Data Modeling

Yanlei Diao
UMass Amherst
Jan 31 and Feb 5, 2007
Outline

- Conceptual Design: ER model
- Relational Model
- Logical Design: from ER to Relational
Overview of Database Design

- **Conceptual design using** Entity-Relationship (ER) Modeling:
  - Entities and relationships in the enterprise
  - Information about these entities and relationships
  - Integrity constraints (or business rules) that hold

- **ER diagrams** pictorially represent all of the above.
**Entity Set**

- **Entity**: real-world object.
  - Described using a set of attributes.
- **Entity Set**: a collection of similar entities.
  - Same set of attributes.
  - **Key**: minimal set of attributes that uniquely identify each entity in the set.
  - **Domain** of attribute: the set of possible values.
Relationship Set

- **Relationship**: association among two or more entities.
- **Relationship Set**: collection of similar relationships.
  - An *n-ary* (*n*≥2) relationship set *R* relates *n* entity sets *E_1*, ..., *E_n*. 
Same entity set can participate in different relationship sets.
Relationship Set (Contd.)

- Same entity set can participate in the same relationship set but in different “roles”.

```
Employees
  
  name
  ssn
  lot

Reports_To
  
  supervisor
  subordinate
```
A Works_In relationship:
- Captures constraint that an employee is in a certain location when working for a particular department
- Why ternary? Why not two binary relationships?
Many-to-Many Relationship

- **Works_In:**
  - an employee can work in 0, 1, or many departments;
  - a dept can have 0, 1, or many employees.
  - "many-to-many" relationship
Key Constraints: one-to-many

- **Key constraint** on Manages: each dept has at most one manager.
  - “at most one” depicted by an arrow from Depts to Manages
  - “one-to-many” relationship
Participation Constraints

- Participation constraint on Manages: each dept has at least one manager
  - The participation of Departments in Manages is total (otherwise partial).
  - “At least one” is depicted by a solid line.

Must every department have a manager?
Key and Participation Constraints

Must every department have one and exactly one manager?

- Key and participation constraints: exactly one
Weak Entities

A weak entity can be identified uniquely only by considering the primary key of another (owner) entity.
- A weak entity must have one and exactly one owner entity.

Does an entity set always have a key?

- A weak entity can be identified uniquely only by considering the primary key of another (owner) entity.
  - A weak entity must have one and exactly one owner entity.
ISA (\textit{`is a'}) Hierarchies

- Y \textbf{ISA} X: every Y entity is also considered to be an X entity.
  - Y entity set inherits all attributes of X entity set.
  - Y entity set has its own descriptive attributes.
Issues with ISA Hierarchies

- Reasons for using ISA: classify entities into subclasses s.t.
  - can add descriptive attributes specific to a subclass.
  - can identify entities that participate in a specific relationship.
- Overlap constraints: Can Joe be an Hourly_Emps as well as a Contract_Emps entity? (Allowed/disallowed)
- Covering constraints: Does every Employees entity have to be an Hourly_Emps or a Contract_Emps entity? (Yes/no)
Aggregation: models a relationship involving a relationship set.
- treats a relationship set as an entity set for participation in another relationship.
Other Issues to Consider…

- **Design choices:**
  - Should a concept be modeled as an entity or an attribute?
  - Should a concept be modeled as part of an entity or a relationship?
  - Identifying relationships: Binary or ternary? Aggregation?
Entity vs. Attribute

- Should *address* be an attribute of Employees or an entity (connected to Employees by a relationship)?
  - What if we have several addresses per employee?
  - What if the structure (city, street, etc.) is important, e.g., we want to retrieve employees in a given city?
Entity vs. Relationship

- First diagram OK if a manager gets a separate budget for each dept.

- What if a manager gets a budget that covers all managed depts?
Binary vs. Ternary Relationships

- Use binary relationships whenever you can.
- But sometimes n-ary (n>2) relationships, rather than separate binary relationships, are needed:
  - Think of our example of an employee staying in a certain location when working for a particular department!
Aggregation vs. Ternary Relationship

- **Aggregation:**
  - Recall the example: employees monitor sponsorships (between departments and projects).
  - Monitors is a distinct relationship with its own attributes; so is Sponsors.
  - Can add *key constraint* (→) from sponsorships to monitors: “monitored by at most one employee”.

