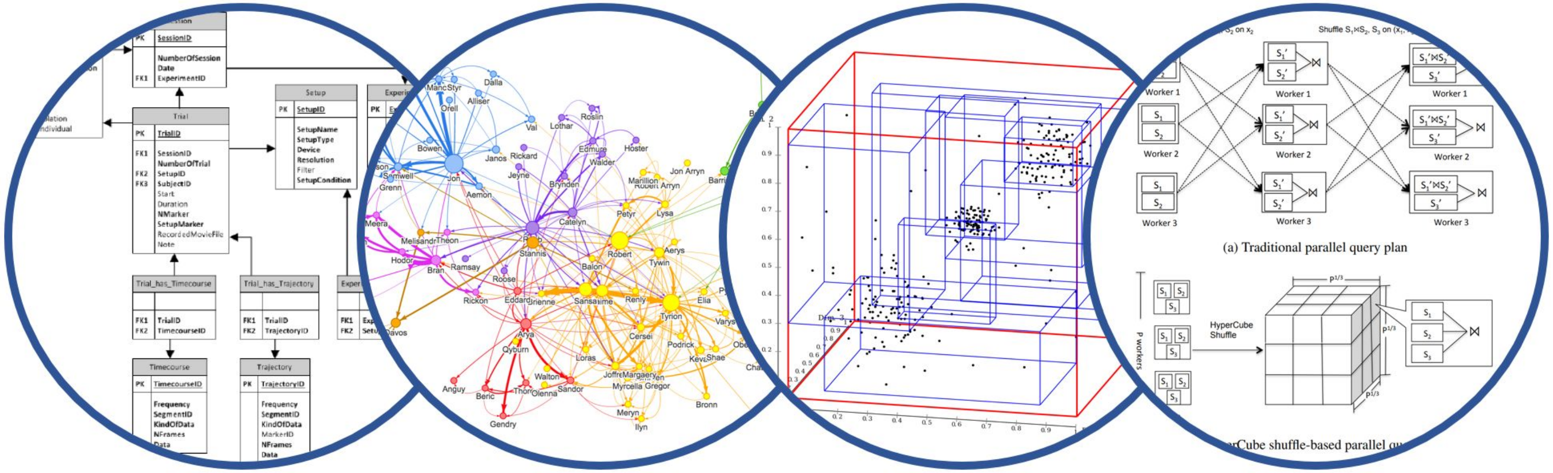


Introduction to Data Management

Joining Tables

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

Outline

- A Little Extra SQL
- Key Recap
- Introduce Foreign Keys
- Introduce Joins
- Demo in Sqlite

A Little Extra SQL

- ORDER BY – Orders result tuples by specified attributes (default ascending)

```
SELECT P.UserID, P.Name, P.Salary
  FROM Payroll AS P
ORDER BY P.Name ASC
```

Default

```
SELECT P.UserID, P.Name, P.Salary
  FROM Payroll AS P
ORDER BY P.Salary DESC
```

A Little Extra SQL

- ORDER BY – Orders result tuples by specified attributes (default ascending)

```
SELECT P.UserID, P.Name, P.Salary
  FROM Payroll AS P
ORDER BY P.Salary, P.Name;
```

UserID	Name	Salary
123	Jack	50000
345	Allison	50000
567	Magda	90000
789	Dan	100000

UserID	Name	Salary
345	Allison	50000
123	Jack	50000
567	Magda	90000
789	Dan	100000

A Little Extra SQL

- DISTINCT – Deduplicates result tuples

- Data exploration:
  “What are the possible jobs in this dataset?”

```
SELECT DISTINCT Job
  FROM Payroll;
```

Job
TA
Prof

A Little Extra SQL

- DISTINCT – Deduplicates result tuples

```
SELECT P.Job
  FROM Payroll AS P
 WHERE P.Salary > 70000;
```

Job
Prof
Prof

A Little Extra SQL

- DISTINCT – Deduplicates result tuples

```
SELECT P.Job
  FROM Payroll AS P
 WHERE P.Salary > 70000;
```

Job
Prof
Prof

```
SELECT DISTINCT P.Job
  FROM Payroll AS P
 WHERE P.Salary > 70000;
```

Job
Prof

Preview!

- Data exploration:

  “How many people are in this dataset?”

Preview!

- Data exploration:

“How many people are in this dataset?”

```
SELECT COUNT(*)
  FROM Payroll;
```

COUNT(*)
4

Preview!

- Data exploration:

“How many people are in this dataset?”

```
SELECT COUNT(*)AS num_people
  FROM Payroll;
```

num_people
4

to rename the column

Recap - Keys

Key

A **Key** is one or more attributes that **uniquely** identify a row.

Good candidate for a key

Definitely not a key

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

Recap - Keys

Unique Identifier

```
CREATE TABLE Payroll (
  UserID INT PRIMARY KEY,
  Name VARCHAR(100),
  Job VARCHAR(100),
  Salary INT);
```

Payroll(UserId, Name, Job, Salary)

Recap - Keys

```
CREATE TABLE Payroll (
  UserID INT,
  Name VARCHAR(100),
  Job VARCHAR(100),
  Salary INT,
  PRIMARY KEY (UserId);
```

Can also define the PK on a new line

Payroll(UserId, Name, Job, Salary)

Recap - Multi-Attribute Keys

Sometimes no single attribute is unique, but combinations of attributes are a unique key for the table.

```
CREATE TABLE Payroll (
  Name VARCHAR(100),
  Job VARCHAR(100),
  Salary INT,
  PRIMARY KEY (Name, Job));
```

Must use the PK definition on a new line for multi-attribute keys

Here the combination of Name and Job are unique
e.g. only one ”Eden, Professor”
but some “Eden, TA” or “Ryan, Professor” can exist

Payroll(Name, Job, Salary)

Foreign Keys

- Databases can hold multiple tables
- How do we capture relationships *between* tables?

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

Foreign Keys

- Databases can hold multiple tables
- How do we capture relationships *between* tables?

