Updated 2/11/2022

# Topic 1: Data models and query languages
# Unit 3: Relational Algebra (continued)
# Lecture 08

Wolfgang Gatterbauer

CS7240 Principles of scalable data management (sp22)

https://northeastern-datalab.github.io/cs7240/sp22/

2/11/2022

# Pre-class conversations

- Last class recapitulation
- Please keep on pointing out any errors on the slides
- It is time to start to hand in your first ~~scribble~~ notes

  scribes
- Project discussions
- Where we are

- today:
  - Algebra: independence and Codd's theorem
  - Recursion (Datalog)

*Topic 1: Data Models and Query Languages*

- **Lecture 1 (Tue 1/18):** Course introduction / SQL / PostgreSQL setup / SQL Activities
- **Lecture 2 (Fri 1/21):** SQL
- **Lecture 3 (Tue 1/25):** SQL
- **Lecture 4 (Fri 1/28):** SQL, Logic & Relational Calculus
- **Lecture 5 (Tue 2/1):** Logic & Relational Calculus
- **Lecture 6 (Fri 2/4):** Relational Algebra & Codd's Theorem
- **Lecture 7 (Tue 2/8):** Relational Algebra & Codd's Theorem / Datalog & Recursion
- **Lecture 8 (Fri 2/11):** Datalog & Recursion
- **Lecture 9 (Tue 2/15):** Alternative Data Models

Pointers to relevant concepts & supplementary material:

- **Unit 1. SQL**: [SAMS'12], [CS 3200], [Cow'03] Ch3 & Ch5, [Complete'08] Ch6, [Silberschatz+'20] Ch3.8
- **Unit 2. Logic & Relational Calculus**: First-Order Logic (FOL), relational calculus (RC): [Barland+'08] 4.1.2 & 4.2.1 & 4.4, [Genesereth+] Ch6, [Halpern+'01], [Cow'03] Ch4.3 & 4.4, [Elmasri, Navathe'15] Ch8.6 & Ch8.7, [Silberschatz+'20] Ch27.1 & Ch27.2, [Alice'95] Ch3.1-3.3 & Ch4.2 & Ch4.4 & Ch5.3-5.4, [Barker-Plummer+'11] Ch11
- **Unit 3. Relational Algebra & Codd's Theorem**: Relational Algebra (RA), Codd's theorem: [Cow'03] Ch4.2, [Complete'08] Ch2.4 & Ch5.1-5.2, [Elmasri, Navathe'15] Ch8, [Silberschatz+'20] Ch2.6, [Alice'95] Ch4.4 & Ch5.4
- **Unit 4. Datalog & Recursion**: Datalog, recursion, Stratified Datalog with negation, Datalog evaluation strategies, Stable Model semantics, Answer Set Programming (ASP): [Complete'08] Ch5.3, [Cow'03] Ch 24, [Koutris'19] L9 & L10, [G., Suciu'10]
- **Unit 5. Alternative Data Models**: NoSQL: [Hellerstein, Stonebraker'05], [Sadalage, Fowler'12], [Harrison'16]

# Algebra and the connection to logic and queries

- Algebra
- Relational Algebra
  - Operators
  - Independence
  - Power of algebra: optimizations
- Equivalence RA and safe RC (Codd's theorem)

# 5 Primitive Operators

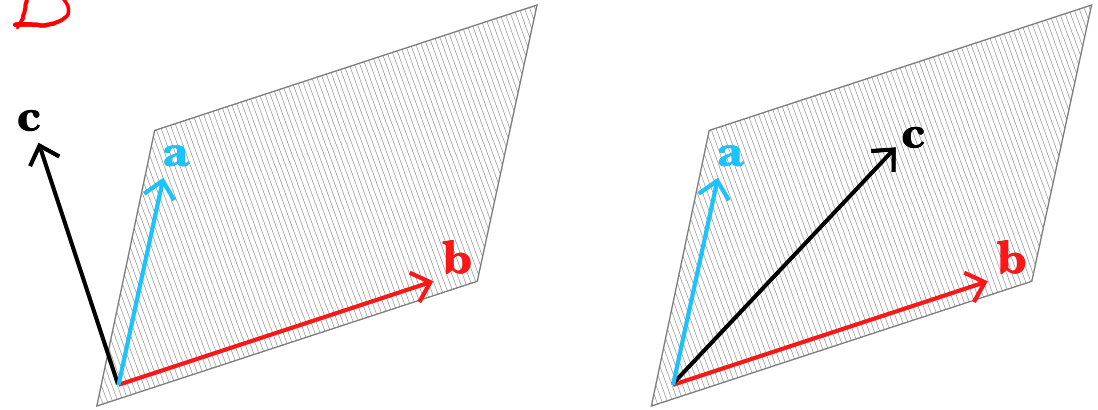
1. Projection ($\pi$)
2. Selection ($\sigma$)
3. Union ($\cup$)
4. Set Difference ($-$)
5. Cross Product ( $\times$ )

Is this a well chosen set of primitives? ?

