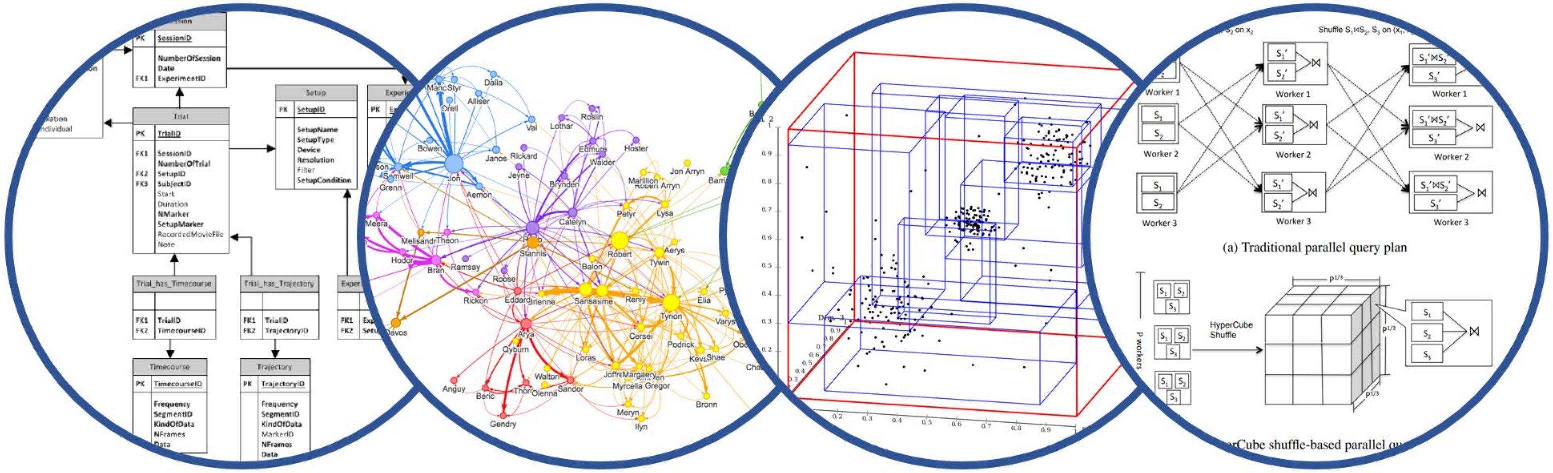


Data Management for Data Science

SQL Basics

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

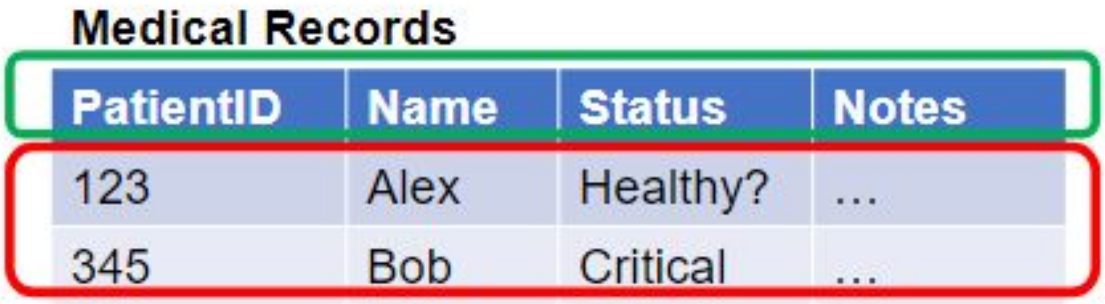
Announcements

- HW 1 released – due Friday 1/12 at **11pm**
  - Submitted via gradescope
  - Try to do HW 1 setup today (should take ~5-10 minutes)
  - Yesterday's section demo should be really useful!
  - The demo and all other section materials are on the course website

Recap - Data Models

The 3 parts of any data model

- Instance
  - The actual **data**
- Schema
  - A **description** of what data is being stored
- Query Language
  - How to retrieve and manipulate data

