My hands are mostly fine now 🤌 but I'll steal UW slides 1 more time

Motivation

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

- The query optimizer needs to convert the query into some intermediate language that can be both optimized, and executed

- That language is Relational Algebra

The Five Basic Relational Operators

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

Rename ρ

Let’s discuss them one by one

1. Selection

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

Returns those tuples in T that satisfy the condition:

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

1. Selection

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

Returns those tuples in T that satisfy the condition:

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

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

Payroll

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

1. Selection

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

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

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

UserID	Name	Job	Salary
345	Allison	TA	60000

Returns those tuples in T that satisfy the condition:

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

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

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

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

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

```
SELECT attrs
FROM T;
```

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

Payroll

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

2. Projection

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

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

```
SELECT attrs
FROM T;
```

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

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

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

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

```
SELECT attrs
FROM T;
```

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

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;
```

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

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

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 =

Any condition

- 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

- 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

$$Payroll \times Regist =$$

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?

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

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

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

R

A	B
1	10
2	10
2	20

S

B	C
10	8
10	9
20	8
50	7

Natural Join

What do these natural joins output?

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

  eqjoin 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?

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

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

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

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

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

Must have same schema

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;
```

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

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;
```

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

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

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 $R \bowtie_{\theta} S = \sigma_{\theta}(R \times S)$
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 $\mathrm{R} \bowtie_{\theta} S = \sigma_{\theta}(\mathrm{R} \times S)$
4. Union $\cup$
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

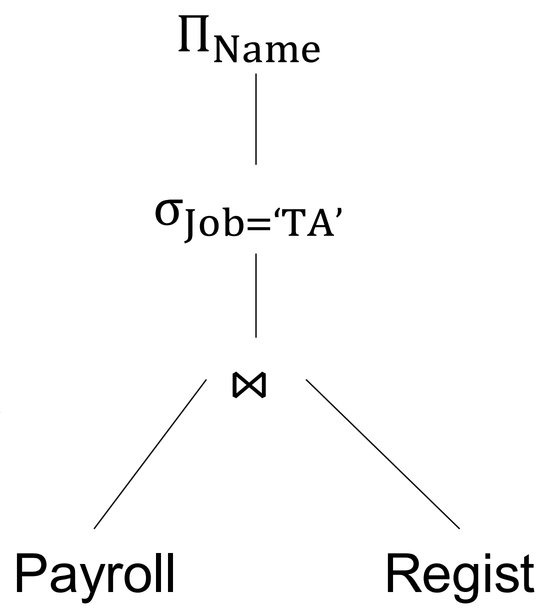
UserID	Car
123	Charger
567	Civic
567	Pinto

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

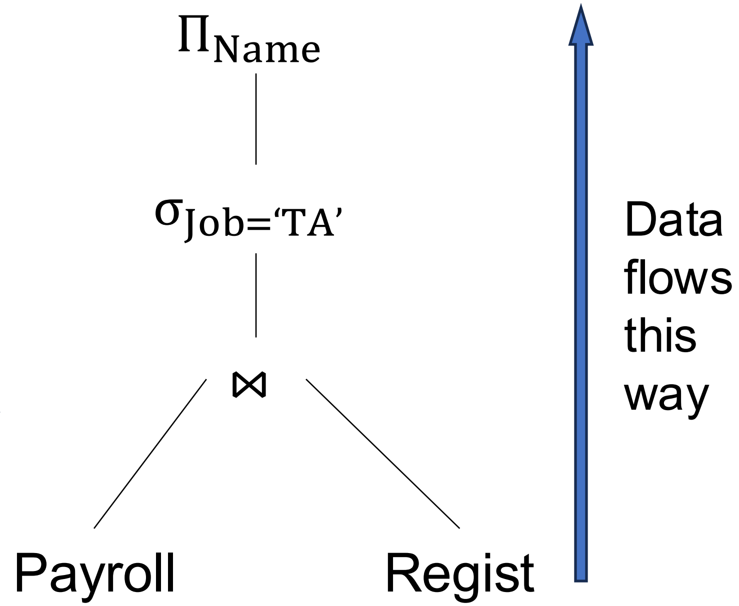
Relational Algebra Plan, or Query Plan

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

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

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

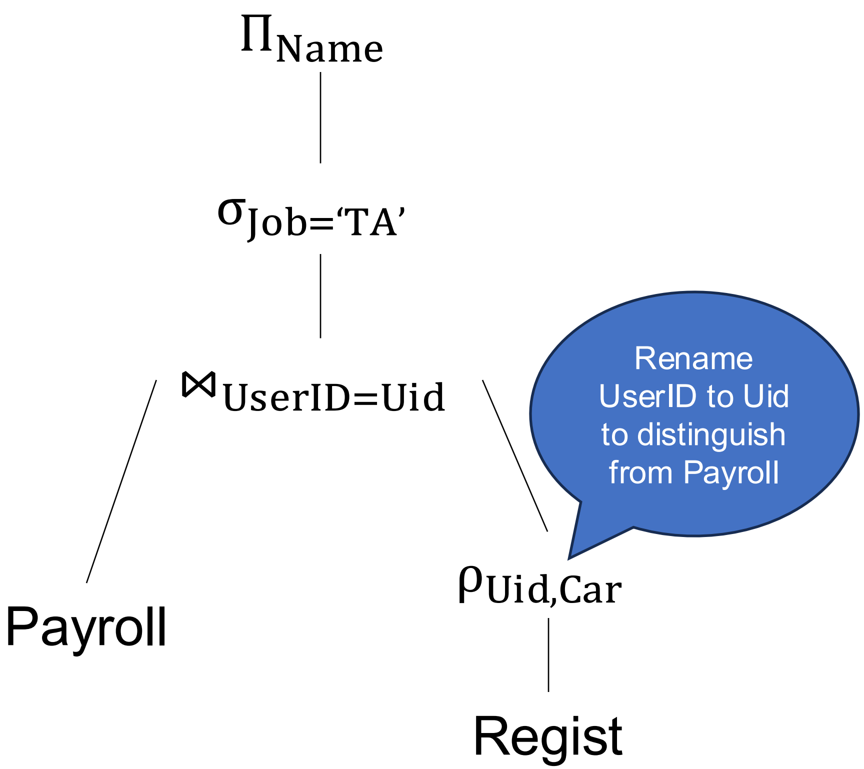
Regist

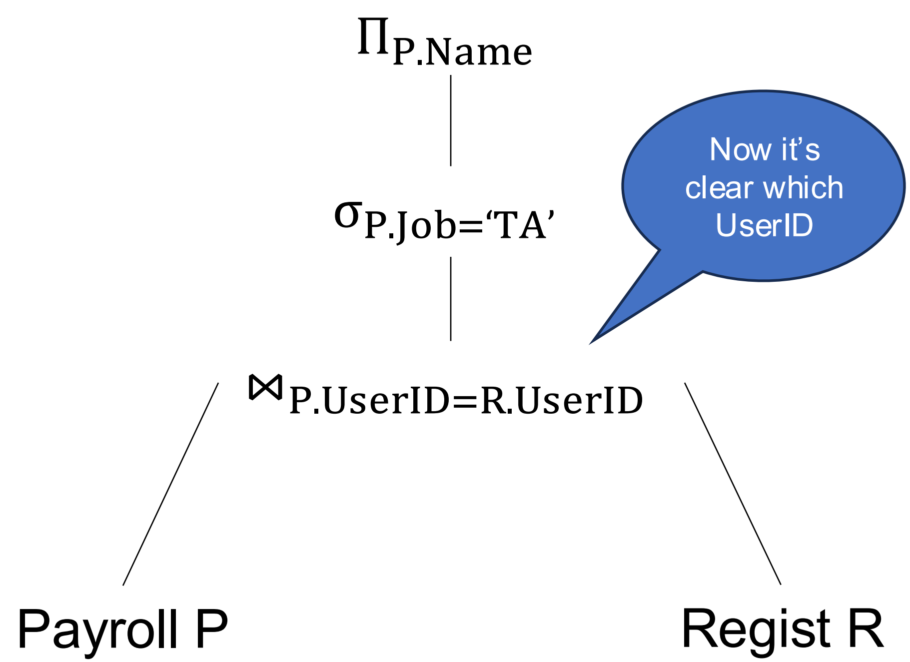
UserID	Car
123	Charger
567	Civic
567	Pinto

Query Plan: Attribute Names

Managing attribute names correctly is tedious

Better: use aliases, much like in SQL





Query Plan: Execution Order

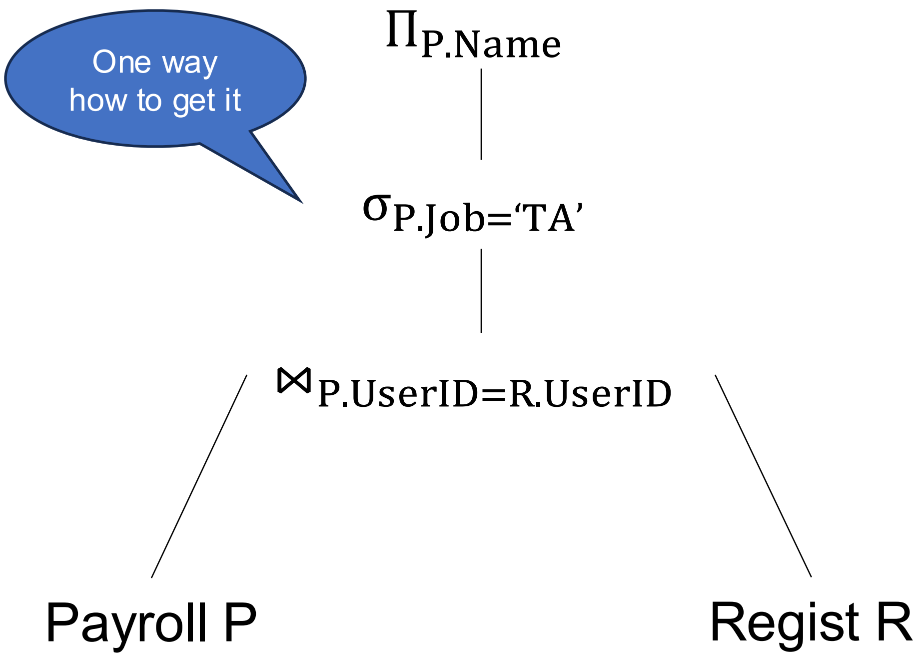
```
SELECT P.Name
FROM Payroll P, Regist R
WHERE P.UserID = R.UserID
   and P.Job = 'TA';
```

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';
```

