

CSE 344: Intro to Data Management

SQL Basics

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

Announcements

- HW1 is due on Wednesday!

Recap – The Relational Model

**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

- Data is stored in simple, flat relations



We start today

- Is retrieved via a set-at-a-time query language
- No prescription for the physical representation

SQL Basics

Structured Query Language - SQL

- Domain-specific: for relational databases only
- Not general purpose like Java, python, C/C++, ...
- Declarative, set-at-a-time
 - You describe **what** data you want
 - System figures out **how** to execute it

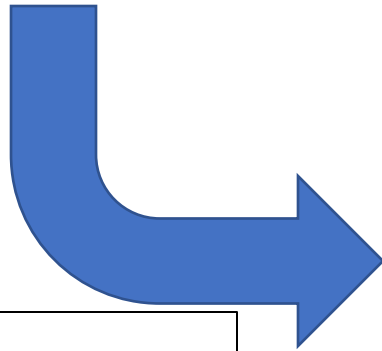
Basic SQL query: SELECT-FROM-WHERE

```
SELECT *  
FROM Payroll;
```

Basic SQL query: SELECT-FROM-WHERE

Payroll

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

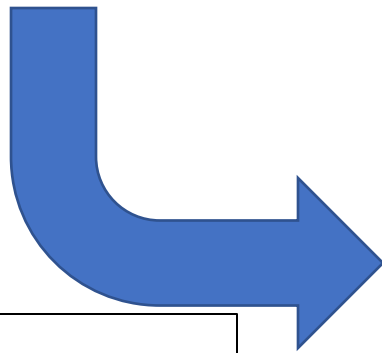


```
SELECT *  
FROM Payroll;
```


Basic SQL query: SELECT-FROM-WHERE

Payroll

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



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: SELECT-FROM-WHERE

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: SELECT-FROM-WHERE

Payroll

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

SELECT attributes

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

Basic SQL query: SELECT-FROM-WHERE

Payroll

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

SELECT attributes

FROM table(s)

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

Basic SQL query: SELECT-FROM-WHERE

Payroll

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

SELECT attributes

FROM table(s)

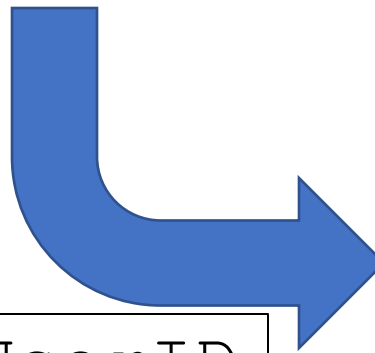
WHERE
filter condition

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

Basic SQL query: SELECT-FROM-WHERE

Payroll

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



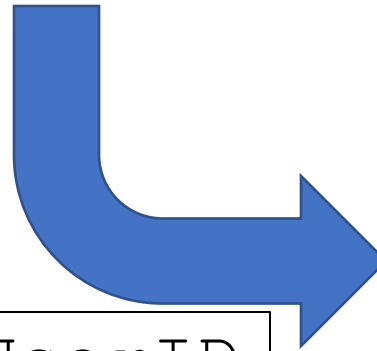
?

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

Basic SQL query: SELECT-FROM-WHERE

Payroll

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



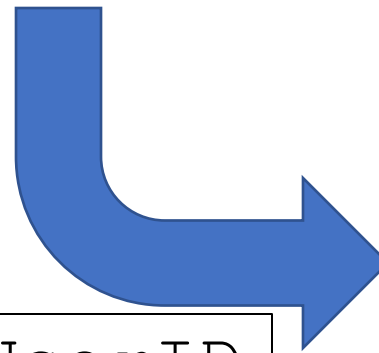
Name	UserID
Jack	123
Allison	345

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

Basic SQL query: SELECT-FROM-WHERE

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';
```

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

Semantics:

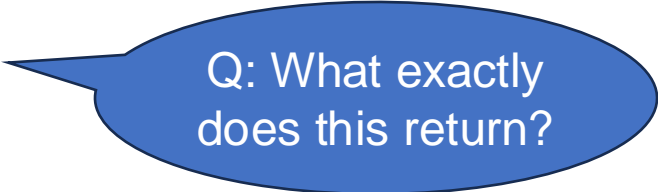
What does a SQL query mean?

For-each semantics

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';
```



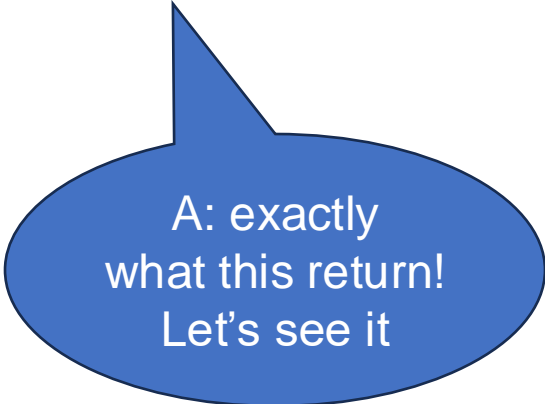
Q: What exactly
does this return?

