By *Marina Barsky*

From E/R Diagrams to Relations

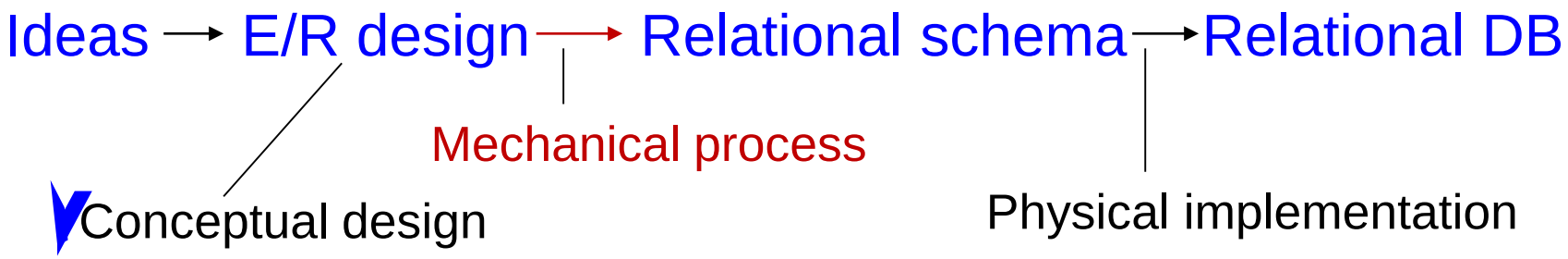
Lecture 3

Quick recap

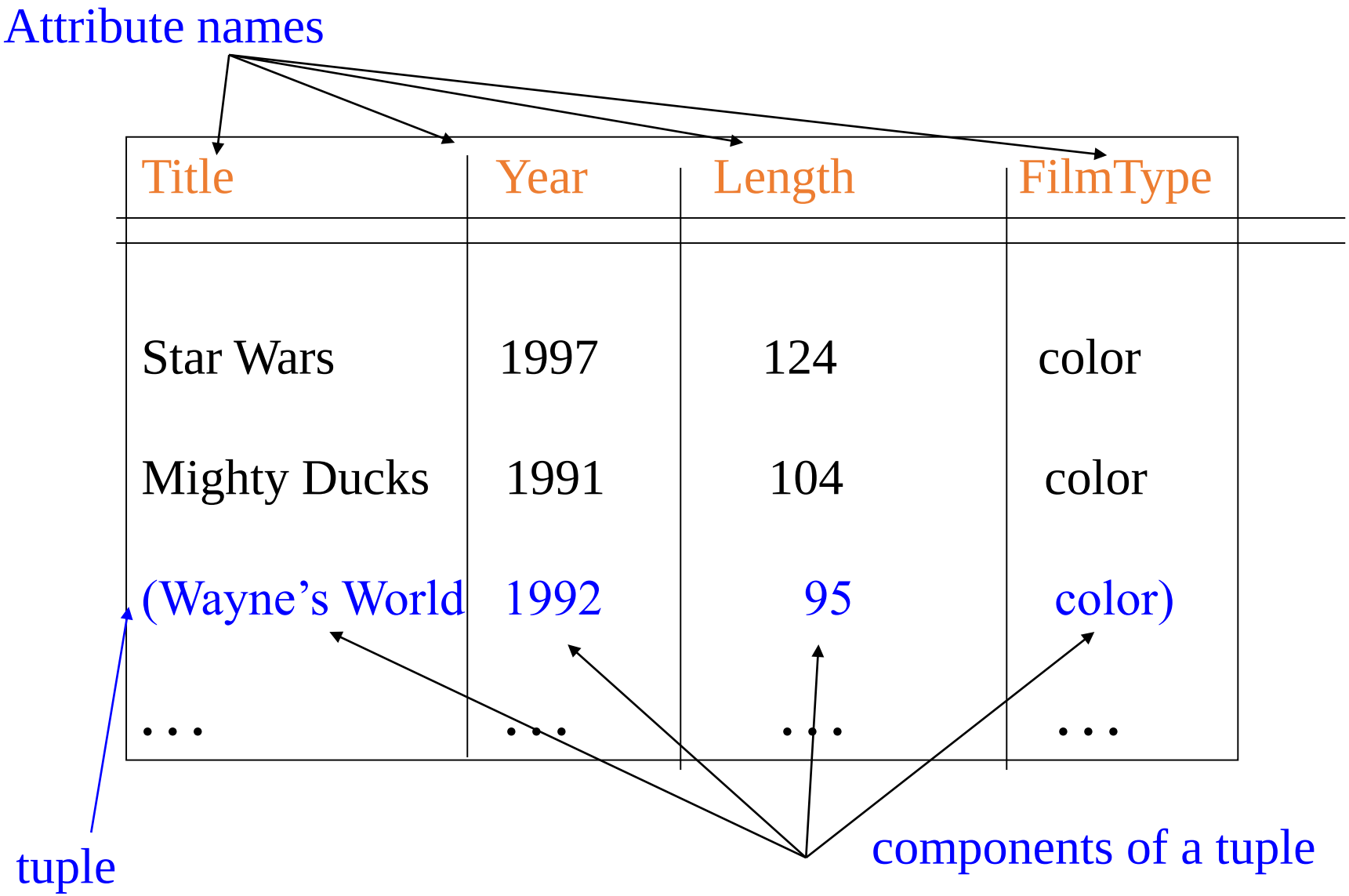
- A *data model* is a collection of concepts for describing data.
- A *schema* is a description of a particular collection of data, using a given data model.
- A *database instance* is a collection of data compliant with the schema

