Informatics 1: Data & Analysis

Lecture 6: Tuple Relational Calculus

Ian Stark

School of Informatics
The University of Edinburgh

Thursday 31 January 2019
Semester 2 Week 3

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*

Homework From Tuesday

1. Read This

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

"...a database designed to seamlessly operate across hundreds of data centers and millions of machines and trillions of rows of information."

Optional: Find out more with Spanner, TrueTime and the CAP Theorem by Eric Brewer, Google, February 2017.

https://ai.google/research/pubs/pub45855

2. Do This

Work through Example 7 and Example 8 from Tutorial 1: do each question yourself and write out your answer; then look at the suggested solution.

Homework (1/2)

1. Read This

Read through either one of these study guides.

- Critical Thinking: *Making Notes in Lectures* and *Note-Making Styles*
  The University of Sussex
  https://www.sussex.ac.uk/skillshub/?id=306
  https://www.sussex.ac.uk/skillshub/?id=305

- Learning Essentials: Note Making
  The University of Manchester
  https://is.gd/manchesternotemaking

Homework (2/2)

2. Do This

Connect remotely to Informatics computing resources. You should work through setting up at least the following kinds of connection. Computing Helpdesk, AT 4.11, 2–4pm weekdays

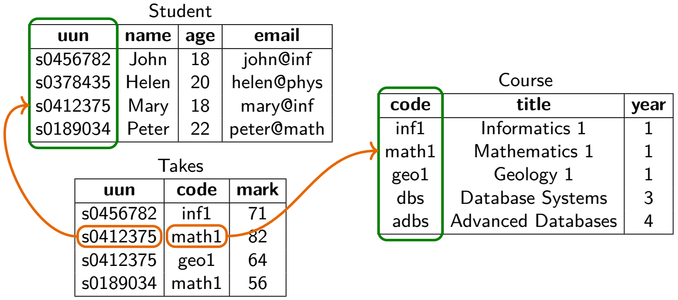
- To the file system — https://ifile.inf.ed.ac.uk
- To a DICE command line with ssh — http://computing.help.inf.ed.ac.uk/external-login
- To a server machine — ssh to ssh.inf.ed.ac.uk and again from there to student.compute
- Over the University VPN —
  https://www.ed.ac.uk/information-services/computing/desktop-personal/vpn
- To a DICE desktop — http://computing.help.inf.ed.ac.uk/remote-desktop

Optional: Set up remote authentication using Kerberos and file system access through AFS. Information on all of this is at https://computing.help.inf.ed.ac.uk

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

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

Takes

uun	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 9075:2016

Tuple Relational Calculus: Example Query

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

uun	**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 Query

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

uun	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 Query

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.

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

uun	**name**	**age**	**email**
s0378435	Helen	20	helen@phys
s0189034	Peter	22	peter@math

Tuple Relational Calculus: Example Query

All records for students more than 19 years old

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

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

The notation $\{\, S \mid \dots \}$ describes the set of "all S such that …".

This use of S at the start of the expression is a binder: it introduces the variable that appears later in the body, the "…" part.

This body, the part from "|" through to "}", is the scope of the binding. Every use of S within the scope refers back to that initial binding appearance.

Tuple Relational Calculus: Example Query

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 notation is known as set comprehension.

We can also write this expression as:

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

This is syntactic sugar: no change the to meaning, just more readable.

Tuple Relational Calculus: Example Query

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.

List comprehension in programming languages uses the same approach:

```
Haskell    [ s | s <- students, age s > 19 ]

Python    [ s for s in students if s.age > 19 ]
```

In both cases the first occurrence of "s" is a binder to introduce the variable.

Tuple Relational Calculus Basics

Queries in tuple relational calculus (TRC) all have this general form:

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

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

The fields of the tuple variable T, its schema, are calculated from the way it is used in the predicate $P(T)$.

