Using a DBMS

Shan-Hung Wu & DataLab
CS, NTHU

DBMS ≠ Database

- A database is a collection of your data stored in a computer
- A DBMS (DataBase Management System) is a software that manages databases

Outline

- Main Features of a DBMS
- Data Models

Outline

- Main Features of a DBMS
- Data Models

Why not file systems?

Advantages of a Database System

- It answers ***queries*** fast
  - – E.g., among all posts, find those written by Bob and contain word “db”
- Groups modifications into ***transactions*** such that either all or nothing happens
  - – E.g., money transfer
- Recovers from crash
  - – Modifications are logged
  - – No corrupt data after recovery

Advantages of a Database System

- It answers ***queries*** fast
  - E.g., among all posts, find those written by Bob and contain word “db”
- Groups modifications into ***transactions*** such that either all or nothing happens
  - E.g., money transfer
- Recovers from crash
  - Modifications are logged
  - No corrupt data after recovery

Queries

Q: find ID and text of all pages written by Bob and containing word “db”

Step1: structure data using ***tables***

users

id	name	karma
729	Bob	35
730	John	0

Column/field

posts

id	text	ts	authorId
33981	‘Hello DB!’	1493897351	729
33982	‘Show me code’	1493854323	812

Row/recor
d

Queries

Q: find ID and text of all pages written by Bob and containing word “db”

Step2:

```
SELECT p.id, p.text
FROM posts AS p, users AS u
WHERE u.id = p.authorId
      AND u.name='Bob'
      AND p.text ILIKE '%db%';
```

users

id	name	karma
729	Bob	35
730	John	0

posts

id	text	ts	authorId
33981	‘Hello DB!’	1493897351	729
33982	‘Show me code’	1493904323	812

How Is a Query Answered?

```
SELECT p.id, p.text
FROM posts AS p, users AS u
WHERE u.id = p.authorId
      AND u.name='Bob'
      AND p.text ILIKE '%db%';
```

(p, u)

p.id	p.text	p.ts	p.authorId	u.id	u.name	u.karma
33981	‘Hello DB!’	…	729	729	Bob	35
33981	‘Hello DB!’	…	729	730	John	0
33982	‘Show me code’	…	812	729	Bob	35
33982	‘Show me code’	…	812	730	John	0

p

id	text	ts	authorId
33981	‘Hello DB!’	…	729
33982	‘Show me code’	…	812

u

id	name	karma
729	Bob	35
730	John	0

How Is a Query Answered?

```
SELECT p.id, p.text
FROM posts AS p, users AS u
WHERE u.id = p.authorId
      AND u.name='Bob'
      AND p.text ILIKE '%db%';
```

where(p, u)

p.id	p.text	p.ts	p.authorId	u.id	u.name	u.karma
33981	‘Hello DB!’	…	729	729	Bob	35

(p, u)

p.id	p.text	p.ts	p.authorId	u.id	u.name	u.karma
33981	‘Hello DB!’	…	729	729	Bob	35
33981	‘Hello DB!’	…	729	730	John	0
33982	‘Show me code’	…	812	729	Bob	35
33982	‘Show me code’	…	812	730	John	0

How Is a Query Answered?

```
SELECT p.id, p.text
FROM posts AS p, users AS u
WHERE u.id = p.authorId
      AND u.name='Bob'
      AND p.text ILIKE '%db%';
```

select(where(p, u))

p.id	p.text
33981	‘Hello DB!’

