Updated 2/8/2020

T1: Data models and query languages
L4: Relational algebra, Codd's theorem

Wolfgang Gatterbauer

CS7240 Principles of scalable data management (sp20)

https://northeastern-datalab.github.io/cs7240/sp20/

Version 1/17/2020

Where we are

Topic 1: Data models and query languages

- **Lecture 1 (Tue 1/7):** Course introduction, SQL refresher
    - Introduction, SQL
- **Lecture 2 (Fri 1/10):** Logic & relational calculus
    - SQL continued, Logic & relational calculus
- **Lecture 3 (Tue 1/14):** Relational Calculus, Relational algebra
    - Relational algebra
- **Lecture 4 (Fri 1/17):** Codd's theorem, Datalog
- **Lecture 5 (Tue 1/21):** Stable model semantics, Information theory & normal forms
- **Lecture 6 (Fri 1/24): (A1 due)** Alternative data models

Parentheses Convention

- We have defined 3 unary operators and 3 binary operators
- It is acceptable to omit the parentheses from o(R) when o is unary
  - Then, unary operators take precedence over binary ones
- Example:

$$(\sigma_{course='DB'}(\text{Course})) \times (\rho_{cid/cid1}(\text{Studies}))$$

becomes

$$\sigma_{course='DB'}\text{Course} \times \rho_{cid/cid1}\text{Studies}$$

Queries and the connection to logic and algebra

- Why logic?
  - A crash course on FOL
- Relational Calculus
  - Syntax and Semantics
  - Domain Independence and Safety
- Relational Algebra
  - Operators
  - Independence
  - Power of algebra: optimizations
- Equivalence RC and RA

6 Primitive Operators

1. Projection (π)
2. Selection (σ)
3. Renaming (ρ)
4. Union ($\cup$)
5. Set Difference ($-$)
6. Cross Product ($\times$)

Q: Is this a "good" 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 six primitives is independent of the other five

$\pi \quad \sigma \quad \rho \quad \cup \quad - \quad \times$

Proof:

- Separate argument for each of the six
- 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

Independence of Union ∪

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 φ that <u>does not use</u> o, the relation $\varphi(\mathrm{I})$ satisfies P

   Induction base: R and S have #tuples<2

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

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

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

   $\psi = R \cup S$

Queries and the connection to logic and algebra

- Why logic?
  - A crash course on FOL
- Relational Calculus
  - Syntax and Semantics
  - Domain Independence and Safety
- Relational Algebra
  - Operators
  - Independence
  - Power of algebra: optimizations
- Equivalence RC and RA

RA commutators

- The basic commutators:
  - Push <u>projection</u> through (1) selection, (2) join
  - Push <u>selection</u> through (3) selection, (4) projection, (5) join
  - Also: Joins can be re-ordered!

- 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!

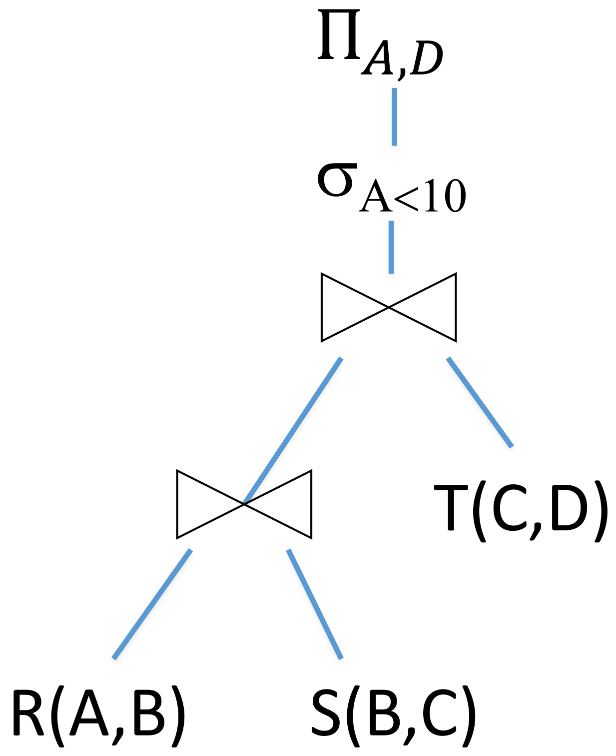
We next illustrate with an SFW (Select-From-Where) query

