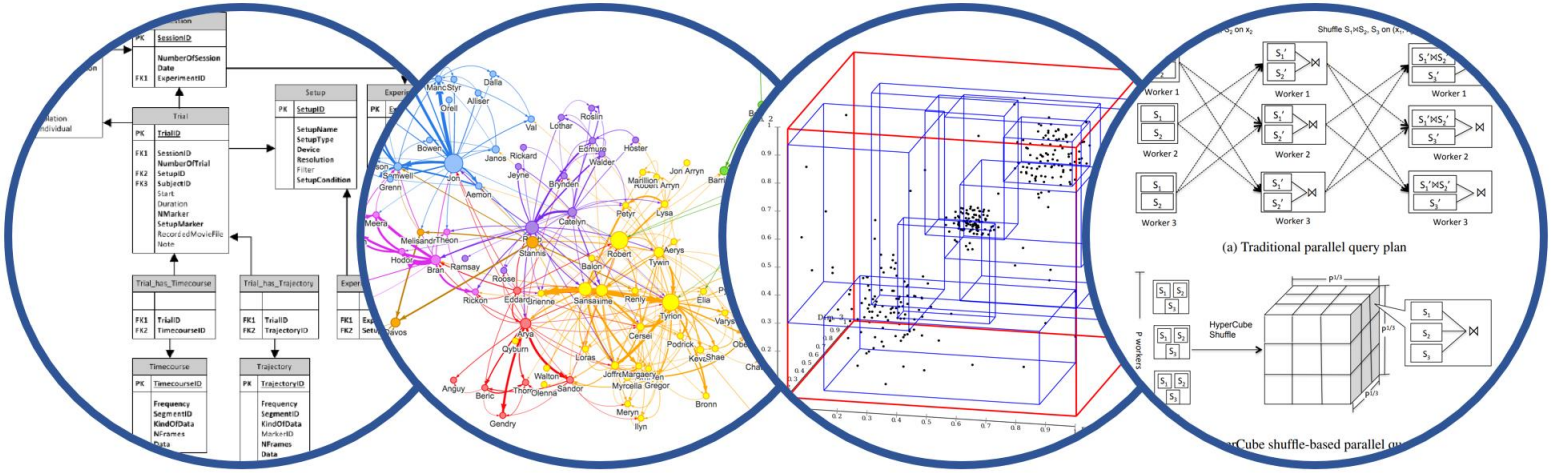


CSE 344: Intro to Data Management

SQL Basics

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

Recap – The Relational Model

Table/ Relation

Columns/Attributes/Fields

Rows/ Tuples/ Records

user_id	name	job	salary
123	Jack	TA	50000
345	Allison	TA	60000
567	Magda	Prof	90000
789	Dan	Prof	100000

A Few Lies

- Relations are sets, so are inherently unordered.
  - But tables are presented in an order?

- Relations are sets, so they have no duplicates.
  - But tables can have duplicates?

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

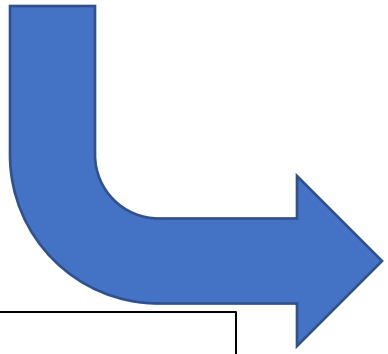
Basic SQL query: SELECT-FROM-WHERE

```
SELECT *
   FROM payroll;
```

Basic SQL query: SELECT-FROM-WHERE

payroll

user_id	name	job	salary
123	Jack	TA	50000
345	Allison	TA	60000
567	Magda	Prof	90000
789	Dan	Prof	100000

