Informatics 1: Data & Analysis

Lecture 6: Tuple Relational Calculus

Ian Stark

School of Informatics
The University of Edinburgh

Friday 30 January 2014
Semester 2 Week 3

THE UNIVERSITY *of* EDINBURGH

Careers in IT

Job Fair

Wednesday 4 February 2015

Informatics Forum
1300–1600
http://is.gd/da15careers

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

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

Takes

matric	code	mark
s0456782	inf1	71
s0412375	math1	82
s0412375	geo1	64
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.

Student

matric	**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

matric	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

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

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

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 19

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

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

matric	**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

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

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.

Another Example

Names and ages of all students over 19

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

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".

The variable occurrences $\{\, T \mid \cdots$ and $\exists S \,.\, \cdots$ are *binding* — they name a variable which is used later in the expression

- The binding of T extends over the whole set $\{\, T \mid \ldots T \ldots T \ldots \}$
- The binding of S includes everything to the right of the binder $\exists S$.

The region over which each variable is bound is called its *scope*.

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.

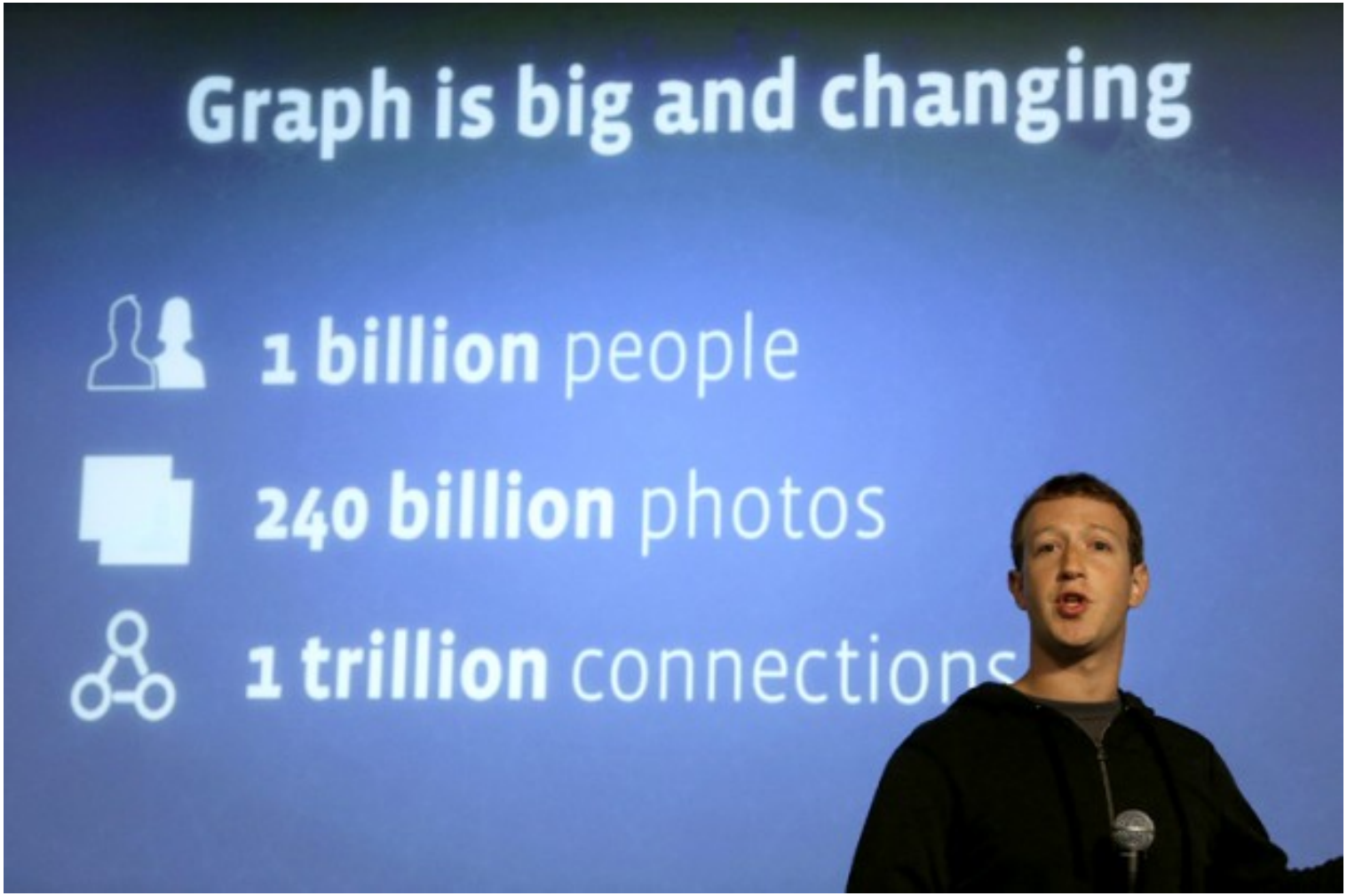
- Simple assertions: $(T \in \text{Table})$, $(T.\text{age} > 65)$, $(S.\text{name} = T.\text{name})$, ...
- Logical combinations: $(P \vee Q)$, $(P \wedge Q \wedge \neg Q')$, ...
- Quantification:

