Databases and Information Systems

Relational Algebra

Dmitriy Traytel
slides partly by Marcos Vaz Salles
UNIVERSITY OF COPENHAGEN

Do-It-Yourself Recap:
Fill in the blanks

P

x	y
1	10
2	12
42	10

$\{x \mapsto 42\} \quad \neg P(x,10)$

$\{x \mapsto 12\} \quad \neg P(x,10)$

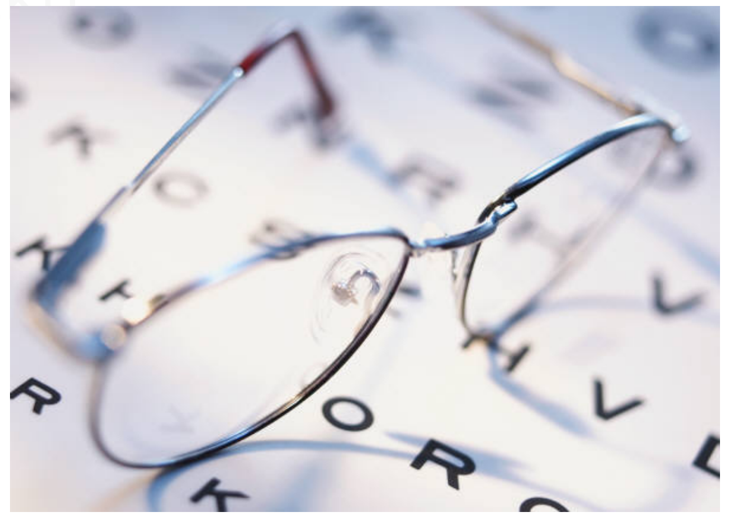
$[\![\neg P(x,10)]\!] =$

$[\![P(x,10) \vee P(2,x)]\!] =$

$[\![\neg P(x,10) \vee P(2,x)]\!] =$

$[\![P(x,10) \vee P(2,y)]\!] =$

What should we learn today?



- Understand the notion of domain independence and be able to argue whether a given query is domain independent
- Understand the relational algebra normal form (RANF) and be able to determine whether a relational calculus query is in RANF
- Explain the operators of the relational algebra
- Formulate queries in the relational algebra
- Explain limitations of the relational algebra in terms of query expressiveness
- Formulate queries with the main extensions of the relational algebra including bag semantics, grouping, aggregation, sorting, and outer join

Finite vs Infinite

- Fundamental problem with relational calculus:
  $[\![\phi]\!]$ is not always a finite relation
- Some examples for such “**unsafe**” queries
  - ϕ = Ships(n,cl,l) $\vee$ Outcomes(n,b,r)
  - ϕ = P(x) $\vee$ Q(y)
  - ϕ = $\neg$ P(x)
  - ϕ = x $\approx$ y
- But: query evaluation works with finite tables (why?)

Domain Independence

- All tables (predicates) are finite
- Q: Where does the infiniteness of $[\![\phi]\!]$ come from?

Domain Independence

- All tables (predicates) are finite
- Q: Where does the infiniteness of $[\![\phi]\!]$ come from?
- A: The domain $\mathbb{D}$

Domain Independence

- All tables (predicates) are finite
- Q: Where does the infiniteness of $[\![\phi]\!]$ come from?
- A: The domain $\mathbb{D}$
- $\mathbb{D}$ can be seen as a parameter of query evaluation: $[\![\phi]\!]_{\mathbb{D}}$

Domain Independence

- All tables (predicates) are finite
- Q: Where does the infiniteness of $[\![\phi]\!]$ come from?
- A: The domain $\mathbb{D}$
- $\mathbb{D}$ can be seen as a parameter of query evaluation: $[\![\phi]\!]_{\mathbb{D}}$
- ϕ is domain-independent if for all $\mathbb{D}$, $\mathbb{E}$: $[\![\phi]\!]_{\mathbb{D}} = [\![\phi]\!]_{\mathbb{E}}$

Domain Independence

- All tables (predicates) are finite
- Q: Where does the infiniteness of $[\![\phi]\!]$ come from?
- A: The domain $\mathbb{D}$
- $\mathbb{D}$ can be seen as a parameter of query evaluation: $[\![\phi]\!]_{\mathbb{D}}$
- ϕ is domain-independent if for all $\mathbb{D}$, $\mathbb{E}$: $[\![\phi]\!]_{\mathbb{D}} = [\![\phi]\!]_{\mathbb{E}}$
- For example: $P(x) \overset{\text{AND}}{\wedge} Q(y)$ is domain-independent: $[\![P(x) \wedge Q(y)]\!]_{\mathbb{D}} = DB(P) \times DB(Q)$

Unsafe $\Longrightarrow$ Not Domain Independent

$[\![\phi]\!]_{\mathbb{D}}$

$\phi = P(x) \vee Q(y)$	$\{(x,y).\ x \in DB(P) \text{ and } y \in \mathbb{D} \text{ or } x \in \mathbb{D} \text{ and } y \in DB(Q)\}$
$\phi = \neg\ P(x)$	$\{(x).\ x \in \mathbb{D} \text{ and } x \notin DB(P)\}$
$\phi = x \approx y$	$\{(x,x).\ x \in \mathbb{D}\}$

Safe $\Longrightarrow$ Domain Independent?

- Here, safe means “finite query result”
- Implication does **not** hold!

Safe $\Longrightarrow$ Domain Independent?

- Here, safe means "finite query result"
- Implication does **not** hold!
- Any formula without free variables (= closed) has a finite query result

Safe $\Longrightarrow$ Domain Independent?

- Here, safe means “finite query result”
- Implication does **not** hold!
- Any formula without free variables (= closed) has a finite query result
- $[\![\forall x.\ P(x)]\!]_{\mathbb{D}} = \{()\}$ if $\mathbb{D} = DB(P)$

Safe $\Longrightarrow$ Domain Independent?

- Here, safe means “finite query result”
- Implication does **not** hold!
- Any formula without free variables (= closed) has a finite query result
- $[\![\forall x.\ P(x)]\!]_{\mathbb{D}} = \{()\}$ if $\mathbb{D} = DB(P)$
- $[\![\forall x.\ P(x)]\!]_{\mathbb{D}} = \{\}$ if $DB(P) \subsetneq \mathbb{D}$

When is a formula domain-independent?

