University of Glasgow

# Data Storage and Retrieval
# Lecture 4
# From ER Models to Tables

Dr. Graham McDonald

Graham.McDonald@glasgow.ac.uk

University of Glasgow

# Database design lifecycle

- Requirements analysis
  - User needs; what must database do?
- Conceptual design
  - High-level description; often using E/R model
- Logical design
  - Translate E/R model into (typically) relational schema

Today

- Schema refinement
  - Check schema for redundancies and anomalies
- Physical design/tuning
  - Consider typical workloads, and further optimise

University of Glasgow

# Overview

- Understanding Entities & Relationships as ‘Tables’ in a database
- Converting your diagram into tables

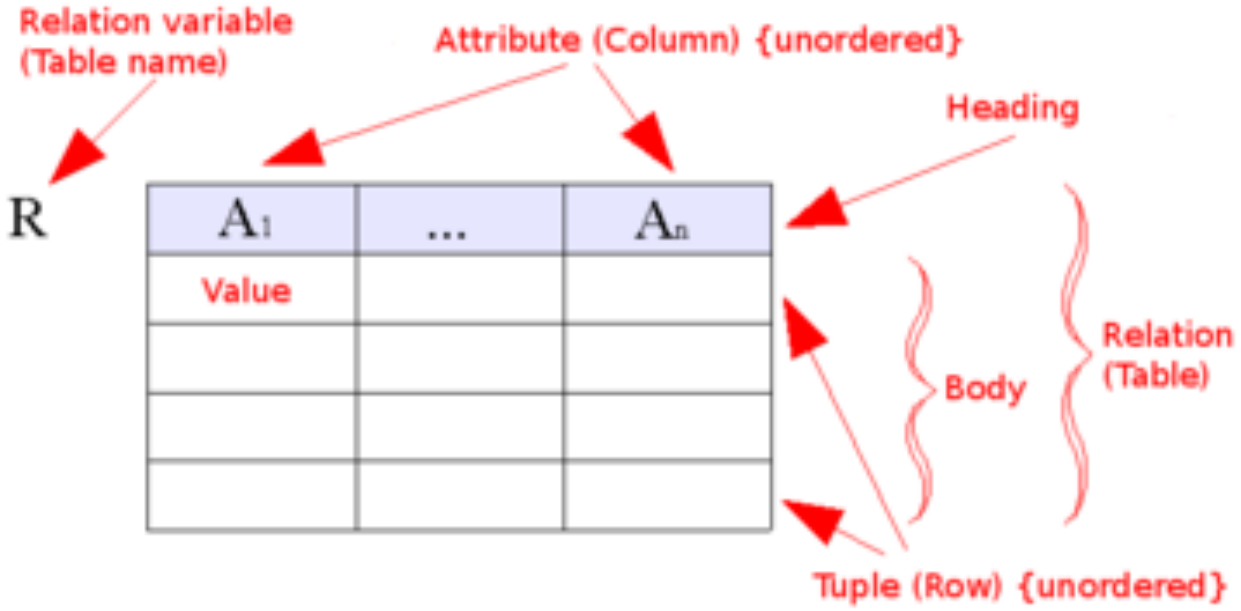
# Reminder - Data Modelling

- ER Model allowed us to establish the relationships and dependencies amongst the information
- We now need to arrange the data into a **logical structure** of **relations**
- The logical structure can then be mapped into the storage objects supported by the database – i.e. **tables**

# Entities → Tables

Roughly, a table (relation) is constructed for each item of interest in a DB

A relation equates (approximately) to an entity type (or some form of relationship) in the ER diagram.



University of Glasgow

# The Heading

- All relations must have a heading
  - Name of relation
    - Student
  - Names of columns of relation (the attributes)
    - Name, student ID, exam1, exam2

STUDENT (Name, Student ID, exam1, exam2)

The number of attributes determines the DEGREE of the relation

University of Glasgow

# Tuples

- The rows of a relation comprise its body
  - These are referred to as TUPLES
- A **tuple** (record) is a row of a relation, i.e. a set of values which are instances of the attributes
  - e.g. < ‘Fraser’, 880123, 66, 90 >
  - The meaning of each value is determined by its position in the tuple

University of Glasgow

# Relations → Schema

- A **relation schema** is a set of attributes
  - written R ($A_1$, $A_{2,}$ . . . $A_n$) e.g.
  - Student (name: Text, matric: Number, ex1: Number, ex2: Number)
- Each attribute in a relation schema has a **domain**
- A **relational database schema** is a set of these relation schemas

University of Glasgow

# Domains

- Domains are a lot like Data Types in programming
  - Defines the set of values that can be assigned to an attribute
  - Determines the range of allowable operations on each value
    - Add, subtract, concatenate…….

