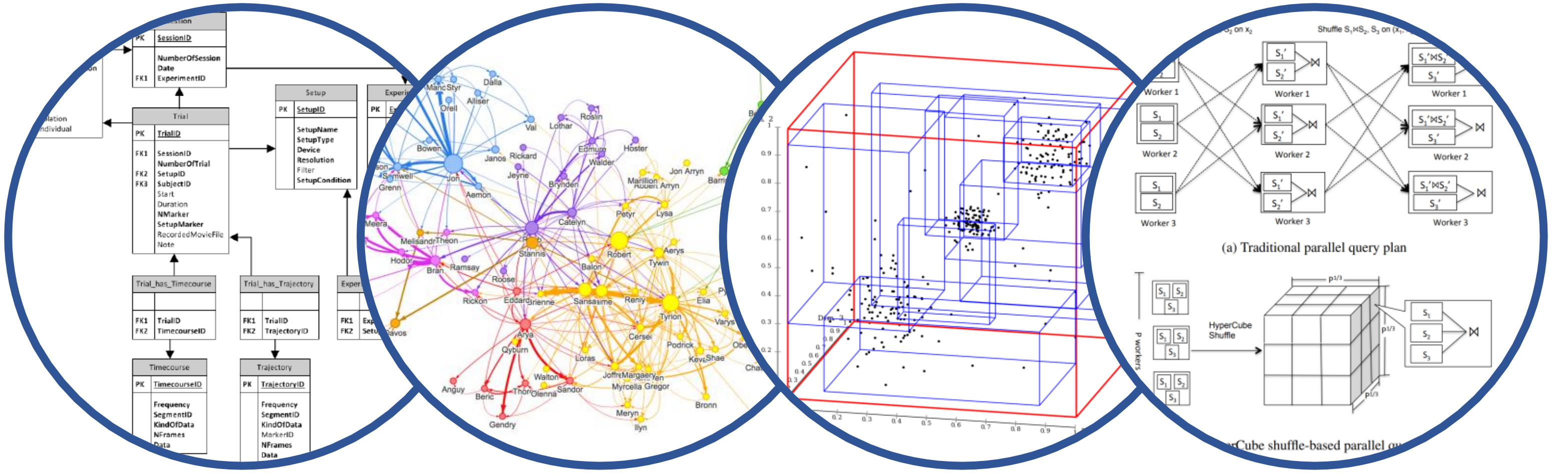
CSE 414, Winter 2020



Introduction to Data Management

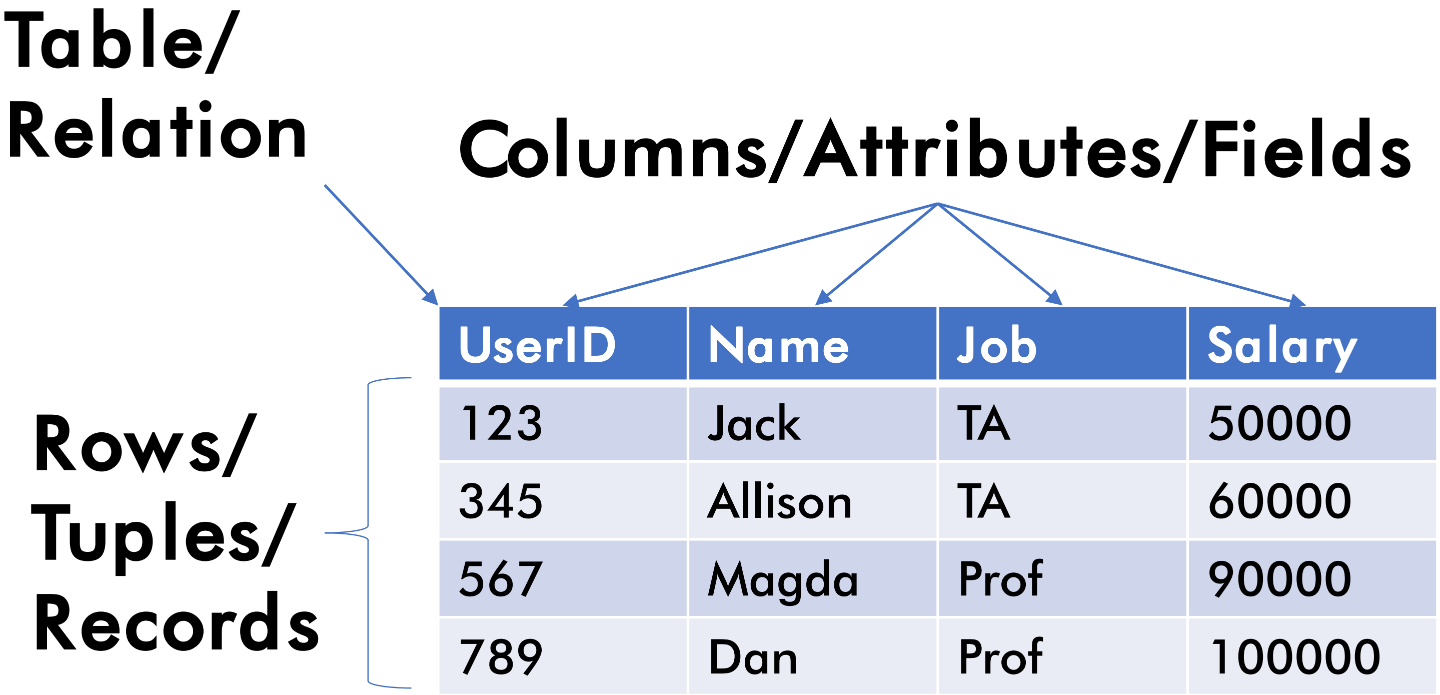
♫ *Join* together… right now… over me ♫

Shana Hutchison

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

Recap: The Relational Model

- Flat tables, static and typed attributes, etc.
  - "It's a spreadsheet with rules"