- Undecidable problem
  (= there exists no algorithm that answers the above question precisely)

- Resort to **syntactic** overapproximations:
  - under easy-to-check conditions a formula is domain-independent
    - e.g., the formula is a conjunction of n predicates (conjunctive queries)
  - conditions not met $\implies$ the formula may or may not be domain-independent

Relational Algebra Normal Form

- A particular syntactic overapproximation
- RANF $\Longrightarrow$ domain-independent
- Even better: RANF $\Longrightarrow$ each “subformula” evaluates to a finite relation
- Has something to do with Relational Algebra (coming soon)

RANF (continued)

```
ranf(P(t1, …, tn))  ⟺ true
ranf(t1 ≈ t2)        ⟺ false
ranf(¬ φ)            ⟺ fv(φ)={} and ranf(φ)
ranf(φ ∨ ψ)          ⟺ ranf(φ) and ranf(ψ) and fv(φ)=fv(ψ)
ranf(φ ∧ ψ)          ⟺ …
ranf(∀x. φ)          ⟺ false
ranf(∃x. φ)          ⟺ ranf(φ)
```

RANF (continued)

ranf(φ) and ranf(ψ) or

ran

ran

ran

ranf(φ ∨ ψ) ⟺ anf(φ) and ranf(ψ) and fv(φ)=fv(ψ)

ranf(φ ∧ ψ) ⟺ …

ranf(∀x. φ) ⟺ false

ranf(∃x. φ) ⟺ ranf(φ)

RANF (continued)

ranf(φ) and ranf(ψ) or
ranf(φ) and ψ=¬χ and ranf(χ) and fv(χ)⊆fv(φ) or

ran
ran
ran

ranf(φ ∨ ψ) ⟺ ranf(φ) and ranf(ψ) and fv(φ)=fv(ψ)
ranf(φ ∧ ψ) ⟺ …
ranf(∀x. φ) ⟺ false
ranf(∃x. φ) ⟺ ranf(φ)

RANF (continued)

ranf(φ) and ranf(ψ) or
ranf(φ) and ψ=¬χ and ranf(χ) and fv(χ)⊆fv(φ) or
ranf(φ) and ψ=t1 ≈ t2 and (fv(t1)⊆fv(φ) or fv(t2)⊆fv(φ)) or

ran
ran
ran
ranf(φ ∨ ψ) ⟺ ranf(φ) and ranf(ψ) and fv(φ)=fv(ψ)
ranf(φ ∧ ψ) ⟺ …
ranf(∀x. φ) ⟺ false
ranf(∃x. φ) ⟺ ranf(φ)

RANF (continued)

$\mathrm{ranf}(\varphi)$ and $\mathrm{ranf}(\psi)$ or
$\mathrm{ranf}(\varphi)$ and $\psi = \neg\chi$ and $\mathrm{ranf}(\chi)$ and $\mathrm{fv}(\chi) \subseteq \mathrm{fv}(\varphi)$ or
$\mathrm{ranf}(\varphi)$ and $\psi = t1 \approx t2$ and $(\mathrm{fv}(t1) \subseteq \mathrm{fv}(\varphi)$ or $\mathrm{fv}(t2) \subseteq \mathrm{fv}(\varphi))$ or
$\mathrm{ranf}(\varphi)$ and $\psi = \neg t1 \approx t2$ and $\mathrm{fv}(t1) \subseteq \mathrm{fv}(\varphi)$ and $\mathrm{fv}(t2) \subseteq \mathrm{fv}(\varphi)$

$\mathrm{ranf}(\varphi \lor \psi) \iff \mathrm{ranf}(\varphi)$ and $\mathrm{ranf}(\psi)$ and $\mathrm{fv}(\varphi) = \mathrm{fv}(\psi)$
$\mathrm{ranf}(\varphi \land \psi) \iff$ …
$\mathrm{ranf}(\forall x.\ \varphi) \iff$ false
$\mathrm{ranf}(\exists x.\ \varphi) \iff \mathrm{ranf}(\varphi)$

Quiz: RANF or Not?

$$P(x) \lor Q(y)$$

$$P(x) \land Q(y)$$

$$P(x,y) \land Q(y,z)$$

$$P(x,y) \land \neg Q(y,z)$$

$$P(x,y,z) \land \neg Q(y,z)$$

$$P(x,y,z) \land \neg\, y \approx z$$

$$P(x,y,z) \land \neg (Q(y,z) \lor R(x,z))$$

$$P(x,y,z) \land \neg Q(y,z) \land \neg R(x,z)$$

Quiz: RANF or Not?

$P(x) \lor Q(y)$ ✗

$P(x) \land Q(y)$

$P(x,y) \land Q(y,z)$

$P(x,y) \land \neg Q(y,z)$

$P(x,y,z) \land \neg Q(y,z)$

$P(x,y,z) \land \neg y \approx z$

$P(x,y,z) \land \neg (Q(y,z) \lor R(x,z))$

$P(x,y,z) \land \neg Q(y,z) \land \neg R(x,z)$

Quiz: RANF or Not?

$P(x) \lor Q(y)$ ✗

$P(x) \land Q(y)$ ✓

$P(x,y) \land Q(y,z)$

$P(x,y) \land \neg Q(y,z)$

$P(x,y,z) \land \neg Q(y,z)$

$P(x,y,z) \land \neg\, y \approx z$

$P(x,y,z) \land \neg (Q(y,z) \lor R(x,z))$

$P(x,y,z) \land \neg Q(y,z) \land \neg R(x,z)$

Quiz: RANF or Not?

$P(x) \lor Q(y)$ ✗

$P(x) \land Q(y)$ ✓

$P(x,y) \land Q(y,z)$ ✓

$P(x,y) \land \neg Q(y,z)$

$P(x,y,z) \land \neg Q(y,z)$

$P(x,y,z) \land \neg y \approx z$

$P(x,y,z) \land \neg (Q(y,z) \lor R(x,z))$

$P(x,y,z) \land \neg Q(y,z) \land \neg R(x,z)$

Quiz: RANF or Not?

$P(x) \lor Q(y)$ ✗

$P(x) \land Q(y)$ ✓

$P(x,y) \land Q(y,z)$ ✓

$P(x,y) \land \neg Q(y,z)$ ✗

$P(x,y,z) \land \neg Q(y,z)$

$P(x,y,z) \land \neg y \approx z$

$P(x,y,z) \land \neg (Q(y,z) \lor R(x,z))$

$P(x,y,z) \land \neg Q(y,z) \land \neg R(x,z)$

Quiz: RANF or Not?

$P(x) \lor Q(y)$ ✗

$P(x) \land Q(y)$ ✓

$P(x,y) \land Q(y,z)$ ✓

$P(x,y) \land \neg Q(y,z)$ ✗

$P(x,y,z) \land \neg Q(y,z)$ ✓

$P(x,y,z) \land \neg\, y \approx z$

$P(x,y,z) \land \neg (Q(y,z) \lor R(x,z))$

$P(x,y,z) \land \neg Q(y,z) \land \neg R(x,z)$

Quiz: RANF or Not?

$P(x) \lor Q(y)$ ✗

$P(x) \land Q(y)$ ✓

$P(x,y) \land Q(y,z)$ ✓

$P(x,y) \land \neg Q(y,z)$ ✗

$P(x,y,z) \land \neg Q(y,z)$ ✓

$P(x,y,z) \land \neg\, y \approx z$ ✓

$P(x,y,z) \land \neg (Q(y,z) \lor R(x,z))$

$P(x,y,z) \land \neg Q(y,z) \land \neg R(x,z)$

Quiz: RANF or Not?

$P(x) \vee Q(y)$ ✗

$P(x) \wedge Q(y)$ ✓

$P(x,y) \wedge Q(y,z)$ ✓

$P(x,y) \ \wedge \neg\, Q(y,z)$ ✗

$P(x,y,z) \ \wedge \neg\, Q(y,z)$ ✓

$P(x,y,z) \ \wedge \neg\, y \approx z$ ✓

$P(x,y,z) \ \wedge \neg\, (Q(y,z) \vee R(x,z))$ ✗

$P(x,y,z) \ \wedge \neg\, Q(y,z) \ \wedge \neg\, R(x,z)$

Quiz: RANF or Not?