Process of database design

- Notation for expressing designs: Entity-Relationship (E/R) model



Relations Terminology



More definitions

- Every attribute has an *atomic* type.
- Relation Schema: relation name + attribute names + attribute types
- Relation instance: a set of tuples

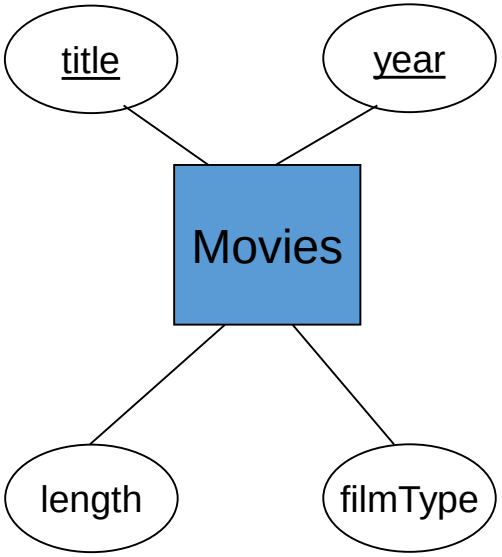
- Database Schema: a set of relation schemas
- Database instance: a relation instance for every relation in the schema

From E/R diagrams to relations

- **Entity sets** become relations with the same set of attributes.

- **Relationships** become relations whose attributes are only:
  - The keys of the connected entity sets.
  - Attributes of the relationship itself.