Table/ Relation

Columns/Attributes/Fields

Rows/ Tuples/ Records

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

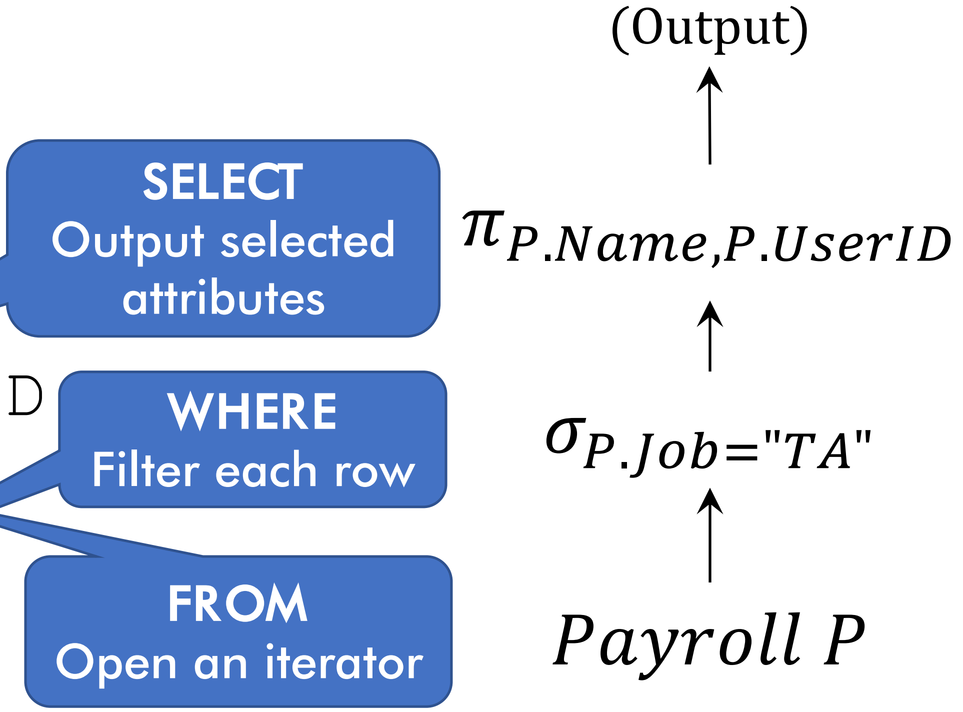
Recap: FROM-WHERE-SELECT

Payroll

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

```
for each row in P:
 if (row.Job == 'TA'):
  output (row.Name, row.UserID)
```

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



SELECT
Output selected attributes

WHERE
Filter each row

FROM
Open an iterator

(Output)

$\uparrow$

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

$\uparrow$

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

$\uparrow$

$Payroll\ P$

RA: FROM-WHERE-SELECT

Payroll

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

```
for each row in P:
 if (row.Job == 'TA'):
  output (row.Name, row.UserID)
```

1-arg **Projection** op (Greek "pi")
Project input onto list of attributes

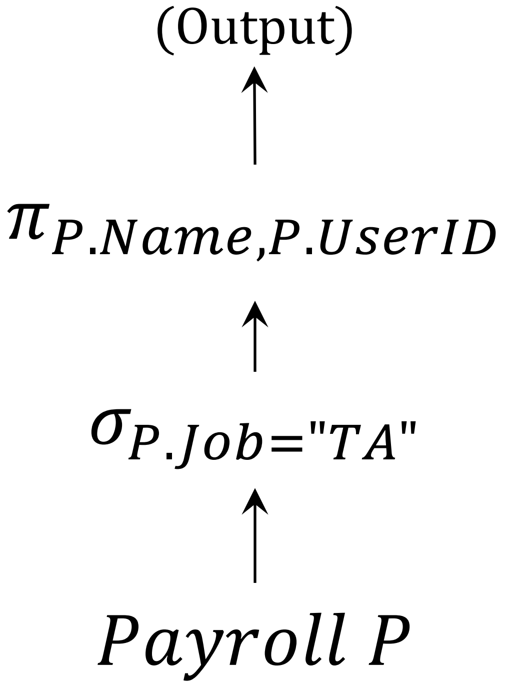
SELECT
Output selected attributes

1-arg **Selection** op (Greek "sigma")
Filter rows of input with condition

WHERE
Filter each row

0-arg **Input** op
Produce an input relation

FROM
Open an iterator



$$\begin{tikzcd}
\text{(Output)} \\
\pi_{P.Name,P.UserID} \arrow[u] \\
\sigma_{P.Job="TA"} \arrow[u] \\
Payroll\ P \arrow[u]
\end{tikzcd}$$

WHERE Practice

```
SELECT *
  FROM Payroll AS P
 WHERE P.UserID > 200;
```

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

↑

$$\sigma_{UserID > 200}$$

↑

Payroll

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

WHERE Practice

```
SELECT *
  FROM Payroll AS P
 WHERE P.UserID < 200
       OR P.Job = "Prof";
```

