Introduction to Data Management

Lecture #5
Relational Model (Cont.)
& E-R Relational Mapping

Instructor: Mike Carey
mjcarey@ics.uci.edu

Today’s Reminders

❖ Continue to follow the course wiki page
  • http://www.ics.uci.edu/~cs122a/
  • Lecture notes live in the Attachments section (at the bottom)

❖ Also follow (and live by) the Piazza page
  • https://piazza.com/uci/spring2018/cs122a/home
  • 14 of you are still missing out...! (Living dangerously 😊)

❖ The first HW assignment is due Friday
  • Conceptual (E-R) database design for PHLOG

❖ Maximum class size expansion is in progress
  • Hear more about it in the next few days
  • Keep working on your assignments and attending discussions even if you are waitlisted...
### 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 being chosen as the primary key.
- Used carelessly, an IC can prevent the storage of database instances that arise in practice!
- “For a given student + course, there is a single grade.” vs.
  “Students can take only one course, and receive a single grade for that course; further, no two students in a course may ever receive the same grade.”

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),
UNIQUE (cid, grade))

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.

  ```
  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>

Enforcing Referential Integrity

- Consider Students and Enrolled; *sid* in Enrolled is a foreign key that references Students.
- 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?
  - Also delete all Enrolled tuples that refer to it. Or...
  - Disallow deletion of a Students tuple if it is referred to.
  - Set sid in Enrolled tuples that refer to it to a *default sid*.
  - (In SQL, also: Set sid in Enrolled tuples that refer to it 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 support all 4 options on deletes and updates.
  - Default is NO ACTION (delete/update is rejected)
  - CASCADE (also delete all tuples that refer to the being-deleted tuple)
  - SET NULL / SET DEFAULT (sets foreign key value of the referring tuples)

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?

- ICs are based upon the semantics of the real-world enterprise that is being described in the database relations (perhaps via an E-R schema)
- We can check a database instance to see if an IC is violated, but we can NEVER infer that an IC is true by looking at an instance.
  - An IC is a statement about all possible instances!
  - From example, we know name is not a key, but the assertion that sid is a key is given to us.

- Key and foreign key ICs are the most common; more general ICs supported too.
### Logical DB Design: ER to Relational

- **Entity sets to tables:**

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

### Relationship Sets to Tables

- **In translating a relationship set to a relation, attributes of the relation must include:**
  - Keys for each participating entity set (as foreign keys).
  - This set of attributes forms a superkey for the relation.
  - All descriptive attributes.

  ```sql
  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)
  ```
Key Constraints (Review)

- Each dept has at most one manager, according to the key constraint on Manages.

Translation to relational model?

Translating ER Diagrams with Key Constraints

- Map the relationship to a table (Manages):
  - Note that did (alone) is the key!
  - Still separate tables for Employees and Departments.
- But, since each department has a unique manager, we could choose to fold Manages right into Departments.
  
  (Q: Why do that...?)

CREATE TABLE Manages (ssn CHAR(11), did INTEGER, since DATE, PRIMARY KEY (did), FOREIGN KEY (ssn) REFERENCES Employees, FOREIGN KEY (did) REFERENCES Departments)

CREATE TABLE Departments2 (did INTEGER, dname CHAR(20), budget REAL, mgr_ssn CHAR(11), mgr_since DATE, PRIMARY KEY (did), FOREIGN KEY (mgr_ssn) REFERENCES Employees)

Note: The relationship info has been pushed to the N-side’s entity table!
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).

```
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)
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...

(If we can 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)