Example: Entity Set to Relation

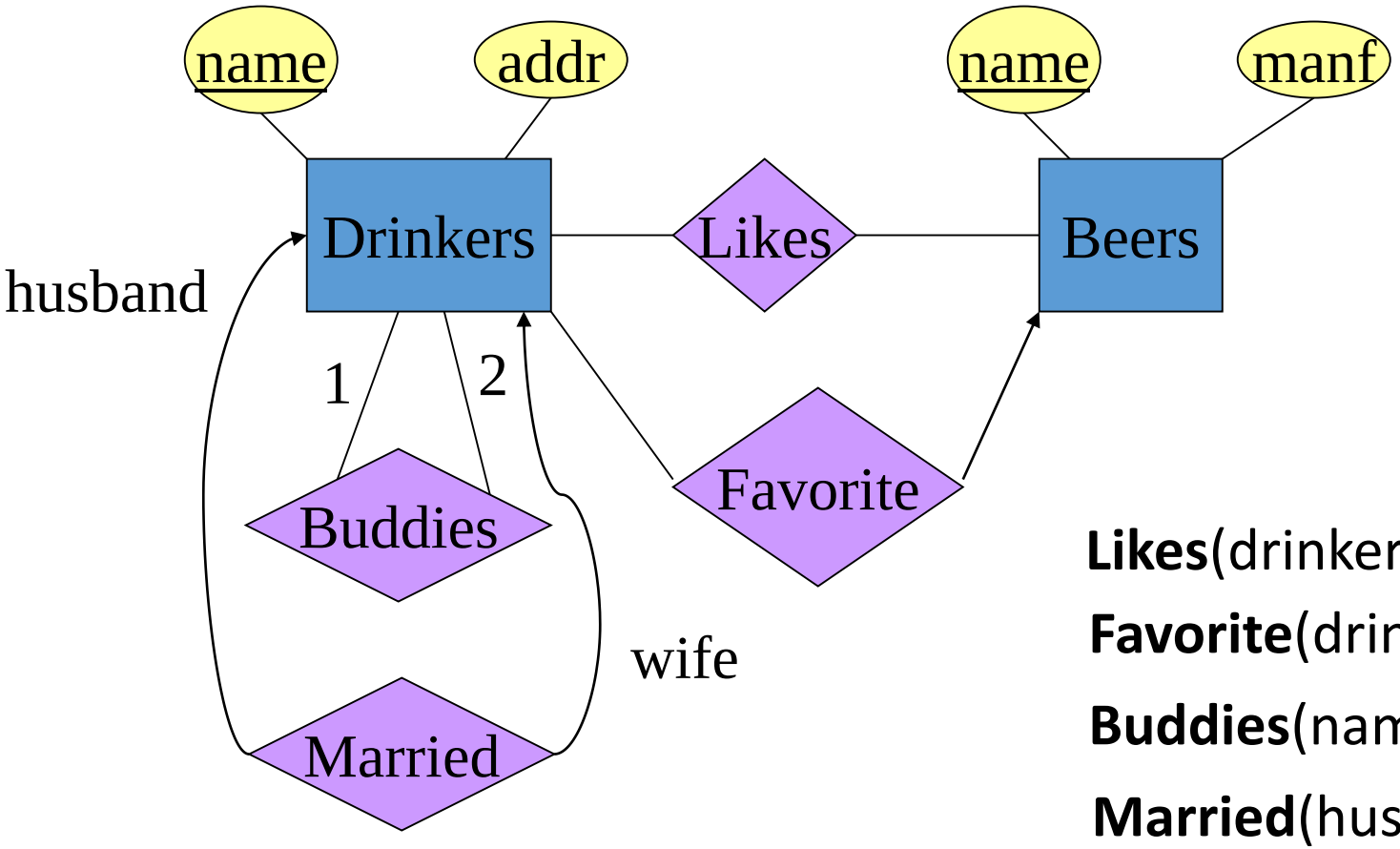


Relation schema:
Movies (title, year, length, filmtype)

Relation instance:

title	**year**	**length**	**filmtype**
Star Wars	1977	124	Color
Mighty Ducks	1991	104	Color
Wayne's World	1992	95	Color

Example 1 (with Renaming)



Likes(drinker, beer)

Favorite(drinker, beer)

Buddies(name1, name2)

Married(husband, wife)

Example 2 (with Renaming)

Relationship **Stars-In** between entity sets **Movies** and **Stars** is represented by a relation with schema:

Stars-In(title, year, starName)

A sample instance is:

We rename here just for clarity.

title	*year*	***starName***
Star Wars	1977	Carrie Fisher
Star Wars	1977	Mark Hamill
Star Wars	1977	Harrison Ford
Mighty Ducks	1991	Emilio Estevez
Wayne's World	1992	Dana Carvey
Wayne's World	1992	Mike Meyers