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

$$\uparrow$$

$$\sigma_{UserID > 200\ OR\ Job = \text{“}Prof\text{”}}$$

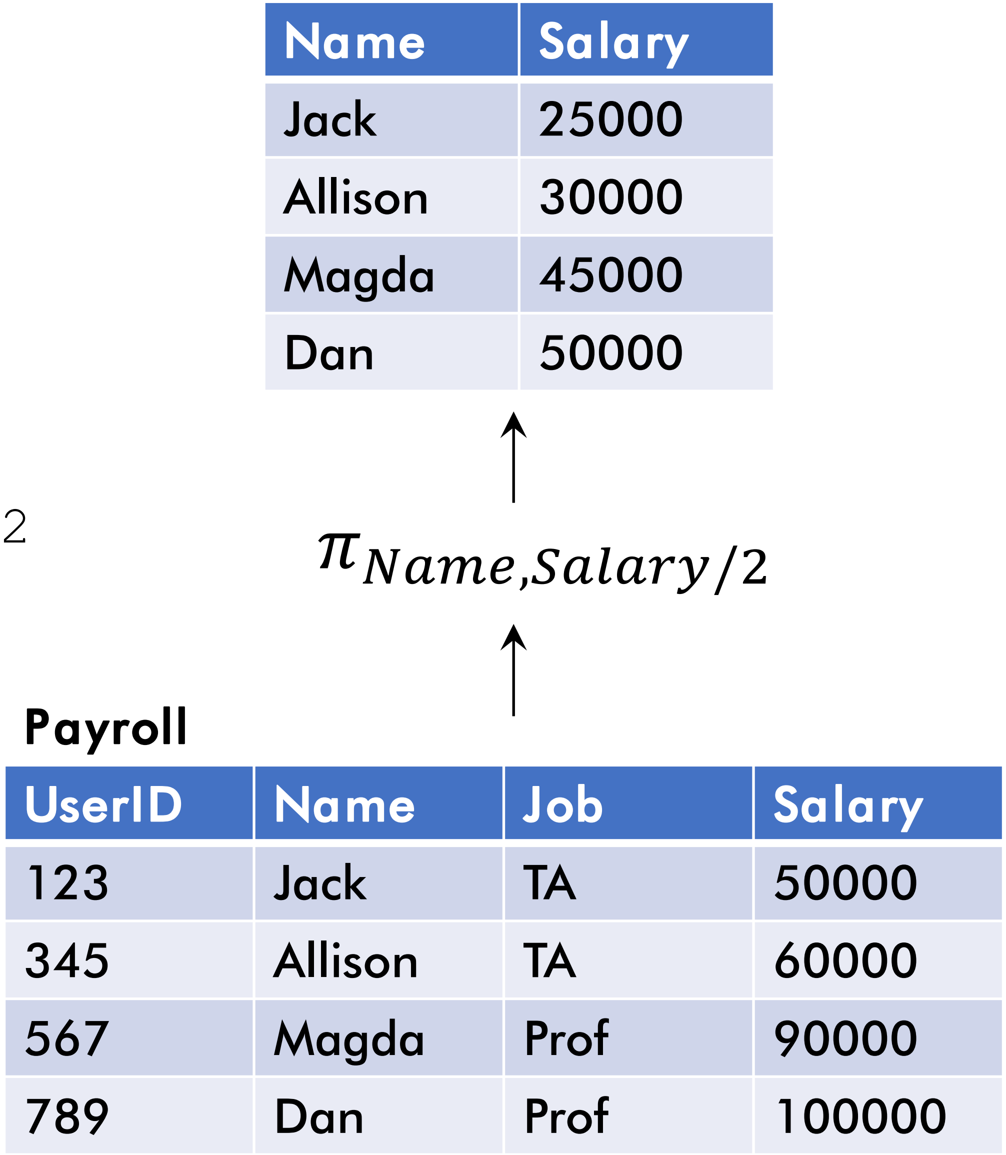
$$\uparrow$$

Payroll

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

SELECT Practice

```
SELECT P.Name, P.Salary/2
  FROM Payroll AS P;
```



Name	Salary
Jack	25000
Allison	30000
Magda	45000
Dan	50000

$$\uparrow$$

$$\pi_{Name,Salary/2}$$

$$\uparrow$$

Payroll

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

SELECT Practice

Duplicates!
May occur in output of an operator.
Never in an input table.

