# Informatics 1: Data & Analysis

## Lecture 6: Tuple Relational Calculus

Ian Stark

School of Informatics
The University of Edinburgh

Friday 1 February 2013
Semester 2 Week 3

# Lecture Plan

## Data Representation

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

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

## 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 matching these fields

## Relational algebra

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

selection $\sigma$, projection $\pi$, renaming $\rho$, 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.

# Simple Example

## All records for students more than 18 years old

$$\{\, S \mid S \in \text{Students} \ \wedge\ S.\text{age} > 18 \,\}$$

The set of tuples $S$ such that $S$ is in the table “Students” and has component “age” least 18.

This is like list comprehension in Haskell

```
[ s | s <- students, age s > 18 ]
```

and similar constructions in other languages.

All are based on “comprehensions” in set theory

# Tuple Relational Calculus Basics

Queries in TRC have the general form

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

where T is a *tuple variable* and $\mathrm{P}(\mathrm{T})$ is a logical formula.

Every tuple variable such as T has a *schema*, like rows in a relational table, with fields and their domains. In practice, the details of the schema are usually inferred from the way T appears in $\mathrm{P}(\mathrm{T})$.

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

The result of the query

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

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

# Another Example

## Names and ages of all students over 20

$$\{\, \mathsf{T} \mid \exists S \,.\, S \in \mathsf{Students} \;\wedge\; S.\mathsf{age} > 20$$
$$\wedge\; \mathsf{T.name} = S.\mathsf{name} \;\wedge\; \mathsf{T.age} = S.\mathsf{age} \,\}$$

The set of tuples T such that there is an $S$ in table "Students" with component "age" at least 20 and where $S$ and T have the same values for "name" and "age".

- Tuple variable $S$ has schema matching the table "Students".
- Tuple variable T has (only) 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 (1/5)

| mn | code | mark |
|---|---|---|
| s0456782 | inf1 | 80 |
| s0412375 | geo1 | 78 |
| s0412375 | inf1 | 56 |
| s0189034 | math1 | 62 |

Referencing relation

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

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

Referenced relations

Primary key

Foreign key

# Students and Courses (1/5)

## Students taking Informatics 1

$$\{\, R \mid \exists S \in \mathsf{Students} \,.\, \exists T \in \mathsf{Takes} \,.\, \exists C \in \mathsf{Courses} \,.$$
$$C.\mathsf{name} = \text{"Informatics 1"} \;\wedge\; C.\mathsf{code} = T.\mathsf{code}$$
$$\wedge\; T.\mathsf{mn} = S.\mathsf{mn} \;\wedge\; S.\mathsf{name} = R.\mathsf{name} \,\}$$

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:

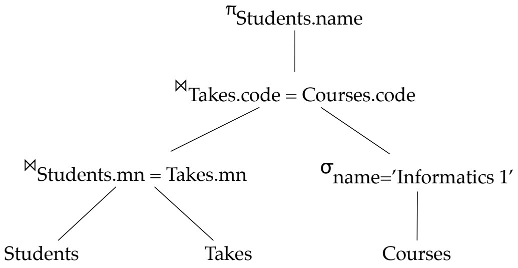
$$\pi_{\mathsf{name}}((\mathsf{Students} \bowtie \mathsf{Takes}) \bowtie (\sigma_{\mathsf{name}=\text{"Informatics 1"}}(\mathsf{Courses})))$$

# Relational Algebra

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

$$\pi_{\mathsf{name}}((\mathsf{Students} \bowtie \mathsf{Takes}) \bowtie (\sigma_{\mathsf{name}=\text{"Informatics 1"}}(\mathsf{Courses})))$$
$$\pi_{\mathsf{name}}(\mathsf{Students} \bowtie (\mathsf{Takes} \bowtie (\sigma_{\mathsf{name}=\text{"Informatics 1"}}(\mathsf{Courses}))))$$
$$\pi_{\mathsf{name}}(\mathsf{Students} \bowtie ((\sigma_{\mathsf{name}=\text{"Informatics 1"}}(\mathsf{Courses})) \bowtie \mathsf{Takes}))$$

