Informatics 1: Data & Analysis

Lecture 5: Relational Algebra

Ian Stark

School of Informatics
The University of Edinburgh

Tuesday 28 January 2014
Semester 2 Week 3

Support (1/3)

If you have questions about something in the lectures, difficulties with tutorial exercises, or want to find out more on the material, ask someone.

- Other students: in your tutorial group, in the lab, elsewhere.
- InfBASE: drop in Monday–Thursday 1600–1800 in AT 5.02
- Your course tutor: in person at your tutorials, or by email.
- The lecturer, Ian Stark: in person after lectures, drop-in office hour IF 5.04 1030–1130 every Wednesday, or by email.
- The course TA, Areti Manataki: AT 5.02 1630–1730 every Tuesday.
- Online: NB; in the discussion group; IRC #inf1; Facebook, etc.

Support 2/3

Here are details for some of these online resources:

NB Collaborative annotation, questions and answers. Follow links from course web page or subscribe at http://is.gd/inf1_da_nb

Forum Anything from Inf1, requires EASE login. http://is.gd/inf1_forum

IRC Chatroom on student-run server: #inf1 at irc.imaginarynet.org.uk

Facebook Informatics - Class of 2017 - University of Edinburgh
https://www.facebook.com/groups/uoeinformatics2017/

Support 3/3

For technical support when machines aren't working or you have problems with software on DICE, fill out the computing support form.

http://computing.help.inf.ed.ac.uk

For administrative support in anything related to teaching, contact the Informatics Teaching Organisation (ITO) by filling out their online contact form, or go to the ITO office on floor 4 of Appleton Tower.

http://www.inf.ed.ac.uk/teaching/contact

If you are having difficulties affecting all of your courses, or issues arising outside the University, contact your personal tutor.

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. relationa**l**

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*

Remember Relations as Tables?

Relational databases take as fundamental the idea of a *relation*, comprising a *schema* and an *instance*.

Fields (a.k.a. attributes, columns)

Schema

Tuples (a.k.a. records, rows)

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

Absolutely everything in a relational database is built from relations and operations upon them.

Every relational database is a linked collection of several tables like this: often much wider, and sometimes very, very much longer.

Languages for Working with Relations

Once we have a quantity of structured data in the linked tables of a relational model we may want to rearrange it, build new data structures, and extract information through the use of *queries*.

To understand how this is done, we'll look at three interlinked languages:

Relational Algebra

High-level mathematical operations for combining and processing relational tables.

Tuple-Relational Calculus

A declarative mathematical notation for expressing queries over structured data.

SQL

The standard programming language for writing queries on relational databases.

Relational Algebra

Relational algebra is a high-level mathematical language for describing certain operations on the schemas and tables of a relational model. Each of these operations takes one or more tables, and returns another.

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

Ted Codd gave a *completeness* proof showing that these operations were enough to express very general kinds of query: so, with an efficient implementation of these operations, you can answer all those queries.

Conversely, Codd's result also shows that to implement any expressive query language requires finding ways to carry out all of these operations.

Selection and Projection

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

Students

name	age
John	18
Mary	18
Helen	20
Peter	22

$\pi_{\text{name, age}}(\text{Students})$

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

$\sigma_{\text{age}>18}(\text{Students})$

name	age
Helen	20
Peter	22

Combination

Selection picks out the rows of a table satisfying a logical predicate

Selection and Projection

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

Students

name	age
John	18
Mary	18
Helen	20
Peter	22

$\pi_{\text{name, age}}(\text{Students})$

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

$\sigma_{\text{age}>18}(\text{Students})$

name	age
Helen	20
Peter	22

Combination

Projection picks out the columns of a table by their field name.

