VirginiaTech

CS 4604: Introduction to Database Management Systems

B. Aditya Prakash

Lecture #2: The Relational Model and Relational Algebra

Course Outline

- Weeks 1–4: Query/ Manipulation Languages and Data Modeling
  - Relational Algebra
  - Data definition
  - Programming with SQL
  - Entity-Relationship (E/R) approach
  - Specifying Constraints
  - Good E/R design
- Weeks 5–8: Indexes, Processing and Optimization
  - Storing
  - Hashing/Sorting
  - Query Optimization
  - NoSQL and Hadoop
- Week 9-10: Relational Design
  - Functional Dependencies
  - Normalization to avoid redundancy
- Week 11-12: Concurrency Control
  - Transactions
  - Logging and Recovery
- Week 13–14: Students' choice
  - Practice Problems
  - XML
  - Data mining and warehousing

Data Model

- A Data Model is a notation for describing data or information.
  - **Structure of data** (e.g. arrays, structs)
    - Conceptual model: In databases, structures are at a higher level.
  - **Operations on data** (Modifications and Queries)
    - Limited Operations: Ease of programmers and efficiency of database.
  - **Constraints on data** (what the data can be)

- Examples of data models
  - The Relational Model
  - The Semistructured-Data Model
    - XML and related standards
  - Object-Relational Model

The Relational Model

Student	Course	Grade
Hermione Grainger	Potions	A
Draco Malfoy	Potions	B
Harry Potter	Potions	A
Ron Weasley	Potions	C

- **Structure:** Table (like an array of structs)
- **Operations:** Relational alebgra (selection, projection, conditions, etc)
- **Constraints:** E.g., grades can be only {A, B, C, F}

The Semi-structured model

```
<CoursesTaken>
<Student>Hermione Grainger</Student>
<Course>Potions</Course>
<Grade>A</Grade>
<Student>Draco Malfoy</Student>
<Course>Potions</Course>
<Grade>B</Grade>
...
</CoursesTaken>
```

- **Structure:** Trees or graphs, tags define role played by different pieces of data.
- **Operations:** Follow paths in the implied tree from one element to another.
- **Constraints:** E.g., can express limitations on data types

Comparing the two models

- Flexibility: XML can represent graphs
- Ease of use: SQL enables programmer to express wishes at high level.

The Relational Model

- Simple: Built around a single concept for modeling data: the relation or table.
  - A relational database is a collection of relations.
  - Each relation is a table with rows and columns.
- Supports high-level programming language (SQL).
  - Limited but very useful set of operations
- Has an elegant mathematical design theory.
- Most current DBMS are relational (Oracle, IBM DB2, MS SQL)

Relations

- A relation is a two-dimensional table:
  - Relation == table.
  - Attribute == column name.
  - Tuple == row (not the header row).
- Database == collection of relations.
- A relation has two parts:
  - Schema defines column heads of the table (attributes).
  - Instance contains the data rows (tuples, rows, or records) of the table.

Student	Course	Grade
Hermione Grainger	Potions	A
Draco Malfoy	Potions	B
Harry Potter	Potions	A
Ron Weasley	Potions	C

Schema

CoursesTaken :

Student	Course	Grade
Hermione Grainger	Potions	A
Draco Malfoy	Potions	B
Harry Potter	Potions	A
Ron Weasley	Potions	C

- The schema of a relation is the name of the relation followed by a parenthesized list of attributes.

  `CoursesTaken(Student, Course, Grade)`

- A design in a relational model consists of a set of schemas.
- Such a set of schemas is called a relational database schema.

Relations: Equivalent Representations

CoursesTaken :

Student	Course	Grade
Hermione Grainger	Potions	A
Draco Malfoy	Potions	B
Harry Potter	Potions	A
Ron Weasley	Potions	C

`CoursesTaken(Student, Course, Grade)`

- Relation is a **set** of tuples and not a list of tuples.
  - Order in which we present the tuples does not matter.
  - Very important!
- The attributes in a schema are also a set (not a list).
  - Schema is the same irrespective of order of attributes.

    `CoursesTaken(Student, Grade, Course)`
  - We specify a "standard" order when we introduce a schema.
- How many equivalent representations are there for a relation with m attributes and n tuples?

m! n!

