EECS 647: Introduction to Database Systems

Instructor: Luke Huan

Spring 2009
Administrative

- Take home background survey is due today
- The grader of this course is Mr. Prashanth Ramani and his email is pramani AT eecs dot ku dot edu
- Your PostgreSQL account has not been created.
  - But you may go to http://people.eecs.ku.edu/~jhuan/EECS647_S09/additionalFiles.html (linked through the class website) and read through the tutorial
What are the two terms used by ER model to describe a miniworld?

- Entity
- Relationship
Today’s Outline

- Relational Model
- Relational Model Constraints and Relational Database Schemas
Relational Model: Thanking in a tabular Way

- Entity: Employees, Departments
- Relationship: Works in
Tabular View

Tabular View of Employees, Departments, and Works

<table>
<thead>
<tr>
<th>E ID</th>
<th>E Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>111</td>
<td>John Smith</td>
</tr>
<tr>
<td>112</td>
<td>Susan Wang</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>D ID</th>
<th>D name</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>HR</td>
</tr>
<tr>
<td>11</td>
<td>Research</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>D ID</th>
<th>E ID</th>
<th>Since</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>111</td>
<td>2001</td>
</tr>
<tr>
<td>11</td>
<td>112</td>
<td>2008</td>
</tr>
</tbody>
</table>
Example

Name of the relation: Employees

<table>
<thead>
<tr>
<th>Attributes</th>
</tr>
</thead>
<tbody>
<tr>
<td>E ID</td>
</tr>
<tr>
<td>111</td>
</tr>
<tr>
<td>112</td>
</tr>
</tbody>
</table>
Historically

- The model was first proposed by Dr. E.F. Codd of IBM in 1970 in the following paper: "A Relational Model for Large Shared Data Banks," Communications of the ACM, June 1970.

The above paper caused a major revolution in the field of Database management and earned Ted Codd the coveted ACM Turing Award.
Relational Model Concepts

- Relational database: a set of relations.
- Relation: made up of 2 parts:
  - Schema: specifies name of relation, plus name and type of each column.
    - E.g. Students (sid: string, name: string, login: string, age: integer, gpa: real)
  - Instance: a table, with rows and columns.
    - #rows = cardinality
    - #fields = degree / arity
Example

- **Schema**
  - *Course* (*CID*: string, *title*: string)
  - *Enroll* (*SID*: integer, *CID*: integer)

- **Instance**
  - `{ 112, Bart, 18, 3.2; 113, Milhouse, 20, 3.1; ... }`
Attribute Types

- Each attribute of a relation has a name, designating the role of the attribute
- The set of allowed values for each attribute is called the *domain* of the attribute
- Attribute values are (normally) required to be *atomic*; that is, indivisible
  - E.g. the value of an attribute can be an account number, but cannot be a set of account numbers
  - The special value *null* is a member of every domain
## Definition Summary

<table>
<thead>
<tr>
<th>Informal Terms</th>
<th>Formal Terms</th>
</tr>
</thead>
<tbody>
<tr>
<td>Table</td>
<td>Relation</td>
</tr>
<tr>
<td>Column</td>
<td>Attribute/Domain</td>
</tr>
<tr>
<td>Row</td>
<td>Tuple</td>
</tr>
<tr>
<td>Possible values in a column</td>
<td>Domain</td>
</tr>
<tr>
<td>Table Definition</td>
<td>Schema of a Relation</td>
</tr>
<tr>
<td>Populated Table</td>
<td>Instances</td>
</tr>
</tbody>
</table>
Characteristics of Relation

- The tuples in a relation \( r(R) \) are not considered to be ordered, even though they appear to be in the tabular form.

- Consider the mathematical definition of relation \( R(A_1, A_2, \ldots, A_n) = A_1 \times A_2 \times \ldots \times A_n \), elements are ordered.
  (well, DB is not math).