For-each semantics

Payroll

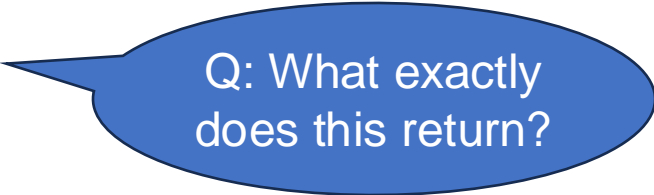
UserID	Name	Job	Salary
123	Jack	TA	50000
345	Allison	TA	60000
567	Magda	Prof	90000
789	Dan	Prof	100000

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



A: exactly
what this return!
Let's see it

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



Q: What exactly
does this return?

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 Payroll:  
    if (row.Job == 'TA'):  
        output (row.Name, row.UserID)
```

Job == 'TA'?



```
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 Payroll:  
    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 Payroll:  
    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 Payroll:  
    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 Payroll:  
    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 Payroll:  
    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 Payroll:  
    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 Payroll:  
    if (row.Job == 'TA'):  
        output (row.Name, row.UserID)
```

Final output



Name	UserID
Jack	123
Allison	345

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

Discussion

- For-each semantics is what SQL computes
- The system decides how to execute it

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

Discussion

- For-each semantics is what SQL computes
- The system decides how to execute it, e.g.:
 - Iterate over Payroll... (like the the for-each semantics)

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

Discussion

- For-each semantics is what SQL computes
- The system decides how to execute it, e.g.:
 - Iterate over Payroll... (like the the for-each semantics)
 - Or: lookup the Job index for 'TA', read those records

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

Discussion

- For-each semantics is what SQL computes
- The system decides how to execute it, e.g.:
 - Iterate over Payroll... (like the the for-each semantics)
 - Or: lookup the Job index for 'TA', read those records
 - Or: iterate over a bit-map index for JOB ...
 - ...

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

ORDER-BY and DISTINCT

Order By

```
SELECT *  
FROM Payroll  
ORDER BY Name;
```

Payroll

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

Order By

```
SELECT *  
FROM Payroll  
ORDER BY Name;
```

Payroll

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



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

Order By

```
SELECT *  
FROM Payroll  
ORDER BY Job;
```

Payroll

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

Order By

```
SELECT *  
FROM Payroll  
ORDER BY Job;
```

Payroll

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



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

Order By

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

Payroll

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

Order By

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

Payroll

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



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

Order By

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

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

What's
the difference?
(in class)

Payroll

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



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

Distinct

```
SELECT Job  
FROM Payroll
```

Payroll

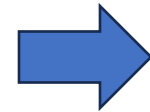
UserID	Name	Job	Salary
123	Jack	TA	50000
345	Allison	TA	60000
567	Magda	Prof	90000
789	Dan	Prof	100000

Distinct

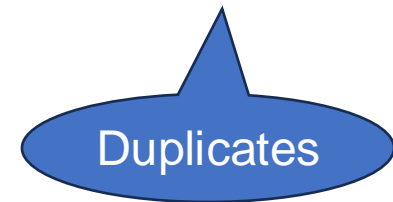
```
SELECT Job  
FROM Payroll
```

Payroll

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



Job
TA
TA
Prof
Prof



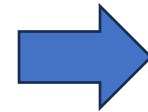
Bag semantics

Distinct

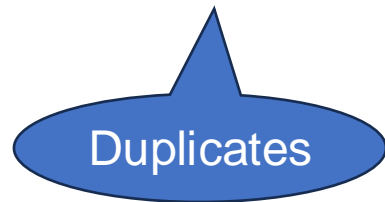
```
SELECT Job  
FROM Payroll
```

Payroll

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



Job
TA
TA
Prof
Prof



Bag semantics



Job
TA
Prof



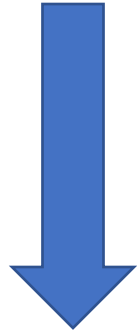
Set semantics

```
SELECT DISTINCT Job  
FROM Payroll
```

Tables in SQL

Create Table Statement

Payroll(UserID, Name, Job, Salary)



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

Case-insensitive,
but use own guidelines
for readability

Data types

- Each attribute has a type
 - Strings: CHAR(20), VARCHAR(50), TEXT
 - Numbers: INT, SMALLINT, FLOAT, ...
 - MONEY, DATETIME, ...
 - ...many, many vendor specific types

- Statically enforced (except Sqlite)

Insert

Payroll(UserID, Name, Job, Salary)

```
INSERT INTO Payroll VALUES (123, 'Jack', 'TA', 50000);  
INSERT INTO Payroll VALUES (345, 'Allison', 'TA', 60000);  
.  
.  
.  
.  
.  
.
```



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

Insert

Payroll(UserID, Name, Job, Salary)