The result of the query

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

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

Students and Courses (1/5)

All records for students taking math1

$$\{\, S \mid S \in \mathsf{Student} \wedge \exists \mathsf{T} \,.\, (\, \mathsf{T} \in \mathsf{Takes} \wedge \mathsf{T.code} = \texttt{"math1"} \wedge S.\mathsf{uun} = \mathsf{T.uun} \,)\, \}$$

The set of tuples S such that S is in the table “Students” and there is a tuple T in table “Takes” with “code” equal to “math1” and where S and T have the same “uun” field.

Here “$\exists$T .” is another binder, this time binding T in the expression body “($\mathsf{T} \in \mathsf{Takes} \ldots$)”.

The scope of this binding extends to the end of the expression in braces.

The statement “$\exists$T . . . .” is an existential quantification: “there is some T such that . . . ” where the “. . . ” describes requirements on T.

Students and Courses (1/5)

All records for students taking math1

$$\{\, S \mid S \in \mathsf{Student} \land \exists \mathsf{T}\,.\,(\, \mathsf{T} \in \mathsf{Takes} \land \mathsf{T.code} = \texttt{"math1"} \land S.\mathsf{uun} = \mathsf{T.uun}\,)\,\}$$

The set of tuples S such that S is in the table “Students” and there is a tuple T in table “Takes” with “code” equal to “math1” and where S and T have the same “uun” field.

Student

uun	**name**	**age**	**email**
s0456782	John	1	john@inf
s0378435	Helen	20	helen@phys
s0412375	Mary	18	mary@inf
s0189034	Peter	22	peter@math

Takes

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

Students and Courses (1/5)

All records for students taking math1

$$\{\, S \mid S \in \mathsf{Student} \wedge \exists \mathsf{T}\,.\,(\,\mathsf{T} \in \mathsf{Takes} \wedge \mathsf{T.code} = \text{"math1"} \wedge S.\mathsf{uun} = \mathsf{T.uun}\,)\,\}$$

The set of tuples S such that S is in the table "Students" and there is a tuple T in table "Takes" with "code" equal to "math1" and where S and T have the same "uun" field.

Student

uun	name	age	email
s0456782	John	1	john@inf
s0378435	Helen	20	helen@phys
s0412375	Mary	18	mary@inf
s0189034	Peter	22	peter@math

Takes

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

Students and Courses (1/5)

All records for students taking math1

$$\{\, S \mid S \in \mathsf{Student} \wedge \exists \mathsf{T}\,.\,(\,\mathsf{T} \in \mathsf{Takes} \wedge \mathsf{T.code} = \texttt{"math1"} \wedge S.\mathsf{uun} = \mathsf{T.uun}\,)\,\}$$

The set of tuples S such that S is in the table “Students” and there is a tuple T in table “Takes” with “code” equal to “math1” and where S and T have the same “uun” field.

Student

uun	**name**	**age**	**email**
s0456782	John	1	john@inf
s0378435	Helen	20	helen@phys
s0412375	Mary	18	mary@inf
s0189034	Peter	22	peter@math

Takes

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

Students and Courses (1/5)

All records for students taking math1

$$\{\, S \mid S \in \mathsf{Student} \wedge \exists \mathsf{T}\,.\,(\,\mathsf{T} \in \mathsf{Takes} \wedge \mathsf{T.code} = \texttt{"math1"} \wedge S.\mathsf{uun} = \mathsf{T.uun}\,)\,\}$$

The set of tuples S such that S is in the table "Students" and there is a tuple T in table "Takes" with "code" equal to "math1" and where S and T have the same "uun" field.

$$\{\, S \mid S \in \mathsf{Student} \wedge \exists \mathsf{T}\,.\,(\mathsf{T} \in \mathsf{Takes} \wedge \mathsf{T.code} = \texttt{"Inf1"} \wedge S.\mathsf{uun} = \mathsf{T.uun})\,\}$$

uun	**name**	**age**	**email**
s0412375	Mary	18	mary@inf
s0189034	Peter	22	peter@math

Students and Courses (1/5)

All records for students taking math1

$$\{\, S \mid S \in \mathsf{Student} \wedge \exists \mathsf{T}\,.\,(\, \mathsf{T} \in \mathsf{Takes} \wedge \mathsf{T.code} = \texttt{"math1"} \wedge S.\mathsf{uun} = \mathsf{T.uun}\,)\,\}$$

The set of tuples S such that S is in the table "Students" and there is a tuple T in table "Takes" with "code" equal to "math1" and where S and T have the same "uun" field.