Selection and Projection

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

Students

name	age
John	18
Mary	18
Helen	20
Peter	22

$\pi_{\text{name, age}}(\text{Students})$

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

$\sigma_{\text{age}>18}(\text{Students})$

name	age
Helen	20
Peter	22

Combination

Combining selection and projection picks out a rectangular subtable.

$$\pi_{\text{name,age}}(\sigma_{\text{age}>18}(\text{Students})) \;=\; \sigma_{\text{age}>18}(\pi_{\text{name,age}}(\text{Students}))$$

Definitions

Selection

Relation $\sigma_P(R)$ is the table of rows in R which satisfy *predicate* P.

Thus $\sigma_P(R)$ has the same schema as R, but possibly lower cardinality.

Predicates like P, Q, ... are made up of

- Assertions about field values: (age $>$ 18), (degree $=$ "CS"), ...
- Logical combinations of these: $(P \vee Q)$, $(P \wedge Q \wedge \neg Q')$, ...

Projection

Relation $\pi_{a_1,\ldots,a_n}(R)$ is the table of all tuples of the fields $a_1, \ldots, a_n$ taken from the rows of R.

Thus $\pi_{a_1,\ldots,a_n}(R)$ usually has a lower-arity schema than R, and may also have lower cardinality.

Colour Coding of Slides

Regular Substantive Slide

Selection

Relation $\sigma_P(R)$ is the table of rows in R which satisfy *predicate* P.

Thus $\sigma_P(R)$ has the same schema as R, but possibly lower cardinality.

Predicates like P, Q, ... are made up of

- Assertions about field values: $(\text{age} > 18)$, $(\text{degree} = \text{"CS"})$, ...
- Logical combinations of these: $(P \vee Q)$, $(P \wedge Q \wedge \neg Q')$, ...

Projection

Relation $\pi_{a_1,\dots,a_n}(R)$ is the table of all tuples of the fields $a_1, \dots, a_n$ taken from the rows of R.

Thus $\pi_{a_1,\dots,a_n}(R)$ usually has a lower-arity schema than R, and may also have lower cardinality.

/ January 27, 2014

Announcement Slide !

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

/ January 27, 2014

Bonus Off-Syllabus Slide +

Logical Operators

Truth	TRUE	$\top$	T	tt	1			
Falsity	FALSE	$\perp$	F	ff	0			
Conjunction	AND	$P \wedge Q$	&	&&	$*$	$\cap$	$\bullet$	.
Disjunction	OR	$P \vee Q$	|	||	$+$	$\cup$		
Implication	IMPLIES	$P \Rightarrow Q$			$\rightarrow$	$\supset$		
Equivalence	IFF	$P \Leftrightarrow Q$			$\leftrightarrow$	$\equiv$		
Negation	NOT	$\neg P$	!	~				

/ January 27, 2014

Logical Operators

Truth	TRUE	$\top$	T	tt	1			
Falsity	FALSE	$\bot$	F	ff	0			
Conjunction	AND	$P \wedge Q$	&	&&	$*$	$\cap$	$\bullet$	.
Disjunction	OR	$P \vee Q$	$\mid$	$\|$	$+$	$\cup$		
Implication	IMPLIES	$P \Rightarrow Q$			$\rightarrow$	$\supset$		
Equivalence	IFF	$P \Leftrightarrow Q$			$\leftrightarrow$	$\equiv$		
Negation	NOT	$\neg P$	!	$\sim$				

Symbol Soup (2/2)

Quantifiers $\forall, \exists$

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

Set Comprehension

$$\{\, x \mid P(x) \,\} \quad \{\, x \mid x \in A \wedge P(x) \,\} \quad \{\, x \in A \mid P(x) \,\} \quad \{\, x : A \mid P(x) \,\}$$

Compare Haskell list comprehension `[ x | x <- [1..20], even x]`.

Brackets

$\Big(\quad\Big)$	Parentheses	Round brackets
$\Big[\quad\Big]$	Brackets	Square brackets
$\Big\{\quad\Big\}$	Braces	Curly brackets
$\Big\langle\quad\Big\rangle$	Chevrons	Angle brackets

Renaming

Students

mn	*name*	*age*	*email*

new table name

$$\rho_{S(mn \rightarrow sid,\ email \rightarrow address)} Students$$

renaming list

S

sid	*name*	*age*	*address*

Renaming changes the names of some or all fields in a table, giving a schema of the same arity and type.