```
INSERT INTO Payroll VALUES (123, 'Jack', 'TA', 50000);  
INSERT INTO Payroll VALUES (345, 'Allison', 'TA', 60000);  
.  
.  
.  
.  
.
```



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

Persistence

The table is *persistent*: it continues to exist after we shut down the computer

Keys

- We often need a way to select exactly one record
- A **key attribute** uniquely identifies the record
- Let's see how this works...

Keys

Key

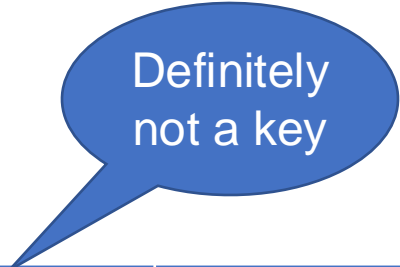
A **Key** is one or more attributes that **uniquely** identify a row.

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.



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

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

```
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);
```

Payroll(UserID, Name, Job, Salary)

Keys

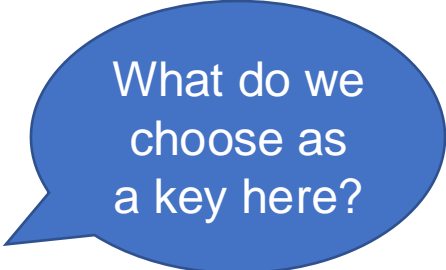
```
CREATE TABLE Payroll (  
    UserID INT,  
    Name TEXT,  