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;
```

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



An example: SQL to RA to Optimized RA

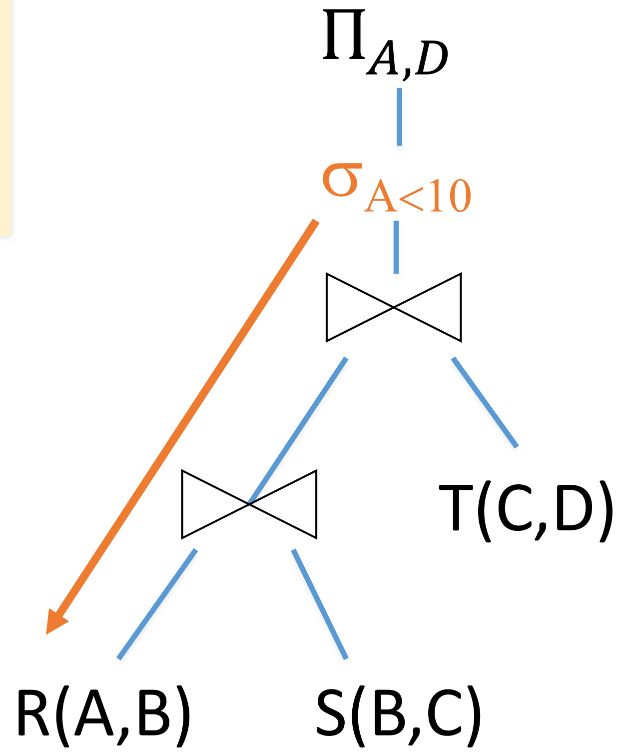
Heuristically, we want selection and projection to occur early to have fewer or smaller "intermediate" tuples

```
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;
```

Push down selection on A so it occurs earlier

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



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

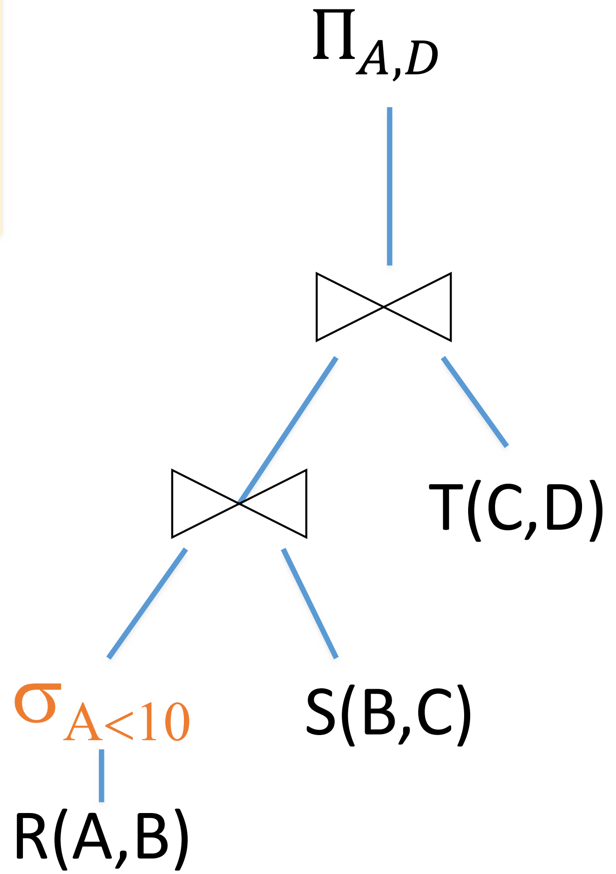
Heuristically, we want selection and projection to occur early to have fewer or smaller "intermediate" tuples

