Of course, we may want to maintain historical data
- So, in practice one may have some indication that the professor no longer works, but still keep historical information about the past
- But we do not assume this for our example
Summary: Strong Entity (1/2)

- Example: *Person*
- Create a table for the entity without multivalued and derived attributes, flattening composite attributes
  - The primary key of this table will consist of the attributes serving as primary key of the entity
  - Example table: Person
- If there is a derived attribute, describe how it is computed, but do not store it
- If there is a multivalued attribute, create a table for it consisting of it and attributes of the primary key of the entity; do not put it in the table for the entity
  - Example table: Child
  - The primary key of this table will consist of all its attributes

Summary: Strong Entity (2/2)

- There could be an attribute that is composite with some components being multivalued and some derived
- And similar complexities
- Example, without drawing the appropriate entity using the ER model (this is getting too hairy)
  - A person has many children (multivalued)
  - Each child has both FirstName and MiddleName
  - The child has DOB
  - The child has Age
- Then the table for child will look like

<table>
<thead>
<tr>
<th>Child</th>
<th>ID#</th>
<th>FirstName</th>
<th>MiddleName</th>
<th>DOB</th>
</tr>
</thead>
<tbody>
<tr>
<td>5432</td>
<td>Krishna</td>
<td>Satya</td>
<td>2006-11-05</td>
<td></td>
</tr>
</tbody>
</table>
Summary: ISA And A Subclass

- Example: *ISA* and *Professor*
  - Do not do anything for ISA
  - The class “above” ISA (here Person) has already been implemented as a table
  - Create a table with all the attributes of the subclass (as for strong entity above) augmented with the primary key of the table “above” ISA, and no other attributes from it
  - The primary key is the same as the primary key of the table “above” ISA
  - Example table: Professor

Summary: Weak Entity And Defining Relationship

- Example: *Offered* and *Section*
  - Do not do anything for the defining relationship, here Offered
  - Imagine that the weak entity is augmented by the primary key of the “stronger” table through which it is defined (the table for it has been created already)
  - Treat the augmented weak entity the same way as a strong entity
  - The primary key is the primary key of the “stronger” table augmented by the attributes in the discriminant of the weak entity (a discriminant may consist of more than one attribute)
  - Example table: Section and Offered
Summary: A Relationship That Is Not Binary Many-To-One

- Example *Took*
  The tables for the participating entities have already been created
  Create a table consisting of the primary keys of the participating tables and the attributes of the relationship itself
  Of course, treat attributes of the relationship that are derived, multivalued, or composite, appropriately, not storing them, producing additional tables, flattening them
  The primary key consists of all the attributes of the primary keys of the participating tables
  Example table: Took

Summary: A Relationship That Is Binary Many-To-One

- Example: *Has*
  Do not create a table for this relationship
  Put the attributes of the primary key of the “one” side and the attributes of the relationship itself into the table of the “many” side
  Of course, treat attributes of the relation that are derived, multivalued, or composite, appropriately, not storing them, producing additional tables, flattening them, as the case may be
  You may decide to treat such a relationship the way you treat a relationship that is not binary many to one (but not in our class)
  If the relationship is one-to-one, choose which side to treat as if it were “many”
  Example table: Has
### Summary: Treating A Relationship As An Entity

- **Example:** *Taught* (before it was modified by removing Approved)
  - We have a table for that was created when we treated it as a relationship
  - We do not need to do anything else to this table
  - Example table: Taught

### Agenda

<table>
<thead>
<tr>
<th></th>
<th>Session Overview</th>
<th>ER and EER to Relational Mapping</th>
<th>Database Design Methodology and UML</th>
<th>Mapping Relational Design to ER/EER Case Study</th>
<th>Summary and Conclusion</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Summary

- Basic ER model concepts of entities and their attributes
  - Different types of attributes
  - Structural constraints on relationships
- ER diagrams represent E-R schemas
- UML class diagrams relate to ER modeling concepts
- Enhanced ER or EER model
  - Extensions to ER model that improve its representational capabilities
  - Subclass and its superclass
  - Category or union type
- Notation and terminology of UML for representing specialization and generalization

Assignments & Readings

- Readings
  - Slides and Handouts posted on the course website
  - Textbook: Chapters 9 & 10
- Assignment #2
  - Textbook exercises: Textbook exercises: 7.17, 7.27, 7.30, 8.19, 8.21, 8.24
- Project Framework Setup (ongoing)