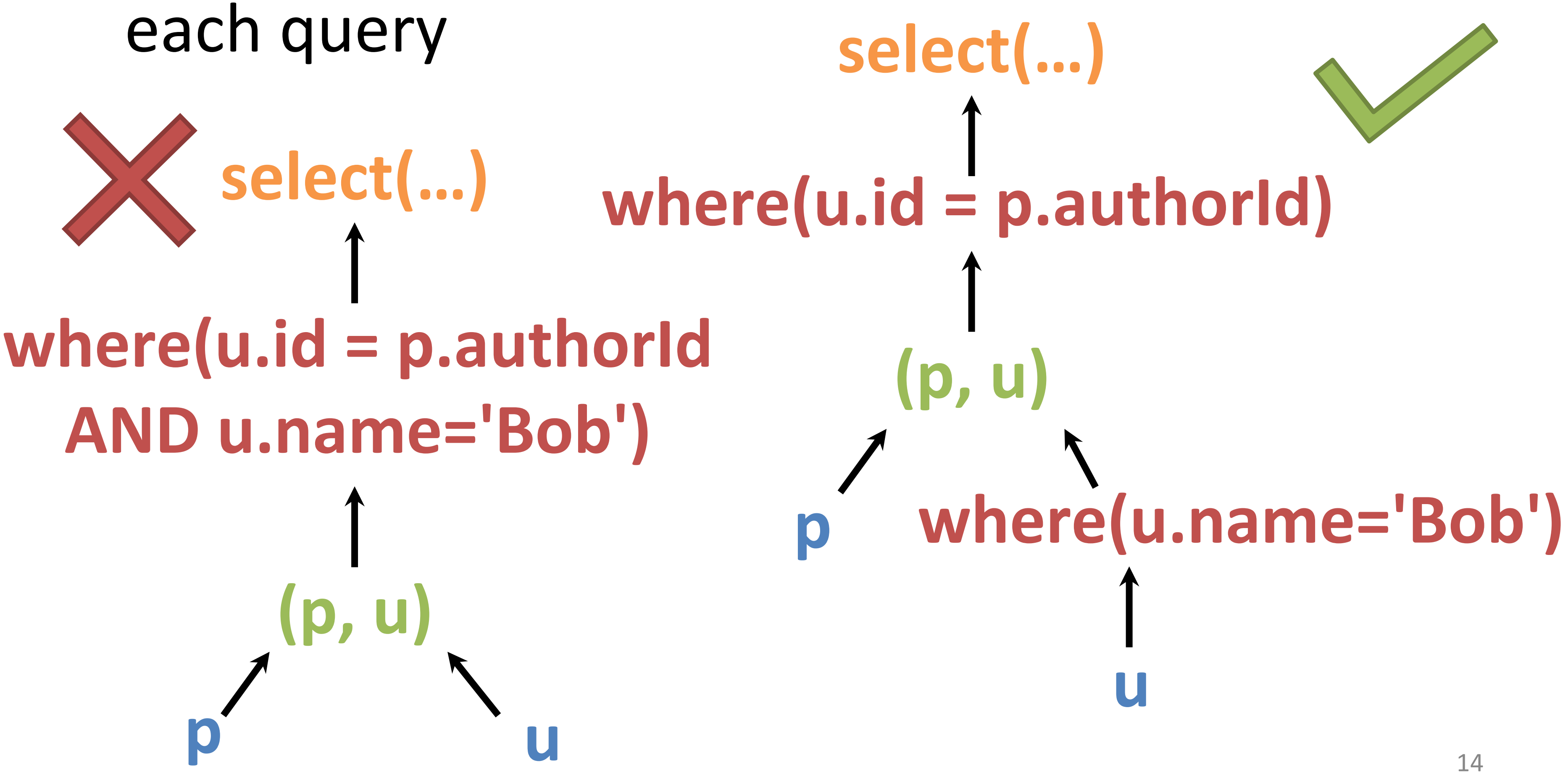
where(p, u)

p.id	p.text	p.ts	p.authorId	u.id	u.name	u.karma
33981	‘Hello DB!’	…	729	729	Bob	35

Why fast?

Query Optimization

- ***Planning***: DBMS finds the best ***plan tree*** for each query



✗

select(…)

where(u.id = p.authorId AND u.name='Bob')

(p, u)

p

u

✓

select(…)

where(u.id = p.authorId)

(p, u)