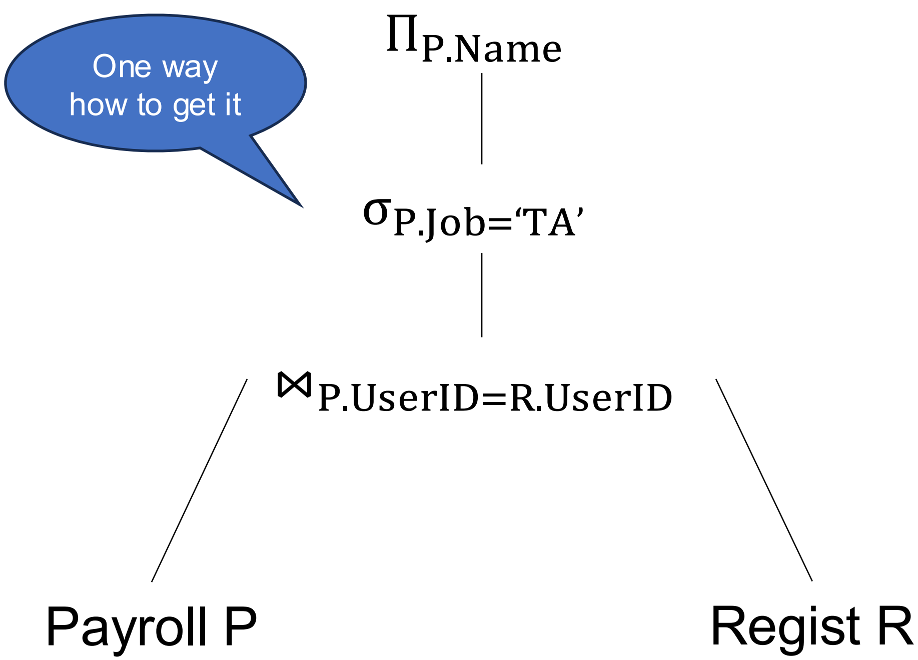
We say what we want, not how to get it

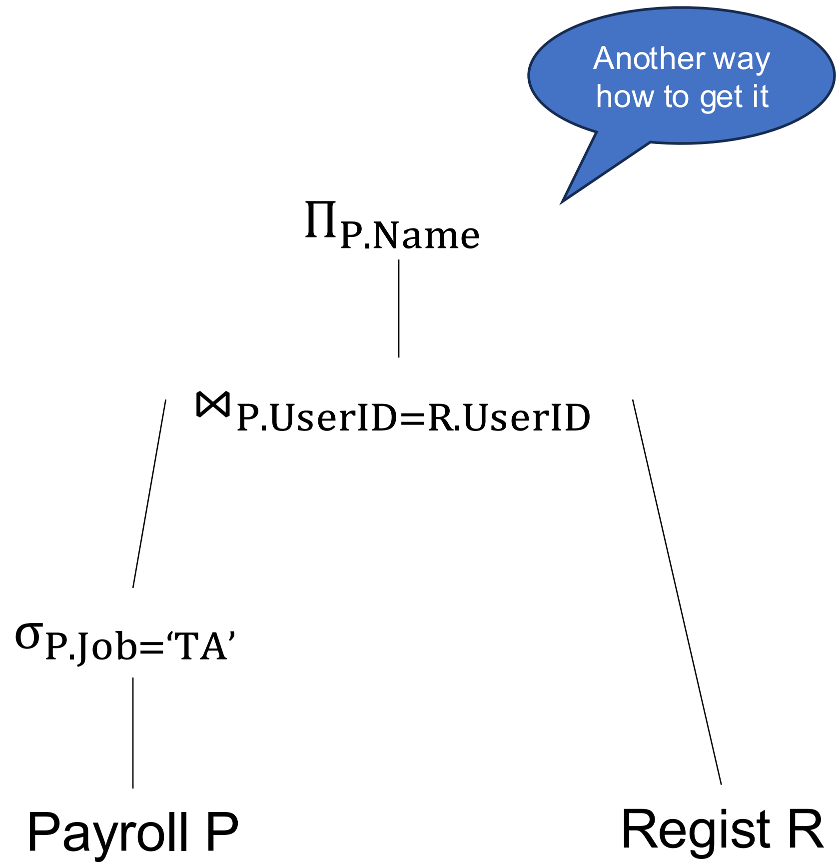


Query Plan: Execution Order

```
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

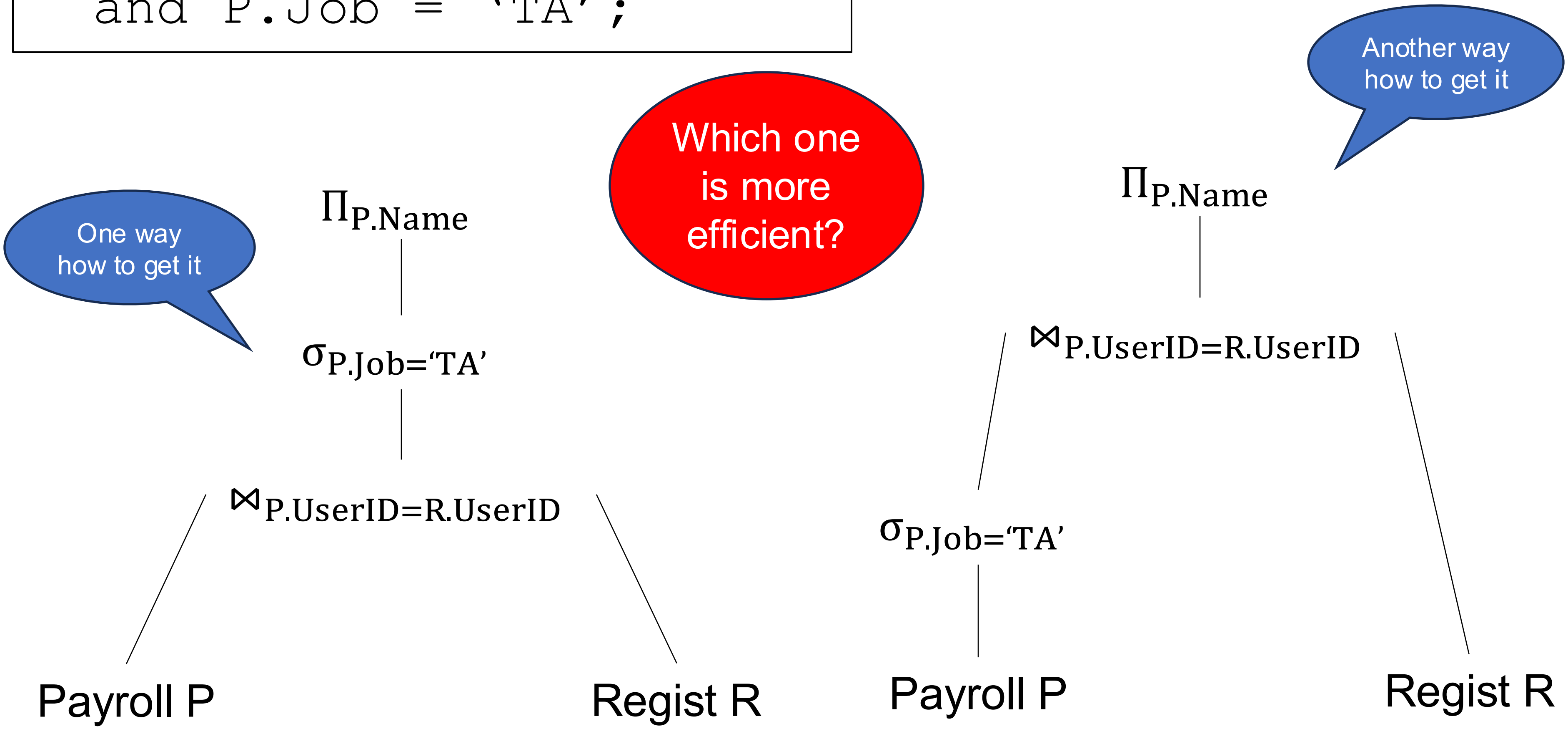




Query Plan: Execution Order

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

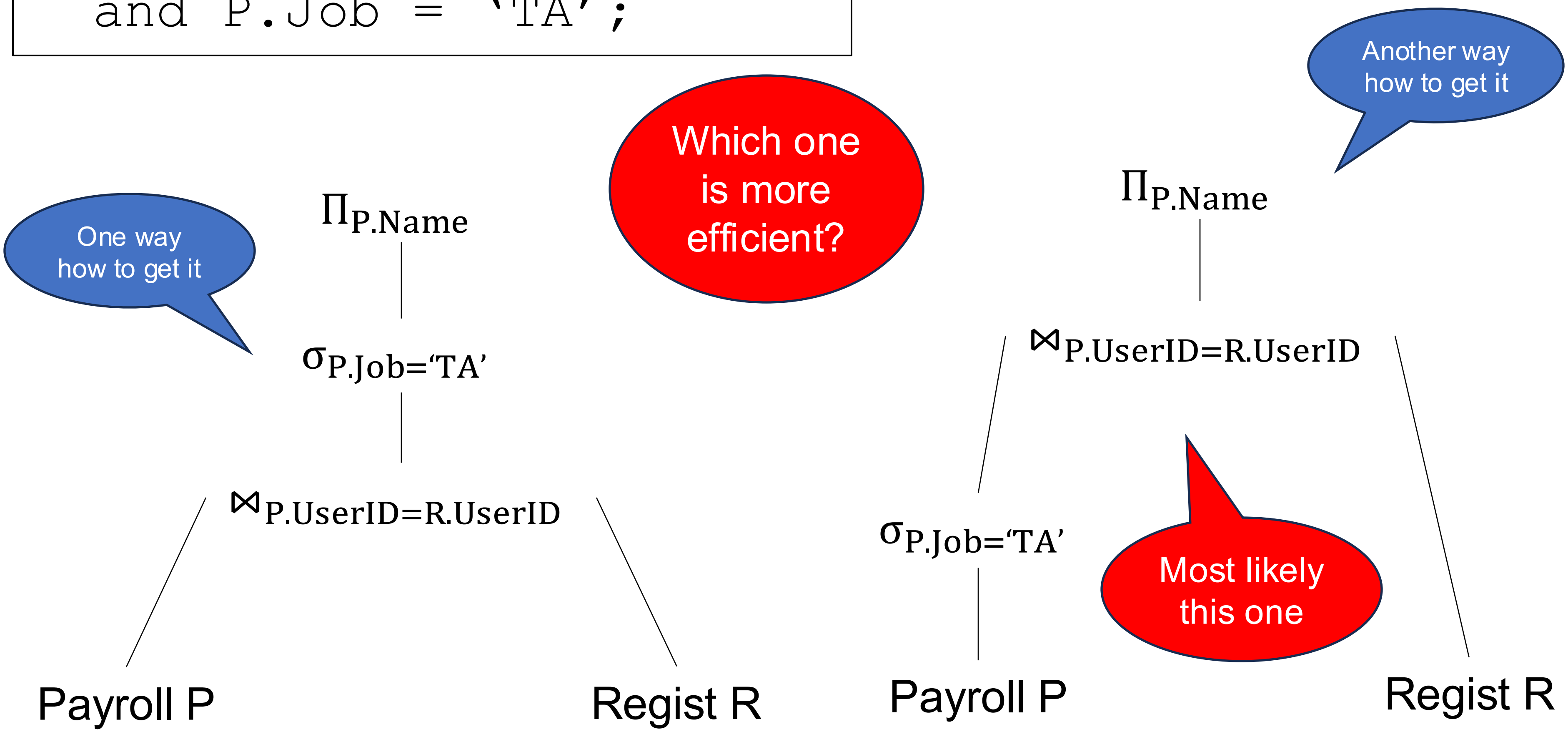
We say what we want,
not how to get it



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:

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

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

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

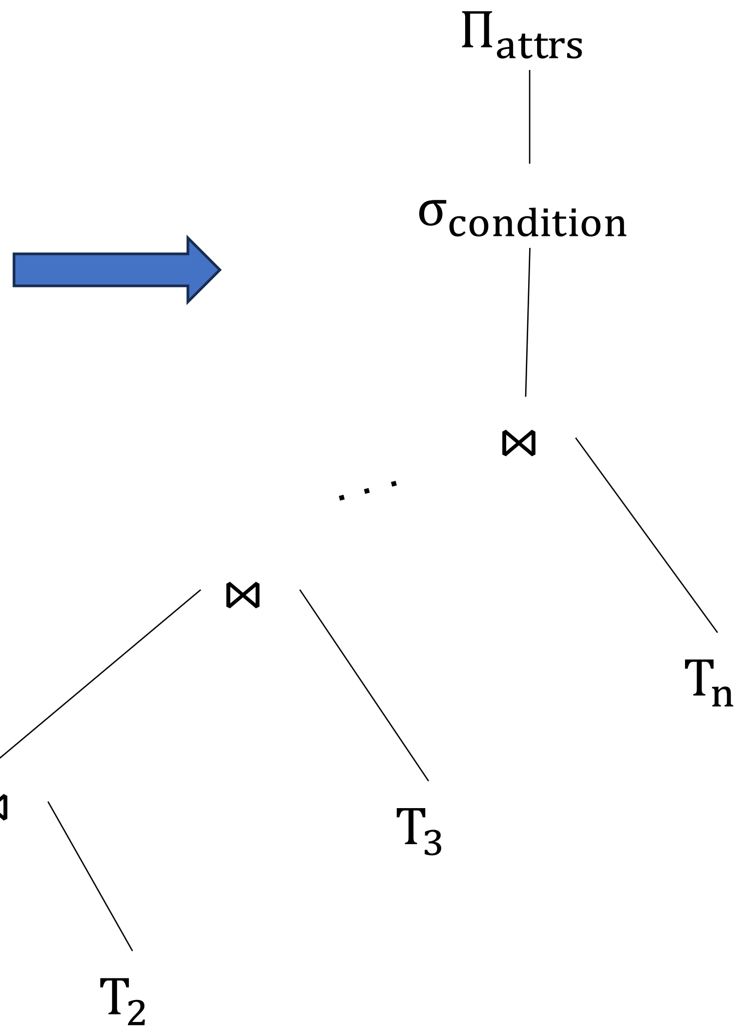
… many others*

*over 500 rules in SQL Server

SQL to RA

Single SELECT-FROM-WHERE query:

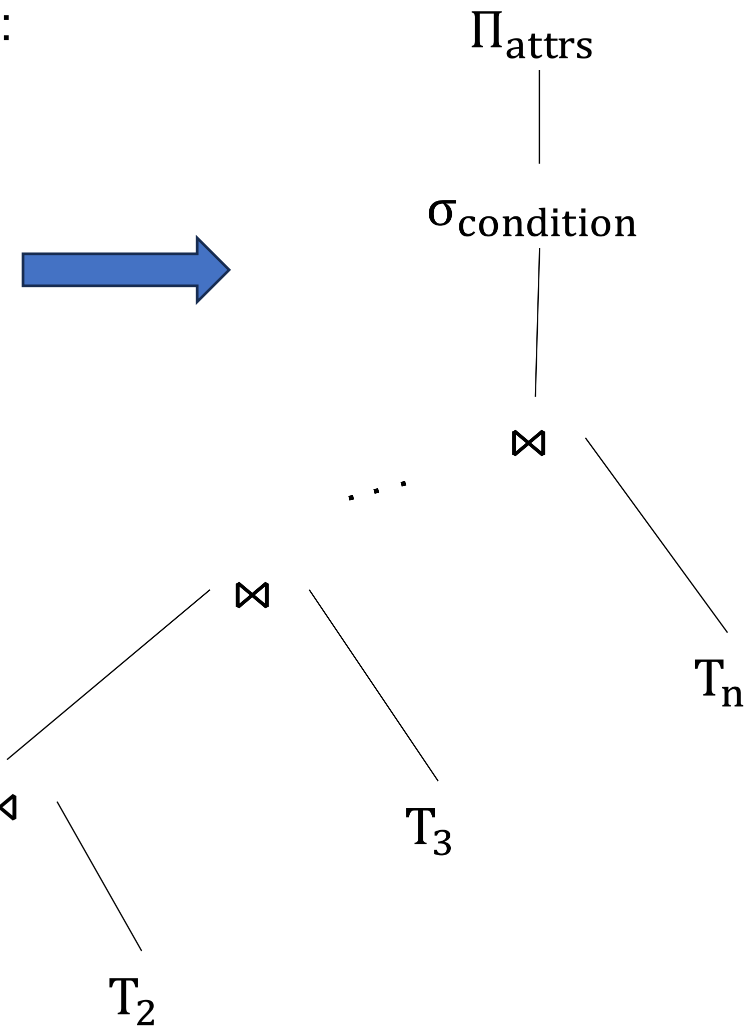
```
SELECT attrs
FROM T1,T2,...,Tn
WHERE condition;
```



SQL to RA

Single SELECT-FROM-WHERE query:

```
SELECT attrs
FROM T1,T2,...,Tn
WHERE condition;
```



Next: to convert group-by we need to extend RA

Extended Relational Algebra

- Duplicate elimination δ

- Group-by aggregate $\gamma_{\text{attr1,attr2,…,agg1,…}}$

Duplicate Elimination

$\delta(T)$

Eliminates duplicates from the bag T

```
SELECT DISTINCT *
FROM T;
```

Duplicate Elimination

$\delta(\mathrm{T})$

Eliminates duplicates from the bag T

```
SELECT DISTINCT *
FROM T;
```

$\delta(\mathrm{R}) =$

R

A	B
1	10
2	10
2	10
2	20
1	10

Duplicate Elimination

$$\delta(\mathrm{T})$$

Eliminates duplicates from the bag T

```
SELECT DISTINCT *
FROM T;
```

A	B
1	10
2	10
2	20

$$\delta(\mathrm{R}) =$$

R

A	B
1	10
2	10
2	10
2	20
1	10

GroupBy-Aggregate

$$\gamma_{attr1, attr2, \ldots, agg1, \ldots}(\mathrm{T})$$

Group-by, then aggregate

```
SELECT attr1,...,agg1,...
FROM T
GROUP BY attr1,...;
```

GroupBy-Aggregate

$$\gamma_{attr1,attr2,\ldots,agg1,\ldots}(\mathrm{T})$$

Group-by, then aggregate

$$\gamma_{\mathrm{Job,avg(Salary)}\rightarrow\mathrm{S}}(\mathrm{Payroll}) =$$

```
SELECT attr1,...,agg1,...
FROM T
GROUP BY attr1,...;
```

Payroll

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

GroupBy-Aggregate

$$\gamma_{attr1, attr2, \ldots, agg1, \ldots}(\mathrm{T})$$

Job	S
TA	55000
Prof	95000

Group-by, then aggregate

$$\gamma_{\mathrm{Job}, \mathrm{avg(Salary)} \to \mathrm{S}}(\mathrm{Payroll}) =$$

```
SELECT attr1,...,agg1,...
FROM T
GROUP BY attr1,...;
```

Payroll

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

GroupBy-Aggregate

No need for a HAVING operator!

Find all jobs where the average salary of employees earning over 55000 is < 70000

Payroll

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

GroupBy-Aggregate

No need for a HAVING operator!

Find all jobs where the average salary of employees earning over 55000 is < 70000

```
SELECT Job
FROM Payroll
WHERE Salary > 55000
GROUP BY Job
HAVING avg(Salary)<70000;
```

Payroll

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

GroupBy-Aggregate

No need for a HAVING operator!

Find all jobs where the average salary of employees earning over 55000 is < 70000