Job
TA
TA
Prof
Prof

↑

π_{Job}

↑

```
SELECT P.Job
  FROM Payroll AS P;
```

Payroll

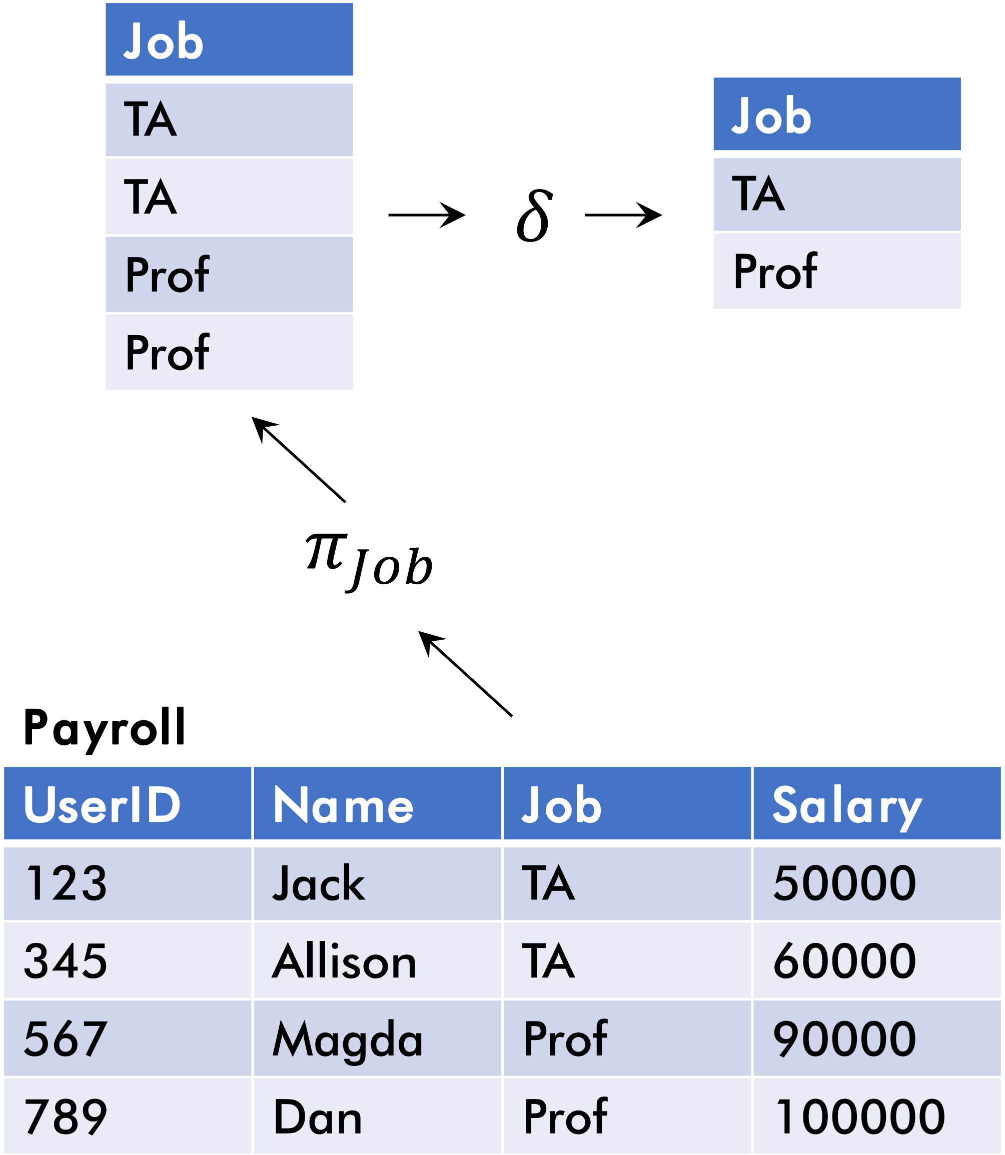
UserID	Name	Job	Salary
123	Jack	TA	50000
345	Allison	TA	60000
567	Magda	Prof	90000
789	Dan	Prof	100000

DISTINCT

1-arg Duplicate Elim op
(Greek "delta")

```
SELECT DISTINCT P.Job
  FROM Payroll AS P;
```

Next lecture:
DISTINCT is equivalent to GROUP BY with all attributes



Payroll

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

π_{Job}

Job
TA
TA
Prof
Prof

$\rightarrow \delta \rightarrow$

Job
TA
Prof

ORDER BY

1-arg Sort op
(Greek “tau”)

```
SELECT    *
  FROM    Payroll AS P
ORDER BY  Job, Name;
```

Use DESC for reverse order

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

↑

$$\tau_{Job,Name}$$

↑

Payroll

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

CREATE TABLE

```
CREATE TABLE Payroll (
      UserID int, Name text, Job text, Salary int);
INSERT INTO Payroll VALUES
      (123, 'Jack', 'TA', 50000),
      (123, 'Allison', 'TA', 60000),
      (123, 'Magda', 'Prof', 90000),
      (123, 'Dan', 'Prof', 100000);
```

Payroll

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

Outline

- FROM-WHERE-SELECT
- DISTINCT, ORDER BY, CREATE TABLE

Now you can do hw1! Next up:

- Keys
- Foreign Keys
- Joins + RA

Keys

Key

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

Payroll

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

Keys

Key

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

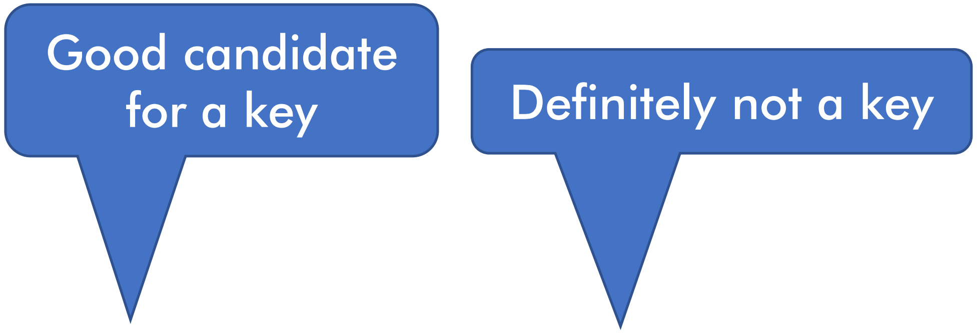
Definitely not a key

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

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

Keys

Key

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

Is this a good candidate for a key?

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

Keys

Key

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

Is this a good candidate for a key?

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

Keys

Key

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

Is this a good candidate for a key?