# 5 Primitive Operators

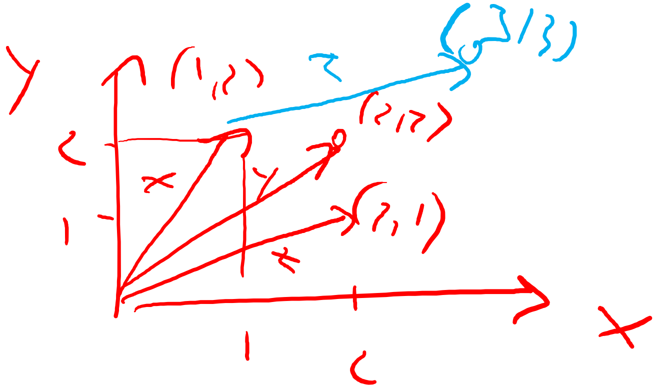
G 3D

1. Projection ($\pi$)
2. Selection ($\sigma$)
3. Union ($\cup$)
4. Set Difference ($-$)
5. Cross Product ($\times$)




INDEP.

7.I.




$y = (x+z) \cdot \frac{2}{3}$

$2x + 3y + 2z = 0$

$\frac{2}{3} \quad -1 \quad \frac{2}{3}$

Is this a well chosen set of primitives?

Could we drop an operator "without losing anything"?

# Independence among Primitives

- Let o be an RA operator, and let A be a set of RA operators
- We say that o is independent of A if o cannot be expressed in A; that is, no expression in A is equivalent to o

THEOREM: Each of the five primitives is independent of the other four

$\{\pi, \sigma, \times, \cup, -\}$

Proof:

- Separate argument for each of the five
- Arguments follow a common pattern (next slide)
- We will do one operator here (union)

# Recipe for Proving Independence of an operator o

1. Fix a schema *S* and an instance I over *S*

2. Find some property P over relations

3. Prove: for every expression φ that does not use o, the relation φ(I) satisfies P

   Such proofs are typically by induction on the size of the expression, since operators compose

4. Find an expression ψ such that ψ uses o and ψ(I) violates P

# Concrete Example: Proving Independence of Union U

1. Fix a schema *S* and an instance I over *S*

   *S*: R(A), S(A)        I: {R(0), S(1)}

*R*

| A |
|---|
| 0 |

*S*

| A |
|---|
| 1 |

2. Find some property P over relations

   #tuples < 2

3. Prove: for every expression $\varphi$ that <u>does not use</u> o, the relation $\varphi(I)$ satisfies P

   **Induction base:** R and S have #tuples<2

   **Induction step:** If $\varphi_1(I)$ and $\varphi_2(I)$ have #tuples<2, then so do:

   $\sigma_c(\varphi_1(I))$, $\pi_A(\varphi_1(I))$, $\varphi_1(I) \times \varphi_2(I)$, $\varphi_1(I) - \varphi_2(I)$, $\rho_{A \to B}(\varphi_1(I))$

4. Find an expression $\psi$ such that $\psi$ uses o and $\psi(I)$ violates P

   $\psi = R \cup S$

# Algebra and the connection to logic and queries

- Algebra
- Relational Algebra
  - Operators
  - Independence
  - Power of algebra: optimizations
- Equivalence RA and safe RC (Codd's theorem)

# Commutativity and distributivity of RA operators

- The basic commutators:
  - Push projection through selection, join, union
  - Push selection through projection, join, union
  - Also: Joins can be re-ordered!

$$\pi_{A_1,\ldots,A_n}(R \cup S) = \pi_{A_1,\ldots,A_n}(R) \cup \pi_{A_1,\ldots,A_n}(S)$$

$$\sigma_c(R \cup S) = \sigma_c(R) \cup \sigma_c(S)$$

$$(R \cup S) \times T = R \times T \cup S \times T$$

$$T \times (R \cup S) = T \times R \cup T \times S$$

What about sorting and joins?

- Note that this is not an exhaustive set of operations

This simple set of tools allows us to greatly improve the execution time of queries by optimizing RA plans!

We next illustrate with an SFW (Select-From-Where) query

# An example: SQL to RA to Optimized RA

```
R(A,B) S(B,C) T(C,D)
```

```
SELECT R.A,T.D
FROM   R,S,T
WHERE  R.B = S.B
   and S.C = T.C
   and R.A < 10;
```

in RA

?

# An example: SQL to RA to Optimized RA

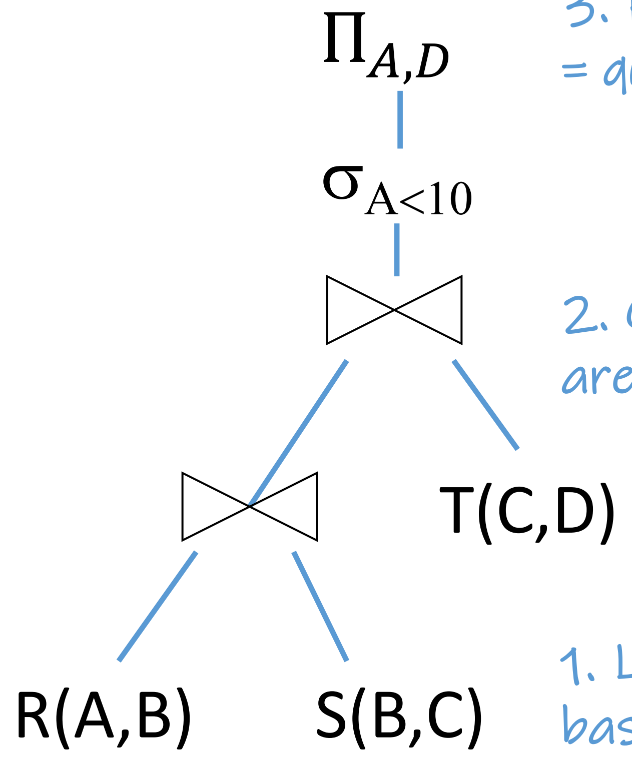
Heuristic: have selection and projection earlier to have fewer (or smaller) "intermediate" tuples

```
R(A,B) S(B,C) T(C,D)
```

```sql
SELECT R.A,T.D
FROM   R,S,T
WHERE  R.B = S.B
   and S.C = T.C
   and R.A < 10;
```

in RA

$$\Pi_{A,D}\left(\sigma_{A<10}\left(T \bowtie (R \bowtie S)\right)\right)$$




3. Root node = query results

2. Other nodes are operators

1. Leaves are base relations

Query tree / expression tree / computation tree / data flow graph

# An example: SQL to RA to Optimized RA

Heuristic: have selection and projection earlier to have fewer (or smaller) "intermediate" tuples

