Introduction to Data Management

Lecture #3
(E-R Design, Cont’d.)

Instructor: Mike Carey
mjcarey@ics.uci.edu

Announcements

- Reminders:
  - Sign up on Piazza! (About 10% still haven’t done this...)
  - First real quiz in discussions this week! (Study! 😊)
- HW #1 is in flight
  - Our startup: “TrueOnlyUSNews.com”
  - Developing your E-R models...
- Today: More on Conceptual DB Design
  - Advanced E-R features
  - Intro to the Relational model
- Any lingering Q’s from last time?
**ISA ("is a") Hierarchies**

- As in Java or other PLs, ER attributes are inherited (including the key attribute).
- If we declare A ISA B, every A entity is also considered to be a B entity.
  - **Covering constraints**: Must every Employees entity be either an Hourly_Emps or a Contract_Emps entity? (Yes or no)
    - Ex: Hourly_Emps AND Contract_Emps COVER Employees (pick 1 of 2 vs. 1 of 3)
  - **Overlap constraints**: Can some Employees entity be an Hourly_Emps as well as a Contract_Emps entity? (Allowed or disallowed)
    - Ex: Hourly_Emps OVERLAPS Contract_Emps (else pick 1 of the 3 types)
- Reasons for using ISA:
  - To add descriptive attributes specific to a subclass.
  - To identify subclasses that participate in a relationship.
- Design: specialization (top-down), generalization (bottom-up)

---

**Aggregation**

- Used when we have to model a relationship involving (entity sets and) a relationship set.
  - **Aggregation** allows us to treat a relationship set as an entity set for purposes of participating in (other) relationships.
- **Aggregation vs. ternary relationship**:
  - Monitors is a distinct relationship; even has its own attribute here.
  - Each sponsorship can monitored by zero or more employees (as above).
### Additional Advanced ER Features

- **Multi-valued (vs. single-valued) attributes**
  
  - `Employees`
  - `ssn`
  - `name`
  - `phone`

- **Derived (vs. base/stored) attributes**
  
  - `Employees`
  - `ssn`
  - `name`
  - `bdate`
  - `age`

- **Composite (vs. atomic) attributes**
  
  - `Employees`
  - `ssn`
  - `name`
  - `address`
    - `snum`
    - `street`
    - `city`
    - `zip`

**NOTE:** Can model (two of) these using additional entity and relationship types.

### Conceptual Design Using the ER Model

- **Design choices:**
  - Should a given concept be modeled as an entity or an attribute?
  - Should a given concept be modeled as an entity or 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…!)
Entity vs. Attribute

- Should *address* be an attribute of Employees or an entity (connected to Employees by a relationship)?
- Depends how we want to use address information, the data semantics, and also the model features:
  - If we have several addresses per employee, *address* must be an entity if we stick only to basic E-R concepts (as attributes cannot be set-valued w/o advanced modeling goodies).
  - If the structure (city, street, etc.) is important, e.g., we want to retrieve employees in a given city, *address* must be modeled as an entity (since attribute values are atomic) w/o advanced modeling goodies
  - *If the address itself is logically separate (e.g., the property that’s located there) and refer-able, it’s rightly an entity in any case!*

Entity vs. Attribute (Cont’d.)

- Works_In4 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**

- First ER diagram OK if a manager gets a separate discretionary budget for each dept.
- What if a manager gets a discretionary budget that covers all managed depts?
  - **Redundancy:** `dbudget` stored for each dept managed by manager.
  - **Misleading:** Suggests `dbudget` is associated with department-mgr combination.

**Binary vs. Ternary Relationships**

- If each policy is owned by just 1 employee, with each dependent tied to their 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**
  - **Bad design**

---

*Database Management Systems 3ed, R. Ramakrishnan and J. Gehrke*
Previous example illustrated a case when two binary relationships were “better” than one ternary relationship.

An example in the other direction: a ternary relation \textit{Contracts} relates entity sets \textit{Parts}, \textit{Departments} and \textit{Suppliers}, and has descriptive attribute \textit{qty}. No combination of binary relationships is an adequate substitute:

- S “can-supply” P, D “needs” P, and D “deals-with” S does not imply that D has agreed to buy P from S.
- And also, how/where else would we record \textit{qty}?