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

Keys

Key

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

Data comes from the real world
so models ought to reflect that

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

Keys

```
CREATE TABLE Payroll (
  UserID INT,
  Name TEXT,
  Job TEXT,
  Salary INT);
```

Payroll(UserId, Name, Job, Salary)

Keys

Unique Identifier

```
CREATE TABLE Payroll (
  UserID INT,
  Name TEXT,
  Job TEXT,
  Salary INT);
```

Payroll(UserId, Name, Job, Salary)

Keys

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

Unique Identifier

Payroll(<u>UserId</u>, Name, Job, Salary)

Keys

```
CREATE TABLE Payroll (
  UserID INT,
  Name TEXT,
  Job TEXT,
  Salary INT);
```

Suppose the set of attributes {Name, Job, Salary} is a key

Payroll(UserId, Name, Job, Salary)

Keys

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

Suppose the set of attributes {Name, Job, Salary} is a key

Payroll(UserId, Name, Job, Salary)

Foreign Keys

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

Payroll

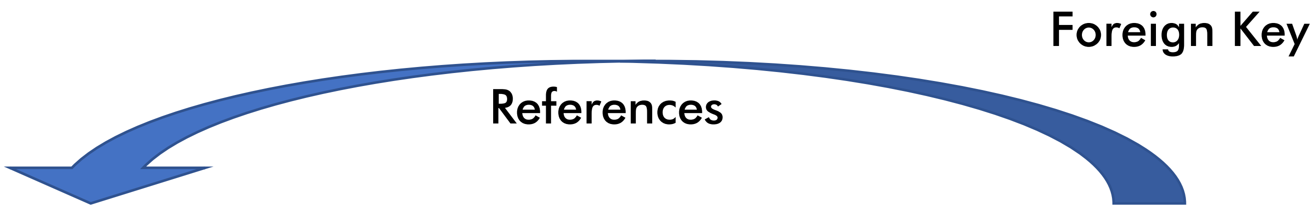
UserID	Name	Job	Salary
123	Jack	TA	50000
345	Allison	TA	60000
567	Magda	Prof	90000
789	Dan	Prof	100000

Regist

UserID	Car
123	Charger
567	Civic
567	Pinto

Foreign Keys

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



Foreign Key

References

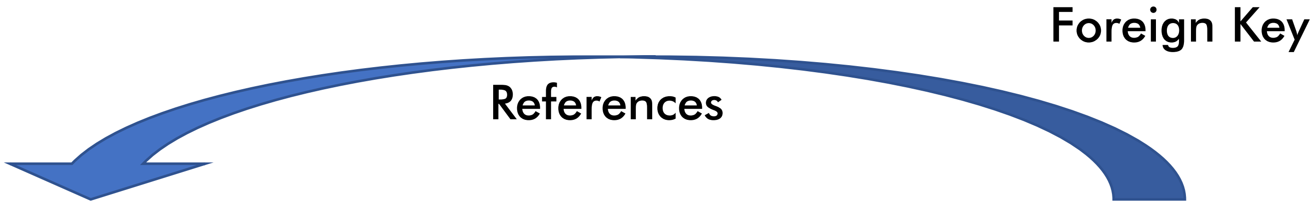
UserID	Name	Job	Salary
123	Jack	TA	50000
345	Allison	TA	60000
567	Magda	Prof	90000
789	Dan	Prof	100000

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*.



Foreign Key

References

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

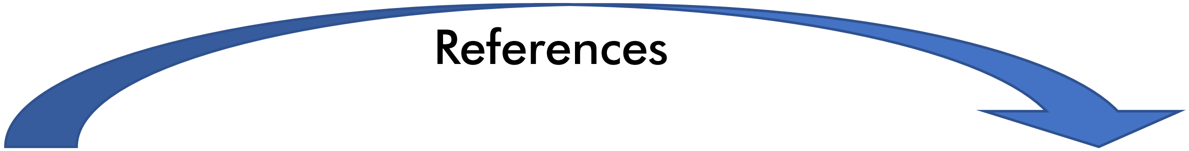
UserID	Car
123	Charger
567	Civic
567	Pinto

Foreign Keys

Foreign Key

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

Is this valid?



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

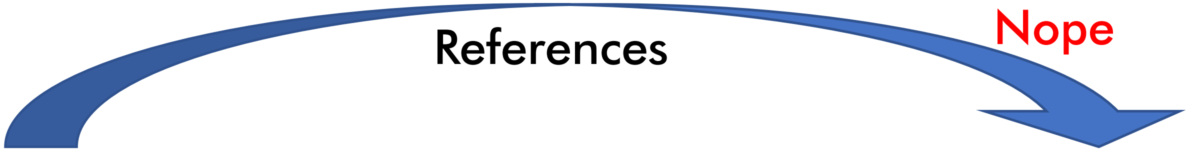
Foreign Keys

Foreign Key

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

Is this valid?

Nope



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

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

```
CREATE TABLE Regist (
  UserID INT,
  Car TEXT);
```

Payroll(UserId, Name, Job, Salary)

Regist(UserId, Car)

Foreign Keys

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

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

Payroll(UserId, Name, Job, Salary)

Regist(UserId, Car)

The Relational Model Revisited

- More complete overview of the Relational Model:
  - Database → collection of tables
  - All tables are flat
  - Keys uniquely ID rows
  - Foreign keys act as a “semantic pointer”
  - **Physical data independence**

Joins

- Foreign keys *describe* a relationship between tables
- Joins *realize* combinations of data across relationships

Inner Joins

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

Nested-Loop Semantics

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

```
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

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