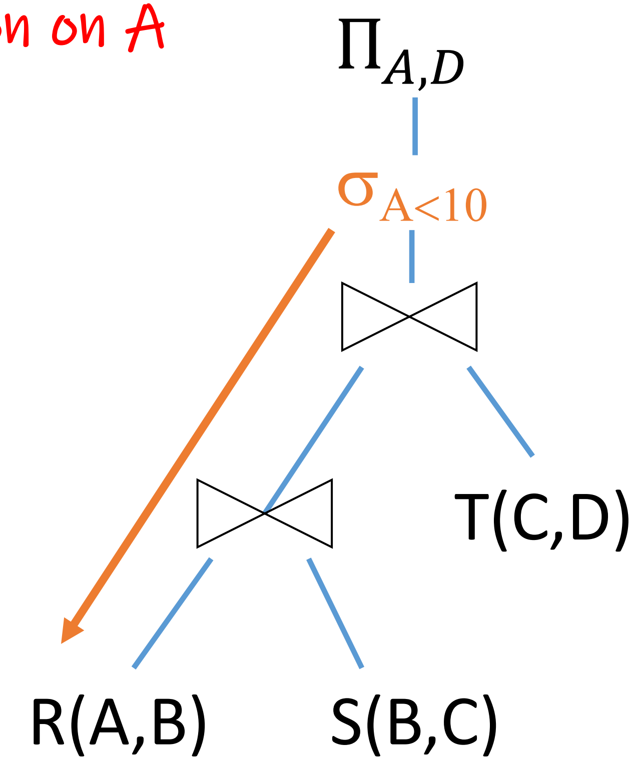
```
R(A,B) S(B,C) T(C,D)
```

```sql
SELECT R.A,T.D
FROM   R,S,T
WHERE  R.B = S.B
   and S.C = T.C
   and R.A < 10;
```

in RA

$$\Pi_{A,D}\left(\sigma_{A<10}\left(T \bowtie (R \bowtie S)\right)\right)$$

1. Push down selection on A




Pushing down may be suboptimal if selection condition is very expensive (e.g. running some image processing algorithm). Projection could be unnecessary effort (but more rarely).

# An example: SQL to RA to Optimized RA

Heuristic: have selection and projection earlier to have fewer (or smaller) "intermediate" tuples

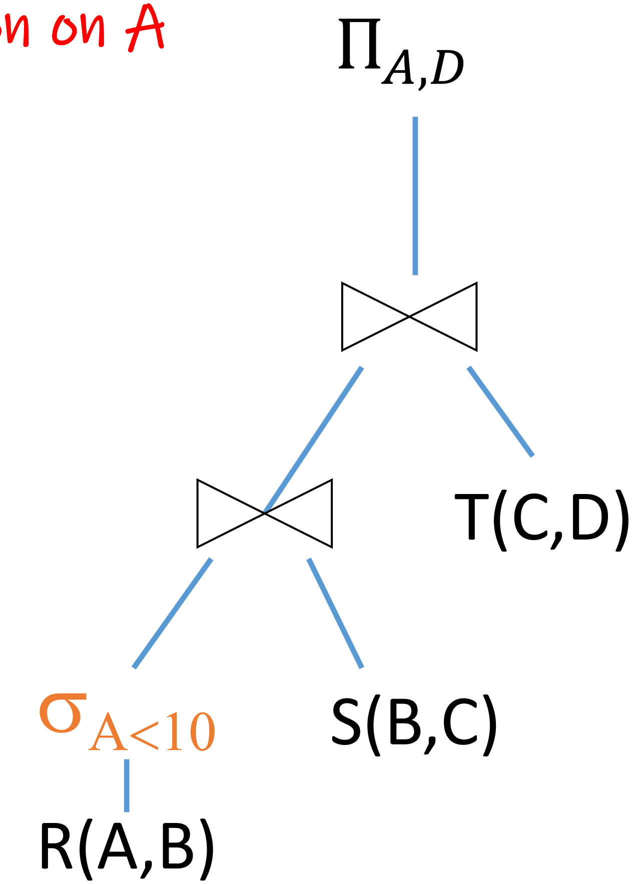
```
R(A,B) S(B,C) T(C,D)
```

```sql
SELECT R.A,T.D
FROM   R,S,T
WHERE  R.B = S.B
   and S.C = T.C
   and R.A < 10;
```

in RA

$$\Pi_{A,D}\left(T \bowtie \left(\sigma_{A<10} R \bowtie S\right)\right)$$

1. Push down selection on A



# An example: SQL to RA to Optimized RA

Heuristic: have selection and projection earlier to have fewer (or smaller) "intermediate" tuples

```
R(A,B) S(B,C) T(C,D)
```

```sql
SELECT R.A,T.D
FROM   R,S,T
WHERE  R.B = S.B
   and S.C = T.C
   and R.A < 10;
```

