Database Applications (15-415)

The Relational Model
Lecture 3, January 17, 2016

Mohammad Hammoud
Today...

- **Last Session:**
  - The entity relationship (ER) model

- **Today’s Session:**
  - ER model (Cont’d): conceptual design choices
  - The relational model
    - Basic Constructs of the relational model
    - Basic SQL

- **Announcement:**
  - PS1 is due on Thursday, Jan 21, 2016 by midnight
Outline

ER Model: Conceptual Design Choices and Summary

The Relational Model: Introduction

The Relational Model: Basic SQL
Conceptual Design Choices

- Should a concept be modeled as an *entity* or an *attribute*?

- Should a concept be modeled as an *entity* or a *relationship*?

- How should we identify relationships?
  - Binary or ternary?
  - Ternary or aggregation?

- Constraints in the ER Model:
  - A lot of data semantics can (and should) be captured
  - But some constraints cannot be captured in ER diagrams
Entity vs. Attribute

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

- This depends upon the use we want to make of address information, and the semantics of the data
  - If we have several addresses per an employee, *address* must be an entity (since attributes cannot be set-valued)
  - 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
Consider the following ER diagram:

- **A problem**: Works_In4 does not allow an employee to work in a department for two or more periods.

- **Solution**: Introduce "Duration" as a new entity set.
Entity vs. Relationship

- Consider the following ER diagram whereby a manager gets a separate discretionary budget for each department:

  - What if a manager gets a discretionary budget that covers all managed departments?
    - Redundant data
    - Misleading
Binary vs. Ternary Relationships