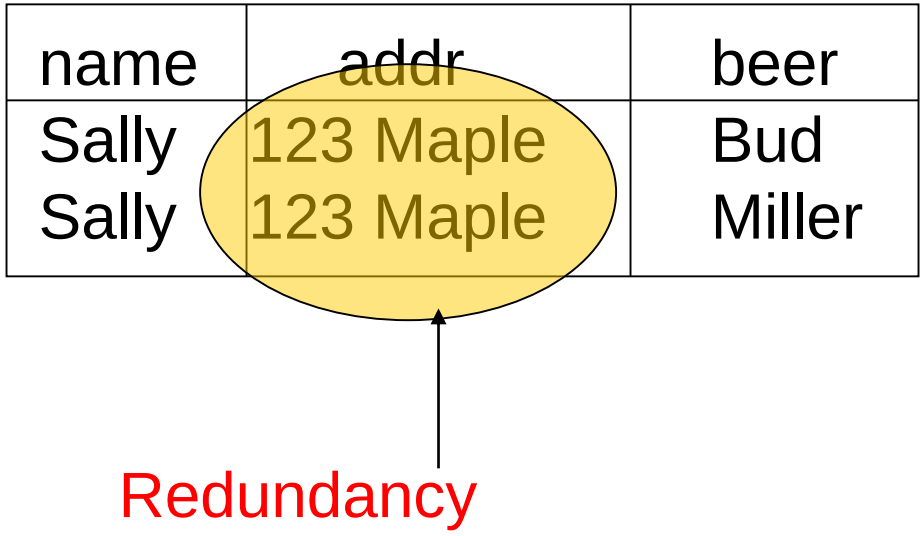
Many-One Relationships

- We not always have a separate relation for them.
  E.g.
  Instead of having
  Drinkers(name, addr) and
  Favorite(drinker, beer)

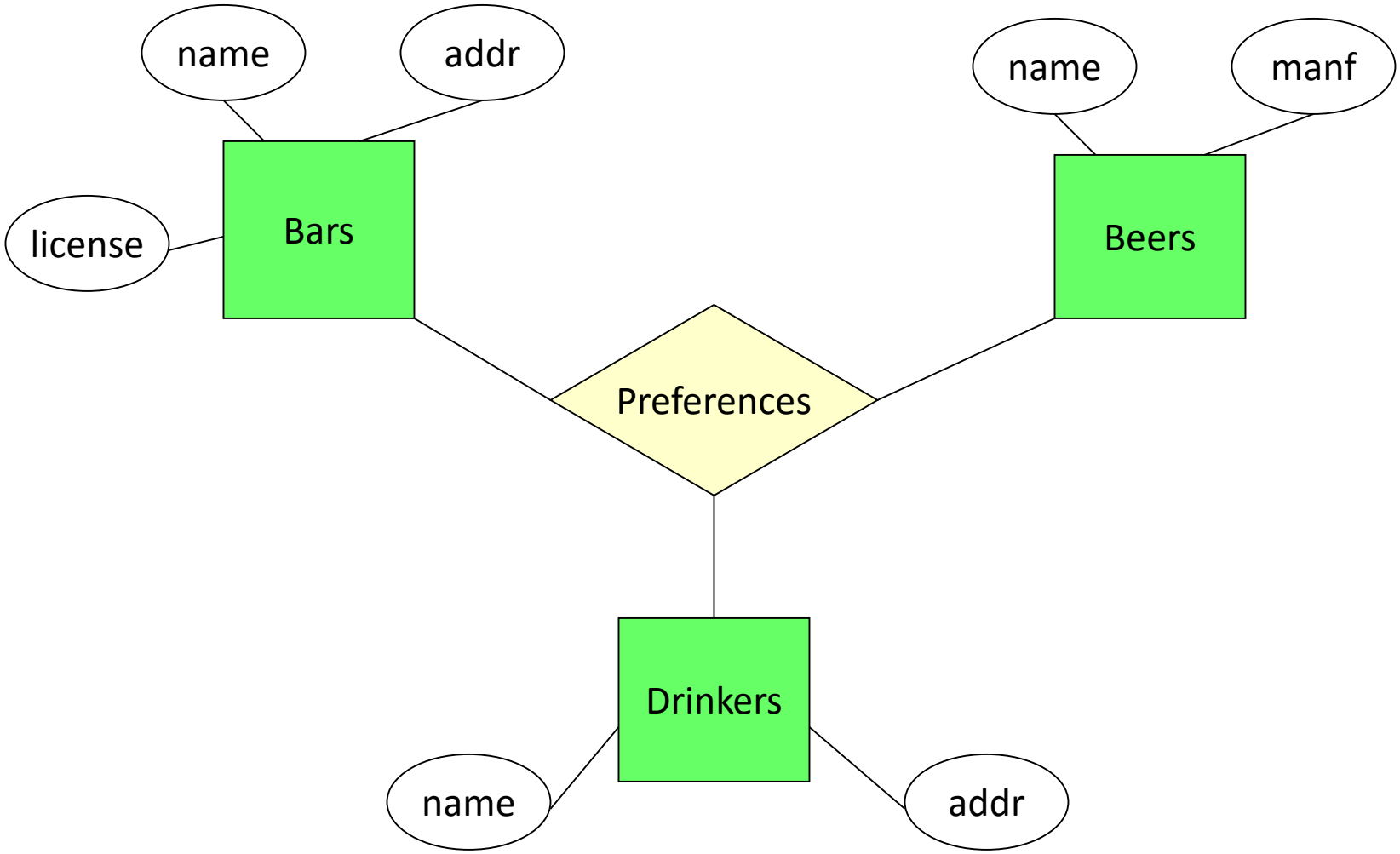
  have
  Drinkers(name, addr, favBeer)