    Job TEXT,  
    Salary INT,  
    PRIMARY KEY (UserId) ;
```

Can also
define the PK
at the end

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.



What do we choose as a key here?

Name	Job	Salary
Alice	TA	20000
Alice	Prof	200000
Bob	Prof	200000

Keys of more than one attribute

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

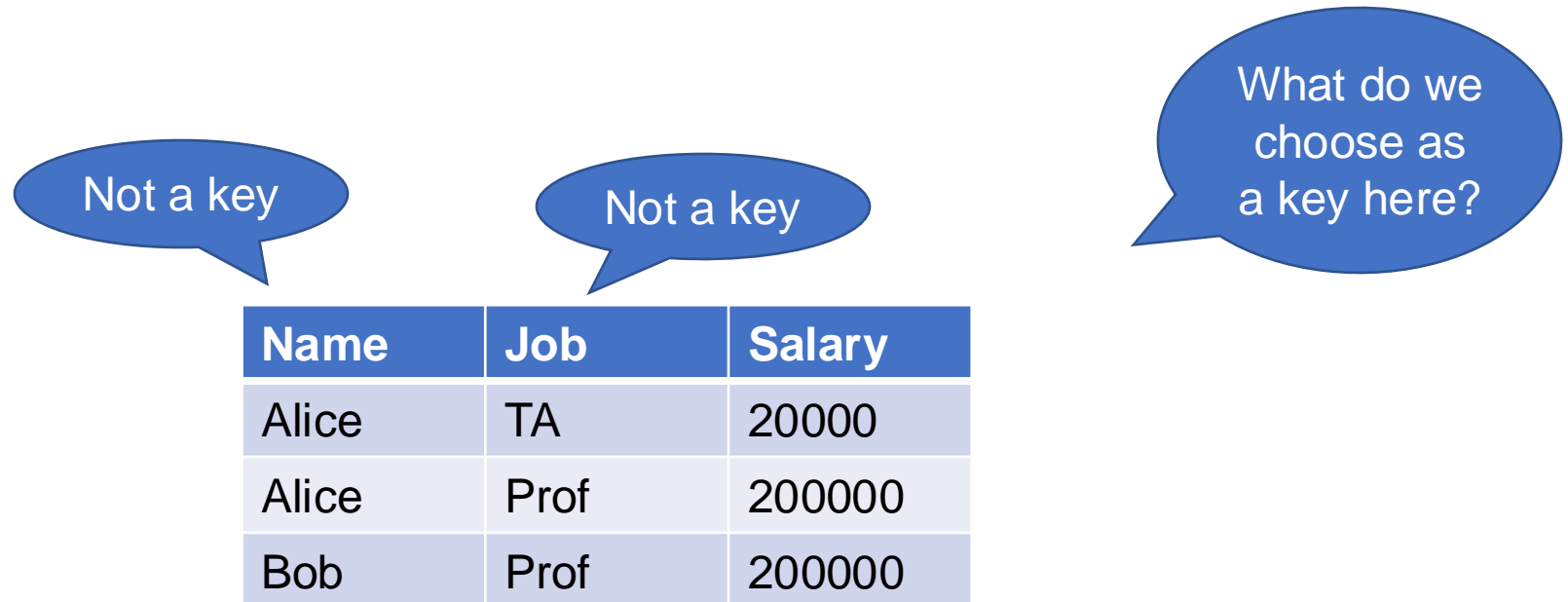
Not a key

Name	Job	Salary
Alice	TA	20000
Alice	Prof	200000
Bob	Prof	200000

What do we choose as a key here?

Keys of more than one attribute

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



Not a key

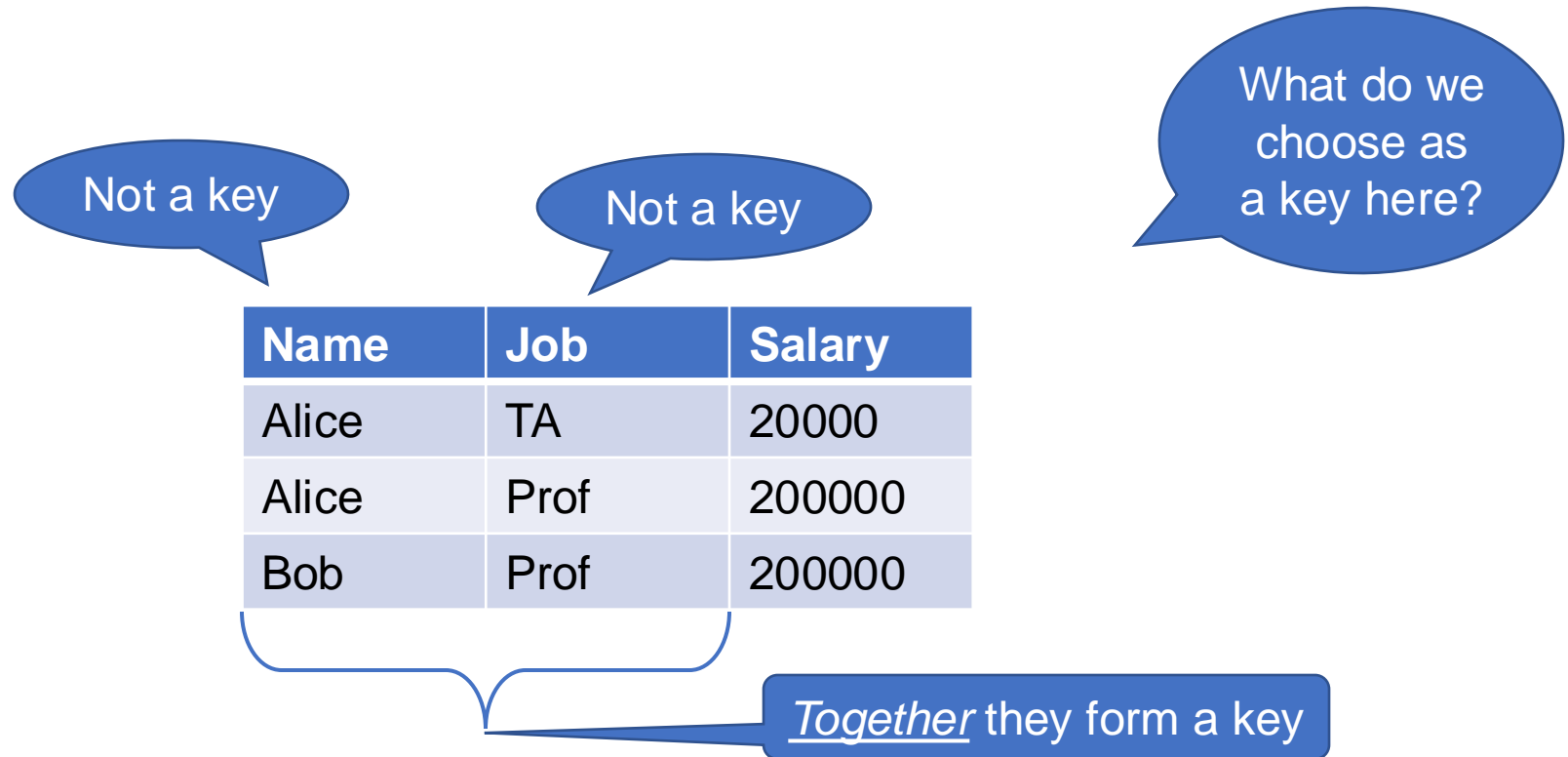
Not a key

What do we choose as a key here?

Name	Job	Salary
Alice	TA	20000
Alice	Prof	200000
Bob	Prof	200000

Keys of more than one attribute

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



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 TEXT,  
    Job TEXT,  
    Salary INT,  
    );
```