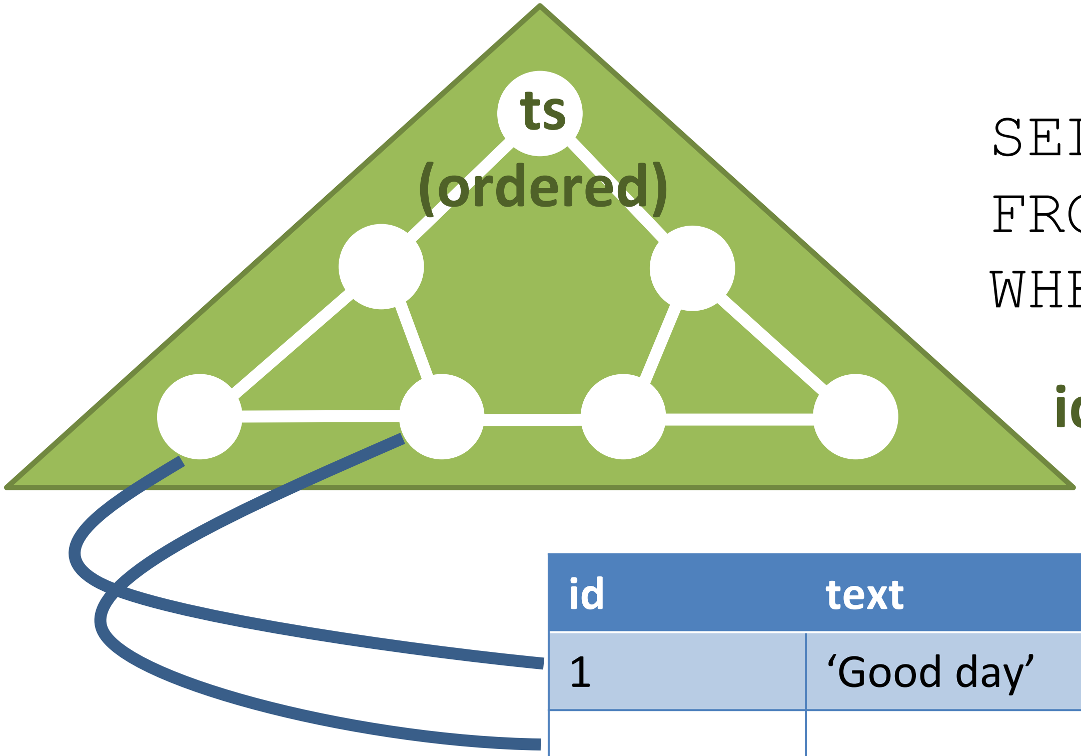
p

where(u.name='Bob')

u

Query Optimization

- ***Indexing***: creates a search tree for column(s)



```
SELECT text
FROM posts
WHERE ts > 1493897220;
```

posts

id	text	ts	authorId
1	‘Good day’	1493880220	664
…	…	…	…
33981	‘Hello DB!’	1493897351	729
33982	‘Show me code’	1493904323	812

Advantages of a Database System

- It answers ***queries*** fast
  - E.g., among all posts, find those written by Bob and contain word “db”
- Groups modifications into ***transactions*** such that either all or nothing happens
  - E.g., money transfer
- Recovers from crash
  - Modifications are logged
  - No corrupt data after recovery

Transactions I

- Each query, by default, is placed in a ***transaction*** (***tx*** for short) automatically

```
BEGIN;
  SELECT ...; -- query
COMMIT;
```

Transactions II

- Can group multiple queries in a tx
  - *All or nothing* takes effect
- E.g., karma transfer

users

id	name	karma
729	Bob	35
730	John	0

```
BEGIN;
  UPDATE users
  SET karma = karma - 10
  WHERE name='Bob';

  UPDATE users
  SET karma = karma + 10
  WHERE name='John';
COMMIT;
```

ACID Guarantees

- ***Atomicity***
  - Operation are all or none in effect
- ***Consistency***
  - Data are correct after each tx commits
  - E.g., `posts.authorId` must be a valid `users.id`
- ***Isolation***
  - Concurrent txs = serial txs (in some order)
- ***Durability***
  - Changes will not be lost after a tx commits (even after crashes)

Outline

- Main Features of a DBMS
- Data Models

Why model data as ***tables***?