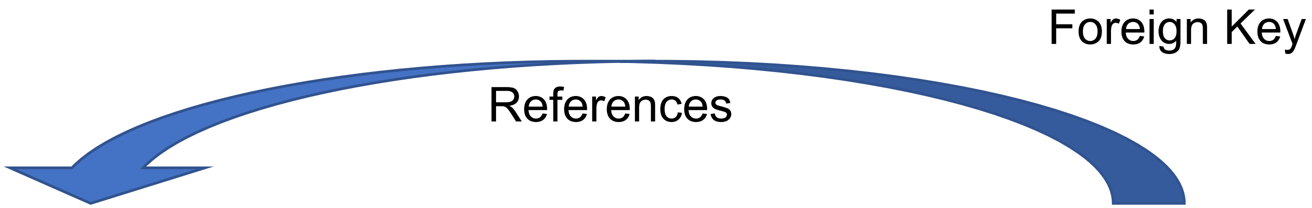
UserID	Name	Job	Salary
123	Jack	TA	50000
345	Allison	TA	60000
567	Magda	Prof	90000
789	Dan	Prof	100000

Foreign Key
UserID

UserID	Car
123	Charger
567	Civic
567	Pinto

Foreign Keys

- Databases can hold multiple tables
- How do we capture relationships *between* tables?



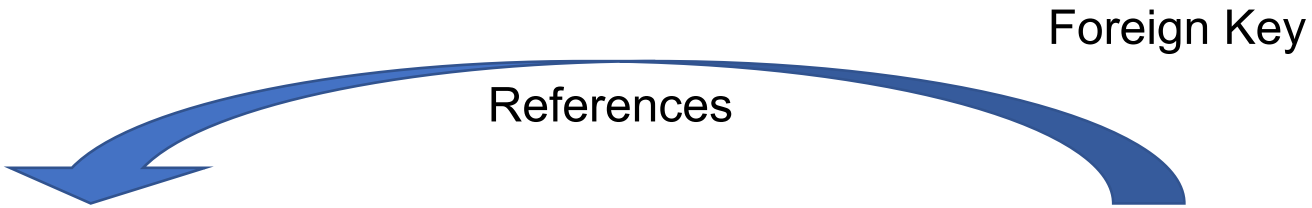
UserID	Name	Job	Salary
123	Jack	TA	50000
345	Allison	TA	60000
567	Magda	Prof	90000
789	Dan	Prof	100000

UserID	Car
123	Charger
567	Civic
567	Pinto

Foreign Keys

Foreign Key

A **Foreign Key** is one or more attributes that uniquely identify a row in *another table*.



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

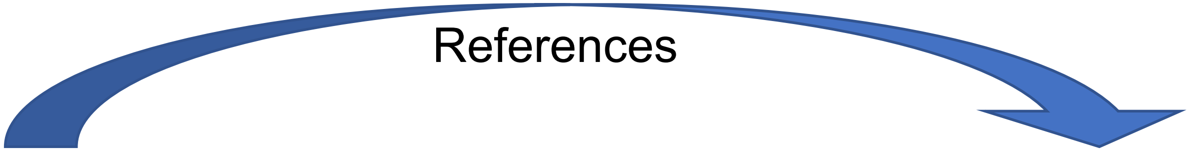
UserID	Car
123	Charger
567	Civic
567	Pinto

Foreign Keys

Foreign Key

A **Foreign Key** is one or more attributes that uniquely identify a row in *another table*.

Is this valid?



References

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

UserID	Car
123	Charger
567	Civic
567	Pinto

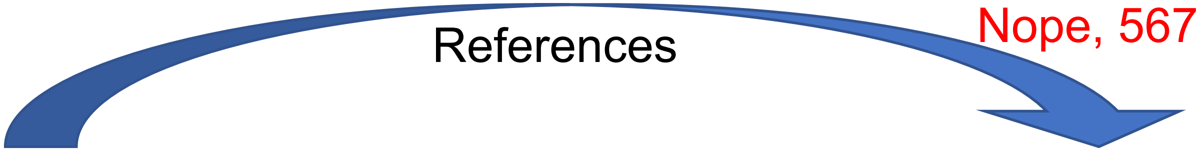
Foreign Keys

Foreign Key

A **Foreign Key** is one or more attributes that uniquely identify a row in *another table*.

Is this valid?

Nope, 567 is not unique in Regist table



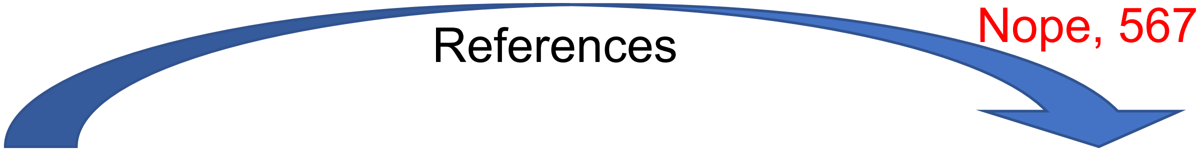
UserID	Name	Job	Salary
123	Jack	TA	50000
345	Allison	TA	60000
567	Magda	Prof	90000
789	Dan	Prof	100000

UserID	Car
123	Charger
567	Civic
567	Pinto

Foreign Keys

Foreign Key

A **Foreign Key** is one or more attributes that uniquely identify a row in *another table*.



Is this valid?