Our example in the other direction: a ternary relation \textit{Contracts} relates entity sets \textit{Parts}, \textit{Departments} and \textit{Suppliers}, and has descriptive attribute \textit{qty}:

\begin{itemize}
  \item \textit{Parts} \quad \textit{Suppliers} \quad \textit{Contracts}
  \item \textit{dno} \quad \textit{dname} \quad \textit{phone} \quad \textit{qty} \quad \textit{sno} \quad \textit{surname} \quad \textit{phone}
  \item \textit{pno} \quad \textit{pname} \quad \textit{cost}
\end{itemize}

\textbf{(Observe: Prescriptions was similar)}
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...
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.

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:

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

<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@ee</td>
<td>18</td>
<td>3.2</td>
</tr>
</tbody>
</table>

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

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

---

Querying Multiple Relations

- What does the following query compute?

```
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@ee</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>
<tr>
<td>53831</td>
<td>Carnatic101</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>53831</td>
<td>Reggae203</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>53650</td>
<td>Topology112</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>53666</td>
<td>History105</td>
<td></td>
<td></td>
<td></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>
Creating Relations in SQL

- Create the Students relation. Observe that the type (domain) of each field is specified and enforced by the DBMS whenever tuples are added or modified.
- As another example, the Enrolled table holds information about courses that students take.

```
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))
```

Destroying and Altering Relations

- Destroys the relation Students. The schema information and the tuples are deleted.

```
DROP TABLE Students
```

- The schema of Students is altered by adding a new field; every tuple in the current instance is extended with a null value in the new field.

```
ALTER TABLE Students
  ADD COLUMN firstYear integer
```

```
DROP TABLE Students
```

```
CREATE TABLE Students
  ADD COLUMN firstYear integer
```
Adding and Deleting Tuples

- Can insert a single tuple using:

  ```sql
  INSERT INTO Students (sid, name, login, age, gpa)
  VALUES (53688, 'Smith', 'smith@ee', 18, 3.2)
  ```

- Can delete all tuples satisfying some condition (e.g., name = Smith):

  ```sql
  DELETE
  FROM Students S
  WHERE S.name = 'Smith'
  ```

> Powerful variants of these commands are available; more later!

Integrity Constraints (ICs)

- **IC**: condition that must be true for any instance of the database; e.g., domain constraints.
  - ICs are specified when schema is defined.
  - ICs are checked when relations are modified.
- A **legal** instance of a relation is one that satisfies all specified ICs.
  - DBMS should not allow illegal instances.
- If the DBMS checks ICs, stored data is more faithful to real-world meaning.
  - Avoids data entry errors (centrally), too!
Primary Key Constraints

- A set of fields is a **key** for a relation if:
  1. No two distinct tuples can have same values in all key fields, and
  2. This is not true for any subset of the key.
     - Part 2 false? In that case, this is a "superkey".
     - If there’s > 1 key for a relation, one of the keys is chosen (by DBA) to be the **primary key**.
     - The others are referred to as **candidate keys**.
- E.g., sid is a key for Students. (What about name?) The set \{sid, gpa\} is a superkey.

Primary and Candidate Keys in SQL

- Possibly many **candidate keys** (specified using UNIQUE), with one chosen as the **primary key**.

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

CREATE TABLE Enrolled
(sid CHAR(20),
cid CHAR(20),
grade CHAR(2),
PRIMARY KEY (sid) )
Foreign Keys, Referential Integrity

- **Foreign key**: Set of fields in one relation used to “refer” to a tuple in another relation. (Must refer to the primary key of the other relation.) Like a “logical pointer”.

- E.g., *sid* is a foreign key referring to *Students*:
  - Enrolled(*sid*: string, *cid*: string, *grade*: string)
  - If all foreign key constraints are enforced, referential integrity is achieved, i.e., no dangling references.

Foreign Keys in SQL

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

```sql
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>sid</th>
<th>cid</th>
<th>grade</th>
</tr>
</thead>
<tbody>
<tr>
<td>53666</td>
<td>Carnatic101</td>
<td>C</td>
</tr>
<tr>
<td>53666</td>
<td>Reggae203</td>
<td>B</td>
</tr>
<tr>
<td>53650</td>
<td>Topology112</td>
<td>A</td>
</tr>
<tr>
<td>53666</td>
<td>History105</td>
<td>B</td>
</tr>
</tbody>
</table>

<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>

Database Management Systems 3ed, R. Ramakrishnan and J. Gehrke

Database Management Systems 3ed, R. Ramakrishnan and J. Gehrke