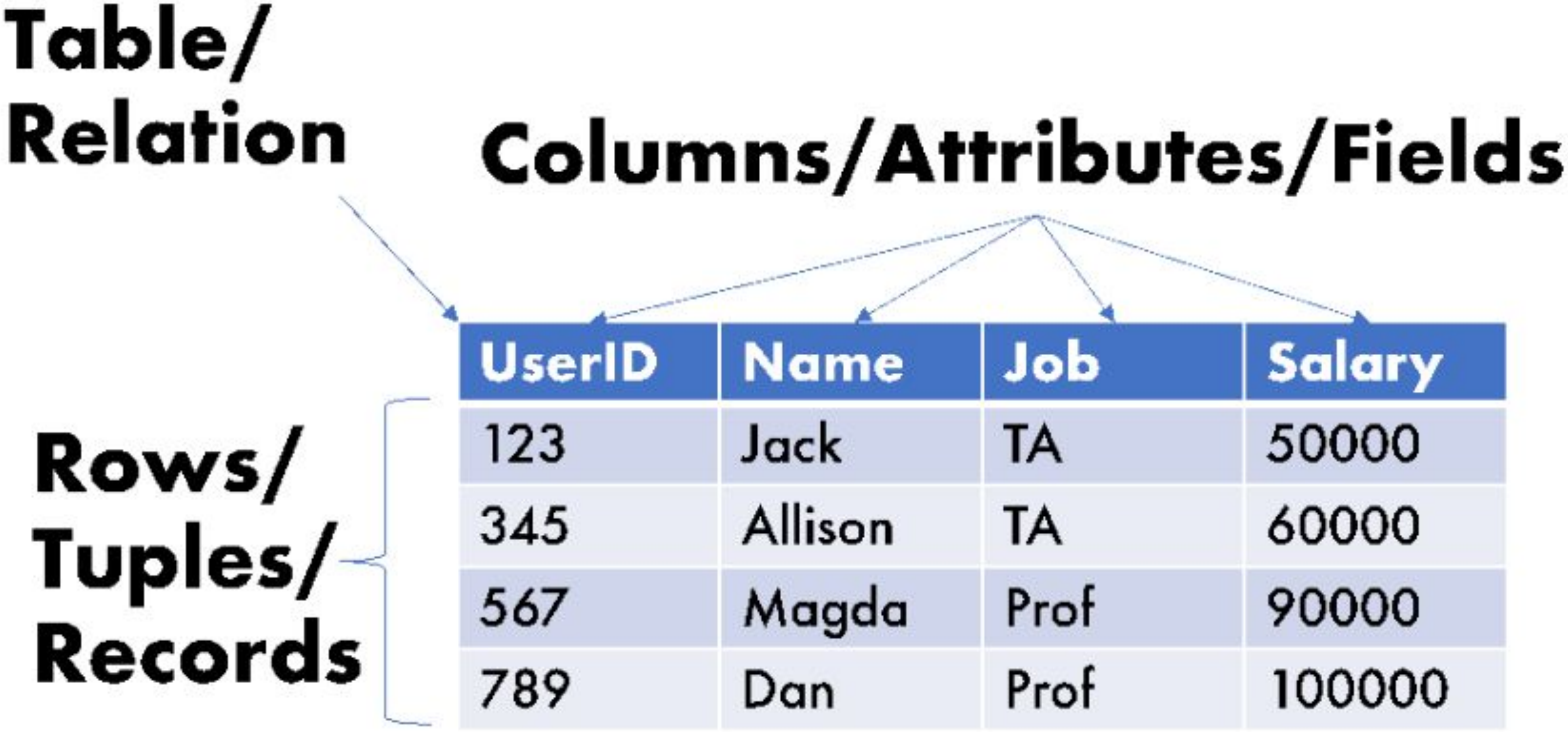


Medical Records

PatientID	Name	Status	Notes
123	Alex	Healthy?	…
345	Bob	Critical	…

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 - The Relational Model

But how is this data ACTUALLY stored?

Payroll

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

Recap - The Relational Model

But how is this data ACTUALLY stored?

Payroll

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

Don’t know. Don’t care.

Physical Data Independence

Structured Query Language - SQL

Alright, I have data and a schema.

How do I access it?

Structured Query Language - SQL

- **Declarative** query language
  - Tell the computer what you want, not how to get it
- Languages like Java/Python are procedural
- Declarative query language allows **physical data independence**

Basic SQL query

Payroll

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

```
SELECT *
  FROM Payroll;
```

Basic SQL query

Payroll

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

```
SELECT *
  FROM Payroll;
```

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

Basic SQL query

Payroll

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

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

Basic SQL query

Payroll

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

Name	UserID
Jack	123
Allison	345

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

Basic SQL query

Payroll

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

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

Name	UserID
Jack	123
Allison	345

"Payroll AS P" makes P an alias. This lets us specify that the attributes come from Payroll

SQL

Wait!

What actually happens when we execute the SQL query?