```
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;
```

Push down selection on A so it occurs earlier

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



An example: SQL to RA to Optimized RA

Heuristically, we want selection and projection to occur early to have fewer or smaller "intermediate" tuples

```
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;
```

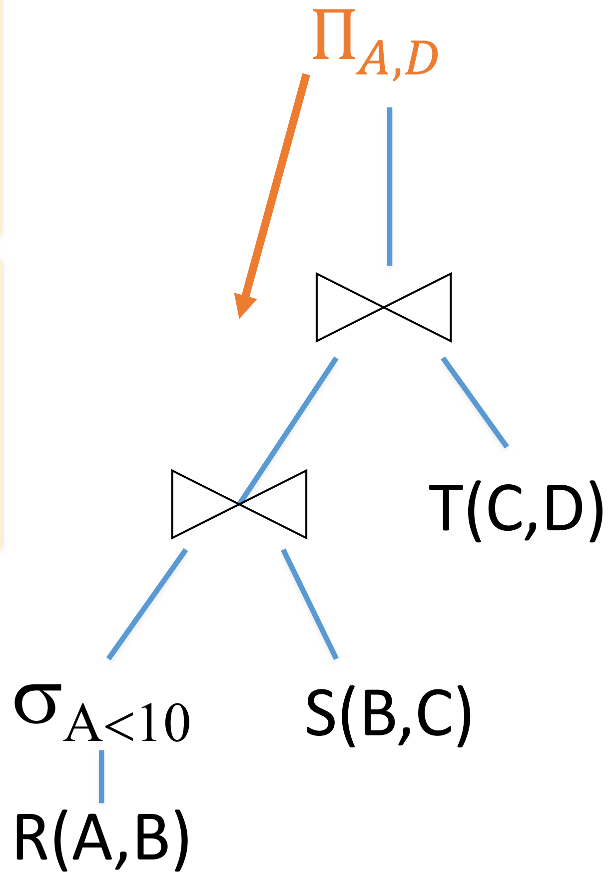
Push down selection on A so it occurs earlier

Push down projection so it occurs earlier

$\Pi_{-B,C}$

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

$\Pi_{-B,C}$



An example: SQL to RA to Optimized RA

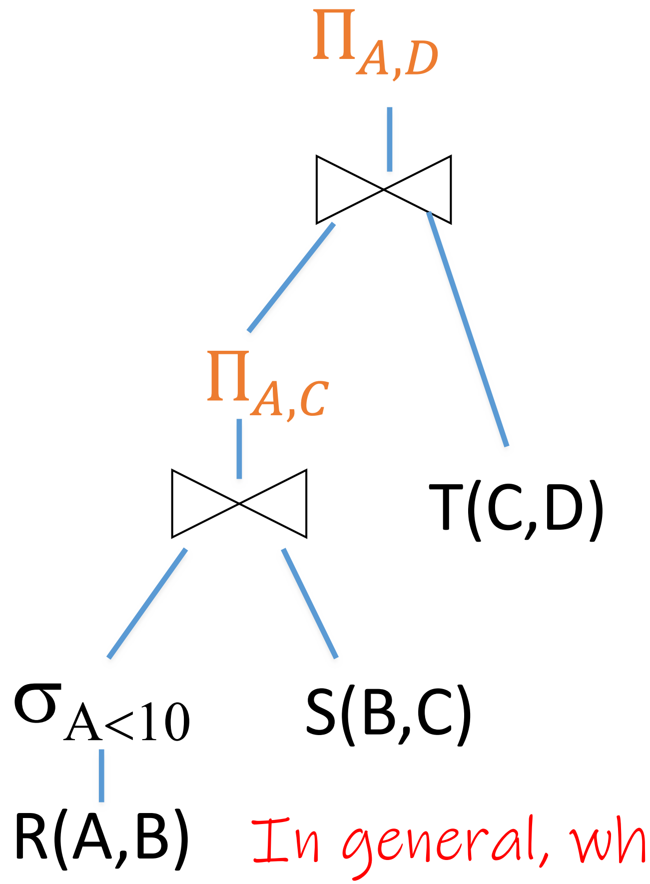
```
R(A,B) S(B,C) T(C,D)
```

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

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

Π_{-C} Π_{-B}

We now eliminate B earlier



Π_{-C}

Π_{-B}

In general, when is an attribute not needed?

Queries and the connection to logic and algebra

- Why logic?
  - A crash course on FOL
- Relational Calculus
  - Syntax and Semantics
  - Domain Independence and Safety
- Relational Algebra
  - Operators
  - Independence
  - Power of algebra: optimizations
- Equivalence RC and RA

An example

Person(id, gender, country)
Spouse(person1, person2)

In RC:

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

In RA:

?

An example

Person(id, gender, country)
Spouse(person1, person2)

In RC:

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

$\Updownarrow$

In RA:

$$\pi_{\mathrm{id}}\mathrm{Person} - \rho_{\mathrm{person1/id}}\pi_{\mathrm{person1}}\mathrm{Spouse}$$

Equivalence Between RA and D.I. 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 E
   there is a domain-independent RC query Q such that $\mathrm{Q} \equiv \mathrm{E}$
2. For every* domain-independent RC query Q
   there is an RA expression E such that $\mathrm{Q} \equiv \mathrm{E}$

* Technicality: we consider only queries that output values from the database (otherwise we need to extend RA accordingly…)

About the proof

The proof has two directions

1. Translate a given RA expression into an equivalent RC query

   Part 1 is fairly easy: induction on the size of the RA expression

2. Translate a given RC query into an equivalent RA expression

   Part 2 is more involved

RA → RC: 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
- Slight technicality: need to maintain a mapping b/w attribute names and variables

Here, ϕ_i is the formula constructed for E_i

RA expression	RC formula ϕ
R (n columns)	$R(X_1,\ldots,X_n)$
$E_1 \times E_2$	
$E_1 - E_2$	
$E_1 \cup E_2$	
$\pi_{a_1,\ldots,a_k}(E_1)$	
$\sigma_c(E_1)$	