in RA

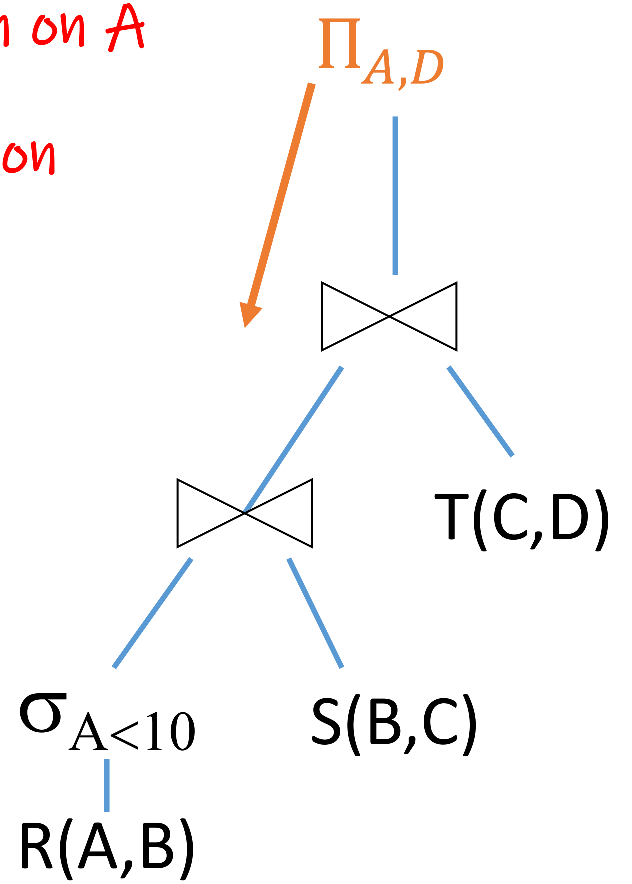
$$\Pi_{A,D}\left(T \bowtie (\sigma_{A<10} R \bowtie S)\right)$$

$\Pi_{-B,C}$

1. Push down selection on A

2. Push down projection

$\Pi_{-B,C}$



# An example: SQL to RA to Optimized RA

Variable Elimination!

```
R(A,B) S(B,C) T(C,D)
```

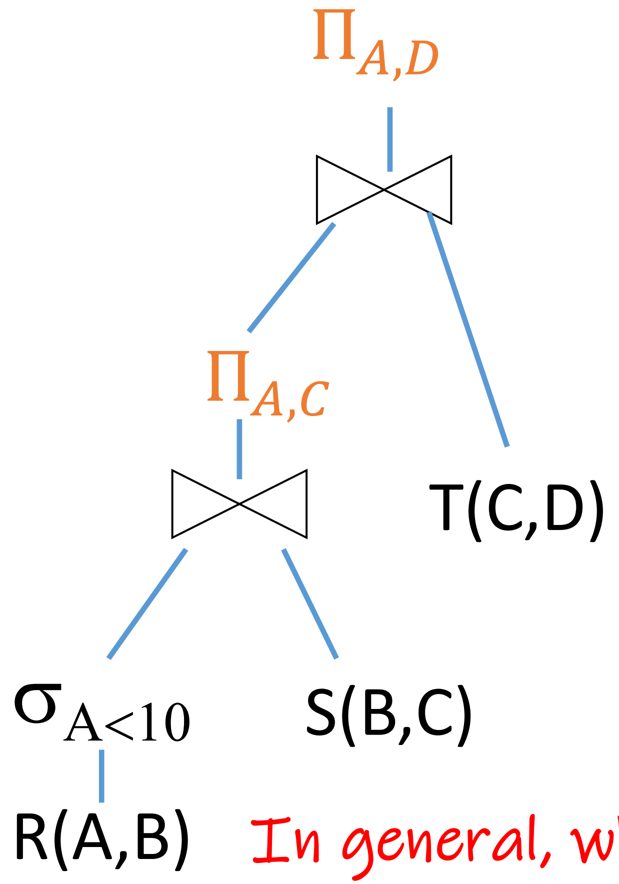
```sql
SELECT R.A,T.D
FROM   R,S,T
WHERE  R.B = S.B
   and S.C = T.C
   and R.A < 10;
```

in RA

$$\Pi_{A,D}\Big(T \bowtie \Pi_{A,C}(\sigma_{A<10} R \bowtie S)\Big)$$

$\Pi_{-C}$ $\Pi_{-B}$

We now eliminate B earlier




$\Pi_{-C}$

$\Pi_{-B}$

In general, when is an attribute not needed?

# Algebra and the connection to logic and queries

- Algebra
- Relational Algebra
  - Operators
  - Independence
  - Power of algebra: optimizations
- Equivalence RA and safe RC (Codd's theorem)

# An example

Person(id, name, country)
Spouse(id1, id2)

In RC:

$$\{\ x \mid \exists z,w.\ \text{Person}(x,z,w) \land \forall y.[\neg \text{Spouse}(x,y)]\ \}$$

In RA:

?

