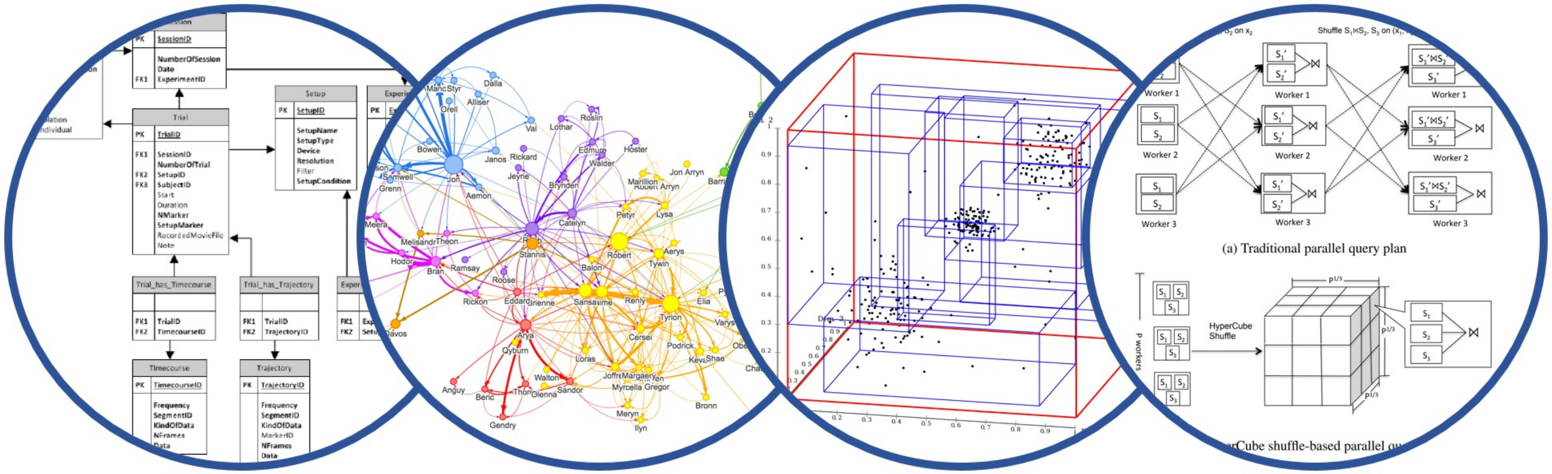


Introduction to Data Management

Relational Algebra

Paul G. Allen School of Computer Science and Engineering
University of Washington, Seattle

Announcements

- HW3 due next Friday

- Midterm on Friday, 4/26 in class
  - Closed books, no cheat sheet (you won't need it)
  - Some practice midterms on the course website

Quantifiers

Recap: Predicates on Subqueries

- EXISTS / NOT EXISTS
- IN / NOT IN
- ANY / ALL

The are “equivalent” meaning that a query that you can write using one, you can also write using the others

Quantifiers

Find people who drive only cars made after 2017

```
SELECT P.*
FROM Payroll P
WHERE 2017 <
 ALL(SELECT R.Year
     FROM Regist R
     WHERE P.UserID = R.UserID);
```

Quantifiers

Find people who drive only cars made after 2017

```
SELECT P.*
FROM Payroll P
WHERE 2017 <
 ALL(SELECT R.Year
     FROM Regist R
     WHERE P.UserID = R.UserID);
```

```
SELECT P.*
FROM Payroll P
WHERE NOT EXISTS
  (SELECT *
   FROM Regist R
   WHERE P.UserID = R.UserID
     and R.Year <= 2017);
```

Quantifiers

Find people who drive only cars made after 2017

```
SELECT P.*
FROM Payroll P
WHERE 2017 <
 ALL(SELECT R.Year
     FROM Regist R
     WHERE P.UserID = R.UserID);
```

```
SELECT P.*
FROM Payroll P
WHERE NOT EXISTS
  (SELECT *
   FROM Regist R
   WHERE P.UserID = R.UserID
     and R.Year <= 2017);
```

```
SELECT P.*
FROM Payroll P
WHERE P.UserID NOT IN
      (SELECT R.UserID
       FROM Regist R
       WHERE R.Year <= 2017);
```

Quantifiers

Find people who drive only cars made after 2017

```
SELECT P.*
FROM Payroll P
WHERE 2017 <
 ALL(SELECT R.Year
     FROM Regist R
     WHERE P.UserID = R.UserID);
```

All these compute the same thing

```
SELECT P.*
FROM Payroll P
WHERE NOT EXISTS
  (SELECT *
   FROM Regist R
   WHERE P.UserID = R.UserID
     and R.Year <= 2017);
```

```
SELECT P.*
FROM Payroll P
WHERE P.UserID NOT IN
      (SELECT R.UserID
       FROM Regist R
       WHERE R.Year <= 2017);
```

Discussion

- SQL can express naturally queries that represent existential quantifiers
- To write a query that uses a universal quantifier, use DeMorgan's laws (next few slides)

Quantifiers

There are two types of quantifiers:

- **Exists** ($\exists x, \ldots$) there is at least 1 that satisfies predicate
- **Forall**: ($\forall x, \ldots$) all elements satisfy the predicate

Quantifiers

There are two types of quantifiers:

- **Exists** $(\exists x, \ldots)$ there is at least 1 that satisfies predicate
- **Forall**: $(\forall x, \ldots)$ all elements satisfy the predicate

SQL makes it easy to write **exists**

Quantifiers

There are two types of quantifiers:

- **Exists** ($\exists x, \ldots$) there is at least 1 that satisfies predicate
- **Forall**: ($\forall x, \ldots$) all elements satisfy the predicate

SQL makes it easy to write **exists**

To write **forall**, use double negation

predicate holds **forall** elements
if and only if
not (**exists** element where not(predicate) holds)

How to Write FORALL in SQL

Find person **P** drives **only** cars made after 2017

How to Write FORALL in SQL

Find person **P** drives **only** cars made after 2017

Negate: find the other persons

Find person **P** drives **some** car made on or before 2017

How to Write FORALL in SQL

Find person **P** drives **only** cars made after 2017

Negate: find the other persons

Find person **P** drives **some** car made on or before 2017

```
SELECT P.*
FROM Payroll P
WHERE EXISTS
  (SELECT R.Year
   FROM Regist R
   WHERE P.UserID = R.UserID
     and R.Year <= 2017);
```

How to Write FORALL in SQL

Find person **P** drives **only** cars made after 2017

Negate: find the other persons
Find person **P** drives **some** car made on or before 2017

```
SELECT P.*
FROM Payroll P
WHERE EXISTS
  (SELECT R.Year
   FROM Regist R
   WHERE P.UserID = R.UserID
     and R.Year <= 2017);
```

