Introduction to Data Management

Lecture #6
E-R Relational Mapping (Cont.)

Instructor: Mike Carey
mjcarey@ics.uci.edu

It’s time again for....

Friday Nights with Databases

Brought to you by...

Starbucks in Macau!
Today’s Reminders

❖ Continue to follow the course wiki page

❖ Continue to live by the Piazza page
  ▪ [https://piazza.com/uci/spring2018/cs122a/home](https://piazza.com/uci/spring2018/cs122a/home)

❖ First HW assignment is due today
  ▪ Up to 24 hours to finish with a 20% late penalty

❖ Next HW assignment is available now
  ▪ Translate E-R PHLOG schema into relational form
  ▪ Use our solution schema (out tomorrow at 5pm)

Today’s Reminders (Cont.)

❖ Be careful on what you post in Piazza:
  ❖ how should I model X in the HW?
    ▪ Propose more general questions
    ▪ Use non-HW related examples
    ▪ Avoid asking repeated questions (especially near the deadline!)

❖ Maximum class size expanded to 447
From ISA Hierarchies to Relations

❖ Most general and “clean” approach (recommended):
   • 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.
   • *Ex*: *Hourly_Emps*(ssn, name, lot, hourly_wages, hours_worked)
   • If each employee must be in *one* of the two subclasses...
     (Q: Can we always do this, then?  A: Not w/o redundancy!)

ISA Hierarchy Translation Options

❖ I. “Delta table” approach (recommended):
   • *Emps*(ssn, name, lot)  (All Emps partly reside here)
   • *Hourly_Emps*(ssn, wages, hrs_worked)
   • *Contract_Emps*(ssn, contractid)

❖ II. “Union of tables” approach:
   • *Emps*(ssn, name, lot)
   • *Hourly_Emps*(ssn, name, lot, wages, hrs_worked)
   • *Contract_Emps*(ssn, name, lot, contractid)

❖ III. “Mashup table” approach:
   • *Emps*(kind, ssn, name, lot, wages, hrs_worked, contractid)

Things to consider:
• Expected queries?
• PK/unique constraints?
• Relationships/FKs?
• Overlap constraints?
• Space/time tradeoffs?
ISA Considerations (cont’d.)

❖ Query convenience
  • Ex: List the names of all Emps in lot 12A

❖ PK enforcement
  • Ex: Make sure that ssn is unique for all Emps

❖ Relationship targets
  • Ex: Lawyers table REFERENCES Contract_Emps

❖ Handling of overlap constraints
  • Ex: Sally is under a contract for her hourly work

❖ Space and query performance tradeoffs
  • Ex: List all the info about hourly employee 123
  • Ex: What if most employees are “just plain employees”?

Mapping Advanced ER Features

❖ Multi-valued (vs. single-valued) attributes

❖ Derived (vs. base/stored) attributes

❖ Composite (vs. atomic) attributes
SQL Views (and Security)

❖ 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 used to present needed information while hiding details of underlying table(s).
   ▪ Given YoungStudents (but not Students or Enrolled), we can see (young) students S who have are enrolled but not see the cid’s of their courses.

SQL Views (Cont’d.)

❖ Other view uses in our ER translation context might include:
   ▪ Derived attributes, e.g., age (vs. birthdate)
   ▪ Simplifying/eliminating join paths (for SQL)
   ▪ Beautifying the “Mashup table” approach (to ISA)

CREATE VIEW EmployeeView (ssn, name, bdate, age)
   AS SELECT E.ssn, E.name, E.bdate,
          TIMESTAMPDIFF(YEAR, E.bdate, CURDATE())
          FROM Employees E
Another Mapping Example: Binary vs. Ternary Relationships

❖ 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 all NOT NULL as well – check documentation to see if that’s implicit or explicit!)

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)

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

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

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)

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: Putting The Basics Together

CREATE TABLE Customer (cid, login, cname)

CREATE TABLE Order (oid, shipto, total)

CREATE TABLE Product (pname, color, sku, listprice)

CREATE TABLE LineItem (lno, price, qty)

Database Management Systems 3ed, R. Ramakrishnan and J. Gehrke
**Review: Putting It Together (Cont’d.)**

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

CREATE TABLE Product (
    sku INTEGER,
    pname VARCHAR(100),
    color VARCHAR(20),
    listprice DECIMAL(8,2),
    PRIMARY KEY (sku))

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)

**Reminder: 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)

Relational Database Design

- Two aspects to the RDB design problem:
  - Logical schema design: We just saw one approach, namely, doing E-R modeling followed by an E-R relational schema translation step
  - Physical schema design: Later, once we learn about indexes, when should we utilize them?
- We will look at both problem aspects this term, starting first with relational schema design
  - Our power tools will be functional dependencies (FDs) and normalization theory
  - Note: FDs also play an important role in other contexts as well, e.g., SQL query optimization
So, Given a Relational Schema...

❖ How do I know if my relational schema is a “good” logical database design or not?
   • What might make it “not good”?
   • How can I fix it, if indeed it’s “not good”?
   • How “good” is it, after I’ve fixed it?