University of Glasgow

# Domains

- A **domain** is a set of atomic values that can be assigned to an attribute

- A domain has two aspects:
  - its meaning - e.g. the set of matriculation numbers
  - its format - e.g. a integer in range 0…9999999

- Different DBMS offer different sets of domains:
  - ***MS Access*** offers: Text, Number, Memo, Date/Time, Currency, AutoNumber, Yes/No, etc. - `NOT SQL STANDARD`
  - ***MySQL*** offers **`standard`** SQL types: CHAR (fixed length strings), VARCHAR (variable length strings), INT, FLOAT, DATE...

University of Glasgow

# Quick Reference: (My)SQL Data Types

| Data Type | Description | Examples |
|---|---|---|
| INT | Integer number | 1, 5, -100 |
| FLOAT, DOUBLE | Floating point number | -1.1, 5, 6e10 |
| BOOLEAN | Boolean | 1, 0 |
| *(MySQL doesn't have BOOLEAN) TINYINT(1)* | *Integer with only 1 bit* | *1, 0* |
| CHAR(x) | Fixed length string of length x | 'A ' |
| VARCHAR(x) | Variable length string up to length x | 'A' |
| DATE, TIME, TIMESTAMP | Various date/time data types | '2016-01-01 00:00:00.000000' |

See also:
https://dev.mysql.com/doc/refman/8.0/en/data-types.html

# Converting Your ER Diagram to Tables

University of Glasgow

# Translating E-R to Relational Schema

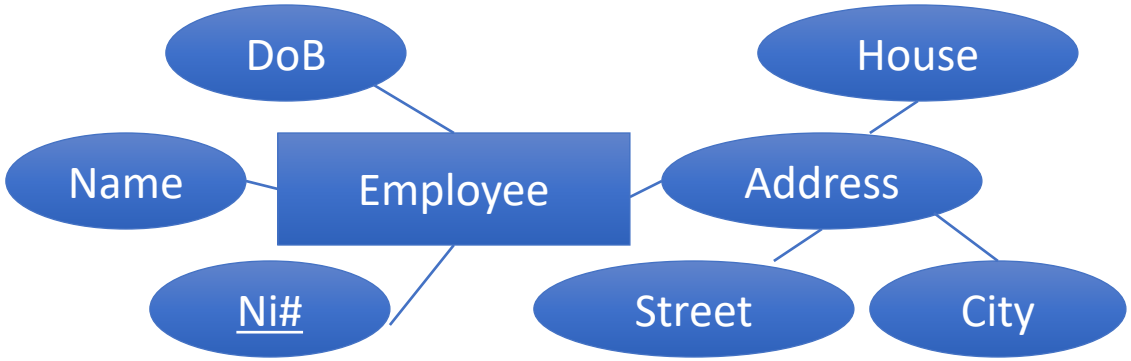
1. **Entities and their simple attributes**
2. Weak entities and their simple attributes
3. 1-M relationships (and their attribute)
4. 1-1 relationships (and their attributes)
5. M-N relationships (and their attributes)
6. Composite attributes
7. Multivalued attributes

University of Glasgow

# ER to Schemas: 1 Strong Entity Types

Step 1

- For each *strong* entity type, create a relation (table) with:
  - A column for each simple attribute
    - composite attributes - broken into several columns
  - A primary key – one or more of the candidate key attributes




**EMPLOYEE**

| name | NINumber | DoB | House | Street | City |
|---|---|---|---|---|---|

***address***

***Good Practice: Use the name of the composite attribute as a prefix for each of the component names***

University of Glasgow

# Attributes -> Columns

Step 1

- A column of a relation is an **attribute** having:
  - a **name** (indicates the role the column has in this relation)
  - a domain (indicates the set of values it may take)

**Student_ID_Number: INT**, **address**: VARCHAR(100),
**dateOfBirth**: DATE

# Primary Keys

Another Example:

Employee (name: VARCHAR(20) ,**<u>NI_no</u>**: INT)

Project (p_name: VARCHAR(20), **<u>P_ID</u>**: INT)