Negate again:
find the other other persons

```
SELECT P.*
FROM Payroll P
WHERE NOT EXISTS
  (SELECT R.Year
   FROM Regist R
   WHERE P.UserID = R.UserID
     and R.Year <= 2017);
```

How to Write FORALL in SQL

Find person **P** drives **only** cars made after 2017

Universal quantifier

Negate: find the other persons

Existential quantifier

Find person **P** drives **some** car made on or before 2017

```
SELECT P.*
FROM Payroll P
WHERE EXISTS
  (SELECT R.Year
   FROM Regist R
   WHERE P.UserID = R.UserID
     and R.Year <= 2017);
```

Negate again:
find the other other persons

```
SELECT P.*
FROM Payroll P
WHERE NOT EXISTS
  (SELECT R.Year
   FROM Regist R
   WHERE P.UserID = R.UserID
     and R.Year <= 2017);
```

Brief Review of Logic

- Implication: A→B   is same as: not(A) or B

Brief Review of Logic

- Implication: $A \rightarrow B$ is same as: not(A) or B

- DeMorgan's Laws:

$$\text{not}(A \text{ and } B) = \text{not}(A) \text{ or } \text{not}(B)$$
$$\text{not}(A \text{ or } B) = \text{not}(A) \text{ and } \text{not}(B)$$

Brief Review of Logic

- Implication: A→B is same as: not(A) or B

- DeMorgan's Laws:

$$\text{not}(\text{A and B}) = \text{not}(\text{A}) \text{ or not}(\text{B})$$
$$\text{not}(\text{A or B}) = \text{not}(\text{A}) \text{ and not}(\text{B})$$

$$\text{not}\big(\exists \text{x}, \text{P}(\text{x})\big) = \forall \text{x}, \text{not}\big(\text{P}(\text{x})\big)$$
$$\text{not}\big(\forall \text{x}, \text{P}(\text{x})\big) = \exists \text{x}, \text{not}\big(\text{P}(\text{x})\big)$$

Brief Review of Logic

- Implication: A→B is same as: not(A) or B

- DeMorgan's Laws:

$$\text{not}(\text{A and B}) = \text{not}(\text{A}) \text{ or not}(\text{B})$$
$$\text{not}(\text{A or B}) = \text{not}(\text{A}) \text{ and not}(\text{B})$$

$$\text{not}\big(\exists x, P(x)\big) = \forall x, \text{not}\big(P(x)\big)$$
$$\text{not}\big(\forall x, P(x)\big) = \exists x, \text{not}\big(P(x)\big)$$

- Consequences

$$A \rightarrow B = \text{not}(\text{A and not}(B))$$

Brief Review of Logic

- Implication: A→B is same as: not(A) or B

- DeMorgan's Laws:

$$\text{not}(\text{A and B}) = \text{not}(\text{A}) \text{ or not}(\text{B})$$
$$\text{not}(\text{A or B}) = \text{not}(\text{A}) \text{ and not}(\text{B})$$

$$\text{not}\big(\exists \text{x}, \text{P}(\text{x})\big) = \forall \text{x}, \text{not}\big(\text{P}(\text{x})\big)$$
$$\text{not}\big(\forall \text{x}, \text{P}(\text{x})\big) = \exists \text{x}, \text{not}\big(\text{P}(\text{x})\big)$$

- Consequences

$$\text{A} \rightarrow \text{B} = \text{not}(\text{A and not}(\text{B}))$$

$$\forall \text{x}, \big(\text{A}(\text{x}) \rightarrow \text{B}(\text{x})\big) = \text{not}(\exists \text{x}(\text{A}(\text{x}) \wedge \text{not}(\text{B}(\text{x}))))$$

Brief Review of First Order Logic

Query: persons **P** that drive only cars made after 2017:

$$\forall \mathrm{R} \in \mathrm{Regist}, (\mathbf{P}.\mathrm{UserID} = \mathrm{R}.\mathrm{UserID}) \Rightarrow (\mathrm{R}.\mathrm{Year} > 2017)$$

Negation: persons **P** that drive some car made on/before 2017:

$$\exists \mathrm{R} \in \mathrm{Regist}, (\mathbf{P}.\mathrm{UserID} = \mathrm{R}.\mathrm{UserID}) \text{ and } (\mathrm{R}.\mathrm{Year} \leq 2017)$$

Discussion

Writing universally quantified queries in SQL requires creativity

- Try using DeMorgan’s laws
- Try using ALL
- Try using aggregates, checking count=0

Relational Algebra

Motivation

- SQL is a declarative language:
  we say what, we don’t say how

- The query optimizer needs to convert the query into some language that can be excecuted

- That language is Relational Algebra

The Five Basic Relational Operators

1. Selection $\sigma_{\text{condition}}(S)$
2. Projection $\Pi_{\text{attrs}}(S)$
3. Join $\mathrm{R} \bowtie_{\theta} \mathrm{S} = \sigma_{\theta}(\mathrm{R} \times \mathrm{S})$
4. Union $\cup$
5. Set difference $-$

- Rename ρ

Let's discuss them one by one

1. Selection

$$\sigma_{\text{condition}}(\mathrm{T})$$

Returns those tuples in T that satisfy the condition:

```
SELECT *
FROM T
WHERE condition;
```

1. Selection

$$\sigma_{\text{condition}}(\text{T})$$

Returns those tuples in T that satisfy the condition:

```
SELECT *
FROM T
WHERE condition;
```

$$\sigma_{\text{salary} \geq 55000}(\text{Payroll}) =$$

Payroll

UserID	Name	Job	Salary
123	Jack	TA	50000
345	Allison	TA	60000
567	Magda	Prof	90000
789	Dan	Prof	100000

1. Selection

$$\sigma_{\text{condition}}(\text{T})$$

Returns those tuples in T that satisfy the condition:

```
SELECT *
FROM T
WHERE condition;
```

UserID	Name	Job	Salary
345	Allison	TA	60000
567	Magda	Prof	90000
789	Dan	Prof	100000

$$\sigma_{\text{salary} \geq 55000}(\text{Payroll}) =$$

Payroll

UserID	Name	Job	Salary
123	Jack	TA	50000
345	Allison	TA	60000
567	Magda	Prof	90000
789	Dan	Prof	100000

1. Selection

$$\sigma_{\text{condition}}(\text{T})$$

Returns those tuples in T that satisfy the condition:

```
SELECT *
FROM T
WHERE condition;
```

$$\sigma_{\text{salary} \geq 55000 \text{ and Job}='\text{TA}'}(\text{Payroll}) =$$

Payroll

UserID	Name	Job	Salary
123	Jack	TA	50000
345	Allison	TA	60000
567	Magda	Prof	90000
789	Dan	Prof	100000