Degree and Cardinality

CoursesTaken :

Student	Course	Grade
Hermione Grainger	Potions	A
Draco Malfoy	Potions	B
Harry Potter	Potions	A
Ron Weasley	Potions	C

- Degree/Arity is the number of fields/attributes in schema (=3 in the table above)
- Cardinality is the number of tuples in relation (=4 in the table above)

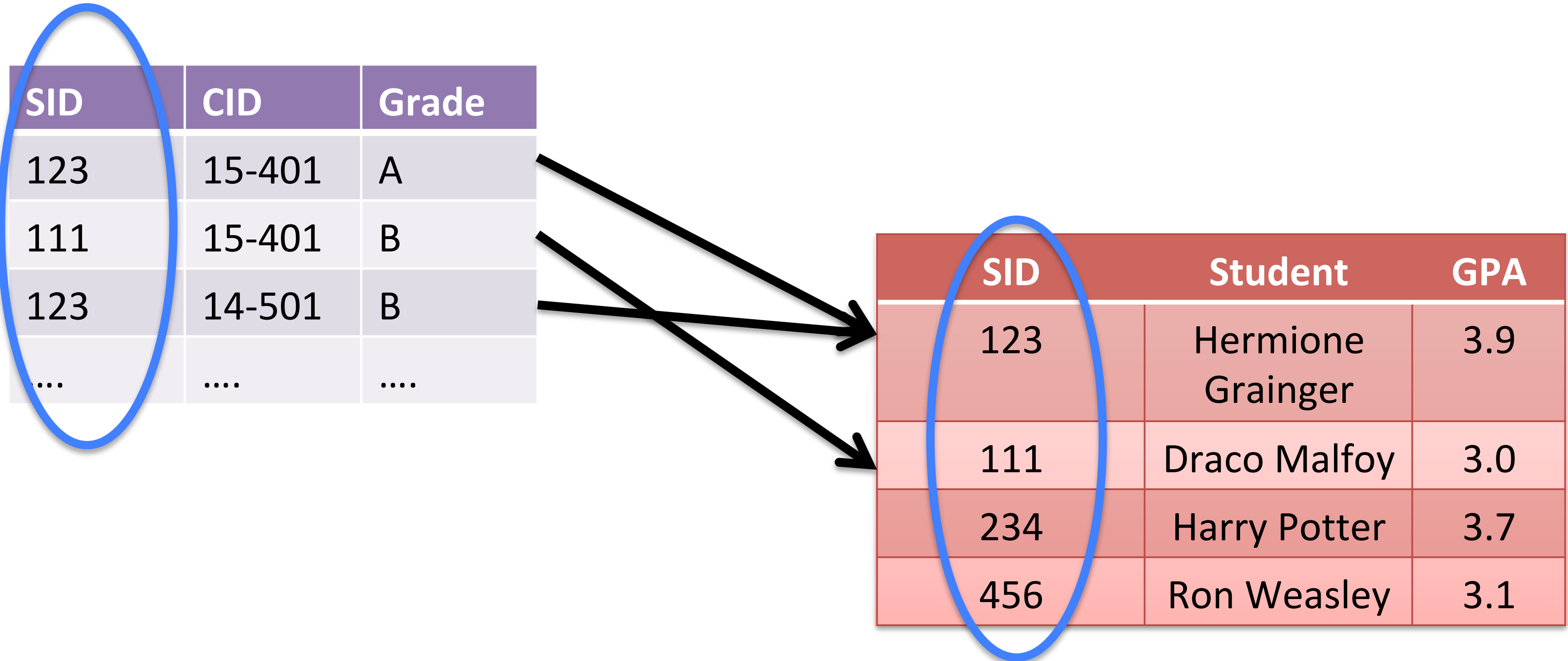
Keys of Relations

- Keys are one form of integrity constraints (IC)
  – No pair of tuples should have identical keys
- What is the key for CoursesTaken?
  – *Student* if only one course in the relation
  – Pair *(Student, Course)* if multiple courses
  – What if student takes same course many times?

Student	Course	Grade
Hermione Grainger	Potions	A
Draco Malfoy	Potions	B
Harry Potter	Potions	A
Ron Weasley	Potions	C

Keys of Relations

- Keys help associate tuples in different relations



Example

- Create a database for managing class enrollments in a single semester. You should keep track of all students (their names, Ids, and addresses) and professors (name, Id, department). Do not record the address of professors but keep track of their ages. Maintain records of courses also. Like what classroom is assigned to a course, what is the current enrollment, and which department offers it. At most one professor teaches each course. Each student evaluates the professor teaching the course. Note that all course offerings in the semester are unique, i.e. course names and numbers do not overlap. A course can have ≥ 0 pre-requisites, excluding itself. A student enrolled in a course must have enrolled in all its pre-requisites. Each student receives a grade in each course. The departments are also unique, and can have at most one chairperson (or dept. head). A chairperson is not allowed to head two or more departments.