- **Ternary relationship**
  - A single relationship involving multiple entities.
Summary of Conceptual Design

- Conceptual design follows requirements analysis,
  - Yields a high-level description of data to be stored
- ER model popular for conceptual design
  - Constructs are expressive, close to the way people think about their applications.
- Basic constructs: entities, relationships, and attributes (of entities and relationships).
- Some additional constructs: weak entities, ISA hierarchies, and aggregation.
Summary of ER (Contd.)

- Integrity constraints: key constraints, participation constraints, and overlap/covering constraints for ISA hierarchies.
  - Some foreign key constraints are also implicit in the definition of a relationship set. More later…
  - Constraints play an important role in determining the best database design for an enterprise.

- Note: There are many variations on ER model.
Outline

- Conceptual Design: ER model
- Relational Model
- Logical Design: from ER to Relational
Relational Model

- **Relational database**: is a set of relations
- **Relation**: schema + instance
  - Schema: specifies name of relation, name and type (domain) of each attribute
  - Instance: a table with rows (tuples) and columns (attributes, fields)
    - cardinality = #rows, degree / arity = #columns.
- Relation is a set of rows (tuples)
  - All rows must be distinct, no duplicates (in theory!)
**Example Instance of Students Relation**

<table>
<thead>
<tr>
<th>Students</th>
<th>sid</th>
<th>name</th>
<th>login</th>
<th>age</th>
<th>gpa</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>53666</td>
<td>Jones</td>
<td>jones@cs</td>
<td>18</td>
<td>3.4</td>
</tr>
<tr>
<td></td>
<td>53688</td>
<td>Smith</td>
<td>smith@eecs</td>
<td>18</td>
<td>3.2</td>
</tr>
<tr>
<td></td>
<td>53650</td>
<td>Smith</td>
<td>smith@math</td>
<td>19</td>
<td>3.8</td>
</tr>
</tbody>
</table>

- Cardinality = ? degree = ?
- All rows are distinct
- Do all columns in a relation instance have to be distinct?
Relational Query Languages

- A major strength of the relational model: supports simple, powerful query of data.
- Relational query languages:
  - High-level declarative: say “what you want” not “how you get it”
  - Based on a formal mathematical model.
- Benefits include:
  - Queries can be written intuitively.
  - Precise semantics of queries.
  - The DBMS is responsible for efficient evaluation.
The SQL Query Language

- Developed by IBM (system R) in the 1970s
- Need for a standard since it was used by many vendors
- Standards:
  - SQL-86
  - SQL-89 (minor revision)
  - SQL-92 (major revision)
  - SQL-99 (major extensions, current standard)
The SQL Query Language

- To find all 18 year old students, we can write:

  ```sql
  SELECT * 
  FROM Students S 
  WHERE S.age=18;
  ```

- To select just names and logins, replace the first line:

  ```sql
  SELECT S.name, S.login 
  FROM Students S 
  WHERE S.age=18;
  ```
Querying Multiple Relations

What does this query compute?

```
SELECT S.name, E.cid
FROM Students S, Enrolled E
WHERE S.sid=E.sid AND E.grade='A';
```

Instances of Students & Enrolled:

<table>
<thead>
<tr>
<th>sid</th>
<th>name</th>
<th>login</th>
<th>age</th>
<th>gpa</th>
</tr>
</thead>
<tbody>
<tr>
<td>53666</td>
<td>Jones</td>
<td>jones@cs</td>
<td>18</td>
<td>3.4</td>
</tr>
<tr>
<td>53688</td>
<td>Smith</td>
<td>smith@eecs</td>
<td>18</td>
<td>3.2</td>
</tr>
<tr>
<td>53650</td>
<td>Smith</td>
<td>smith@math</td>
<td>19</td>
<td>3.8</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>sid</th>
<th>cid</th>
<th>grade</th>
</tr>
</thead>
<tbody>
<tr>
<td>53831</td>
<td>Carnatic101</td>
<td>C</td>
</tr>
<tr>
<td>53831</td>
<td>Reggae203</td>
<td>B</td>
</tr>
<tr>
<td>53650</td>
<td>Topology112</td>
<td>A</td>
</tr>
<tr>
<td>53666</td>
<td>History105</td>
<td>B</td>
</tr>
</tbody>
</table>

we get:

<table>
<thead>
<tr>
<th>S.name</th>
<th>E.cid</th>
</tr>
</thead>
<tbody>
<tr>
<td>Smith</td>
<td>Topology112</td>
</tr>
</tbody>
</table>
Creating Relations in SQL

- Creates the Students relation.
- Specifies domain Constraints:
  - type of each field
  - later enforced by the DBMS upon tuple insertion or update.

```sql
CREATE TABLE Students
(sid CHAR(20),
 name CHAR(20),
 login CHAR(10),
 age INTEGER,
 gpa REAL);
```

```sql
CREATE TABLE Enrolled
(sid CHAR(20),
 cid CHAR(20),
 grade CHAR(2));
```
Destroying and Altering Relations

- Destroys the relation Students. The schema information and the tuples are deleted.

```
DROP TABLE Students;
```

- The schema of Students is altered by adding a new field; every tuple in the current instance is extended with a null value in the new field.

```
ALTER TABLE Students
ADD COLUMN firstYear: integer;
```
Adding and Deleting Tuples

- Can insert a single tuple using:

```
INSERT INTO Students (sid, name, login, age, gpa)
VALUES (53688, 'Smith', 'smith@ee', 18, 3.2);
```

- Can delete all tuples satisfying some condition (e.g., name = Smith):

```
DELETE
FROM Students S
WHERE S.name = 'Smith';
```

★ Powerful variants of these commands are available; more later!
Integrity Constraints

- **Integrity Constraint (IC):** condition that must be true for *any* instance of the database.
  - e.g., *domain constraints.*
  - specified when schema is defined.

- A *legal* instance of a relation is one that satisfies all specified ICs.

- DBMS later enforces ICs.
  - Stored data is more faithful to real-world meaning.
  - Avoids data entry errors, too!
Primary Key Constraints

- A set of fields is a **key** for a relation if:
  1. **Unique**: no two distinct tuples can have the same values in all key fields, and
  2. **Minimal**: this is not true for any subset of the key.
     - Part 2 false? A **superkey**.
     - If there’s >1 key for a relation, **candidate keys**.
     - One of the keys is chosen to be the **primary key** for the relation.

- E.g., Students(sid, name, login, age, gpa)
  - What is a key for Students?
  - What about *name*? What about {sid, gpa}?

- E.g., Enrolled(sid, cid, grade)
  - What is a key for Enrolled?
Primary and Candidate Keys in SQL

- Specify *candidate keys using UNIQUE*.
- Choose one of candidate keys as the *primary key*.

“For a given student and course, there is a single grade.”

“… and no two students in a course receive the same grade.”

```
CREATE TABLE Enrolled
    (sid CHAR(20),
     cid CHAR(20),
     grade CHAR(2),
     PRIMARY KEY (sid,cid));
```

```
CREATE TABLE Enrolled
    (sid CHAR(20),
     cid CHAR(20),
     grade CHAR(2),
     PRIMARY KEY (sid,cid),
     UNIQUE (cid, grade) );
```
Foreign Keys

- **Foreign key**: Set of fields in one relation used to `refer` to a tuple in another relation.
  - Must correspond to primary key of referenced relation.
  - Like a `logical pointer`.