```
SELECT Job
FROM Payroll
WHERE Salary > 55000
GROUP BY Job
HAVING avg(Salary)<70000;
```

$$\Pi_{\text{Job}}$$

$$\sigma_{\text{S<70000'}}$$

$$\gamma_{\text{Job,avg(Salary)}\rightarrow \text{S}}$$

$$\sigma_{\text{Salary>55000}}$$

Payroll

Payroll

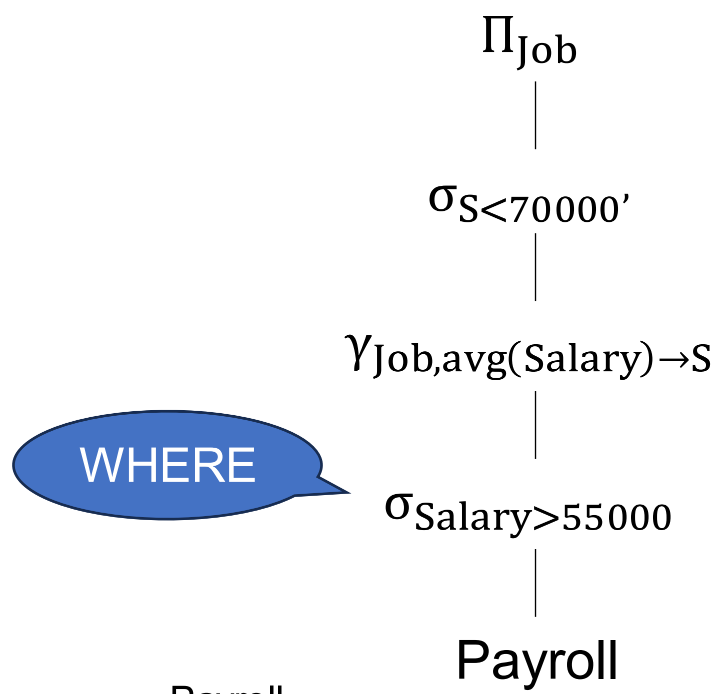
UserID	Name	Job	Salary
123	Jack	TA	50000
345	Allison	TA	60000
567	Magda	Prof	90000
789	Dan	Prof	100000

GroupBy-Aggregate

No need for a HAVING operator!

Find all jobs where the average salary of employees earning over 55000 is < 70000

```
SELECT Job
FROM Payroll
WHERE Salary > 55000
GROUP BY Job
HAVING avg(Salary)<70000;
```



Payroll

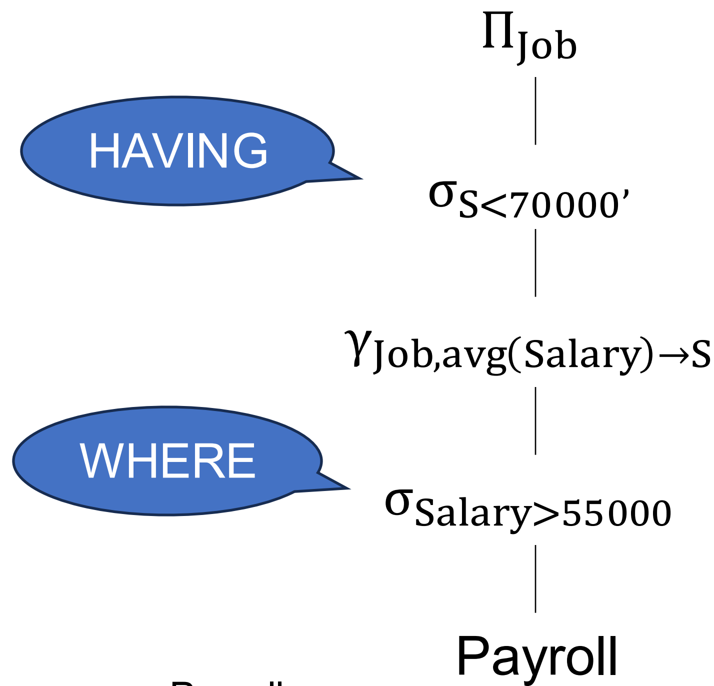
UserID	Name	Job	Salary
123	Jack	TA	50000
345	Allison	TA	60000
567	Magda	Prof	90000
789	Dan	Prof	100000

GroupBy-Aggregate

No need for a HAVING operator!

Find all jobs where the average salary of employees earning over 55000 is < 70000

```
SELECT Job
FROM Payroll
WHERE Salary > 55000
GROUP BY Job
HAVING avg(Salary)<70000;
```



Payroll

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

Discussion

The Greek alphabet soup:

- $\sigma, \Pi, \delta, \gamma$

- They are standard RA symbols, get used to them

Next: converting nested SQL queries to RA

Nested SQL to RA

Nested Queries to RA

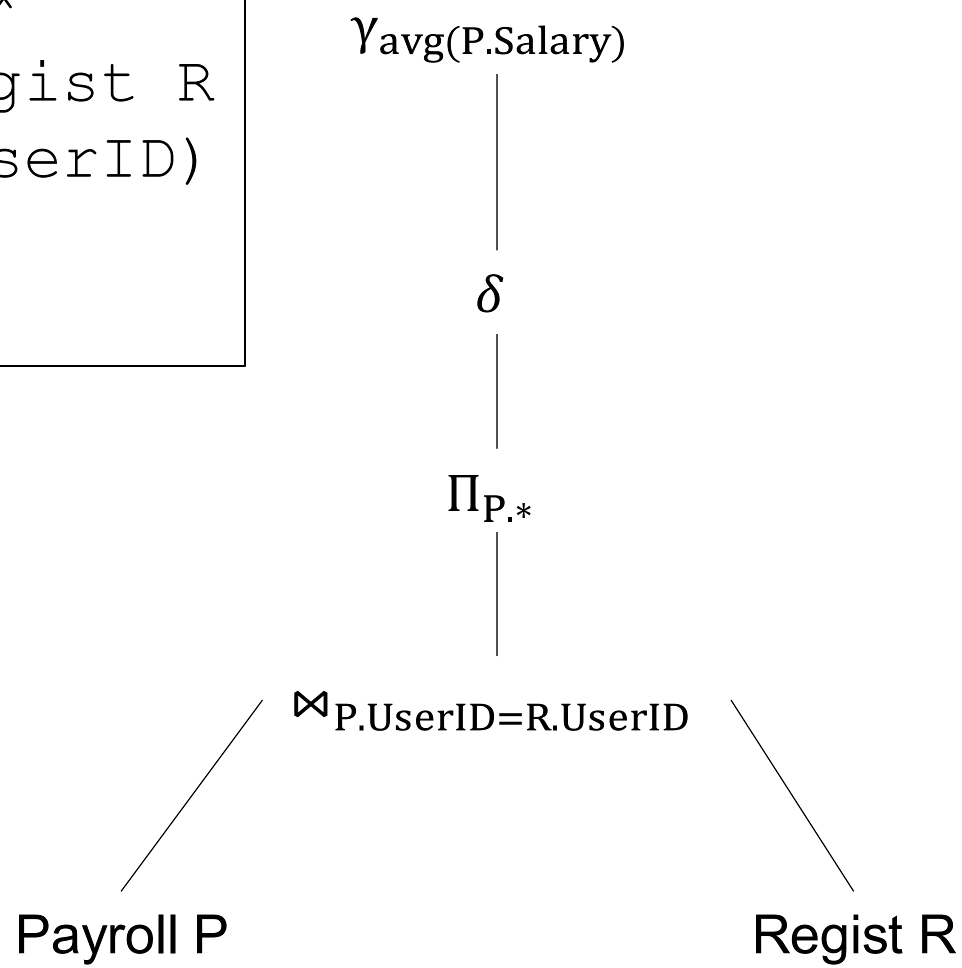
- RA is an algebra: has no nested expressions
- We cannot write EXISTS or NOT EXISTS in σ
- First unnest SQL query, then convert to RA

A Simple Case: the WITH Clause

```
WITH Cardrivers AS
   (SELECT DISTINCT P.*
    FROM Payroll P, Regist R
    WHERE P.UserId=R.UserID)
SELECT avg(Salary)
FROM Cardrivers;
```

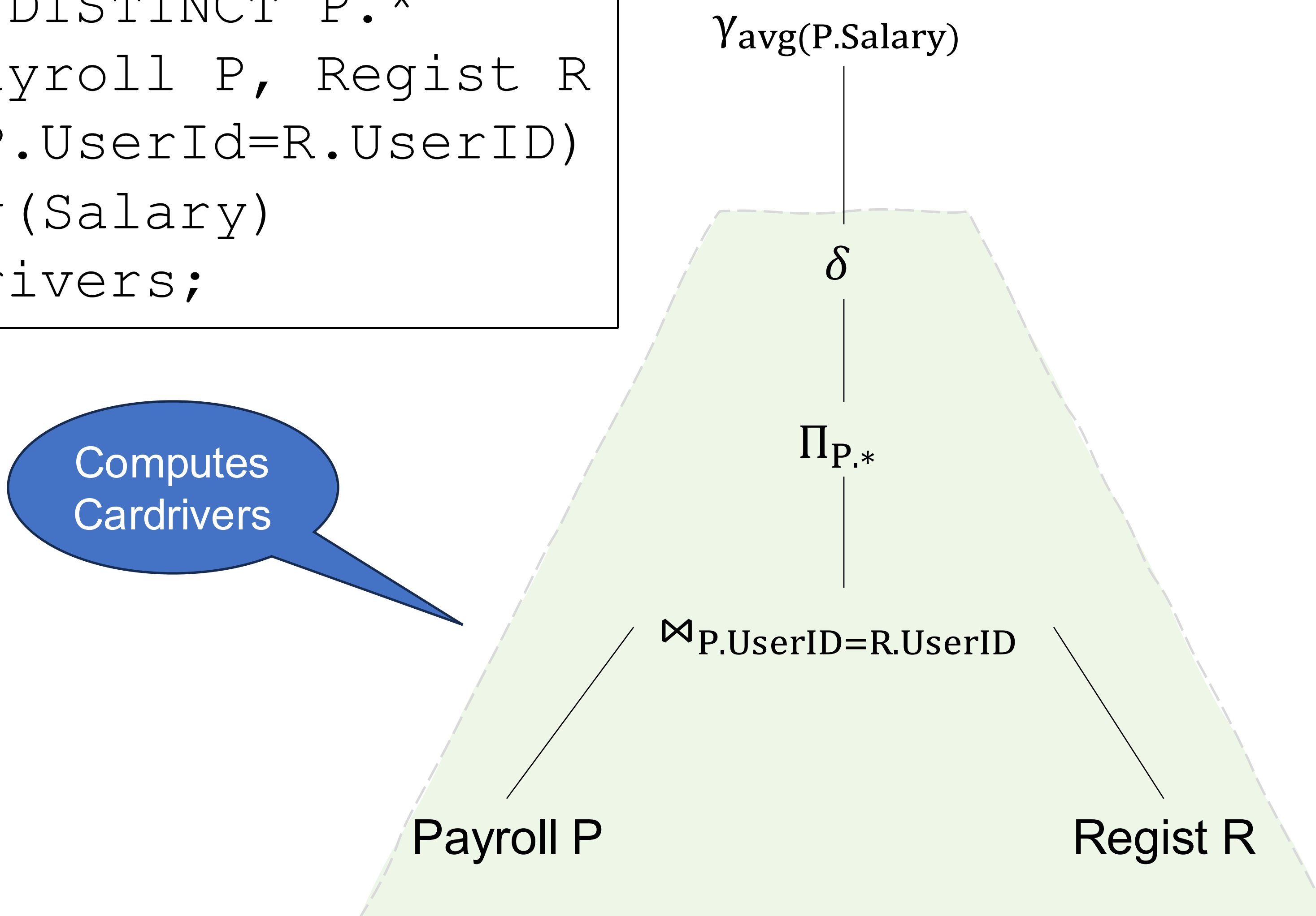
A Simple Case: the WITH Clause

```
WITH Cardrivers AS
   (SELECT DISTINCT P.*
    FROM Payroll P, Regist R
    WHERE P.UserId=R.UserID)
SELECT avg(Salary)
FROM Cardrivers;
```



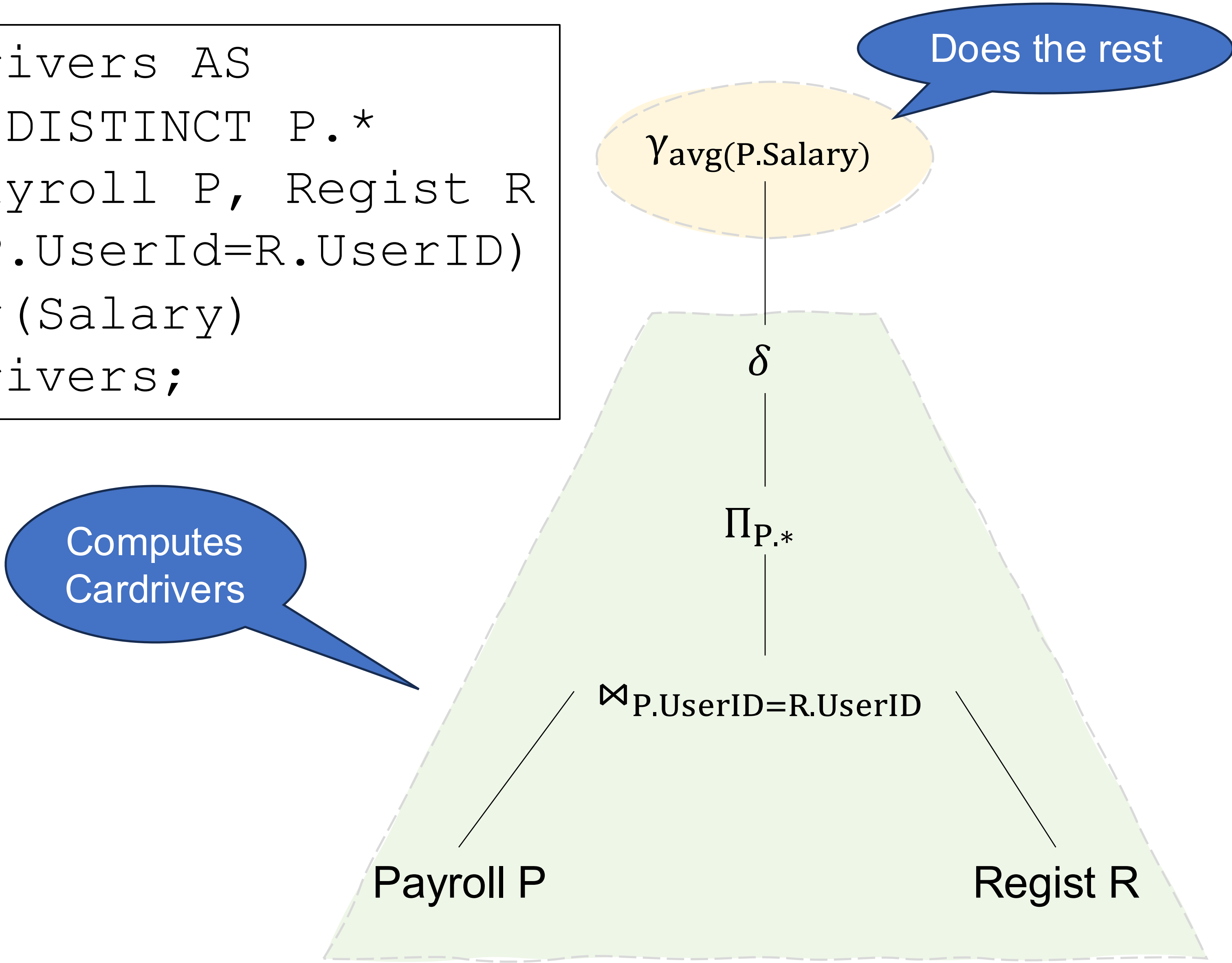
A Simple Case: the WITH Clause

```
WITH Cardrivers AS
   (SELECT DISTINCT P.*
    FROM Payroll P, Regist R
    WHERE P.UserId=R.UserID)
SELECT avg(Salary)
FROM Cardrivers;
```



A Simple Case: the WITH Clause

```
WITH Cardrivers AS
   (SELECT DISTINCT P.*
    FROM Payroll P, Regist R
    WHERE P.UserId=R.UserID)
SELECT avg(Salary)
FROM Cardrivers;
```



A Simple Case: a Monotone Query

```
SELECT P.UserID, P.Name
FROM Payroll P
WHERE exists
       (SELECT *
        FROM Regist R
        WHERE P.UserID = R.UserID);
```

A Simple Case: a Monotone Query