```
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

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

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

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

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

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

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

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

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

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

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

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

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

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 ←

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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;
```

Inner Joins: RA

Payroll

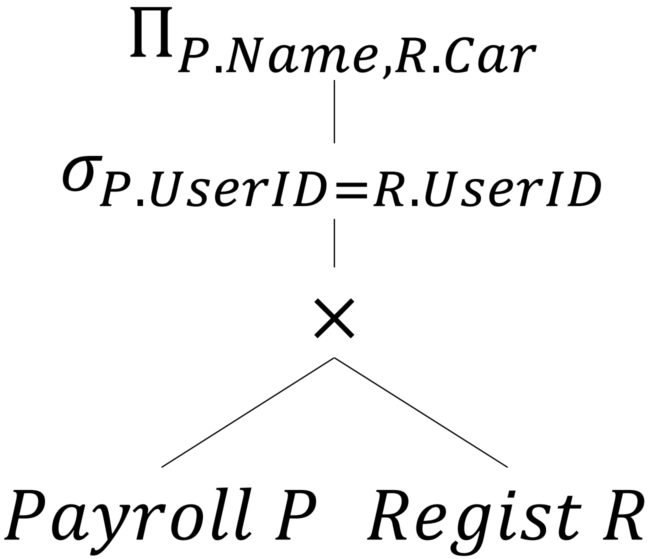
UserID	Name	Job	Salary
123	Jack	TA	50000
345	Allison	TA	60000
567	Magda	Prof	90000
789	Dan	Prof	100000

Regist

UserID	Car
123	Charger
567	Civic
567	Pinto

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

2-arg Cartesian product op (Symbol "x")
Take all pairs of tuples



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

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

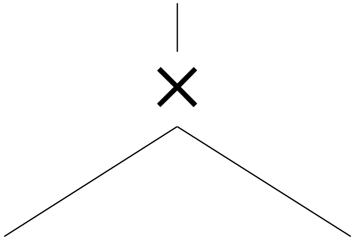
$\times$

$Payroll\ P$ $Regist\ R$

Cartesian product

UserID	Name	Job	Salary	UserID	Car
123	Jack	TA	50000	123	Charger
123	Jack	TA	50000	567	Civic
123	Jack	TA	50000	567	Pinto
345	Allison	TA	60000	123	Charger
345	Allison	TA	60000	567	Civic
…	…	…	…	…	…

(4 x 3 = 12 rows total)



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

Inner Joins: RA

Payroll

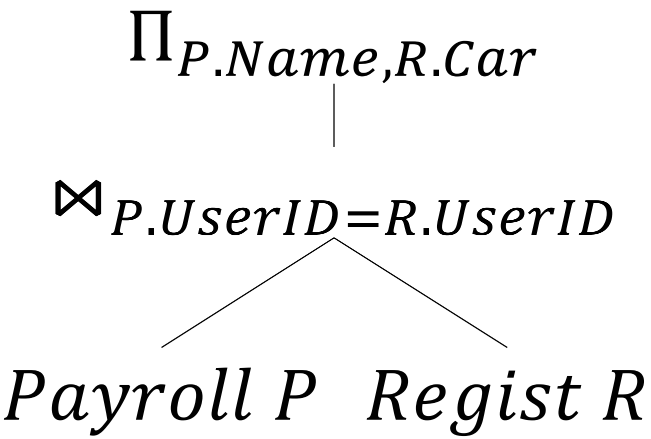
UserID	Name	Job	Salary
123	Jack	TA	50000
345	Allison	TA	60000
567	Magda	Prof	90000
789	Dan	Prof	100000

Regist

UserID	Car
123	Charger
567	Civic
567	Pinto

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

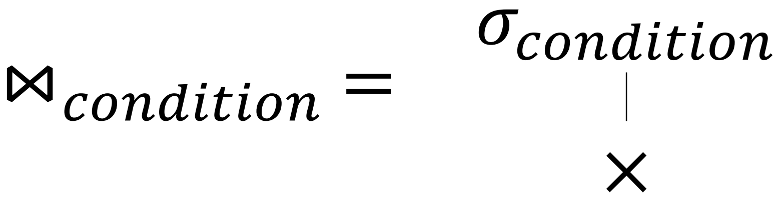
2-arg Join op (Symbol “bowtie”)
Take Cartesian product & filter



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

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

$Payroll\ P$ $Regist\ R$



$\bowtie_{condition} =$ $\sigma_{condition}$

$\times$

Outer Joins

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

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

```
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

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

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

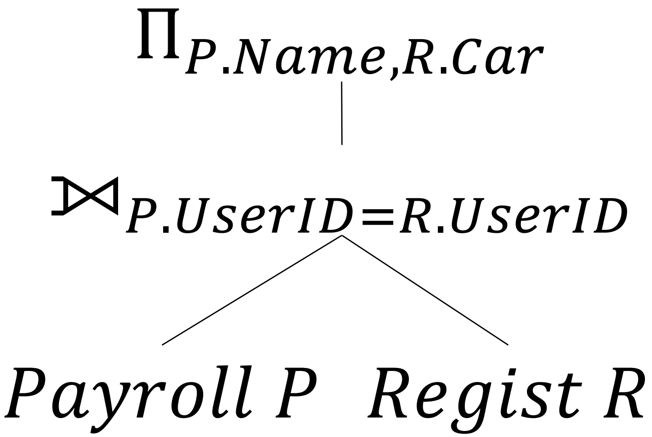
```
SELECT P.Name, R.Car
  FROM Payroll AS P LEFT OUTER JOIN Regist AS R
       ON P.UserID = R.UserID;
```

Outer Joins: RA

UserID	Name	Job	Salary
123	Jack	TA	50000
345	Allison	TA	60000
567			
789			

UserID	Car
123	Charger
567	Civic
567	Pinto

2-arg **Left Outer Join** op
(Symbol “bowtie with left edge”)
Take Join + pair non-matching entries on left with NULLs on right