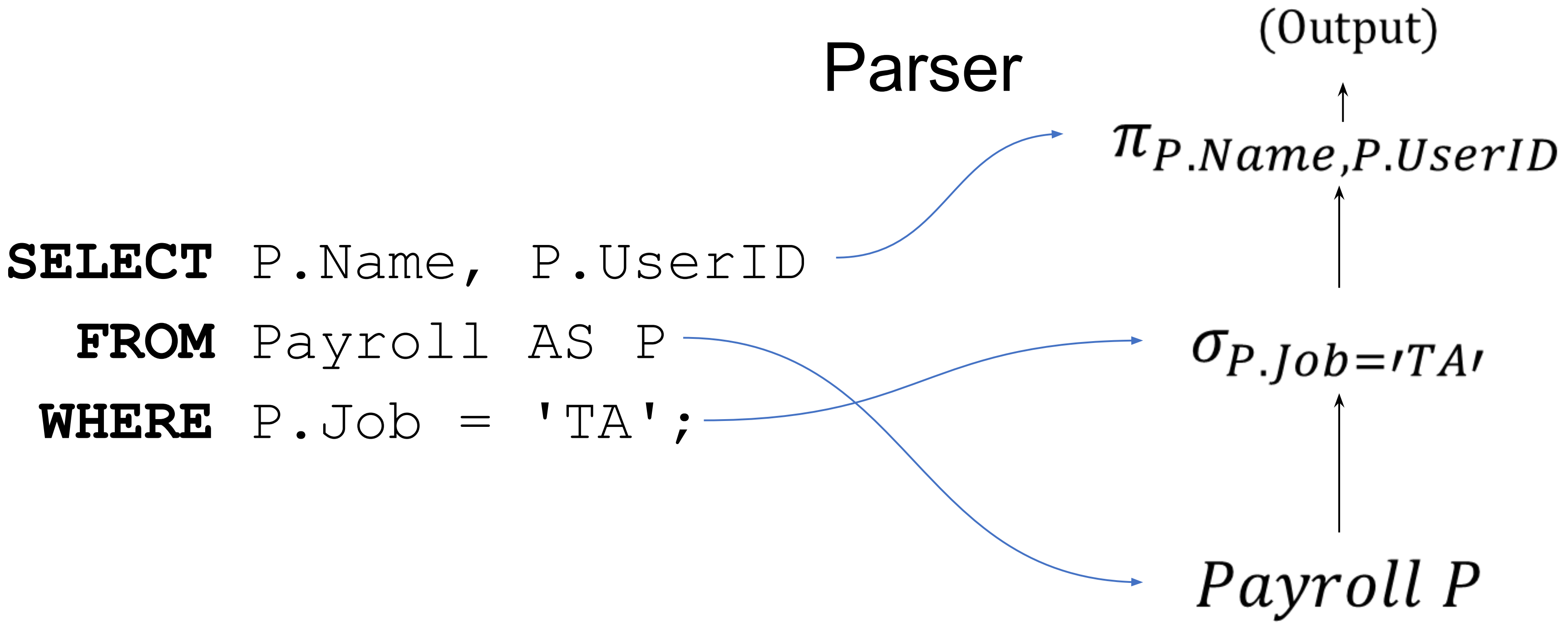
Database Internals

- Code has to boil down to instructions at some point
- Relational Database Management Systems (RDBMSs) use **Relational Algebra** (RA)

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

Database Internals

- Code has to boil down to instructions at some point
- Relational Database Management Systems (RDBMSs) use **Relational Algebra** (RA).



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

Parser

(Output)