If each policy is owned by just 1 employee:
Binary vs. Ternary Relationships

If each policy is owned by just 1 employee:

```
<table>
<thead>
<tr>
<th>Policyid</th>
<th>Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

Key constraint on Policies would mean policy can only cover 1 dependent!

Bad design!
Binary vs. Ternary Relationships

If each policy is owned by just 1 employee:

Better design!
Binary vs. Ternary Relationships

- But sometimes ternary relationships cannot be replaced by a set of binary relationships.
Binary vs. Ternary Relationships

- But sometimes ternary relationships cannot be replaced by a set of binary relationships
Binary vs. Ternary Relationships

- But sometimes ternary relationships cannot be replaced by a set of binary relationships.

Diagram:
- Parts
  - Contract
    - qty
    - Suppliers
- Departments
  - VS.
  - can-supply
  - Suppliers
  - deals-with
  - Parts
  - Departments
Binary vs. Ternary Relationships

- But sometimes ternary relationships cannot be replaced by a set of binary relationships

Why is it bad?
Binary vs. Ternary Relationships

- But sometimes ternary relationships cannot be replaced by a set of binary relationships

- \( S \) “can-supply” \( P \), \( D \) “needs” \( P \), and \( D \) “deals-with” \( S \) do not imply that \( D \) has agreed to buy \( P \) from \( S \)

- How do we record \( qty \)?
Aggregation

- Aggregation allows indicating that a relationship set (identified through a *dashed box*) participates in another relationship set.
Ternary vs. Aggregation Relationships

- When to use aggregation?
  - If we want to attach a relationship to a relationship

- What if we do not want to record the *until* attribute of Monitors relationship?
Ternary vs. Aggregation Relationships (Cont’d)

- We might reasonably use a ternary relationship instead of an aggregation

What if each sponsorship (of a project by a department) is to be monitored by at most one employee?
ER Model: Summary

- Conceptual design follows requirements analysis
  - Yields a high-level description of data to be stored

- The ER model is popular for conceptual design
  - Its constructs are expressive, close to the way people think about their applications

- The basic constructs of the ER model are:
  - Entities, relationships, and attributes (of entities and relationships)
ER Model: Summary

- Some additional constructs of the ER model are:
  - Weak entities, ISA hierarchies, and aggregation

- Several kinds of integrity constraints can be expressed in the ER model
  - Key constraints, participation constraints, and overlap/covering constraints for ISA hierarchies

- Note: there are many variations on the ER model
ER Model: Summary

- 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 (e.g., ternary)
  - Whether or not to use ISA hierarchies
  - Whether or not to use aggregation
Outline

ER Model: Conceptual Design Choices and Summary

The Relational Model: Introduction

The Relational Model: Basic SQL
Why Studying the Relational Model?

- Most widely used model
  - Vendors: IBM/Informix, Microsoft, Oracle, Sybase, etc.

- “Legacy systems” in older models
  - E.g., IBM’s IMS

- Object-Oriented concepts have merged into
  - An object-relational model
    - Informix->IBM DB2, Oracle 8i
What is the Relational Model?

- The relational model adopts a “tabular” representation
  - A database is a *collection* of one or more *relations*
  - Each relation is a *table* with rows and columns

- What is unique about the relational model as opposed to older data models?
  - Its simple data representation
  - Ease with which complex queries can be expressed
Basic Constructs

- The main construct in the relational model is the *relation*

- A relation consists of:
  1. A *schema* which includes:
     - The relation’s name
     - The name of each column
     - The *domain* of each column
  2. An *instance* which is a set of tuples
    - Each tuple has the same number of columns as the relation schema
The Domain Constraints

- A relation schema specifies the *domain* of each column which entails domain constraints

- A domain constraint specifies a condition by which each instance of a relation should satisfy
  - The values that appear in a column must be drawn from the domain associated with that column

- Who defines a domain constraint?
  - DBA

- Who enforces a domain constraint?
  - DBMS
More Details on the Relational Model

Degree (or arity) = # of fields

<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 = # of tuples

An instance of the “Students” relation

- What is the relational database schema (not the relation schema)?
  - A collection of schemas for the relations in the database

- What is the instance of a relational database (not the instance of a relation)?
  - A collection of relation instances
Outline

- ER Model: Conceptual Design Choices and Summary
- The Relational Model: Introduction
- The Relational Model: Basic SQL
SQL - A Language for Relational DBs

- **SQL** (a.k.a. “Sequel”) stands for *Structured Query Language*

- SQL was developed by IBM (system R) in the 1970s

- There is a need for a standard since SQL is used by many vendors

- Standards:
  - SQL-86
  - SQL-89 (minor revision)
  - SQL-92 (major revision)
  - SQL-99 (major extensions)
  - SQL-2003 (minor revision)
  - SQL-2011
DDL and DML

- The SQL language has two main aspects (there are other aspects which we discuss next week)
  - Data Definition Language (DDL)
    - Allows creating, modifying, and deleting relations and views
    - Allows specifying constraints
    - Allows administering users, security, etc.
  - Data Manipulation Language (DML)
    - Allows posing queries to find tuples that satisfy criteria
    - Allows adding, modifying, and removing tuples
Creating Relations in SQL

- S1 can be used to create the “Students” relation

- S2 can be used to create the “Enrolled” relation

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))

S1

S2

The DBMS enforces domain constraints whenever tuples are added or modified
Adding and Deleting Tuples

- We can insert a single tuple to the “Students” relation using:

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

- We can delete all tuples from the “Students” relation which satisfy some condition (e.g., name = Smith):

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

**Powerful variants of these commands are available; more next week!**
Querying a Relation

- How can we find all 18-year old students?

<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@eeecs</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>

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

- How can we find just names and logins?