- All values are considered atomic (indivisible). A special null value is used to represent values that are unknown or inapplicable to certain tuples.
Relational Integrity Constraints

- Constraints are *conditions* that must hold on *all* valid relation instances. There are four main types of constraints:
  1. Domain constraints
     - The value of a attribute must come from its domain
  2. Key constraints
  3. Referential integrity constraints

<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>53661</td>
<td>History105</td>
<td>B</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>name</th>
<th>login</th>
<th>age</th>
<th>gpa</th>
</tr>
</thead>
<tbody>
<tr>
<td>Jones</td>
<td>jones@cs</td>
<td>-1</td>
<td>3.4</td>
</tr>
<tr>
<td>Smith</td>
<td>smith@eecs</td>
<td>18</td>
<td>3.2</td>
</tr>
<tr>
<td>Smith</td>
<td>smith@math</td>
<td>19</td>
<td>3.8</td>
</tr>
</tbody>
</table>
Primary Key Constraints

- A set of fields is a *candidate 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? A *superkey*.
- If there’s >1 key for a relation, one of the keys is chosen (by DBA) to be the *primary key*.
- E.g., given a schema Student (sid: string, name: string, gpa: float) we have:
  - *sid* is a key for Students. (What about *name*)? The set \{*sid, gpa*\} is a superkey.
Key Example

- CAR (licence_num: string, Engine_serial_num: string, make: string, model: string, year: integer)
  - What is the candidate key(s)
  - Which one you may use as a primary key
  - What are the super keys
Entity Integrity

- Entity Integrity: The *primary key attributes* PK of each relation schema R in S cannot have null values in any tuple of r(R).

- Other attributes of R may be similarly constrained to disallow null values, even though they are not members of the primary key.
Foreign Keys, Referential Integrity

- **Foreign key**: Set of fields in one relation that is used to ‘refer’ to a tuple in another relation. (Must correspond to primary key of the second relation.) Like a ‘logical pointer’.

- E.g. `sid` is a foreign key referring to *Students*:
  - `Student(sid: string, name: string, gpa: float)`
  - `Enrolled(sid: string, cid: string, grade: string)`

- If all foreign key constraints are enforced, *referential integrity* is achieved, i.e., no dangling references.

- Can you name a data model w/o referential integrity?
  - Links in HTML!
Foreign Keys

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

<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>

- Or, use NULL as the value for the foreign key in the referencing tuple when the referenced tuple does not exist.

<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>53661</td>
<td>History105</td>
<td>B</td>
</tr>
</tbody>
</table>
In-Class Exercise

Consider the following relations for a database that keeps track of student enrollment in courses and the books adopted for each course:

STUDENT(SSN, Name, Major, Bdate)

COURSE(Course#, Cname, Dept)

ENROLL(SSN, Course#, Quarter, Grade)

BOOK_ADOPTION(Course#, Quarter, Book_ISBN)

TEXT(Book_ISBN, Book_Title, Publisher, Author)

Draw a relational schema diagram specifying the foreign keys for this schema.
Other Types of Constraints

- Semantic Integrity Constraints:
  - based on application semantics and cannot be expressed by the model per se
  - e.g., “the max. no. of hours per employee for all projects he or she works on is 56 hrs per week”
  - SQL-99 allows triggers and ASSERTIONS to support some of these
Database Design

Miniworld

REQUIREMENTS COLLECTION AND ANALYSIS

Functional Requirements

Data Requirements

FUNCTIONAL ANALYSIS

High-Level Transaction Specification

DBMS-independent

DBMS-specific

CONCEPTUAL DESIGN

Conceptual Schema (In a high-level data model)

LOGICAL DESIGN (DATA MODEL MAPPING)

Logical (Conceptual) Schema (In the data model of a specific DBMS)

PHYSICAL DESIGN

Internal Schema

APPLICATION PROGRAM DESIGN

TRANSACTION IMPLEMENTATION

Application Programs