❖ Note that your relational schema might have come from one of several places
   • You started from an E-R model (but maybe that model was “wrong” or incomplete in some way?)
   • You went straight to relational in the first place
   • It’s not your schema – you inherited it! 😊

Ex: Wisconsin Sailing Club

Proposed schema design #1:

<table>
<thead>
<tr>
<th>sid</th>
<th>sname</th>
<th>rating</th>
<th>age</th>
<th>date</th>
<th>bid</th>
<th>bname</th>
<th>color</th>
</tr>
</thead>
<tbody>
<tr>
<td>22</td>
<td>Dustin</td>
<td>7</td>
<td>45.0</td>
<td>10/10/98</td>
<td>101</td>
<td>Interlake</td>
<td>blue</td>
</tr>
<tr>
<td>22</td>
<td>Dustin</td>
<td>7</td>
<td>45.0</td>
<td>10/10/98</td>
<td>102</td>
<td>Interlake</td>
<td>red</td>
</tr>
<tr>
<td>22</td>
<td>Dustin</td>
<td>7</td>
<td>45.0</td>
<td>10/8/98</td>
<td>103</td>
<td>Clipper</td>
<td>green</td>
</tr>
<tr>
<td>22</td>
<td>Dustin</td>
<td>7</td>
<td>45.0</td>
<td>10/7/98</td>
<td>104</td>
<td>Marine</td>
<td>red</td>
</tr>
<tr>
<td>31</td>
<td>Lubber</td>
<td>8</td>
<td>55.5</td>
<td>11/10/98</td>
<td>102</td>
<td>Interlake</td>
<td>red</td>
</tr>
<tr>
<td>31</td>
<td>Lubber</td>
<td>8</td>
<td>55.5</td>
<td>11/6/98</td>
<td>103</td>
<td>Clipper</td>
<td>green</td>
</tr>
<tr>
<td>31</td>
<td>Lubber</td>
<td>8</td>
<td>55.5</td>
<td>11/12/98</td>
<td>104</td>
<td>Marine</td>
<td>red</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

Q: Do you think this is a “good” design? (Why or why not?)
Proposed schema design #2:

<table>
<thead>
<tr>
<th>sid</th>
<th>sname</th>
<th>rating</th>
<th>age</th>
</tr>
</thead>
<tbody>
<tr>
<td>22</td>
<td>Dustin</td>
<td>7</td>
<td>45.0</td>
</tr>
<tr>
<td>31</td>
<td>Lubber</td>
<td>8</td>
<td>55.5</td>
</tr>
<tr>
<td></td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>sid</th>
<th>bid</th>
<th>date</th>
</tr>
</thead>
<tbody>
<tr>
<td>22</td>
<td>101</td>
<td>10/10/98</td>
</tr>
<tr>
<td>22</td>
<td>102</td>
<td>10/10/98</td>
</tr>
<tr>
<td>22</td>
<td>103</td>
<td>10/8/98</td>
</tr>
<tr>
<td>22</td>
<td>104</td>
<td>10/7/98</td>
</tr>
<tr>
<td>31</td>
<td>102</td>
<td>11/10/98</td>
</tr>
<tr>
<td>31</td>
<td>103</td>
<td>11/6/98</td>
</tr>
<tr>
<td>31</td>
<td>104</td>
<td>11/12/98</td>
</tr>
<tr>
<td></td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

Q: What about this design?
• Is #2 “better than #1...? Explain!
• Is it a “best” design?
• How can we go from design #1 to this one?

Proposed schema design #3:

<table>
<thead>
<tr>
<th>sid</th>
<th>sname</th>
<th>rating</th>
<th>age</th>
</tr>
</thead>
<tbody>
<tr>
<td>22</td>
<td>Dustin</td>
<td>7</td>
<td>45.0</td>
</tr>
<tr>
<td>31</td>
<td>Lubber</td>
<td>8</td>
<td>55.5</td>
</tr>
<tr>
<td></td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>sid</th>
<th>bid</th>
<th>date</th>
</tr>
</thead>
<tbody>
<tr>
<td>22</td>
<td>101</td>
<td>10/10/98</td>
</tr>
<tr>
<td>22</td>
<td>102</td>
<td>10/10/98</td>
</tr>
<tr>
<td>22</td>
<td>103</td>
<td>10/8/98</td>
</tr>
<tr>
<td>22</td>
<td>104</td>
<td>10/7/98</td>
</tr>
<tr>
<td>31</td>
<td>102</td>
<td>11/10/98</td>
</tr>
<tr>
<td>31</td>
<td>103</td>
<td>11/6/98</td>
</tr>
<tr>
<td>31</td>
<td>104</td>
<td>11/12/98</td>
</tr>
<tr>
<td></td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

Q: What about this design?
• Is #3 “better” or “worse” than #2...?
• What sort of tradeoffs do you see between the two?
The Evils of Redundancy  
(or: The Evils of Redundancy)

- **Redundancy** is at the root of several problems associated with relational schemas:
  - Redundant storage (space)
  - Insert/delete/update anomalies

- Functional dependencies can help in identifying problem schemas and suggesting refinements.

- Main refinement technique: decomposition, e.g., replace R(ABCD) with R1(AB) + R2(BCD).

- Decomposition should be used judiciously:
  - Is there reason to decompose a relation?
  - Does the decomposition cause any problems?

*Good rule to follow: “One fact, one place!”*