Example

- Create a database for managing class enrollments in a single semester. You should keep track of all **students** (their names, Ids, and addresses) and **professors** (name, Id, department). Do not record the address of professors but keep track of their ages. Maintain records of **courses** also. Like what classroom is assigned to a course, what is the current enrollment, and which department offers it. At most one professor **teaches** each course. Each student **evaluates** the professor teaching the course. Note that all course offerings in the semester are unique, i.e. course names and numbers do not overlap. A course can have ≥ 0 **pre-requisites**, excluding itself. A student enrolled in a course must have enrolled in all its pre-requisites. Each student receives a **grade** in each course. The **departments** are also unique, and can have at most one chairperson (or dept. head). A chairperson is not allowed to head two or more departments.

VirginiaTech

Relational Design for the Example

- Students (PID: *string*, Name: *string*, Address: *string*)
- Professors (PID: *string*, Name: *string*, Office: *string*, Age: *integer*, DepartmentName: *string*)
- Courses (Number: integer, DeptName: *string*, CourseName: *string*, Classroom: *string*, Enrollment: *integer*)
- Teach (ProfessorPID: *string*, Number: *integer*, DeptName: *string*)
- Take (StudentPID: *string*, Number: *integer*, DeptName: *string*, Grade: *string*, ProfessorEvaluation: integer)
- Departments (Name: *string*, ChairmanPID: *string*)
- PreReq (Number: *integer*, DeptName: *string*, PreReqNumber: *integer*, PreReqDeptName: *string*)

Relational Design Example: Keys?

- Students (PID: *string*, Name: *string*, Address: *string*)
- Professors (PID: *string*, Name: *string*, Office: *string*, Age: *integer*, DepartmentName: *string*)
- Courses (Number: integer, DeptName: *string*, CourseName: *string*, Classroom: *string*, Enrollment: *integer*)
- Teach (ProfessorPID: *string*, Number: *integer*, DeptName: *string*)
- Take (StudentPID: *string*, Number: *integer*, DeptName: *string*, Grade: *string*, ProfessorEvaluation: integer)
- Departments (Name: *string*, ChairmanPID: *string*)
- PreReq (Number: *integer*, DeptName: *string*, PreReqNumber: *integer,* PreReqDeptName: *string*)

Relational Design: Keys?

- Students (PID: *string*, Name: *string*, Address: *string*)
- Professors (PID: *string*, Name: *string*, Office: *string*, Age: *integer*, DepartmentName: *string*)
- Courses (Number: integer, DeptName: *string*, CourseName: *string*, Classroom: *string*, Enrollment: *integer*)
- Teach (ProfessorPID: *string*, Number: *integer*, DeptName: *string*)
- Take (StudentPID: *string*, Number: *integer*, DeptName: *string*, Grade: *string*, ProfessorEvaluation: integer)
- Departments (Name: *string*, ChairmanPID: *string*)
- PreReq (Number: *integer*, DeptName: *string*, PreReqNumber: *integer*, PreReqDeptName: *string*)

Issues to Consider in the Design

- Can we merge Courses and Teach since each professor teaches at most one course?
- Do we need a separate relation to store evaluations?
- How can we handle pre-requisites that are "or"s, e.g., you can take CS 4604 if you have taken either CS 3114 or CS 2606?
- How do we generalize this schema to handle data over more than one semester?
- What modifications does the schema need if more than one professor can teach a course?

Formal query languages

- How do we collect information?
- Eg., find ssn's of people in 415
- (recall: everything is a set!)
- One solution: Rel. algebra, ie., set operators
- Q1: Which ones??
- Q2: what is a minimal set of operators?

Relational operators

- .
- .
- .
- set union **U**
- set difference '-'

Example:

- Q: find all students (part or full time)
- A: PT-STUDENT union FT-STUDENT

FT-STUDENT		
Ssn	Name	
129	peters	main str
239	lee	5th ave

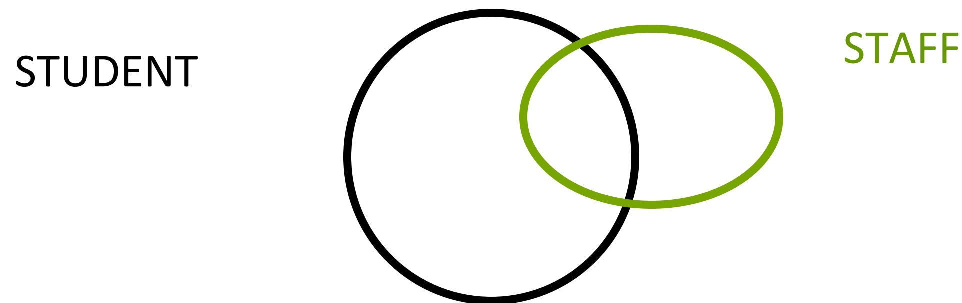
PT-STUDENT		
Ssn	Name	Address
123	smith	main str
234	jones	forbes ave