Risk with Many-Many Relationships

- Combining **Drinkers** with **Likes** would be a mistake. Why?
- It leads to harmful redundancy, as:



Many-to-Many Ternary Relationships



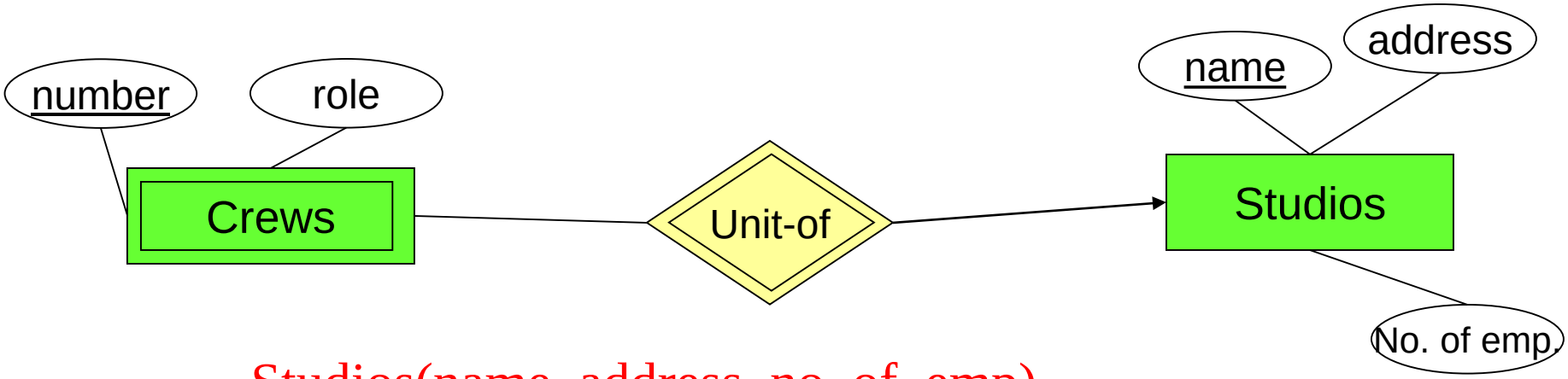
Preferences(drinker_name, beer_name, bar_name)

Handling weak entity sets

- Relation for a weak entity set must include attributes for its complete key (including those belonging to other entity sets), as well as its own, nonkey attributes.

- A supporting (double-diamond) relationship is redundant and yields no relation.

Example: weak entity sets



Studios(name, address, no_of_emp)