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

$⟕_{P.UserID=R.UserID}$

$Payroll\ P \quad Regist\ R$

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

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

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

```
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

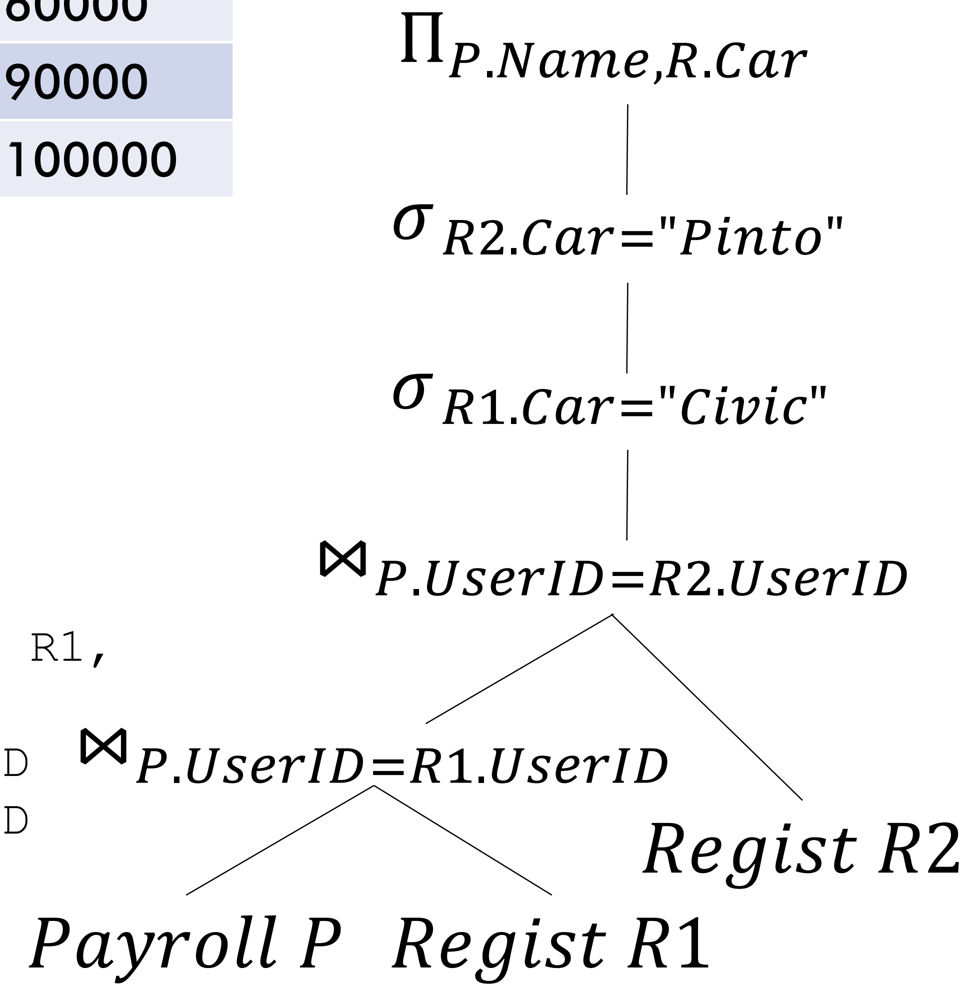
```
SELECT P.Name, R1.Car
  FROM Payroll AS P, Regist AS R1, Regist AS R2
 WHERE P.UserID = R1.UserID AND
       P.UserID = R2.UserID AND
       R1.Car = ‘Civic’ AND
       R2.Car = ‘Pinto’;
```

Self Joins

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';
```