```
SELECT P.UserID, P.Name
FROM Payroll P
WHERE exists
       (SELECT *
        FROM Regist R
        WHERE P.UserID = R.UserID);
```

First unnest

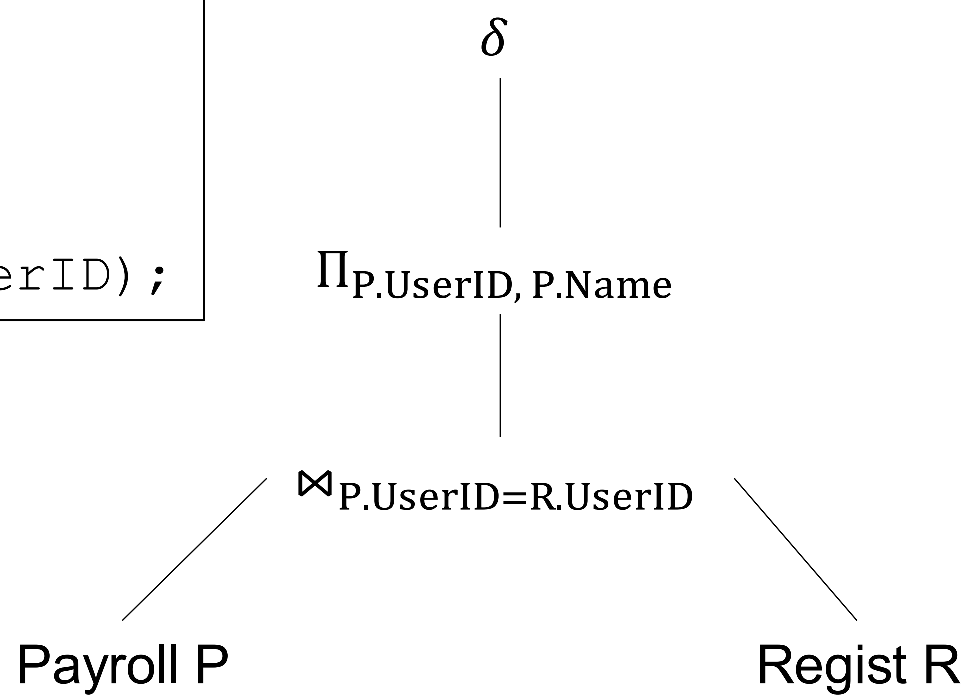
```
SELECT DISTINCT P.UserID, P.Name
FROM Payroll P, Regist R
WHERE P.UserID = R.UserID;
```

A Simple Case: a Monotone Query

```
SELECT P.UserID, P.Name
FROM Payroll P
WHERE exists
       (SELECT *
        FROM Regist R
        WHERE P.UserID = R.UserID);
```

First unnest

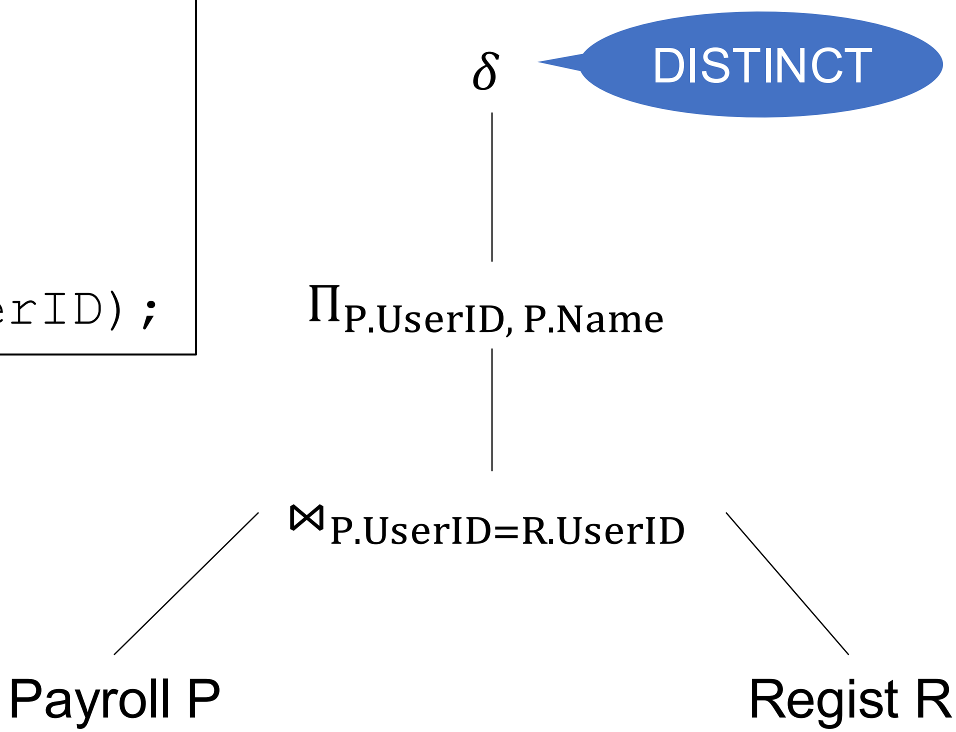
```
SELECT DISTINCT P.UserID, P.Name
FROM Payroll P, Regist R
WHERE P.UserID = R.UserID;
```



The convert to RA

A Simple Case: a Monotone Query

```
SELECT P.UserID, P.Name
FROM Payroll P
WHERE exists
       (SELECT *
        FROM Regist R
        WHERE P.UserID = R.UserID);
```



First unnest

```
SELECT DISTINCT P.UserID, P.Name
FROM Payroll P, Regist R
WHERE P.UserID = R.UserID;
```

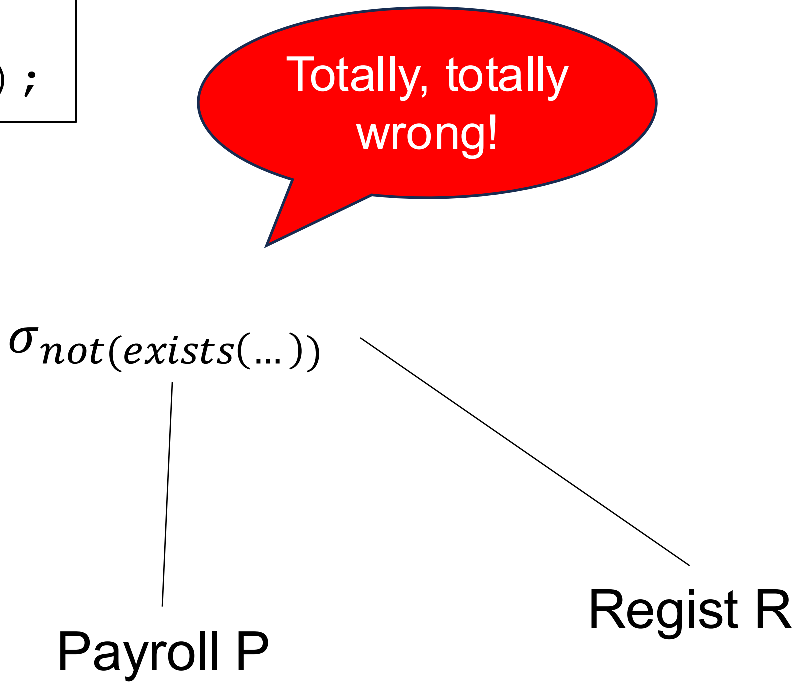
The convert to RA

A Difficult Case: a Non-Monotone Query

```
SELECT P.UserID
FROM Payroll P
WHERE not exists
      (SELECT *
       FROM Regist R
       WHERE P.UserID = R.UserID);
```

A Difficult Case: a Non-Monotone Query

```
SELECT P.UserID
FROM Payroll P
WHERE not exists
       (SELECT *
        FROM Regist R
        WHERE P.UserID = R.UserID);
```

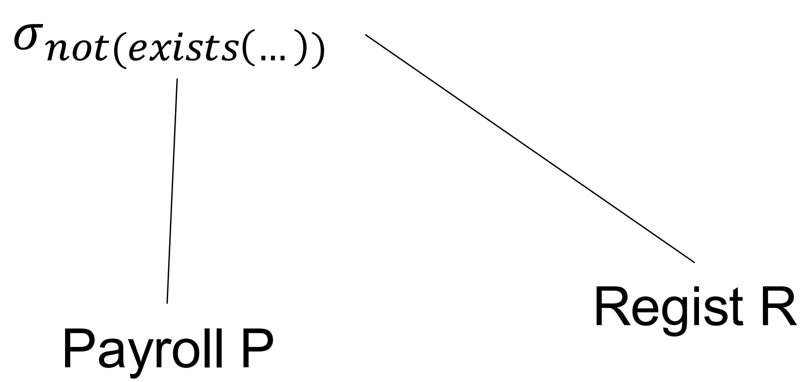


A Difficult Case: a Non-Monotone Query

```
SELECT P.UserID
FROM Payroll P
WHERE not exists
       (SELECT *
        FROM Regist R
        WHERE P.UserID = R.UserID);
```

Totally, totally wrong!

There are no subqueries in RA.

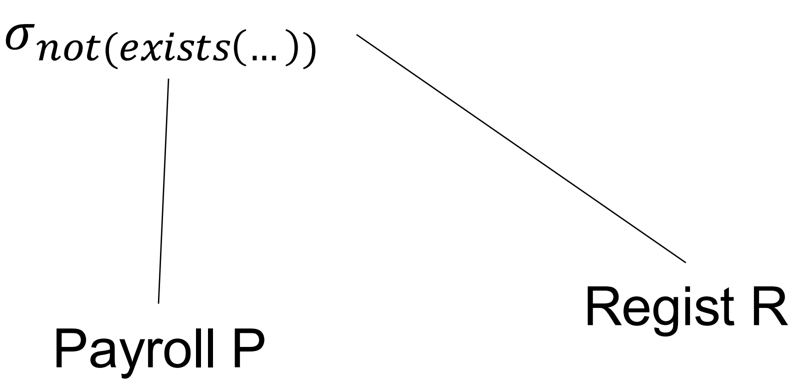


A Difficult Case: a Non-Monotone Query

```
SELECT P.UserID
FROM Payroll P
WHERE not exists
        (SELECT *
         FROM Regist R
         WHERE P.UserID = R.UserID);
```

Totally, totally wrong!

There are no subqueries in RA.



Need to unnest,
but first need to de-correlate.

A Difficult Case: a Non-Monotone Query

```
SELECT P.UserID
FROM Payroll P
WHERE not exists
       (SELECT *
        FROM Regist R
        WHERE P.UserID = R.UserID);
```

First
de-correlate

```
SELECT P.UserID
FROM Payroll P
WHERE P.UserID not in
       (SELECT R.UserID
        FROM Regist R);
```

A Difficult Case: a Non-Monotone Query

```
SELECT P.UserID
FROM Payroll P
WHERE not exists
       (SELECT *
        FROM Regist R
        WHERE P.UserID = R.UserID);
```

First
de-correlate

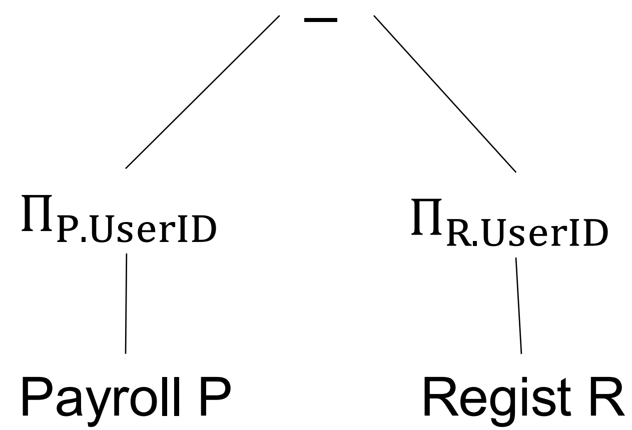
```
SELECT P.UserID
FROM Payroll P
WHERE P.UserID not in
       (SELECT R.UserID
        FROM Regist R);
```

Then unnest
using set difference

```
SELECT P.UserID
FROM Payroll P
   EXCEPT
SELECT R.UserID
FROM Regist R;
```

A Difficult Case: a Non-Monotone Query

```
SELECT P.UserID
FROM Payroll P
WHERE not exists
       (SELECT *
        FROM Regist R
        WHERE P.UserID = R.UserID);
```



First
de-correlate

```
SELECT P.UserID
FROM Payroll P
WHERE P.UserID not in
       (SELECT R.UserID
        FROM Regist R);
```

Then unnest
using set difference