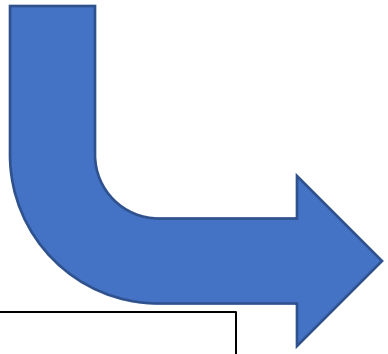


```
SELECT *
  FROM payroll;
```

Basic SQL query: SELECT-FROM-WHERE

payroll

user_id	name	job	salary
123	Jack	TA	50000
345	Allison	TA	60000
567	Magda	Prof	90000
789	Dan	Prof	100000



```
SELECT *
  FROM payroll;
```

user_id	name	job	salary
123	Jack	TA	50000
345	Allison	TA	60000
567	Magda	Prof	90000
789	Dan	Prof	100000

Basic SQL query: SELECT-FROM-WHERE

payroll

user_id	name	job	salary
123	Jack	TA	50000
345	Allison	TA	60000
567	Magda	Prof	90000
789	Dan	Prof	100000

```
SELECT p.name, p.user_id
  FROM payroll as p
 WHERE p.job = 'TA';
```

Basic SQL query: SELECT-FROM-WHERE

payroll

user_id	name	job	salary
123	Jack	TA	50000
345	Allison	TA	60000
567	Magda	Prof	90000
789	Dan	Prof	100000


```
SELECT p.name, p.user_id
  FROM payroll as p
 WHERE p.job = 'TA';
```

Basic SQL query: SELECT-FROM-WHERE

payroll

user_id	name	job	salary
123	Jack	TA	50000
345	Allison	TA	60000
567	Magda	Prof	90000
789	Dan	Prof	100000


```
SELECT p.name, p.user_id
  FROM payroll as p
 WHERE p.job = 'TA';
```

Basic SQL query: SELECT-FROM-WHERE

payroll

user_id	name	job	salary
123	Jack	TA	50000
345	Allison	TA	60000
567	Magda	Prof	90000
789	Dan	Prof	100000

```
SELECT p.name, p.user_id
  FROM payroll as p
 WHERE p.job = 'TA';
```

SELECT attributes

FROM table(s)

WHERE filter condition

Basic SQL query: SELECT-FROM-WHERE

payroll

user_id	name	job	salary
123	Jack	TA	50000
345	Allison	TA	60000
567	Magda	Prof	90000
789	Dan	Prof	100000

?

```
SELECT p.name, p.user_id
  FROM payroll as p
 WHERE p.job = 'TA';
```

Basic SQL query: SELECT-FROM-WHERE

payroll

user_id	name	job	salary
123	Jack	TA	50000
345	Allison	TA	60000
567	Magda	Prof	90000
789	Dan	Prof	100000

name	user_id
Jack	123
Allison	345

```
SELECT p.name, p.user_id
  FROM payroll as p
 WHERE p.job = 'TA';
```

Basic SQL query: SELECT-FROM-WHERE

payroll

user_id	name	job	salary
123	Jack	TA	50000
345	Allison	TA	60000
567	Magda	Prof	90000
789	Dan	Prof	100000

name	user_id
Jack	123
Allison	345

```
SELECT p.name, p.user_id
  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

user_id	name	job	salary
123	Jack	TA	50000
345	Allison	TA	60000
567	Magda	Prof	90000
789	Dan	Prof	100000

```
SELECT p.name, p.user_id
  FROM payroll as p
 WHERE p.job = 'TA';
```

Q: What exactly does this return?

For-each semantics

payroll

user_id	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.user_id)
```

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

```
SELECT p.name, p.user_id
  FROM payroll as p
 WHERE p.job = 'TA';
```

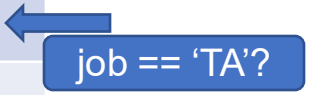
Q: What exactly does this return?

