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
  - 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](https://piazza.com/uci/spring2018/cs122a/home)
  - 26 of you are still missing out…! (Living dangerously 😃)
- The first HW assignment is due Friday
  - Conceptual (E-R) database design for PEEaza
- A waiting list (size progress) update
  - New room, all impacts now done w.r.t. discussions
  - Newbies have hopefully been coming/reading/working on CS122a stuff in the meantime (💬) – clear your calendars if not…
Querying Multiple Relations (Review)

- 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@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>

<table>
<thead>
<tr>
<th>sid</th>
<th>cid</th>
<th>grade</th>
</tr>
</thead>
<tbody>
<tr>
<td>53831</td>
<td>Carnatic101</td>
<td>C</td>
</tr>
<tr>
<td>53831</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>

We will get:

- S.name
- E.cid

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.
  ```
  CREATE TABLE Students
  (sid CHAR(20),
   name VARCHAR(50),
   login CHAR(10),
   age INTEGER,
   gpa REAL)
  ```

- As another example, the Enrolled table holds information about courses that students take.
  ```
  CREATE TABLE Enrolled
  (sid CHAR(20),
   cid CHAR(20),
   grade CHAR(2))
  ```
Destroying and Altering Relations

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

- ALTER TABLE Students
  - ADD COLUMN firstYear integer
  - 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.

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 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.”

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

```sql
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.

```sql
CREATE TABLE Enrolled
(sid CHAR(20), cid CHAR(20), grade CHAR(2),
 PRIMARY KEY (sid, cid),
 FOREIGN KEY (sid) REFERENCES Students )
```

---

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)*

```sql
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:**

  \[
  \text{CREATE TABLE Employees} \\
  (\text{ssn}\ \text{CHAR}(11), \\
  \text{name}\ \text{CHAR}(20), \\
  \text{lot}\ \text{INTEGER}, \\
  \text{PRIMARY KEY}\ \text{(ssn)})
  \]

**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.

  \[
  \text{CREATE TABLE Works_In(} \\
  \text{ssn}\ \text{CHAR}(11), \\
  \text{did}\ \text{INTEGER}, \\
  \text{since}\ \text{DATE}, \\
  \text{PRIMARY KEY (ssn, did),} \\
  \text{FOREIGN KEY (ssn)} \\
  \text{REFERENCES Employees,} \\
  \text{FOREIGN KEY (did)} \\
  \text{REFERENCES Departments)}
  \]
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
)
```

vs.

```
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?  \(\rightarrow\) FooBar\((\text{fooId}, \text{barId}, \text{baz})\)
  - FooBar is N:1?  \(\rightarrow\) FooBar\((\text{fooId}, \text{barId}, \text{baz})\)
  - FooBar is 1:N?  \(\rightarrow\) FooBar\((\text{fooId}, \text{barId}, \text{baz})\)
  - FooBar is 1:1?  \(\rightarrow\) FooBar\((\text{fooId}, \text{barId}, \text{baz})\)  \(\text{(Note: unique)}\)