users

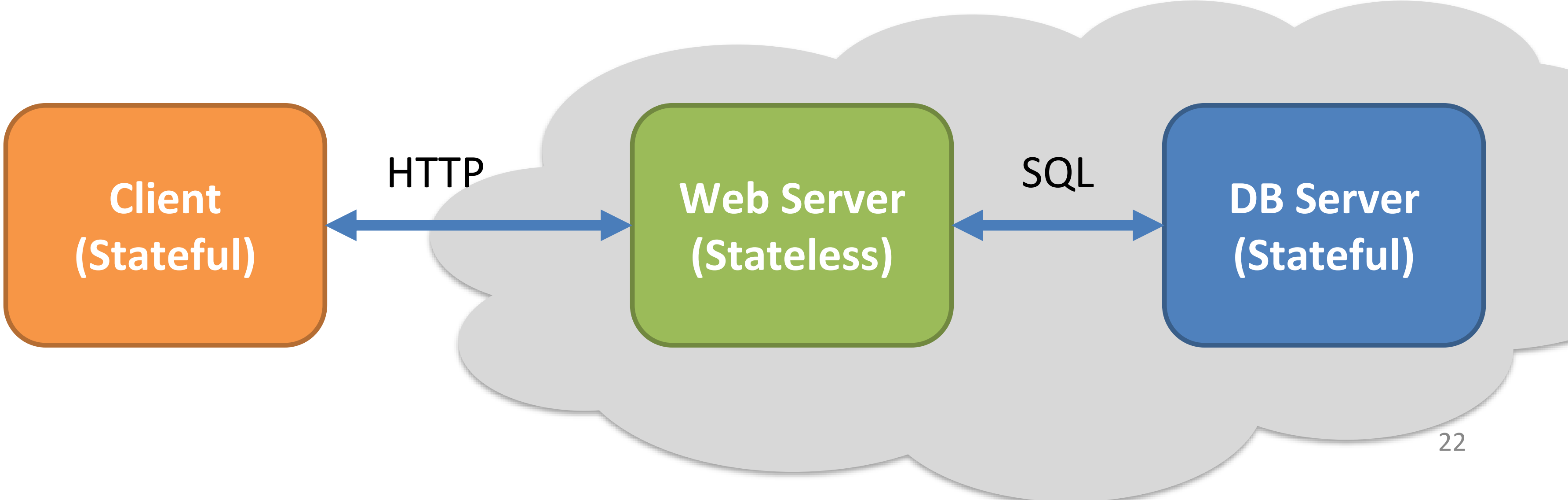
id	name	karma
729	Bob	35
730	John	0

posts

id	text	ts	authorId
33981	‘Hello DB!’	1493897351	729
33982	‘Show me code’	1493904323	812

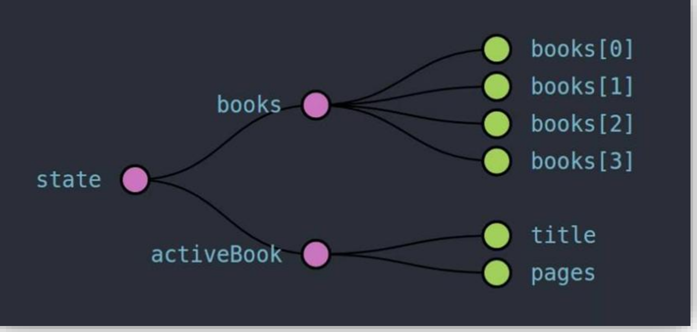
Storing Data

- Let’s say, you have data/states in memory to store
- What do states look like?
  - Objects
  - References to objects
- Objects formatted by classes you defined
- Can we store these objects and references directly?



Data Models

- Definition: A ***data model*** is a framework for describing the structure of databases in a DBMS
- Common data models at client side:
  - Tree model
- Common data models at server side:
  - ***ER model*** and ***relational model***
- A DBMS supporting the relational model is called the relational DBMS



Tree Model

- At client side, data are usually stored as ***trees***

```
{ // state of client 1
  name: 'Bob',
  karma: 32,
  posts: [...],
  friends: [{
    name: 'Alice',
    karma: 10
  }, {
    name: 'John',
    karma: 17
  }, ...],
  ...
}
```

```
{ // state of client 2
  name: 'Alice',
  karma: 10,
  posts: [...],
  friends: [{
    name: 'Bob',
    karma: 32
  }, {
    name: 'John',
    karma: 17
  }, ...],
  ...
}
```

Problems at Server Side

- Space complexity: large ***redundancy***

```
{ // state of a client 1
  name: 'Bob',
  karma: 35,
  posts: [...],
  friends: [{
    name: 'Alice',
    karma: 10
  }, {
    name: 'John',
    karma: 17
  }, ...],
  ...
}
```

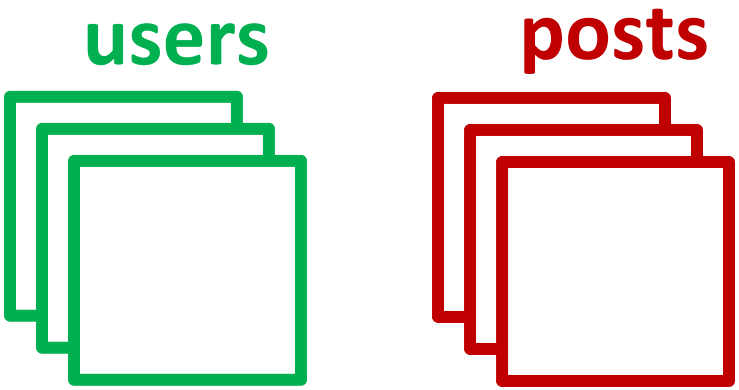
```
{ // state of a client 2
  name: 'Alice',
  karma: 10,
  posts: [...],
  friends: [{
    name: 'Bob',
    karma: 35
  }, {
    name: 'John',
    karma: 17
  }, ...],
  ...
}
```