For-each semantics

payroll

user_id	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.user_id)
```




```
SELECT p.name, p.user_id
  FROM payroll as p
 WHERE p.job = 'TA';
```

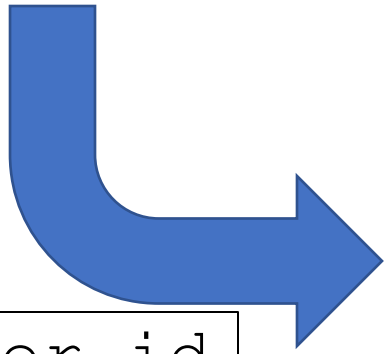
For-each semantics

payroll

user_id	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.user_id)
```

job == 'TA'?



name	user_id

```
SELECT p.name, p.user_id
  FROM payroll as p
 WHERE p.job = 'TA';
```

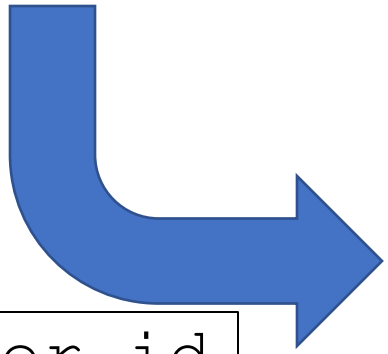
For-each semantics

payroll

user_id	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.user_id)
```

job == 'TA'?



name	user_id
Jack	123

```
SELECT p.name, p.user_id
  FROM payroll as p
 WHERE p.job = 'TA';
```

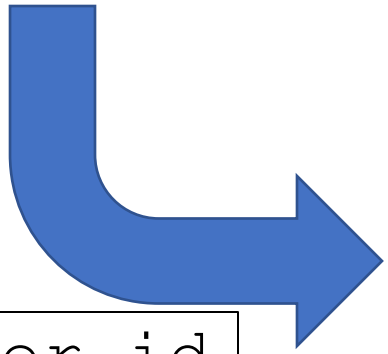
For-each semantics

payroll

user_id	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.user_id)
```

job == 'TA'?



name	user_id
Jack	123

```
SELECT p.name, p.user_id
  FROM payroll as p
 WHERE p.job = 'TA';
```

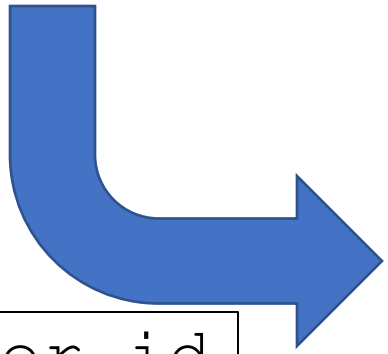
For-each semantics

payroll

user_id	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.user_id)
```

job == 'TA'?



name	user_id
Jack	123
Allison	345

```
SELECT p.name, p.user_id
  FROM payroll as p
 WHERE p.job = 'TA';
```

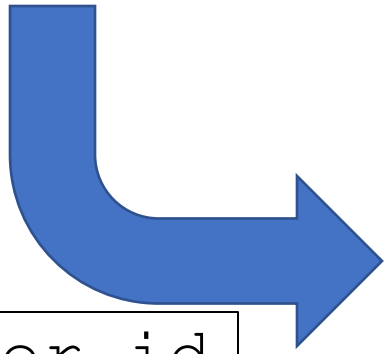
For-each semantics

payroll

user_id	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.user_id)
```

job == 'TA'?



name	user_id
Jack	123
Allison	345

```
SELECT p.name, p.user_id
  FROM payroll as p
 WHERE p.job = 'TA';
```

For-each semantics

payroll

user_id	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.user_id)
```

job == 'TA'?

name	user_id
Jack	123
Allison	345

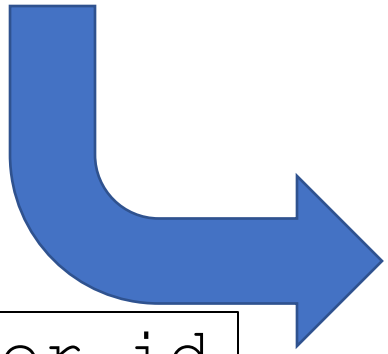
```
SELECT p.name, p.user_id
  FROM payroll as p
 WHERE p.job = 'TA';
```

For-each semantics

payroll

user_id	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.user_id)
```



Final output

name	user_id
Jack	123
Allison	345

```
SELECT p.name, p.user_id
  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.user_id
  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.user_id
  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.user_id
  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.user_id
  FROM payroll as p
 WHERE p.job = 'TA';
```