- E.g., `Enrolled(sid: string, cid: string, grade: string)`:
  - `sid` is a foreign key referring to Students.
Foreign Keys in SQL

- Only students listed in the Students relation should be allowed to enroll for courses.

```sql
CREATE TABLE Enrolled
(sid CHAR(20), cid CHAR(20), grade CHAR(2),
 PRIMARY KEY (sid,cid),
 FOREIGN KEY (sid) REFERENCES Students);
```

<table>
<thead>
<tr>
<th>sid</th>
<th>cid</th>
<th>grade</th>
</tr>
</thead>
<tbody>
<tr>
<td>53666</td>
<td>Carnatic101</td>
<td>C</td>
</tr>
<tr>
<td>53666</td>
<td>Reggae203</td>
<td>B</td>
</tr>
<tr>
<td>53650</td>
<td>Topology112</td>
<td>A</td>
</tr>
<tr>
<td>53666</td>
<td>History105</td>
<td>B</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>sid</th>
<th>name</th>
<th>login</th>
<th>age</th>
<th>gpa</th>
</tr>
</thead>
<tbody>
<tr>
<td>53666</td>
<td>Jones</td>
<td>jones@cs</td>
<td>18</td>
<td>3.4</td>
</tr>
<tr>
<td>53688</td>
<td>Smith</td>
<td>smith@eecs</td>
<td>18</td>
<td>3.2</td>
</tr>
<tr>
<td>53650</td>
<td>Smith</td>
<td>smith@math</td>
<td>19</td>
<td>3.8</td>
</tr>
</tbody>
</table>
Referential Integrity

- **Referential integrity**: any value of a foreign key must have a corresponding primary key value in the referenced relation.
  - No dangling references.

  - *sid* is a foreign key referring to Students.
  - Every *sid* value in Enrolled must appear in Students.

- Can you name a data model w/o referential integrity?
Enforcing Referential Integrity

- What should be done if an Enrolled tuple with a non-existent student id is inserted?
  - **Reject it!**

- What should be done if a Students tuple is deleted?
  - **CASCADE:** Also delete all Enrolled tuples that refer to it.
  - **NO ACTION:** Disallow deletion of a Students tuple that is referred to.
  - **SET DEFAULT:** Set sid in a relevant Enrolled tuple to a default sid.
  - **SET NULL:** In SQL, can set sid in a relevant Enrolled tuple to a special value `null`, denoting `unknown` or `inapplicable`.

- What if a Students tuple is updated? Similar.
Referential Integrity in SQL

- SQL/92 and SQL/99 on deletes and updates.
  - **NO ACTION**: the default \textit{(delete/update is rejected)}
  - **CASCADE**: (also delete all tuples that refer to deleted tuple)
  - **SET NULL / SET DEFAULT**: (sets foreign key value of referencing tuple)

```sql
CREATE TABLE Enrolled
(sid CHAR(20),
cid CHAR(20),
grade CHAR(2),
PRIMARY KEY (sid,cid),
FOREIGN KEY (sid)
  REFERENCES Students
    ON DELETE CASCADE
    ON UPDATE NO ACTION);
```
Where do ICs Come From?

- Based upon the semantics of the real-world enterprise being described.
- Can check violation against a database instance once declared, but can NEVER infer that an IC is true by looking at an instance.
  - An IC is a statement about *all possible* instances!
  - E.g., *name* of the Students relation.
Outline

- Conceptual Design: ER model
- Relational Model
- Logical Design: from ER to Relational
Logical DB Design: ER to Relational

- Entity set is translated to a table:

```sql
CREATE TABLE Employees
(ssn CHAR(11),
 name CHAR(20),
 lot INTEGER,
 PRIMARY KEY (ssn));
```
Relationship Set to Table

- Relationship set is also translated to a table.

- Attributes include:
  1. Descriptive attributes of the relationship set.
  2. Primary key of each related entity set as a foreign key.
  3. The foreign keys together form a superkey (⊆ key) for the relation.

```sql
CREATE TABLE Works_In(
  ssn CHAR(11),
  did INTEGER,
  since DATE,
  PRIMARY KEY (ssn, did),
  FOREIGN KEY (ssn)
    REFERENCES Employees,
  FOREIGN KEY (did)
    REFERENCES Departments
);```
Review: Key Constraints

- Each dept has at most one manager, according to the **key constraint** on Manages.
Translating ER Diagrams w. Key Constraints

- **Choice 1: A separate table for Manages**
  - did is key now!
  - Borrow primary key from the entity with the key constraint.

