VirginiaTech

CS 4604: Introduction to Database Management Systems

B. Aditya Prakash

Lecture #2: The Relational Model

News

- Project Assignment 1 is out
- Due Date: Friday Feb 1, 2013, start of class

Course Outline

- Weeks 1–5, 13: Query/ Manipulation Languages
  - Relational Algebra
  - Data definition
  - Programming with SQL
- Weeks 6–8: Data Modeling
  - Entity-Relationship (E/R) approach
  - Specifying Constraints
  - Good E/R design
- Weeks 9–13: Relational Design
  - The relational model
  - Converting ER to "R"
  - Normalization to avoid redundancy
- Week 14–15: Students' choice
  - Practice Problems
  - XML
  - Query optimization
  - Data mining

Data Model

- A Data Model is a notation for describing data or information.
  - **Structure of data** (e.g. arrays, structs)
    - Conceptual model: In databases, structures are at a higher level.
  - **Operations on data** (Modifications and Queries)
    - Limited Operations: Ease of programmers and efficiency of database.
  - **Constraints on data** (what the data can be)

- Examples of data models
  - The Relational Model
  - The Semistructured-Data Model
    - XML and related standards
  - Object-Relational Model

The Relational Model

Student	Course	Grade
Hermione Grainger	Potions	A
Draco Malfoy	Potions	B
Harry Potter	Potions	A
Ron Weasley	Potions	C

- **Structure:** Table (like an array of structs)
- **Operations:** Relational alebgra (selection, projection, conditions, etc)
- **Constraints:** E.g., grades can be only {A, B, C, F}

The Semi-structured model

```
<CoursesTaken>
<Student>Hermione Grainger</Student>
<Course>Potions</Course>
<Grade>A</Grade>
<Student>Draco Malfoy</Student>
<Course>Potions</Course>
<Grade>B</Grade>
...
</CoursesTaken>
```

- **Structure:** Trees or graphs, tags define role played by different pieces of data.
- **Operations:** Follow paths in the implied tree from one element to another.
- **Constraints:** E.g., can express limitations on data types

Comparing the two models

- Flexibility: XML can represent graphs
- Ease of use: SQL enables programmer to express wishes at high level.

The Relational Model

- Simple: Built around a single concept for modeling data: the relation or table.
  - A relational database is a collection of relations.
  - Each relation is a table with rows and columns.

- Supports high-level programming language (SQL).
  - Limited but very useful set of operations

- Has an elegant mathematical design theory.
- Most current DBMS are relational (Oracle, IBM DB2, MS SQL)

Relations

- A relation is a two-dimensional table:
  - Relation == table.
  - Attribute == column name.
  - Tuple == row (not the header row).
- Database == collection of relations.
- A relation has two parts:
  - Schema defines column heads of the table (attributes).
  - Instance contains the data rows (tuples, rows, or records) of the table.

Student	Course	Grade
Hermione Grainger	Potions	A
Draco Malfoy	Potions	B
Harry Potter	Potions	A
Ron Weasley	Potions	C

Schema

CoursesTaken :

Student	Course	Grade
Hermione Grainger	Potions	A
Draco Malfoy	Potions	B
Harry Potter	Potions	A
Ron Weasley	Potions	C

- The schema of a relation is the name of the relation followed by a parenthesized list of attributes.

```
CoursesTaken(Student, Course, Grade)
```

- A design in a relational model consists of a set of schemas.
- Such a set of schemas is called a relational database schema.

Relations: Equivalent Representations

CoursesTaken :

Student	Course	Grade
Hermione Grainger	Potions	A
Draco Malfoy	Potions	B
Harry Potter	Potions	A
Ron Weasley	Potions	C

`CoursesTaken(Student, Course, Grade)`

- Relation is a **set** of tuples and not a list of tuples.
  - Order in which we present the tuples does not matter.
  - Very important!
- The attributes in a schema are also a set (not a list).
  - Schema is the same irrespective of order of attributes.

    `CoursesTaken(Student, Grade, Course)`
  - We specify a "standard" order when we introduce a schema.
- How many equivalent representations are there for a relation with m attributes and n tuples?

  m! n!

Degree and Cardinality

CoursesTaken :

Student	Course	Grade
Hermione Grainger	Potions	A
Draco Malfoy	Potions	B
Harry Potter	Potions	A
Ron Weasley	Potions	C

- Degree/Arity is the number of fields/attributes in schema (=3 in the table above)
- Cardinality is the number of tuples in relation (=4 in the table above)

Keys of Relations

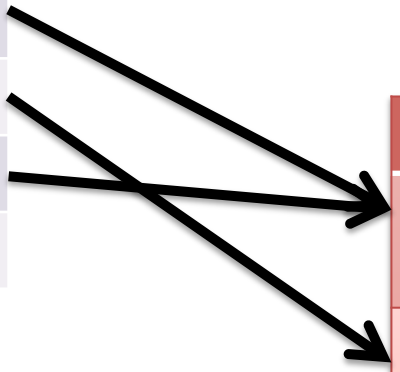
- Keys are one form of integrity constraints (IC)
  - No pair of tuples should have identical keys
- What is the key for CoursesTaken?
  - *Student* if only one course in the relation
  - Pair *(Student, Course)* if multiple courses
  - What if student takes same course many times?