Another Semantics

Set-builder notation

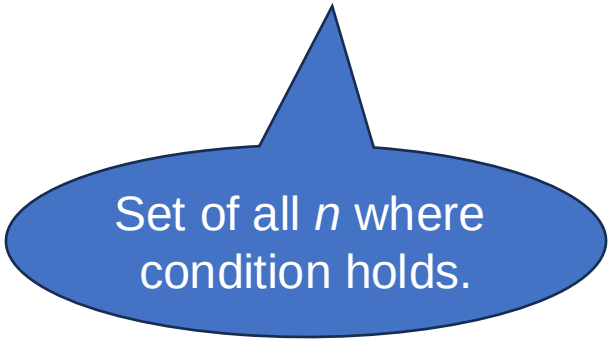
- Recall set-builder notation from CSE 311

$$\{ n \in \mathbb{N} \mid n \equiv 0 \pmod 2 \}$$

Set-builder notation

- Recall set-builder notation from CSE 311

$$\{ n \in \mathbb{N} \mid n \equiv 0 \pmod 2 \}$$

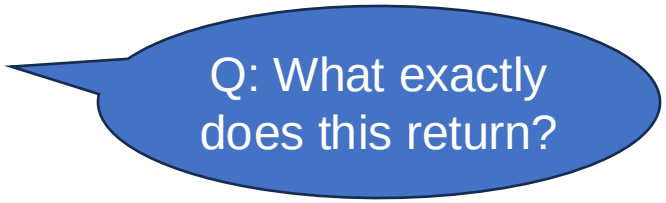


Set-builder semantics

payroll

user_id	name	job	salary
123	Jack	TA	50000
345	Allison	TA	60000
567	Magda	Prof	90000
789	Dan	Prof	100000

```
SELECT p.name, p.user_id
  FROM payroll as p
 WHERE p.job = 'TA';
```



Set-builder semantics

payroll

user_id	name	job	salary
123	Jack	TA	50000
345	Allison	TA	60000
567	Magda	Prof	90000
789	Dan	Prof	100000

A: exactly this set!

$$\{ (n, u) \mid (u, n, j, s) \in \text{payroll} \wedge j = \text{“TA”} \}$$

```
SELECT p.name, p.user_id
  FROM payroll as p
 WHERE p.job = 'TA';
```

Q: What exactly does this return?

ORDER-BY and DISTINCT

and other lies

Order By

```
SELECT *
FROM payroll
ORDER BY name;
```

payroll

user_id	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

user_id	name	job	salary
123	Jack	TA	50000
345	Allison	TA	60000
567	Magda	Prof	90000
789	Dan	Prof	100000

user_id	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

user_id	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

user_id	name	job	salary
123	Jack	TA	50000
345	Allison	TA	60000
567	Magda	Prof	90000
789	Dan	Prof	100000

user_id	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

user_id	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

user_id	name	job	salary
123	Jack	TA	50000
345	Allison	TA	60000
567	Magda	Prof	90000
789	Dan	Prof	100000

user_id	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

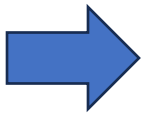
user_id	name	job	salary
123	Jack	TA	50000
345	Allison	TA	60000
567	Magda	Prof	90000
789	Dan	Prof	100000

Distinct

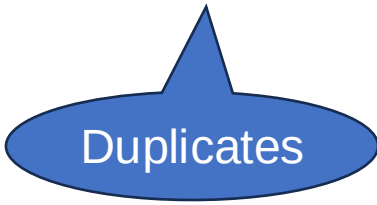
```
SELECT job
FROM payroll
```

payroll

user_id	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

user_id	name	job	salary
123	Jack	TA	50000
345	Allison	TA	60000
567	Magda	Prof	90000
789	Dan	Prof	100000