Name	Job	Salary
Alice	TA	20000
Alice	Prof	200000
Bob	Prof	200000



Together they form a key

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 TEXT,  
  Job TEXT,  
  Salary INT,  
  PRIMARY KEY (Name, Job) );
```

Must use this syntax for multi-attribute key

Name	Job	Salary
Alice	TA	20000
Alice	Prof	200000
Bob	Prof	200000

Together they form a key

Discussion

- Key= attribute(s) that uniquely identifies a record
- Sometimes more than one choice
SQL requires choosing one: **the primary key**
- Good practice: each table has a primary key
(but there are exceptions)
- How do we use keys? With **foreign keys**. Next...

Foreign Keys

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

Payroll

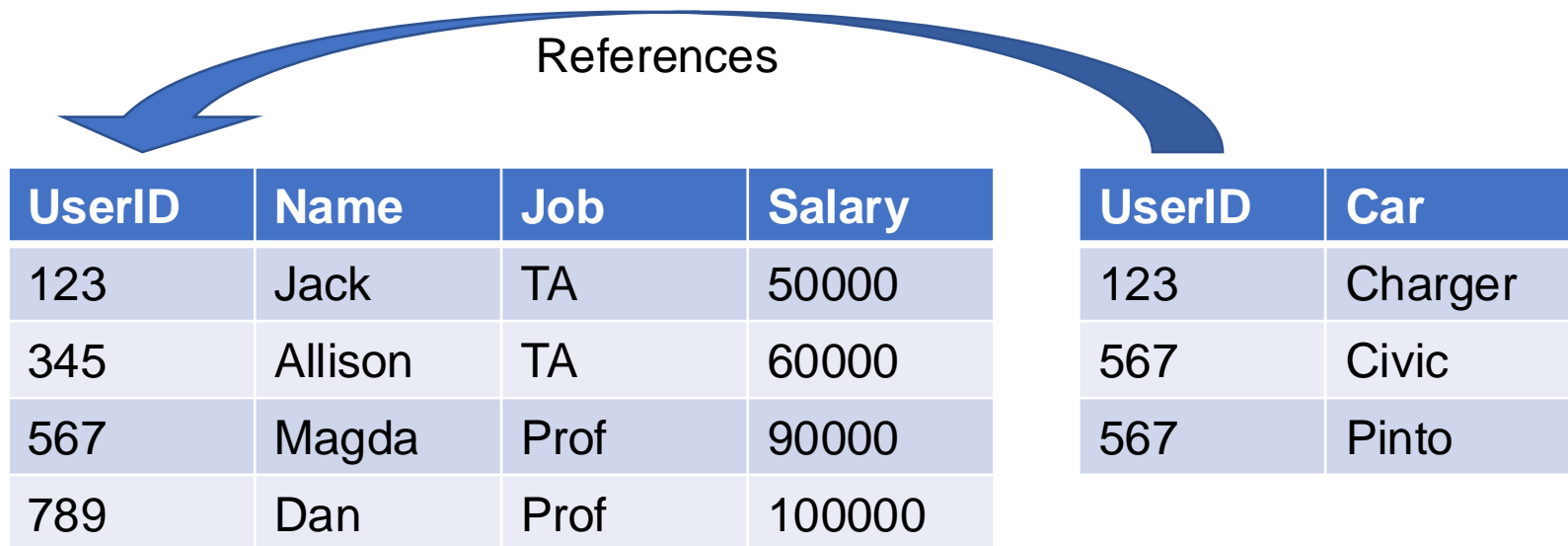