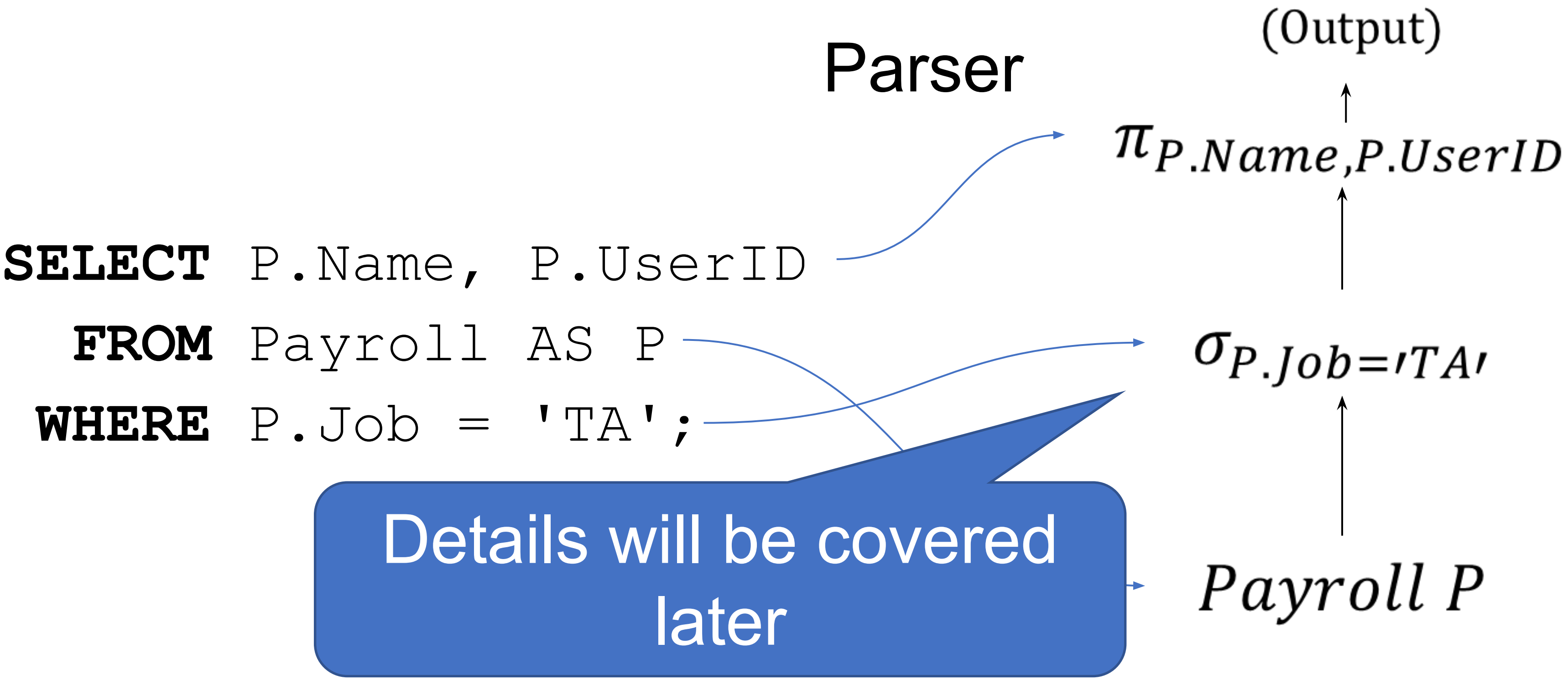
$\pi_{P.Name,P.UserID}$

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

$Payroll\ P$

Database Internals

- Code has to boil down to instructions at some point
- Relational Database Management Systems (RDBMSs) use **Relational Algebra** (RA).



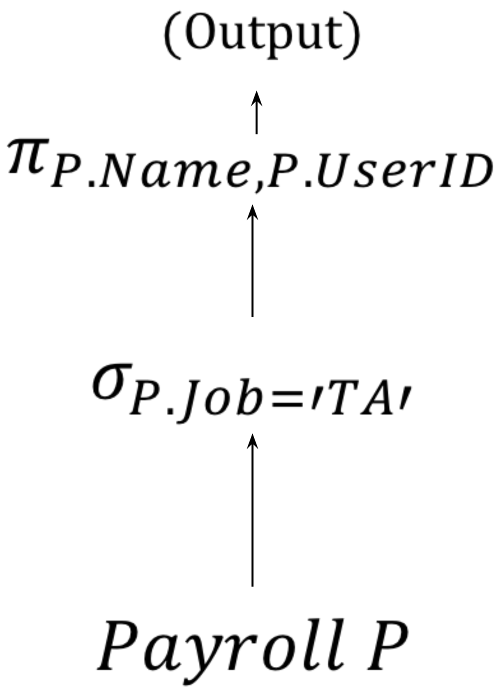
```
SELECT P.Name, P.UserID
  FROM Payroll AS P
 WHERE P.Job = 'TA';
```

Details will be covered later

Database Internals

- It's important to define the semantics (meaning) of a query

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

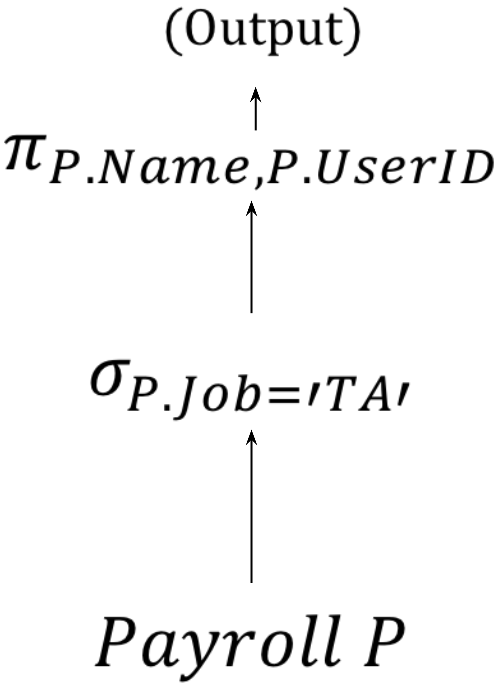


For-each semantics

Database Internals

- It's important to define the semantics (meaning) of a query

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



(Output)