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

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:

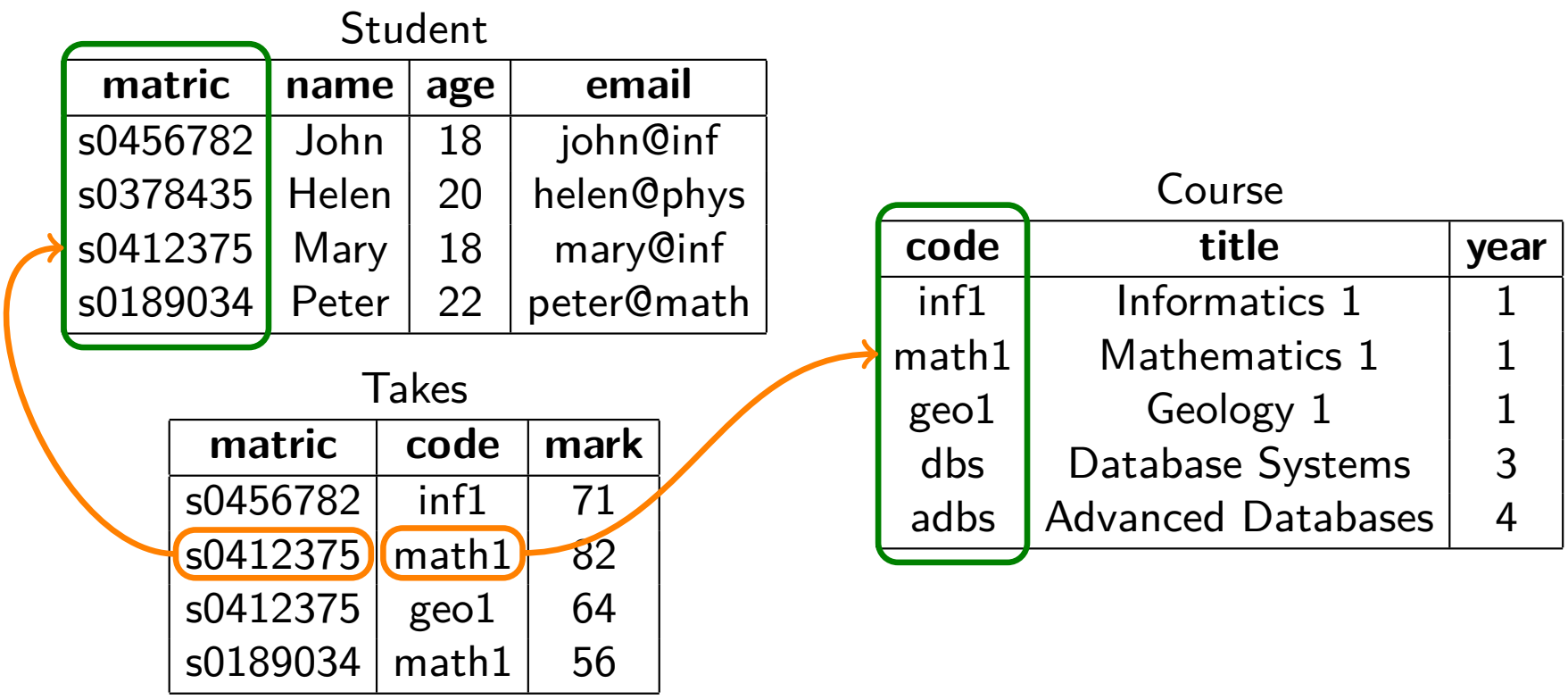
$$\exists S \in \text{Table} \,.\, P(S) \quad \text{to mean} \quad \exists S \,.\, S \in \text{Table} \wedge P(S)$$