1. Selection

UserID	Name	Job	Salary
345	Allison	TA	60000

$$\sigma_{\text{condition}}(\text{T})$$

Returns those tuples in T that satisfy the condition:

```
SELECT *
FROM T
WHERE condition;
```

$$\sigma_{\text{salary} \geq 55000 \text{ and Job}='\text{TA}'}(\text{Payroll}) =$$

Payroll

UserID	Name	Job	Salary
123	Jack	TA	50000
345	Allison	TA	60000
567	Magda	Prof	90000
789	Dan	Prof	100000

2. Projection

$$\Pi_{\text{attrs}}(\text{T})$$

Returns all tuples in T keeping only the attributes in the subscript:

```
SELECT attrs
FROM T;
```

2. Projection

$$\Pi_{attrs}(T)$$

Returns all tuples in T keeping only the attributes in the subscript:

```
SELECT attrs
FROM T;
```

$$\Pi_{Name,Salary}(Payroll) =$$

Payroll

UserID	Name	Job	Salary
123	Jack	TA	50000
345	Allison	TA	60000
567	Magda	Prof	90000
789	Dan	Prof	100000

2. Projection

$$\Pi_{attrs}(T)$$

Returns all tuples in T keeping only the attributes in the subscript:

```
SELECT attrs
FROM T;
```

$$\Pi_{Name,Salary}(Payroll) =$$

Name	Salary
Jack	50000
Allison	60000
Magda	90000
Dan	100000

Payroll

UserID	Name	Job	Salary
123	Jack	TA	50000
345	Allison	TA	60000
567	Magda	Prof	90000
789	Dan	Prof	100000

2. Projection

$$\Pi_{\text{attrs}}(\text{T})$$

Returns all tuples in T keeping only the attributes in the subscript:

```
SELECT attrs
FROM T;
```

$$\Pi_{\text{Job}}(\text{Payroll}) =$$

Payroll

UserID	Name	Job	Salary
123	Jack	TA	50000
345	Allison	TA	60000
567	Magda	Prof	90000
789	Dan	Prof	100000

2. Projection

$$\Pi_{attrs}(T)$$

Returns all tuples in T keeping only the attributes in the subscript:

```
SELECT attrs
FROM T;
```

$$\Pi_{Job}(Payroll) =$$

Job
TA
TA
Prof
Prof

Payroll

UserID	Name	Job	Salary
123	Jack	TA	50000
345	Allison	TA	60000
567	Magda	Prof	90000
789	Dan	Prof	100000

2. Projection

$$\Pi_{\text{attrs}}(\text{T})$$

Returns all tuples in T keeping only the attributes in the subscript:

```
SELECT attrs
FROM T;
```

Job
TA
TA
Prof
Prof

RA can be defined using bag semantics or set semantics. We always need to say which one we mean.

Job
TA
Prof

$$\Pi_{\text{Job}}(\text{Payroll}) =$$

Payroll

UserID	Name	Job	Salary
123	Jack	TA	50000
345	Allison	TA	60000
567	Magda	Prof	90000
789	Dan	Prof	100000

3. Join

$$S \bowtie_{\theta} T$$

Join S and T using condition θ

```
SELECT *
FROM S,T
WHERE θ;
```

3. Join

$$S \bowtie_\theta T$$

Join S and T using condition θ

```
SELECT *
FROM S,T
WHERE θ;
```

$$\text{Payroll} \bowtie_{\text{UserID}=\text{UserID}} \text{Regist} =$$

Payroll

UserID	Name	Job	Salary
123	Jack	TA	50000
345	Allison	TA	60000
567	Magda	Prof	90000
789	Dan	Prof	100000

Regist

UserID	Car
123	Charger
567	Civic
567	Pinto

3. Join

$$S \bowtie_{\theta} T$$

UserID	Name	Job	Salary	UserID	Car
123	Jack	TA	50000	123	Charger
567	Magda	Prof	90000	567	Civic
567	Magda	Prof	90000	567	Pinto

Join S and T using condition θ

$$\text{Payroll} \bowtie_{\text{UserID=UserID}} \text{Regist} =$$

```
SELECT *
FROM S,T
WHERE θ;
```

Payroll

UserID	Name	Job	Salary
123	Jack	TA	50000
345	Allison	TA	60000
567	Magda	Prof	90000
789	Dan	Prof	100000

Regist

UserID	Car
123	Charger
567	Civic
567	Pinto

Many Variants of Join

- Eq-join: $\text{Payroll} \bowtie_{\text{UserID}=\text{UserID}} \text{Regist}$
- Theta-join: $\text{Payroll} \bowtie_{\text{UserID}<\text{UserID}} \text{Regist}$
- Cartesian product: $\text{Payroll} \times \text{Regist}$
- Natural Join: $\text{Payroll} \bowtie \text{Regist}$

Many Variants of Join

Only =

- Eq-join: $\text{Payroll} \bowtie_{\text{UserID}=\text{UserID}} \text{Regist}$

Any condition

- Theta-join: $\text{Payroll} \bowtie_{\text{UserID}<\text{UserID}} \text{Regist}$
- Cartesian product: $\text{Payroll} \times \text{Regist}$
- Natural Join: $\text{Payroll} \bowtie \text{Regist}$

Many Variants of Join

- Eq-join: $\text{Payroll} \bowtie_{\text{UserID}=\text{UserID}} \text{Regist}$ — Only =
- Theta-join: $\text{Payroll} \bowtie_{\text{UserID}<\text{UserID}} \text{Regist}$ — Any condition
- Cartesian product: $\text{Payroll} \times \text{Regist}$
- Natural Join: $\text{Payroll} \bowtie \text{Regist}$

Next

Cartesian Product / Cross Product

$$S \times T$$

Cross product of S and T

```
SELECT *
FROM S,T
```

Cartesian Product / Cross Product

$$S \times T$$

Cross product of S and T

```
SELECT *
FROM S,T
```

$$\text{Payroll} \times \text{Regist} =$$

Payroll

UserID	Name	Job	Salary
123	Jack	TA	50000
345	Allison	TA	60000
567	Magda	Prof	90000
789	Dan	Prof	100000

Regist

UserID	Car
123	Charger
567	Civic
567	Pinto

Cartesian Product / Cross Product

$S \times T$

Cross product of S and T

```
SELECT *
FROM S,T
```

12 tuples

UserID	Name	Job	Salary	UserID	Car
123	Jack	TA	50000	123	Charger
123	Jack	TA	50000	567	Civic
			. . .		
789	Dan	Prof	100000	567	Pinto

$\text{Payroll} \times \text{Regist} =$

Payroll

UserID	Name	Job	Salary
123	Jack	TA	50000
345	Allison	TA	60000
567	Magda	Prof	90000
789	Dan	Prof	100000

