Data Modeling

Yanlei Diao
UMass Amherst
Outline

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

- Start with application requirements
- Use *Entity-Relationship (ER) Model*:
  - *Entities and relationships* in the enterprise
  - *Integrity constraints* (or business rules) that hold
  - Pictorially represented by an ER diagram
ER Model Basics: Entities

- **Entity**: A real-world object.
  - Described using a set of attributes.

- **Entity Set**: A collection of entities described by the same set of attributes.
  - **Domain** of an attribute.
  - **Key** of an entity set: **minimum** set of attributes that uniquely "identify" each entity in the set.
**ER Model Basics: Relationships**

- **Relationship**: Association between 2+ entities.
  - E.g., Joe works in the accounting dept since 01/2008.

- **Relationship Set**: Collection of similar relationships.
A Works_In relationship involves:
- an employee
- a department
- a location
More on Relationships

- An entity set can participate in the same relationship set, but in different roles.

![Diagram showing an entity set labeled Employees with attributes name, ssn, office, supervisor, and subordinate, connected by a diamond labeled Reports_To.](image-url)
Key Constraints

- **Works_In**: an employee can work in many depts; a dept can have many employees
  - many-to-many

- **Manages**: each dept is managed by at most one manager
  - key constraint on Manages (←)
  - one-to-many
**Participation Constraints**

- **Works_In**: every employee works in at least one dept (or, every employee must work for some dept)
  - *Participation constraint* on Works_in (denoted using a *thick line*).
  - Participation of Employees in Works_In is *total* (vs. *partial*).
- 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 an exactly one relationship with its owner.
- Notation: see the example.

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.
It is sometimes natural to classify entities into subclasses.

Y 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

- **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

- Reasons for using ISA:
  - Add descriptive attributes specific to a subclass.
  - Identify entities that participate in a specific relationship.
Useful things to know about ER

- When reading application requirements:
  - Entities, attributes are often extracted from *nouns*
    - Entities vs. attributes: should *address* be an attribute of Employees or an entity?
      - Sub-structure?
      - Participation in other relationships?
  - Relationships are often from *verbs*

- Software tools for ER modeling
  - Word on both Mac and Windows
  - Microsoft Visio for Windows
  - OmniGraffle for Mac
  - Draw.io (Google)
Outline

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

- A **relational database** is a set of relations.
- A **relation** has:
  - *Schema*: specifies (1) name of relation, (2) name and domain of each attribute.
  - *Instance*: a table with rows (**tuples**) and columns (*attributes, fields*). *cardinality* = #rows, *degree / arity* = #columns.

- Relation is a **set** of tuples (in theory).
  - All rows must be distinct, no duplicates.
Example Instance of Students Relation

<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>

- Cardinality = 3, degree = 5
- All rows are distinct.
- Some columns of two rows can be the same.
Creating Relations in SQL

- Create the Students relation
  - Specify schema
  - Domain constraint:
    - type of each attribute
    - 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

- Destroy the relation Students.

\[
\text{DROP TABLE Students;}
\]

- Alter the Students relation by adding a new field.
  - Every tuple in the current instance is extended with a null value in the new field.

\[
\text{ALTER TABLE Students}
\text{ ADD COLUMN firstYear: integer;}
\]
Integrity Constraints

- *Integrity Constraints* (IC’s): condition that must be true for **any** instance of the database.
  - Domain constraint
  - Primary key constraint
  - Foreign key constraint
  - ...
  - Specified when schema is defined.
Primary Key Constraints

- **Key** of a relation: **minimum** set of attributes that **uniquely** identify each entity.
  1. No two tuples can have same values in all key fields.
  2. This is not true for any subset of the key.
  - Part 2 false? A **superkey**.
  - If more than 1 key for a relation, **candidate keys**.
  - One of candidate keys is chosen to be the **primary key**.

- E.g., Students(sid, name, login, age, gpa)
- E.g., Enrolled (sid, cid, grade)
1. Specify *candidate keys* using `UNIQUE`.
2. Choose one candidate key as the *primary key*.

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

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

“(… 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),
    UNIQUE (cid, grade));
Foreign Keys

- **Foreign key**: set of attributes used to ‘refer’ to the primary key of another relation.

- E.g., Enrolled\((\text{sid}: \text{string, cid: string, grade: string})\):
  - \(\text{sid}\) is a foreign key referring to \text{sid} in \text{Students}.

<table>
<thead>
<tr>
<th>Enrolled</th>
<th>Students</th>
</tr>
</thead>
<tbody>
<tr>
<td>sid</td>
<td>cid</td>
</tr>
<tr>
<td>53666</td>
<td>Carnatic101</td>
</tr>
<tr>
<td>53666</td>
<td>Reggae203</td>
</tr>
<tr>
<td>53650</td>
<td>Topology112</td>
</tr>
</tbody>
</table>
**Foreign Keys in SQL**

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

- **Referential integrity:**
  - Any foreign key value must have a matching primary key value in the referenced reln.
  - E.g., every sid value in Enrolled must appear in Students.
  - No dangling references.