Facebook Graph Search



Mark Zuckerberg (Credit: AP/Jeff Chiu)

Students and Courses



Students and Courses (1/5)

Students taking Geology 1

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

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:

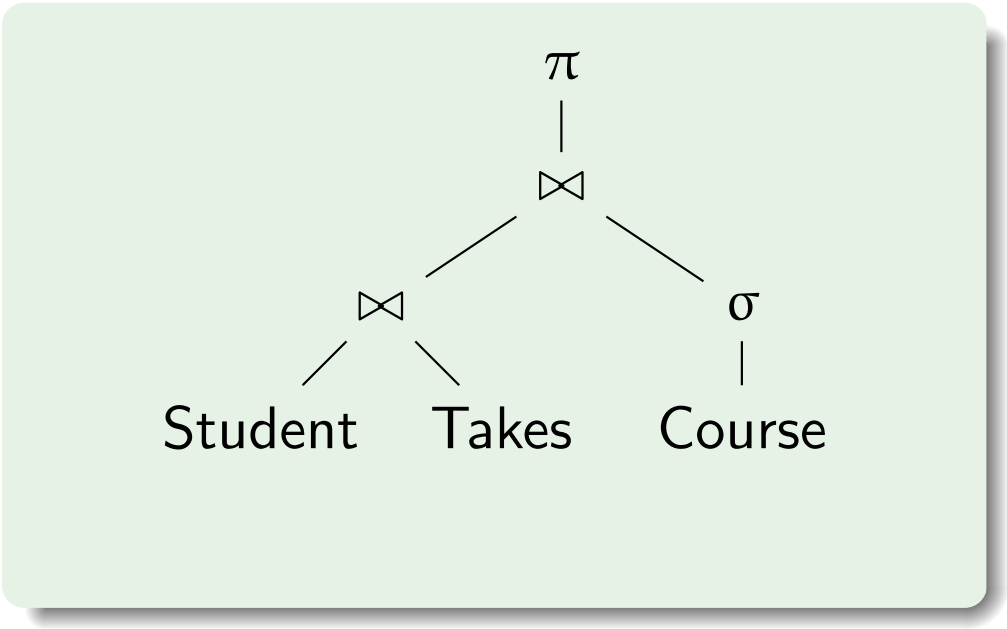
$$\pi_{\mathsf{name}}((\mathsf{Student} \bowtie \mathsf{Takes}) \bowtie (\sigma_{\mathsf{title}=\text{"Geology 1"}}(\mathsf{Course})))$$

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:

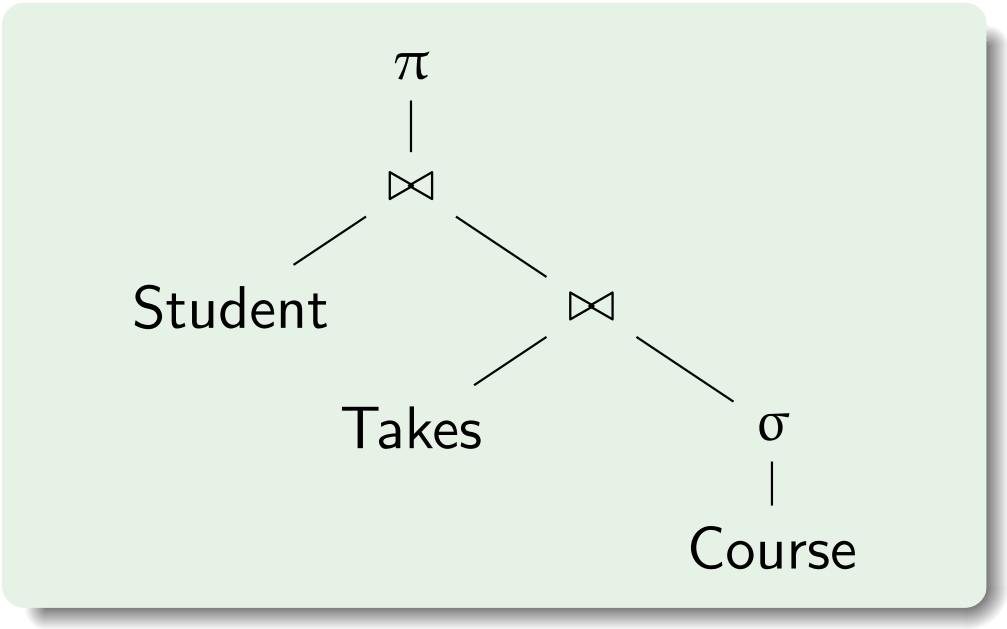


Relational Algebra

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:

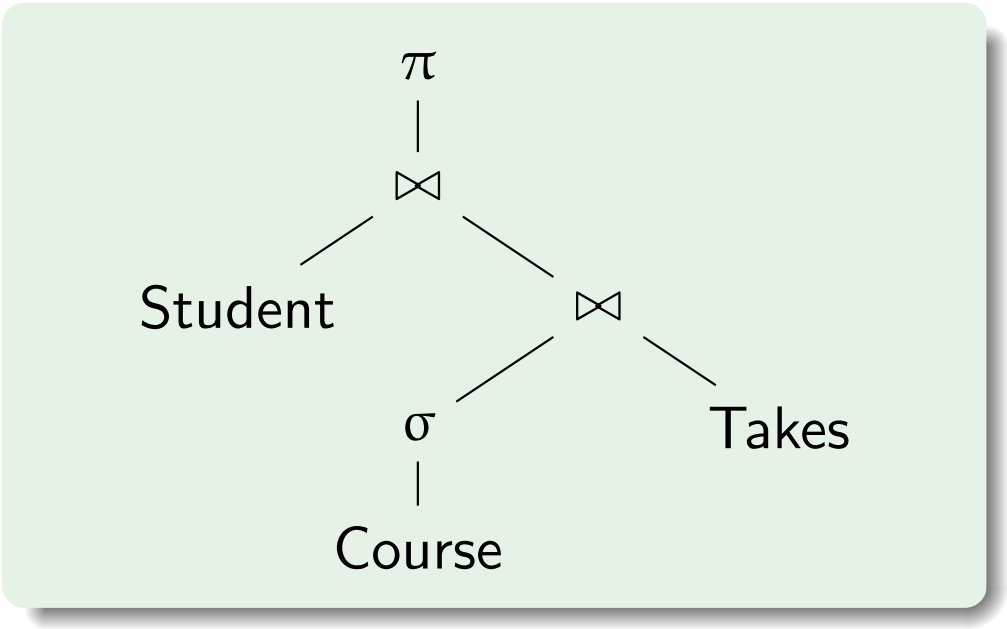


Relational Algebra

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

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

We can also visualise this as rearrangements of a tree:



Students and Courses (2/5)

Courses taken by students called "Ada"

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

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

Computing this in relational algebra:

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

Students and Courses (3/5)

Students taking Informatics 1 or Geology 1

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

Now the logical formula becomes a little more elaborate.

Computing this in relational algebra:

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

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

Students and Courses (4/5)

Students taking both Informatics 1 and Geology 1

$$\{ \mathrm{R} \mid \exists S \in \mathrm{Student}, \mathrm{T}_1, \mathrm{T}_2 \in \mathrm{Takes}, \mathrm{C}_1, \mathrm{C}_2 \in \mathrm{Course} \,.$$
$$\mathrm{C}_1.\mathrm{title} = \text{"Informatics 1"} \wedge \mathrm{C}_2.\mathrm{title} = \text{"Geology 1"}$$
$$\wedge\ \mathrm{C}_1.\mathrm{code} = \mathrm{T}_1.\mathrm{code} \wedge \mathrm{T}_1.\mathrm{matric} = S.\mathrm{matric}$$
$$\wedge\ \mathrm{C}_2.\mathrm{code} = \mathrm{T}_2.\mathrm{code} \wedge \mathrm{T}_2.\mathrm{matric} = S.\mathrm{matric} \wedge S.\mathrm{name} = \mathrm{R}.\mathrm{name} \}$$

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

$$\{ \mathrm{R} \mid \exists S \in \mathsf{Student} \,.\, S.\mathsf{name} = \mathrm{R}.\mathsf{name} \\ \wedge\ \forall \mathrm{T} \in \mathsf{Takes} \,.\, \mathrm{T}.\mathsf{matric} \neq S.\mathsf{matric} \}$$

Computing this in relational algebra:

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

$\star$ Challenge: why not one of these instead?

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

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

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

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;
- ...

Homework !

Read This

Inside Google Spanner, the Largest Single Database on Earth
Wired 2012-11-26
http://www.wired.com/2012/11/google-spanner-time/all/

Do This

Tuple-relational calculus can be quite tricky to understand, and it's not always obvious to follow what a query means. So, homework this time is to read the lectures slides again, going through the examples to see how each query works.