$P(x) \lor Q(y)$	✗
$P(x) \land Q(y)$	✓
$P(x,y) \land Q(y,z)$	✓
$P(x,y) \land \neg Q(y,z)$	✗
$P(x,y,z) \land \neg Q(y,z)$	✓
$P(x,y,z) \land \neg\, y \approx z$	✓
$P(x,y,z) \land \neg (Q(y,z) \lor R(x,z))$	✗
$P(x,y,z) \land \neg Q(y,z) \land \neg R(x,z)$	✓

Competition Task: Smells Like Cheese Spirit

```
loves(who:string,what:string)
smells(what:string)
```

e.g.

```
loves(Bill,cheese)
loves(Bill,jackfruit)
loves(Bill,tomato)
loves(Elon,cheese)
loves(Elon,fish)
loves(Elon,jackfruit)
loves(Jeff,cheese)
loves(Jeff,Jeff)
smells(cheese)
smells(jackfruit)
smells(fish)
```

Competition Task: Smells Like Cheese Spirit

```
loves(who:string,what:string)
smells(what:string)
```

Compute the lovers of cheese that love all things that smell.

e.g.
```
loves(Bill,cheese)
loves(Bill,jackfruit)
loves(Bill,tomato)
loves(Elon,cheese)
loves(Elon,fish)
loves(Elon,jackfruit)
loves(Jeff,cheese)
loves(Jeff,Jeff)
smells(cheese)
smells(jackfruit)
smells(fish)
```

Competition Task: Smells Like Cheese Spirit

```
loves(who:string,what:string)
smells(what:string)
```

Compute the lovers of cheese
that love all things that smell.

The most efficient RC query in RANF wins.

e.g.

```
loves(Bill,cheese)
loves(Bill,jackfruit)
loves(Bill,tomato)
loves(Elon,cheese)
loves(Elon,fish)
loves(Elon,jackfruit)
loves(Jeff,cheese)
loves(Jeff,Jeff)
smells(cheese)
smells(jackfruit)
smells(fish)
```

Competition Task: Smells Like Cheese Spirit

```
loves(who:string,what:string)
smells(what:string)
```

Compute the lovers of cheese
that love all things that smell.

The most efficient RC query in RANF wins.
Hint #1: any feature supported by RC-eval is allowed

e.g.
```
loves(Bill,cheese)
loves(Bill,jackfruit)
loves(Bill,tomato)
loves(Elon,cheese)
loves(Elon,fish)
loves(Elon,jackfruit)
loves(Jeff,cheese)
loves(Jeff,Jeff)
smells(cheese)
smells(jackfruit)
smells(fish)
```

Competition Task: Smells Like Cheese Spirit

```
loves(who:string,what:string)
smells(what:string)
```

Compute the lovers of cheese
that love all things that smell.

The most efficient RC query in RANF wins.
Hint #1: any feature supported by RC-eval is allowed
Hint #2: RC-eval uses VeriMon to evaluate RANF queries
https://doi.org/10.1007/978-3-031-17715-6_1

e.g.
```
loves(Bill,cheese)
loves(Bill,jackfruit)
loves(Bill,tomato)
loves(Elon,cheese)
loves(Elon,fish)
loves(Elon,jackfruit)
loves(Jeff,cheese)
loves(Jeff,Jeff)
smells(cheese)
smells(jackfruit)
smells(fish)
```

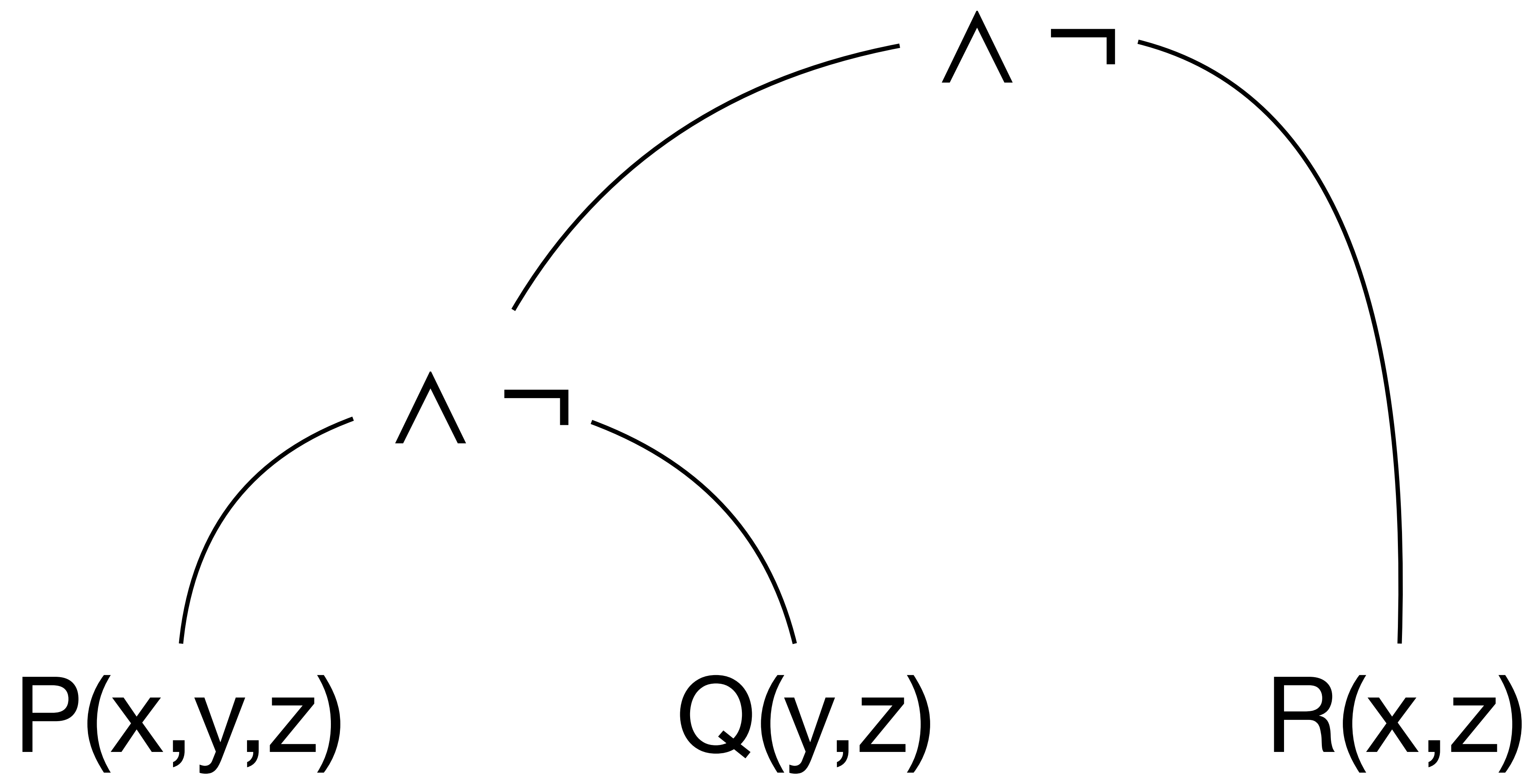
Codd’s Theorem

For every domain-independent relational calculus query there exists an equivalent query in RANF