Again there is a syntactically sugared alternative:

$$\{\, S \in \mathsf{Student} \mid \exists \mathsf{T} \in \mathsf{Takes}\,.\,\mathsf{T.code} = \texttt{"Inf1"} \wedge S.\mathsf{uun} = \mathsf{T.uun}\,\}$$

This time there are also no parentheses "$(\ldots)$". It makes no difference here as the scope of the existential always extends to the end of the enclosing braces.

Students and Courses (1/5)

All records for students taking math1

$$\{\, S \mid S \in \text{Student} \wedge \exists \mathrm{T} \,.\, (\, \mathrm{T} \in \text{Takes} \wedge \mathrm{T}.\text{code} = \texttt{"math1"} \wedge S.\text{uun} = \mathrm{T}.\text{uun} \,)\,\}$$

The set of tuples S such that S is in the table "Students" and there is a tuple T in table "Takes" with "code" equal to "math1" and where S and T have the same "uun" field.

We can deduce the schemas for each variable by their use in the expression.

- Tuple variable S has a schema to match the table "Student".
- Tuple variable T has a schema to match the table "Takes".

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:

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

Symbol Soup

Quantifiers $\forall$, $\exists$

Universal	FORALL	$\forall$	!	$\forall x.P(x)$	$\forall x\ P(x)$	$\forall x(P(x))$
Existential	EXISTS	$\exists$	?	$\exists x.P(x)$	$\exists x\ P(x)$	$\exists x(P(x))$

Both $\forall$ and $\exists$ are binders that *bind* the named variable in the *body* of the expression: everything to the right of the binder.

Syntactic Sugar

$\forall x \in A \,.\, P(x)$	for	$\forall x.(x \in A) \wedge P(x)$	$\forall x, y, z.P(x, y, z)$	for	$\forall x.(\forall y.(\forall z.P(x, y, z)))$
$\exists x \in A \,.\, P(x)$	for	$\exists x.(x \in A) \wedge P(x)$	$\exists x, y, z.P(x, y, z)$	for	$\exists x.(\exists y.(\exists z.P(x, y, z)))$

Symbol Soup

Quantifiers $\forall$, $\exists$

Universal	FORALL	$\forall$	!	$\forall x . P(x)$	$\forall x\ P(x)$	$\forall x (P(x))$
Existential	EXISTS	$\exists$	?	$\exists x . P(x)$	$\exists x\ P(x)$	$\exists x (P(x))$

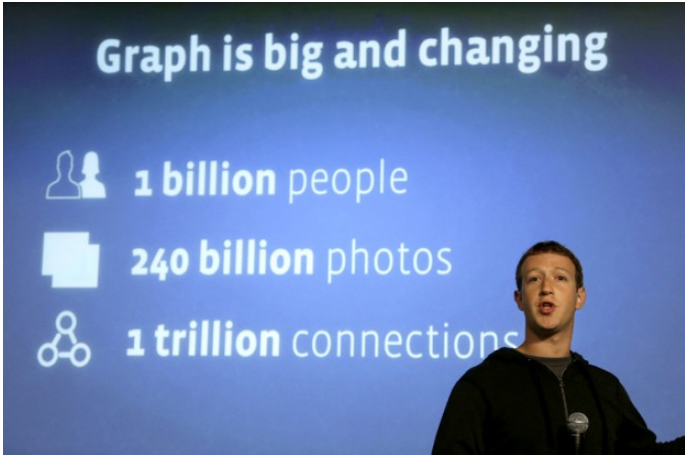
Both $\forall$ and $\exists$ are binders that *bind* the named variable in the *body* of the expression: everything to the right of the binder.