Regist

UserID	Car
123	Charger
567	Civic
567	Pinto

Cartesian Product / Cross Product

$$S \times T$$

Cross product of S and T

```
SELECT *
FROM S,T
```

Join = cartesian product + selection

$$R \bowtie_{\theta} S = \sigma_{\theta}(R \times S)$$

Natural Join

$S \bowtie T$

Join S, T on
common attributes,
retain only one copy
of those attributes

Natural Join

$S \bowtie T$

Join S, T on common attributes, retain only one copy of those attributes

$\text{Payroll} \bowtie \text{Regist} =$

Payroll

UserID	Name	Job	Salary
123	Jack	TA	50000
345	Allison	TA	60000
567	Magda	Prof	90000
789	Dan	Prof	100000

Regist

UserID	Car
123	Charger
567	Civic
567	Pinto

Natural Join

$S \bowtie T$

Join S, T on common attributes, retain only one copy of those attributes

UserID	Name	Job	Salary	Car
123	Jack	TA	50000	Charger
567	Magda	Prof	90000	Civic
567	Magda	Prof	90000	Pinto

Only one UserID attr

$\text{Payroll} \bowtie \text{Regist} =$

Payroll

UserID	Name	Job	Salary
123	Jack	TA	50000
345	Allison	TA	60000
567	Magda	Prof	90000
789	Dan	Prof	100000

Regist

UserID	Car
123	Charger
567	Civic
567	Pinto

Natural Join

What do these natural joins output?

- $R(A, B) \bowtie S(B, C)$

- $R(A, B) \bowtie S(C, D)$

- $R(A, B) \bowtie S(A, B)$

Natural Join

What do these natural joins output?

- $R(A, B) \bowtie S(B, C)$

R

A	B
1	10
2	10
2	20

S

B	C
10	8
10	9
20	8
50	7

- $R(A, B) \bowtie S(C, D)$

- $R(A, B) \bowtie S(A, B)$

Natural Join

What do these natural joins output?

- $R(A, B) \bowtie S(B, C)$
  equjoin on attribute B (5 tuples)

R

A	B
1	10
2	10
2	20

S

B	C
10	8
10	9
20	8
50	7

- $R(A, B) \bowtie S(C, D)$

- $R(A, B) \bowtie S(A, B)$

Natural Join

What do these natural joins output?

- $R(A, B) \bowtie S(B, C)$
  equjoin on attribute B (5 tuples)

R

A	B
1	10
2	10
2	20

S

B	C
10	8
10	9
20	8
50	7

- $R(A, B) \bowtie S(C, D)$

R

A	B
1	10
2	10
2	20

S

C	D
8	u
9	v
8	v
7	w

- $R(A, B) \bowtie S(A, B)$

Natural Join

What do these natural joins output?

- $R(A, B) \bowtie S(B, C)$
  equjoin on attribute B (5 tuples)

R

A	B
1	10
2	10
2	20

S

B	C
10	8
10	9
20	8
50	7

- $R(A, B) \bowtie S(C, D)$
  cross product (12 tuples)

R

A	B
1	10
2	10
2	20

S

C	D
8	u
9	v
8	v
7	w

- $R(A, B) \bowtie S(A, B)$

Natural Join

What do these natural joins output?

- $R(A,B) \bowtie S(B,C)$
  equjoin on attribute B (5 tuples)

R

A	B
1	10
2	10
2	20

S

B	C
10	8
10	9
20	8
50	7

- $R(A,B) \bowtie S(C,D)$
  cross product (12 tuples)

R

A	B
1	10
2	10
2	20

S

C	D
8	u
9	v
8	v
7	w

- $R(A,B) \bowtie S(A,B)$

R

A	B
1	10
2	10
2	20

S

A	B
1	10
2	20

Natural Join

What do these natural joins output?

- $R(A,B) \bowtie S(B,C)$
  equjoin on attribute B (5 tuples)

R

A	B
1	10
2	10
2	20

S

B	C
10	8
10	9
20	8
50	7

- $R(A,B) \bowtie S(C,D)$
  cross product (12 tuples)

R

A	B
1	10
2	10
2	20

S

C	D
8	u
9	v
8	v
7	w

- $R(A,B) \bowtie S(A,B)$
  intersection (2 tuples)

R

A	B
1	10
2	10
2	20

S

A	B
1	10
2	20

Even More Joins

- Inner join ⋈
  - Eq-join, theta-join, cross product, natural join
- Outer join
  - Left outer join ⟕
  - Right outer join ⟖
  - Full outer join ⟗
- Semi join ⋉

4. Union

$S \cup T$

The union of S and T

```
S UNION T;
```

SQL

4. Union

$$S \cup T$$

The union of S and T

$$\text{Regist} \cup \text{Bicycle} =$$

```
S UNION T;
```

Regist

UserID	Model
123	Charger
567	Civic
567	Pinto

Bicycle

UserID	Model
345	Schwinn
567	Sirrus

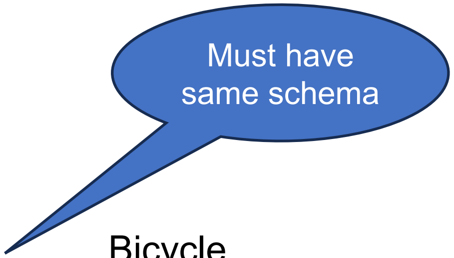
4. Union

$S \cup T$

The union of S and T

```
S UNION T;
```

$\text{Regist} \cup \text{Bicycle} =$



Regist

UserID	Model
123	Charger
567	Civic
567	Pinto

Bicycle

UserID	Model
345	Schwinn
567	Sirrus

4. Union

$S \cup T$

The union of S and T

```
S UNION T;
```

$Regist \cup Bicycle =$

UserID	Model
123	Charger
567	Civic
567	Pinto
345	Schwinn
567	Sirrus

Must have same schema

Regist

UserID	Model
123	Charger
567	Civic
567	Pinto

Bicycle

UserID	Model
345	Schwinn
567	Sirrus

5. Difference

$$S - T$$

The set difference of S and T

```
S EXCEPT T;
```

SQL

5. Difference

$$S - T$$

The set difference of S and T

$$\text{Regist} - \text{Bicycle} =$$

```
S EXCEPT T;
```

Must have same schema

Regist

UserID	Model
123	Charger
567	Civic
567	Pinto

Bicycle

UserID	Model
345	Schwinn
567	Civic

5. Difference

$$S - T$$

The set difference of S and T

```
S EXCEPT T;
```

$$\text{Regist} - \text{Bicycle} =$$

UserID	Model
123	Charger
567	Pinto

Must have same schema

Regist

UserID	Model
123	Charger
567	Civic
567	Pinto

Bicycle

UserID	Model
345	Schwinn
567	Civic

Renaming

$$\rho_{attrs'}(T)$$

Rename attributes

```
SELECT a1 as a1',
       a2 as a2',
       ...
FROM T;
```