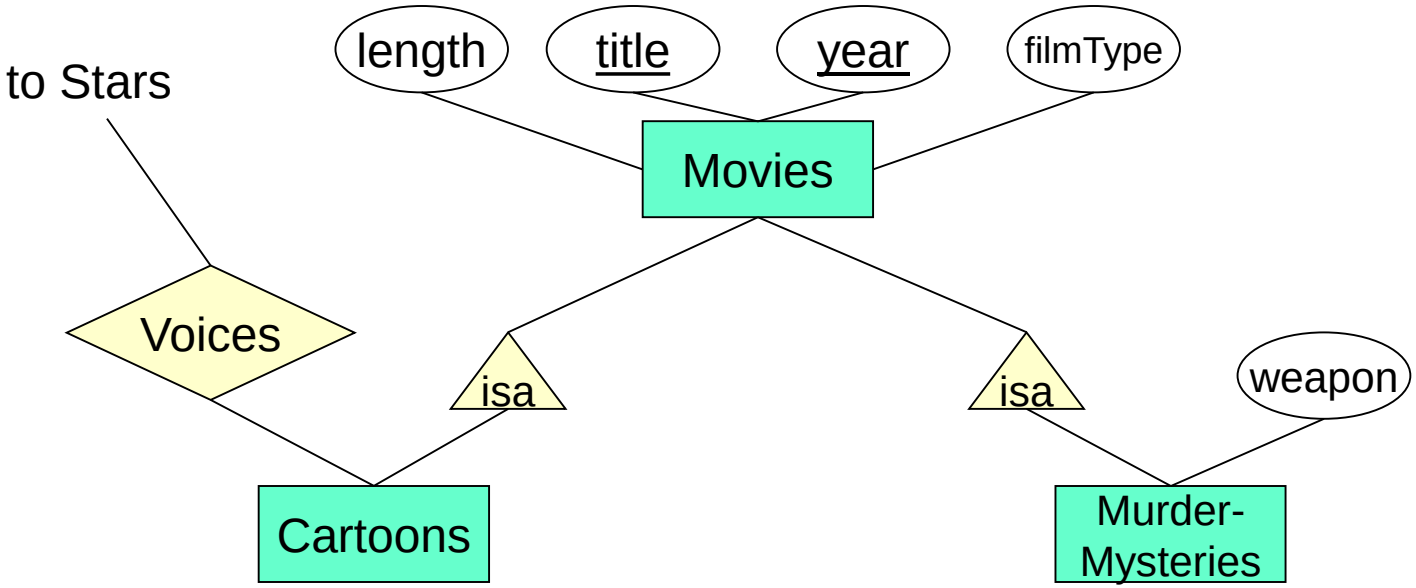
Crews(number, studioName, role)

~~Unit-of(number, studioName, studioName2)~~

Must be the same

Unit-of becomes part of Crews

Our Movie Example (with ISA)



How to convert to relations?

Subclass Structures to Relations

Two different approaches

- **OO Approach**
  - An object belongs to **exactly one** class.
    - An object inherits properties from all its super-classes but it is **not** a member of them.

- **E/R Approach**
  - An "object" can be represented by entities belonging to several entity sets that are related by **isa** relationships.
    - The linked entities together represent the object and give that object its properties (attributes and relationships).

OO approach: example

- Every subclass has its own relation.
  - All the properties of that subclass, including all its inherited properties, are represented in this relation.

- **Example:** For our example the relational database schema would be:

  Movies (*title, year, length, filmType* **)**

  Cartoons (*title, year, length, filmType* **)**

  MurderMysteries (*title, year, length, filmType, weapon***)**

  Cartoon-MurderMysteries (*title, year, length, filmType, weapon***)**

- Can we merge **Cartoons** with **Movies**?
  - If we do, we lose information about which movies are cartoons.

- For the relationship **Voices**, we create:
  - **Voices(title, year, starName)**

- Is it necessary to create two relations one connecting **cartoons** with **stars**, and one connecting **cartoon-murder-mysteries** with **stars**?
  - Not, really. We can use the same relation (table).

E/R Approach: example

- We will have the following relations:

  Movies (*title, year, length, filmType***).**

  MurderMystery (*title, year, weapon***).**

  Cartoons (*title, year***).**

  Voices (*title, year, name***).**

- **Remark:**
  - There is no relation for class **Cartoon-MurderMystery**.
  - For a movie that is both, we obtain:
    - its voices from the **Voices** relation,
    - its weapon from the **MurderMystery** relation,
    - and all other information from the **Movies** relation.

- Relation **Cartoons** has a schema that is a **subset** of the schema for the relation **Voices**. Should we eliminate the relation **Cartoons**?
- However there may be **silent** cartoons in our database. Those cartoons would have no voices and we would lose them.

Comparison of Approaches

OO translation **advantage**:

- The **OO** translation keeps **all** properties of an object together in **one** relation.

OO translation **drawback**:

- Too many tables!
  - If we have a root and ***n*** children we need $\mathbf{2^n}$ different tables!!!

Comparison of Approaches

E/R translation **advantage**:

- The **E/R** translation allows us to find in one relation tuples from all classes in the hierarchy.

E/R translation **drawback**:

- We may have to look in several relations to gather information about a single object.

Examples

- What movies of 2009 were longer than 150 minutes?
  - Can be answered directly in the E/R approach.
  - In the OO approach we have to examine all the relations.