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

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

$Payroll\ P$

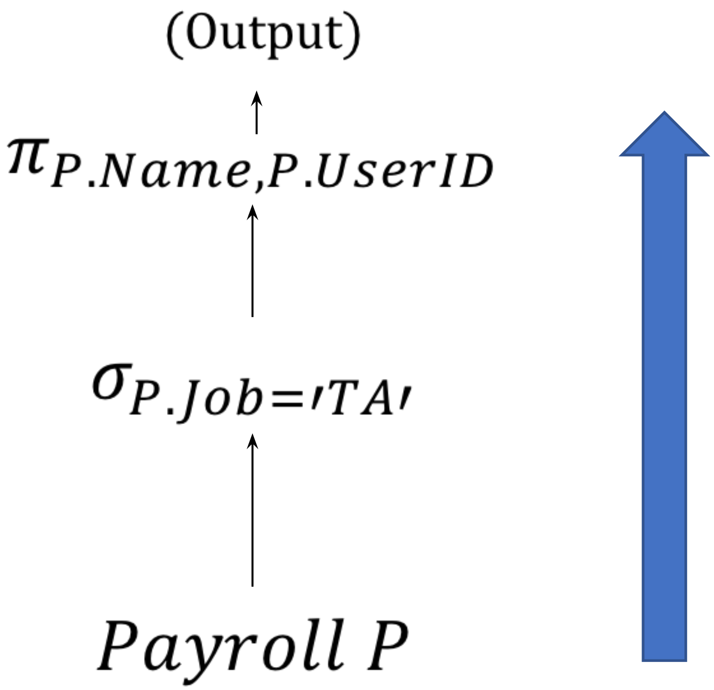
For-each semantics

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

Database Internals

- It's important to define the semantics (meaning) of a query

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



Tuples “flow” up the query plan, getting filtered and modified

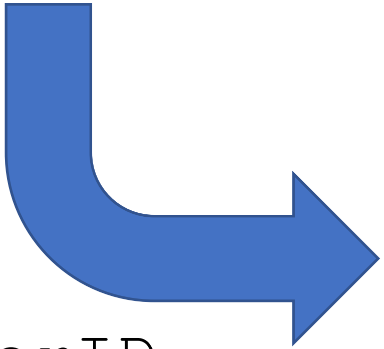
For-Each Semantics

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)
```

Job == ‘TA’?



Name	UserID

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

For-Each Semantics

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)
```

Job == 'TA'?

Name	UserID
Jack	123

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

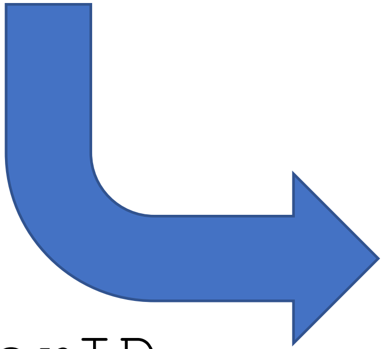
For-Each Semantics

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)
```

Job == ‘TA’?



Name	UserID
Jack	123

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

For-Each Semantics

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)
```

Job == ‘TA’?

Name	UserID
Jack	123
Allison	345

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

For-Each Semantics

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)
```

Job == ‘TA’?

Name	UserID
Jack	123
Allison	345

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

For-Each Semantics

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)
```

Job == ‘TA’?

Name	UserID
Jack	123
Allison	345

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

For-Each Semantics

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)
```

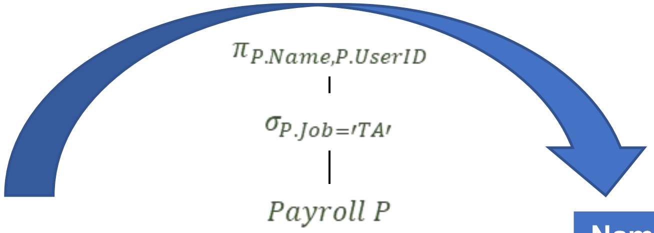
```
SELECT P.Name, P.UserID
  FROM Payroll AS P
 WHERE P.Job = 'TA';
```

Name	UserID
Jack	123
Allison	345

Recap – SQL and RA

- SQL (Next few lectures)
  - “What data do I want”
- RA (After SQL)
  - “How do I get the data”

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



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

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

$Payroll\ P$

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

Name	UserID
Jack	123
Allison	345

What's Next?

- Creating tables
- Keys □ Identification
- Foreign Keys □ Relationships
- Joins in SQL and RA
  - Inner joins
  - Outer joins
  - Self joins

Create Table Statement

Payroll(UserId, Name, Job, Salary)

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

Data Types

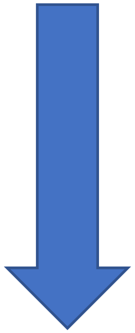
- Each attribute has a type.
  - Examples types:
    - Strings: CHAR(20), VARCHAR(50), TEXT
    - Numbers: INT, SMALLINT, FLOAT
    - MONEY, DATETIME, …
    - Few more that are DBMS specific
  - Statically and strictly enforced

Data Types

- Generally you will use:
  - **VARCHAR(N)** for strings where N is the maximum character length
    - Generally set this to as large as you need, like 256 or 1000.
  - **INT**, **FLOAT** for numbers (INTEGER works in SQLite)
  - **DATETIME** for dates
    - Can use VARCHAR(N) in SQLite

Create Table Statement

Payroll(**UserId**, Name, Job, Salary)



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

Create Table Statement

Payroll(**UserId**, Name, Job, Salary)

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

Everything is case-insensitive, but having your own guidelines is useful for readability

Keys

Key

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

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
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 VARCHAR(100),
  Job VARCHAR(100),
  Salary INT);
```

Payroll(UserId, Name, Job, Salary)

Keys

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

Unique Identifier

Payroll(UserId, Name, Job, Salary)

Keys

Unique Identifier

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

Payroll(UserId, Name, Job, Salary)

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)

Keys of more than one attribute

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

Keys of more than one attribute

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)

A little extra SQL

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

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

- DISTINCT – Deduplicates result tuples

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

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?

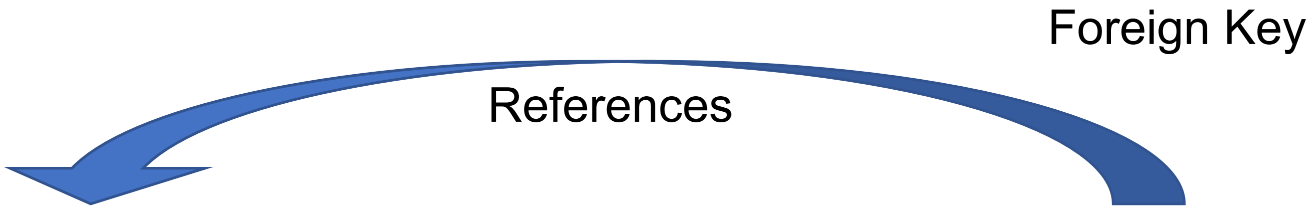
UserID	Name	Job	Salary
123	Jack	TA	50000
345	Allison	TA	60000
567	Magda	Prof	90000
789	Dan	Prof	100000

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?