# An example

Person(id, name, country)
Spouse(id1, id2)

In RC:

$$\{\, x \mid \exists z,w.\ \mathrm{Person}(x,z,w) \wedge \forall y.[\neg \mathrm{Spouse}(x,y)]\,\}$$

R(A,B) — S(C,D)

S(D,C)

In RA:

$$\pi_{id}\mathrm{Person} - \pi_{id1}\mathrm{Spouse}$$

$$\pi_{id}\mathrm{Person} - \rho_{id1\rightarrow id}\,(\pi_{id1}\mathrm{Spouse})$$

Recall: named vs ordered perspective

# Equivalence Between RA and Domain-Independent RC

CODD'S THEOREM:
RA and **domain-independent** RC
have the same expressive power.

More formally, on every schema $\mathbf{S}$:

1. For every RA expression $\mathrm{E}$, there is a domain-independent RC query $\mathrm{Q}$ s.t. $\mathrm{Q} \equiv \mathrm{E}$
2. For every domain-independent RC query $\mathrm{Q}$, there is an RA expression $\mathrm{E}$ s.t. $\mathrm{Q} \equiv \mathrm{E}$

The proof has two directions:

RA→RC:
by induction on the size of the RA expression

RC→RA:
more involved

See also: https://en.wikipedia.org/wiki/Codd%27s_theorem

# RA → DRC: Intuition

Intuition: $\{x \mid \exists y.[R(x,y)] \land \exists y.[S(x,y)]\}$

contrast with: $\{x \mid \exists y.[R(x,y)] \land \exists z.[S(x,z)]\}$

- Construction by induction
- Key technical detail: need to maintain a mapping b/w attribute names and variables

$Q(1) \leftarrow R(1,2), S(1,3)$

$y=2 \quad y=3$

| RA expression | DRC formula $\phi$ Here, $\phi_i$ is the formula constructed for expression $E_i$ |
|---|---|
| R (n columns) | $R(X_1,\ldots,X_n)$ |
| $E_1 \times E_2$ | |
| $E_1 \cup E_2$ | |
| $E_1 - E_2$ | |
| $\pi_{a_1,\ldots,a_k}(E_1)$ | |
| $\sigma_c(E_1)$ | |

# RA → DRC: Intuition

Intuition: $\{x \mid \exists y.[R(x,y)] \wedge \exists y.[S(x,y)]\}$

contrast with: $\{x \mid \exists y.[R(x,y)] \wedge \exists z.[S(x,z)]\}$

- Construction by induction
- Key technical detail: need to maintain a mapping b/w attribute names and variables

| RA expression | DRC formula $\phi$ Here, $\phi_i$ is the formula constructed for expression $E_i$ |
|---|---|
| R (n columns) | $R(X_1,\ldots,X_n)$ |
| $E_1 \times E_2$ | $\phi_1 \wedge \phi_2$ disjoint variables (rename) |
| $E_1 \cup E_2$ | $\phi_1 \vee \phi_2$ use identical variables (rename) UNION COMPATIBLE |
| $E_1 - E_2$ | $\phi_1 \wedge \neg\phi_2$ use identical variables (rename) |
| $\pi_{a_1,\ldots,a_k}(E_1)$ | |
| $\sigma_c(E_1)$ | |

# RA → DRC: Intuition

Intuition: $\{x \mid \exists y.[R(x,y)] \wedge \exists y.[S(x,y)]\}$

contrast with: $\{x \mid \exists y.[R(x,y)] \wedge \exists z.[S(x,z)]\}$

- Construction by induction
- Key technical detail: need to maintain a mapping b/w attribute names and variables

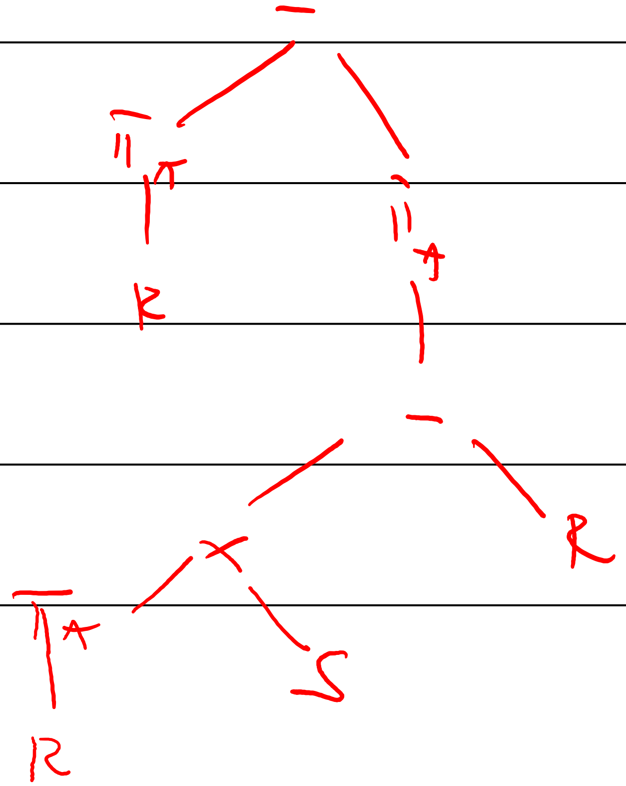
| RA expression | DRC formula $\phi$ Here, $\phi_i$ is the formula constructed for expression $E_i$ |
|---|---|
| R (n columns) | $R(X_1,\ldots,X_n)$ |
| $E_1 \times E_2$ | $\phi_1 \wedge \phi_2$ disjoint variables (rename) |
| $E_1 \cup E_2$ | $\phi_1 \vee \phi_2$ use identical variables (rename) |
| $E_1 - E_2$ | $\phi_1 \wedge \neg\phi_2$ use identical variables (rename) |
| $\pi_{a_1,\ldots,a_k}(E_1)$ | $\exists X_1 \ldots \exists X_m. \phi_1$ where $X_1, \ldots, X_m$ are the variables not among $a_1,\ldots, a_k$ |
| $\sigma_c(E_1)$ | $\phi_1 \wedge c$ |