```
SELECT P.UserID
FROM Payroll P
   EXCEPT
SELECT R.UserID
FROM Regist R;
```

Finally,
rewrite to RA

Discussion

- SQL = declarative language; what we want
  RA = an algebra; how to get it

- We write in SQL, optimizers generates RA

- Some language resemble RA more than SQL, e.g. Spark

Next topic: how to design a database from scratch

Database Design

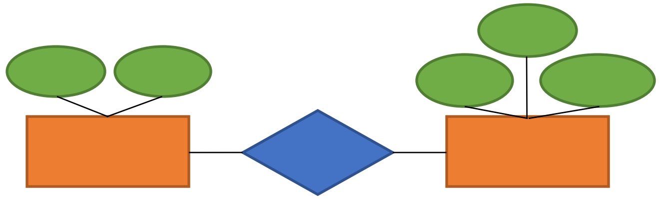
Database Design

- New application needs persistent database.
- The database will persist for a long period of time. We need a good design from day 1.
- Incorporate feedback from many stakeholders
  - Programmers, business teams, analysts, data scientists, product managers, …

The Database Design Process

Today

Conceptual Model

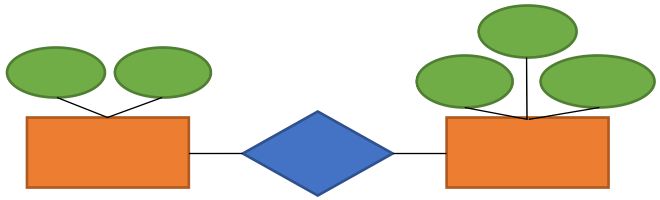