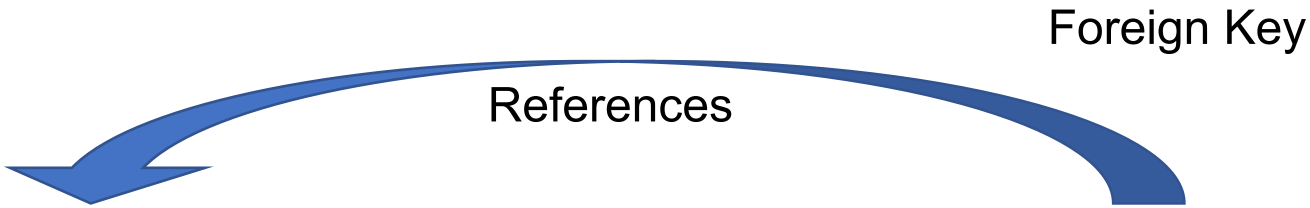
UserID	Name	Job	Salary
123	Jack	TA	50000
345	Allison	TA	60000
567	Magda	Prof	90000
789	Dan	Prof	100000

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

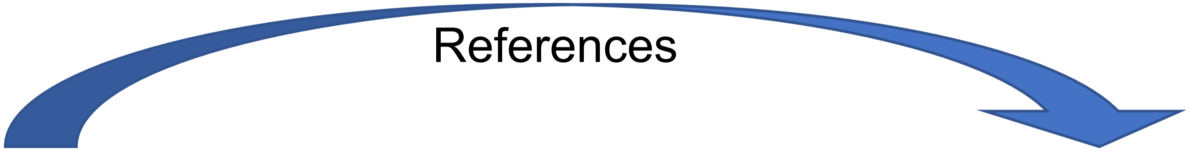
UserID	Car
123	Charger
567	Civic
567	Pinto

Foreign Keys

Foreign Key

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

Is this valid?



References

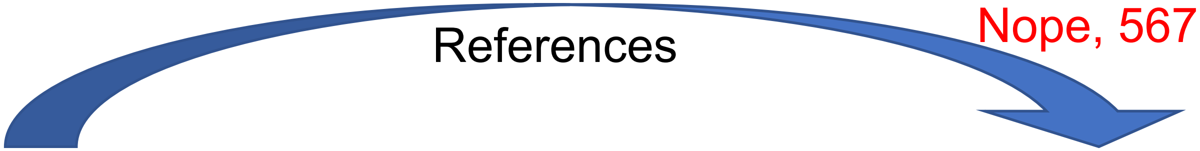
UserID	Name	Job	Salary
123	Jack	TA	50000
345	Allison	TA	60000
567	Magda	Prof	90000
789	Dan	Prof	100000

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

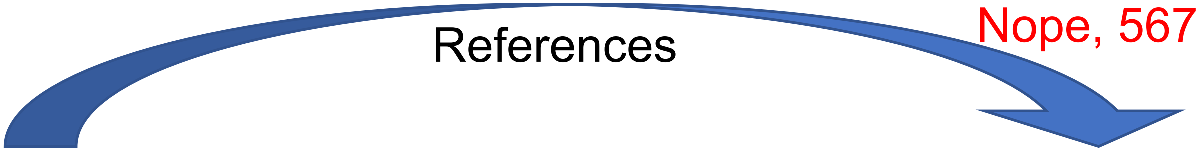
Foreign Keys

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 (point to) a unique attribute, almost always a 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

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

Payroll(UserID, Name, Job, Salary)