Quantification examples

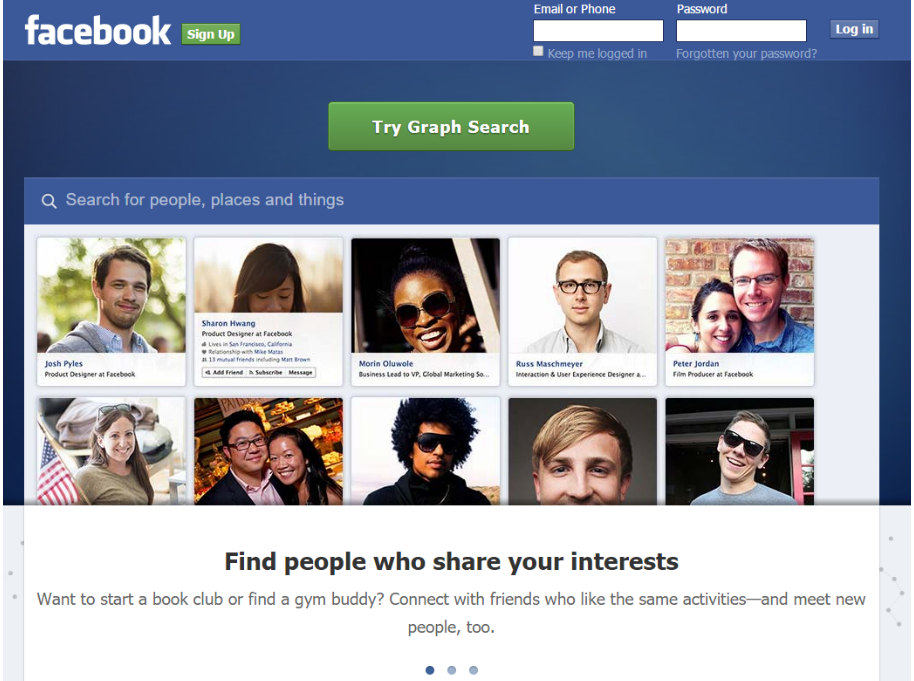
$$\forall k\ .\ (k > 4) \Rightarrow (k > 2) \qquad \exists n\ (n^2 + n = 30)$$

$$\forall x \in \mathbb{N}\ .\ x^2 \geqslant x \qquad \forall x, y, z \in \mathbb{R}\ (\ x \geqslant y \wedge y \geqslant z \Rightarrow x \geqslant z\ )$$

Facebook Graph Search



Mark Zuckerberg (Credit: AP/Jeff Chiu)



Josh Pyles
Product Designer at Facebook

Sharon Hwang
Product Designer at Facebook
Lives in San Francisco, California
Relationship with Mike Matas
13 mutual friends including Matt Brown

Morin Oluwole
Business Lead to VP, Global Marketing So...

Russ Maschmeyer
Interaction & User Experience Designer a...

Peter Jordan
Film Producer at Facebook

Find people who share your interests

Want to start a book club or find a gym buddy? Connect with friends who like the same activities—and meet new people, too.

Some Queries on Facebook Graph Search

https://www.facebook.com/search/4/photos-tagged

https://www.facebook.com/search/4/places-visited/273819889375819/places/intersect

https://www.facebook.com/search/str/moscow/pages-named/residents/present/intersect/str/new%20york/pages-named/visitors/intersect

https://www.facebook.com/search/str/Google/pages-named/employees/past/intersect/str/Facebook/pages-named/employees/present/intersect

https://www.facebook.com/search/str/Facebook/pages-named/employees/friends/str/Google/pages-named/employees/intersect/str/paris/pages-named/residents/intersect