RA → RC: 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
- Slight technicality: need to maintain a mapping b/w attribute names and variables

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

RA → RC: 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
- Slight technicality: need to maintain a mapping b/w attribute names and variables

Here, ϕ_i is the formula constructed for E_i

RA expression	RC formula ϕ
R (n columns)	$R(X_1,\ldots,X_n)$
$E_1 \times E_2$	$\phi_1 \wedge \phi_2$ disjoint variables (rename)
$E_1 - E_2$	$\phi_1 \wedge \neg\phi_2$ use identical variables (rename)
$E_1 \cup E_2$	$\phi_1 \vee \phi_2$ use identical variables (rename)
$\pi_{a_1,\ldots,a_k}(E_1)$	$\exists X_1 \ldots \exists Xm.\, \phi_1$ where $X_1, \ldots, Xm$ are the variables not among $a_1,\ldots a_k$
$\sigma_c(E_1)$	$\phi_1 \wedge c$

RA → RC: Example R÷S

R(A,B)   S(B)

RA	RC	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 → RC: Example R÷S

$R(A,B) \quad S(B)$

RA	RC	Mapping
R	$R(x, y)$	$x{:}A,\ y{:}B$
$\pi_A(R)$	$\exists y.\, R(x, y)$	$x{:}A$
S	$S(z)$	$z{:}B$
$\pi_A(R) \times S$		
$(\pi_A(R) \times S) - R$		
$\pi_A\big((\pi_A(R) \times S) - R\big)$		
$\pi_A(R) -$ $\pi_A\big((\pi_A(R) \times S) - R\big)$		

RA → RC: Example R÷S

R(A,B)   S(B)

RA	RC	Mapping
R	$R(x, y)$	x:A, y:B
$\pi_A(R)$	$\exists y. R(x, y)$	x:A
S	$S(z)$	z:B
$\pi_A(R) \times S$	$\exists y. R(x, y) \land S(z)$ z needs to be different from y	x:A, z:B
$(\pi_A(R) \times S) - R$	$\left(\exists y. R(x, y) \land S(z)\right) \land \neg R(x, z)$	x:A, z:B
$\pi_A\left((\pi_A(R) \times S) - R\right)$	$\exists z\left[\left(\exists y. R(x, y) \land S(z)\right) \land \neg R(x, z)\right]$	x:A
$\pi_A(R) -$ $\pi_A\left((\pi_A(R) \times S) - R\right)$	$\exists y. R(x, y) \land$ x's need to be same variable $\neg\exists z\left[\left(\exists y. R(x, y) \land S(z)\right) \land \neg R(x, z)\right]$	x:A

y's don't need to be same variable

"Clear" variables (not a standard term)

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

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

? which variables are free or bound?

"Clear" variables (not a standard term)

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

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

bound free bound

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

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

? how to make it clear

"Clear" variables (not a standard term)

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

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

bound   free   bound

recall operator precedence: $\exists$ before $\land$

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