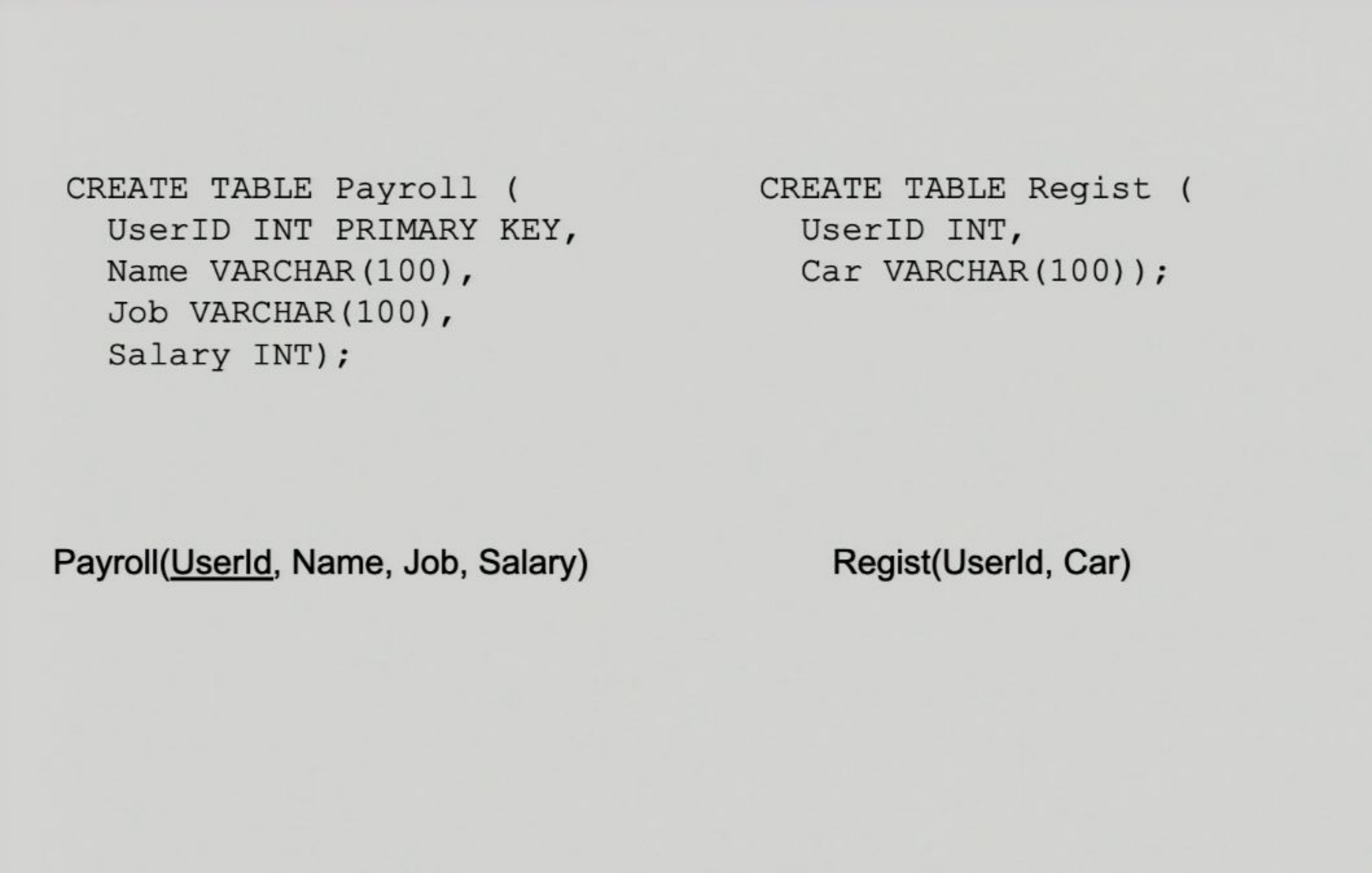
Nope, 567 is not unique in Regist table

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

UserID	Car
123	Charger
567	Civic
567	Pinto

Foreign keys must reference a unique attribute (or set of attributes), almost always the primary key

Foreign Keys



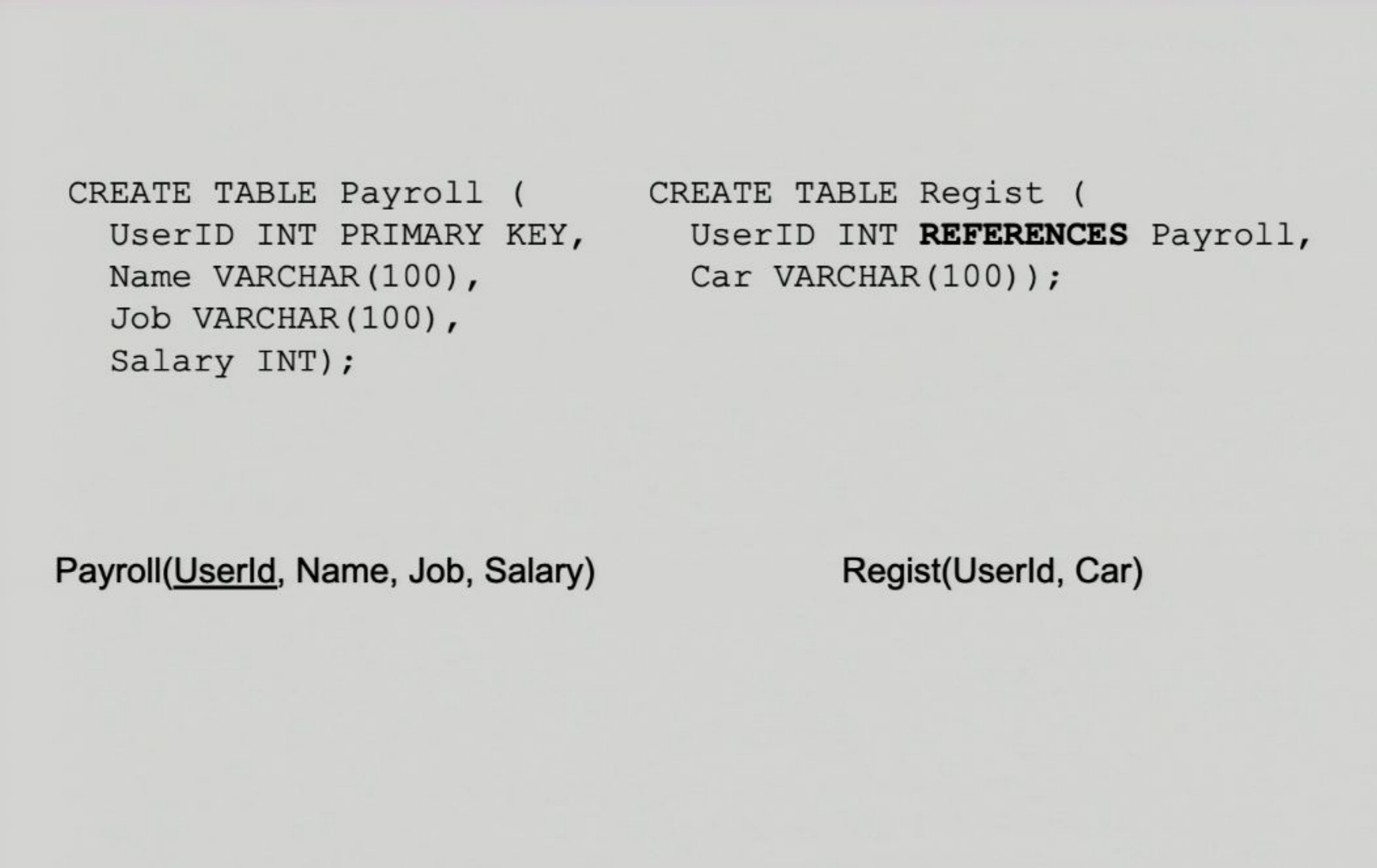
```
CREATE TABLE Payroll (
  UserID INT PRIMARY KEY,
  Name VARCHAR(100),
  Job VARCHAR(100),
  Salary INT);
```

```
CREATE TABLE Regist (
  UserID INT,
  Car VARCHAR(100));
```

Payroll(UserId, Name, Job, Salary)

Regist(UserId, Car)