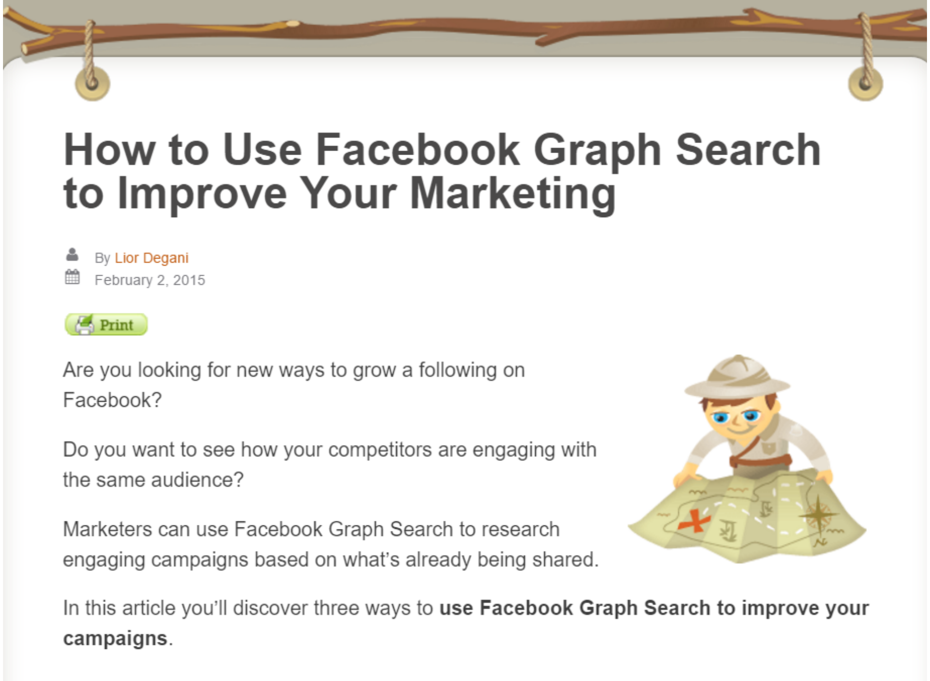
Try for yourself: http://www.intel-sw.com/blog/facebook-search/

How to Use Facebook's Graph Search to Supercharge Your Professional Network

2.5k
SHARES

Share on Faceboo
Share on Twitte
+

WHAT'S THIS?



How to Use Facebook Graph Search to Improve Your Marketing

By Lior Degani
February 2, 2015

Print

Are you looking for new ways to grow a following on Facebook?

Do you want to see how your competitors are engaging with the same audience?

Marketers can use Facebook Graph Search to research engaging campaigns based on what's already being shared.

In this article you'll discover three ways to **use Facebook Graph Search to improve your campaigns**.

17 Ways Marketers Can Leverage Facebook Graph Search

By Christian Karasiewicz
July 22, 2013

Print

Are you using Facebook Graph Search as a marketing power tool?

If you knew more about your Facebook fans, how would that improve your marketing efforts?

Marketers can now **use Facebook Graph Search to perform focused searches** that will return better, more accurate search results that can be of real use to your business.

What Can I Do With Facebook Graph Search?

With Graph Search, you can do much more than meets the eye.

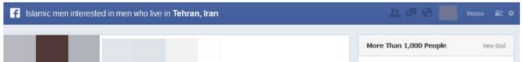
INTERNET AND NEW MEDIA BLOG

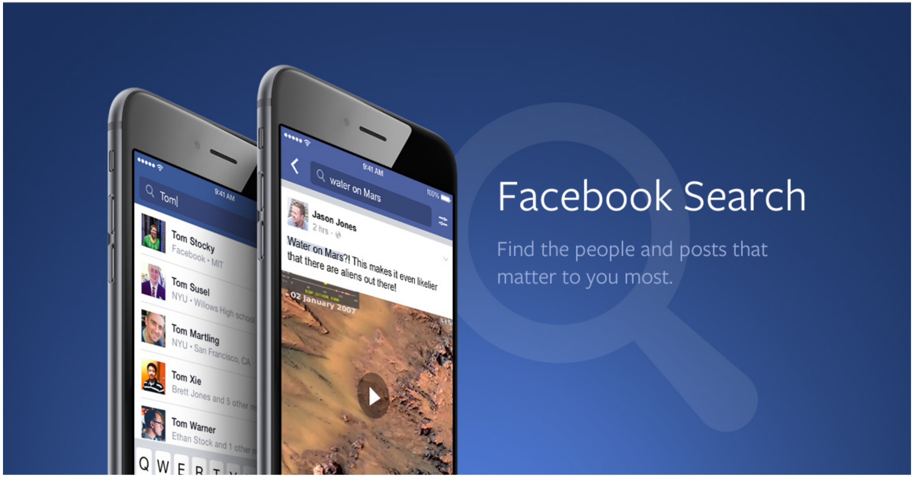
Why Graph Search Could Be Facebook's Largest Privacy Invasion Ever