Observations:

- two tables are 'union compatible' if they have the same attributes ('domains')
- Q: how about intersection $\cap$

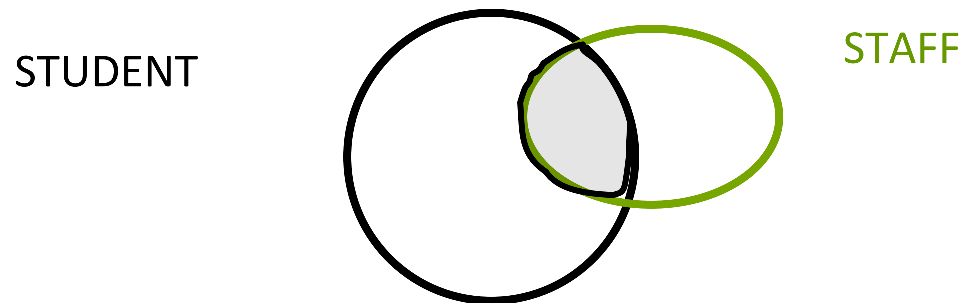
Observations:

- A: redundant:
- STUDENT intersection STAFF =



Observations:

- A: redundant:
- STUDENT intersection STAFF =

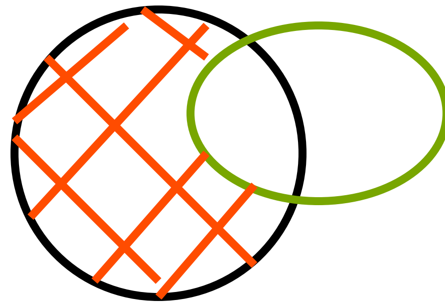


Observations:

- A: redundant:
- STUDENT intersection STAFF =

STUDENT - (STUDENT - STAFF)

STUDENT



STAFF

Observations:

- A: redundant:
- STUDENT intersection STAFF =

  STUDENT - (STUDENT - STAFF)

Double negation:
We'll see it again, later…

Relational operators

- .
- .
- .
- set union U
- set difference '-'

Other operators?

- eg, find all students on 'Main street'
- A: 'selection'

$$\sigma_{address='main\,str'}(STUDENT)$$

STUDENT		
Ssn	Name	Address
123	smith	main str
234	jones	forbes ave

Other operators?

- Notice: selection (and rest of operators) expect tables, and produce tables (-> can be cascaded!!)
- For selection, in general:

$$\sigma_{condition}(RELATION)$$

Selection - examples

- Find all 'Smiths' on 'Main St.'

$$\sigma_{name='Smith' \wedge address='Main\ st.'}(STUDENT)$$

'condition' can be any boolean combination of '= ', '>' , '>= ', '<>', ...

Relational operators

- selection $\sigma_{condition}(R)$
- .
- .
- set union R ∪ S
- set difference R - S

Relational operators

- selection picks rows - how about columns?
- A: 'projection' - eg.:

$$\pi_{ssn}(STUDENT)$$

finds all the 'ssn' - removing duplicates

STUDENT		
Ssn	Name	Address
123	smith	main str
234	jones	forbes ave

Relational operators

Cascading: ‘find ssn of students on ‘main st.’

$$\pi_{ssn}(\sigma_{address='main\ st'}(STUDENT))$$

STUDENT		
Ssn	Name	Address
123	smith	main str
234	jones	forbes ave

Relational operators

- selection $\sigma_{condition}(R)$
- projection $\pi_{att-list}(R)$
- .
- set union R ∪ S
- set difference R - S

Relational operators

Are we done yet?

Q: Give a query we can **not** answer yet!

Relational operators

A: any query across **two** or more tables,

eg., 'find names of students in 4604'

Q: what extra operator do we need??

STUDENT		
Ssn	Name	Address
123	smith	main str
234	jones	forbes ave

SSN	c-id	grade
123	4604	A
234	5614	B

Relational operators

A: any query across **two** or more tables,

eg., 'find names of students in 4604'

Q: what extra operator do we need??