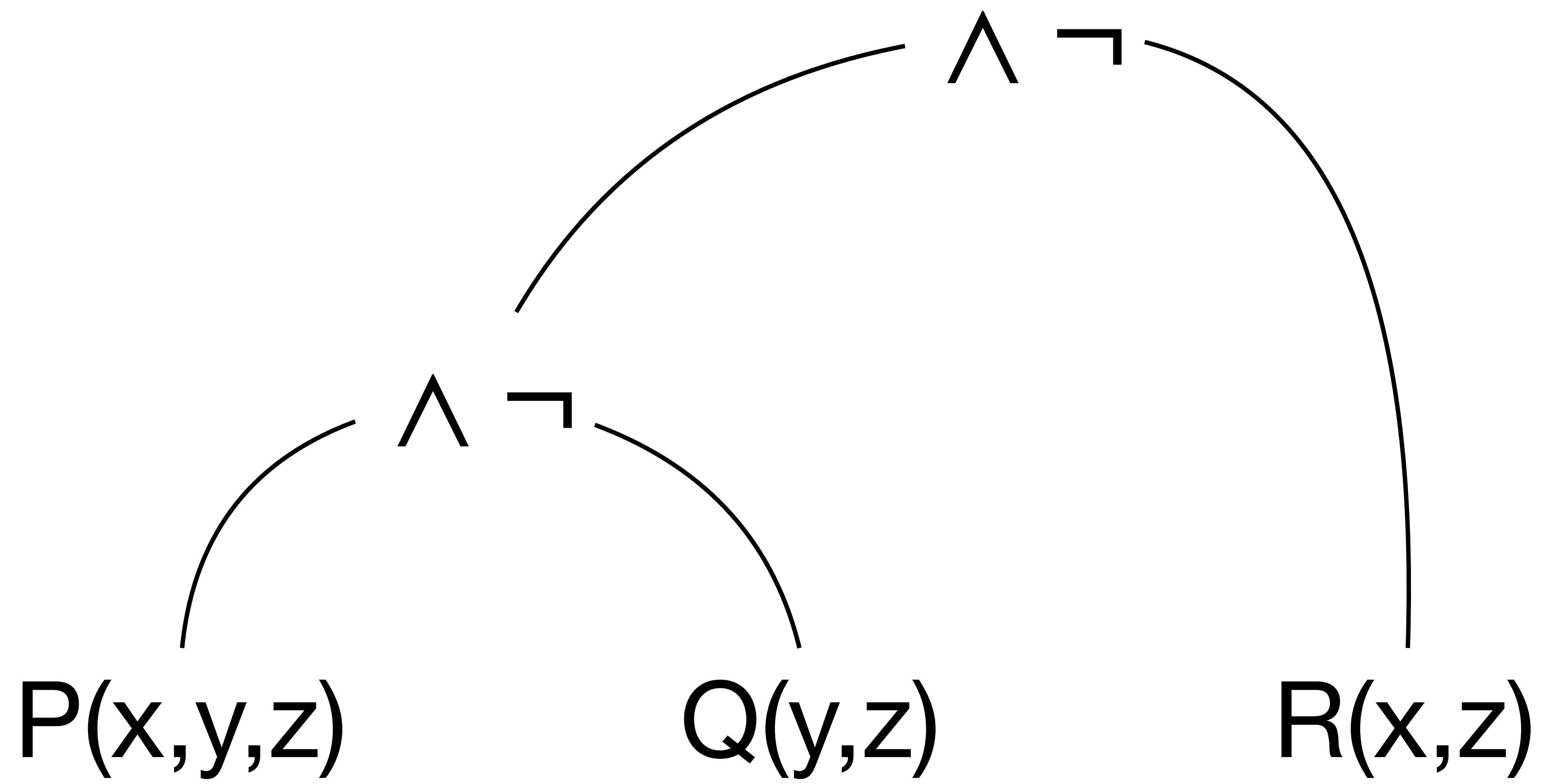
RANF $\Longrightarrow$ each “subformula” evaluates to a finite relation