- In contrast, consider links in HTML. Does referential integrity hold?
Enforcing Referential Integrity

- What if an Enrolled tuple with a non-existent student id is inserted to the DB?
  - *Reject it!*

- What if a Students (referenced) tuple is deleted?
  - **CASCADE**: delete all Enrolled tuples that refer to it.
  - **NO ACTION**: disallow if the Students tuple is referred to.
  - **SET DEFAULT**: set the foreign key to a *default sid*.
  - **SET NULL**: set the foreign key to a special value *null*, denoting ‘unknown’ or ‘inapplicable’.
Enforcing Referential Integrity

- Updates to *sid* in Students are treated similarly.
  - **CASCADE**: propagate the update to Enrolled
  - **NO ACTION**: disallow the update in Students
  - **SET DEFAULT**: set the sid in Enrolled to a default value
  - **SET NULL**: set the sid in Enrolled to *null*
Referential Integrity in SQL

CREATE TABLE Enrolled
  (sid CHAR(20),
   cid CHAR(20),
   grade CHAR(2),
   PRIMARY KEY (sid,cid),
   FOREIGN KEY (sid) REFERENCES Students(sid)
    ON DELETE NO ACTION
    ON UPDATE CASCADE);
Comments on Integrity Constraints

- IC’s are based on real-world business logic.
  - Can check violation in a database instance, but can never infer an IC by looking at an instance.
  - An IC is a statement about all possible instances!
  - E.g., name of Students can be unique in an instance.

- IC’s are specified when defining the schema (CREATE TABLE).

- DBMS later enforces IC’s.
  - Stored data is faithful to the real-world meaning.
  - Avoids data entry errors, too!
Outline

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

- An entity set is translated to a table.

```sql
CREATE TABLE Employees
(ssn CHAR(11),
 name CHAR(20),
 office INTEGER,
 PRIMARY KEY (ssn));
```
Relationship Sets to Tables

- Each relationship set is also translated to a table with
  - all descriptive attributes,
  - primary key of each related entity set as a foreign key.

- All foreign keys form a superkey of the relation.

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

- **Key constraint:** Each dept is managed by *at most one* manager.

![Entity-Relationship Diagram]

- Employees
  - ssn
  - name
  - office

- Manages
  - since

- Departments
  - dname
  - did
  - budget
Translating ER Diagrams w. Key Constraints

- A separate table for Manages:
  - 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 (ssn),
  FOREIGN KEY (did) REFERENCES Departments(did));
```

- Merge Manages into Departments:
  - Merge the relationship 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);
```
Review: Participation Constraints

- **Participation constraint:**
  - Every employee works in at least one dept.
  - Each Dept has at least one employee.

- **Participation + Key constraints:**
  - Every department must have one manager.
Key and Participation Constraints

- If we have key + 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(ssn)
    ON DELETE NO ACTION
    ON UPDATE CASCADE);
```

- For participation constraints only, need to resort to assertions (dynamic checks in SQL). More in SQL lecture…
A weak entity can be identified uniquely only by considering the primary key of the owner entity.

- Must have an exactly one relationship with its owner.
Translating Weak Entity Sets

- Merge weak entities and identifying relationships in one table.
  - What is the primary key?
  - What if the owner entity is deleted?

```sql
CREATE TABLE Depndt_Policy (
    pname CHAR(20),
    age INTEGER,
    cost REAL,
    ssn CHAR(11),
    PRIMARY KEY (ssn, pname, age),
    FOREIGN KEY (ssn) REFERENCES Employees(ssn)
    ON DELETE CASCADE
    ON UPDATE CASCADE);
```
Review: ISA Hierarchies

- **Employees**
  - name
  - ssn
  - office

- **Hourly_Emps**
  - hourly_wages

- **Contract_Emps**
  - hours_worked
  - contractid

ISA
Translating ISA Hierarchies to Relations

1. Create tables for both parent and child entities
   - Employees: \((ssn, name, lot)\)
   - Hourly_Emps: \((ssn, hourly_wages, hours_worked)\)
     - \(ssn\) is both primary and foreign key!
     - Must delete Hourly_Emp if referenced Emp is deleted.

2. Create tables only for child entities
   - Hourly_Emps: \((ssn, name, lot, hourly_wages, hours_worked)\).
   - Each employee must be in one of these two subclasses.

3. Create a table only for the parent entity

Which one is better for: covering constraint (Y/N), overlap constraint (Y/N)?