A: surprisingly, cartesian product is enough!

STUDENT		
Ssn	Name	Address
123	smith	main str
234	jones	forbes ave

SSN	c-id	grade
123	4604	A
234	5614	B

Cartesian product

- eg., dog-breeding: MALE x FEMALE
- gives all possible couples

MALE
name
spike
spot

x

FEMALE
name
lassie
shiba

=

M.name	F.name
spike	lassie
spike	shiba
spot	lassie
spot	shiba

so what?

- Eg., how do we find names of students taking 4604?

Cartesian product

- A: $\ldots\ldots\ldots\sigma_{STUDENT.ssn=TAKES.ssn}(STUDENT \ x \ TAKES)$

Ssn	Name	Address	ssn	cid	grade
123	smith	main str	123	4604	A
~~234~~	~~jones~~	~~forbes ave~~	~~123~~	~~4604~~	~~A~~
~~123~~	~~smith~~	~~main str~~	~~234~~	~~5614~~	~~B~~
234	jones	forbes ave	234	5614	B

Cartesian product

$$..\sigma_{cid=4604}(\sigma_{STUDENT.ssn=TAKES.ssn}(STUDENT\ x\ TAKES))$$

Ssn	Name	Address	ssn	cid	grade
123	smith	main str	123	4604	A
~~234~~	~~jones~~	~~forbes ave~~	~~123~~	~~4604~~	~~A~~
~~123~~	~~smith~~	~~main str~~	~~234~~	~~5614~~	~~B~~
234	jones	forbes ave	234	5614	B

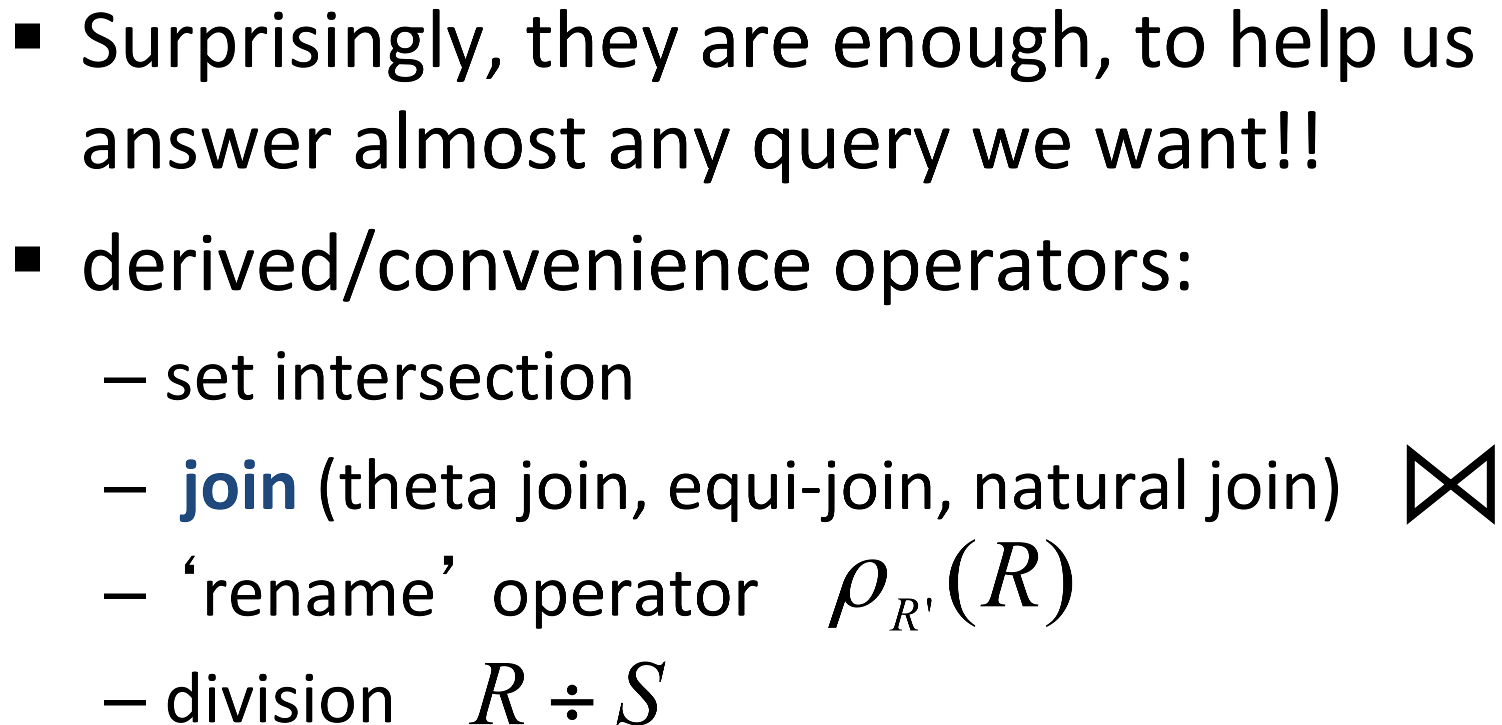
$$\pi_{name}($$
$$\sigma_{cid=4604}(\sigma_{STUDENT.ssn=TAKES.ssn}(STUDENT\ x\ TAKES))$$
$$)$$