- a particular staff record can be identified as: *the record in the Employee table where NI_No= 9912345*
- a particular project record can be identified as: *the record in the Project table where P_ID= 125*
- ***all of the record's other data will be accessed via these 'keys'***
  - i.e. given a key value, we can determine everything else about the corresponding record

University of Glasgow

# Examples of Primary Keys?

Step 1

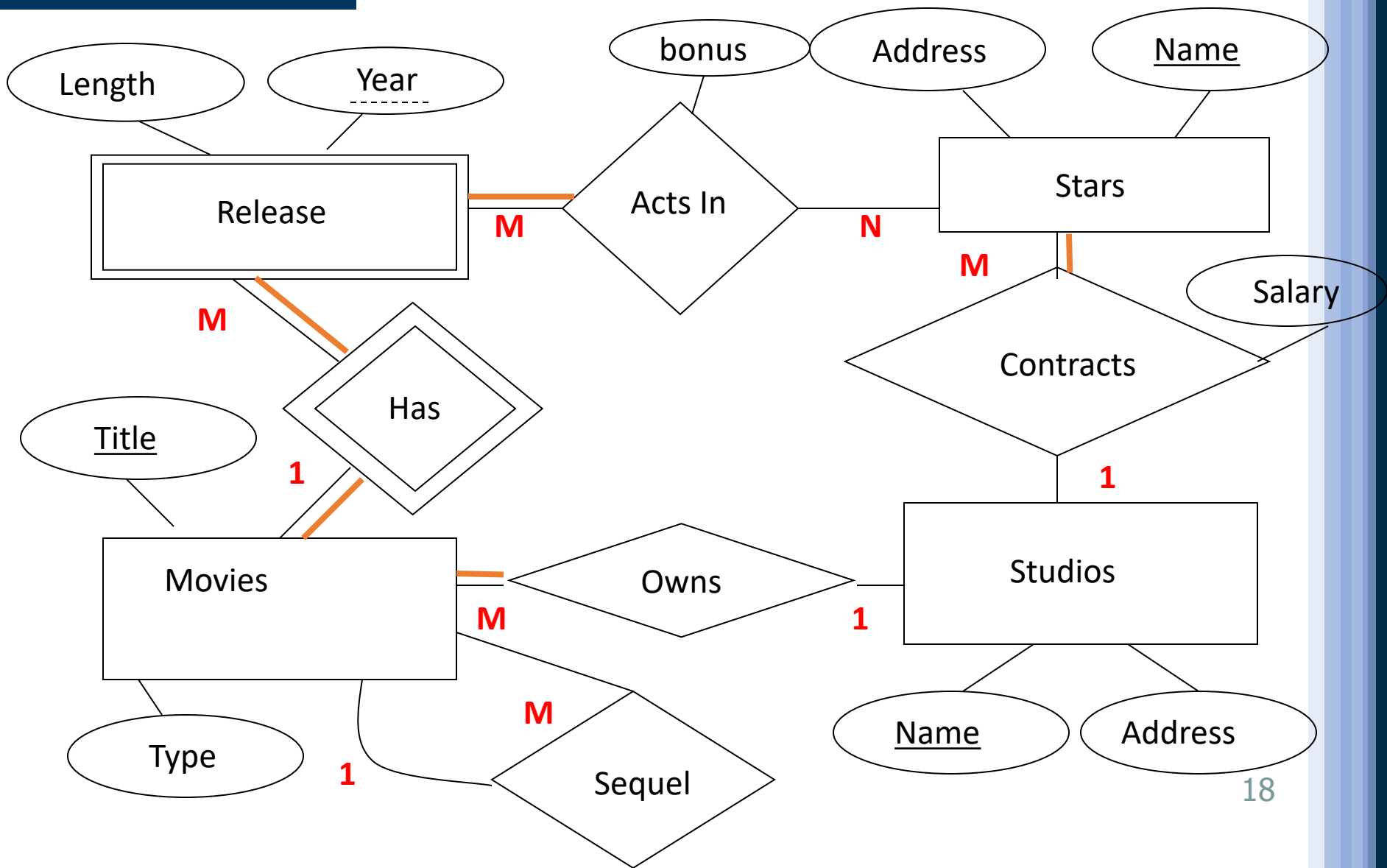
- Student Id
- Staff number
- National Insurance Number/Social Security Number
- Course Id