This can be used to avoid *naming conflicts* when combining tables.

Union

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

S_1

mn	*name*	*age*	*email*
s0489967	Basil	19	basil@inf
s0412375	Mary	18	mary@inf
s9989232	Ophelia	24	oph@bio
s0189034	Peter	22	peter@math
s0289125	Michael	21	mike@geo

S_2

mn	*name*	*age*	*email*
s0456782	John	18	john@inf
s0412375	Mary	18	mary@inf
s0378435	Helen	20	helen@phys
s0189034	Peter	22	peter@math
s0489967	Basil	19	basil@inf
s9989232	Ophelia	24	oph@bio
s0289125	Michael	21	mike@geo

$S_1 \cup S_2$

Union combines the rows of two tables that have the same schema.

Difference

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

S_1

mn	*name*	*age*	*email*
s0489967	Basil	19	basil@inf
s0412375	Mary	18	mary@inf
s9989232	Ophelia	24	oph@bio
s0189034	Peter	22	peter@math
s0289125	Michael	21	mike@geo

S_2

mn	*name*	*age*	*email*
s0456782	John	18	john@inf
s0378435	Helen	20	helen@phys

S_1-S_2

Difference takes all the rows of one table which do not appear in another.

Intersection

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

S_1

mn	*name*	*age*	*email*
s0489967	Basil	19	basil@inf
s0412375	Mary	18	mary@inf
s9989232	Ophelia	24	oph@bio
s0189034	Peter	22	peter@math
s0289125	Michael	21	mike@geo

S_2

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

$S_1 \cap S_2$

Intersection takes all the rows of one table which do appear in another.

$$S_1 \cap S_2 = S_1 - (S_1 - S_2)$$

Definitions

Union

Relation $R_1 \cup R_2$ contains every tuple that appears in either R_1 or R_2.

Difference

Relation $R_1 - R_2$ contains every tuple that appears R_1 but not in R_2.

Intersection

Relation $R_1 \cap R_2$ contains every tuple that appears in R_1 and also in R_2.

In all of these cases the schemas of R_1 and R_2 must be *compatible* — all the same fields with all the same types.

Intersection can be defined in terms of difference, but not the other way around. (Try it and see)

Cross Product

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

S_1

code	*name*	*year*
inf1	Informatics 1	1
math1	Mathematics 1	1

R

mn	*name*	*age*	*email*	*code*	*name*	*year*
s0456782	John	18	john@inf	inf1	Informatics 1	1
s0456782	John	18	john@inf	math1	Mathematics 1	1
s0412375	Mary	18	mary@inf	inf1	Informatics 1	1
s0412375	Mary	18	mary@inf	math1	Mathematics 1	1
s0378435	Helen	20	helen@phys	inf1	Informatics 1	1
s0378435	Helen	20	helen@phys	math1	Mathematics 1	1
s0189034	Peter	22	peter@math	inf1	Informatics 1	1
s0189034	Peter	22	peter@math	math1	Mathematics 1	1

$S_1 \times R$

Cross product combines every row of one table with every row of another.

Definition

Cross product

For any relations R and S, the *cross product* $R \times S$, also known as the *Cartesian product*, is a relation defined as follows.

Schema

All the fields and types from R, plus all fields and types from S.
If necessary the renaming operation ρ can ensure none of these clash.

Rows

For every row $(u_1, \ldots, u_n)$ of R and every row $(v_1, \ldots, v_m)$ of S the product $R \times S$ contains row $(u_1, \ldots, u_n, v_1, \ldots, v_m)$.

The arity of $R \times S$ is the sum of the arities of R and S.

The cardinality of $R \times S$ is the product of the cardinalities of R and S.

Relational Join

The most commonly used relational operation is the *join* $R \bowtie_P S$ which combines cross-product with selection.