Renaming

$$\rho_{attrs'}(T)$$

Rename attributes

```
SELECT a1 as a1',
       a2 as a2',
       ...
FROM T;
```

$$\rho_{\mathrm{UserID,Model}}(\mathrm{Regist}) =$$

Regist

UserID	Car
123	Charger
567	Civic
567	Pinto

Renaming

$$\rho_{attrs'}(T)$$

Rename attributes

```
SELECT a1 as a1',
       a2 as a2',
       ...
FROM T;
```

UserID	Model
123	Charger
567	Civic
567	Pinto

$$\rho_{\text{UserID,Model}}(\text{Regist}) =$$

Regist

UserID	Car
123	Charger
567	Civic
567	Pinto

Renaming

$$\rho_{attrs'}(T)$$

Rename attributes

```
SELECT a1 as a1’,
       a2 as a2’,
       ...
FROM T;
```

UserID	Model
123	Charger
567	Civic
567	Pinto

$$\rho_{\text{UserID,Model}}(\text{Regist}) =$$

Regist

UserID	Car
123	Charger
567	Civic
567	Pinto

Corrected union:

$$\rho_{\text{UserID,Model}}(\text{Regist}) \cup \text{Bicycle}$$

The Five Basic Relational Operators

1. Selection $\sigma_{\text{condition}}(S)$
2. Projection $\Pi_{\text{attrs}}(S)$
3. Join $\mathrm{R} \bowtie_{\theta} \mathrm{S} = \sigma_{\theta}(\mathrm{R} \times \mathrm{S})$
4. Union $\cup$
5. Set difference $-$

- Rename ρ

Which operators are monotone?

The Five Basic Relational Operators

1. Selection $\sigma_{\text{condition}}(S)$
2. Projection $\Pi_{\text{attrs}}(S)$
3. Join $\mathrm{R} \bowtie_{\theta} \mathrm{S} = \sigma_{\theta}(\mathrm{R} \times \mathrm{S})$
4. Union ∪
5. Set difference −

Monotone (1–4)

Non-monotone (5)

- Rename ρ Monotone, but doesn't do anything

Which operators are monotone?

Query Plans

Relational Algebra Plan, or Query Plan

```
SELECT P.Name
FROM Payroll P, Regist R
WHERE P.UserID = R.UserID
   and P.Job = 'TA';
```

Payroll

UserID	Name	Job	Salary
123	Jack	TA	50000
345	Allison	TA	60000
567	Magda	Prof	90000
789	Dan	Prof	100000

Regist

UserID	Car
123	Charger
567	Civic
567	Pinto

Relational Algebra Plan, or Query Plan

```
SELECT P.Name
FROM Payroll P, Regist R
WHERE P.UserID = R.UserID
  and P.Job = 'TA';
```

↓

$$\Pi_{\text{Name}}(\sigma_{\text{Job}='\text{TA}'}(\text{Payroll} \bowtie \text{Regist}))$$

Payroll

UserID	Name	Job	Salary
123	Jack	TA	50000
345	Allison	TA	60000
567	Magda	Prof	90000
789	Dan	Prof	100000

Regist

UserID	Car
123	Charger
567	Civic
567	Pinto

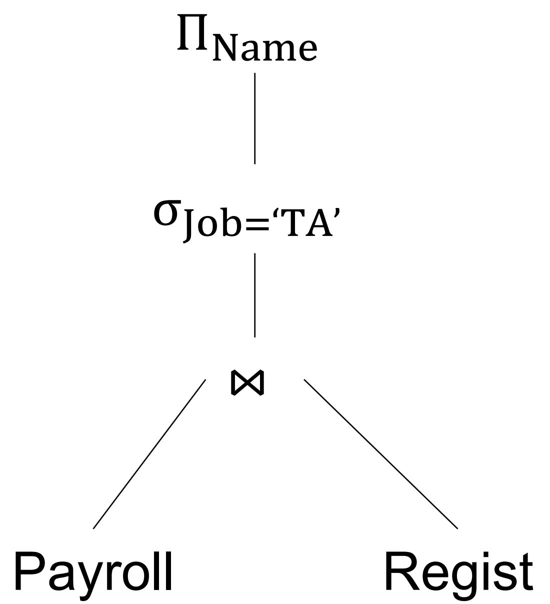
Relational Algebra Plan, or Query Plan

```
SELECT P.Name
FROM Payroll P, Regist R
WHERE P.UserID = R.UserID
  and P.Job = 'TA';
```

$\Pi_{\text{Name}}(\sigma_{\text{Job}='\text{TA}'}(\text{Payroll} \bowtie \text{Regist}))$

We write it as

a query plan



Payroll

UserID	Name	Job	Salary
123	Jack	TA	50000
345	Allison	TA	60000
567	Magda	Prof	90000
789	Dan	Prof	100000

Regist

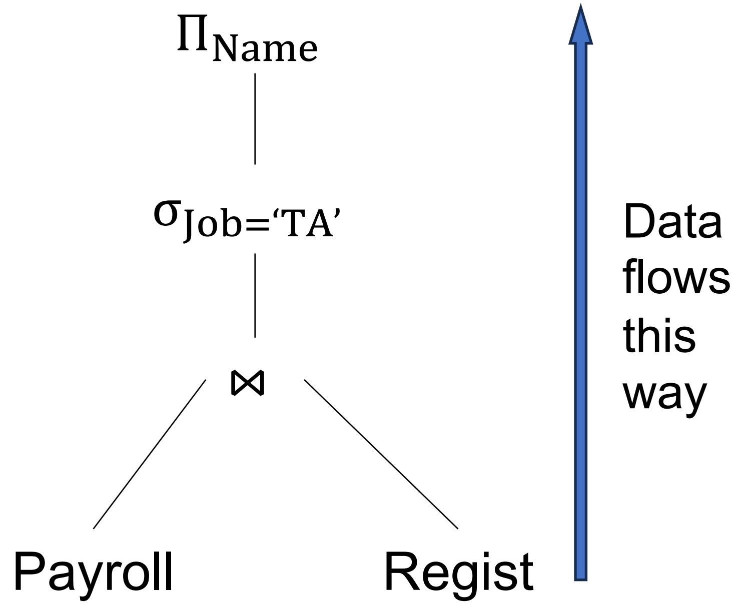
UserID	Car
123	Charger
567	Civic
567	Pinto

Relational Algebra Plan, or Query Plan

```
SELECT P.Name
FROM Payroll P, Regist R
WHERE P.UserID = R.UserID
  and P.Job = 'TA';
```

$$\Pi_{Name}(\sigma_{Job='TA'}(Payroll \bowtie Regist))$$

We write it as a query plan



Payroll

UserID	Name	Job	Salary
123	Jack	TA	50000
345	Allison	TA	60000
567	Magda	Prof	90000
789	Dan	Prof	100000

