Introduction to Data Management

Lecture #6
(E-R → Relational, Cont.)

Instructor: Mike Carey
mjcarey@ics.uci.edu

Announcements

- HW#2 is underway!
  - Based on our provided HW #1 solution
- Today’s plan:
  - E-R → Relational translation (cont’d.)
  - Some (hopefully) clarifying examples!
- This week’s discussion sessions will...
  - Start out with a quick E-R quiz
  - Move on to Q&A about HW #1’s solution
  - Also time for MySQL installation Q&A
**Properly Reflecting Key Constraints**

- So what are the translated relationship table’s keys (etc.) when…
  - FooBar is M:N? → FooBar(fooId, barId, baz)
  - FooBar is N:1? → FooBar(fooId, barId, baz)
  - FooBar is 1:N? → FooBar(fooId, barId, baz)
  - FooBar is 1:1? → FooBar(fooId, barId, baz) (Note: unique)

**Review: Participation Constraints**

- Does every department have a manager?
  - If so, this is a participation constraint: the participation of Departments in Manages is said to be total (vs. partial).
  - Every did value in Departments table must appear in a row of the Manages table (with a non-null ssn value!!)
Participation Constraints in SQL

- We can capture participation constraints involving one entity set in a binary relationship, but little else (without resorting to the use of triggers).

```sql
CREATE TABLE Department2 (
    did INTEGER,
    dname CHAR(20),
    budget REAL,
    mgr_ssn CHAR(11) NOT NULL,
    mgr_since DATE,
    PRIMARY KEY (did),
    FOREIGN KEY (mgr_ssn) REFERENCES Employees,
    ON DELETE NO ACTION
) /*or: RESTRICT*/
```

Review: Weak Entities

- A **weak entity** can be identified (uniquely) only by considering the primary key of another (**owner**) entity.
  - Owner entity set and weak entity set must participate in a one-to-many relationship set (1 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 of its owned weak entities must also be deleted.

```sql
CREATE TABLE Dependents2 ( 
  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

- As in C++, or other PLs, attributes are inherited.
- If we declare A ISA B, then every A entity is also considered to be a B entity.

- **Overlap constraints**: Can employee Joe be an Hourly_Emps as well as a Contract_Emps entity? *(Allowed/disallowed)*
- **Covering constraints**: Must each Employees entity be either an Hourly_Emps or a Contract_Emps entity? *(Yes/no)*
Translating ISA Hierarchies to Relations

- **Most general and “clean” approach:**
  - 3 relations: Employees, Hourly_Emps, and Contract_Emps.
  - *Hourly_Emps:* Every employee recorded in Employees. For hourly emps, extra info recorded in Hourly_Emps (hourly_wages, hours_worked, ssn); delete Hourly_Emps tuple if referenced Employees tuple is deleted.
  - Queries about all employees easy; those involving just Hourly_Emps require a join to access the extra attributes.

- **Another alternative:** Hourly_Emps and Contract_Emps.
  - *Hourly_Emps:* ssn, name, lot, hourly_wages, hours_worked.
  - Each employee must be in one of these two subclasses. (Q: Can we always do this, then? A: Not w/o redundancy!)

Another Mapping Example: Binary vs. Ternary Relationships

(“Better design”)
Binary vs. Ternary Relationships (Contd.)

- The key constraints let us combine Purchaser with Policies and Beneficiary with Dependents.
- Participation constraints lead to NOT NULL constraints. *(Note: Primary key attributes are NOT NULL as well – check documentation to see if that’s implicit or explicit!)*

```sql
CREATE TABLE Policies (
    policyid INTEGER,
    cost REAL,
    emp_ssn CHAR(11) NOT NULL,
    PRIMARY KEY (policyid),
    FOREIGN KEY (emp_ssn) REFERENCES Employees
    ON DELETE CASCADE)
```

```sql
CREATE TABLE Dependents (
    pname CHAR(20),
    age INTEGER,
    policyid INTEGER NOT NULL,
    PRIMARY KEY (pname, policyid),
    FOREIGN KEY (policyid) REFERENCES Policies
    ON DELETE CASCADE)
```

Review: Binary vs. Ternary Relationships

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