Rows in Join

For every row $(u_1, \ldots, u_n)$ of R and every row $(v_1, \ldots, v_m)$ of S the join relation $R \bowtie_P S$ contains row $(u_1, \ldots, u_n, v_1, \ldots, v_m)$ if and only if that tuple of values satisfies predicate P.

Here R and S are any two relations, with P any predicate defined on the fields of R and S together

$$R \bowtie_P S = \sigma_P(R \times S)$$

Example of Join

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

Students

mn	*code*	*mark*
s0412375	inf1	80
s0378435	math1	70

Takes

mn	*name*	*age*	*email*	*mn*	*code*	*mark*
s0456782	John	18	john@inf	s0412375	inf1	80
s0456782	John	18	john@inf	s0378435	math1	70
s0412375	Mary	18	mary@inf	s0412375	inf1	80
s0412375	Mary	18	mary@inf	s0378435	math1	70
s0378435	Helen	20	helen@phys	s0412375	inf1	80
s0378435	Helen	20	helen@phys	s0378435	math1	70
s0189034	Peter	22	peter@math	s0412375	inf1	80
s0189034	Peter	22	peter@math	s0378435	math1	70

$$\sigma_{Students.mn = Takes.mn}(Students \times Takes)$$

Example of Join

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

Students

mn	*code*	*mark*
s0412375	inf1	80
s0378435	math1	70

Takes

mn	*name*	*age*	*email*	*mn*	*code*	*mark*
s0456782	John	18	john@inf	s0412375	inf1	80
s0456782	John	18	john@inf	s0378435	math1	70
s0412375	Mary	18	mary@inf	s0412375	inf1	80
s0412375	Mary	18	mary@inf	s0378435	math1	70
s0378435	Helen	20	helen@phys	s0412375	inf1	80
s0378435	Helen	20	helen@phys	s0378435	math1	70
s0189034	Peter	22	peter@math	s0412375	inf1	80
s0189034	Peter	22	peter@math	s0378435	math1	70

$Students \bowtie_{Students.mn\,=\,Takes.mn} Takes$

Refined Joins

In general, a join $R \bowtie_P S$ can use an arbitrary predicate P.

However, some kinds of predicate are particularly common, and often followed by projection to eliminate duplicate or redundant columns.

Equijoin

An *equijoin* starts with a join where the predicate states that particular fields from each relation must be equal.

That is, P has the form $(a_1 = b_1) \wedge \cdots \wedge (a_k = b_k)$ for some fields $a_1, \ldots a_k$ of R and $b_1, \ldots, b_k$ of S.

For example, the relation $(\text{Students} \bowtie_{\text{Students.mn}=\text{Takes.mn}} \text{Takes})$ above.

The equijoin then projects onto all columns of the product except $b_1, \ldots, b_k$, as they now duplicate $a_1, \ldots, a_k$.

Natural Join

The *natural join* $R \bowtie S$ of relations R and S is the equijoin requiring equalities between any fields in the two relations that share the same name.

For example, the natural join of the "Students" and "Takes" relations:

$$\text{Students} \bowtie \text{Takes} =$$

$$\pi_{\substack{\text{mn,name,age,}\\ \text{email,code,mark}}}(\sigma_{\text{Students.mn}=\text{Takes.mn}}(\text{Students} \times \text{Takes}))$$

This records every student in combination with every course they take.

This example is typical: a natural join between two tables where one has a foreign key constraint referring to the other.

Specialist Joins

The SQL standard defines no less than five different types of join.

Inner Join is the basic join $R \bowtie_P S$ described earlier.

Left Outer Join is the basic join, plus rows for every tuple in the left-hand table R that matches nothing in the right-hand table S. Missing fields are filled with **NULL**.

Right Outer Join is the basic join plus rows for every tuple in the right-hand table S that matches nothing in the left-hand table R. Missing fields are filled with **NULL**.

Full Outer Join has every row from all three previous joins.

Cross Join is the cross-product $R \times S$, with every tuple from R paired with every tuple from S, and no matching done at all.