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

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

Regist(UserID, Name, 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

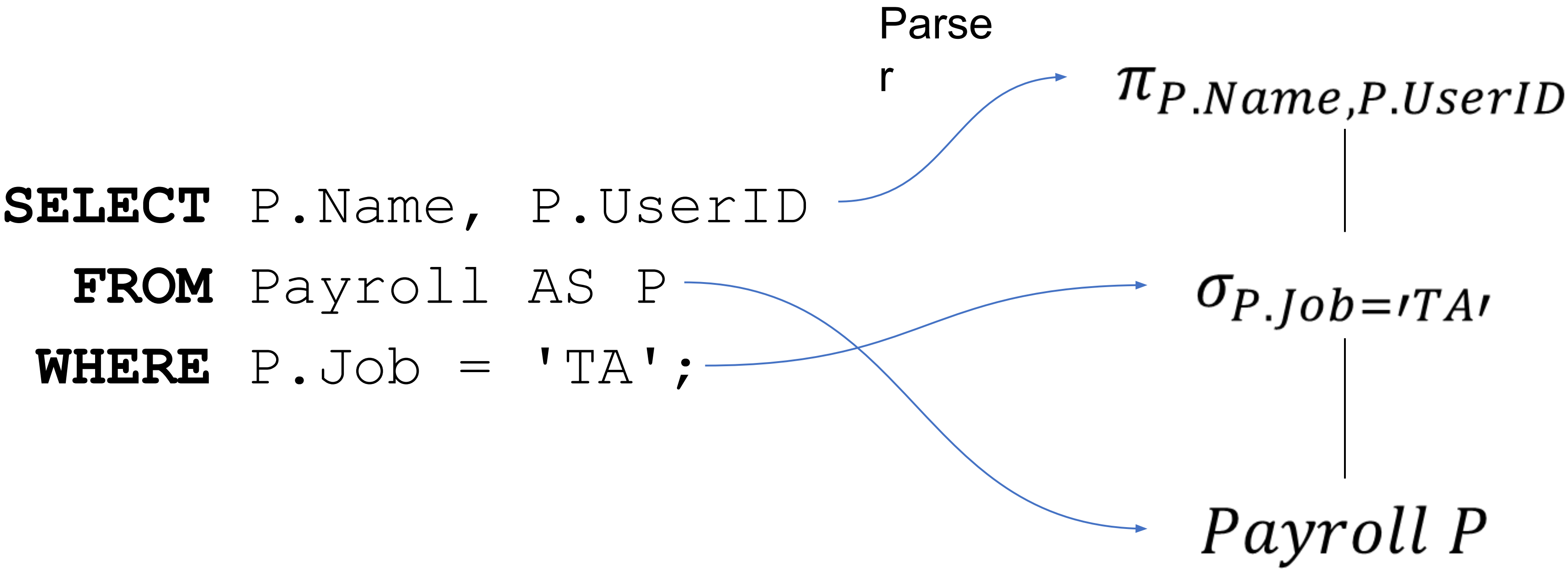
- Foreign keys are able to *describe* a relationship between tables
- Joins are able to realize combinations of data

RA Equivalencies

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

RA Equivalencies

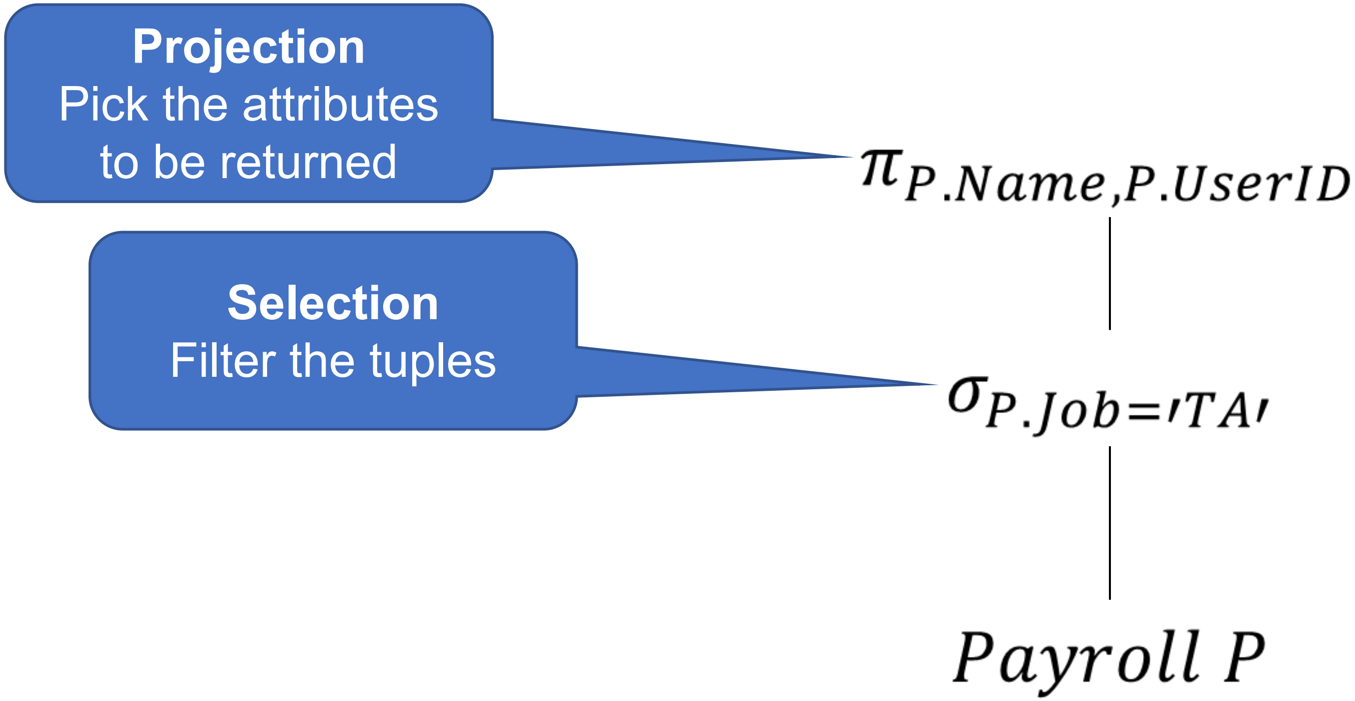
So far we haven't discussed equivalent RA trees.
But all joins can be parsed directly into a “join tree”