Relational Model
+ Schema
+ Constraints

The Database Design Process

Today

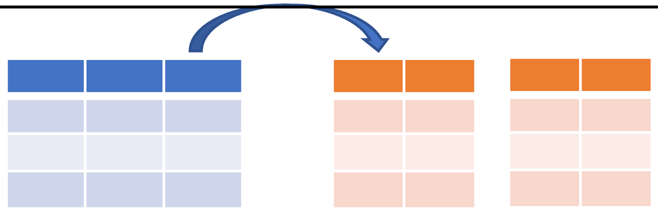
Conceptual Model



Relational Model
+ Schema
+ Constraints

Next Lectures

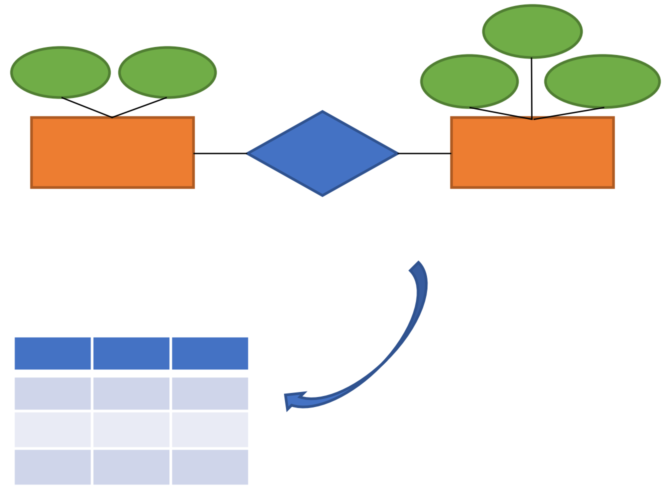
Conceptual Schema
+ Normalization



The Database Design Process

Today

Conceptual Model



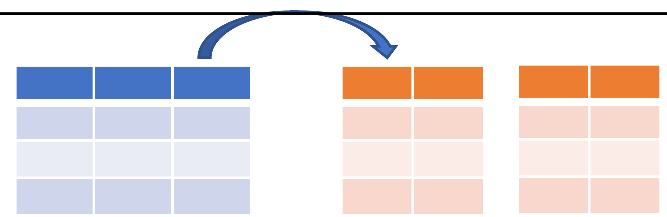
Relational Model

+ Schema
+ Constraints

Next Lectures

Conceptual Schema

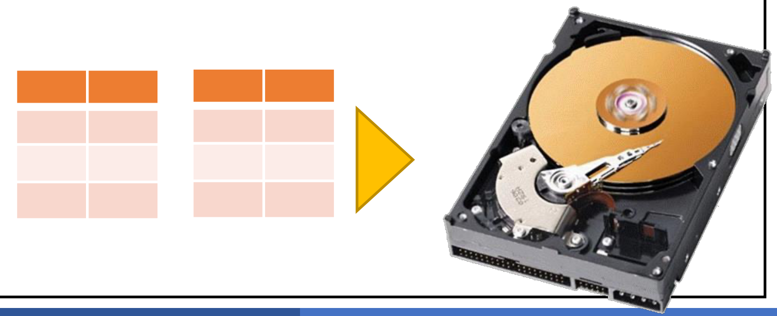
+ Normalization



Later…

Physical Schema

+ Partitioning
+ Indexing



ER Diagrams

Entity-Relationship (ER) Diagrams

- A visual way to describe the schema of a database
- Language independent: may implement in SQL, or some other data model

Example