Foreign Keys



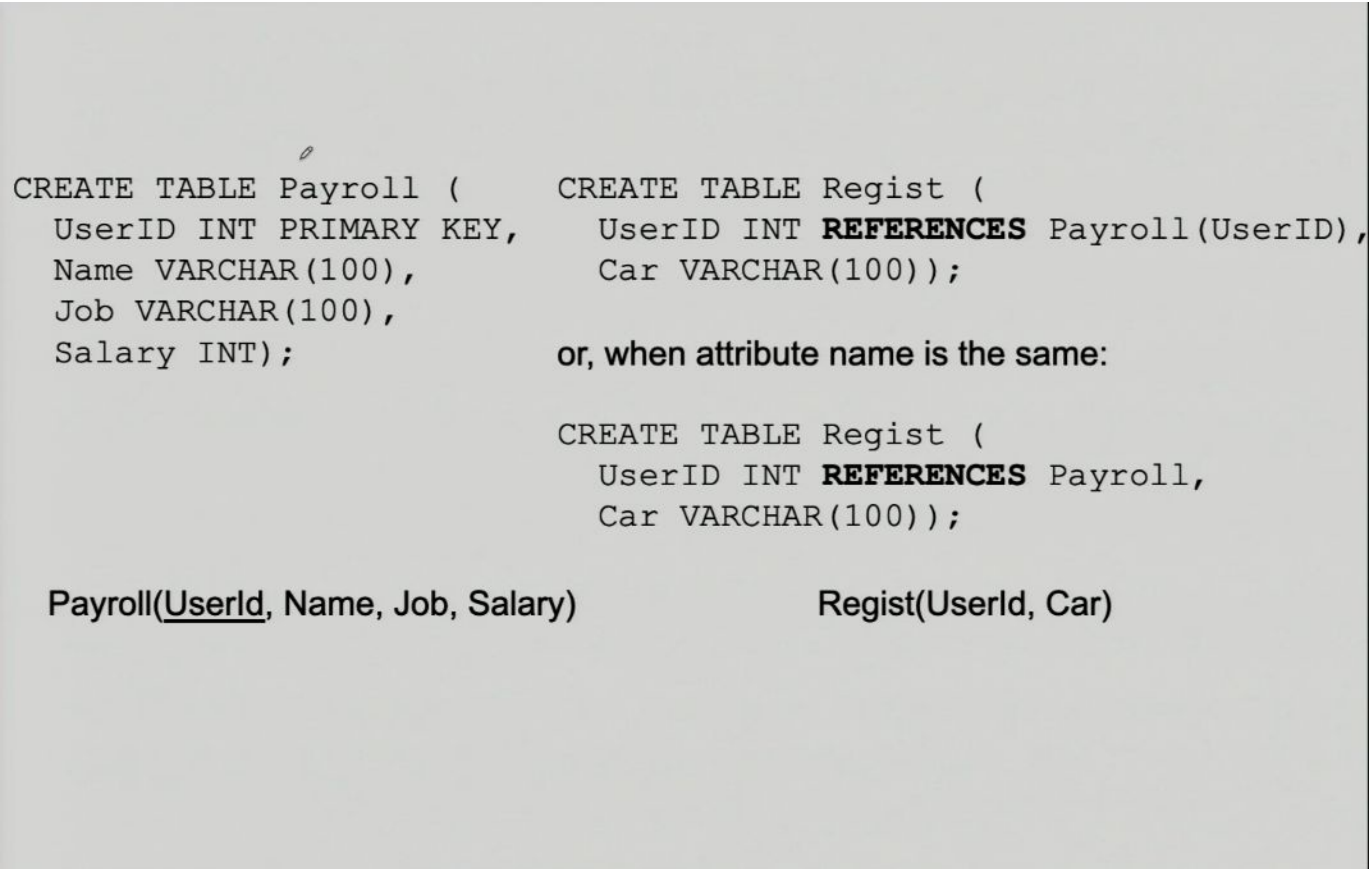
```
CREATE TABLE Payroll (
  UserID INT PRIMARY KEY,
  Name VARCHAR(100),
  Job VARCHAR(100),
  Salary INT);
```

```
CREATE TABLE Regist (
  UserID INT REFERENCES Payroll,
  Car VARCHAR(100));
```

Payroll(UserId, Name, Job, Salary)

Regist(UserId, Car)

Foreign Keys



```
CREATE TABLE Payroll (
  UserID INT PRIMARY KEY,
  Name VARCHAR(100),
  Job VARCHAR(100),
  Salary INT);
```

```
CREATE TABLE Regist (
  UserID INT REFERENCES Payroll(UserID),
  Car VARCHAR(100));
```