Speed: slow update

Data Modeling at Server Side

1. Identify ***entity groups/classes***
   - Each class represents an “atomic” part of the data
2. Store entities of the same class in a ***table***
   - A rows/record denotes an entity
   - A column/field denote an attribute (e.g., “name”)
3. Define ***primary keys*** for each table
   - Special column(s) that uniquely identifies an entity
   - E.g., “ID”

Identifying Entity Classes



```
{ // state of a client 1
  name: 'Bob',
  karma: 32,
  posts: [...],
  friends: [{
    name: 'Alice',
    karma: 10
  }, {
    name: 'John'
    karma: 17
  }, ...],
  ...
}
```

```
{ // state of a client 2
  name: 'Alice',
  karma: 10,
  posts: [...],
  friends: [{
    name: 'Bob',
    karma: 32
  }, {
    name: 'John'
    karma: 17
  }, ...],
  ...
}
```

One Table per Entity Class

users

id	name	karma
729	Bob	35
730	John	0

posts

id	text
33981	‘Hello DB!’
33982	‘Show me code’

- No redundancy
- No repeated update

Wait, relationship is missing!

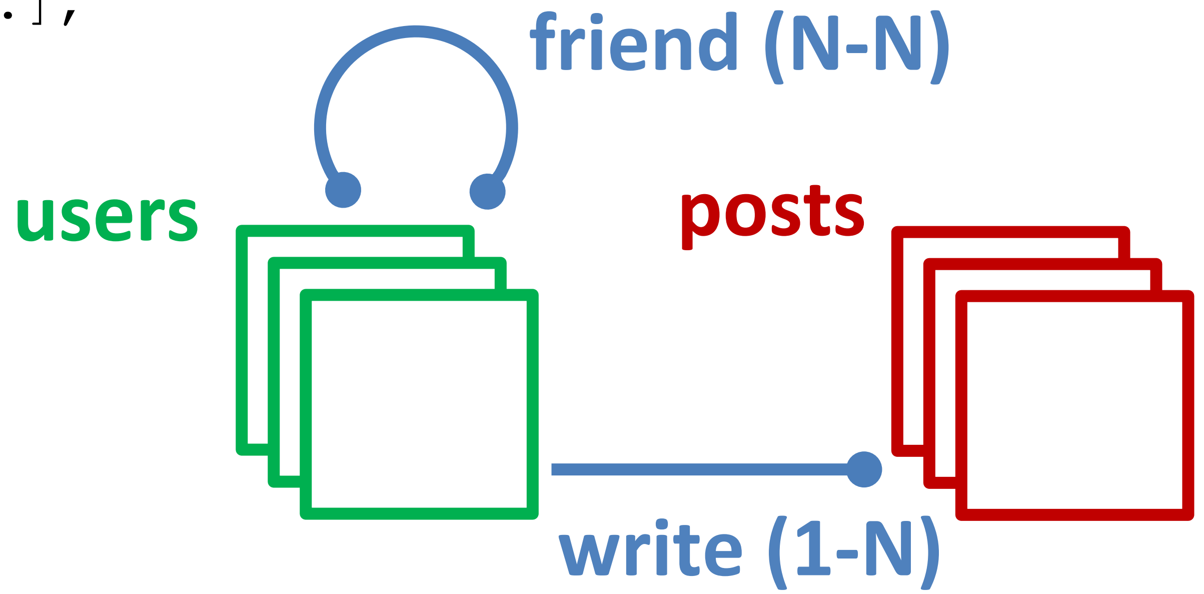
Step1 (ER Model)

```
{ // state of a client 1
  name: 'Bob',
  karma: 32,
  posts: [...],
  friends: [{
    name: 'Alice',
    karma: 10
  }, {
    name: 'John'
    karma: 17
  }, ...],
  ...
}
```