Ssn	Name	Address	ssn	cid	grade
123	smith	main str	123	4604	A
~~234~~	~~jones~~	~~forbes ave~~	~~123~~	~~4604~~	~~A~~
~~123~~	~~smith~~	~~main str~~	~~234~~	~~5614~~	~~B~~
234	jones	forbes ave	234	5614	B

FUNDAMENTAL Relational operators

- selection $\sigma_{condition}(R)$
- projection $\pi_{att-list}(R)$
- cartesian product MALE x FEMALE
- set union R ∪ S
- set difference R - S

Relational ops



Joins

- Equijoin:

$$R \bowtie_{R.a=S.b} S \quad = \sigma_{R.a=S.b}(R \times S)$$

Cartesian product

- A: ………$\sigma_{STUDENT.ssn=TAKES.ssn}(STUDENT \ x \ TAKES)$

Ssn	Name	Address	ssn	cid	grade
123	smith	main str	123	4604	A
~~234~~	~~jones~~	~~forbes ave~~	~~123~~	~~4604~~	~~A~~
~~123~~	~~smith~~	~~main str~~	~~234~~	~~5614~~	~~B~~
234	jones	forbes ave	234	5614	B

Joins

- Equijoin: $R \bowtie_{R.a=S.b} S = \sigma_{R.a=S.b}(R \times S)$
- theta-joins: $R \bowtie_{\theta} S$

  generalization of equi-join - any condition θ

Joins

- **very** popular: natural join: R ⋈ S
- like equi-join, but it drops duplicate columns:

  STUDENT (ssn, name, address)

  TAKES (ssn, cid, grade)

Joins

- nat. join has 5 attributes $STUDENT \bowtie TAKES$

Ssn	Name	Address	ssn	cid	grade
123	smith	main str	123	4604	A
234	jones	forbes ave	123	4604	A
123	smith	main str	234	5614	B
234	jones	forbes ave	234	5614	B

equi-join: 6 $STUDENT \bowtie_{STUDENT.ssn=TAKES.ssn} TAKES$

Natural Joins - nit-picking

- if no attributes in common between R, S:

  nat. join -> cartesian product

Overview - rel. algebra

- fundamental operators
- derived operators
  - joins etc
  - **rename**
  - division
- examples

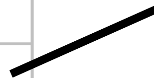
Rename op.

- Q: why? $\rho_{AFTER}(BEFORE)$
- A: shorthand; self-joins; …
- for example, find the grand-parents of 'Tom', given PC (parent-id, child-id)

Rename op.

- PC (parent-id, child-id) $PC \bowtie PC$

PC	
p-id	c-id
Mary	Tom
Peter	Mary
John	Tom



PC	
p-id	c-id
Mary	Tom
Peter	Mary
John	Tom

Rename op.

- first, WRONG attempt:

$$PC \bowtie PC$$

- (why? how many columns?)
- Second WRONG attempt:

$$PC \bowtie_{PC.c-id=PC.p-id} PC$$

Rename op.

- we clearly need two different names for the same table - hence, the ‘rename’ op.

$$\rho_{PC1}(PC) \bowtie_{PC1.c-id=PC.p-id} PC$$

Overview - rel. algebra

- fundamental operators
- derived operators
  - joins etc
  - rename
  - **division**
- examples

Division

- Rarely used, but powerful.
- Example: find suspicious suppliers, ie., suppliers that supplied all the parts in A_BOMB

Division

SHIPMENT	
s#	p#
s1	p1
s2	p1
s1	p2
s3	p1
s5	p3

÷

ABOMB
p#
p1
p2

=

BAD_S
s#
s1

Example: find suspicious suppliers, ie., suppliers that supplied all the parts in A_BOMB

Division

- Observations: ~reverse of cartesian product
- It can be derived from the 5 fundamental operators (!!)
- How?