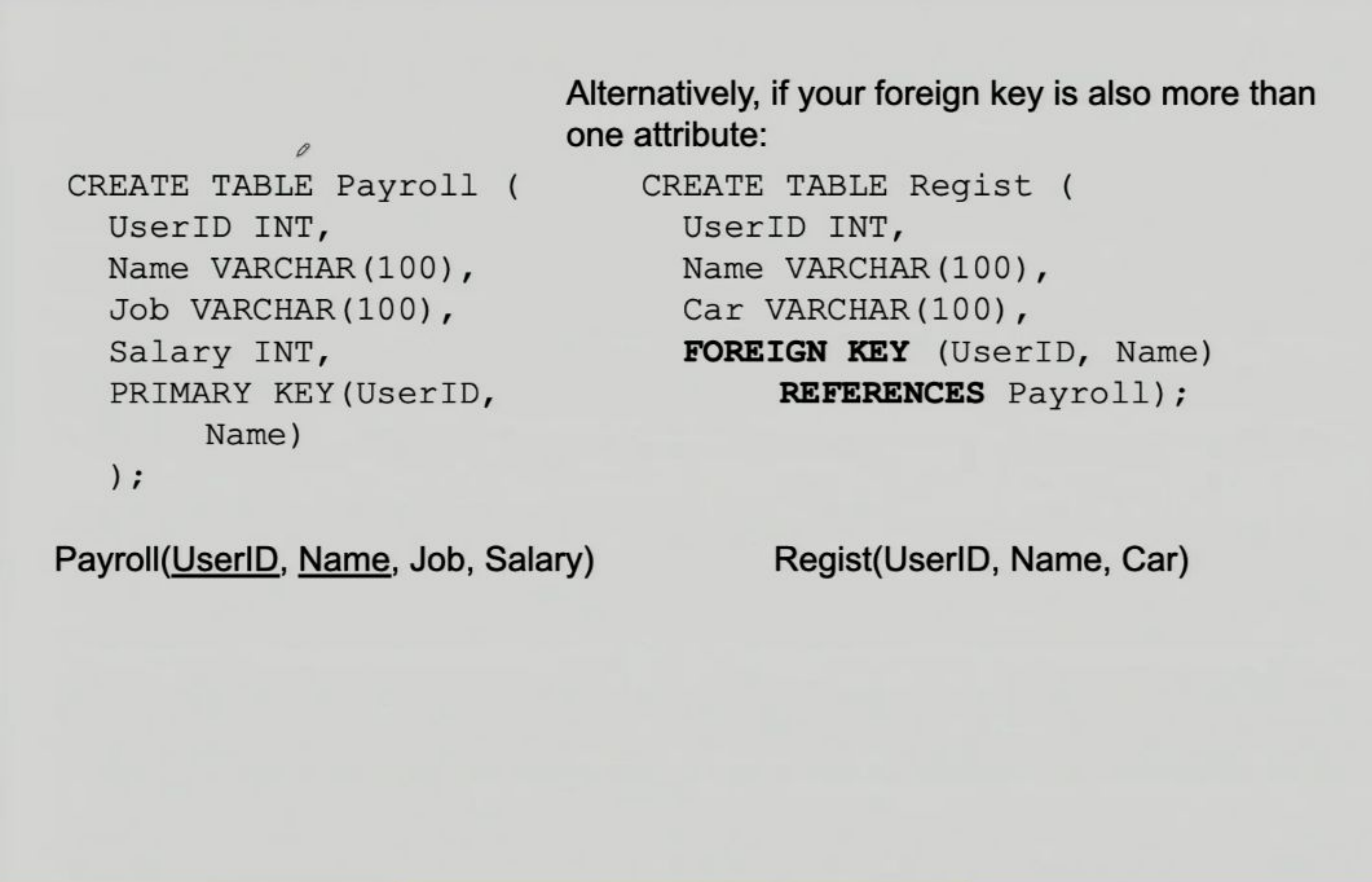
or, when attribute name is the same:

```
CREATE TABLE Regist (
  UserID INT REFERENCES Payroll,
  Car VARCHAR(100));
```

Payroll(UserId, Name, Job, Salary)

Regist(UserId, Car)

Foreign Keys



Alternatively, if your foreign key is also more than one attribute:

```
CREATE TABLE Payroll (
  UserID INT,
  Name VARCHAR(100),
  Job VARCHAR(100),
  Salary INT,
  PRIMARY KEY(UserID,
       Name)
  );
```

```
CREATE TABLE Regist (
  UserID INT,
  Name VARCHAR(100),
  Car VARCHAR(100),
  FOREIGN KEY (UserID, Name)
       REFERENCES Payroll);
```

Payroll(UserID, Name, Job, Salary)

Regist(UserID, Name, Car)

Joins

- Foreign keys are able to *describe* a relationship between tables
- Joins are able to realize combinations of data
- Joins do **not** require a foreign key, but often they go together

Inner Joins

- Bread and butter of SQL queries
  - "Inner join" is often interchangeable with just "join"

- Inner Join syntax:

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

Join Predicate

```
SELECT P.Name, R.Car
  FROM Payroll AS P JOIN Regist AS R ON P.UserID = R.UserID;
```

Nested-Loop Semantics

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

UserID	Car
123	Charger
567	Civic
567	Pinto

```
SELECT P.Name, R.Car
  FROM Payroll AS P JOIN Regist AS R
       ON P.UserID = R.UserID;
```

How do we algorithmically get our results?

Name	Car
Jack	Charger
Magda	Civic
Magda	Pinto

Nested-Loop Semantics

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

UserID	Car
123	Charger
567	Civic
567	Pinto

```
SELECT P.Name, R.Car
  FROM Payroll AS P JOIN Regist     R
       ON P.UserID = R.UserID;
```

How do we algorithmically get our results?

Name	Car
Jack	Charger
Magda	Civic
Magda	Pinto

Compare every possible combination and filter the results that match

Nested-Loop Semantics

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

UserID	Car
123	Charger
567	Civic
567	Pinto

```
SELECT P.Name, R.Car
  FROM Payroll AS P JOIN Regist AS R
       ON P.UserID = R.UserID;
```

```
for each row1 in Payroll:
   for each row2 in Regist:
      if (row1.UserID = row2.UserID):
         output (row1.Name, row2.Car)
```

Nested-Loop Semantics

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

UserID	Car
123	Charger
567	Civic
567	Pinto

Name	Car

```
for each row1 in Payroll:
    for each row2 in Regist:
        if (row1.UserID = row2.UserID):
            output (row1.Name, row2.Car)
```

Nested-Loop Semantics

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

UserID	Car
123	Charger
567	Civic
567	Pinto

Name	Car
Jack	Charger

```
for each row1 in Payroll:
    for each row2 in Regist:
        if (row1.UserID = row2.UserID):
            output (row1.Name, row2.Car)
```

Nested-Loop Semantics

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

UserID	Car
123	Charger
567	Civic
567	Pinto

Name	Car
Jack	Charger

```
for each row1 in Payroll:
    for each row2 in Regist:
        if (row1.UserID = row2.UserID):
            output (row1.Name, row2.Car)
```

Nested-Loop Semantics

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

UserID	Car
123	Charger
567	Civic
567	Pinto

Name	Car
Jack	Charger

```
for each row1 in Payroll:
    for each row2 in Regist:
        if (row1.UserID = row2.UserID):
            output (row1.Name, row2.Car)
```

Nested-Loop Semantics

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

UserID	Car
123	Charger
567	Civic
567	Pinto

Name	Car
Jack	Charger

```
for each row1 in Payroll:
    for each row2 in Regist:
        if (row1.UserID = row2.UserID):
            output (row1.Name, row2.Car)
```

Nested-Loop Semantics

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

UserID	Car
123	Charger
567	Civic
567	Pinto

Name	Car
Jack	Charger

```
for each row1 in Payroll:
    for each row2 in Regist:
        if (row1.UserID = row2.UserID):
            output (row1.Name, row2.Car)
```

Nested-Loop Semantics

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

UserID	Car
123	Charger
567	Civic
567	Pinto

Name	Car
Jack	Charger

```
for each row1 in Payroll:
    for each row2 in Regist:
        if (row1.UserID = row2.UserID):
            output (row1.Name, row2.Car)
```

Nested-Loop Semantics

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

UserID	Car
123	Charger
567	Civic
567	Pinto

Name	Car
Jack	Charger

```
for each row1 in Payroll:
    for each row2 in Regist:
        if (row1.UserID = row2.UserID):
            output (row1.Name, row2.Car)
```