UserID	Name	Job	Salary
123	Jack	TA	50000
345	Allison	TA	60000
567	Magda	Prof	90000
789	Dan	Prof	100000

Regist

UserID	Car
123	Charger
567	Civic
567	Pinto

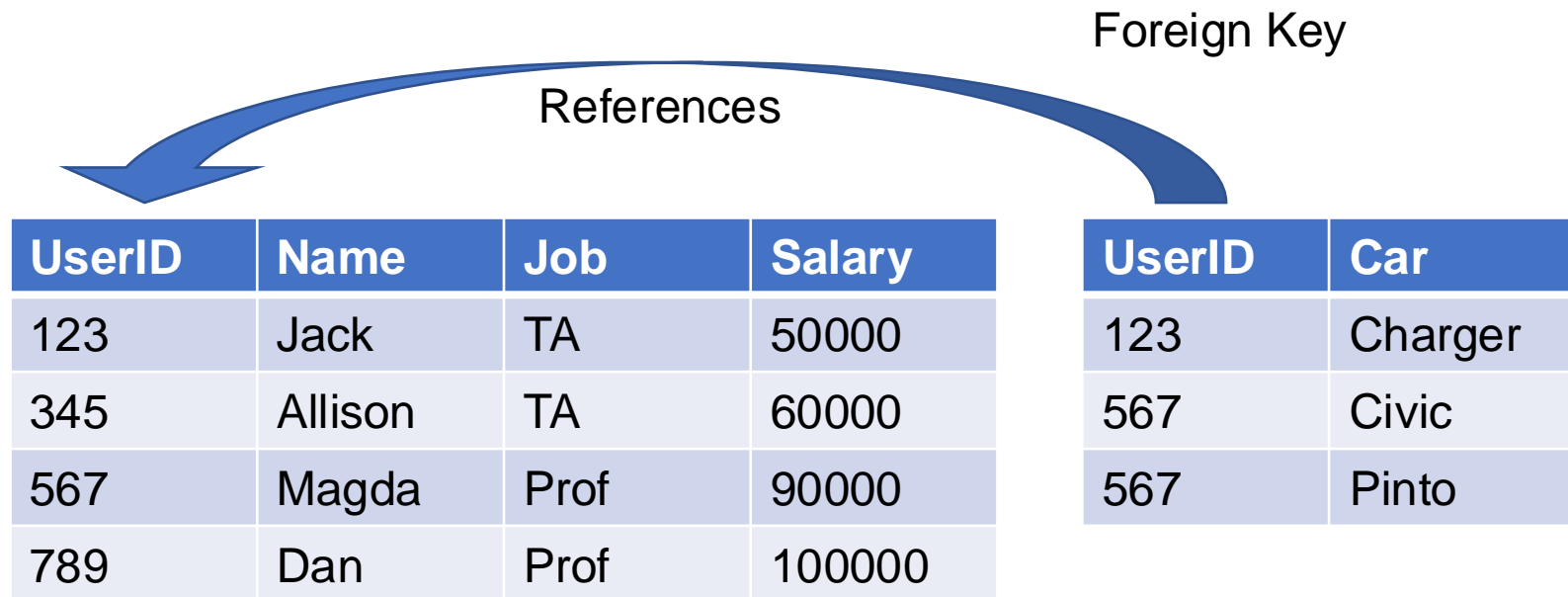
Foreign Keys

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



Foreign Keys

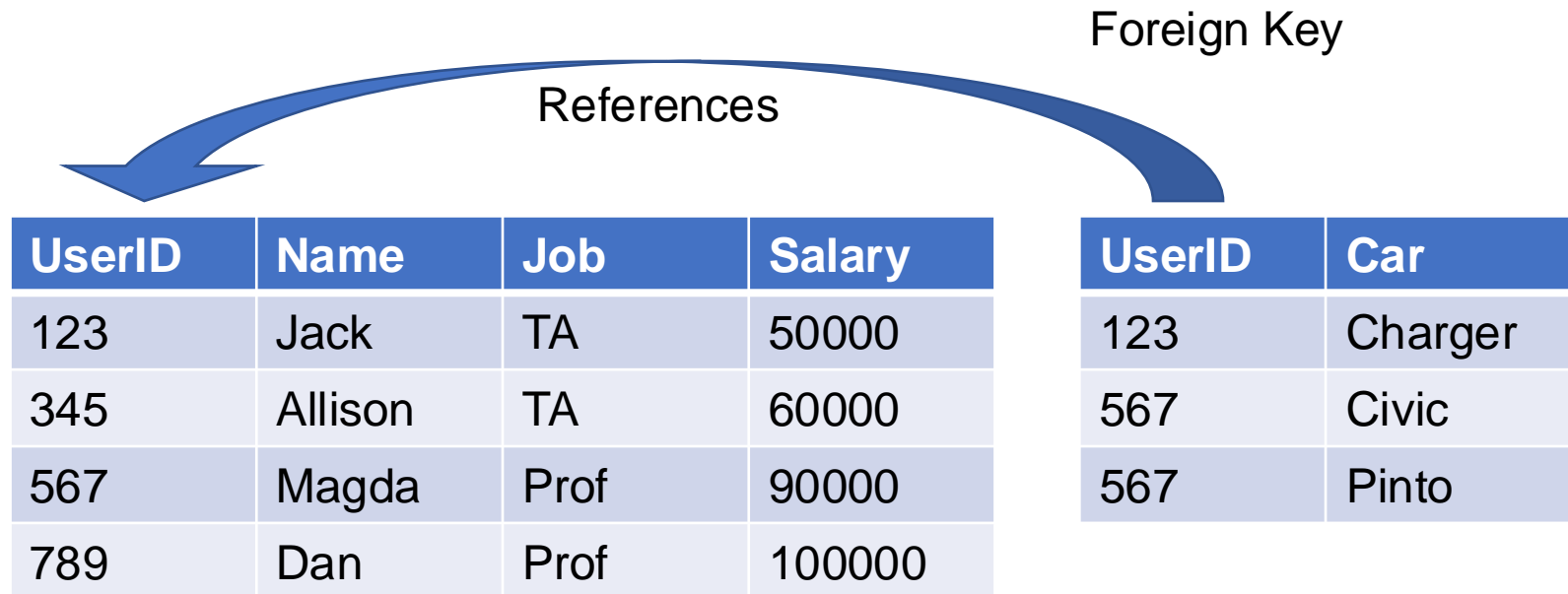
- Databases can hold multiple tables
- How do we capture relationships *between* tables?



Foreign Keys

Foreign Key

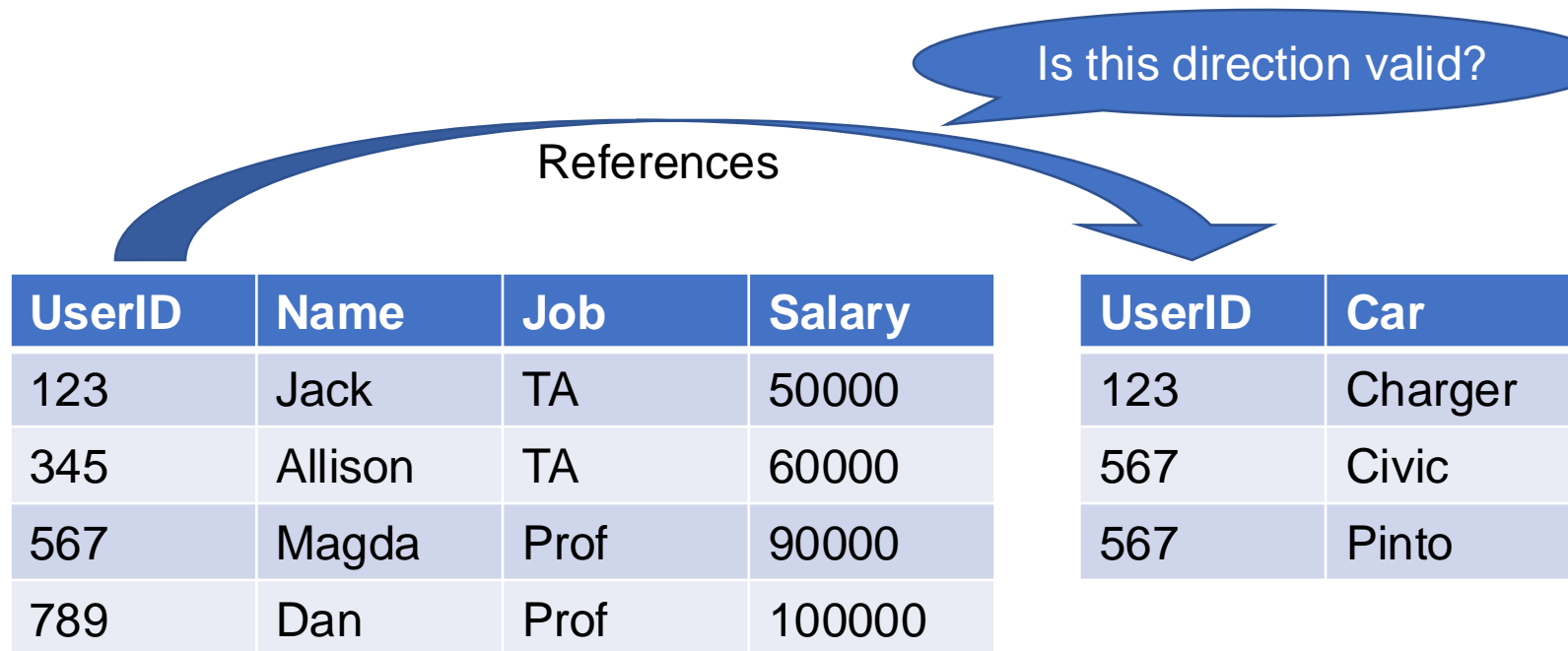
A **Foreign Key** is one or more attributes that uniquely identify a row in *another table*.



Foreign Keys

Foreign Key

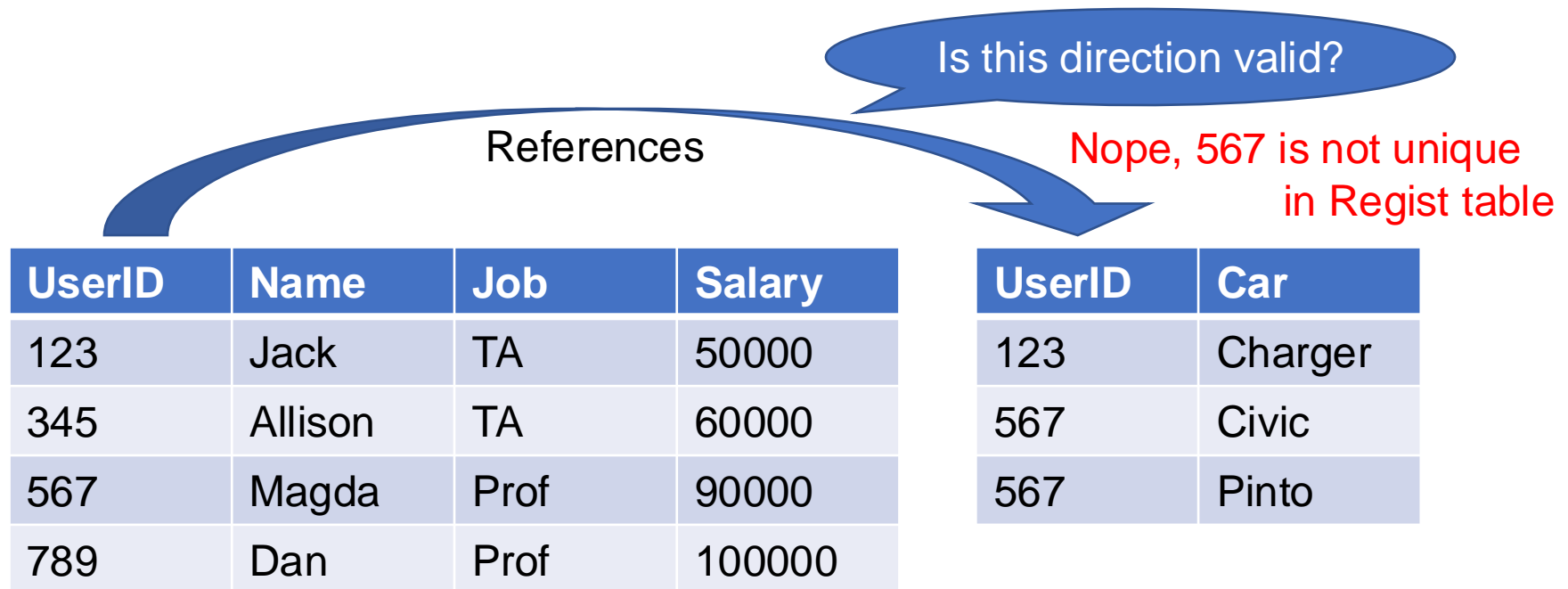
A **Foreign Key** is one or more attributes that uniquely identify a row in *another table*.



Foreign Keys

Foreign Key