```sql
CREATE TABLE Manages(
    ssn CHAR(11),
    did INTEGER,
    since DATE,
    PRIMARY KEY (did),
    FOREIGN KEY (ssn) REFERENCES Employees,
    FOREIGN KEY (did) REFERENCES Departments);
```

- **Choice 2: Merge Manages into Departments**
  - Merge the relationship set into the entity with the key constraint.

```sql
CREATE TABLE Dept_Mgr(
    did INTEGER,
    dname CHAR(20),
    budget REAL,
    ssn CHAR(11),
    since DATE,
    PRIMARY KEY (did),
    FOREIGN KEY (ssn) REFERENCES Employees);
```

ssn can be null!
Review: Participation Constraints

- **Participation constraint:** at least one
  - Every Departments entity must appear in an instance of the Manages relationship, with a non-null ssn value!
  - The participation of Departments in Manages is total.
Participation Constraints in SQL

- Capture both key and participation constraints (exactly-one):

```sql
CREATE TABLE Dept_Mgr(
    did INTEGER,
    dname CHAR(20),
    budget REAL,
    ssn CHAR(11) NOT NULL,
    since DATE,
    PRIMARY KEY (did),
    FOREIGN KEY (ssn) REFERENCES Employees,
    ON DELETE NO ACTION);
```

- But little else, without resorting to assertions (ICs over several tables) — see textbook Ch 5.7.
Weak Entities

- A **weak entity** can be identified uniquely only by considering the primary key of another (**owner**) entity.
  - A weak entity must have one and exactly one owner entity.
Translating Weak Entity Sets

- Weak entity set and identifying relationship set are translated into a single table.
  - When the owner entity is deleted, all associated weak entities must also be deleted.

```sql
CREATE TABLE Depndt_Policy (  
  pname CHAR(20),  
  age INTEGER,  
  cost REAL,  
  ssn CHAR(11) NOT NULL,  
  PRIMARY KEY (pname, ssn),  
  FOREIGN KEY (ssn) REFERENCES Employees,  
  ON DELETE CASCADE);
```
Review: ISA Hierarchies

Y ISA X: every Y entity is also considered to be an X entity.

- **Overlap constraints**: Can Joe be an Hourly_Emps as well as a Contract_Emps entity? *(Allowed/disallowed)*
- **Covering constraints**: Does every Employees entity also have to be an Hourly_Emps or a Contract_Emps entity? *(Yes/no)*
Translating ISA Hierarchies to Relations

- 3 relations: Employees, Hourly_Emps and Contract_Emps.
  - Employees: \((snn, \text{name}, \text{lot})\)
  - Hourly_Emps: \((snn, \text{hourly\_wages}, \text{hours\_worked})\)
    - \(snn\) both primary and foreign key!
    - What happens to Hourly_Emps tuple if referenced Employees tuple is deleted?
    - What if queries want to retrieve both name and hourly\_wages of an Hourly Employee?
  - Contrast_Emps: \((snn, \text{contract\_id})\)
    - Issues similar to those with Hourly_Emps.
Translating ISA Hierarchies to Relations

- Just Hourly_Emps and Contract_Emps.
  - Hourly_Emps: (ssn, name, lot, hourly_wages, hours_worked).
  - Contract_Emps: (ssn, name, lot, contract_id)
  - Each employee must be in one of these two subclasses (i.e., if the covering constraint is true).
Translating ER Diagrams w. Aggregation

- Straightforward solution:
  - first translate the relationship set (e.g., Sponsors) that participates in aggregation as a pseudo entity set.
  - then translate the aggregation relationship set (e.g., Monitors).

- Because there is no real distinction between entities and relationships in the relational model!
Relational Model: Summary

- A tabular representation of data.
  - Simple and intuitive, currently the most widely used.
- Integrity constraints (ICs) are specified based on application semantics (not inferred from data).
  - Two important ICs: primary and foreign keys
  - In addition, always have domain constraints.
  - DBMS automatically checks for violations.
- Powerful and natural query language: SQL.
- Rules to translate ER to relational model.