Nested-Loop Semantics

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

UserID	Car
123	Charger
567	Civic
567	Pinto

Name	Car
Jack	Charger

```
for each row1 in Payroll:
    for each row2 in Regist:
        if (row1.UserID = row2.UserID):
            output (row1.Name, row2.Car)
```

Nested-Loop Semantics

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

UserID	Car
123	Charger
567	Civic
567	Pinto

Name	Car
Jack	Charger
Magda	Civic

```
for each row1 in Payroll:
    for each row2 in Regist:
        if (row1.UserID = row2.UserID):
            output (row1.Name, row2.Car)
```

Nested-Loop Semantics

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

UserID	Car
123	Charger
567	Civic
567	Pinto

Name	Car
Jack	Charger
Magda	Civic

```
for each row1 in Payroll:
    for each row2 in Regist:
        if (row1.UserID = row2.UserID):
            output (row1.Name, row2.Car)
```

Nested-Loop Semantics

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

UserID	Car
123	Charger
567	Civic
567	Pinto

Name	Car
Jack	Charger
Magda	Civic
Magda	Pinto

```
for each row1 in Payroll:
    for each row2 in Regist:
        if (row1.UserID = row2.UserID):
            output (row1.Name, row2.Car)
```

Nested-Loop Semantics

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

UserID	Car
123	Charger
567	Civic
567	Pinto

Name	Car
Jack	Charger
Magda	Civic
Magda	Pinto

```
for each row1 in Payroll:
    for each row2 in Regist:
        if (row1.UserID = row2.UserID):
            output (row1.Name, row2.Car)
```

Nested-Loop Semantics

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

UserID	Car
123	Charger
567	Civic
567	Pinto

Name	Car
Jack	Charger
Magda	Civic
Magda	Pinto

```
for each row1 in Payroll:
    for each row2 in Regist:
        if (row1.UserID = row2.UserID):
            output (row1.Name, row2.Car)
```

Nested-Loop Semantics

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

UserID	Car
123	Charger
567	Civic
567	Pinto

Name	Car
Jack	Charger
Magda	Civic
Magda	Pinto

```
for each row1 in Payroll:
    for each row2 in Regist:
        if (row1.UserID = row2.UserID):
            output (row1.Name, row2.Car)
```

Nested-Loop Semantics

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

UserID	Car
123	Charger
567	Civic
567	Pinto

Name	Car
Jack	Charger
Magda	Civic
Magda	Pinto

```
for each row1 in Payroll:
    for each row2 in Regist:
        if (row1.UserID = row2.UserID):
            output (row1.Name, row2.Car)
```

Inner Joins

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

UserID	Car
123	Charger
567	Civic
567	Pinto

Explicit

```
SELECT P.Name, R.Car
  FROM Payroll AS P JOIN Regist AS R
       ON P.UserID = R.UserID;
```

Implicit

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

Both of them have the same meaning (for inner joins)

Inner Joins

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

- What if we have no join predicate?

```
SELECT P.Name, R.Car
  FROM Payroll AS P, Regist AS R

  for each row1 in Payroll:
      for each row2 in Regist:
             output (row1.Name, row2.Car)
```

- Output every possible pair: “Cross product”

Outer Joins

Now I want to include everyone, even if they don’t drive.

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

UserID	Car
123	Charger
567	Civic
567	Pinto

Outer Joins

Now I want to include everyone, even if they don’t drive.

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

UserID	Car
123	Charger
567	Civic
567	Pinto

```
SELECT P.Name, R.Car
  FROM Payroll AS P LEFT OUTER JOIN Regist AS R
       ON P.UserID = R.UserID;
```

Outer Joins

Now I want to include everyone, even if they don’t drive.

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

UserID	Car
123	Charger
567	Civic
567	Pinto

```
SELECT P.Name, R.Car
  FROM Payroll AS P LEFT OUTER JOIN Regist AS R
       ON P.UserID = R.UserID;
```

Outer Joins

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

UserID	Car
123	Charger
567	Civic
567	Pinto

Name	Car
Jack	Charger
Allison	NULL
Magda	Civic
Magda	Pinto
Dan	NULL

Outer Joins

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

UserID	Car
123	Charger
567	Civic
567	Pinto

Name	Car
Jack	Charger
Allison	NULL
Magda	Civic
Magda	Pinto
Dan	NULL

NULL is a value placeholder. Depending on context, it may mean unknown, not applicable, etc.

Outer Joins

- LEFT OUTER JOIN
  - All rows in left table are preserved
- RIGHT OUTER JOIN
  - All rows in right table are preserved
- FULL OUTER JOIN
  - All rows are preserved

Self-Joins

Find all people who drive a Civic and Pinto

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

UserID	Car
123	Charger
567	Civic
567	Pinto

```
SELECT P.Name, R.Car
  FROM Payroll AS P, Regist AS R
 WHERE P.UserID = R.UserID AND
       R.Car = ‘Civic’;
```

Self-Joins

Find all people who drive a Civic and Pinto

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

UserID	Car
123	Charger
567	Civic
567	Pinto

```
SELECT P.Name, R.Car
  FROM Payroll AS P, Regist AS R
 WHERE P.UserID = R.UserID AND
       R.Car = 'Civic' AND
       R.Car = 'Pinto';
```

Will this work?

Self-Joins

Find all people who drive a Civic and Pinto

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

UserID	Car
123	Charger
567	Civic
567	Pinto

```
SELECT P.Name, R.Car
  FROM Payroll AS P, Regist AS R
 WHERE P.UserID = R.UserID AND
       R.Car = 'Civic' AND
       R.Car = 'Pinto';
```

Will this work?

Nope, empty set is returned

Self-Joins

Find all people who drive a Civic and Pinto

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

UserID	Car
123	Charger
567	Civic
567	Pinto
789	Pinto

```
SELECT P.Name, R.Car
  FROM Payroll AS P, Regist AS R
 WHERE P.UserID = R.UserID AND
       R.Car = ‘Civic’ AND
       R.Car = ‘Pinto’;
```

Discuss with the people around you how you would solve this.