job
TA
TA
Prof
Prof

Duplicates

Bag semantics

```
SELECT DISTINCT job
FROM payroll
```

job
TA
Prof

No duplicates

Set semantics

Tables in SQL

Create Table Statement

payroll(user_id, name, job, salary)

```
CREATE TABLE payroll (
  user_id 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(user_id, name, job, salary)

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

user_id	name	job	salary
123	Jack	TA	50000
345	Allison	TA	60000
567	Magda	Prof	90000
789	Dan	Prof	100000

Insert

payroll(user_id, name, job, salary)

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

user_id	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

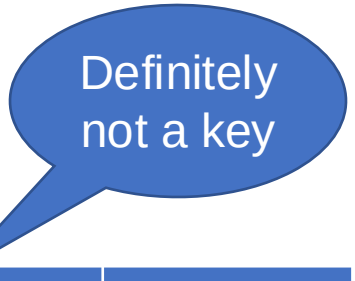
A **Key** is one or more attributes that **uniquely** identify a row.

user_id	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.



user_id	name	job	salary
123	Jack	TA	50000
345	Allison	TA	60000
567	Magda	Prof	90000
789	Dan	Prof	100000

Keys

Key

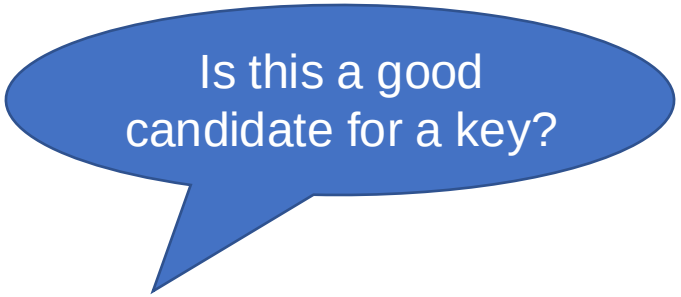
A **Key** is one or more attributes that **uniquely** identify a row.

user_id	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.

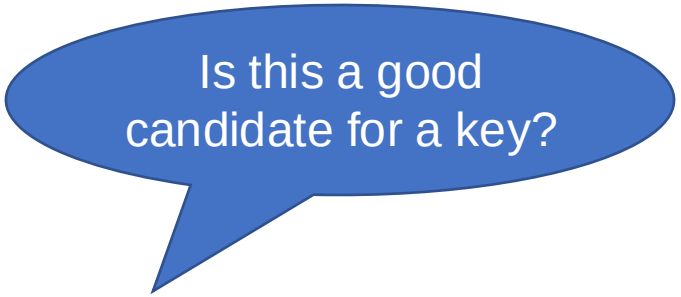


user_id	name	job	salary
123	Jack	TA	50000
345	Allison	TA	60000
567	Magda	Prof	90000
789	Dan	Prof	100000

Keys

Key

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



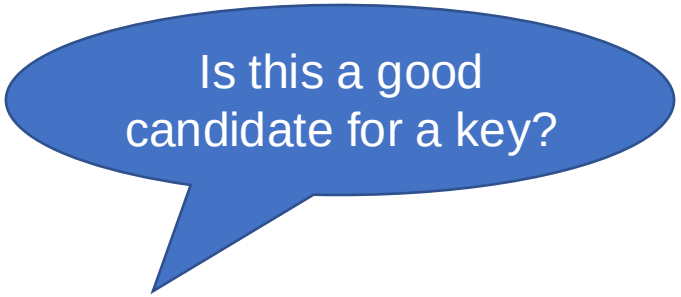
user_id	name	job	salary
123	Jack	TA	50000
345	Allison	TA	60000
567	Magda	Prof	90000
789	Dan	Prof	100000
913	Peter	TA	60000

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



user_id	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 (
  user_id INT,
  name TEXT,
  job TEXT,
  salary INT
);
```

payroll(user_id, name, job, salary)

Keys

```
CREATE TABLE payroll (
  user_id INT PRIMARY KEY,
  name TEXT,
  job TEXT,
  salary INT
);
```

payroll(user_id, name, job, salary)

Keys

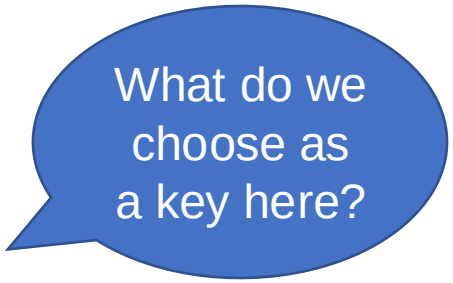
```
CREATE TABLE payroll (
  user_id INT,
  name TEXT,
  job TEXT,
  salary INT,
  PRIMARY KEY (user_id)
);
```

Can also define the PK at the end

payroll(<u>user_id</u>, 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.