friend

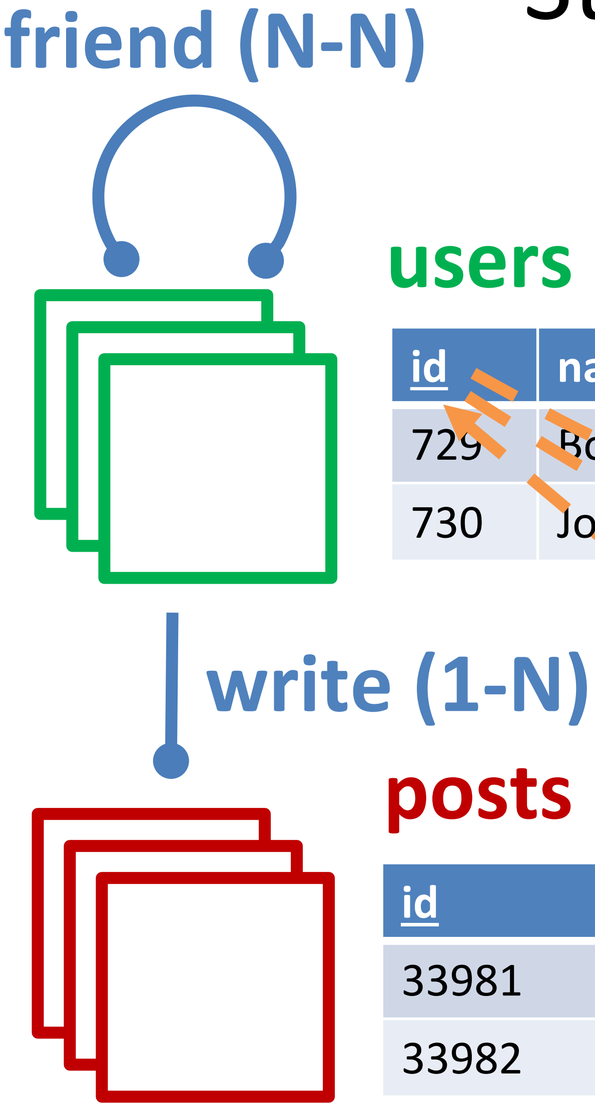
write

- Identify relationships between entities



Step 2 (Relational Model)

- Relationships as ***foreign keys***



users

id	name	karma
729	Bob	35
730	John	0

friend

uId1	uId2	since
729	730	14928063
729	882	14827432

foreign keys

posts

write

id	text	authorId	ts
33981	‘Hello DB!’	729	1493897351
33982	‘Show me code’	729	1493854323

Recap on Terminology

- Columns = fields = attributes
- Rows = records = tuples
- Tables = ***relations***

- Relational database: a collection of tables
  ≠ Relational DBMS
- ***Schema***: column definitions of tables in a database
  - Basically, the “look” of a database
  - Schema of a relation/table is fields and field types

Why ER Model?

- Allows thinking your data in OOP way
- ***Entity***
  - An object (or instance of a class)
  - With attributes
- ***Entity group/class***
  - A class
  - Must define the ID attribute for each entity
- ***Relationship*** between entities
  - References ("has-a" relationship)
  - Could be 1-1, 1-N, or N-N

Why Relational Model?

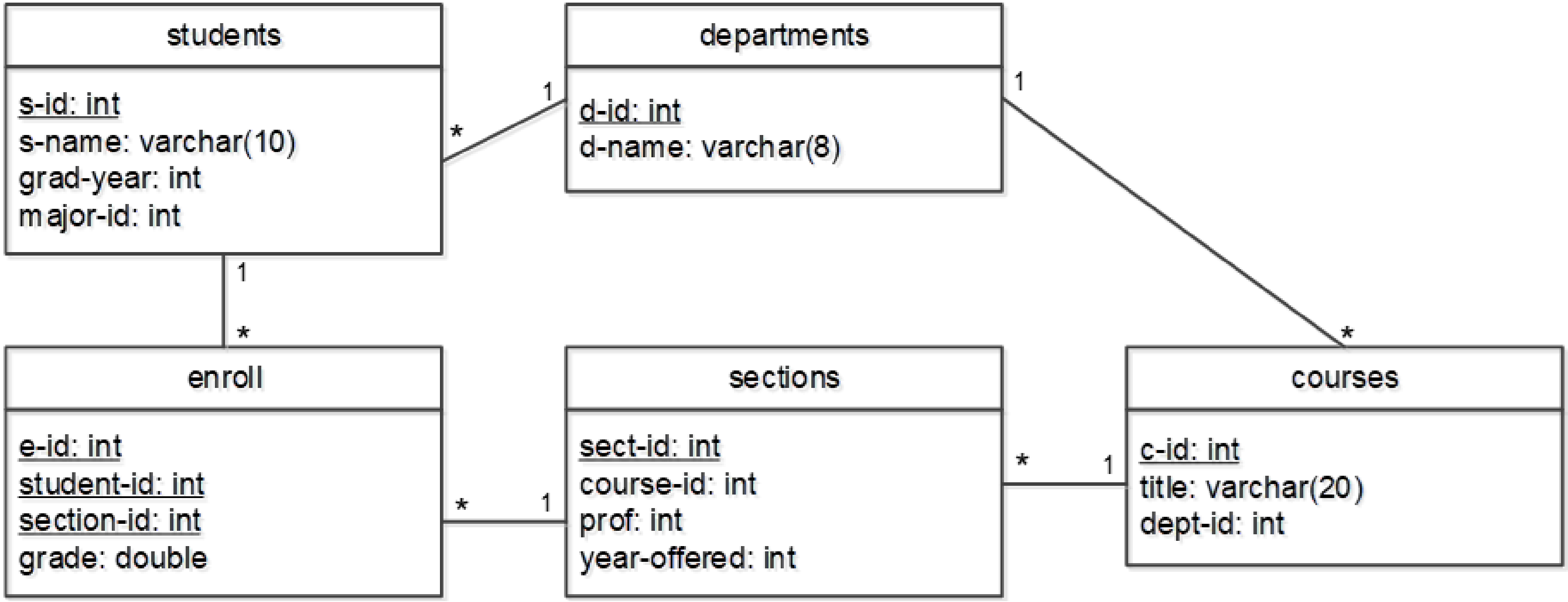
- Simplifies data management and query processing
    - Leverage the “arbitrary table join” in SQL queries
- ***Table/relations*** for all kinds of entity classes
- ***Primary/foreign keys*** for all kinds of relationships between entities
- Relational schema is logical
    - ***Not*** how your data stored physically
    - Vs. physical schema