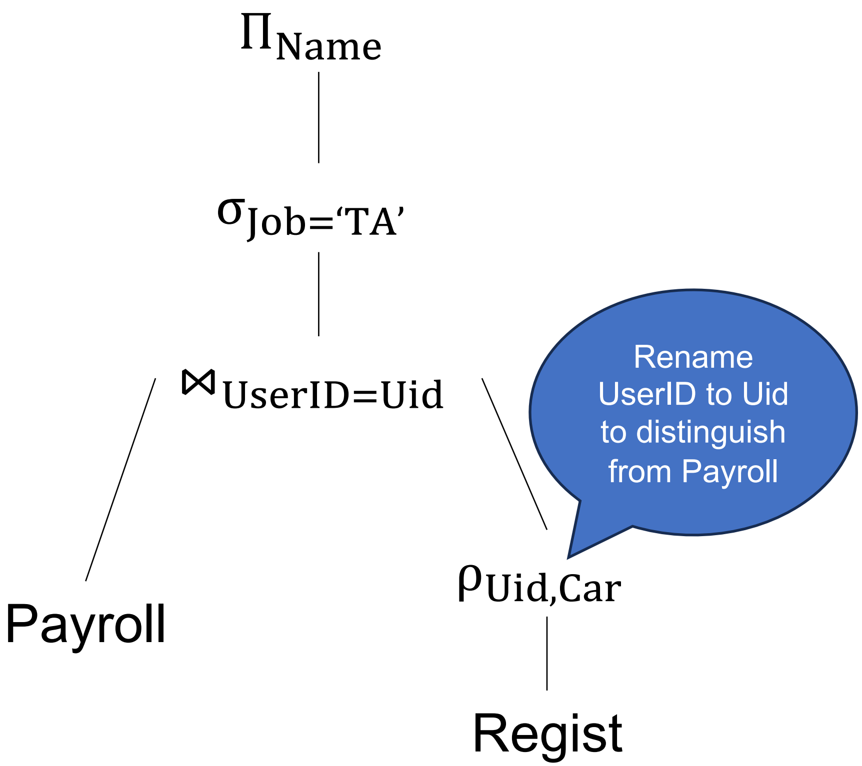
Regist

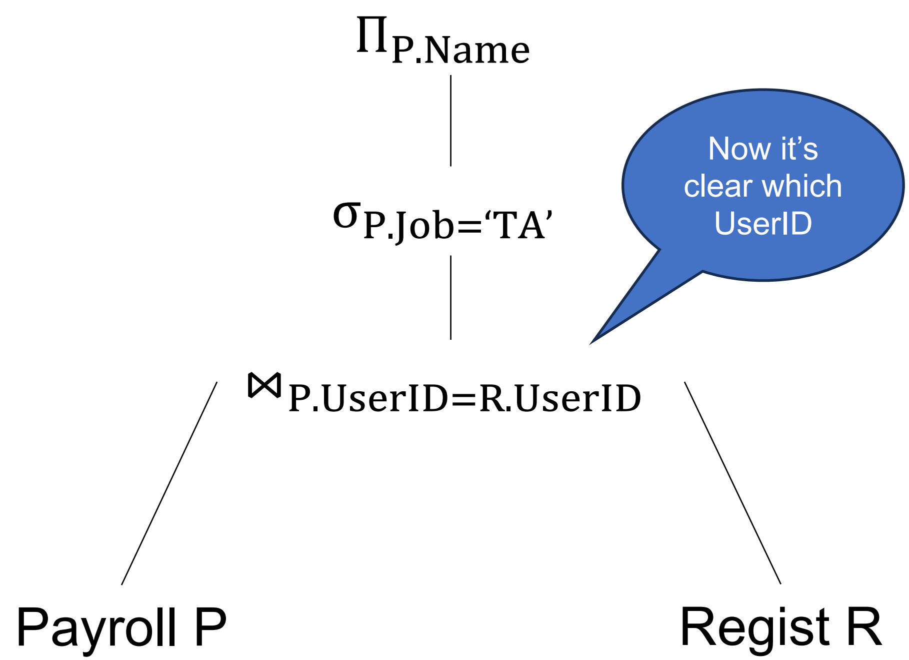
UserID	Car
123	Charger
567	Civic
567	Pinto

Query Plan: Attribute Names

Managing attribute names correctly is tedious

Better: use aliases, much like in SQL





Query Plan: Execution Order

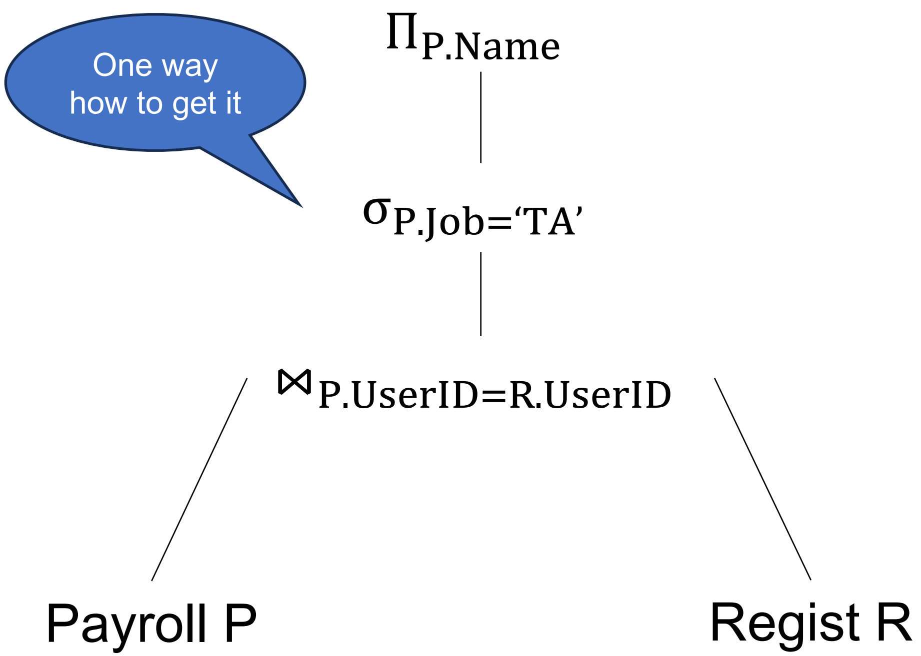
```
SELECT P.Name
FROM Payroll P, Regist R
WHERE P.UserID = R.UserID
  and P.Job = 'TA';
```

We say what we want, not how to get it

Query Plan: Execution Order

```
SELECT P.Name
FROM Payroll P, Regist R
WHERE P.UserID = R.UserID
  and P.Job = 'TA';
```

