Informatics 1: Data & Analysis

Lecture 6: Tuple Relational Calculus

Ian Stark

School of Informatics
The University of Edinburgh

Friday 31 January 2014
Semester 2 Week 3

Careers in IT

Job Fair

Wednesday 5 February 2014

Informatics Forum
1300–1600
http://is.gd/it_careers

Careers advice and stalls from 35+ local, national and international employers

Lecture Plan for Weeks 1–4

Data Representation

This first course section starts by presenting two common data representation models.

- The *entity-relationship (ER)* model
- The *relational* model

Note slightly different naming:
-relation**ship** vs. relation**al**

Data Manipulation

This is followed by some methods for manipulating data in the relational model and using it to extract information.

- *Relational algebra*
- The *tuple-relational calculus*
- The query language *SQL*

The State We're In

Relational models

- Relations: Tables matching schemas
- Schema: A set of field names and their domains
- Table: A set of tuples of values for these fields

Student

mn	**name**	**age**	**email**
s0456782	John	18	john@inf
s0378435	Helen	20	helen@phys
s0412375	Mary	18	mary@inf
s0189034	Peter	22	peter@math

Takes

mn	**code**	**mark**
s0456782	inf1	71
s0412375	geo1	64
s0412375	math1	82
s0189034	math1	56

Course

code	**title**	**year**
inf1	Informatics 1	1
math1	Mathematics 1	1
geo1	Geology 1	1
dbs	Database Systems	3
adbs	Advanced Databases	4

The State We're In

Relational algebra

A mathematical language of bulk operations on relational tables. Each operation takes one or more tables, and returns another.

> selection σ, projection π, renaming ρ, union $\cup$, difference $-$, cross-product $\times$, intersection $\cap$ and different kinds of join $\bowtie$

Tuple relational calculus (TRC)

A declarative mathematical notation for writing queries: specifying information to be drawn from the linked tables of a relational model.

Structured Query Language (SQL)

A mostly-declarative programming language for interacting with relational database management systems (RDBMS): defining tables, changing data, writing queries. International Standard ISO 9075

Tuple Relational Calculus: Example

All records for students more than 19 years old

$$\{\ S \mid S \in \text{Student} \ \wedge\ S.\text{age} > 19\ \}$$

The set of tuples S such that S is in the table "Student" and has component "age" greater than 19.

Tuple Relational Calculus: Example

All records for students more than 19 years old

$$\{\, S \mid S \in \text{Student} \;\wedge\; S.\text{age} > 19 \,\}$$

The set of tuples S such that S is in the table "Student" and has component "age" greater than 19.

Student

mn	name	age	email
s0456782	John	18	john@inf
s0378435	Helen	20	helen@phys
s0412375	Mary	18	mary@inf
s0189034	Peter	22	peter@math

Tuple Relational Calculus: Example

All records for students more than 19 years old

$$\{ \, S \mid S \in \text{Student} \, \wedge \, S.\text{age} > 19 \, \}$$

The set of tuples S such that S is in the table "Student" and has component "age" greater than 19.

Student

mn	name	age	email
s0456782	John	18	john@inf
s0378435	Helen	20	helen@phys
s0412375	Mary	18	mary@inf
s0189034	Peter	22	peter@math

Tuple Relational Calculus: Example

All records for students more than 19 years old

$$\{\, S \mid S \in \text{Student} \wedge S.\text{age} > 19 \,\}$$

The set of tuples S such that S is in the table "Student" and has component "age" greater than 19.

This is like list comprehension in programming languages:

Haskell	[s | s <− students, age s > 19]
Python	[s **for** s **in** students **if** s.age > 19]

All are based on "comprehensions" in set theory

Tuple Relational Calculus Basics

Queries in TRC have the general form

$$\{\, T \mid P(T) \,\}$$

where T is a *tuple variable* and $P(T)$ is a predicate, a logical formula.

Every tuple variable such as T has a *schema*, listing its fields and their domains. In practice, the details of the schema are usually inferred from the way T is used in the predicate $P(T)$.

A tuple variable ranges over all possible tuple values matching its schema.

The result of the query

$$\{\, T \mid P(T) \,\}$$

is then the set of all possible tuple values for T such that $P(T)$ is true.

