Introduction to Data Management

Lecture #4
E-R Model, Still Going…

Instructor: Mike Carey
mjcarey@ics.uci.edu

Today’s Reminders

- Continue to follow the course wiki page
  - http://www.ics.uci.edu/~cs122a/
- Also follow (and live by) the Piazza page
  - https://piazza.com/uci/spring2018/cs122a/home
  - 50 or so of you are missing out at the moment…!
- The first HW assignment is underway
  - Conceptual (E-R) database design for PEEEza
- A waiting list (size progress) update
  - TBD (hopefully more news by lecture time today)
- This week’s quiz will be on E-R modeling
  - Also bring any lingering HW #1 related questions
Conceptual Design Using the ER Model

- **Design choices:**
  - Should a given concept be modeled as an entity or as an attribute?
  - Should a given concept be modeled as an entity or as a relationship?
  - Characterizing relationships: Binary or ternary? Aggregation? …

- **Constraints in the ER Model:**
  - A lot of data semantics can (and should) be captured
  - But, not all constraints cannot be captured by ER diagrams. *(Ex: Department heads from earlier…!)*

Advanced Attribute Considerations

- Should *address* be an attribute of Employees or an entity (connected to Employees by a relationship)?

- Depends on how we want to use addresses, on the data semantics, and what model features we use:
  - If we have several addresses per employee, *address* would be a multivalued attribute or an entity if we stick only to basic E-R concepts (as they can’t be set-valued w/o advanced modeling).
  - If address structure (city, street, etc.) is important, e.g., to query for employees in a given city, *address* should be modeled as a composite attribute or an entity (as attribute values are atomic otherwise) – i.e., it shouldn’t just be an address string.
  - *If the address itself is logically separate (e.g., representing a property that’s located there) and refer-able, it’s rightly an entity in any case!*
Attribute Considerations (Cont’d.)

- Works_In here does not allow an employee to work in a department for two or more periods. (Q: Why...?)

- Similar to the problem of wanting to record several addresses for an employee: We want to record several values of the descriptive attributes for each instance of this “relationship”. Could be accomplished by having a multivalued relationship attribute:

Entity vs. Relationship

- ER diagram on the right is okay if a manager gets a separate discretionary budget (dbudget) per dept.
- What if a manager gets a discretionary budget that covers all managed depts?
  - Redundancy: dbudget could be stored (repeated) with each dept managed by the manager.
  - Misleading: Suggests dbudget is associated with department-mgr combination.

Also note ISA and the relationship...
Binary vs. Ternary Relationships

- If each policy is owned by just 1 employee, with each dependent tied to their one covering policy, first diagram is inaccurate.
- Q: What are the additional constraints in the 2nd diagram? (And what else was wrong with the 1st diagram? 😊)

Better design

Binary vs. Ternary Relationships (Cont’d.)

- Previous example illustrated a case when two binary relationships were “better” than one ternary relationship.
- Now an example in the other direction: a ternary relation Contracts relates the entity sets Parts, Departments and Suppliers, and has descriptive attribute qty. No combination of binary relationships would be an adequate substitute:
  - S “can-supply” P, D “needs” P, and D “deals-with” S would not imply that D has agreed to buy P from S.
  - And also, how/where else would we record qty?
Our example where ternary is needed: a ternary relation **Contracts** relates the entity sets **Parts**, **Departments** and **Suppliers**, and has descriptive attribute *qty*:

**NOTE:** Some might argue that **Contracts** should actually be an entity set…

Observe: **Prescriptions** was similar.

What the entity set perspective would lead to: an entity set **Contracts** related to the entity sets **Parts**, **Departments** and **Suppliers**, with the descriptive attribute *qty*:

**NOTE:** Some might argue that **Contracts** should actually be an entity set, so…

(This is also a workaround if your modeling tool is limited to binary relationships.)
**Database Design Process (Flow)**

- Requirements Gathering (interviews)
- Conceptual Design (using ER)
- Platform Choice (which DBMS?)
- Logical Design (for target data model)
- Physical Design (for target DBMS, workload)
- Implement (and test, of course 😊)

(Expect backtracking, iteration, and also incremental adjustments – and, we will actually be giving you a bit of practice with that last one in the next few HW assignments... 😊)

**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).
- Additionally: weak entities, ISA hierarchies, aggregation, and multi-valued, composite and/or derived attributes.
- Note: Many variations on the ER model (and many notations in use as well) – and also, UML...
Summary of ER (Cont’d.)

- Several kinds of integrity constraints can be expressed in the ER model: *cardinality constraints, participation constraints, also overlap/covering constraints* for ISA hierarchies. Some *foreign key constraints* are also implicit in the definition of a relationship set (more about key constraints will be coming soon).
  - Some constraints (notably, *functional dependencies*) cannot be expressed in the ER model.
  - Constraints play an important role in determining the best database design for an enterprise.

Summary of ER (Cont’d.)

- ER design is *subjective*. There are often many ways to model a given scenario! Analyzing alternatives can be tricky, especially for a large enterprise. Common choices include:
  - Entity vs. attribute, entity vs. relationship, binary or n-ary relationship, whether or not to use an ISA hierarchy, and whether or not to use aggregation.

- Ensuring good database design: The resulting relational schema should be analyzed and refined further. For this, FD information and normalization techniques are especially useful (coming soon).
Relational Database: Definitions

- **Relational database**: a set of relations
- **Relation**: consists of 2 parts:
  - **Instance**: a table, with rows and columns.
    - #Rows = cardinality, #fields = degree or arity.
  - **Schema**: specifies name of relation, plus name and type of each column.
- Can think of a relation as a set of rows or tuples (i.e., all rows are distinct) in the pure relational model (vs. reality of SQL 😁).

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
Relational Query Languages

- A major strength of the relational model: supports simple, powerful querying of data.
- Queries can be written intuitively, and the DBMS is responsible for efficient evaluation.
  - The key: precise (and set-based) semantics for relational queries.
  - Allows the optimizer to extensively re-order operations, and still ensure that the answer does not change.

The SQL Query Language (Preview)

- Developed by IBM (System R) in the 1970s
- Need for a standard, since it is used by many vendors (Oracle, IBM, Microsoft, …)
- ANSI/ISO Standards:
  - SQL-86
  - SQL-89 (minor revision)
  - SQL-92 (major revision, very widely supported)
  - SQL-99 (major extensions, current standard)
The SQL Query Language (Preview)

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

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

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

```sql
SELECT S.name, S.login
```

### Querying Multiple Relations

- What does the following query compute?

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

Given the following instances of Students and 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>

We will 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>