Self-Joins

Find all people who drive a Civic and Pinto

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

UserID	Car
123	Charger
567	Civic
567	Pinto

```
SELECT P.Name, R1.Car
  FROM Payroll AS P, Regist AS R1, Regist AS R2
 WHERE P.UserID = R1.UserID AND
       P.UserID = R2.UserID AND
       R1.Car = ‘Civic’ AND
       R2.Car = ‘Pinto’;
```

Self-Joins

Find all people who drive a Civic and Pinto

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

UserID	Car
123	Charger
567	Civic
567	Pinto

All pairs of cars a person can drive

```
SELECT P.Name, R1.Car
  FROM Payroll AS P, Regist AS R1, Regist AS R2
 WHERE P.UserID = R1.UserID AND
       P.UserID = R2.UserID AND
       R1.Car = 'Civic' AND
       R2.Car = 'Pinto';
```

Self-Joins

- When a relation occurs twice in the FROM clause we call it a self-join;

- If we have a self-join, we must use tuple variables (aka table aliases) (why?)

Self-Joins

- When a relation occurs twice in the FROM clause we call it a self-join;

- If we have a self-join, we must use tuple variables (aka table aliases)

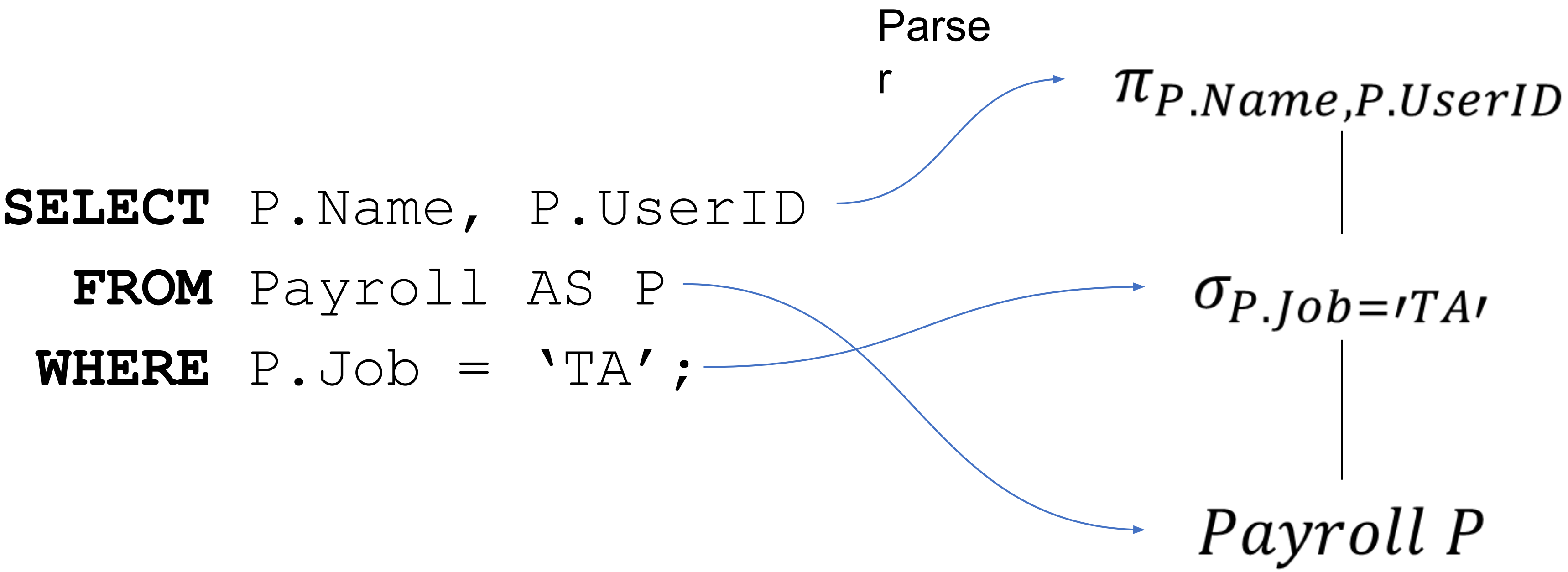
- Two different tables have an attribute of the same name

RA Equivalencies

So far we haven’t discussed equivalent RA trees.
But all joins can be parsed directly into a “join tree”

RA Equivalencies

So far we haven’t discussed equivalent RA trees.
But all joins can be parsed directly into a “join tree”



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

Parser

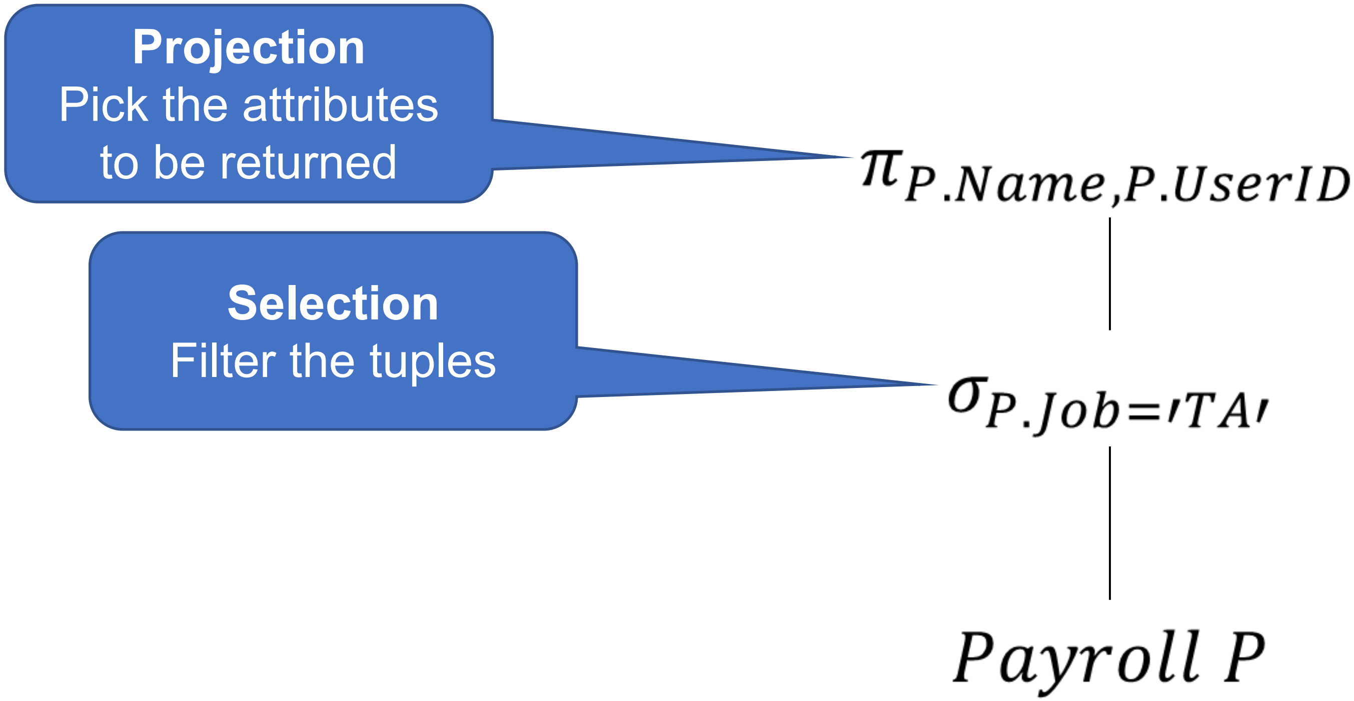
$$\pi_{P.Name,P.UserID}$$

$$\sigma_{P.Job='TA'}$$

$$Payroll\ P$$

RA Equivalencies

So far we haven’t discussed equivalent RA trees.
But all joins can be parsed directly into a “join tree”