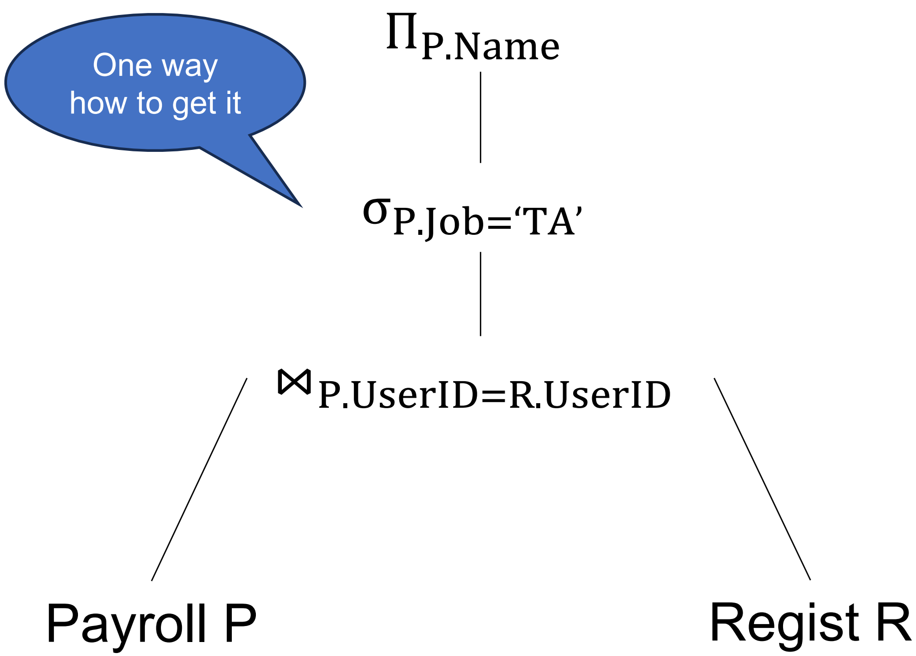
We say what we want, not how to get it

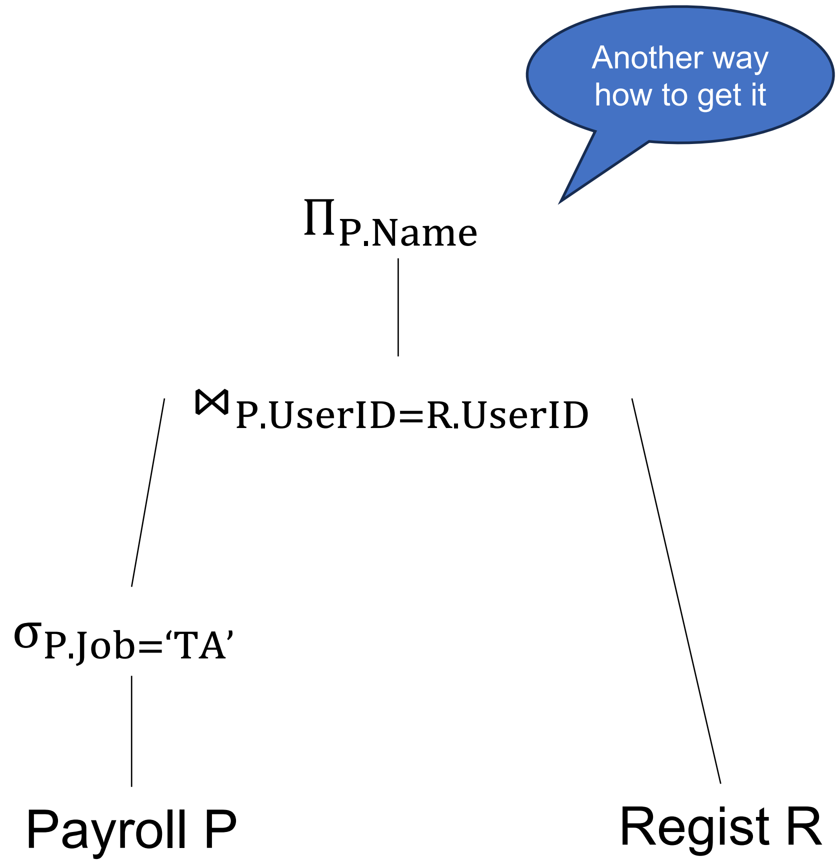


Query Plan: Execution Order

```
SELECT P.Name
FROM Payroll P, Regist R
WHERE P.UserID = R.UserID
  and P.Job = ‘TA’;
```