$$(P(x,y,z) \ \wedge \neg \ Q(y,z)) \ \wedge \neg \ R(x,z)$$

RANF $\Longrightarrow$ each “subformula” evaluates to a finite relation



RANF $\Longrightarrow$ each "subformula" evaluates to a finite relation

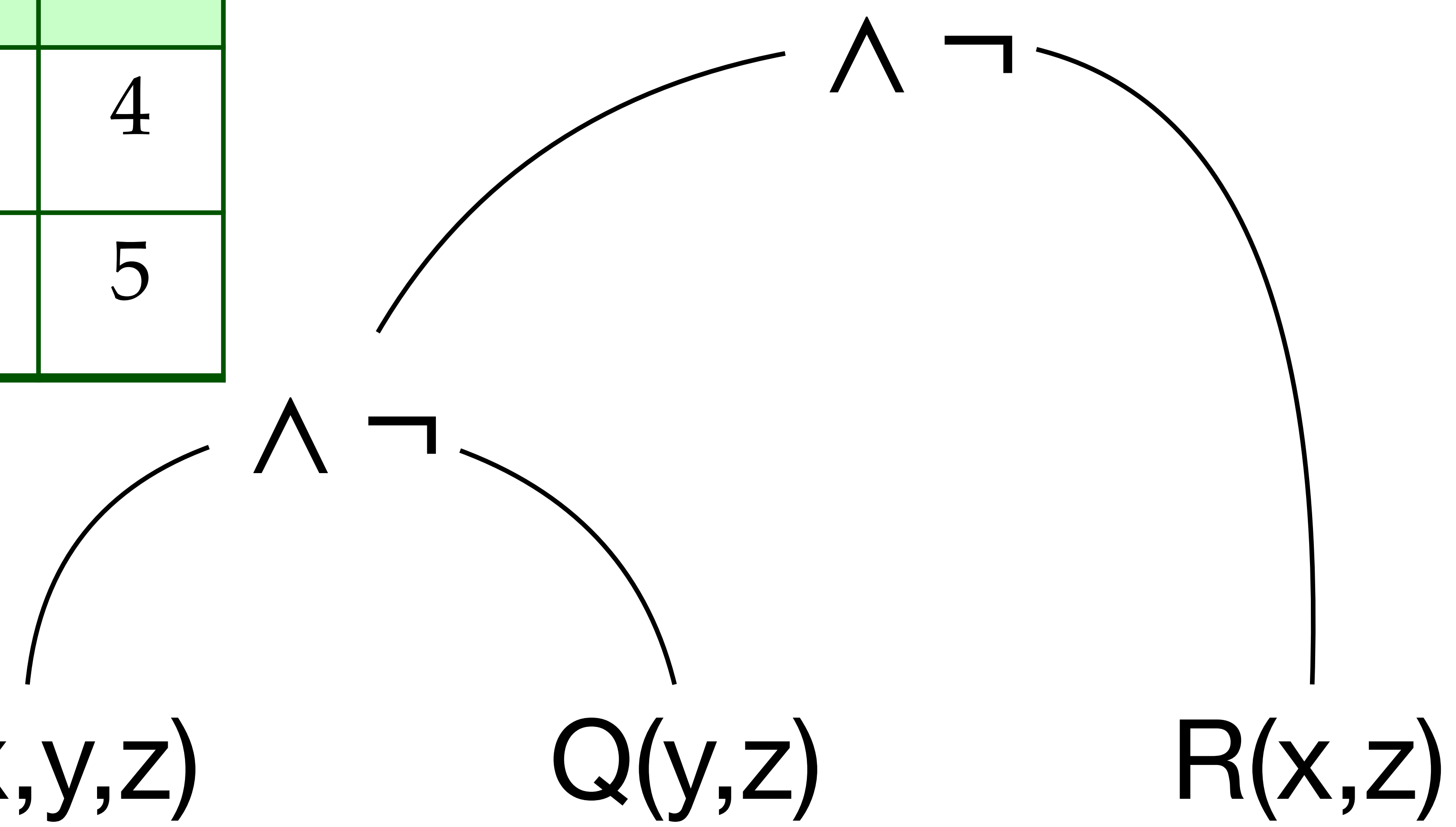


x	y	z
1	10	4
2	12	4
42	10	5

y	z
10	4
13	4

RANF $\Longrightarrow$ each "subformula" evaluates to a finite relation

x	y	z
2	12	4
42	10	5

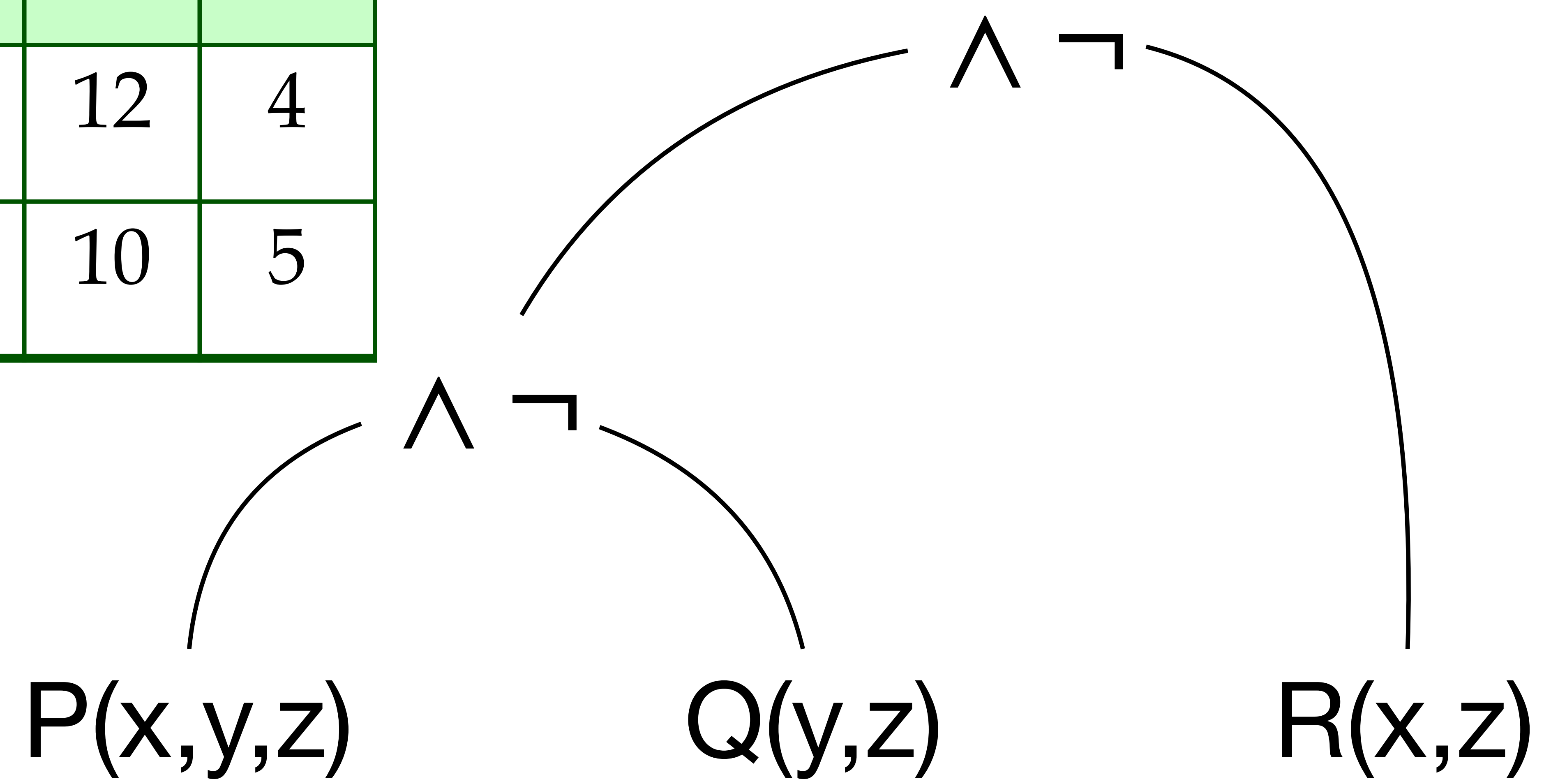


x	y	z
1	10	4
2	12	4
42	10	5

y	z
10	4
13	4

RANF $\Longrightarrow$ each "subformula" evaluates to a finite relation

x	y	z
2	12	4
42	10	5



x	y	z
1	10	4
2	12	4
42	10	5

y	z
10	4
13	4

x	z
12	4
42	5

RANF $\Longrightarrow$ each "subformula" evaluates to a finite relation

x	y	z
2	12	4

$\wedge\ \neg$

x	y	z
2	12	4
42	10	5

$\wedge\ \neg$

P(x,y,z)

x	y	z
1	10	4
2	12	4
42	10	5

Q(y,z)

y	z
10	4
13	4

R(x,z)

x	z
12	4
42	5

Relational Algebra

Give names to the supported table operations
Use these as basic operators

x	y	z
1	10	4
2	12	4
42	10	5

$\wedge \neg$

y	z
10	4
13	4

$=$

Relational Algebra

Give names to the supported table operations

Use these as basic operators

x	y	z
1	10	4
2	12	4
42	10	5

$\wedge \neg$

y	z
10	4
13	4

$=$

x	y	z
2	12	4
42	10	5

Relational Algebra

x	z
12	4
42	5

$\wedge$

y	z
10	4
13	4

$=$

Give names to the supported table operations

Use these as basic operators

x	y	z
1	10	4
2	12	4
42	10	5

$\wedge \neg$

y	z
10	4
13	4

$=$

x	y	z
2	12	4
42	10	5

Relational Algebra

x	z
12	4
42	5

inner join (42, 5, not printed)

$\wedge$

y	z
10	4
13	4

$=$

x	y	z
12	10	4
12	13	4

Give names to the supported table operations
Use these as basic operators

x	y	z
1	10	4
2	12	4
42	10	5

$\wedge \neg$

y	z
10	4
13	4

$=$

x	y	z
2	12	4
42	10	5

Relational Algebra

- Basic operations:
  - Selection σ Selects a subset of rows from relation
  - Projection π Deletes unwanted columns from relation
  - Cross-product $\times$ Allows us to combine two relations
  - Set-difference $-$ Tuples in relation 1, but not in relation 2
  - Union $\cup$ Tuples in relation 1 and in relation 2

- Additional operations:
  - Intersection $\cap$, join $\bowtie$, antijoin $\triangleright$, division $\div$, renaming ρ: Not essential, but (very!) useful.

- Each operation returns a finite relation, i.e., operations can be composed! (Algebra is “closed”.)