- First Name and Last Name?
- FlightNumber and FlightDate
- Bank Sort Code and Account Number

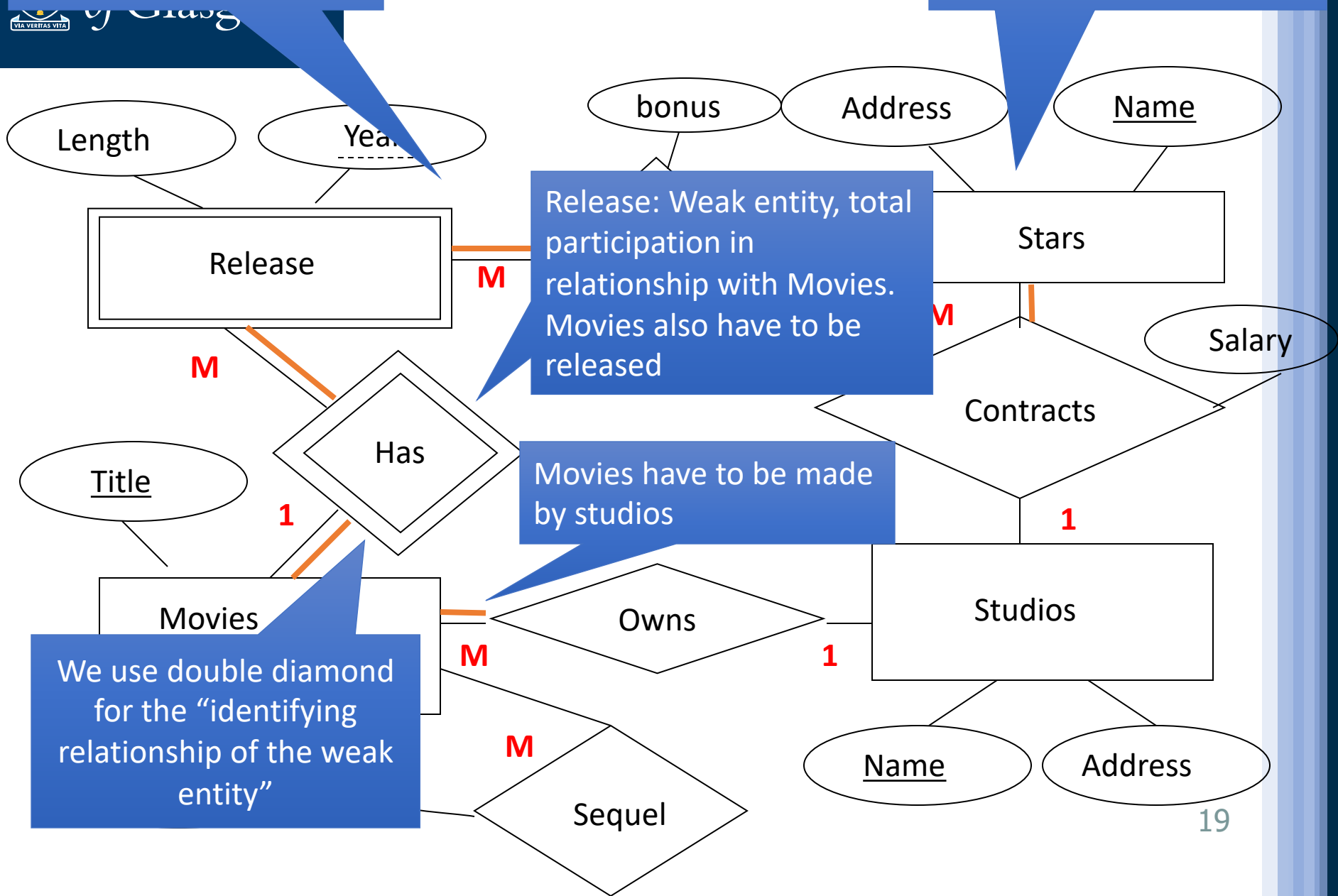
- Primary keys can consist of a single attribute or **multiple attributes** in combination
  - Called a **composite key**

University of Glasgow

# Movies ER Diagram



# Movies ER Diagram



University of Glasgow

Step 2

# Relations - Entities

- Movie(Title, Type)
- Stars(Name, Address)
- Studio(Name, Address)

What about Release?

University of Glasgow

# The Relational Model

Step 2

- A Movie has a Release

- In the relational database
  - How does each member of the Release entity set know which Movie they are related to?

