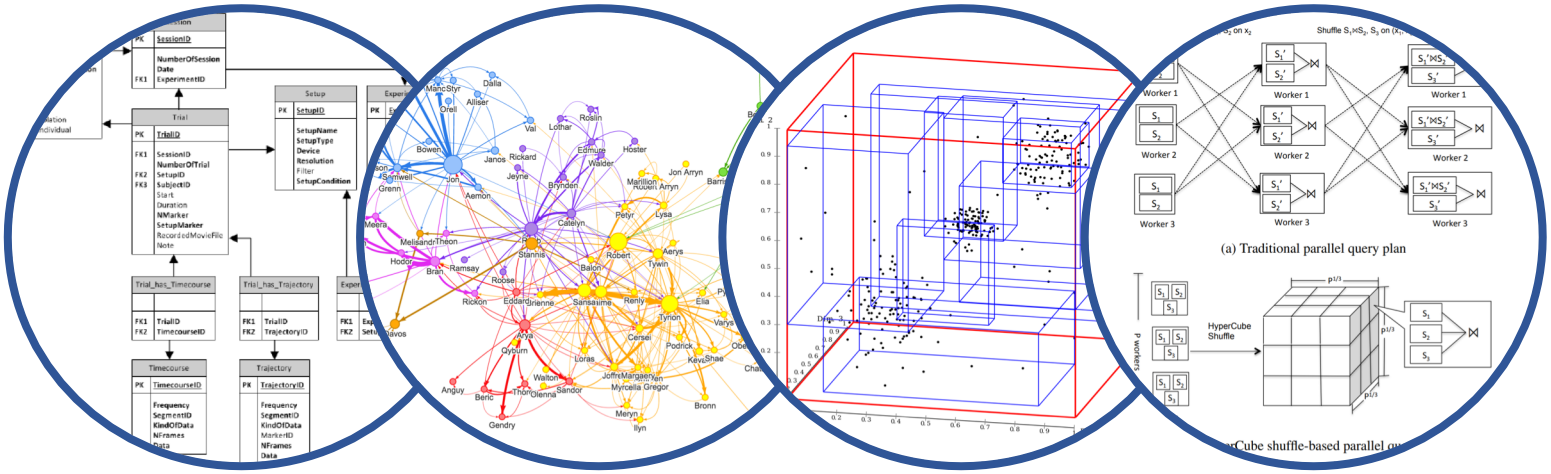

Introduction to Data Management

SQL, Keys, and Joins

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

Announcements

- Section and TA OH start tomorrow
  - Double check your section, it may have changed in MyUW:

AA	Thursday	2:20 – 3:20PM	MOR 234
AB	Thursday	1:10 – 2:10PM	MOR 221

- HW1 will be released soon via GitLab

Last time's takeaways!

- Elements of the relational data model
- How a basic **SELECT-FROM-WHERE** query works

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

Recap – SQL and RA

- SQL (Next several 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

Loop Semantics

Payroll

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

Name	UserID

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

Loop Semantics

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

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

Loop Semantics

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

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

Loop Semantics

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';
```

Loop Semantics

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';
```

Loop Semantics

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';
```

Loop Semantics

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';
```

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

Characteristics of the Relational Model

- **Set semantics**
  - No duplicate tuples
- Attributes are **typed** and **static**
  - INTEGER, FLOAT, VARCHAR(n), DATETIME, …
- Tables are **flat**

Today's Outline

- Creating Tables
- Keys → Identification
- Foreign Keys → Relationships
- Joins in SQL
  - 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);
```

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);
```

Case insensitive, but useful for readability

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 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

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

Unique Identifier

Payroll(<u>UserId</u>, 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,
  Name VARCHAR(100),
  Job VARCHAR(100),
  Salary INT,
  PRIMARY KEY (UserId, Name));
```

Payroll(<u>UserId</u>, <u>Name</u>, 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	Car
123	Charger
567	Civic
567	Pinto

Foreign Keys

- Databases can hold multiple tables
- How do we 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 VARCHAR(100),
  Job VARCHAR(100),
  Salary INT);
```

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

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

Regist(UserId, Car)

Joins

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

Inner Joins

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

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 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;
```

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

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';
```

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;
```

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