```sql
CREATE TABLE Policies (policyid INTEGER,
    cost REAL,
    emp_ssn CHAR(11) NOT NULL,
    PRIMARY KEY (policyid),
    FOREIGN KEY (emp_ssn) REFERENCES Employees
    ON DELETE CASCADE)
```

```sql
CREATE TABLE Dependents (pname CHAR(20),
    age INTEGER,
    policyid INTEGER NOT NULL,
    PRIMARY KEY (pname, policyid),
    FOREIGN KEY (policyid) REFERENCES Policies
    ON DELETE CASCADE)
```
An Example: Putting It Together

```sql
CREATE TABLE Customer (
    cid INTEGER,
    cname VARCHAR(50),
    login VARCHAR(20) NOT NULL,
    PRIMARY KEY (cid),
    UNIQUE (login))

CREATE TABLE Order (
    oid INTEGER,
    custid INTEGER,
    shipto VARCHAR(200),
    total DECIMAL(8,2),
    PRIMARY KEY (oid),
    FOREIGN KEY (custid) REFERENCES Customer)

CREATE TABLE LineItem (
    oid INTEGER,
    lno INTEGER,
    price DECIMAL(8,2),
    qty INTEGER,
    sku INTEGER,
    PRIMARY KEY (oid, lno),
    FOREIGN KEY (oid) REFERENCES Order
    ON DELETE CASCADE, FOREIGN KEY (sku) REFERENCES Product)
```

Putting It Together (Cont’d.)
Putting It Together (Cont’d.)

<table>
<thead>
<tr>
<th>cid</th>
<th>cname</th>
<th>login</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Smith, James</td>
<td><a href="mailto:jsmith@aol.com">jsmith@aol.com</a></td>
</tr>
<tr>
<td>2</td>
<td>White, Susan</td>
<td><a href="mailto:suzie@gmail.com">suzie@gmail.com</a></td>
</tr>
<tr>
<td>3</td>
<td>Smith, James</td>
<td><a href="mailto:js@hotmail.com">js@hotmail.com</a></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>sku</th>
<th>pname</th>
<th>color</th>
<th>listprice</th>
</tr>
</thead>
<tbody>
<tr>
<td>123</td>
<td>Frozen DVD</td>
<td>null</td>
<td>24.95</td>
</tr>
<tr>
<td>456</td>
<td>Graco Twin Stroller</td>
<td>green</td>
<td>199.99</td>
</tr>
<tr>
<td>789</td>
<td>Moen Kitchen Sink</td>
<td>black</td>
<td>350.00</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>oid</th>
<th>custid</th>
<th>shipto</th>
<th>total</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>3</td>
<td>J. Smith, 1 Main St., USA</td>
<td>199.95</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>Mrs. Smith, 3 State St., USA</td>
<td>300.00</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>oid</th>
<th>lno</th>
<th>price</th>
<th>qty</th>
<th>item</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>169.95</td>
<td>1</td>
<td>456</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>15.00</td>
<td>2</td>
<td>123</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>300.00</td>
<td>1</td>
<td>789</td>
</tr>
</tbody>
</table>

Relational Model and E-R Schema Translation: Summary

- Relational model: a tabular representation of data.
- Simple and intuitive, also widely used.
- Integrity constraints can be specified by the DBA based on application semantics. DBMS then checks for violations.
  - Two important ICs: Primary and foreign keys (PKs, FKs).
  - In addition, we *always* have domain constraints.
- Powerful and natural query languages exist (soon!)
- Rules to translate E-R to relational model
  - Can be done by a human, or automatically (using a tool)
PS: SQL Views

- A view is just a relation, but we store its definition rather than storing the (materialized) set of tuples.

```
CREATE VIEW YoungActiveStudents (name, grade)
    AS SELECT S.name, E.grade
    FROM Students S, Enrolled E
    WHERE S.sid = E.sid and S.age < 21
```

- Views can be dropped using the DROP VIEW command.
  - How to handle DROP TABLE if there’s a view on the table?
    - DROP TABLE command has options to let the user specify this.

SQL Views and Security

- Views can be used to present necessary information (or a summary) while hiding some details in underlying relation(s).
  - Given YoungStudents, but not Students or Enrolled, we can find students S who have are enrolled, but not the cid’s of the courses they are enrolled in.

- Other view uses in our ER translation context might include:
  - Derived attributes, e.g., age (vs. birthdate)
  - Simplifying/eliminating join paths (for SQL)