Source: Ramakrishnan & Gehrke

Example Instances

R1

sid	bid	day
22	101	10.10
58	103	11.12

S1

sid	sname	rating	age
22	dustin	7	45.0
31	lubber	8	55.5
58	rusty	10	35.0

S2

sid	sname	rating	age
28	yuppy	9	35.0
31	lubber	8	55.5
44	guppy	5	35.0
58	rusty	10	35.0

- “Sailors” and “Reserves” relations for our examples.
- In relational calculus: formula’s variables = column names
- In relational algebra: column names (= attributes) fixed
- Names in results “inherited” from names in input relations.

Source: Ramakrishnan & Gehrke

Projection

S2

sid	sname	rating	age
28	yuppy	9	35.0
31	lubber	8	55.5
44	guppy	5	35.0
58	rusty	10	35.0

- Deletes attributes that are not in projection list.
- Schema of result contains exactly the fields in the projection list, with the same names that they had in the (only) input relation.
- Projection operator has to eliminate duplicates! (Why??)
  - Note: real systems typically don't do duplicate elimination unless the user explicitly asks for it. (Why not?)

$\pi_{sname,rating}(S2)$

sname	rating
yuppy	9
lubber	8
guppy	5
rusty	10

$\pi_{age}(S2)$

age
35.0
55.5

Projection

S2

sid	sname	rating	age
28	yuppy	9	35.0
31	lubber	8	55.5
44	guppy	5	35.0
58	rusty	10	35.0

- Deletes attributes that are not in projection list.
- Schema of result contains exactly the fields in the projection list, with the same names that they had in the (only) input relation.
- Projection operator has to eliminate duplicates! (Why??)
  - Note: real systems typically don't do duplicate elimination unless the user explicitly asks for it. (Why not?)

$\pi_{sname,rating}(S2)$

sname	rating
yuppy	9
lubber	8
guppy	5
rusty	10

$\exists sid, age.$
$S2_{(sid,sname,rating,age)}$

$\pi_{age}(S2)$

age
35.0
55.5

Source: Ramakrishnan & Gehrke

Projection

S2

sid	sname	rating	age
28	yuppy	9	35.0
31	lubber	8	55.5
44	guppy	5	35.0
58	rusty	10	35.0

- Deletes attributes that are not in projection list.
- Schema of result contains exactly the fields in the projection list, with the same names that they had in the (only) input relation.
- Projection operator has to eliminate duplicates! (Why??)
  - Note: real systems typically don't do duplicate elimination unless the user explicitly asks for it. (Why not?)

$\pi_{sname,rating}(S2)$

$\exists sid,age.\ S2_{(sid,sname,rating,age)}$

sname	rating
yuppy	9
lubber	8
guppy	5
rusty	10

$\pi_{age}(S2)$

$\exists sid,sname,rating.\ S2_{(sid,sname,rating,age)}$

age
35.0
55.5

Source: Ramakrishnan & Gehrke

Selection

S2

sid	sname	rating	age
28	yuppy	9	35.0
31	lubber	8	55.5
44	guppy	5	35.0
58	rusty	10	35.0

- Selects rows that satisfy selection condition.
- No duplicates in result! (Why?)
- Schema of result identical to schema of (only) input relation.
- Result relation can be the input for another relational algebra operation! (Operator composition.)

$\sigma_{rating>8}(S2)$

sid	sname	rating	age
28	yuppy	9	35.0
58	rusty	10	35.0

$\pi_{sname,rating}(\sigma_{rating>8}(S2))$

sname	rating
yuppy	9
rusty	10

Source: Ramakrishnan & Gehrke

Selection

S2

sid	sname	rating	age
28	yuppy	9	35.0
31	lubber	8	55.5
44	guppy	5	35.0
58	rusty	10	35.0

- Selects rows that satisfy selection condition.
- No duplicates in result! (Why?)
- Schema of result identical to schema of (only) input relation.
- Result relation can be the input for another relational algebra operation! (Operator composition.)

$\sigma_{\text{rating}>8}(S2)$

sid	sname	rating	age
28	yuppy	9	35.0
58	rusty	10	35.0

$\pi_{\text{sname,rating}}(\sigma_{\text{rating}>8}(S2))$

sname	rating
yuppy	9
rusty	10

$S2_{(\text{sid,sname,rating,age})}$

$\land$ rating > 8

Selection

S2

sid	sname	rating	age
28	yuppy	9	35.0
31	lubber	8	55.5
44	guppy	5	35.0
58	rusty	10	35.0

- Selects rows that satisfy selection condition.
- No duplicates in result! (Why?)
- Schema of result identical to schema of (only) input relation.
- Result relation can be the input for another relational algebra operation! (Operator composition.)

$\sigma_{rating>8}(S2)$

sid	sname	rating	age
28	yuppy	9	35.0
58	rusty	10	35.0

$S2_{(sid,sname,rating,age)}$

$\wedge$ rating > 8

$\pi_{sname,rating}(\sigma_{rating>8}(S2))$

sname	rating
yuppy	9
rusty	10

$\exists$sid,age.

$S2_{(sid,sname,rating,age)}$

$\wedge$ rating > 8

Source: Ramakrishnan & Gehrke

Union, Intersection, Set-Difference

- These operations take two input relations that must be union-compatible:
- Same number of fields.
- "Corresponding" fields have the same type.
- What is the schema of result?

sid	sname	rating	age
22	dustin	7	45.0
28	yuppy	9	35.0
31	lubber	8	55.5
44	guppy	5	35.0
58	rusty	10	35.0

$S1 \cup S2$

sid	sname	rating	age
31	lubber	8	55.5
58	rusty	10	35.0

$S1 \cap S2$

sid	sname	rating	age
22	dustin	7	45.0

$S1 - S2$

Union, Intersection, Set-Difference

- These operations take two input relations that must be union-compatible:
- Same number of fields.
- "Corresponding" fields have the same type.
- What is the schema of result?

sid	sname	rating	age
22	dustin	7	45.0
28	yuppy	9	35.0
31	lubber	8	55.5
44	guppy	5	35.0
58	rusty	10	35.0

$S1 \cup S2$

sid	sname	rating	age
31	lubber	8	55.5
58	rusty	10	35.0

$S1 \cap S2$

sid	sname	rating	age
22	dustin	7	45.0

$S1 - S2$

$S1_{(sid,sname,rating,age)} \vee S2_{(sid,sname,rating,age)}$

Source: Ramakrishnan & Gehrke

Union, Intersection, Set-Difference

- These operations take two input relations that must be union-compatible:
- Same number of fields.
- “Corresponding” fields have the same type.
- What is the schema of result?