- What weapons were used in cartoons of over 150 minutes in length?
  - More difficult in the E/R approach.
    - We should access **Movies** to find those of over 150 mins.
    - Then, we have to access **Cartoons** to see if they are cartoons.
    - Then we should access **MurderMysteries** to find the weapon.
  - In OO approach we need only access the **Cartoon-MurderMysteries** table.

Null Values to Combine Relations

- If we are **allowed** to use **NULL** in tuples, we can handle a hierarchy of classes with a single relation.

- For the *Movie* hierarchy, we would create a single relation:
  - **Movie (**title, year, length, filmType, studioName, starName, voice, weapon**)**
  - "*Who Framed Roger Rabbit?*", being both a cartoon and a murder-mystery, is represented by a tuple that had no NULL's.
  - *The "Little Mermaid,"* being a cartoon but not a murder-mystery, has NULL in the *weapon* component.

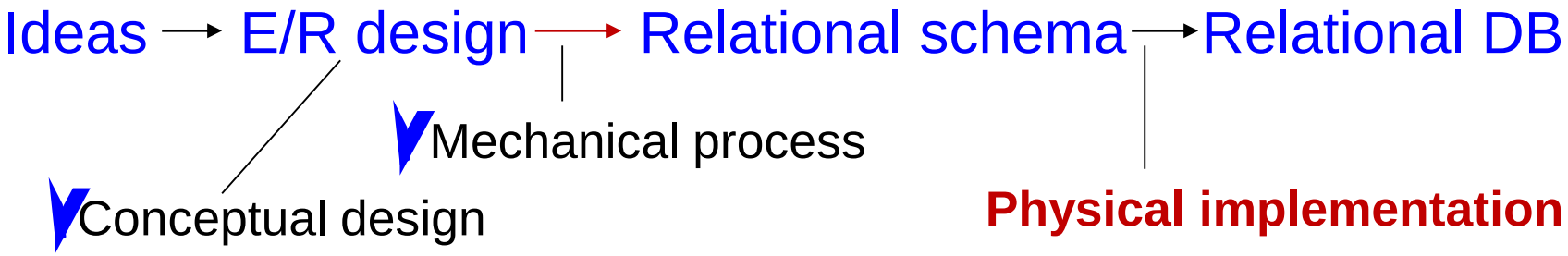
- This approach allows us to find **all** the information about an object in one relation. Drawback?

Will the schema be “good”?

- If we use this translation process, will the schema we get be a good one?
- The process should ensure that there is no redundancy.
- But only with respect to what the E/R diagram represents.

- Crucial thing we are missing: functional dependencies (We only have keys, not other FDs.)
- So we still need to learn the design theory.

Converting logical schema into physical tables



Using DBMS: PostgreSQL

- Powerful object-relational database management system (ORDBMS)
- Open source, originally developed at the University of California at Berkeley CS Department.
- Pioneered many concepts that only became available in some commercial database systems much later.
- Because of the liberal license, PostgreSQL can be used, modified, and distributed by anyone free of charge for any purpose, be it private, commercial, or academic.

2-tier client-server architecture

The DBMS software is running on Database server.

Your interaction with database consists of 2 processes:

- A server process: manages the database files, maintains connection pool, performs database actions on behalf of clients.
- The client (frontend) application: a text-oriented tool, a graphical application, a web server that accesses the database to display web pages, or a specialized database maintenance tool.

Note: The client and the server can be on different hosts. They communicate over a TCP/IP network connection. The files that can be accessed on a client machine might not be accessible on the database server machine.

Connecting to DB server

- Login to CDF
- ssh into dbsrv1.cdf.toronto.edu
- Now you are connected to the DB server as u_name.
- The databases are already created for each student, and they have name: csc343h-u_name.
- Each student has its own single database

Interactive shell client

- Connect to your specific database:

psql csc343h-u_name

- You see the following prompt:

csc343h-u_name=>

- You are now connected and you can enter sql commands

Schema in PostgreSQL

- A database contains one or more named schemas, which in turn contain tables.
- To create or access objects in a schema, write a *qualified name* consisting of the schema name and table name separated by a dot:

schema.table

- There is a default schema called *public*, for which you don't need to specify the qualified name, only the name of the table

Documentation: http://www.postgresql.org/docs/9.1/static/ddl-schemas.html

PostgreSQL – SQL standards

- PostgreSQL supports most of the major features of SQL:2003.
- Out of 164 mandatory features required for full Core conformance, PostgreSQL conforms to at least 150.
- In addition, there is a long list of supported optional features. (No current version of any database management system claims full conformance to Core SQL:2003).