➢This is done via **KEYS**

  ➢**Primary**

  ➢**Foreign**: references to the primary key of another table

University of Glasgow

# Weak Entities Mapping

Step 2

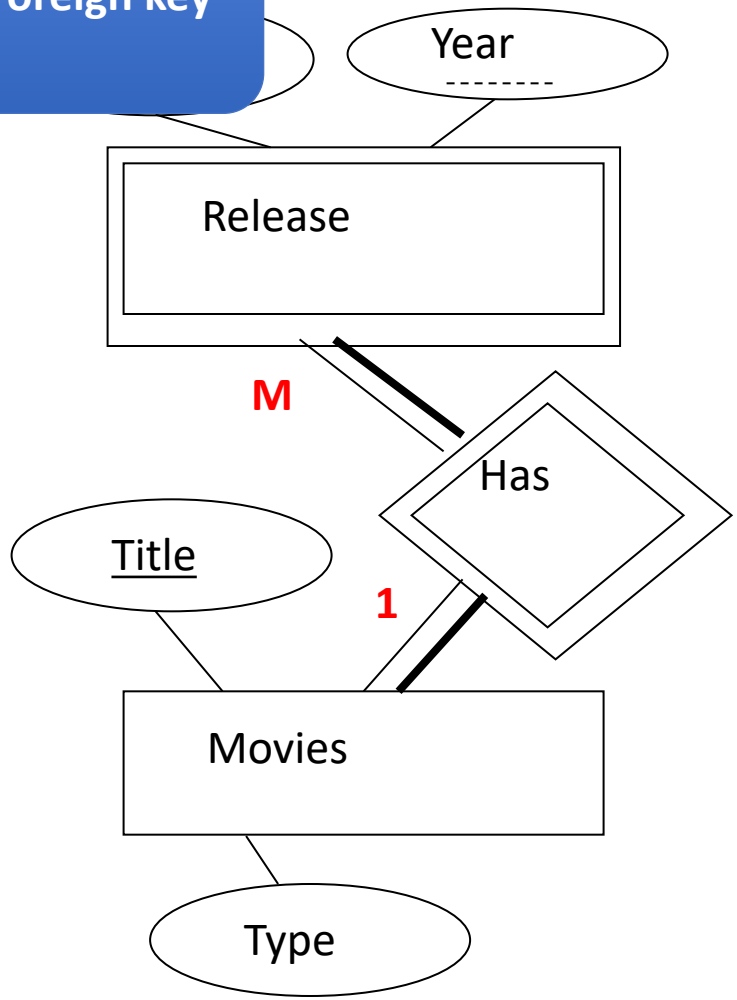
- Create primary key for a weak entity type from
  - primary key attributes of the identifying relationship types
  - partial keys of the weak entity

- ***Rule:*** *For each weak entity*
  - *create a new relation schema*
  - *for each identifying relationship: add the key attributes of the related entity to the new schema as foreign key attributes*
  - *declare the composite primary key of the schema*
  - *add the simple attributes*

# Relations, Relationships

- Movie(Title, Type)
- **Release(Title, Year, Length)**
- Stars(Name, Address)
- Studio(Name, Address)

Title is a **Foreign key**



# Foreign Keys: Definition

Step 2

- The foreign key is used to *cross-reference* between tables
- Definition: A **foreign key** is an attribute (or set of attributes) that *exist in one or more tables* and which is the *primary key* for *one* of those tables.
- The foreign key restricts the domain of the attribute
  - A value in a foreign key MUST exist in the referenced primary key attribute

**Movies**

| Title | Type |
|---|---|
| 42nd Street | Musical |
| Miracle on 34th Street | Chrismas |

**Release**

| Title | Year | Length |
|---|---|---|
| Snake Pit | 1948 | 108 |

Violation of Foreign Key constraint

University of Glasgow

# Using Foreign Keys

Step 2

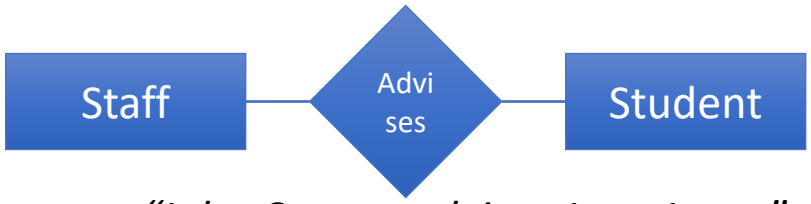
- A foreign key is a *referential constraint* between two tables, i.e. a value in a foreign key MUST exist in the referenced primary key
- A table may have multiple foreign keys and each foreign key can reference a different table
- Foreign keys need *not* have the same attribute name across tables
  - Could have been Release(MovieTitle, Year, Length)
  - Names should differ to help readability!
- The MUST have the same data type

- Foreign keys are important in modelling **weak entities** and **relationships**.

