VirginiaTech

CS 4604: Introduction to Database Management Systems

B. Aditya Prakash

Lecture #2: The Relational Model

Course Outline

- Weeks 1–4: Query/ Manipulation Languages and Data Modeling
  - Relational Algebra
  - Data definition
  - Programming with SQL
  - Entity-Relationship (E/R) approach
  - Specifying Constraints
  - Good E/R design
- Weeks 5–8: Indexes, Processing and Optimization
  - Storing
  - Hashing/Sorting
  - Query Optimization
  - NoSQL and Hadoop
- Week 9-10: Relational Design
  - Functional Dependencies
  - Normalization to avoid redundancy
- Week 11-12: Concurrency Control
  - Transactions
  - Logging and Recovery
- Week 13–14: Students' choice
  - Practice Problems
  - XML
  - Data mining and warehousing

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 **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.

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?

Formal Query Languages

- How do we collect information?
- E.g. Find SSNs of people in 4604

(recall everything is a set!)

What is SQL

- SQL = Structured Query Language (pronounced "sequel").
- Language for defining as well as querying data in an RDBMS.
- Primary mechanism for querying and modifying the data in an RDBMS.
- SQL is declarative:
  - Say what you want to accomplish, without specifying how.
  - One of the main reasons for the commercial success of RDMBSs.
- SQL has many standards and implementations:
  - ANSI SQL
  - SQL-92/SQL2 (null operations, outerjoins)
  - SQL-99/SQL3 (recursion, triggers, objects)
  - Vendor-specific variations.

Relational Algebra

- Relational algebra is a notation for specifying queries about the contents of relations
- Notation of relational algebra eases the task of reasoning about queries
- Operations in relational algebra have counterparts in SQL

What is an Algebra?

- An algebra is a set of operators and operands
  - Arithmetic: operands are variables and constants, operators are +,−,×,÷, /, etc.
  - Set algebra: operands are sets and operators are ∩, U, -

- An algebra allows us to
  - **construct expressions** by combining operands and expression using operators
  - has **rules for reasoning** about expressions

$a^2 + 2 \times a \times b + 2b, \quad (a + b)^2$

$R - (R - S), \quad R \cap S$

Basics of Relational Algebra

- Operands are relations, thought of as sets of tuples.
- Think of operands as variables, whose tuples are unknown.
- Results of operations are also sets of tuples.
- Think of expressions in relational algebra as queries, which construct new relations from given relations.

- Four types of operators:
  - Select/Show parts of a single relation: projection and selection.
  - Usual set operations (union, intersection, difference).
  - Combine the tuples of two relations, such as cartesian product and joins.
  - Renaming.

Projection

- The projection operator produces from a relation R a new relation containing only **some of R's columns**

- "Delete" (i.e. not show) attributes not in projection list
- Duplicates eliminated (sets vs *multi*sets)

- To obtain a relation containing only the columns $A_1, A_2, \ldots A_n$ of R

RA: $\pi\ A_1, A_2, \ldots A_n\ (R)$

SQL: **SELECT DISTINCT** $A_1, A_2, \ldots A_n$ **FROM** R;

Projection Example

S2

sid	sname	rating	age
28	yuppy	9	35.0
31	lubber	8	55.5
44	guppy	5	35.0
58	rusty	10	35.0

$\pi_{sname,rating}(S2)$

sname	rating
yuppy	9
lubber	8
guppy	5
rusty	10

$\pi_{age}(S2)$

age
35.0
55.5

Selection

- The selection operator applied to a relation R produces a new relation with a **subset of R's tuples**
- The tuples in the resulting relation satisfy some condition C that involves the attributes of R
  - with duplicate removal

RA: $\boldsymbol{\sigma}_C$ (R)

SQL: **SELECT** * **FROM** R **WHERE** C;

- The WHERE clause of a SQL command corresponds to **σ**( )

Selection: Syntax of Conditional

- Syntax of conditional (C): similar to conditionals in programming languages.

- Values compared are constants and attributes of the relations mentioned in the FROM clause.

- We may apply usual arithmetic operators to numeric values before comparing them.

RA Compare values using standard arithmetic operators.

SQL Compare values using =, <>, <, >, <=, >=.

Selection Example

S2

sid	sname	rating	age
28	yuppy	9	35.0
31	lubber	8	55.5
44	guppy	5	35.0
58	rusty	10	35.0

$$\sigma_{rating>8}(S2)$$

sid	sname	rating	age
28	yuppy	9	35.0
58	rusty	10	35.0

$$\pi_{sname,rating}(\sigma_{rating>8}(S2))$$

sname	rating
yuppy	9
rusty	10

Combining Operators

Set Operations: Union

- Standard definition: The union of two relations R and S is the set of tuples that are in R, or S or in both.
- When is it valid?
  - R and S must have identical sets of attributes and the types of the attributes must be the same.
  - The attributes of R and S must occur in the same order.

Set Operations: Union

- RA R U S
- SQL (SELECT * FROM R)
  UNION
  (SELECT * FROM S);

Set Operations: Intersection

- The intersection of R and S is the set of tuples in both R and S
- Same conditions hold on R and S as for the union operator
- RA $R \cap S$
- SQL
```
(SELECT * FROM R)
 INTERSECT
(SELECT * FROM S);
```

Set Operations: Difference

- Set of tuples in R but NOT in S
- Same conditions on R and S as union
- RA $R \cap S$
- SQL
```
(SELECT * FROM R)
 EXCEPT
(SELECT * FROM S);
```
- $R - (R - S) = R \cap S$

Difference

S1

sid	sname	rating	age
22	dustin	7	45.0
31	lubber	8	55.5
58	rusty	10	35.0

S2

sid	sname	rating	age
28	yuppy	9	35.0
31	lubber	8	55.5
44	guppy	5	35.0
58	rusty	10	35.0

$S1 - S2$

sid	sname	rating	age
22	dustin	7	45.0