SQL syntax is very similar to MySQL and Oracle

SQL tutorials: http://www.postgresql.org/docs/9.6/static/tutorial-sql.html

Data Definition Language (DDL): converitng Schema into physical tables

CREATE TABLE **table_name**
(
    column_name1 *data_type*,
    column_name2 *data_type*,
    column_name3 *data_type*,
    ....
)

Create Table

```
CREATE TABLE Movies (
   title VARCHAR(50),
   year INT,
   length INT,
   rating CHAR(2),
   studioname VARCHAR(20)
);

CREATE TABLE Studios(
   name VARCHAR(20),
   website VARCHAR(255)
);
CREATE TABLE Stars (
   name VARCHAR(20),
   gender CHAR(1),
   birthyear INT,
   birthplace VARCHAR(40)
);
```

Data types:

- **NUMERIC (precision, scale) :**
  - scale - count of decimal digits in the fractional part, to the right of the decimal point.
  - precision - the total count of significant digits in the whole number
- **CHAR(n)** allocates a fixed space, and if the string that we store is shorter than **n**, then it is padded with blanks.
- Differently, **VARCHAR(n)** denotes a string of up to n characters.
- CHAR has better performance. Use CHAR(n) for frequently used fields, and use VARCHAR(n) otherwise.
- Default date format: '1994-11-28'

Declaring primary keys

```
DROP TABLE IF EXISTS Movies;
DROP TABLE IF EXISTS Studios;

CREATE TABLE Studios (
   name VARCHAR(20) PRIMARY KEY,
   address VARCHAR(255)
);

CREATE TABLE Movies (
   title VARCHAR(20),
   year INT,
   length INT,
   rating CHAR(2),
   studioname VARCHAR(20),
   PRIMARY KEY (title, year)
);
```

Insert

```
INSERT INTO Movies
VALUES('Walk the Line', 2005, 136, 'PG', 'Fox');

INSERT INTO Movies
VALUES('Pretty Woman', 1990, 119, 'R', 'Disney');

INSERT INTO Movies
VALUES('Wayne''s World', 1991, 104, 'PG', 'Paramount');

INSERT INTO Movies
VALUES('Unfaithful', 2002, 124, 'R', 'Fox');

INSERT INTO Movies
VALUES('Runaway Bride', 1999, 116, 'PG', 'Paramount');

INSERT INTO Movies
VALUES('The Princess and the Frog', 2009, 97, 'G', 'Disney');
```

Altering, Dropping

```
ALTER TABLE Stars ADD [COLUMN] phone
   CHAR(16);


ALTER TABLE Stars ALTER COLUMN phone TYPE
   CHAR(26);


ALTER TABLE Stars DROP COLUMN phone;


DROP TABLE Stars;

DROP TABLE Movies;

DROP TABLE Studios;
```

Getting information about tables

- Describe all tables:

```
\dt
```

```
  List of relations
 Schema |    Name    | Type  |  Owner
--------+------------+-------+----------
 public | movie      | table | mgbarsky
 public | movie_exec | table | mgbarsky
 public | movie_star | table | mgbarsky
 public | starsin    | table | mgbarsky
 public | studio     | table | mgbarsky
(5 rows)
```

Describe columns of table movie

\d+ movie;

```
 Table "public.movie"
  Column     |         Type          | Modifiers | Storage
-------------+-----------------------+-----------+----------+-------------
 title       | character varying(30) | not null  | extended |
 year        | integer               | not null  | plain    |
 length      | integer               |           | plain    |
 incolor     | integer               |           | plain    |
 studioname  | character varying(20) |           | extended |
 producerc   | character varying(3)  |           | extended |
Indexes:
    "movie_pkey" PRIMARY KEY, btree (title, year)
```

Explain to each other the following terms:

- data model
- relational data model
- tuple
- component in a tuple
- data type of a component
- attribute
- relation
- schema
- relation instance

Identify any unclarities about the terms and discuss.