Facebook's new program will allow anyone to search for, say, "Chinese residents who have family members that like Falun Gong."

By Ari Melber

JANUARY 27, 2013





Search is now better than ever

See what the world is saying right now about the topics that matter to you, with results that are up-to-the-minute and personalized.

How It Works

Unicorn: A System for Searching the Social Graph
2013 International Conference on Very Large Data Bases (VLDB)

Mike Curtiss, Iain Becker, Tudor Bosman, Sergey Doroshenko, Lucian Adrian Grijincu, Tom Jackson, Soren Lassen, Philip Pronin, Guanghao Shen, Gintaras Woss, Chao Yang, Ning Zhang, Sriram Sankar

https://research.fb.com/publications/unicorn-a-system-for-searching-the-social-graph

"Unicorn is an online, in-memory social graph-aware indexing system designed to search trillions of edges between tens of billions of users and entities on thousands of commodity servers."

"Unicorn is designed to answer billions of queries per day at latencies in the hundreds of milliseconds."

Another Example

Names and ages of all students over 19

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

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

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

$\{\, \mathsf{T} \mid \ldots \,\}$

name	age
Helen	20
Peter	22

Another Example

Names and ages of all students over 19

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

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”.

Syntactically sugaring things a little:

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

Another Example

Names and ages of all students over 19

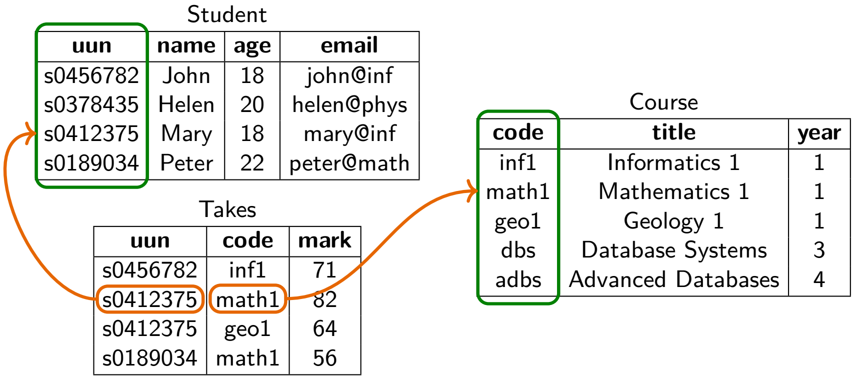
$$\{\, \mathrm{T} \mid \exists S \,.\, S \in \mathrm{Student} \;\wedge\; S.\mathrm{age} > 19 \;\wedge\; \mathrm{T.name} = S.\mathrm{name} \;\wedge\; \mathrm{T.age} = S.\mathrm{age} \,\}$$

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”.

We can deduce the schemas for each variable by their use in the expression.

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

Students and Courses



Student

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

Takes

uun	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

Students and Courses (2/5)

Names of students taking Geology 1

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

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 in the expression.