Another Example

Names and ages of all students over 19

$$\{\, \mathrm{T} \mid \exists S \,.\, S \in \mathrm{Student} \;\wedge\; S.\mathrm{age} > 19 \;\wedge\; \mathrm{T.name} = S.\mathrm{name} \;\wedge\; \mathrm{T.age} = S.\mathrm{age} \,\}$$

The set of tuples T such that there is a tuple S in table "Student" with field "age" greater than 19 and where S and T have the same values for "name" and "age".

Student

mn	name	age	email
s0456782	John	18	john@inf
s0378435	Helen	20	helen@phys
s0412375	Mary	18	mary@inf
s0189034	Peter	22	peter@math

T

name	age
Helen	20
Peter	22

Another Example

Names and ages of all students over 19

$$\{\, \mathrm{T} \mid \exists S \,.\, S \in \mathrm{Student} \;\wedge\; S.\mathrm{age} > 19 \;\wedge\; \mathrm{T.name} = S.\mathrm{name} \;\wedge\; \mathrm{T.age} = S.\mathrm{age} \,\}$$

The set of tuples T such that there is a tuple S in table "Student" with field "age" greater than 19 and where S and T have the same values for "name" and "age".

- Tuple variable S has schema matching the table "Student".
- Tuple variable T has fields "name" and "age", with domains to match those of S.
- Even if S has other fields, they do not appear in T or the overall result.

Formula Syntax

Inside TRC expression $\{\, T \mid P(T)\, \}$ the logical formula $P(T)$ may be quite long, but is built up from standard logical components.

- Simple assertions: $(T \in \text{Table})$, $(T.\text{age} > 65)$, $(S.\text{name} = T.\text{name})$, ...
- Logical combinations: $(P \vee Q)$, $(P \wedge Q \wedge \neg Q')$, ...
- Quantification:

$\exists S \,.\, P(S)$	There exists a tuple S such that $P(S)$
$\forall T \,.\, Q(T)$	For all tuples T it is true that $Q(T)$

For convenience, we require that for $\exists S \,.\, P(S)$ the variable S must actually appear in $P(S)$; and the same for $\forall T \,.\, Q(T)$. We also write:

$$\exists S \in \text{Table} \,.\, P(S) \quad \text{to mean} \quad \exists S \,.\, S \in \text{Table} \wedge P(S)$$

Students and Courses

Student

mn	name	age	email
s0456782	John	18	john@inf
s0378435	Helen	20	helen@phys
s0412375	Mary	18	mary@inf
s0189034	Peter	22	peter@math

Takes

mn	code	mark
s0456782	inf1	71
s0412375	geo1	64
s0412375	math1	82
s0189034	math1	56

Course

code	title	year
inf1	Informatics 1	1
math1	Mathematics 1	1
geo1	Geology 1	1
dbs	Database Systems	3
adbs	Advanced Databases	4

Students and Courses (1/5)

Students taking Geology 1

$$\{\, R \mid \exists S \in \text{Student} \,.\, \exists T \in \text{Takes} \,.\, \exists C \in \text{Course} \,.$$
$$C.\text{title} = \text{"Geology 1"} \;\wedge\; C.\text{code} = T.\text{code}$$
$$\wedge\; T.\text{mn} = S.\text{mn} \;\wedge\; S.\text{name} = R.\text{name} \,\}$$

Schema for S, T and C match those of the tables from which they are drawn. The schema for result R is a single field "name" with string domain, because that's all that appears here.

One way to compute this in relational algebra:

$$\pi_{\text{name}}((\text{Student} \bowtie \text{Takes}) \bowtie (\sigma_{\text{title}=\text{"Geology 1"}}(\text{Course})))$$