University of Glasgow

# Foreign Keys

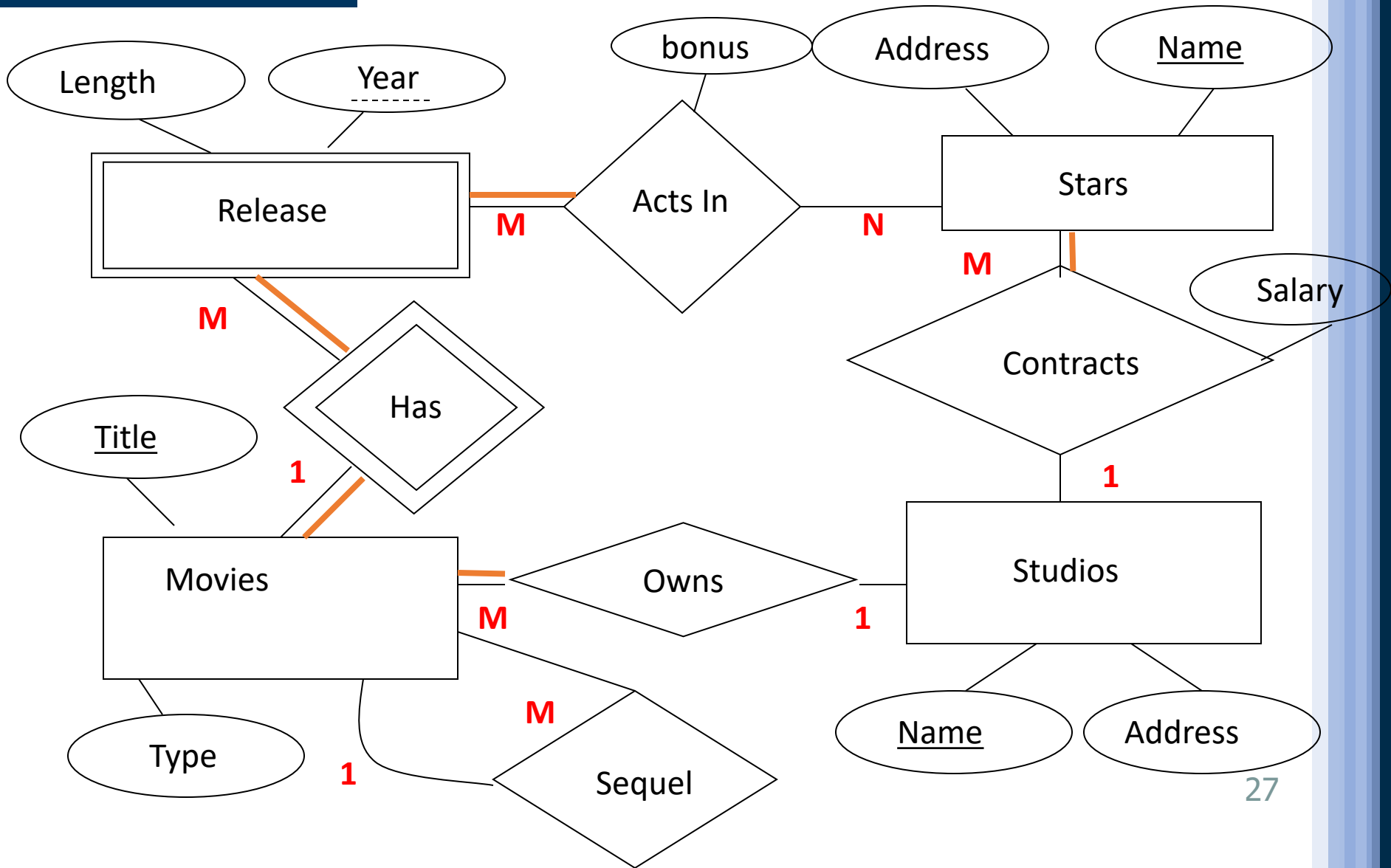
Step 2




*"John Cooper advises Jane Jones"*

- There are only two ways of connecting two related pieces of data in a relational database
  - 1. They are in the same tuple (row) of the same table, i.e. with the same primary key value
    - "Jane" and "Jones" are connected since they are in the same record
  - 2. They are in tuples which are connected by a foreign key or a chain of foreign keys
    - "Jones" and "Cooper" are connected by the foreign key, *adviser*