We say what we want,
not how to get it

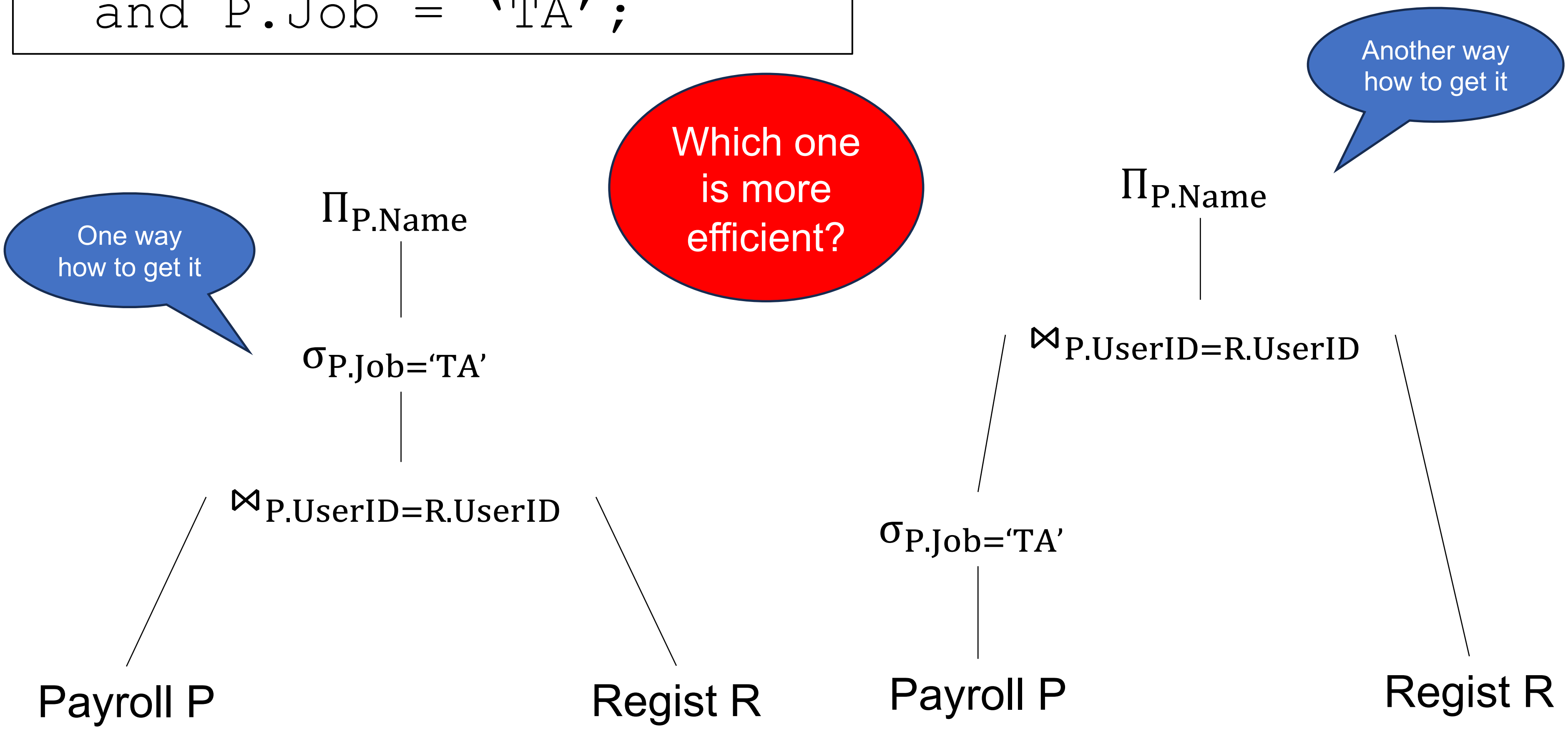




Query Plan: Execution Order

```
SELECT P.Name
FROM Payroll P, Regist R
WHERE P.UserID = R.UserID
  and P.Job = 'TA';
```

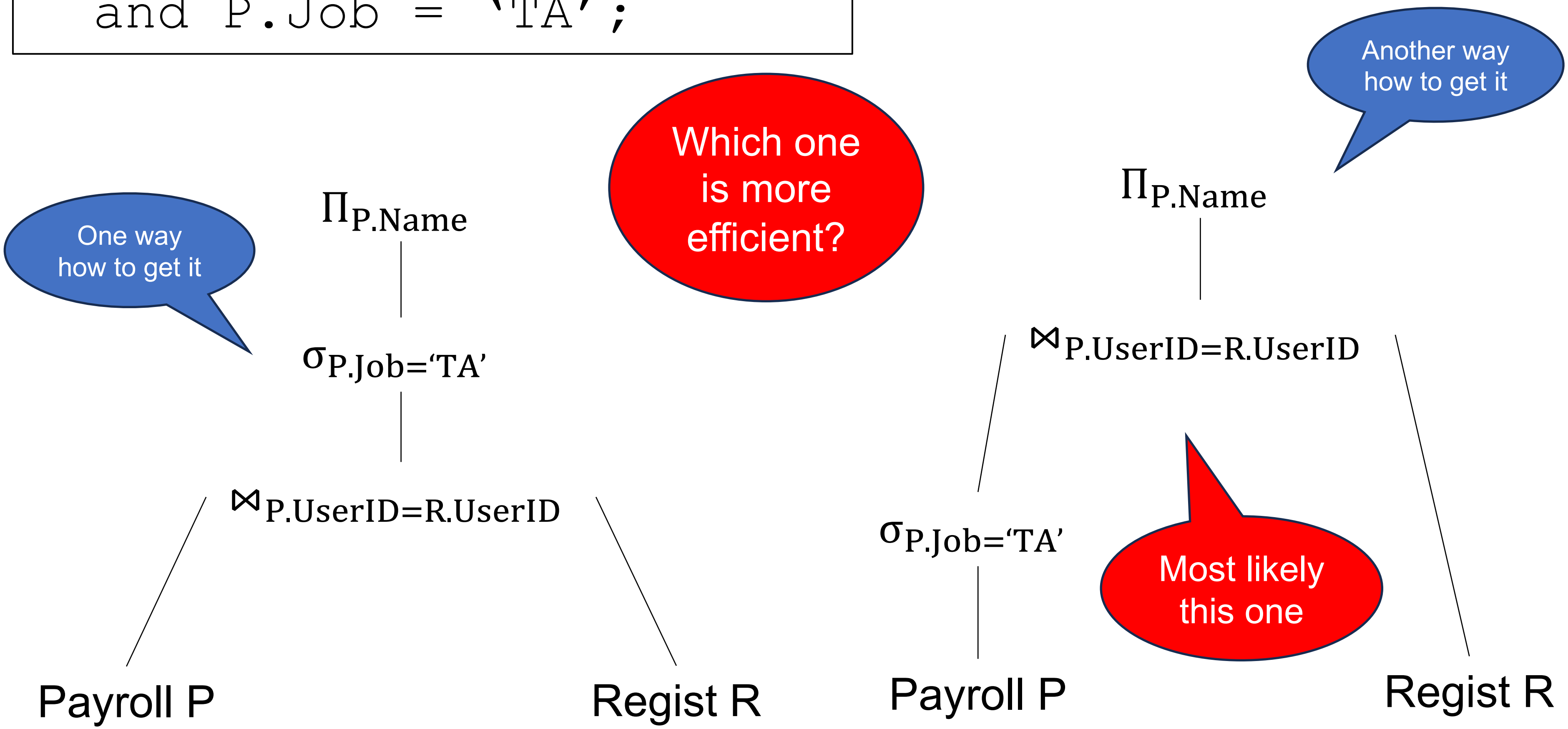
We say what we want,
not how to get it



Query Plan: Execution Order

```
SELECT P.Name
FROM Payroll P, Regist R
WHERE P.UserID = R.UserID
  and P.Job = 'TA';
```

We say what we want,
not how to get it



Discussion

- Database system converts a SQL query to a Relational Algebra Plan

Discussion

- Database system converts a SQL query to a Relational Algebra Plan

- Then it optimizes the plan by exploring equivalent plans, using simple algebraic identities:

$$R \bowtie S = S \bowtie R$$

$$R \bowtie (S \bowtie T) = (R \bowtie S) \bowtie T$$

$$\sigma_\theta(R \bowtie S) = \sigma_\theta(R) \bowtie S$$

… many others*

*over 500 rules in SQL Server

Discussion

- Database system converts a SQL query to a Relational Algebra Plan

- Then it optimizes the plan by exploring equivalent plans, using simple algebraic identities:

  $R \bowtie S = S \bowtie R$

  $R \bowtie (S \bowtie T) = (R \bowtie S) \bowtie T$

  $\sigma_\theta(R \bowtie S) = \sigma_\theta(R) \bowtie S$

  … many others*

- Next lecture: how to convert SQL to RA plan

*over 500 rules in SQL Server