Correspondence more natural with project-away operator: $\pi_{-a_1,\ldots,a_m}(E_1)$

# RA → DRC: Example R÷S

R(A,B) S(B)

| RA | DRC | Mapping |
|---|---|---|
| $R$ | | |
| $\pi_A(R)$ | | |
| $S$ | | |
| $\pi_A(R) \times S$ | | |
| $(\pi_A(R) \times S) - R$ | | |
| $\pi_A\big((\pi_A(R) \times S) - R\big)$ | | |
| $\pi_A(R) - \pi_A\big((\pi_A(R) \times S) - R\big)$ | | |



# RA → DRC: Example R÷S

R(A,B)   S(B)

| RA | DRC | Mapping |
|---|---|---|
| $R$ | $R(x, y)$ | x:R.A, y:R.B |
| $\pi_A(R)$ | $\exists y. R(x, y)$ | x:R.A |
| $S$ | $S(z)$ | z:S.B |
| $\pi_A(R)\times S$ | | |
| $(\pi_A(R)\times S) - R$ | | |
| $\pi_A\big((\pi_A(R)\times S) - R\big)$ | | |
| $\pi_A(R) -$ <br> $\pi_A\big((\pi_A(R)\times S) - R\big)$ | | |

# RA → DRC: Example R÷S

R(A,B)   S(B)

| RA | DRC | Mapping |
|---|---|---|
| $R$ | $R(x,y)$ | x:R.A, y:R.B |
| $\pi_A(R)$ | $\exists y.\, R(x,y)$ | x:R.A |
| $S$ | $S(z)$ | z:S.B |
| $\pi_A(R)\times S$ | $\exists y.\, \boxed{R(x,y)} \wedge S(z)$ z needs to be different from y | x:R.A, z:S.B |
| $(\pi_A(R)\times S) - R$ | $\big(\exists y.\, R(x,y) \wedge S(z)\big) \wedge \neg R(x,z)$ | x:R.A, z:S.B |
| $\pi_A\big((\pi_A(R)\times S) - R\big)$ | $\exists z\big[\big(\exists y.\, R(x,y) \wedge S(z)\big) \wedge \neg R(x,z)\big]$ | x:R.A |
| $\pi_A(R) - \pi_A\big((\pi_A(R)\times S) - R\big)$ | $\exists y.\, R(x,y) \wedge \neg\exists z\big[\big(\exists y.\, R(x,y) \wedge S(z)\big) \wedge \neg R(x,z)\big]$ x's need to be same variable | x:R.A |

y's don't need to be same variable

# "Clear" variables*

Formula with clear variables : each quantifier "has its own variables" & each variable has only free or only bound occurrences

$$\forall x.\, \exists y.\, R(x, y, z) \wedge \neg\exists x.\, S(y, x)$$

? which variables are free or bound?

* "Clear variable" is used in slides by Marie Duzi: http://www.cs.vsb.cz/duzi/

# "Clear" variables*

Formula with clear variables : each quantifier "has its own variables" & each variable has only free or only bound occurrences

$$\forall x.\ \exists y.\boxed{R(x, y, z)} \wedge \neg\exists x.\ S(y, x)$$

bound      free      bound

recall operator precedence: ∃ before ∧

$\forall x.\exists y.[R(x,y,z)] \wedge \neg\exists x.[S(y,x)]$

Not "clear": Two x's and y's are different variables.

? how to make it "clear"

* "Clear variable" is used in slides by Marie Duzi: http://www.cs.vsb.cz/duzi/

# "Clear" variables*

Formula with clear variables : each quantifier "has its own variables" & each variable has only free or only bound occurrences

$$\forall x.\, \exists y.\, R(x, y, z) \wedge \neg \exists x.\, S(y, x)$$

bound free bound

recall operator precedence: ∃ before ∧
$\forall x. \exists y. [R(x,y,z)] \wedge \neg \exists x. [S(y,x)]$

Not "clear": Two x's and y's are different variables.

$$\forall x.\, \exists y.\, R(x, y, z) \wedge \neg \exists u.\, S(v, u)$$

now "clear"

$$\{(z, v) \mid \forall x.\, \exists y.\, R(x, y, z) \wedge \neg \exists u.\, S(v, u)\}$$

Now a query. But how to make it domain-independent ?

* "Clear variable" is used in slides by Marie Duzi: http://www.cs.vsb.cz/duzi/

# "Clear" variables*

Formula with clear variables : each quantifier "has its own variables" & each variable has only free or only bound occurrences

$$\forall x.\exists y.R(x,y,z) \wedge \neg\exists x.S(y,x)$$

bound free bound

recall operator precedence: $\exists$ before $\wedge$
$\forall x.\exists y.[R(x,y,z)] \wedge \neg\exists x.[S(y,x)]$

Not "clear": Two x's and y's are different variables.

$$\forall x.\exists y.R(x,y,z) \wedge \neg\exists u.S(v,u)$$

now "clear"

$\exists s,t.R(s,t,z) \wedge$ $\exists p.S(p,v) \wedge$

$$\{(z,v) \mid \cancel{\forall x.\exists y.R(x,y,z)} \wedge \neg\exists u.S(v,u)\}$$

$\forall x.[\exists w,t.R(x,w,t) \rightarrow \exists y.R(x,y,z)]$

Now a query. But how to make it domain-independent

# Repeated variable names

In sentences with multiple quantifiers, distinct variables do not need to range over distinct objects! (cp. homomorphism vs. isomorphism)

? Which of the following formulas imply each other?

