Lecture 32: The Relational Model

Professor Xiannong Meng
Spring 2018

Lecture and activity contents are based on what Prof Chris Ré of Stanford used in his CS 145 in the fall 2016 term with permission

Today's Lecture

1. The Relational Model & Relational Algebra
2. Relational Algebra Pt. II

2

1. The Relational Model & Relational Algebra

3

What you will learn about in this section

1. The Relational Model
2. Relational Algebra: Basic Operators
3. Execution
4. ACTIVITY: From SQL to RA & Back (*Can't quite convert Python2 code yet*.)

4

Motivation

The Relational model is **precise**, **implementable**, and we can operate on it (query/update, etc.)

Database maps internally into this *procedural language.*

A Little History

- Relational model due to Edgar "Ted" Codd, a mathematician at IBM in 1970
  - "[A Relational Model of Data for Large Shared Data Banks](#)". *Communications of the ACM* **13** (6): 377–387

Won Turing award 1981

- IBM didn't want to use relational model (taken money away from IMS, or information management system)
  - *Apparently used in the moon landing…*

The Relational Model: Schemata

- Relational Schema:

Students(sid:*string*, name:*string*, gpa:*float*)

Students (sid: *string*, name: *string*, gpa: *float*)

Relation name

String, float, int, etc. are the **domains** of the attributes

Attributes

The Relational Model: Data

Student

sid	name	gpa
001	Bob	3.2
002	Joe	2.8
003	Mary	3.8
004	Alice	3.5

An **attribute** (or **column**) is a typed data entry present in each tuple in the relation

The number of attributes is the **arity** of the relation

8

The Relational Model: Data

Student

sid	name	gpa
001	Bob	3.2
002	Joe	2.8
003	Mary	3.8
004	Alice	3.5

The number of tuples is the **cardinality** of the relation

A **tuple** or **row** (or *record)* is a single entry in the table having the attributes specified by the schema

9

The Relational Model: Data

Student

sid	name	gpa
001	Bob	3.2
002	Joe	2.8
003	Mary	3.8
004	Alice	3.5

Recall: In practice DBMSs relax the set requirement, and use multisets.

A **relational instance** is a ***set*** of tuples all conforming to the same *schema*

10

To Reiterate

- A *relational schema* describes the data that is contained in a *relational instance*

Let $R(f_1{:}Dom_1,\dots,f_m{:}Dom_m)$ be a *relational schema* then, an *instance* of R is a subset of $Dom_1 \times Dom_2 \times \dots \times Dom_n$

In this way, a *relational schema* R is a **total function from attribute *names* to types**

One More Time

- A *relational schema* describes the data that is contained in a *relational instance*

A relation R of arity *t* is a function: R : $Dom_1 \times \dots \times Dom_t \rightarrow \{0,1\}$ or {false, true}

I.e. returns whether or not a tuple of matching types is a member of it

Then, the schema is simply the *signature* (or *prototype*) of the function

Note here that order matters, attribute name doesn't… We'll (mostly) work with the other model (last slide) in which **attribute name matters, order doesn't!**

A relational database

- A *relational database schema* is a set of relational schemata, one for each relation
- A *relational database instance* is a set of relational instances, one for each relation

Two conventions:
1. We call relational database instances as simply ***databases***
2. We assume all instances are valid, i.e., satisfy the *domain constraints*

Remember the CMS (Course Management System)

Note that the schemas impose effective domain / type constraints, i.e. Gpa can't be "Apple"

- *Relation DB Schema*
  - Students(sid: *string*, name: *string*, gpa: *float*)
  - Courses(cid: *string*, cname: *string*, credits: *int*)
  - Enrolled(sid: *string*, cid: *string*, grade: *string*)

Relation Instances

Sid	Name	Gpa
101	Bob	3.2
123	Mary	3.8

Students

cid	cname	credits
564	564-2	4
308	417	2

Courses

sid	cid	Grade
123	564	A

Enrolled

14

2nd Part of the Model: Querying

```
SELECT S.name
FROM Students S
WHERE S.gpa > 3.5;
```

"Find names of all students with GPA > 3.5"

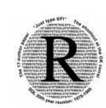
We don't tell the system *how* or *where* to get the data- **just what we want, i.e., Querying is *declarative***

To make this happen, we need to translate the *declarative* query into a series of operators… we'll see this next!

Actually, I showed how to do this translation for a much richer language!

Virtues of the model

- Physical independence (logical too), Declarative
- Simple, elegant clean: Everything is a relation
- Why did it take multiple years?
  - Doubted it could be done *efficiently*.



Relational Algebra

RDBMS Architecture

How does a SQL engine work ?

SQL Query → Relational Algebra (RA) Plan → *Optimized* RA Plan → Execution

Declarative query (from user)

Translate to relational algebra expresson

Find logically equivalent- but more efficient- RA expression

Execute each operator of the optimized plan!

RDBMS Architecture

How does a SQL engine work ?

SQL Query → **Relational Algebra (RA) Plan** → *Optimized RA Plan* → Execution

Relational Algebra allows us to translate declarative (SQL) queries into precise and optimizable expressions!

Relational Algebra (RA)

- Five **basic** operators:
  1. Selection: σ
  2. Projection: Π
  3. Cartesian Product: $\times$
  4. Union: $\cup$
  5. Difference: -
- Derived or auxiliary operators:
  - Intersection, complement
  - Joins (natural,equi-join, theta join, semi-join)
  - Renaming: ρ
  - Division

We'll look at these first!

And also at one example of a derived operator (natural join) and a special operator (renaming)

Keep in mind: RA operates on sets!

- RDBMSs use *multisets*, however in relational algebra formalism we will consider **sets!**
- Also: we will consider the ***named perspective***, where every attribute must have a unique name
  - →attribute order does not matter…

Now on to the basic RA operators…

1. Selection (σ)

Students(sid,sname,gpa)

- Returns all tuples which satisfy a condition
- Notation: $\sigma_c(R)$
- Examples
  - $\sigma_{\text{Salary} > 40000}$ (Employee)
  - $\sigma_{\text{name} = \text{"Smith"}}$ (Employee)
- The condition c can be =, <, ≤, >, ≥, <>

SQL:

```
SELECT *
FROM Students
WHERE gpa > 3.5;
```

RA:

$\sigma_{gpa > 3.5}(Students)$

Another example:

SSN	Name	Salary
1234545	John	200000
5423341	Smith	600000
4352342	Fred	500000

$\sigma_{\text{Salary} > 40000}$ (Employee)

SSN	Name	Salary
5423341	Smith	600000
4352342	Fred	500000

2. Projection (Π)

Students(sid,sname,gpa)

- Eliminates columns, then removes duplicates
- Notation: $\Pi_{A1,\dots,An}(R)$
- Example: project social-security number and names:
  - $\Pi_{\text{SSN, Name}}$ (Employee)
  - Output schema: Answer(SSN, Name)

SQL:

```
SELECT DISTINCT
  sname,
  gpa
FROM Students;
```

RA:

$\Pi_{sname,gpa}(Students)$

Another example:

SSN	Name	Salary
1234545	John	200000
5423341	John	600000
4352342	John	200000

$\Pi_{\text{Name,Salary}}$ (Employee)

Name	Salary
John	200000
John	600000