Exercise: Student DB

- Storing course-enrollment info in a school
  - Each department has many students and offers different courses
  - Each courses can have multiple sections (e.g., 2018 spring, 2019 fall, etc.)
  - Each students can enroll in different sections

- Can you model data and draw a relational schema?

Exercise: Student DB



students

s-id: int
s-name: varchar(10)
grad-year: int
major-id: int

departments

d-id: int
d-name: varchar(8)

enroll

e-id: int
student-id: int
section-id: int
grade: double

sections

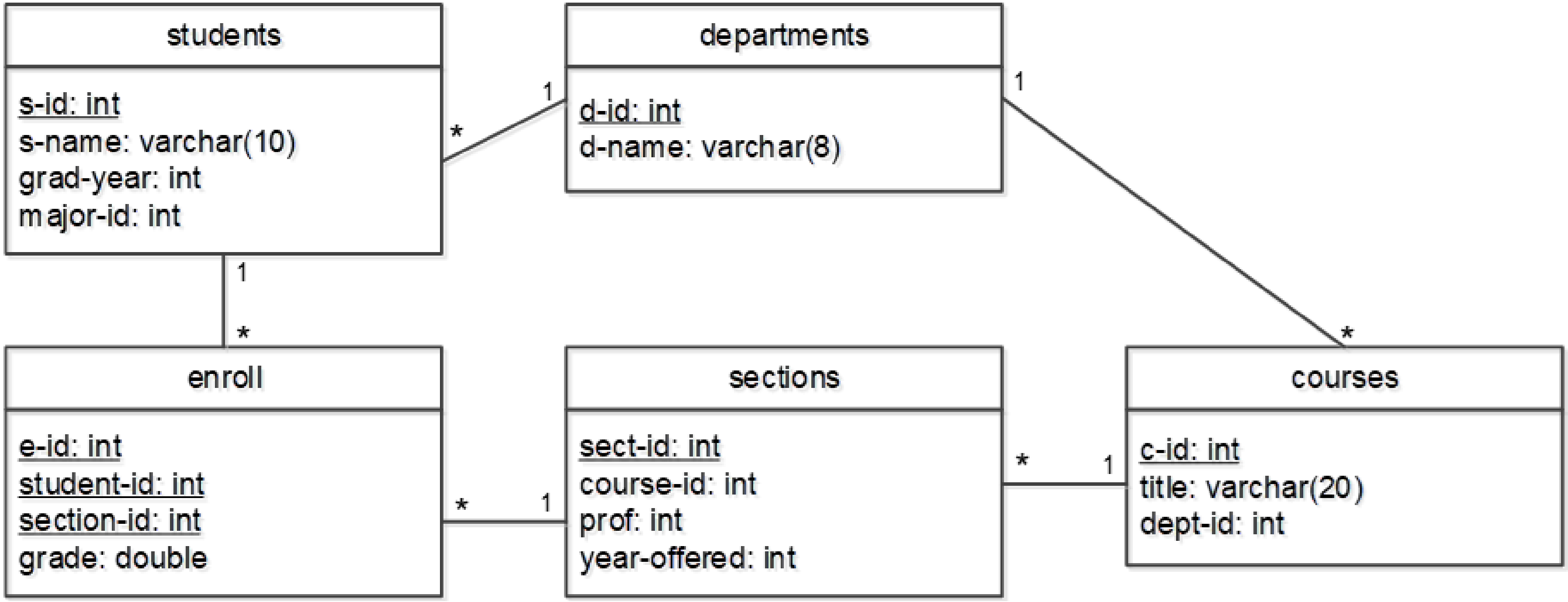
sect-id: int
course-id: int
prof: int
year-offered: int

courses

c-id: int
title: varchar(20)
dept-id: int

- Relation (table)
  - Realization of 1) an entity group via table; or 2) a relationship
  - ***Fields/attributes*** as columns
  - ***Records/tuples*** as rows