S1

sid	sname	rating	age
22	dustin	7	45.0
28	yuppy	9	35.0
31	lubber	8	55.5
44	guppy	5	35.0
58	rusty	10	35.0

S2

$S1 \cup S2$

$S1_{(sid,sname,rating,age)} \vee S2_{(sid,sname,rating,age)}$

or

sid	sname	rating	age
31	lubber	8	55.5
58	rusty	10	35.0

and - суворий

$S1_{(sid,sname,rating,age)} \wedge S2_{(sid,sname,rating,age)}$

$S1 \cap S2$

sid	sname	rating	age
22	dustin	7	45.0

$S1 - S2$

Source: Ramakrishnan & Gehrke

Union, Intersection, Set-Difference

- These operations take two input relations that must be union-compatible:
- Same number of fields.
- "Corresponding" fields have the same type.
- What is the schema of result?

sid	sname	rating	age
22	dustin	7	45.0
28	yuppy	9	35.0
31	lubber	8	55.5
44	guppy	5	35.0
58	rusty	10	35.0

$S1 \cup S2$

$S1_{(sid,sname,rating,age)} \lor S2_{(sid,sname,rating,age)}$

sid	sname	rating	age
31	lubber	8	55.5
58	rusty	10	35.0

$S1_{(sid,sname,rating,age)} \land S2_{(sid,sname,rating,age)}$

$S1 \cap S2$

sid	sname	rating	age
22	dustin	7	45.0

$S1_{(sid,sname,rating,age)} \land \neg S2_{(sid,sname,rating,age)}$

$S1 - S2$

Source: Ramakrishnan & Gehrke

Cross-Product

- Each row of S1 is paired with each row of R1.
- Result schema has one field per field of S1 and R1, with field names "inherited" if possible.
- Conflict: Both S1 and R1 have a field called sid.

R1

sid	bid	day
22	101	10.10
58	103	11.12

S1

sid	sname	rating	age
22	dustin	7	45.0
31	lubber	8	55.5
58	rusty	10	35.0

$S1 \times R1$

sid	sname	rating	age	sid	bid	day
22	dustin	7	45.0	22	101	10.10
22	dustin	7	45.0	58	103	11.12
31	lubber	8	55.5	22	101	10.10
31	lubber	8	55.5	58	103	11.12
58	rusty	10	35.0	22	101	10.10
58	rusty	10	35.0	58	103	11.12

Source: Ramakrishnan & Gehrke (partial)

Cross-Product

- Each row of S1 is paired with each row of R1.
- Result schema has one field per field of S1 and R1, with field names "inherited" if possible.
- Conflict: Both S1 and R1 have a field called sid.

R1

sid	bid	day
22	101	10.10
58	103	11.12

S1

sid	sname	rating	age
22	dustin	7	45.0
31	lubber	8	55.5
58	rusty	10	35.0

S1 × R1

sid	sname	rating	age	sid	bid	day
22	dustin	7	45.0	22	101	10.10
22	dustin	7	45.0	58	103	11.12
31	lubber	8	55.5	22	101	10.10
31	lubber	8	55.5	58	103	11.12
58	rusty	10	35.0	22	101	10.10
58	rusty	10	35.0	58	103	11.12

$S1_{(sid1,sname,rating,age)} \wedge R1_{(sid2,bid,day)}$

Renaming

- The ρ operator gives a new schema to a relation.
- $\rho_{R1(A1,\ldots,An)}(R2)$ makes R1 be a relation with attributes A1,…,An and the same tuples as R2.
- Simplified notation: R1(A1,…,An) := R2.
- In our previous example: Res(sid1,sname,rating,age,sid2,bid,day) := S1 × R1

Source: Bulskov (partial)

Joins

- Condition Join: $R \bowtie_C S = \sigma_C (R \times S)$

- $S1 \bowtie_{S1.sid<R1.sid} R1$

where sid1 < sid2 (condition)

sid	sname	rating	age	sid	bid	day
22	dustin	7	45.0	58	103	11.12
31	lubber	8	55.5	58	103	11.12

- Result schema same as that of cross-product.

- Fewer tuples than cross-product,
  might be able to compute more efficiently

- Sometimes called a theta-join.

Source: Ramakrishnan & Gehrke

Joins

- Condition Join: $R \bowtie_C S = \sigma_C (R \times S)$
- $S1 \bowtie_{S1.sid<R1.sid} R1$

sid	sname	rating	age	sid	bid	day
22	dustin	7	45.0	58	103	11.12
31	lubber	8	55.5	58	103	11.12

$S1_{(sid1,sname,rating,age)} \wedge R1_{(sid2,bid,day)}$

$\wedge\ sid1 < sid2$

- Result schema same as that of cross-product.
- Fewer tuples than cross-product,
  might be able to compute more efficiently
- Sometimes called a theta-join.

Source: Ramakrishnan & Gehrke

Joins

- Equijoin: A special case of condition join where the condition contains only equalities

- S1 $\bowtie_{sid}$ R1

sid	sname	rating	age	bid	day
22	dustin	7	45.0	101	10.10
58	rusty	10	35.0	103	11.12

- Result schema similar to cross-product, but only one copy of fields for which equality is specified.

- Natural Join $\bowtie$: Equijoin on all common fields.

Source: Ramakrishnan & Gehrke

Joins

- Equijoin: A special case of condition join where the condition contains only equalities

- $S1 \bowtie_{sid} R1$

sid	sname	rating	age	bid	day
22	dustin	7	45.0	101	10.10
58	rusty	10	35.0	103	11.12

$S1_{(sid,sname,rating,age)} \wedge R1_{(sid,bid,day)}$

- Result schema similar to cross-product, but only one copy of fields for which equality is specified.

- Natural Join $\bowtie$: Equijoin on all common fields.

Source: Ramakrishnan & Gehrke

Find names of sailors who've reserved boat #103

$\pi_{sname}(\sigma_{bid=103}(R1) \bowtie S1)$

$Tmp1 := \sigma_{bid=103}(R1)$

$Tmp2 := Tmp1 \bowtie S1$

$Tmp3 := \pi_{sname}(Tmp2)$

$\pi_{sname}(\sigma_{bid=103}(R1 \bowtie S1))$

Different ways to state the same query

R1

sid	bid	day
22	101	10.10
58	103	11.12

S1

sid	sname	rating	age
22	dustin	7	45.0
31	lubber	8	55.5
58	rusty	10	35.0

Relational Query Languages

- Query Languages **!=** programming languages!
  - QLs not expected to be “Turing complete”.
  - QLs not intended to be used for complex calculations.
  - QLs support easy, efficient access to large data sets.

- Relational Algebra has limited expressibility
  - What queries are hard to answer?
  - What queries can’t we answer?

Hard, but possible

- Relational Schema:
  - Employees(ssn: integer; sal: real; mgr_ssn: integer).
- **Find the employees with the highest salary:**
  - How would you do it?
- Intuition:
  - Create “dominance” relation: e1 dominates e2 if e1 has salary higher than or equal to e2
  - Divide “dominance” relation by original table: Finds employees who dominate all others
- What about the second highest salary?

Impossible, but very useful

- Relational Schema:
  - Employees(ssn: integer; sal: real; mgr_ssn: integer).
  - Works_In(ssn: integer; did: integer)
- Find the total amount paid in salaries by department
- Problem: **We do not know how to count!**
- What about finding employees that work in exactly three departments?

Impossible, but very useful

- Relational Schema:
  - Employees(ssn: integer; sal: real; mgr_ssn: integer).
  - Works_In(ssn: integer; did: integer)
- Find all the superiors of employee with SSN 123-22-3666
- We want a **Transitive Closure**: Comes in handy when you query graphs
- **Problem: We can’t write arbitrary loops/recursion!**
  - RA operators only implement implicit “foreach element in relation” loops

Extensions to Relational Algebra

- Relational query languages restrict expressiveness to obtain ease of use and of optimization.
- There are some very useful queries we cannot express in the relational algebra.
- Many extensions proposed to handle those queries made into SQL.

We will study an extended relational algebra next!