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

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

now clear

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

but as query not domain-independent

Repeated variable names

When evaluating a sentence with multiple quantifiers, don't fall into the trap of thinking that distinct variables range over distinct objects.

? Which of the following formulas imply each other?

$\forall x. \forall y.\ P(x,y)$ $\qquad$ $\forall x.\ P(x,x)$

$\exists x. \exists y.\ P(x,y)$ $\qquad$ $\exists x.\ P(x,x)$

Repeated variable names

When evaluating a sentence with multiple quantifiers, don't fall into the trap of thinking that distinct variables range over distinct objects.

Recall that distinct variables do not need to range over distinct objects.

$$\forall x.\forall y.\ P(x,y) \quad \Leftarrow \quad \forall x.\ P(x,x)$$

P		
	1	1
	2	2

$$\exists x.\exists y.\ P(x,y) \quad \Rightarrow \quad \exists x.\ P(x,x)$$

P	A	B
	1	2

Repeated variable names

When evaluating a sentence with multiple quantifiers, don't fall into the trap of thinking that distinct variables range over distinct objects.

Recall that distinct variables do not need to range over distinct objects.

$$\forall x. \forall y.\ P(x,y) \quad \Leftarrow \quad \forall x.\ P(x,x)$$

$$\Downarrow \qquad \text{Only if domain is not empty! } \mathrm{Dom} \neq \emptyset \qquad \Downarrow$$

$$\exists x. \exists y.\ P(x,y) \quad \Rightarrow \quad \exists x.\ P(x,x)$$

RC → RA: Intution

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 relational calculus formulas to obtain a translation of relational calculus under the active domain interpretation to relational algebra.

RC → RA: Intuition

$R(A,B)$

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

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

RC → RA: Intuition

R(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: $\forall y.\, R(x, y) \equiv \neg \exists y.\, \neg R(x, y)$

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

RC formula ϕ	RA expression for ϕ^{adom}
$\neg R(x, y)$	
$\exists y.\, \neg R(x, y)$	
$\neg \exists y.\, \neg R(x, y)$	

RC → RA: Intuition

$R(A,B)$

- 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: $\forall y.\,R(x,y) \equiv \neg\exists y.\,\neg R(x,y)$

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

RC formula ϕ	RA expression for ϕ^{adom}
$\neg R(x,y)$	$(\,ADom(D) \times ADom(D)\,) - R$
$\exists y.\,\neg R(x,y)$	$\pi_A[(\,ADom(D) \times ADom(D)\,) - R]$
$\neg\exists y.\,\neg R(x,y)$	$ADom(D) - \pi_A[(\,ADom(D) \times ADom(D)\,) - R]$

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]