Informatics 1: Data & Analysis

Lecture 5: Relational Algebra

Ian Stark

School of Informatics
The University of Edinburgh

Tuesday 29 January 2013
Semester 2 Week 3

Announcement: Computing Clinic

Informatics Student Computing Clinic

2.30–3.30pm Wednesday 30 January

Appleton Tower Level 4 Open Area

Drop-in session with Computing Officers from the School of Informatics.

This is an open session — please bring along any IT questions or problems you may have:

- Using Linux and DICE
- Accessing DICE services from your own machine
- Working from home
- ⟨*your question here*⟩

The COs will do their best to provide an answer on the spot, or to direct you to people who can help.

Announcement: Student Experience Surveys

Several annual surveys have just opened, asking you to comment on you experience at the University.

- National Student Survey (NSS) — Final-year undergraduates
- Edinburgh Student Experience Survey (ESES) — All other undergraduates
- Postgraduate Taught Experience Survey (PTES) — Masters students
- Postgraduate Research Experience (PRES) — PhD students

You can fill out the appropriate survey online through MyEd.

Please do complete the survey: we use the results of these to help plan how to improve our teaching overall; and the more information we have, the better we can do that.

This is more general than the individual course feedback forms available at the end of each semester: those go directly to course lecturers.

Announcement: Student Experience Surveys

Several annual surveys have just opened, asking you to comment on you experience at the University.

- National Student Survey (NSS) — Final-year undergraduates
- Edinburgh Student Experience Survey (ESES) — All other undergraduates
- Postgraduate Taught Experience Survey (PTES) — Masters students
- Postgraduate Research Experience (PRES) — PhD students

You can fill out the appropriate survey online through MyEd.

Please do complete the survey: we use the results of these to help plan how to improve our teaching overall; and the more information we have, the better we can do that.

This is more general than the individual course feedback forms available at the end of each semester: those go directly to course lecturers.

Survey Scoring

Many of the questions on these surveys use a 5+1-point response scale:

> Please indicate the extent to which you agree or disagree with each of the following statements . . .

Definitely agree	Mostly agree	Neither agree nor disagree	Mostly disagree	Definitely disagree	Not applicable
○	○	○	○	○	○

Survey Scoring

Many of the questions on these surveys use a 5+1-point response scale:

> Please indicate the extent to which you agree or disagree with each of the following statements . . .

Definitely agree	Mostly agree	Neither agree nor disagree	Mostly disagree	Definitely disagree	Not applicable
○	○	○	○	○	○

However, most reported statistics use the *proportion of satisfied students.*

Measure of Student Satisfaction

"We report on the percentage of respondents that are satisfied; in other words the sum of Definitely agree and Mostly agree respondents, divided by the total number of respondents (defined as the sum of Definitely agree to Definitely disagree respondents) for that question or category of question."

National Student Survey
Findings and Trends 2006 to 2010

Survey Scoring

Many of the questions on these surveys use a 5+1-point response scale:

> Please indicate the extent to which you agree or disagree with each of the following statements . . .

Definitely agree	Mostly agree	Neither agree nor disagree	Mostly disagree	Definitely disagree	Not applicable
○	○	○	○	○	○

However, most reported statistics use the *proportion of satisfied students.*

Measure of Student Satisfaction

"We report on the percentage of respondents that are satisfied; in other words the sum of Definitely agree and Mostly agree respondents, divided by the total number of respondents (defined as the sum of Definitely agree to Definitely disagree respondents) for that question or category of question."

This has the effect of a *forced choice*: there is no 'meh'.

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*

Remember Relation as Tables?

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

Schema

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

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

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

Building Blocks

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

- The schema is the format of the relation:
  - A set of named *fields* (or *attributes* or *columns*)
  - For each field its *domain* (or *type*)
- The instance of a relation is a *table*:
  - A set of *rows* (or *records* or *tuples*)
  - Each row gives a value for every field, from the appropriate domain.
- The *arity* of a relation is the number of fields in its schema.
- The *cardinality* of a relation is the number of rows in its table.

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

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 $\bowtie$

Codd's key *completeness* proof showed 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 kinds of query.

Conversely, his result also shows that implementing any moderately expressive query language requires finding ways to perform 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 $\pi_{a_1,\dots,a_n}(R)$ is the table of all tuples of the attributes $a_1, \dots, a_n$ 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.

Renaming

Students

mn	name	age	email

new table name

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

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 use 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* — the same fields of the same types — and that is also the result schema.

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.

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, with 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 $a_1, \dots a_k$ of R and $b_1, \dots, b_k$ 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 $b_1, \dots, b_k$, as they now duplicate $a_1, \dots, a_k$.

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.