```sql
SELECT S.name, S.login
FROM Students S
WHERE S.age=18
```
Querying Multiple Relations

- What does the following query compute assuming S and E?

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

We get:

<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>

<table>
<thead>
<tr>
<th>S.name</th>
<th>E.cid</th>
</tr>
</thead>
<tbody>
<tr>
<td>Smith</td>
<td>Topology112</td>
</tr>
</tbody>
</table>
Destroying and Altering Relations

- How to destroy the relation "Students"?
  
  ```
  DROP TABLE Students
  ```

  The schema information *and* the tuples are deleted

- How to alter the schema of "Students" in order to add a new field?
  
  ```
  ALTER TABLE Students
  ADD COLUMN firstYear: integer
  ```

  Every tuple in the current instance is extended with a *null* value in the new field!
Integrity Constraints (ICs)

- An IC is a condition that must be true for any instance of the database (e.g., domain constraints)
  - ICs are specified when schemas are 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, too!
Keys

- Keys help associate tuples in different relations

- Keys are one form of integrity constraints (ICs)

<table>
<thead>
<tr>
<th>Enrolled</th>
<th>Students</th>
</tr>
</thead>
<tbody>
<tr>
<td>sid</td>
<td>name</td>
</tr>
<tr>
<td>53666</td>
<td>Jones</td>
</tr>
<tr>
<td>53666</td>
<td>Smith</td>
</tr>
<tr>
<td>53650</td>
<td>Smith</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>sid</th>
<th>cid</th>
<th>grade</th>
</tr>
</thead>
<tbody>
<tr>
<td>53666</td>
<td>15-101</td>
<td>C</td>
</tr>
<tr>
<td>53666</td>
<td>18-203</td>
<td>B</td>
</tr>
<tr>
<td>53650</td>
<td>15-112</td>
<td>A</td>
</tr>
<tr>
<td>53666</td>
<td>15-105</td>
<td>B</td>
</tr>
</tbody>
</table>
Keys

- Keys help associate tuples in different relations
- Keys are one form of integrity constraints (ICs)

<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@cs</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>53666</td>
<td>15-101</td>
<td>C</td>
</tr>
<tr>
<td>53666</td>
<td>18-203</td>
<td>B</td>
</tr>
<tr>
<td>53650</td>
<td>15-112</td>
<td>A</td>
</tr>
<tr>
<td>53666</td>
<td>15-105</td>
<td>B</td>
</tr>
</tbody>
</table>
Superkey, Primary and Candidate Keys

- A set of fields is a *superkey* if:
  - No two distinct tuples can have same values in *all* key fields

- A set of fields is a *primary key* for a relation if:
  - It is a *minimal* superkey

- What if there is more than one key for a relation?
  - One of the keys is chosen (by DBA) to be the primary key
  - Other keys are called *candidate keys*

- Examples:
  - *sid* is a key for Students (what about *name*?)
  - The set \{sid, name\} is a superkey (or a set of fields that contains a key)
Primary and Candidate Keys in SQL

- Many candidate keys (specified using `UNIQUE`) can be designated and one is chosen as a *primary key*.

- Keys must be used carefully!

- “For a given student and course, there is a single grade”
Primary and Candidate Keys in SQL

- Many candidate keys (specified using `UNIQUE`) can be designated and one is chosen as a primary key

- Keys must be used carefully!

- “For a given student and course, there is a single grade”

**CREATE TABLE Enrolled**

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

**vs.**

```sql
CREATE TABLE Enrolled
    (sid CHAR(20)
     cid CHAR(20),
     grade CHAR(2),
     PRIMARY KEY (sid),
     UNIQUE (cid, grade))
```
Primary and Candidate Keys in SQL

- Many candidate keys (specified using `UNIQUE`) can be designated and one is chosen as a primary key.
- Keys must be used carefully!
- “For a given student and course, there is a single 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))
```

Q: What does this mean?
Primary and Candidate Keys in SQL

- Many candidate keys (specified using `UNIQUE`) can be designated and one is chosen as a primary key

- Keys must be used carefully!

- “For a given student and course, there is a single grade”

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

vs.