Extensions to Relational Algebra

- Relational query languages restrict expressiveness to obtain ease of use and of optimization.
- There are some very useful queries we cannot express in the relational algebra.
- Many extensions proposed to handle those queries made into SQL.

We will study an extended relational algebra next!

- Expressions in projections
- Bags vs sets
- Duplicate elimination
- Grouping/aggregation
- Sorting
- Outer joins
- ~~Recursion~~

Extending Projection with Expressions

- Using the same π_L operator, we allow the list L to contain arbitrary expressions involving attributes:
  - Arithmetic on attributes, e.g., A+B→C.
  - Duplicate occurrences of the same attribute.

$\pi_{rating+sid \rightarrow rs, age, age}(S1)$

rs	age	age
29	45.0	45.0
39	55.5	55.5
68	35.0	35.0

S1

sid	sname	rating	age
22	dustin	7	45.0
31	lubber	8	55.5
58	rusty	10	35.0

Source: Bulskov (partial)

Relational Algebra on Bags

- A bag (or multiset) is an unordered collection where duplicates are allowed
  - Like a set, but an element may appear more than once.
- Example: {1,2,1,3} is a bag.
- Example: {1,2,3} is a bag that happens to also be a set.
- SQL uses bag semantics

$\pi_{age}(S2)$

age
35.0
55.5
35.0
35.0

$\sigma_{age<40.0}(\pi_{age}(S2))$

age
35.0
35.0
35.0

S2

sid	sname	rating	age
28	yuppy	9	35.0
31	lubber	8	55.5
44	guppy	5	35.0
58	rusty	10	35.0

Source: Bulskov (partial)

Bag Product

$\sigma_{age<40.0}(\pi_{age}(S2))$

age
35.0
35.0
35.0

$\pi_{sname}(\sigma_{sid<40}(S2))$

sname
yuppy
lubber

$\sigma_{age<40.0}(\pi_{age}(S2)) \times \pi_{sname}(\sigma_{sid<40}(S2))$

age	sname
35.0	yuppy
35.0	yuppy
35.0	yuppy
35.0	lubber
35.0	lubber
35.0	lubber

S2

sid	sname	rating	age
28	yuppy	9	35.0
31	lubber	8	55.5
44	guppy	5	35.0
58	rusty	10	35.0

Source: Bulskov

Bag Union, Intersection, and Difference

- An element appears in the union of two bags the sum of the number of times it appears in each bag.
  - Example: {1,2,1} ∪ {1,1,2,3,1} = {1,1,1,1,1,2,2,3}
- An element appears in the intersection of two bags the minimum of the number of times it appears in either.
  - Example: {1,2,1,1} ∩ {1,2,1,3} = {1,1,2}.
- An element appears in the difference A — B of bags as many times as it appears in A, minus the number of times it appears in B, but never less than 0 times.
  - Example: {1,2,1,1} — {1,2,3} = {1,1}.

Source: Bulskov (partial)

Beware: Bag Laws != Set Laws

- Some, but not all algebraic laws that hold for sets also hold for bags.
- Examples
  - The commutative law for union R ∪ S = S ∪ R holds for bags, since addition is commutative
  - However, set union is idempotent, meaning that S ∪ S = S.
  - For bags, if x appears n times in S, then it appears 2n times in S ∪ S.
  - Thus S ∪ S != S in general, e.g., {1} ∪ {1} = {1,1} != {1}.

Source: Bulskov (partial)

Duplicate Elimination

- R1 := δ(R2).
- R1 consists of one copy of each tuple that appears in R2 one or more times.

π_{age}(S2)

age
35.0
55.5
35.0
35.0

$\delta(\pi_{age}$(S2))

age
35.0
55.5

Aggregation Operators

- Aggregation operators are not operators of relational algebra.
- Rather, they apply to entire columns of a table and produce a single result.
- The most important examples: SUM, AVG, COUNT, MIN, and MAX.

SUM(rating) = 32
COUNT(sid) = 4
MAX(age) = 55.5
AVG(rating) = 8

S2

sid	sname	rating	age
28	yuppy	9	35.0
31	lubber	8	55.5
44	guppy	5	35.0
58	rusty	10	35.0

Source: Bulskov (partial)

Grouping Operator

- γ_L(R) where L is a list of elements that are either:
  - Individual (grouping) attributes.
  - AGG(A), where AGG is one of the aggregation operators and A is an attribute.
    - An arrow and a new attribute name renames the component.

- Semantics
  - Form one group for each distinct list of values in R for grouping attributes in list L.
  - Within each group, compute AGG(A) for each aggregation on list L.
  - Result has one tuple for each group:
    - The grouping attributes and
    - Their group's aggregations.

$\gamma_{\text{age,COUNT(rating)}\rightarrow\text{cr}}$(S2)

cr	age
3	35.0
1	55.5

$\gamma_{\text{age,MAX(rating)}\rightarrow\text{mr}}$(S2)

mr	age
10	35.0
8	55.5

S2

sid	sname	rating	age
28	yuppy	9	35.0
31	lubber	8	55.5
44	guppy	5	35.0
58	rusty	10	35.0

Source: Bulskov (partial)

Sorting

- τ_L(R) is the list of tuples of R sorted first on the value of the first attribute on L, then on the second attribute of L, and so on.
  - Break ties arbitrarily.
- τ is relation itself, but with tuples sorted.
  - Semantics in SQL are murky: result should be a sequence; however, result is treated as a bag if fed to any other operators
  - Some operators can be made order-preserving

τ_{rating}(S2)

sid	sname	rating	age
44	guppy	5	35.0
31	lubber	8	55.5
28	yuppy	9	35.0
58	rusty	10	35.0

S2

sid	sname	rating	age
28	yuppy	9	35.0
31	lubber	8	55.5
44	guppy	5	35.0
58	rusty	10	35.0

Source: Bulskov (partial)

Outer Join

- Suppose we join R $\bowtie$ S.
- A tuple of R that has no tuple of S with which it joins is called dangling.
  - Similarly for a tuple of S.
- Outer join $\overset{o}{\bowtie}$ preserves dangling tuples by padding them with $\perp$ (NULL in SQL).
- Variants that preserve only left/right dangling tuples: $\overset{o}{\bowtie}_L$, $\overset{o}{\bowtie}_R$

$$S1 \overset{o}{\bowtie}_L R1 = S1 \overset{o}{\bowtie} R1$$

sid	sname	rating	age	bid	day
22	dustin	7	45.0	101	10.10
58	rusty	10	35.0	103	11.12
31	lubber	8	55.5	$\perp$	$\perp$

$$S1 \overset{o}{\bowtie}_R R1 = S1 \bowtie R1$$

R1

sid	bid	day
22	101	10.10
58	103	11.12

S1

sid	sname	rating	age
22	dustin	7	45.0
31	lubber	8	55.5
58	rusty	10	35.0

Source: Bulskov (partial)

What should we learn today?

- Understand the notion of domain independence and be able to argue whether a given query is domain independent
- Understand the relational algebra normal form (RANF) and be able to determine whether a relational calculus query is in RANF
- Explain the operators of the relational algebra
- Formulate queries in the relational algebra
- Explain limitations of the relational algebra in terms of query expressiveness
- Formulate queries with the main extensions of the relational algebra including bag semantics, grouping, aggregation, sorting, and outer join