Student	Course	Grade
Hermione Grainger	Potions	A
Draco Malfoy	Potions	B
Harry Potter	Potions	A
Ron Weasley	Potions	C

Keys of Relations

- Keys help associate tuples in different relations

SID	CID	Grade
123	15-401	A
111	15-401	B
123	14-501	B
….	….	….



SID	Student	GPA
123	Hermione Grainger	3.9
111	Draco Malfoy	3.0
234	Harry Potter	3.7
456	Ron Weasley	3.1

Example

- Create a database for managing class enrollments in a single semester. You should keep track of all students (their names, Ids, and addresses) and professors (name, Id, department). Do not record the address of professors but keep track of their ages. Maintain records of courses also. Like what classroom is assigned to a course, what is the current enrollment, and which department offers it. At most one professor teaches each course. Each student evaluates the professor teaching the course. Note that all course offerings in the semester are unique, i.e. course names and numbers do not overlap. A course can have ≥ 0 pre-requisites, excluding itself. A student enrolled in a course must have enrolled in all its pre-requisites. Each student receives a grade in each course. The departments are also unique, and can have at most one chairperson (or dept. head). A chairperson is not allowed to head two or more departments.

Example

- Create a database for managing class enrollments in a single semester. You should keep track of all **students** (their names, Ids, and addresses) and **professors** (name, Id, department). Do not record the address of professors but keep track of their ages. Maintain records of **courses** also. Like what classroom is assigned to a course, what is the current enrollment, and which department offers it. At most one professor **teaches** each course. Each student **evaluates** the professor teaching the course. Note that all course offerings in the semester are unique, i.e. course names and numbers do not overlap. A course can have ≥ 0 p**re-requisites**, excluding itself. A student enrolled in a course must have enrolled in all its pre-requisites. Each student receives a **grade** in each course. The **departments** are also unique, and can have at most one chairperson (or dept. head). A chairperson is not allowed to head two or more departments.

Relational Design for the Example

- Students (PID: *string*, Name: *string*, Address: *string*)
- Professors (PID: *string*, Name: *string*, Office: *string*, Age: *integer*, DepartmentName: *string*)
- Courses (Number: integer, DeptName: *string*, CourseName: *string*, Classroom: *string*, Enrollment: *integer*)
- Teach (ProfessorPID: *string*, Number: *integer*, DeptName: *string*)
- Take (StudentPID: *string*, Number: *integer*, DeptName: *string*, Grade: *string*, ProfessorEvaluation: integer)
- Departments (Name: *string*, ChairmanPID: *string*)
- PreReq (Number: *integer*, DeptName: *string*, PreReqNumber: *integer,* PreReqDeptName: *string*)

Relational Design Example: Keys?

- Students (PID: *string*, Name: *string*, Address: *string*)
- Professors (PID: *string*, Name: *string*, Office: *string*, Age: *integer*, DepartmentName: *string*)
- Courses (Number: integer, DeptName: *string*, CourseName: *string*, Classroom: *string*, Enrollment: *integer*)
- Teach (ProfessorPID: *string*, Number: *integer*, DeptName: *string*)
- Take (StudentPID: *string*, Number: *integer*, DeptName: *string*, Grade: *string*, ProfessorEvaluation: integer)
- Departments (Name: *string*, ChairmanPID: *string*)
- PreReq (Number: *integer*, DeptName: *string*, PreReqNumber: *integer,* PreReqDeptName: *string*)

Relational Design: Keys?

- Students (PID: *string*, Name: *string*, Address: *string*)
- Professors (PID: *string*, Name: *string*, Office: *string*, Age: *integer*, DepartmentName: *string*)
- Courses (Number: integer, DeptName: *string*, CourseName: *string*, Classroom: *string*, Enrollment: *integer*)
- Teach (ProfessorPID: *string*, Number: *integer*, DeptName: *string*)
- Take (StudentPID: *string*, Number: *integer*, DeptName: *string*, Grade: *string*, ProfessorEvaluation: integer)
- Departments (Name: *string*, ChairmanPID: *string*)
- PreReq (Number: *integer*, DeptName: *string*, PreReqNumber: *integer*, PreReqDeptName: *string*)

Issues to Consider in the Design

- Can we merge Courses and Teach since each professor teaches at most one course?
- Do we need a separate relation to store evaluations?
- How can we handle pre-requisites that are "or"s, e.g., you can take CS 4604 if you have taken either CS 3114 or CS 2606?
- How do we generalize this schema to handle data over more than one semester?
- What modifications does the schema need if more than one professor can teach a course?