Exercise: Student DB



students

s-id: int
s-name: varchar(10)
grad-year: int
major-id: int

departments

d-id: int
d-name: varchar(8)

enroll

e-id: int
student-id: int
section-id: int
grade: double

sections

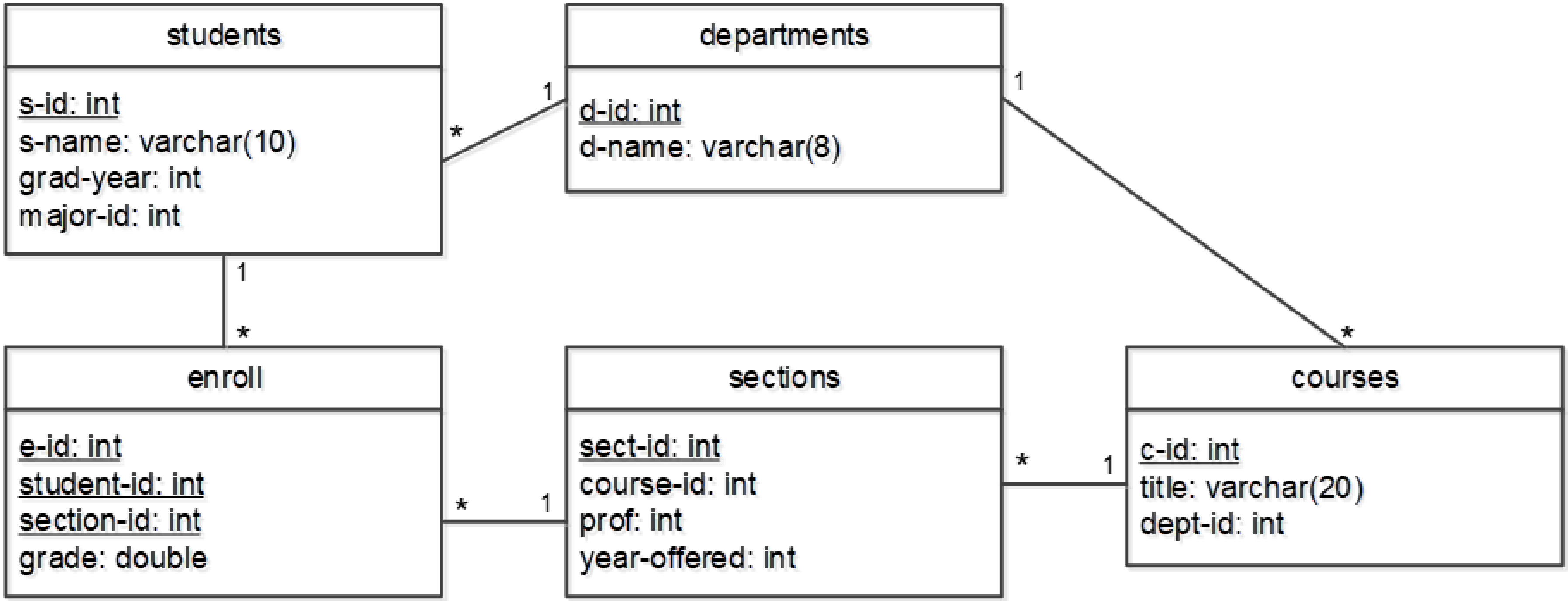
sect-id: int
course-id: int
prof: int
year-offered: int

courses

c-id: int
title: varchar(20)
dept-id: int

- ***Primary Key***
  - Realization of ID via a group of fields

Exercise: Student DB



students
s-id: int
s-name: varchar(10)
grad-year: int
major-id: int

departments
d-id: int
d-name: varchar(8)

enroll
e-id: int
student-id: int
section-id: int
grade: double

sections
sect-id: int
course-id: int
prof: int
year-offered: int

courses
c-id: int
title: varchar(20)
dept-id: int

- ***Foreign key***
  - Realization of relationship
  - A record can point to the primary key of the other record
  - Only 1-1 and 1-many
  - Intermediate relation is needed for many-many