One way to compute this in relational algebra:

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

Relational Algebra

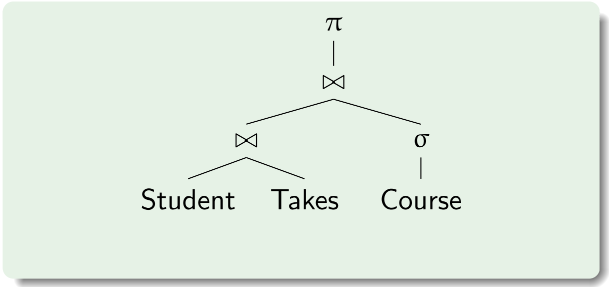
The relational algebra expression can be rearranged without changing its value, but possibly affecting the time and memory needed for computation:

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

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

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

We can also visualise this as rearrangements of a tree:

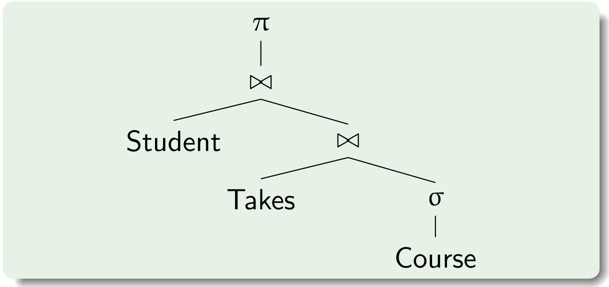


Relational Algebra

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

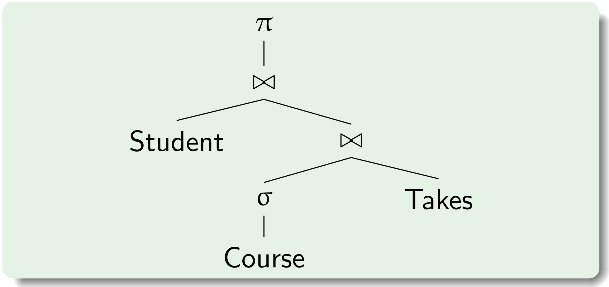
The relational algebra expression can be rearranged without changing its value, but possibly affecting the time and memory needed for computation:

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

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

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

We can also visualise this as rearrangements of a tree:



Students and Courses (3/5)

Titles of courses taken by students called "Ada"

$$\{\, R \mid \exists S \in \mathsf{Student}, T \in \mathsf{Takes}, C \in \mathsf{Course}\ .\ S.\mathsf{name} = \texttt{"Ada"} \wedge S.\mathsf{uun} = T.\mathsf{uun} \wedge T.\mathsf{code} = C.\mathsf{code} \wedge C.\mathsf{title} = R.\mathsf{title} \,\}$$

Note the syntactic sugar here for multiple quantification: we use comma-separated $\exists \ldots, \ldots, \ldots$ instead of $\exists \ldots \exists \ldots \exists \ldots$

Computing this in relational algebra:

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

Students and Courses (4/5)

Names of students taking Informatics 1 or Geology 1

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

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 (5/5)

Names of students taking both Informatics 1 and Geology 1

$$\begin{aligned} \{\, R \mid \exists S \in \text{Student}, T_1, T_2 \in \text{Takes}, C_1, C_2 \in \text{Course}\;. \\ C_1.\text{title} = \text{"Informatics 1"} \;\wedge\; C_2.\text{title} = \text{"Geology 1"} \\ \wedge\; C_1.\text{code} = T_1.\text{code} \;\wedge\; T_1.\text{uun} = S.\text{uun} \\ \wedge\; C_2.\text{code} = T_2.\text{code} \;\wedge\; T_2.\text{uun} = S.\text{uun} \;\wedge\; S.\text{name} = R.\text{name} \,\} \end{aligned}$$

Computing this in relational algebra:

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

But this definitely doesn't give the right answer:

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

Tuple Relational Calculus vs. Relational Algebra

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

So why have both?

Why have languages at all? Why not just write a program?
Because *Domain-Specific Languages* are awesome.

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 relational 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;
- ...

Summary

Tuple Relational Calculus

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

$$\{\, R \mid \exists S \in \text{Student}, T \in \text{Takes}, C \in \text{Course} \ldots . \,\}$$

Set comprehension and logical quantifiers

Notation like $\{\, R \mid \ldots \,\}$ and $(\exists S \ldots . \,)$ can bind the variables R and S in the body of an expression to define more complex sets and logical expressions.

Relational Algebra

Queries in TRC can be compiled into expressions of relational algebra. These guide computation: and a query optimizer will use algebraic rearrangement to make that computation efficient.