$\forall x.\forall y.\ E(x,y)$

$\forall x.\ E(x,x)$

$\exists x.\exists y.\ E(x,y)$

$\exists x.\ E(x,x)$

# Repeated variable names

In sentences with multiple quantifiers, <u>distinct variables do not need to range over distinct objects</u>! (cp. homomorphism vs. isomorphism)

Assume DOM = {1, 2}:

E

| s | t |
|---|---|
| 1 | 1 |
| 1 | 2 |
| 2 | 1 |
| 2 | 2 |

$\forall x. \forall y.\, E(x,y) \quad \Rightarrow \quad \forall x.\, E(x,x)$

$\exists x. \exists y.\, E(x,y) \quad \Leftarrow \quad \exists x.\, E(x,x)$

E

| s | t |
|---|---|
| 1 | 2 |

# Repeated variable names

In sentences with multiple quantifiers, distinct variables do not need to range over distinct objects! (cp. homomorphism vs. isomorphism)

Assume DOM = $\emptyset$ :

$$
\begin{array}{ccc}
\forall x.\forall y.\ E(x,y) & \Rightarrow & \forall x.\ E(x,x) \\
\Downarrow & & \Downarrow \\
\exists x.\exists y.\ E(x,y) & \Leftarrow & \exists x.\ E(x,x)
\end{array}
$$

Only if domain is not empty! Dom $\neq \emptyset$

E

| | |
|---|---|
| s | t |

# DRC → RA: Intuition

Proof (Sketch):

- Show first that for every relational database schema **S**, there is a relational algebra expression E such that for every database instance **D**, we have that ADom(D) = E(D).
- Use the above fact and induction on the construction of RC formulas to obtain a translation of RC under the active domain interpretation to RA.

# DRC → RA: Intuition

E(A,B)

- In this translation, the most interesting part is the simulation of the universal quantifier ∀ in relational algebra

uses the logical equivalence: $\forall y.\, \phi \equiv$ ?

# DRC → RA: Intuition

$E(A,B)$

- In this translation, the most interesting part is the simulation of the universal quantifier ∀ in relational algebra

  uses the logical equivalence: $\forall y. \phi \equiv \neg \exists y. \neg \phi$

- As an illustration, consider: $\forall y. E(x, y) \equiv$ ?

# DRC → RA: Intuition

$E(A,B)$

- In this translation, the most interesting part is the simulation of the universal quantifier ∀ in relational algebra

  uses the logical equivalence: $\forall y.\, \phi \equiv \neg\exists y.\, \neg\phi$

- As an illustration, consider: $\forall y.\, E(x, y) \equiv \neg\exists y.\, \neg E(x, y)$

  and recall: $\mathrm{ADom}(D) =$ ?

# DRC → RA: Intuition

$E(A,B)$

- In this translation, the most interesting part is the simulation of the universal quantifier ∀ in relational algebra

  uses the logical equivalence: $\forall y.\,\phi \equiv \neg\exists y.\,\neg\phi$

- As an illustration, consider: $\forall y.\,E(x,y) \equiv \neg\exists y.\,\neg E(x,y)$

  and recall: $\boxed{ADom(D) = \pi_A(E) \cup \pi_B(E)}$

| DRC formula $\phi$ | RA expression for $\phi^{adom}$ |
|---|---|
| $\neg E(x,y)$ | |
| $\exists y.\,\neg E(x,y)$ | ? |
| $\neg\exists y.\,\neg E(x,y)$ | |

# DRC → RA: Intuition

$E(A,B)$

- In this translation, the most interesting part is the simulation of the universal quantifier ∀ in relational algebra

  uses the logical equivalence: $\forall y.\, \phi \equiv \quad \neg\exists y.\, \neg\phi$

- As an illustration, consider: $\forall y.\, E(x,y) \equiv \neg\exists y.\, \neg E(x,y)$

  and recall: $\boxed{ADom(D) = \pi_A(E) \cup \pi_B(E)}$

| DRC formula $\phi$ | RA expression for $\phi^{adom}$ |
|---|---|
| $\neg E(x,y)$ | $(ADom(D) \times ADom(D)) - E$ |
| $\exists y.\, \neg E(x,y)$ | $\pi_A[(ADom(D) \times ADom(D)) - E]$ |
| $\neg\exists y.\, \neg E(x,y)$ | $ADom(D) - \pi_A[(ADom(D) \times ADom(D)) - E]$ |

# Entire Story in One Slide (repeated slide)

1. RC = FOL over DB
2. RC can express “bad queries” that depend not only on the DB, but also on the domain from which values are taken (domain dependence)
3. We cannot test whether an RC query is “good,” but we can use a ”good” subset of RC that captures all “good” queries (safety)
4. “Good” RC and RA can express the same queries! (equivalence = Codd's theorem)

# Discussion

- What is the monotone fragment of RA ?

?

- What are the safe queries in RA ?

?

- Where do we use RA (applications) ?

?

# Discussion

- What is the monotone fragment of RA ?
  - Basic except difference ($-$): $\cup$, $\sigma$, $\pi$, $\bowtie$

- What are the safe queries in RA ?