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.

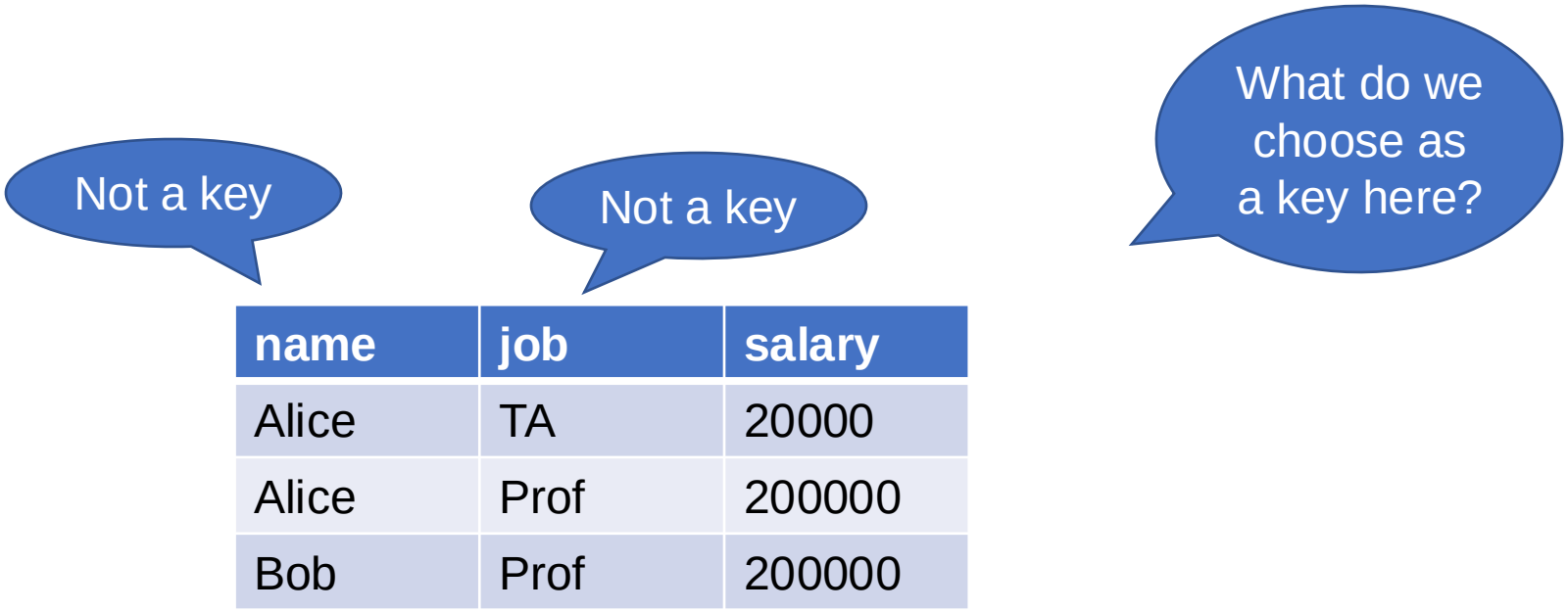
Not a key

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