Application to track the lifetime of products

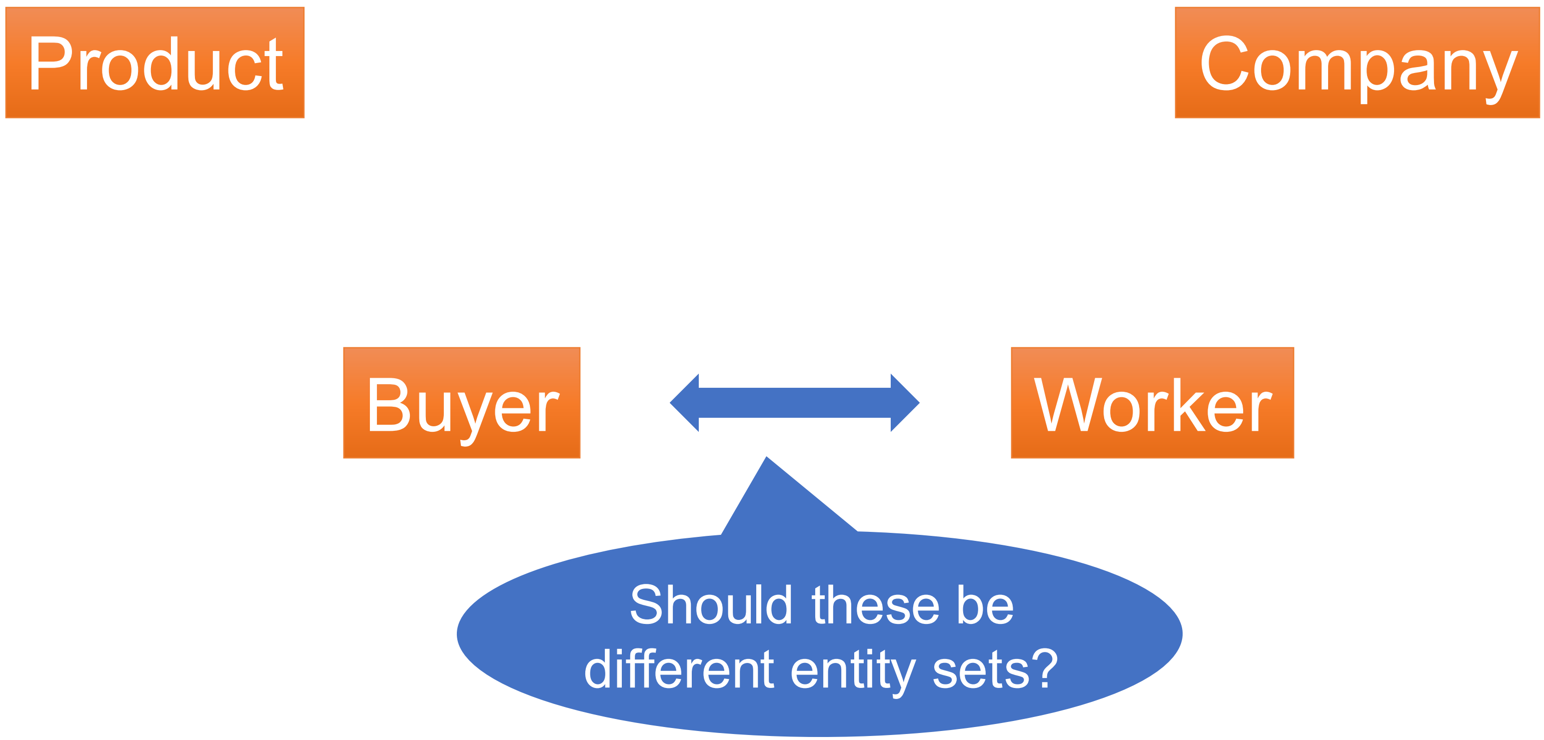
- Keep information about Products: name, price, …
- Who manufactures them? Company name, address, their workers, …
- Who buys them? Customers with their names, …

Example: designing the Entity Sets

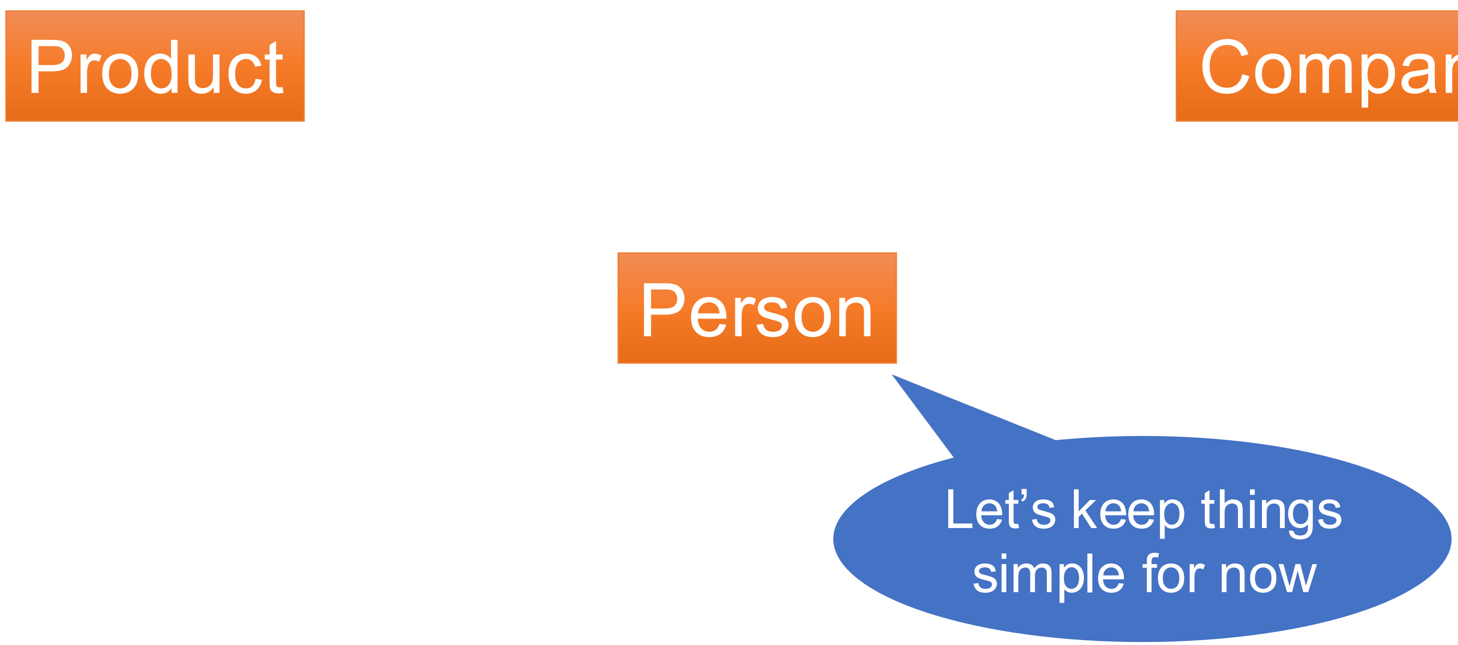
Example: designing the Entity Sets

Example: designing the Entity Sets

Example: designing the Entity Sets

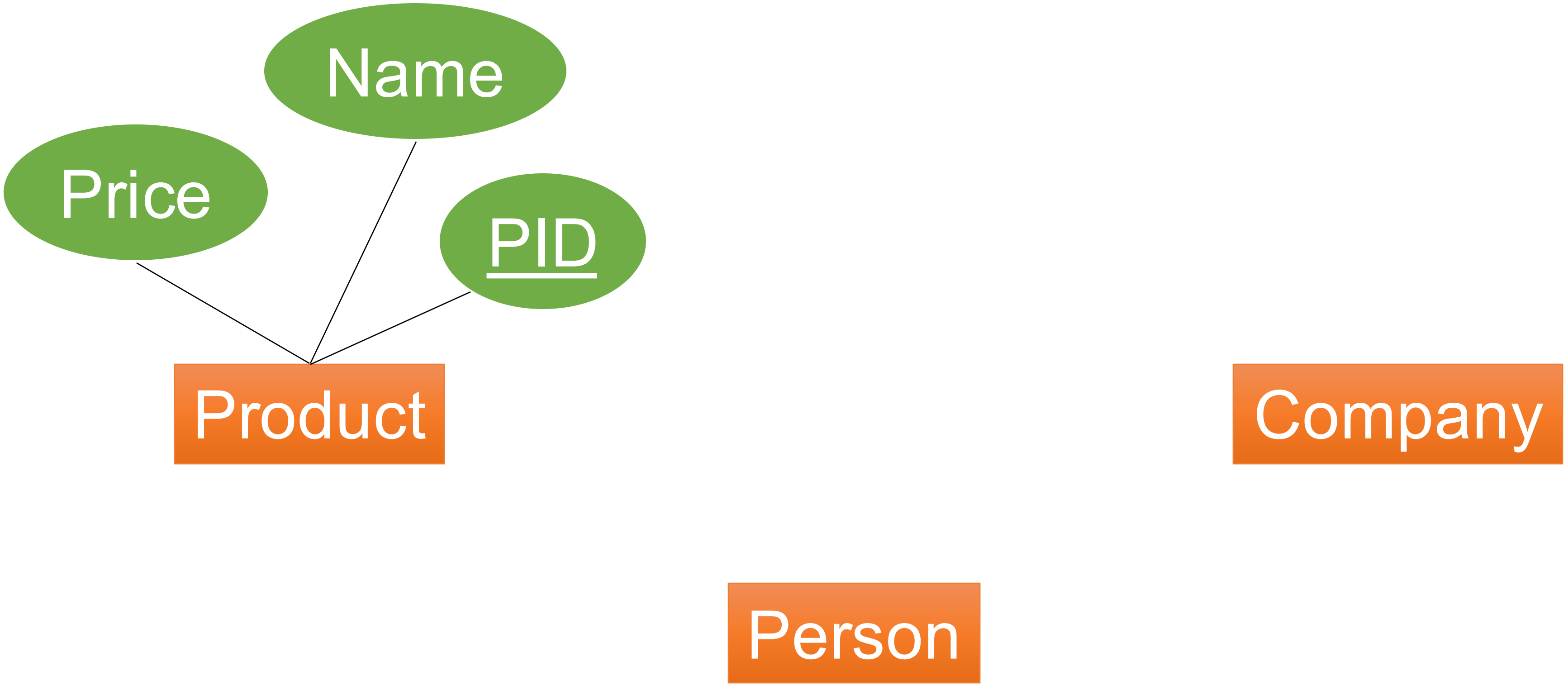


Example: designing the Entity Sets

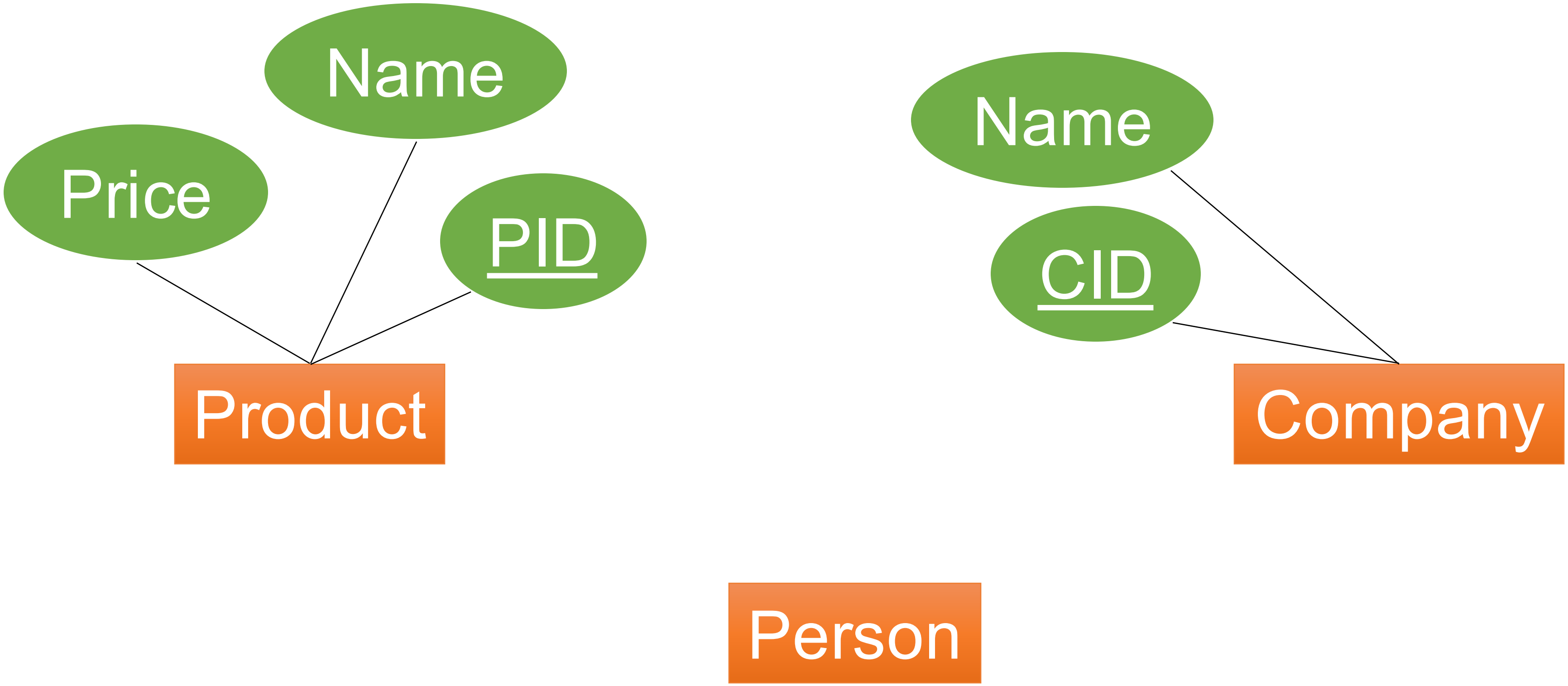


Example: adding Attributes

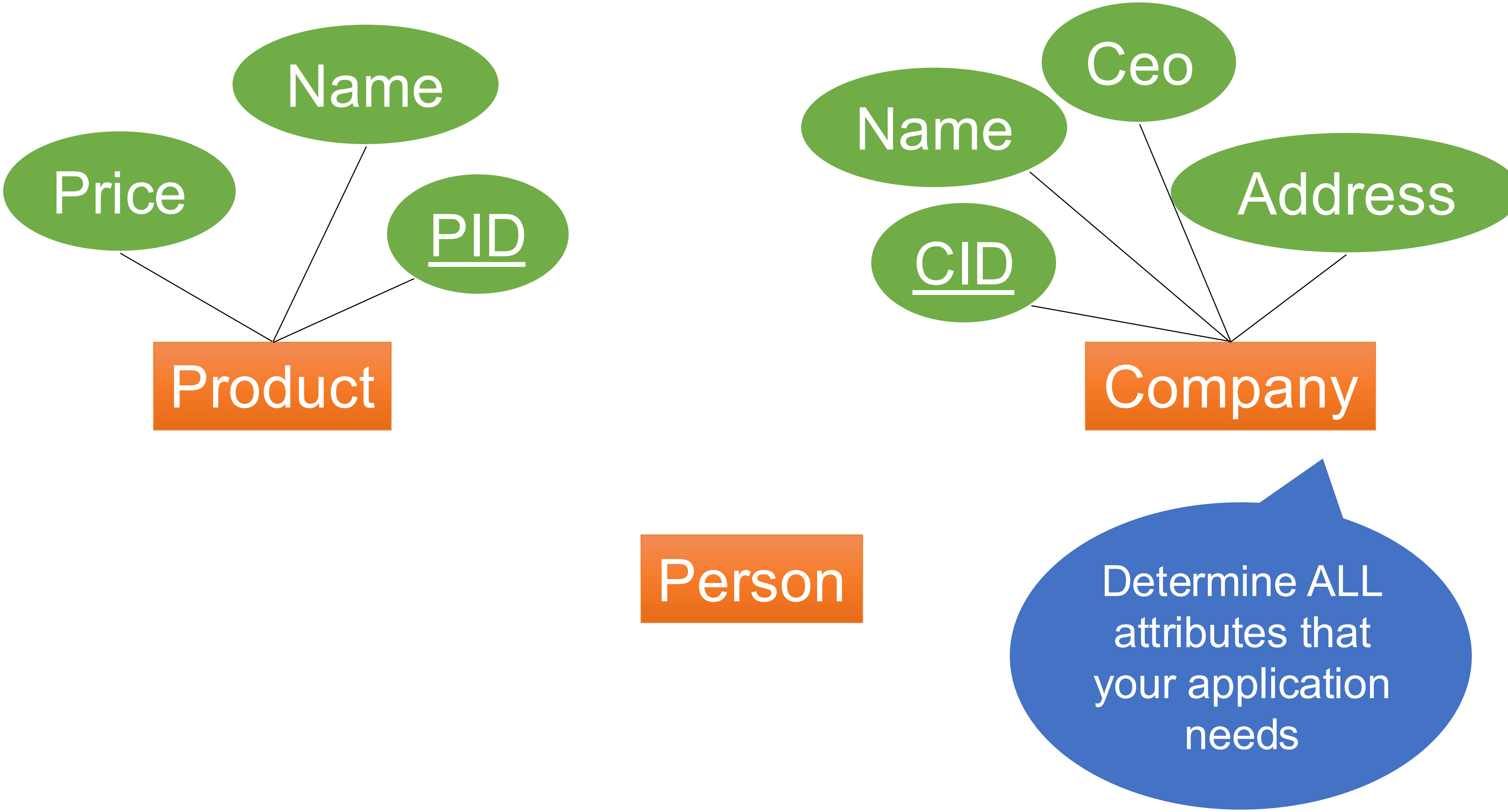
Example: adding Attributes



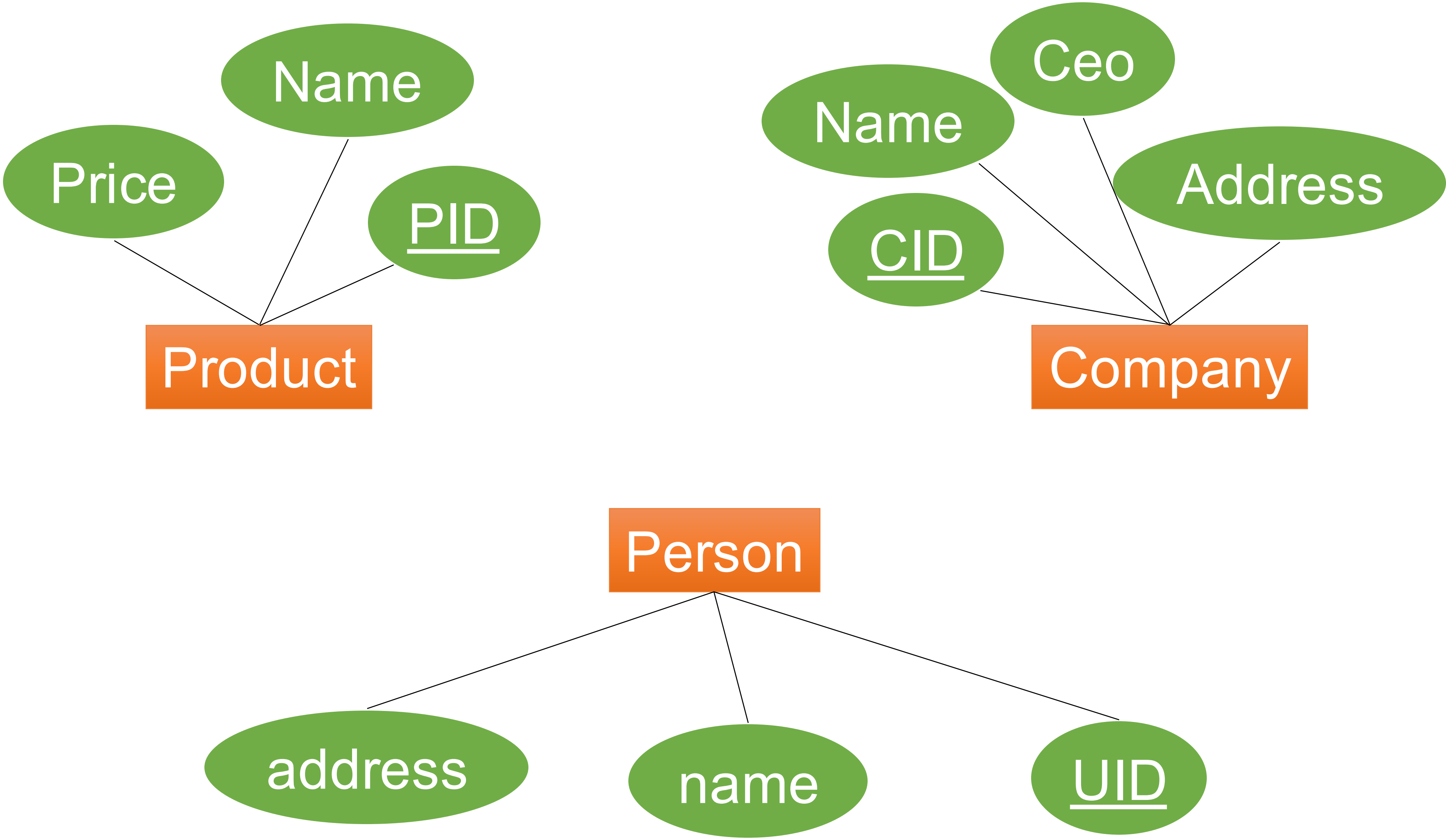
Example: adding Attributes



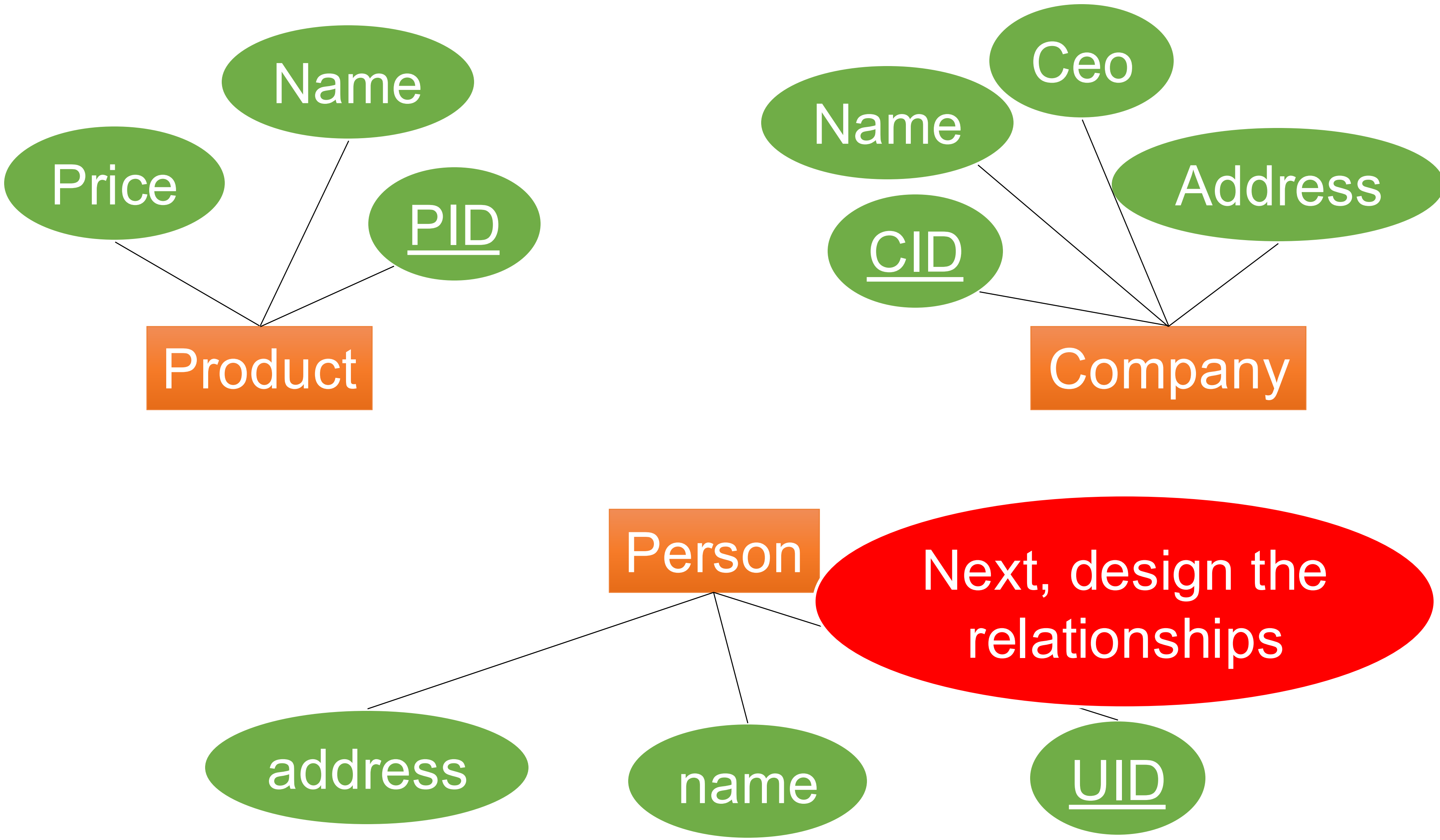
