Data Modeling
Yanlei Diao
UMass Amherst
Feb 1, 2007

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

Overview of Database Design
- Conceptual design using 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.
- Can map an ER diagram into a relational schema.
**ER Model Basics**

- **Entity**: Real-world object. Described using a set of attributes.
- **Entity Set**: A collection of similar entities.
  - Each entity in an entity set has the same set of attributes.
  - Each entity set has a **key**, minimal set of attributes that uniquely identify the entity in the set.
  - Each attribute has a **domain** of possible values.

**ER Model Basics (contd.)**

- **Relationship**: Association among two or more entities.
- **Relationship Set**: Collection of similar relationships.
  - An n-ary relationship set \( R \) relates n entity sets \( E_1 \ldots E_n \).
  - Each relationship in \( R \) involves entities \( e_1 \) in \( E_1 \), \ldots, \( e_n \) in \( E_n \).
  - Same entity set can participate in different relationship sets or in different “roles” in same set.

**Ternary Relationships**

- A Works_In relationship involves:
  - an employee
  - a department
  - a location
**Key Constraints**
- **Works_In**: an employee can work in many departments; a dept can have many employees.
  - "many-to-many"
- **Manages**: each dept has at most one manager, a key constraint on Manages.
  - Each dept appears in at most one relationship in manages.
  - "one-to-many"

**Participation Constraints**
- Participation constraint: at least one
  - Every Departments entity must appear in an instance of the Manages relationship.
  - The participation of Departments in Manages 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.
  - One-to-many relationship set, one owner, many weak entities.
  - Weak entity set must have total participation in this identifying relationship set.
ISA (‘is a’) Hierarchies

- It is sometimes natural to classify entities into subclasses.
- A ISA B: every A entity is also considered to be a B entity.
- A entity set inherits all attributes of B entity set.
- A entity set has other descriptive attributes.

```
Employees

hourly_wages  hours_worked

Hourly_Emps  Contract_Emps
```

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 also have to be an Hourly_Emps or a Contract_Emps entity? (Yes/no)

- Reasons for using ISA:
  - To add descriptive attributes specific to a subclass.
  - To identify entities that participate in a relationship.

Aggregation

- **Aggregation**: models a relationship involving a relationship set.
- Treats a relationship set as an entity set
- For participation in another relationship.

```
Departments  Projects  Sponsors

name  did  started_on  since

Employees  Monitors

ssn  lot  sname  dname

Projects  budget

Sponsors  did
```

- **Aggregation vs. ternary relationship**:
  - Monitors is a distinct relationship
  - Can add key constraint (→) from sponsorships to monitors: “monitored by at most one employee”
Relational Model

- Relational database: is a set of relations
- Relation: made up of 2 parts
  - Schema: specifies name of relation, as well as name and type (domain) of each column.
  - Instance: a table with rows (tuples) and columns (attributes, fields).
    - #Rows = cardinality, #Columns = degree / arity.
- Relation is a set of rows or tuples
  - All rows must be distinct.
  - This is in theory, not in practice!

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 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 querying 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)

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

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

- To select just names and logins, replace the first line:
  
  ```
  SELECT S.name, S.login 
  ```

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 Enrolled & Students:

<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>

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

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

CREATE TABLE Enrolled
(sid: CHAR(20),
cid: CHAR(20),
grade: CHAR(2))

Integrity Constraints (ICs)

- **IC**: condition that must be true for *any* instance of the database; e.g., *domain constraints*.
  - specified when schema is defined.
  - checked by DBMS when relations are modified.
- A *legal* instance of a relation is one that satisfies all specified ICs.
- As the DBMS checks 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. **Uniqueness**: No two distinct tuples can have same values in all key fields, and
  2. **Minimality**: 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*.
- E.g., sid is a key for Students. What about name? What about [sid, gpa]?
Primary and Candidate Keys in SQL

- Possibly many candidate keys (specified using UNIQUE), one of which is chosen as the primary key.

```
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, Referential Integrity

- **Foreign key**: Set of fields in one relation that is used to ‘refer’ to a tuple in another relation.
  - Like a ‘logical pointer’.
  - Must correspond to primary key of the second relation.

- E.g. Enrolled(sid: string, cid: string, grade: string):
  - sid is a foreign key referring to Students.
  - If all foreign key constraints are enforced, referential integrity is achieved, i.e., no dangling references.
  - Can you name a data model w/o referential integrity?

Foreign Keys in SQL

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

```
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>Enrolled</th>
<th>Students</th>
</tr>
</thead>
<tbody>
<tr>
<td>sid</td>
<td>sid</td>
</tr>
<tr>
<td>53666</td>
<td>53666</td>
</tr>
<tr>
<td>Carnatic101</td>
<td>jones@cs</td>
</tr>
<tr>
<td>53666</td>
<td>53688</td>
</tr>
<tr>
<td>Reggae203</td>
<td>smith@eecs</td>
</tr>
<tr>
<td>53650</td>
<td>53650</td>
</tr>
<tr>
<td>Topology112</td>
<td>smith@math</td>
</tr>
<tr>
<td>53666</td>
<td>53666</td>
</tr>
<tr>
<td>History105</td>
<td>jones@math</td>
</tr>
</tbody>
</table>
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'.
- Similar if primary key of Students tuple is updated.

Referential Integrity in SQL

- SQL/92 and SQL 1999 on deletes and updates.
  - Default is **NO ACTION** (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)

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 SET DEFAULT)

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 is not a key, but it can be unique in a small instance.
Logical Design: ER to Relational

- Entity set is translated to table:

CREATE TABLE Employees
(ssn CHAR(11),
name CHAR(20),
lot INTEGER,
PRIMARY KEY (ssn))

Relationship Sets to Tables

- Relationship set is also translated to table.
- Attributes of the table must include:
  - All descriptive attributes.
  - Primary key for each related entity set as a foreign key.
  - The foreign keys together form a superkey for the relation.

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

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

- Merge Manages into Departments
  - Merge the relationship set into 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)
```

```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:** 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 participation constraints involving one entity set in a binary relationship:

```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.
  - One-to-many relationship set, one owner, many weak entities.
  - Weak entity set must have total participation in this identifying relationship set.

**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**

- A ISA B: every A entity
  - is also considered to be a B entity
  - inherits all attributes of B entity set and has other descriptive attributes.

- 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: \((\text{ssn}, \text{name}, \text{lot})\)
  - Hourly_Emps: \((\text{ssn}, \text{hourly_wages}, \text{hours_worked})\)
    - \text{ssn} both primary and foreign key!
    - Must delete Hourly_Emps tuple if referenced Employees tuple is deleted.
    - Queries involving Hourly_Emps may require a join with Employees to get some attributes.
  - Just Hourly_Emps and Contract_Emps.
    - Hourly_Emps: \((\text{ssn}, \text{name}, \text{lot}, \text{hourly_wages}, \text{hours_worked})\).
    - Each employee must be in one of these two subclasses.

Translating ER Diagrams w. Aggregation

- Easy because there is no real distinction between entities and relationships in the relational model!

Questions