Relational Algebra

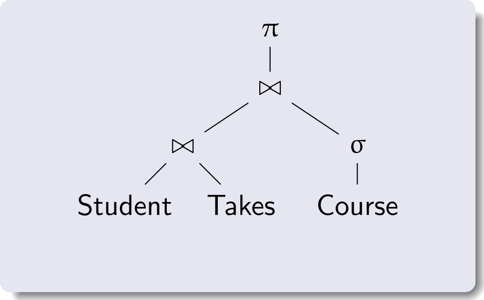
The relational algebra expression can be rearranged without changing its value, but possibly affecting the time and memory needed for computation:

$$\pi_{\text{name}}((\text{Student} \bowtie \text{Takes}) \bowtie (\sigma_{\text{title}=\text{"Geology 1"}}(\text{Course})))$$

$$\pi_{\text{name}}(\text{Student} \bowtie (\text{Takes} \bowtie (\sigma_{\text{title}=\text{"Geology 1"}}(\text{Course}))))$$

$$\pi_{\text{name}}(\text{Student} \bowtie ((\sigma_{\text{title}=\text{"Geology 1"}}(\text{Course})) \bowtie \text{Takes}))$$

We can also visualise this as rearrangements of a tree:



Relational Algebra

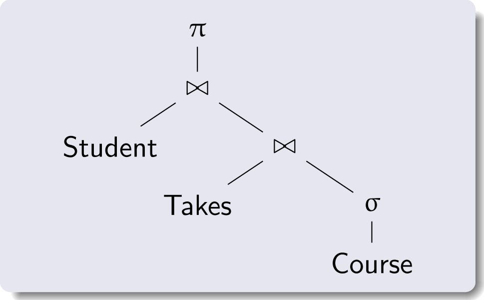
The relational algebra expression can be rearranged without changing its value, but possibly affecting the time and memory needed for computation:

$$\pi_{\text{name}}((\text{Student} \bowtie \text{Takes}) \bowtie (\sigma_{\text{title}=\text{"Geology 1"}}(\text{Course})))$$

$$\pi_{\text{name}}(\text{Student} \bowtie (\text{Takes} \bowtie (\sigma_{\text{title}=\text{"Geology 1"}}(\text{Course}))))$$

$$\pi_{\text{name}}(\text{Student} \bowtie ((\sigma_{\text{title}=\text{"Geology 1"}}(\text{Course})) \bowtie \text{Takes}))$$

We can also visualise this as rearrangements of a tree:



Relational Algebra

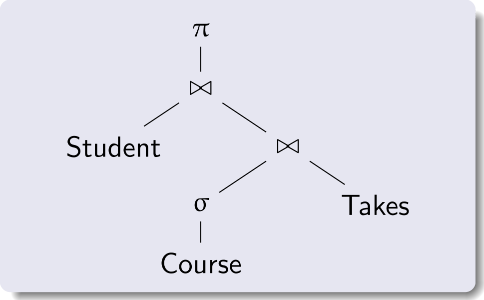
The relational algebra expression can be rearranged without changing its value, but possibly affecting the time and memory needed for computation:

$$\pi_{\mathsf{name}}((\mathsf{Student} \bowtie \mathsf{Takes}) \bowtie (\sigma_{\mathsf{title}=\texttt{"}\mathsf{Geology\ 1}\texttt{"}}(\mathsf{Course})))$$

$$\pi_{\mathsf{name}}(\mathsf{Student} \bowtie (\mathsf{Takes} \bowtie (\sigma_{\mathsf{title}=\texttt{"}\mathsf{Geology\ 1}\texttt{"}}(\mathsf{Course}))))$$

$$\pi_{\mathsf{name}}(\mathsf{Student} \bowtie ((\sigma_{\mathsf{title}=\texttt{"}\mathsf{Geology\ 1}\texttt{"}}(\mathsf{Course})) \bowtie \mathsf{Takes}))$$

We can also visualise this as rearrangements of a tree:



Students and Courses (2/5)

Courses taken by students called "Joe"

$$\{\ R \mid \exists S \in \mathsf{Student}, T \in \mathsf{Takes}, C \in \mathsf{Course}\ .$$
$$S.\mathsf{name} = \text{"Joe"} \ \wedge\ S.\mathsf{mn} = T.\mathsf{mn}$$
$$\wedge\ C.\mathsf{code} = T.\mathsf{code} \ \wedge\ C.\mathsf{title} = R.\mathsf{title}\ \}$$

Note the slightly abbreviated syntax for multiple quantification: we use comma-separated $\exists..,..,..$ instead of $\exists..\exists..\exists..$

Computing this in relational algebra:

$$\pi_{\mathsf{title}}((\mathsf{Course} \bowtie \mathsf{Takes}) \bowtie (\sigma_{\mathsf{name}=\text{"Joe"}}(\mathsf{Student})))$$

Students and Courses (3/5)

Students taking Informatics 1 or Geology 1

$$\{ R \mid \exists S \in \text{Student}, T \in \text{Takes}, C \in \text{Course} \,.$$
$$(C.\text{title} = \text{"Informatics 1"} \vee C.\text{title} = \text{"Geology 1"})$$
$$\wedge\; C.\text{code} = T.\text{code} \;\wedge\; T.\text{mn} = S.\text{mn} \;\wedge\; S.\text{name} = R.\text{name} \}$$

Now the logical formula becomes a little more elaborate.

