

## Part I — Structured Data

### Data Representation:

**I.1** The entity-relationship (ER) data model

**I.2** The relational model

### Data Manipulation:

**I.3 Relational algebra**

**I.4** Tuple relational calculus

**I.5** The SQL query language

Related reading: Chapter 4 of [DMS]: §§ 4.1,4.2

## Querying

Once data is organised in a relational schema, the natural next step is to *manipulate* that data. For our purposes, this means querying.

*Querying* is the process of identifying the parts of stored data that have properties of interest

We consider three approaches.

- **Relational algebra** (today's topic): a *procedural* way of expressing queries over relationally represented data
- **Tuple-relational calculus** (see I.4): a *declarative* way of expressing queries, tightly coupled to first order predicate logic
- **SQL** (see I.5): a widely implemented query language influenced by relational algebra and relational calculus

## Operators

The key concept in relational algebra is an *operator*

Operators accept a single relation or a pair of relations as input

Operators produce a single relation as output

Operators can be *composed* by using one operator's output as input to another operator (composition of functions)

There are five basic operators: *selection*, *projection*, *union*, *difference* and *cross-product*

From these fundamentals we can also define various other operators, like *intersection*, *renaming*, *join* and *equijoin*.

## Selection and projection: $\sigma$ and $\pi$

Recall that relational data is stored in *tables*

*Selection* and *projection* allow one to isolate any “rectangular subset” of a single table

- Selection identifies *rows* of interest
- Projection identifies *columns* of interest

If both are used on a single table, we extract a *rectangular subset* of the table

## Selection: example

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

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

name	age
John	18
Mary	18
Helen	20
Peter	22

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

name	age
Helen	20
Peter	22

Combination

## Selection: general form

General form:  $\sigma_{\text{predicate}}(\text{Relation instance})$

A *predicate* is a condition that is applied on each row of the table

- It should evaluate to either true or false
- If it evaluates to true, the row is propagated to the output, if it evaluates to false the row is dropped
- The output table may thus have lower cardinality than the input

Predicates are written in the Boolean form

$$\text{term}_1 \text{ bop } \text{term}_2 \text{ bop } \dots \text{ bop } \text{term}_m$$

- Where  $\text{bop} \in \{\vee, \wedge\}$
- $\text{term}_i$ 's are of the form  $\text{attribute rop constant}$  or  $\text{attribute}_1 \text{ rop attribute}_2$  (where  $\text{rop} \in \{>, <, =, \neq, \geq, \leq\}$ )

## Projection: example

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: general form

General form:  $\pi_{\text{column list}}(\text{Relation instance})$

All rows of the input are propagated in the output

Only columns appearing in the *column list* appear in the output

Thus the *arity* of the output table may be lower than that of the input table

The resulting relation has a different schema!



## Selection and projection: example

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

Note the *algebraic equivalence* between:

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

## Set operations

There are three basic set operations in relational algebra:

- *union*
- *difference*
- *cross-product*

A fourth, *intersection*, can be expressed in terms of the others

All these set operations are binary.

Essentially, they are the well-known set operations from set theory, but extended to deal with tuples

## Union

Let  $R$  and  $S$  be two relations. For union, set difference and intersection  $R$  and  $S$  are required to have compatible schemata:

- Two schemata are said to be *compatible* if they have the same number of fields and corresponding fields in a left-to-right order have the same domains. N.B., the names of the fields are not used

The *union*  $R \cup S$  of  $R$  and  $S$  is a new relation with the same schema as  $R$ . It contains exactly the tuples that appear in at least one of the relations  $R$  and  $S$

**N.B.** For naming purposes it is assumed that the output relation inherits the field names from the relation appearing first in the specification ( $R$  in the previous case)

## Union example

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

 $S_1$ 

<i>mn</i>	<i>name</i>	<i>age</i>	<i>email</i>
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

 $S_2$ 

<i>mn</i>	<i>name</i>	<i>age</i>	<i>email</i>
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

 $S_1 \cup S_2$

## Set difference and intersection

The *set difference*  $R - S$  and *intersection*  $R \cap S$  are also new relations with the same schema as  $R$  and  $S$ .

$R - S$  contains exactly those tuples that appear in  $R$  but which do not appear in  $S$

$R \cap S$  contains exactly those tuples that appear in both  $R$  and  $S$

For both operations, the same naming conventions apply as for union

Note that intersection can be defined from set difference by

$$R \cap S = R - (R - S)$$

## Set difference example

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

 $S_1$ 

<i>mn</i>	<i>name</i>	<i>age</i>	<i>email</i>
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

 $S_2$ 

<i>mn</i>	<i>name</i>	<i>age</i>	<i>email</i>
s0456782	John	18	john@inf
s0378435	Helen	20	helen@phys

 $S_1 - S_2$

## Intersection example

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

 $S_1$ 

<i>mn</i>	<i>name</i>	<i>age</i>	<i>email</i>
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

 $S_2$ 

<i>mn</i>	<i>name</i>	<i>age</i>	<i>email</i>
s0412375	Mary	18	mary@inf
s0189034	Peter	22	peter@math

 $S_1 \cap S_2$

**Cross product** The *cross-product* (also known as the *Cartesian product*)

$R \times S$  of two relations  $R$  and  $S$  is a new relation where

- The schema of the relation is obtained by first listing all the fields of  $R$  (in order) followed by all the fields of  $S$  (in order).
- The resulting relation contains one tuple  $\langle r, s \rangle$  for each pair of tuples  $r \in R$  and  $s \in S$ . (Here  $\langle r, s \rangle$  denotes the tuple obtained by appending  $r$  and  $s$  together, with  $r$  first and  $s$  second.)