We can also visualise this as rearrangements of a tree:



# Students and Courses (2/5)

## Courses taken by students called “Joe”

$$\{\, R \mid \exists S \in \text{Students}, T \in \text{Takes}, C \in \text{Courses} \;.\; S.\text{name} = \text{"Joe"} \;\wedge\; S.\text{mn} = T.\text{mn} \;\wedge\; C.\text{code} = T.\text{code} \;\wedge\; C.\text{name} = R.\text{name} \,\}$$

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

Computing this in relational algebra:

$$\pi_{\text{name}}((\text{Courses} \bowtie \text{Takes}) \bowtie (\sigma_{\text{name}=\text{"Joe"}}(\text{Students})))$$

# Students and Courses (3/5)

## Students taking Informatics 1 or Geology 1

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

Now the logical formula becomes a little more elaborate.

Computing this in relational algebra:

$$\pi_{\text{name}}((\text{Students} \bowtie \text{Takes}) \bowtie (\sigma_{\text{name}=\text{"Informatics 1"}}(\text{Courses})))\\ \cup\, \pi_{\text{name}}((\text{Students} \bowtie \text{Takes}) \bowtie (\sigma_{\text{name}=\text{"Geology 1"}}(\text{Courses})))$$

$$\pi_{\text{name}}((\text{Students} \bowtie \text{Takes}) \bowtie (\sigma_{(\text{name}=\text{"Informatics 1"} \vee \text{name}=\text{"Geology 1"})}(\text{Courses}))$$

# Students and Courses (4/5)

## Students taking both Informatics 1 and Geology 1

$$
\begin{aligned}
\{\, R \mid \exists S \in \text{Students}, T, T' \in \text{Takes}, C, C' \in \text{Courses}\ .\\
&C.\text{name} = \text{"Informatics 1"} \ \wedge\ C.\text{code} = T.\text{code} \ \wedge\ T.\text{mn} = S.\text{mn}\\
&C'.\text{name} = \text{"Geology 1"} \ \wedge\ C'.\text{code} = T'.\text{code} \ \wedge\ T'.\text{mn} = S.\text{mn}\\
&\ \wedge\ S.\text{name} = R.\text{name} \,\}
\end{aligned}
$$

Computing this in relational algebra:

$$
\begin{aligned}
&\pi_{\text{name}}((\text{Students} \bowtie \text{Takes}) \bowtie (\sigma_{\text{name}=\text{"Informatics 1"}}(\text{Courses})))\\
&\qquad \cap\ \pi_{\text{name}}((\text{Students} \bowtie \text{Takes}) \bowtie (\sigma_{\text{name}=\text{"Geology 1"}}(\text{Courses})))
\end{aligned}
$$

# Students and Courses (5/5)

## Students taking no courses

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

Computing this in relational algebra:

$$\pi_{\mathsf{name}}(\mathsf{Students} - \pi_{\mathsf{name,mn}}(\mathsf{Students} \bowtie \mathsf{Takes}))$$

⋆ Challenge: why not one of these instead?

$$\pi_{\mathsf{name}}(\mathsf{Students} - (\mathsf{Students} \bowtie \mathsf{Takes}))$$

$$\pi_{\mathsf{name}}(\mathsf{Students}) - \pi_{\mathsf{name}}(\mathsf{Students} \bowtie \mathsf{Takes}))$$

# 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.

# Domain-Specific Languages


Charles V, 1500–1558
Holy Roman Emperor, King of Spain, Archduke of Austria

# Domain-Specific Languages


Charles V, 1500–1558
Holy Roman Emperor, King of Spain, Archduke of Austria

"I speak Spanish to God, Italian to women, French to men and German to my horse."

# Domain-Specific Languages


Charles V, 1500–1558
Holy Roman Emperor, King of Spain, Archduke of Austria

"I speak Spanish to God, Italian to women, French to men and German to my horse."

Attributed, but even Wikipedia is sceptical.

# 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;
- . . .