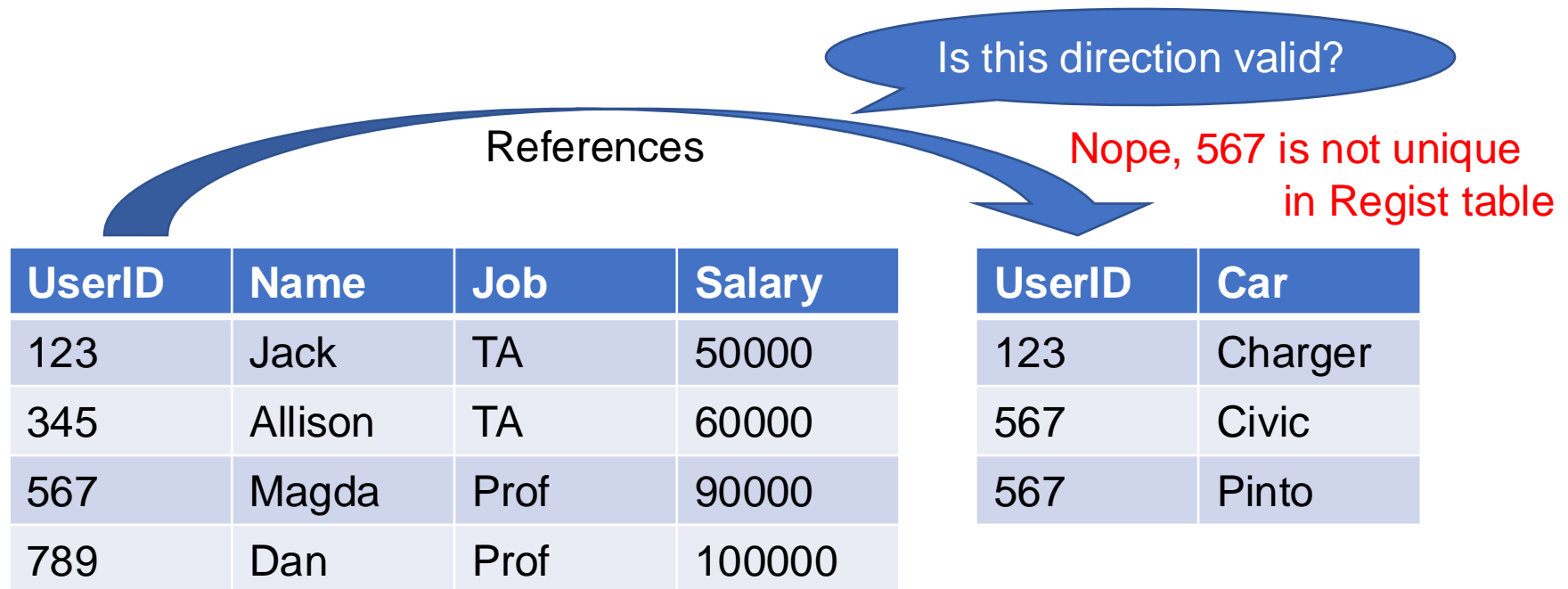
A **Foreign Key** is one or more attributes that uniquely identify a row in *another table*.



Foreign Keys

Foreign Key

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



Foreign keys must reference (point to) a unique attribute, almost always a primary key

Foreign Keys

We add foreign key declaration in the same way as a primary key.

```
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

We add foreign key declaration in the same way as a primary key.

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

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

Payroll(UserId, Name, Job, Salary)

Regist(UserId, Car)



Foreign Keys

We add foreign key declaration in the same way as a primary key.

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

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

or, when attribute
name is the key:

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

Payroll(UserId, Name, Job, Salary)

Regist(UserId, Car)



Foreign Keys

Can also put foreign key declaration on a new line, need to do this for multiple attributes

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

```
CREATE TABLE Regist (  
  Name TEXT,  
  Job TEXT,  
  Car TEXT,  
  FOREIGN KEY (Name, Job)  
    REFERENCES Payroll);
```

Payroll(Name, Job, Salary)

Regist(Name, Job, Car)



Recap for Today

- SELECT-FROM-WHERE
- DISTINCT, ORDER-BY
- Semantics: for-each
- CREATE TABLE, KEY, FOREIGN KEY

Next lecture(s): SQL continued