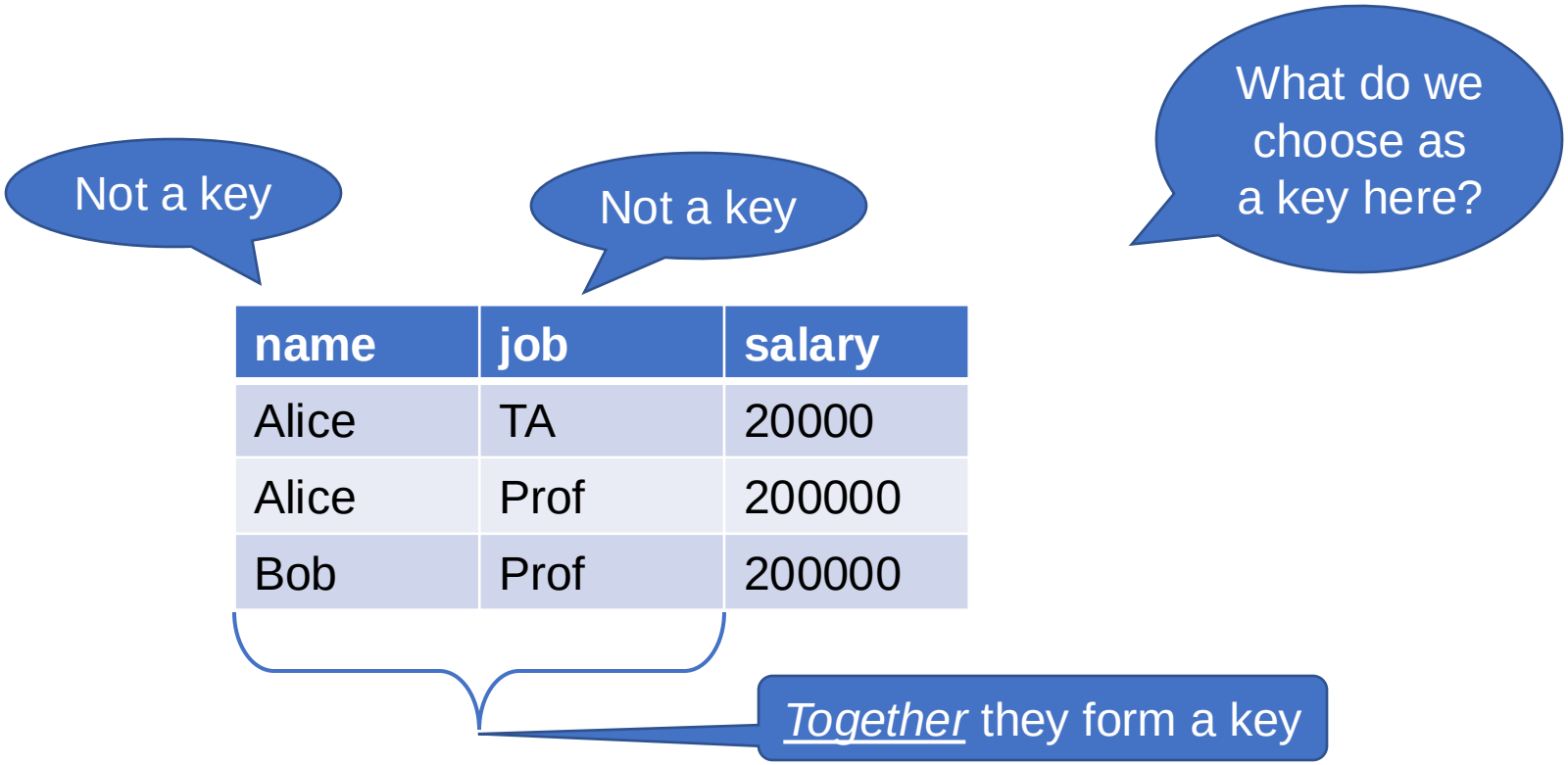
Sometimes no single attribute is unique, but combinations of attributes are a unique key for the table.