Note that if there is a field name common to  $R$  and  $S$  then two separate columns with this name appear in the cross-product schema, as defined above, causing a *naming conflict*.

**N.B.** The two relations need not have the same schema to begin with.



## Cross-product example

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

$S_1$

<i>code</i>	<i>name</i>	<i>year</i>
inf1	Informatics 1	1
math1	Mathematics 1	1

$R$

<i>mn</i>	<i>name</i>	<i>age</i>	<i>email</i>	<i>code</i>	<i>name</i>	<i>year</i>
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

$S_1 \times R$

## Renaming

The renaming operator changes the names of tables and columns.

This can be used to avoid *naming conflicts* when the application of an operator results in a schema with duplicate column names

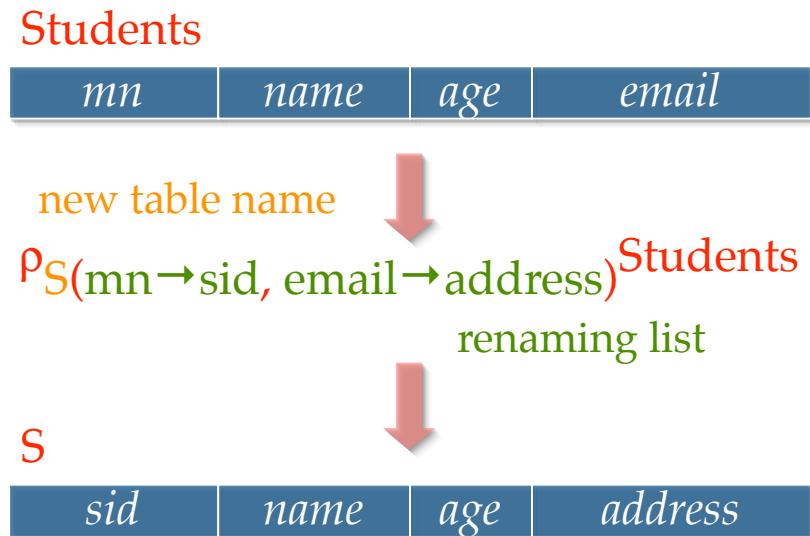
General form

$\rho_{\text{New-relation-name}(\text{renaming-list})}(\text{Original-relation-name})$

Semantics:

- The relation is assigned the new relation name
- The renaming list consists of terms of the form  $\text{oldname} \rightarrow \text{newname}$  which rename a field named  $\text{oldname}$  to  $\text{newname}$
- For  $\rho$  to be well-defined there should be no naming conflicts in the output

## Renaming example



N.B.

- The types of the columns do not change
- Either the renaming list, or the new table name may be empty

## Join

The *relational join*  $R \bowtie_p S$  is the most frequently used relational operator.

It is a *derived operator*, it can be defined in terms of cross-product and selection.

The format for a join is  $R \bowtie_p S$  where  $R$  and  $S$  are relations and the *join predicate*  $p$  is a predicate (as defined on slide 3.57) that applies to the schema of  $R \times S$ .

For example,  $p$  may have the form  $\text{col}_1 \text{ rop } \text{col}_2$  where  $\text{col}_1, \text{col}_2$  are columns of  $R, S$  and  $\text{rop} \in \{>, <, =, \neq, \geq, \leq\}$

Formally, the relational join is *defined* by:

$$R \bowtie_p S = \sigma_p(R \times S)$$

## Join example

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

*Students*

<i>mn</i>	<i>code</i>	<i>mark</i>
s0412375	inf1	80
s0378435	math1	70

*Takes*

<i>mn</i>	<i>name</i>	<i>age</i>	<i>email</i>	<i>mn</i>	<i>code</i>	<i>mark</i>
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

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

## Equijoin

An *equijoin* is a commonly occurring join operation in which the predicate is a conjunction of equalities of the form  $R.name_1 = S.name_2$ .

(A *conjunction* is a list of conditions connected by  $\wedge$ .)

The schema of the equijoin consists of the fields of  $R$ , followed by just those fields of  $S$  that are not mentioned in the join equalities. The equijoin is computed by *projecting* the join onto the fields that remain (all those of  $R$ , and those from  $S$  that have not been removed). Put more simply: remove from the join those columns labelled with  $S$ -fields that appear in the equalities.

Note that the example on the previous slide,

`Students`  $\bowtie_{\text{Students.mn} = \text{Takes.mn}}$  `Takes`, is naturally treated as an equijoin. The resulting relation is then as before, but with the second column labelled `mn` removed.

## Natural join

The *natural join* is a special equijoin in which the equalities are between *all* fields that have the same name in  $R$  and  $S$ .

We simply write  $R \bowtie S$  for such an equijoin.

Note that the equijoin version of the example on slide 3.72 is in fact the natural join  $\text{Students} \bowtie \text{Takes}$ . (The common field name is `mn`.) This is a very natural way of joining two relations, hence the name. It frequently occurs when joining two tables in which one has a foreign key constraint referencing the other.