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

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

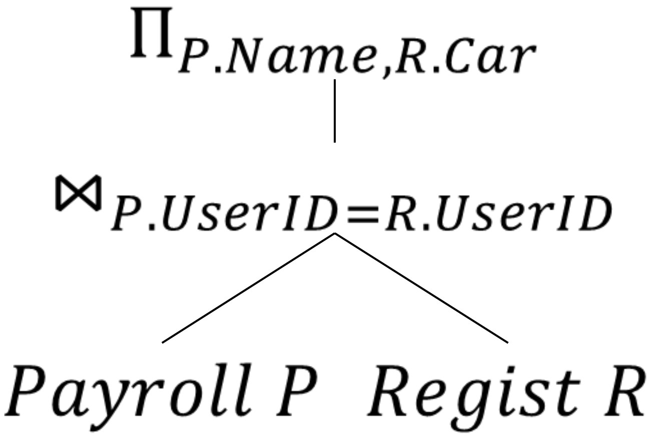
$\pi_{P.Name,P.UserID}$

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

$Payroll\ P$

RA Equivalencies

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



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

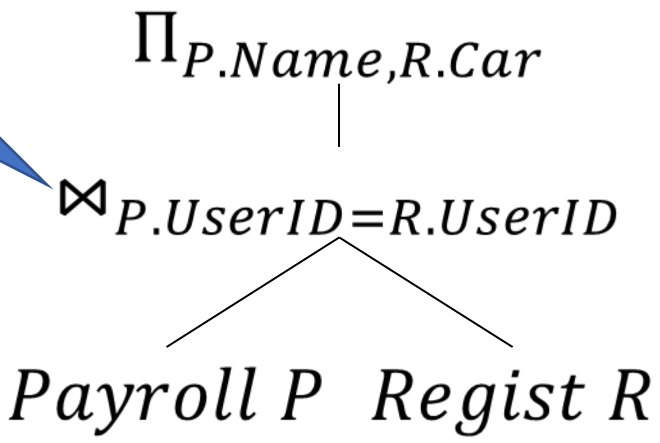
$\bowtie_{P.UserID=R.UserID}$

$Payroll\ P$ $Regist\ R$

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



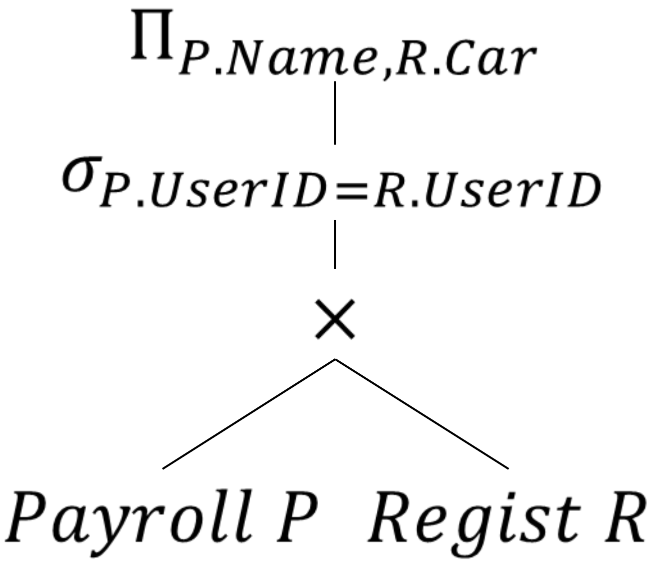
$\Pi_{P.Name,R.Car}$

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

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

RA Equivalencies

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



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

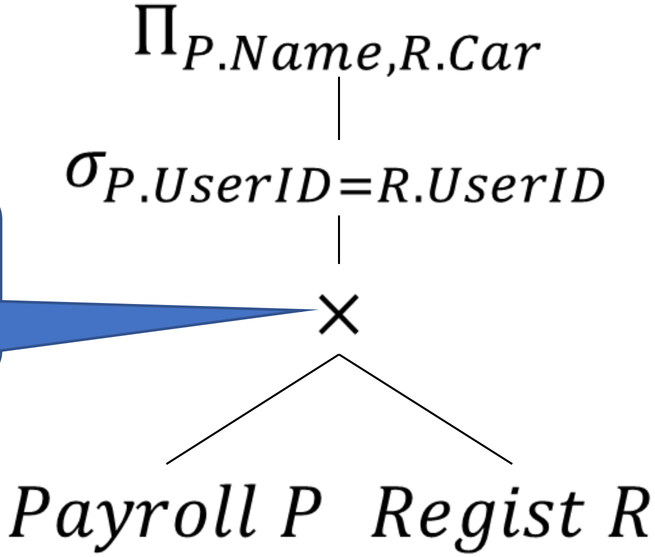
$\sigma_{P.UserID=R.UserID}$

$\times$

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

RA Equivalencies

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



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