Division

- Answer:

$$r \div s = \pi_{(R-S)}(r) - \pi_{(R-S)}[(\pi_{(R-S)}(r) \times s) - r]$$

- Observation: find 'good' suppliers, and subtract! (**double negation**)

Division

- Answer:

SHIPMENT	
s#	p#
s1	p1
s2	p1
s1	p2
s3	p1
s5	p3

÷

ABOMB
p#
p1
p2

=

BAD_S
s#
s1

$$r \div s = \pi_{(R-S)}(r) - \pi_{(R-S)}[(\pi_{(R-S)}(r) \times s) - r]$$

- Observation: find 'good' suppliers, and subtract! (**double negation**)

Division

- Answer:

Table 'r'

SHIPMENT	
s#	p#
s1	p1
s2	p1
s1	p2
s3	p1
s5	p3

÷

Table 's'

ABOMB
p#
p1
p2

=

BAD_S
s#
s1

$$r \div s = \pi_{(R-S)}(r) - \pi_{(R-S)}[(\pi_{(R-S)}(r) \times s) - r]$$

All suppliers

All bad parts

Division

- Answer:

Table 'r'

SHIPMENT	
s#	p#
s1	p1
s2	p1
s1	p2
s3	p1
s5	p3

÷

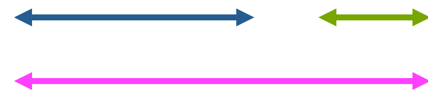
Table 's'

ABOMB
p#
p1
p2

=

BAD_S
s#
s1

$$r \div s = \pi_{(R-S)}(r) - \pi_{(R-S)}[(\pi_{(R-S)}(r) \times s) - r]$$



all possible
suspicious shipments

Division

- Answer:

Table 'r'

SHIPMENT	
s#	p#
s1	p1
s2	p1
s1	p2
s3	p1
s5	p3

÷

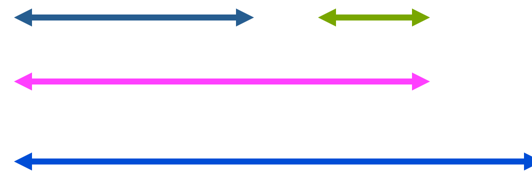
Table 's'

ABOMB
p#
p1
p2

=

BAD_S
s#
s1

$$r \div s = \pi_{(R-S)}(r) - \pi_{(R-S)}[(\pi_{(R-S)}(r) \times s) - r]$$



all possible
suspicious shipments
that didn't happen

Division

- Answer:

Table 'r'

SHIPMENT	
s#	p#
s1	p1
s2	p1
s1	p2
s3	p1
s5	p3

÷

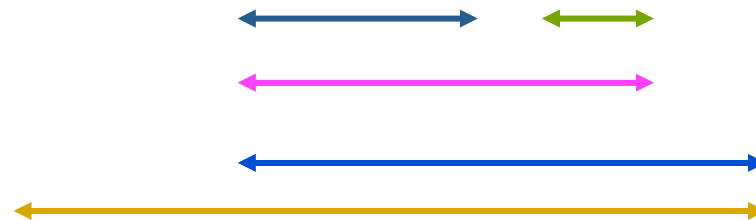
Table 's'

ABOMB
p#
p1
p2

=

BAD_S
s#
s1

$$r \div s = \pi_{(R-S)}(r) - \pi_{(R-S)}[(\pi_{(R-S)}(r) \times s) - r]$$



all suppliers who missed
at least one suspicious shipment,
i.e.: '<u>good</u>' suppliers

Overview - rel. algebra

- fundamental operators
- derived operators
  - joins etc
  - rename
  - division
- **examples**