Projection
Pick the attributes to be returned

Selection
Filter the tuples

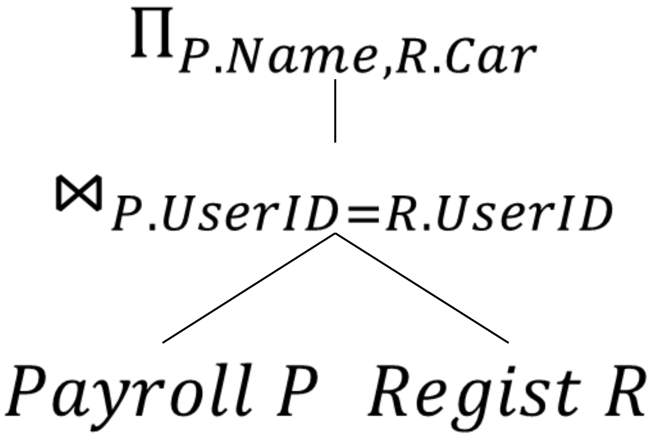
$\pi_{P.Name,P.UserID}$

$\sigma_{P.Job='TA'}$

$Payroll\ P$

RA Equivalencies

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



$\Pi_{P.Name,R.Car}$

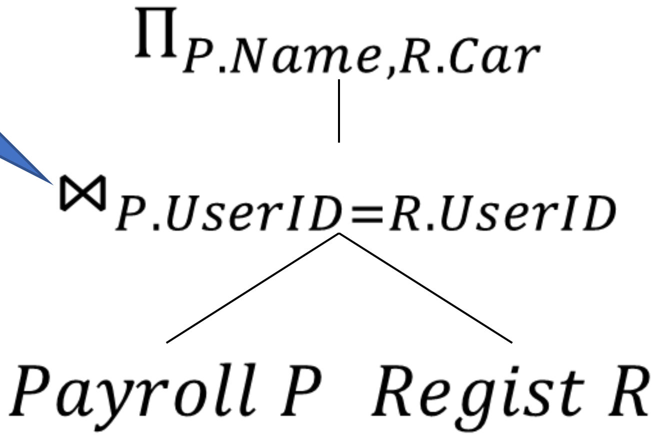
$\bowtie_{P.UserID=R.UserID}$

$Payroll\ P$ $Regist\ R$

RA Equivalencies

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

Join
Combine tuples on the provided predicate



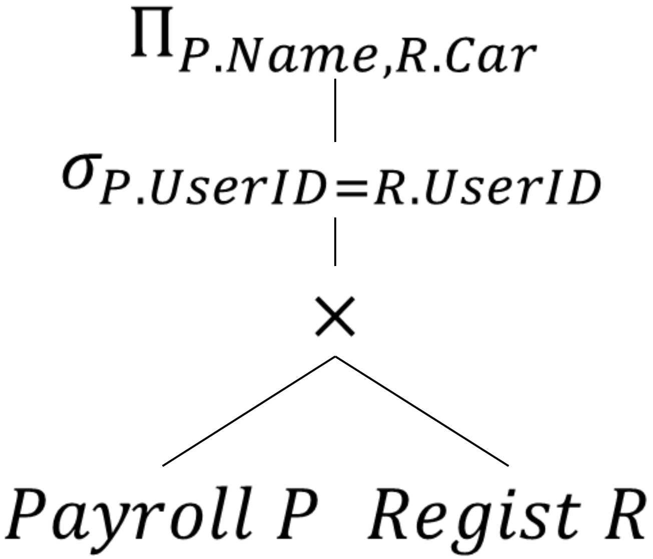
$\Pi_{P.Name,R.Car}$

$\bowtie_{P.UserID=R.UserID}$

$Payroll\ P$ $Regist\ R$

RA Equivalencies

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



$\Pi_{P.Name,R.Car}$

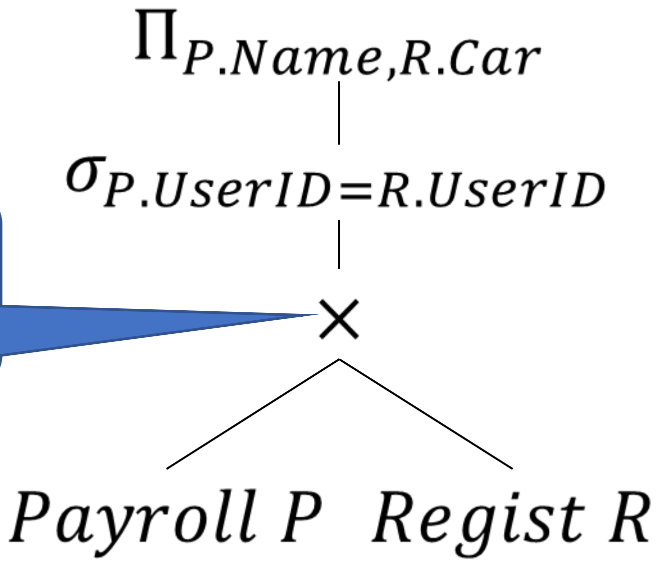
$\sigma_{P.UserID=R.UserID}$

$\times$

$Payroll\ P$ $\quad$ $Regist\ R$

RA Equivalencies

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



$\Pi_{P.Name,R.Car}$

$\sigma_{P.UserID=R.UserID}$

$\times$

$Payroll\ P$ $\quad$ $Regist\ R$

Cross Product
Same intuition from set theory

Takeaways

- We can describe relationships between tables with keys and foreign keys
- Different joining techniques can be used to achieve particular goals
- Our SQL toolbox is growing!
  - Not just reading and filtering data anymore
  - Starting to answer complex questions