Example: adding Attributes



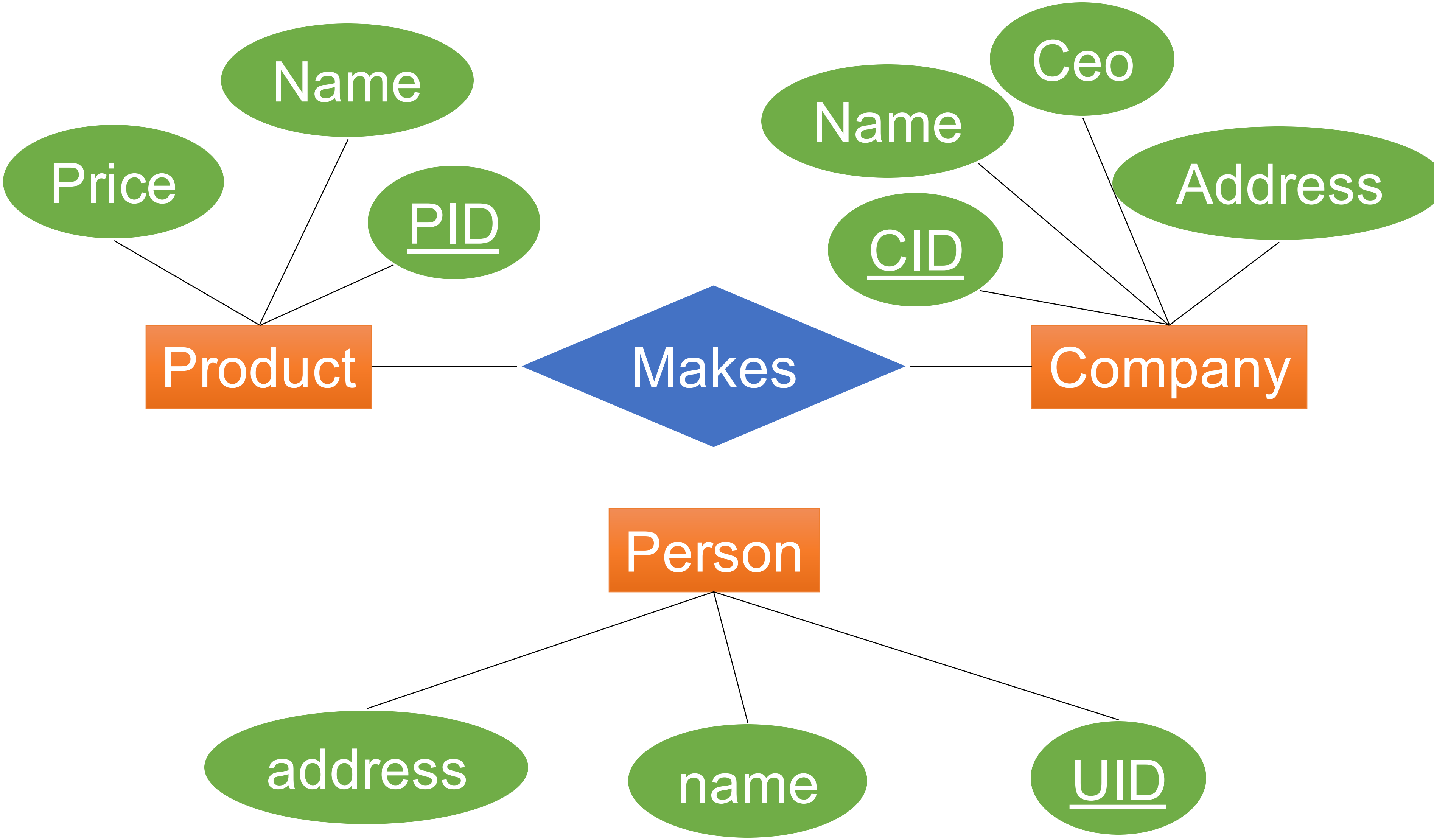
Example: adding Attributes



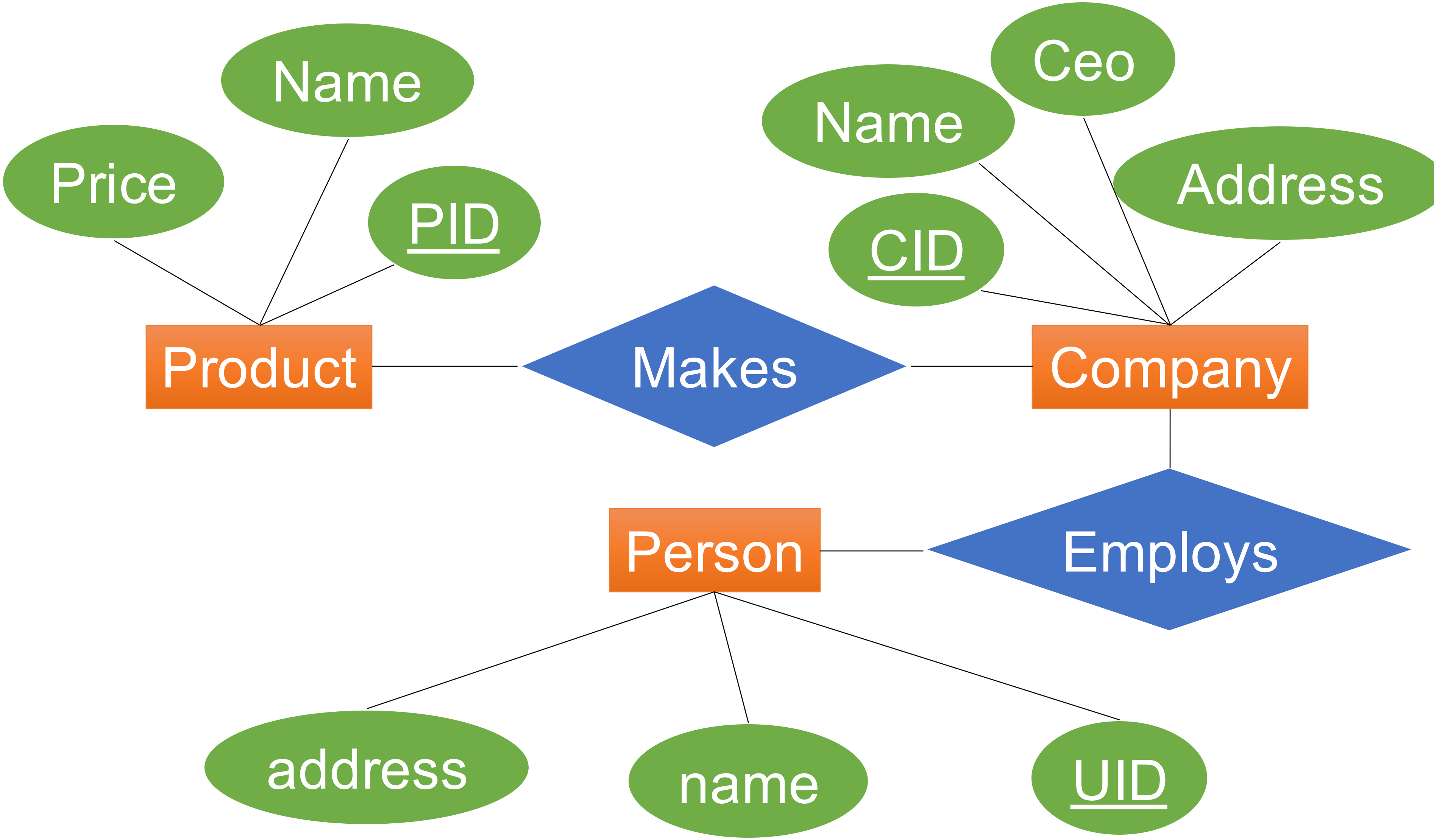
Example: adding Relationships



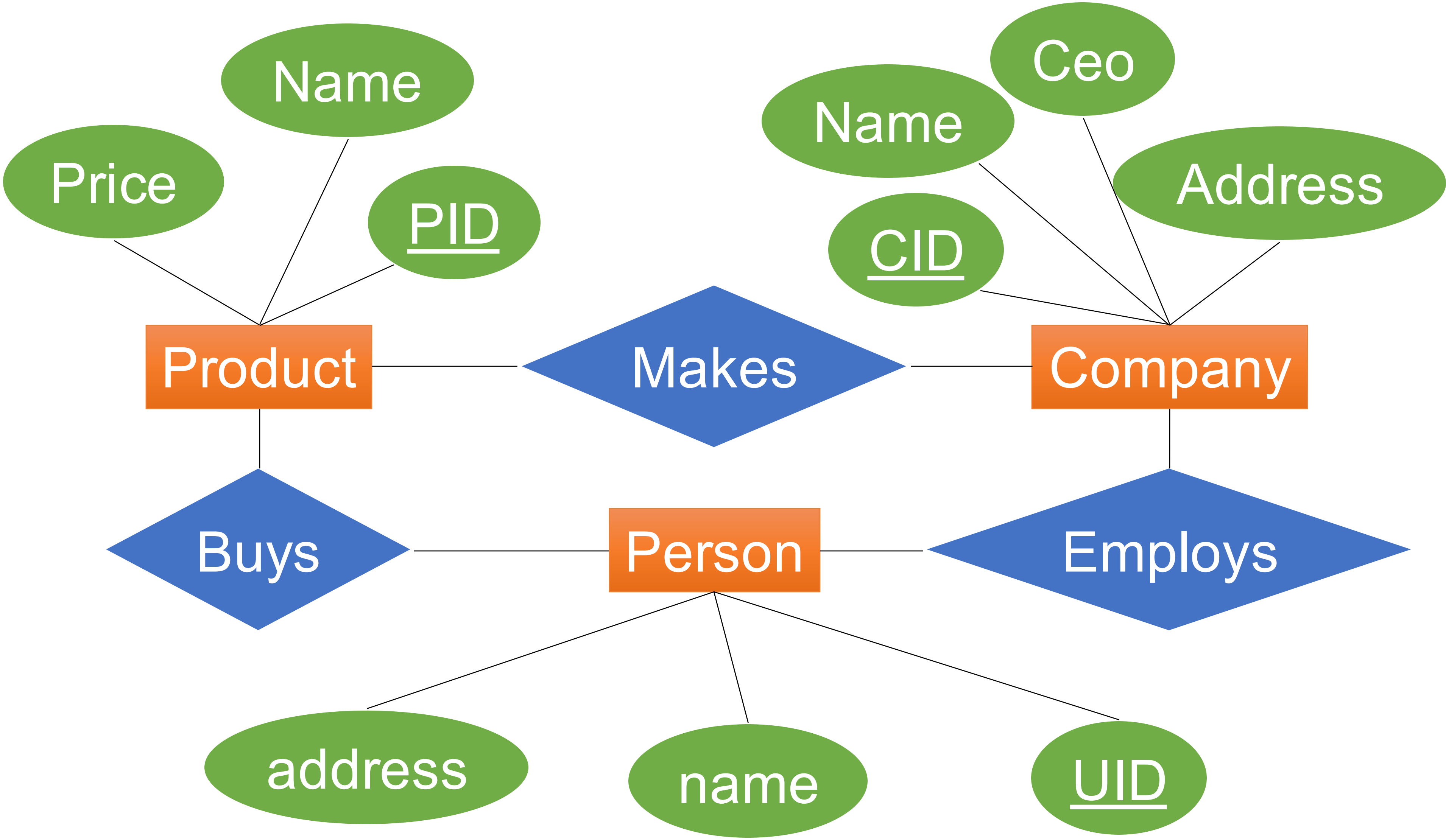
Example: adding Relationships



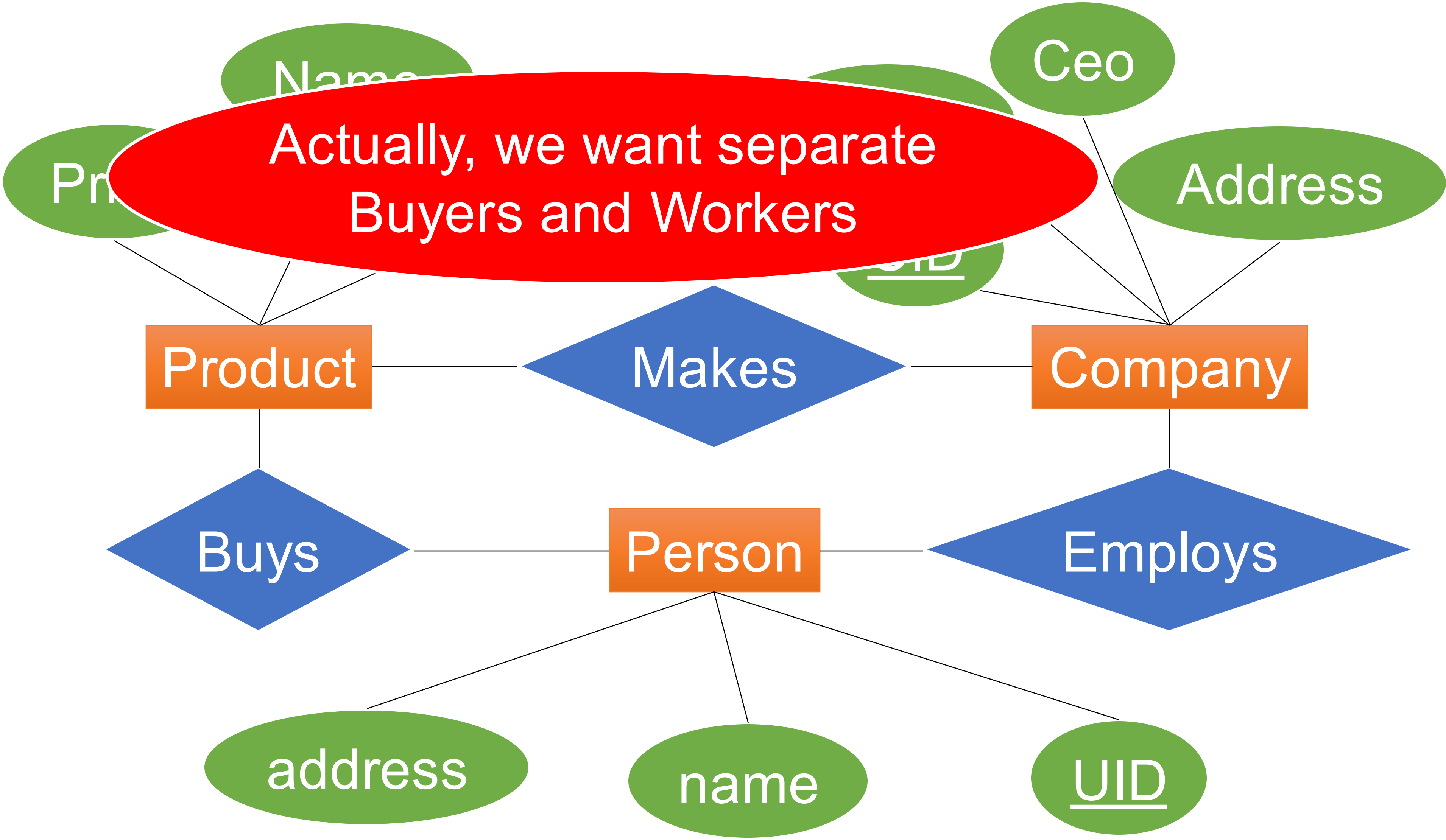
Example: adding Relationships



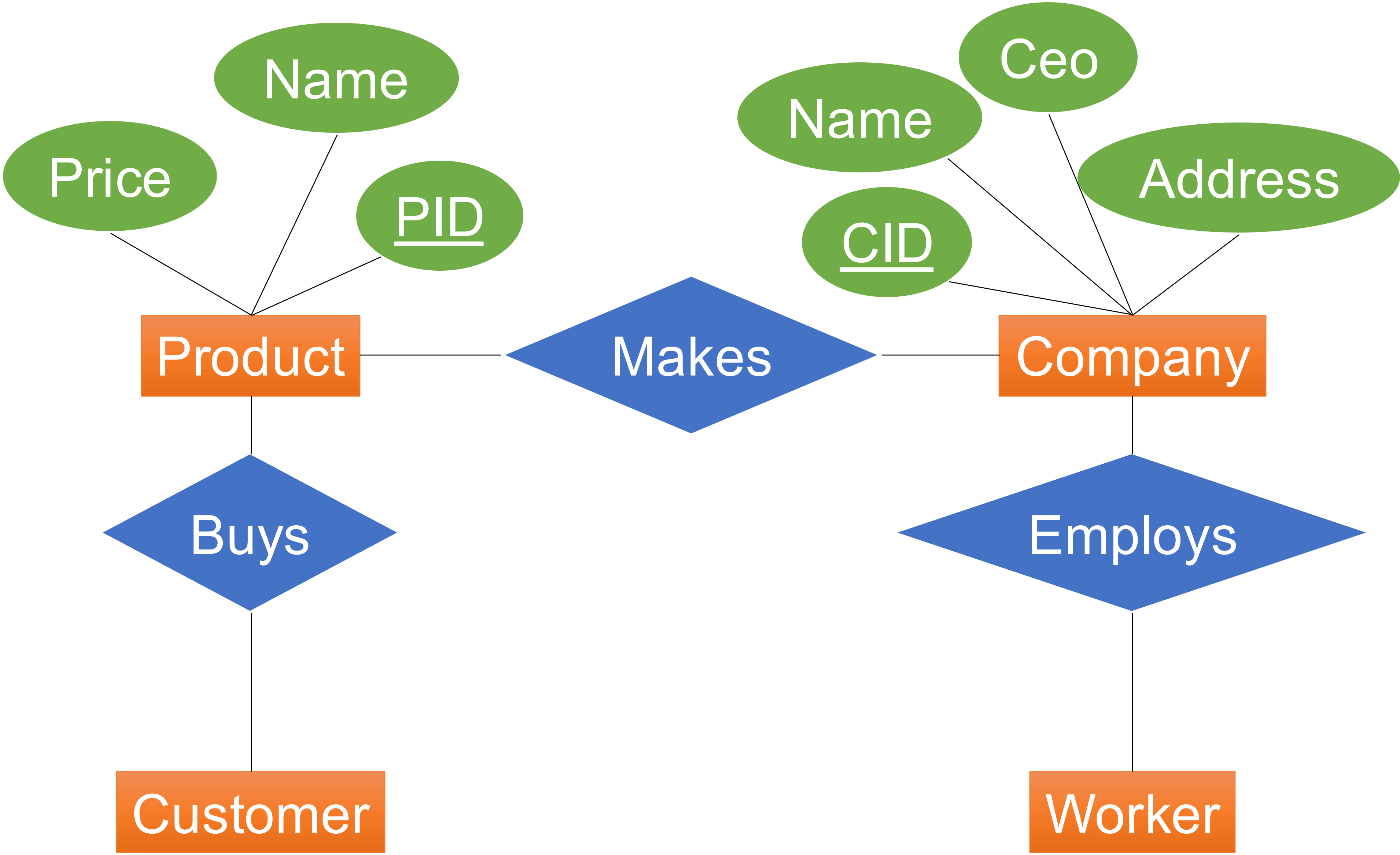
Example: adding Relationships



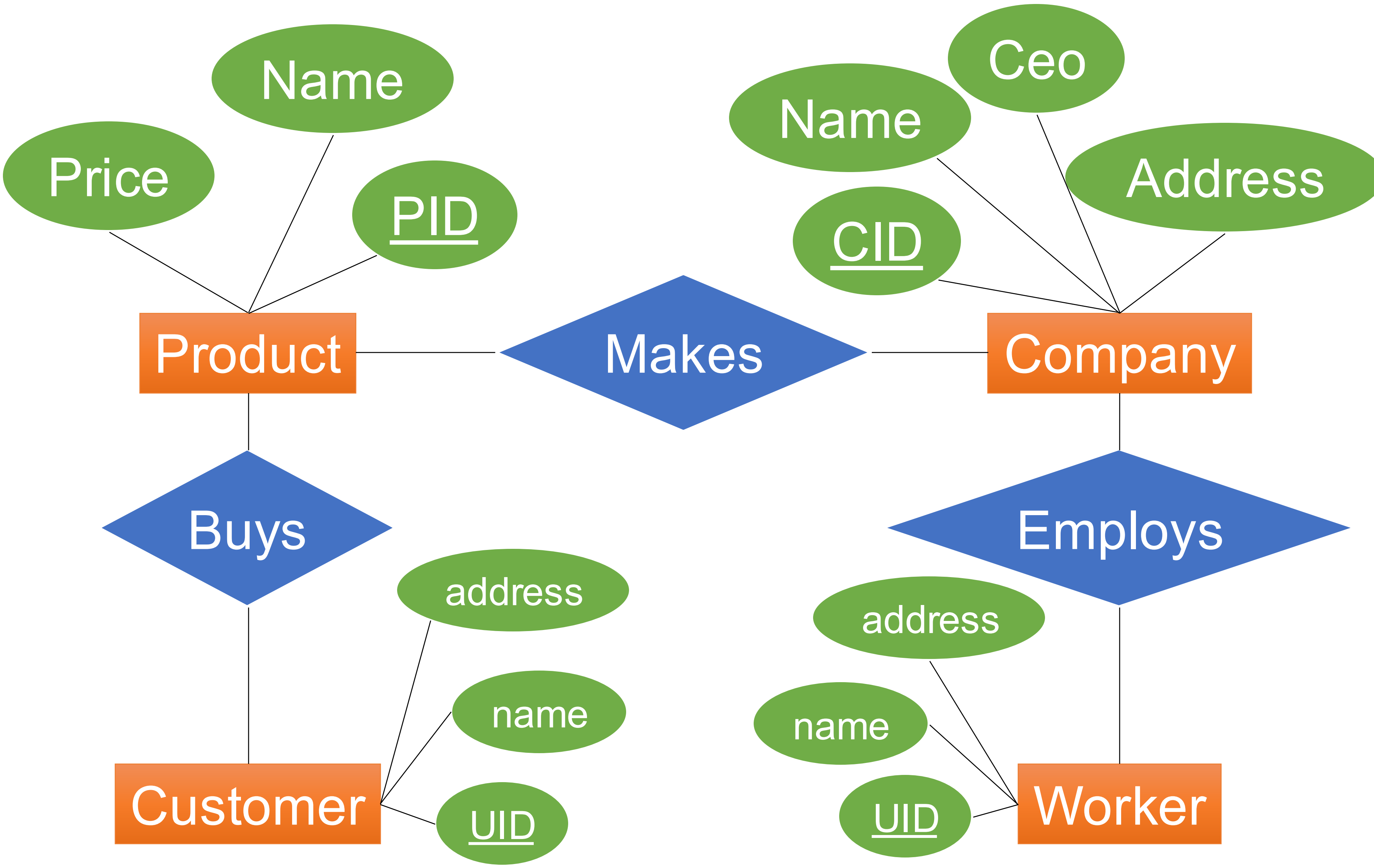
Example: Refining the Schema



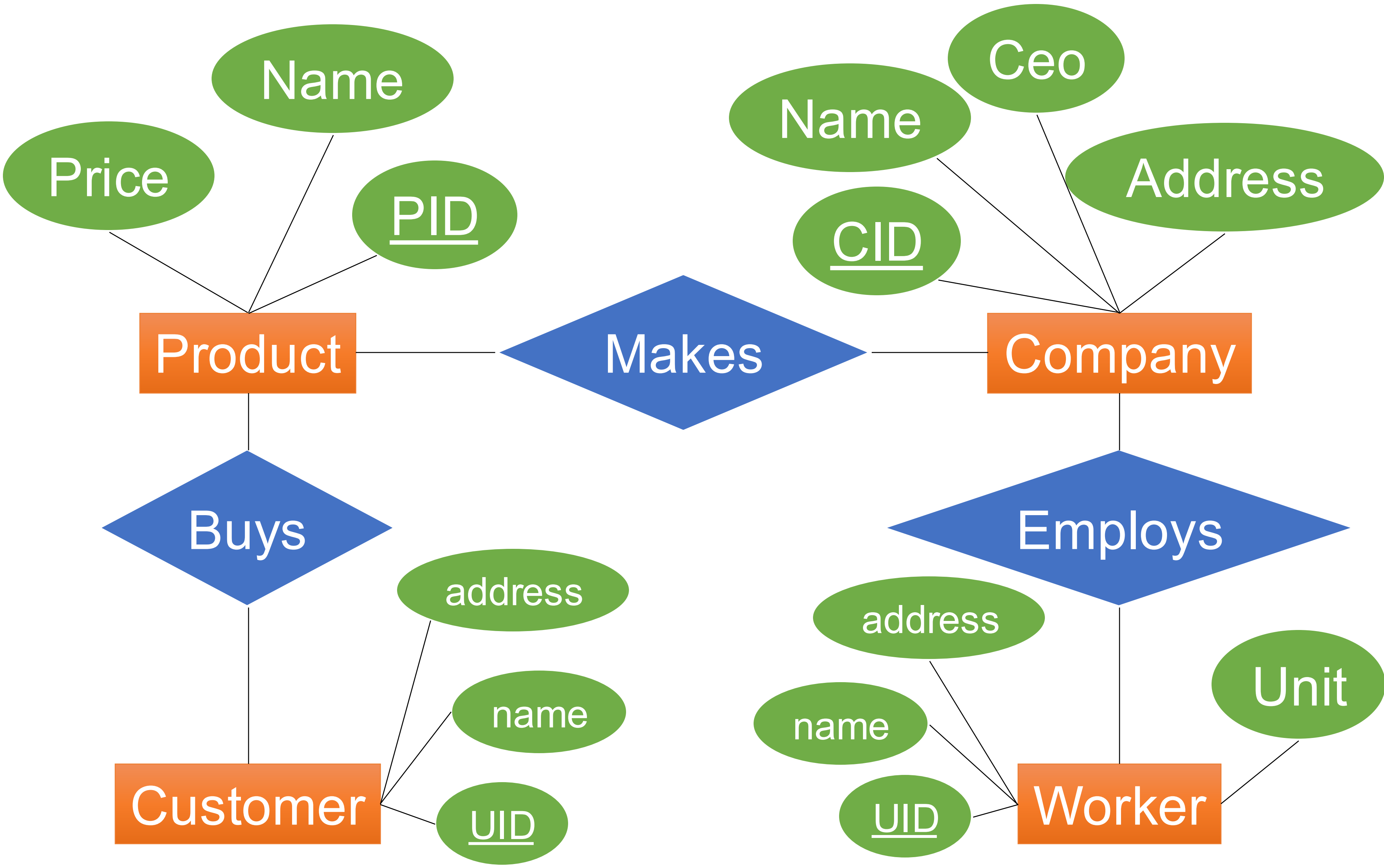
Example: Refining the Schema



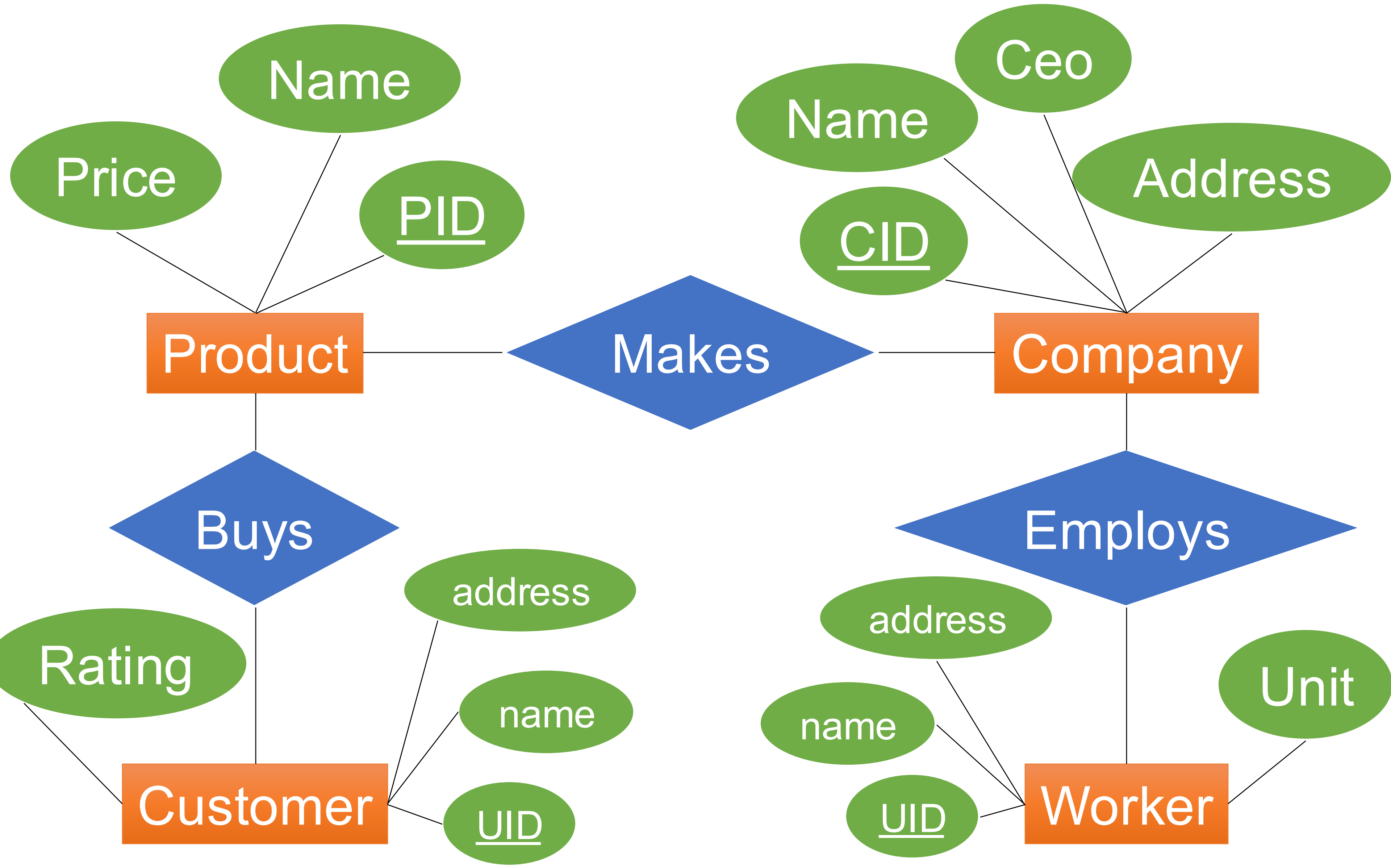
Example: Refining the Schema



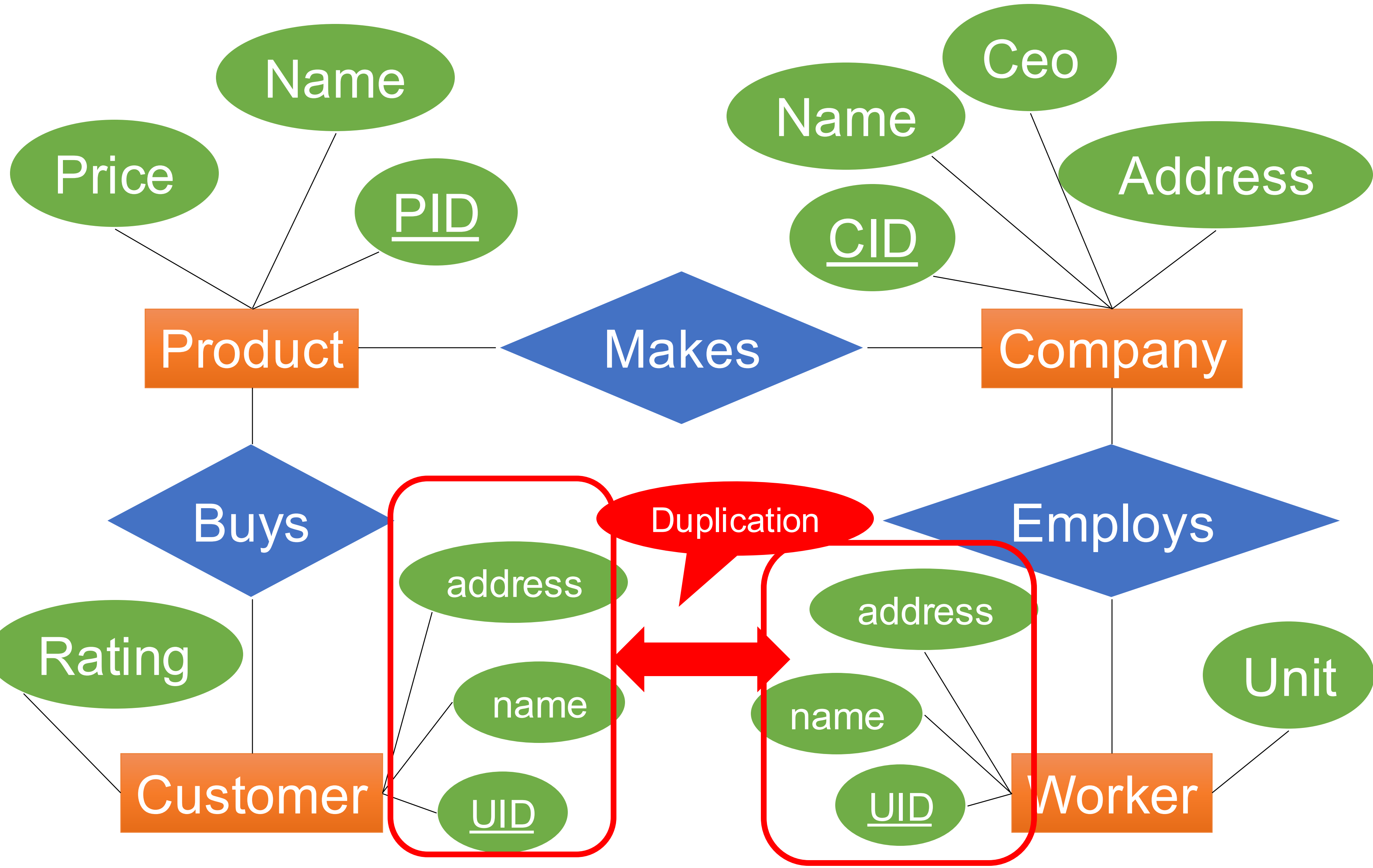
Example: Refining the Schema



Example: Refining the Schema



Example: Refining the Schema



Example: Refining the Schema