Sample schema

find names of students that take 4604

STUDENT

Ssn	Name	Address
123	smith	main str
234	jones	forbes ave

CLASS

c-id	c-name	units
4513	s.e.	2
4512	o.s.	2

TAKES

SSN	c-id	grade
123	4513	A
234	4513	B

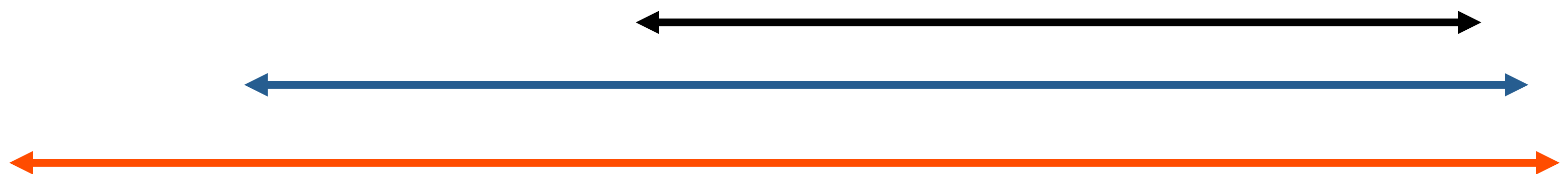
Examples

- find names of students that take 4604

Examples

- find names of students that take 4604

$$\pi_{name}[\sigma_{c-id=4604}(STUDENT \bowtie TAKES)]$$



Sample schema

find course names of 'smith'

STUDENT

Ssn	Name	Address
123	smith	main str
234	jones	forbes ave

CLASS

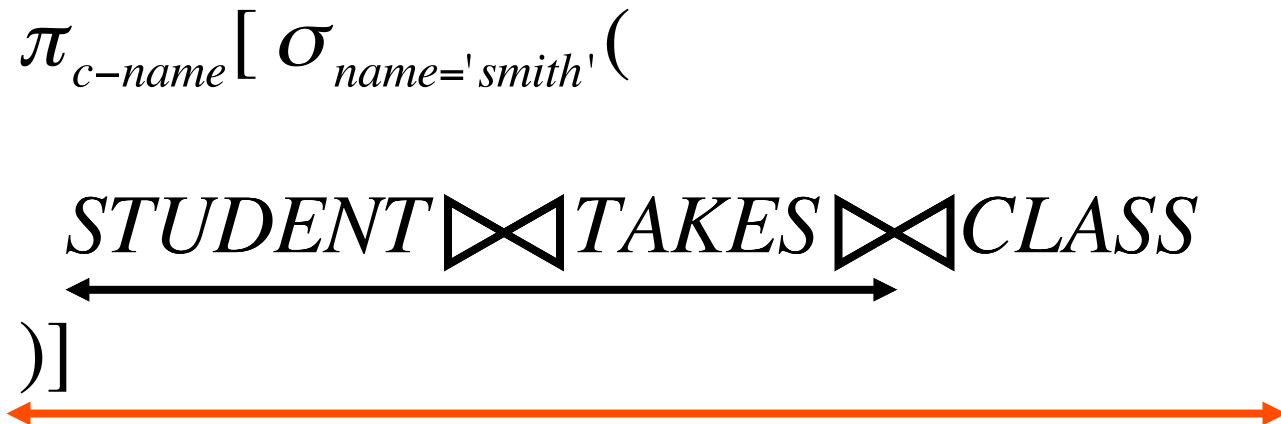
c-id	c-name	units
4613	s.e.	2
4612	o.s.	2

TAKES

SSN	c-id	grade
123	4613	A
234	4613	B

Examples

- find course names of ‘smith’



VirginiaTech

Examples

- find ssn of ‘overworked’ students, ie., that take 4612, 4613, 4604

Examples

- find ssn of 'overworked' students, ie., that take 4612, 4613, 4604: almost correct answer:

$$\sigma_{c-name=4612}(TAKES) \bigcap$$
$$\sigma_{c-name=4613}(TAKES) \bigcap$$
$$\sigma_{c-name=4604}(TAKES)$$

Examples

- find ssn of 'overworked' students, ie., that take 4612, 4613, 4604 - Correct answer:

$$\pi_{ssn}[\sigma_{c-name=4612}(TAKES)] \cap$$
$$\pi_{ssn}[\sigma_{c-name=4613}(TAKES)] \cap$$
$$\pi_{ssn}[\sigma_{c-name=4604}(TAKES)]$$

Examples

- find ssn of students that work at least as hard as ssn=123, ie., they take all the courses of ssn=123, and maybe more

Sample schema

STUDENT		
Ssn	Name	Address
123	smith	main str
234	jones	forbes ave

CLASS

c-id	c-name	units
4613	s.e.	2
4612	o.s.	2

TAKES

SSN	c-id	grade
123	4613	A
234	4613	B

Examples

- find ssn of students that work at least as hard as ssn=123 (ie., they take all the courses of ssn=123, and maybe more

$$[\pi_{ssn,c-id}(TAKES)] \div \pi_{c-id}[\sigma_{ssn=123}(TAKES)]$$

VirginiaTech

Conclusions

- Relational model: only tables (‘relations’)
- relational algebra: powerful, minimal: 5 operators can handle almost any query!