Computing this in relational algebra:

$$\pi_{\text{name}}((\text{Student} \bowtie \text{Takes}) \bowtie (\sigma_{\text{title}=\text{"Informatics 1"}}(\text{Course})))$$
$$\cup\; \pi_{\text{name}}((\text{Student} \bowtie \text{Takes}) \bowtie (\sigma_{\text{title}=\text{"Geology 1"}}(\text{Course})))$$

$$\pi_{\text{name}}((\text{Student} \bowtie \text{Takes}) \bowtie (\sigma_{(\text{title}=\text{"Informatics 1"} \vee \text{title}=\text{"Geology 1"})}(\text{Course})))$$

Students and Courses (4/5)

Students taking both Informatics 1 and Geology 1

$$\{\, R \mid \exists S \in \mathsf{Student}, T, T' \in \mathsf{Takes}, C, C' \in \mathsf{Course}\;. \\ C.\mathsf{title} = \text{"Informatics 1"} \wedge C.\mathsf{code} = T.\mathsf{code} \wedge T.\mathsf{mn} = S.\mathsf{mn} \\ C'.\mathsf{title} = \text{"Geology 1"} \wedge C'.\mathsf{code} = T'.\mathsf{code} \wedge T'.\mathsf{mn} = S.\mathsf{mn} \\ \wedge\; S.\mathsf{name} = R.\mathsf{name} \,\}$$

Computing this in relational algebra:

$$\pi_{\mathsf{name}}((\mathsf{Student} \bowtie \mathsf{Takes}) \bowtie (\sigma_{\mathsf{title}=\text{"Informatics 1"}}(\mathsf{Course}))) \\ \cap\; \pi_{\mathsf{name}}((\mathsf{Student} \bowtie \mathsf{Takes}) \bowtie (\sigma_{\mathsf{title}=\text{"Geology 1"}}(\mathsf{Course})))$$

Students and Courses (5/5)

Students taking no courses

$$\{ R \mid \exists S \in \text{Student} \,.\, S.\text{name} = R.\text{name} \;\wedge\; \forall T \in \text{Takes} \,.\, T.\text{mn} \neq S.\text{mn}$$

Computing this in relational algebra:

$$\pi_{\text{name}}(\text{Student} - \pi_{\text{name,mn,age,email}}(\text{Student} \bowtie \text{Takes}))$$

⋆ Challenge: why not one of these instead?

$$\pi_{\text{name}}(\text{Student} - (\text{Student} \bowtie \text{Takes}))$$

$$\pi_{\text{name}}(\text{Student}) - \pi_{\text{name}}(\text{Student} \bowtie \text{Takes}))$$

Relational Algebra vs. Tuple Relational Calculus

Codd gave a proof that relational algebra and TRC are equally expressive: anything expressed in one language can also be written in the other.

So why have both?

They give different perspectives and allow the following approach:

- Use relational calculus to specify the information wanted;
- Translate into relational algebra to give a procedure for computing it;
- Rearrange the algebra to make that procedure efficient.

The database language SQL is based on the calculus: well-suited to giving logical specifications, independent of any eventual implementation.

The algebra beneath it is good for rewriting, equations, and calculation.

Query Optimization

- … Rearrange the algebra to make that procedure efficient.

This last part is central to the viability of modern large databases. An effective *query optimizer* will draw up a list of possible *query plans* and compare the costs of all of them, taking account of:

- How much data there is, where it is, how it is arranged;
- What indexes are available, for which tables, and where they are;
- Selectivity: estimates of how many rows a subquery will return;
- Estimated size of any intermediate tables;
- What parts can be done in parallel;
- What I/O and computing resources are available;
- . . .