?

- Where do we use RA (applications) ?

?

# Discussion

- What is the monotone fragment of RA ?
  - Basic except difference ($-$): $\cup$, $\sigma$, $\pi$, $\bowtie$
- What are the safe queries in RA ?
  - All RA queries are safe
- Where do we use RA (applications) ?

?

Source: Dan Suciu, CSE 554, 2011.

# Discussion

- What is the monotone fragment of RA ?
  - Basic except difference (−): $\cup, \sigma, \pi, \bowtie$
- What are the safe queries in RA ?
  - All RA queries are safe
- Where do we use RA (applications) ?
  - Translating SQL (from WHAT to HOW)
  - Directly as query languages (e.g. Pig-Latin)

See next pages

Source: Dan Suciu, CSE 554, 2011.

EXAMPLE 1. *Suppose we have a table* **urls: (url, category, pagerank)**. *The following is a simple SQL query that finds, for each sufficiently large category, the average pagerank of high-pagerank urls in that category.*

```
SELECT category, AVG(pagerank)
FROM urls WHERE pagerank > 0.2
GROUP BY category HAVING COUNT(*) > 10^6
```

*An equivalent Pig Latin program is the following. (Pig Latin is described in detail in Section 3; a detailed understanding of the language is not required to follow this example.)*

```
good_urls = FILTER urls BY pagerank > 0.2;
groups = GROUP good_urls BY category;
big_groups = FILTER groups BY COUNT(good_urls)>10^6;
output = FOREACH big_groups GENERATE
             category, AVG(good_urls.pagerank);
```

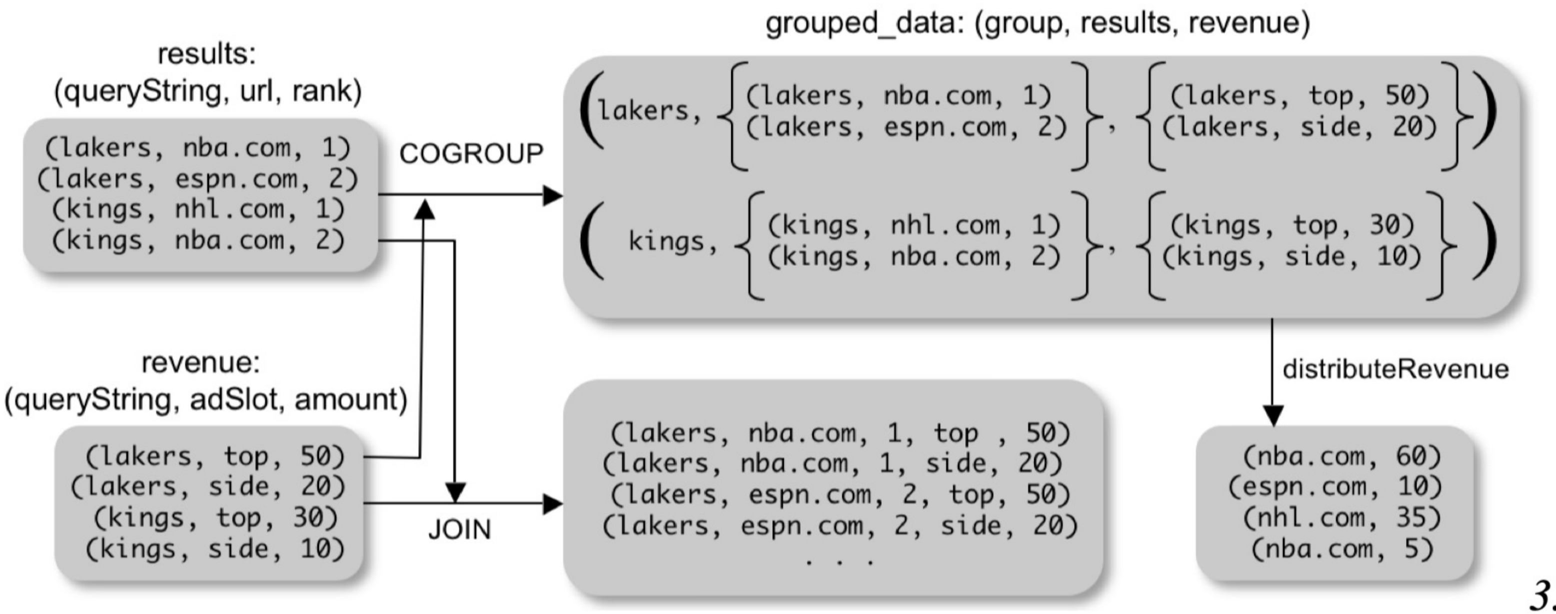



**Figure 2: COGROUP versus JOIN.**

```
grouped_data  =  COGROUP results BY queryString,
                         revenue BY queryString;
```

### *3.5.2* JOIN *in Pig Latin*

Not all users need the flexibility offered by COGROUP. In many cases, all that is required is a regular equi-join. Thus, Pig Latin provides a JOIN keyword for equi-joins. For example,

```
join_result  =  JOIN results BY queryString,
                revenue BY queryString;
```

It is easy to verify that JOIN is only a syntactic shortcut for COGROUP followed by flattening. The above join command is equivalent to:

```
   temp_var  =  COGROUP results BY queryString,
                revenue BY queryString;
join_result  =  FOREACH temp_var GENERATE
                FLATTEN(results), FLATTEN(revenue);
```