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

user_id	name	job	salary
123	Jack	TA	50000
345	Allison	TA	60000
567	Magda	Prof	90000
789	Dan	Prof	100000

regist

user_id	car
123	Charger
567	Civic
567	Pinto

Foreign Keys

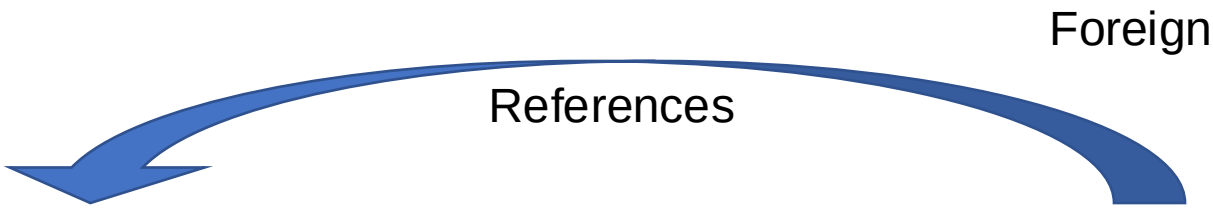
- Databases can hold multiple tables
- How do we capture relationships *between* tables?

user_id	name	job	salary
123	Jack	TA	50000
345	Allison	TA	60000
567	Magda	Prof	90000
789	Dan	Prof	100000

user_id	car
123	Charger
567	Civic
567	Pinto

Foreign Keys

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



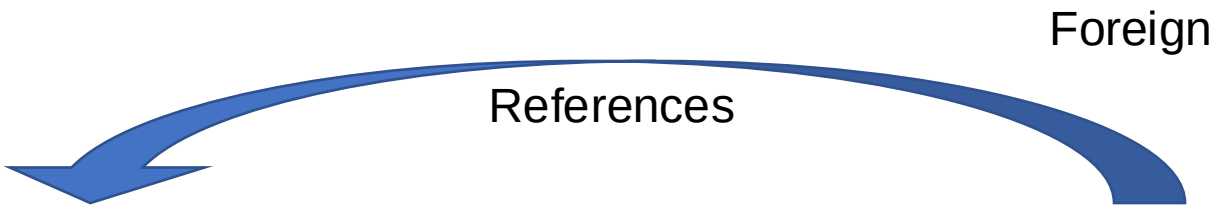
user_id	name	job	salary
123	Jack	TA	50000
345	Allison	TA	60000
567	Magda	Prof	90000
789	Dan	Prof	100000

user_id	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*.



user_id	name	job	salary
123	Jack	TA	50000
345	Allison	TA	60000
567	Magda	Prof	90000
789	Dan	Prof	100000

user_id	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*.

user_id	name	job	salary
123	Jack	TA	50000
345	Allison	TA	60000
567	Magda	Prof	90000
789	Dan	Prof	100000

user_id	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*.

user_id	name	job	salary
123	Jack	TA	50000
345	Allison	TA	60000
567	Magda	Prof	90000
789	Dan	Prof	100000

user_id	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*.

user_id	name	job	salary
123	Jack	TA	50000
345	Allison	TA	60000
567	Magda	Prof	90000
789	Dan	Prof	100000

user_id	car
123	Charger
567	Civic
567	Pinto

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 (
  user_id INT PRIMARY KEY,
  name TEXT,
  job TEXT,
  salary INT);
```

```
CREATE TABLE regist (
  user_id INT,
  car TEXT);
```

payroll(user_id, name, job, salary)

regist(user_id, car)

Foreign Keys

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

```
CREATE TABLE payroll (
  user_id INT PRIMARY KEY,
  name TEXT,
  job TEXT,
  salary INT);
```

```
CREATE TABLE regist (
  user_id INT
      REFERENCES payroll(user_id),
  car TEXT);
```

payroll(user_id, name, job, salary)

regist(user_id, car)

Foreign Keys

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

```
CREATE TABLE payroll (
  user_id INT PRIMARY KEY,
  name TEXT,
  job TEXT,
  salary INT);
```

```
CREATE TABLE regist (
  user_id INT
      REFERENCES payroll(user_id),
  car TEXT);
```

or, when attribute name is the key:

```
CREATE TABLE regist (
  user_id INT REFERENCES payroll,
  car TEXT);
```

payroll(user_id, name, job, salary)

regist(user_id, 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);
```

Recap for Today

- SELECT-FROM-WHERE
- DISTINCT, ORDER-BY
- Semantics: for-each, set-builder
- CREATE TABLE, INSERT
- Keys, PRIMARY KEY, FOREIGN KEY