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

“A student can take only one course, and no two students in a course receive the same grade”
Foreign Keys and Referential Integrity

- A foreign key is a set of fields referring to a tuple in another relation
  - It must correspond to the primary key of the other relation
  - It acts like a `logical pointer`

- If all foreign key constraints are enforced, referential integrity is said to be achieved (i.e., no dangling references)
Foreign Keys in SQL

- Example: Only existing students may enroll for courses
  - `sid` is a foreign key referring to Students
  - How can we write this in SQL?

<table>
<thead>
<tr>
<th>Enrolled</th>
<th>Students</th>
</tr>
</thead>
<tbody>
<tr>
<td>sid</td>
<td>cid</td>
</tr>
<tr>
<td>53666</td>
<td>15-101</td>
</tr>
<tr>
<td>53666</td>
<td>18-203</td>
</tr>
<tr>
<td>53650</td>
<td>15-112</td>
</tr>
<tr>
<td>53666</td>
<td>15-105</td>
</tr>
</tbody>
</table>
Foreign Keys in SQL

- Example: Only existing students may 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>Enrolled</th>
<th>Students</th>
</tr>
</thead>
<tbody>
<tr>
<td>sid</td>
<td>cid</td>
</tr>
<tr>
<td>53666</td>
<td>15-101</td>
</tr>
<tr>
<td>53666</td>
<td>18-203</td>
</tr>
<tr>
<td>53650</td>
<td>15-112</td>
</tr>
<tr>
<td>53666</td>
<td>15-105</td>
</tr>
</tbody>
</table>

Example: Only existing students may enroll for courses
Enforcing Referential Integrity

- 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?
  - Disallow its deletion
  - Delete all Enrolled tuples that refer to it
  - Set sid in Enrolled tuples that refer to it to a *default sid*
  - Set sid in Enrolled tuples that refer to it to a special value *null*, denoting ‘unknown’ or ‘inapplicable’

- What if a “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** (i.e., delete/update is rejected)
  - **CASCADE** (also delete all tuples that refer to the deleted tuple)
  - **SET NULL / SET DEFAULT** (sets foreign key value of referencing tuple)

```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
  )
```

What does this mean?
Where do ICs Come From?

- ICs are based upon the semantics of the real-world enterprise that is being described in the database relations.

- 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 the “Students” relation, 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 are supported too.
Views

- A **view** is a table whose rows are not explicitly stored but computed as needed

```sql
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 queried
  - Querying `YoungActiveStudents` would necessitate computing it first then applying the query on the result as being like any other relation

- Views can be dropped using the `DROP VIEW` command
  - How to handle `DROP TABLE` if there’s a view on the table?
    - `DROP TABLE` command has options to let the user specify this
Views and Security

- Views can be used to present necessary information, while hiding details in underlying relation(s)
  - If the schema of an old relation is *changed*, a view can be defined to represent the old schema
    - This allows applications to *transparently* assume the old schema

- Views can be defined to give a group of users access to just the information they are allowed to see
  - E.g., we can define a view that allows students to see other students’ names and ages, but not GPAs (also students can be prevented from accessing the underlying “Students” relation)
Views and Security

- Views can be used to present necessary information, while hiding details in underlying relation(s)
  - If the schema of an old relation is changed, a view can be defined to represent the old schema
    - This allows applications to transparently assume the old schema

- Logical Data Independence!

- Views can be defined to give a group of users access to just the information they are allowed to see
  - E.g., we can define a view that allows students to see other students’ names and ages, but not GPAs (also students can be prevented from accessing the underlying “Students” relation)
Views and Security

- Views can be used to present necessary information, while hiding details in underlying relation(s)
  - If the schema of an old relation is changed, a view can be defined to represent the old schema
    - This allows applications to transparently assume the old schema
  - Logical Data Independence!

- Views can be defined to give a group of users access to just the information they are allowed to see
  - E.g., we can define a view that allows students to see other students’ names and ages, but not GPAs (also students can be prevented from accessing the underlying “Students” relation)
  - Security!
Next Class

The Relational Model (Cont’d) and Relational Algebra