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

$\sigma_{R2.Car="Pinto"}$

$\sigma_{R1.Car="Civic"}$

$\bowtie_{P.UserID=R2.UserID}$

$\bowtie_{P.UserID=R1.UserID}$

$Regist\ R2$

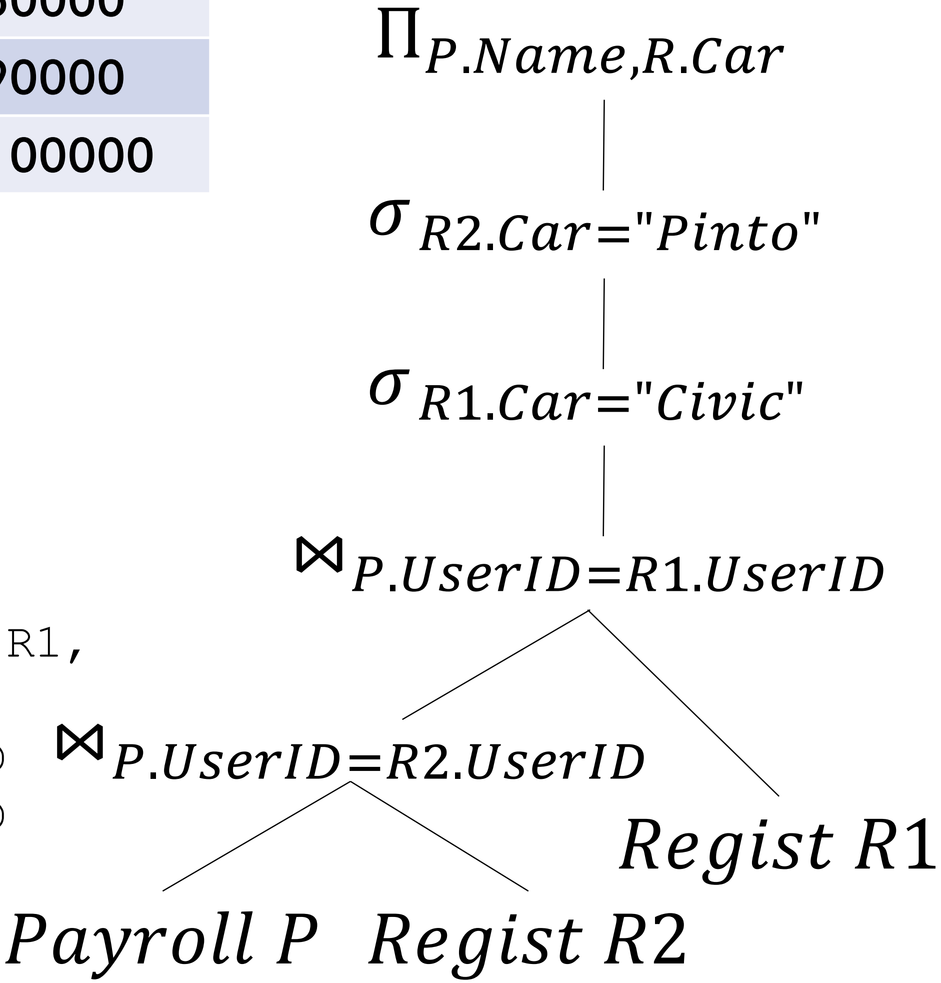
$Payroll\ P$ $Regist\ R1$

Self Joins (Equivalent RA)

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’;
```



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

$\sigma_{R2.Car="Pinto"}$

$\sigma_{R1.Car="Civic"}$

$\bowtie_{P.UserID=R1.UserID}$

$\bowtie_{P.UserID=R2.UserID}$

$Regist\ R1$

$Payroll\ P$

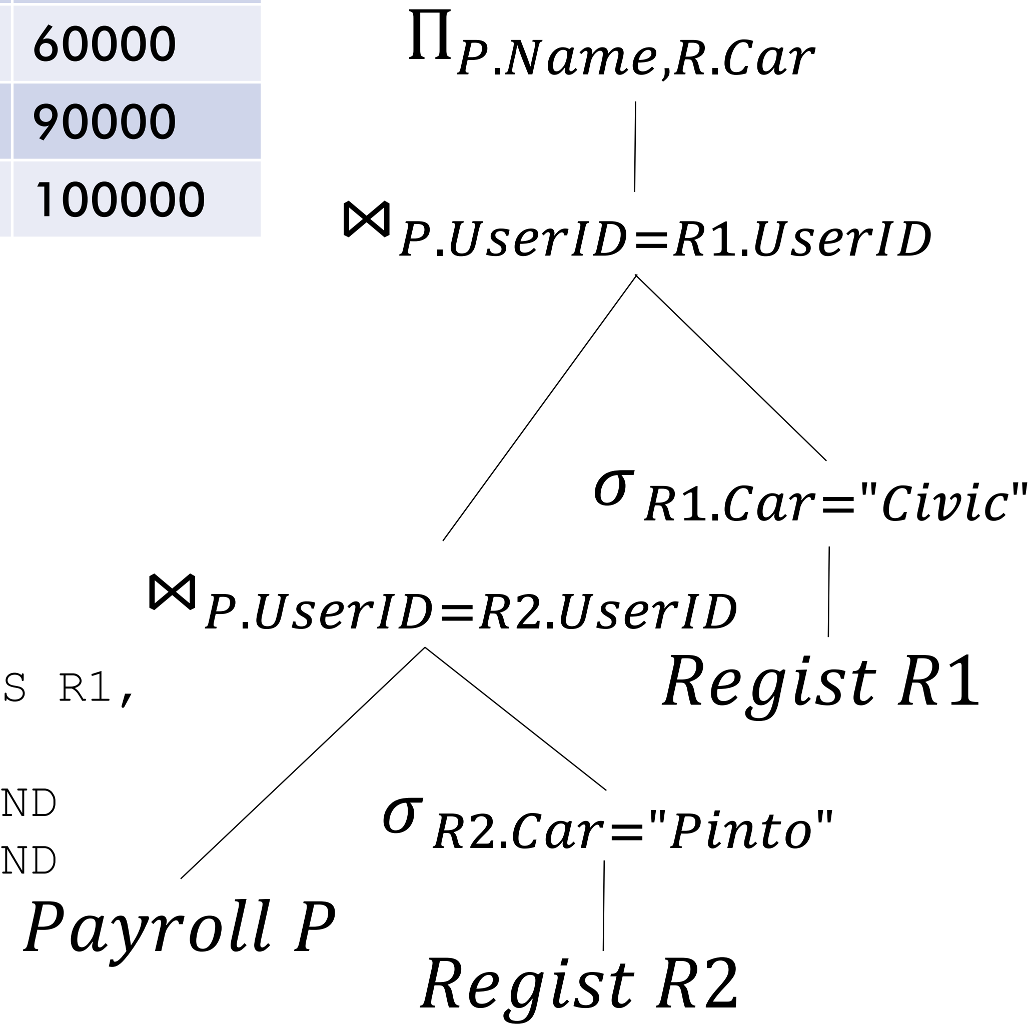
$Regist\ R2$

Self Joins (Equivalent RA)

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’;
```



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

$\bowtie_{P.UserID=R1.UserID}$

$\sigma_{R1.Car="Civic"}$

$\bowtie_{P.UserID=R2.UserID}$

$Regist\ R1$

$\sigma_{R2.Car="Pinto"}$

$Payroll\ P$

$Regist\ R2$

Takeaways

- We can describe relationships between tables with keys and foreign keys
- Different joining techniques can be used to achieve particular goals
- RA plans can be rearranged equivalently

- Our SQL toolbox is growing!
  - Not just reading and filtering data anymore
  - Starting to answer complex questions