# Back to the ER Diagram: Relations - Entities



University of Glasgow

# Back to the ER Diagram: Relations - Entities

Step 2

What do we have thus far?

- Movie(Title : VARCHAR(50), Type : VARCHAR(20))
- Release(Title : VARCHAR(50), Year : INT, Length : INT)
- Stars(Name : VARCHAR(50), Address : VARCHAR(50))
- Studio(Name : VARCHAR(50), Address : VARCHAR(50))

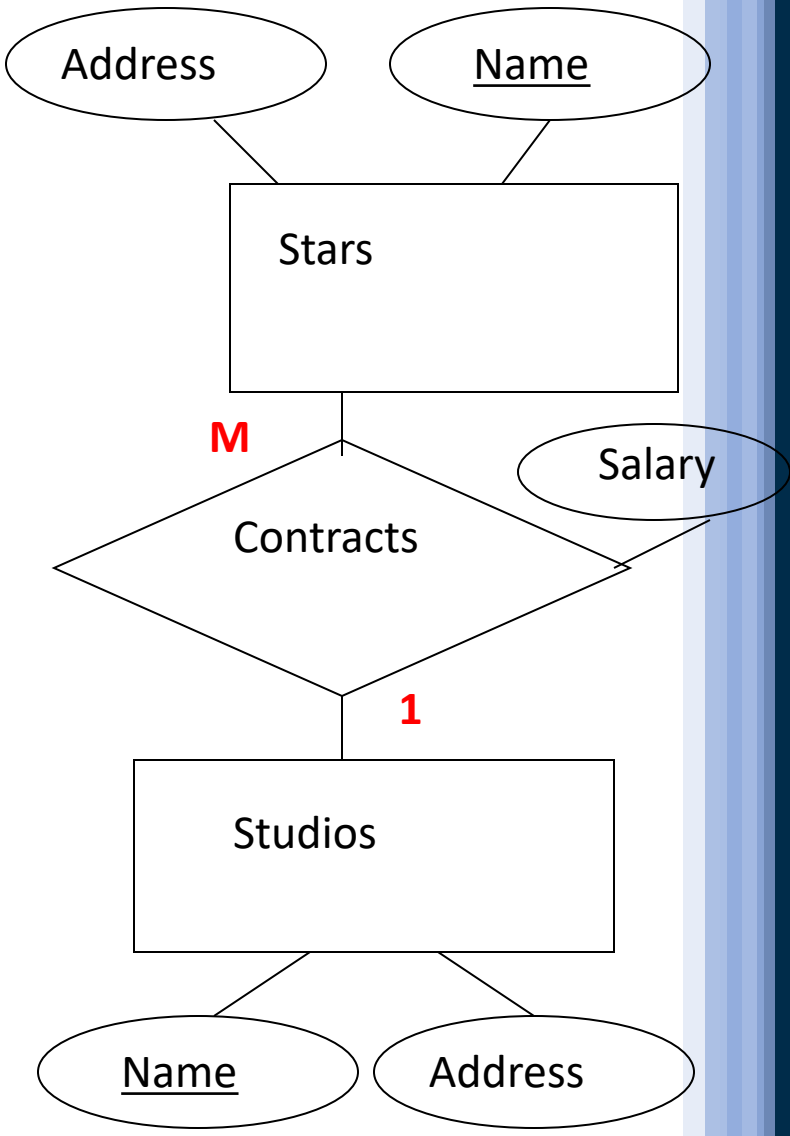
Notes:

- each table has primary key attribute(s)
- Release's Title is a foreign key reference to Movie's Title

# Relationships: 1-to-many

Step 3

- Movie(Title, Type)
- Release(Title, Year, Length)
- Stars(Name, Address)
- Studio(Name, Address)




1-to-many rule:

*the primary key on the 'one side' of the relationship is added to the 'many side' as a foreign key.*

# Relationships: 1-to-many

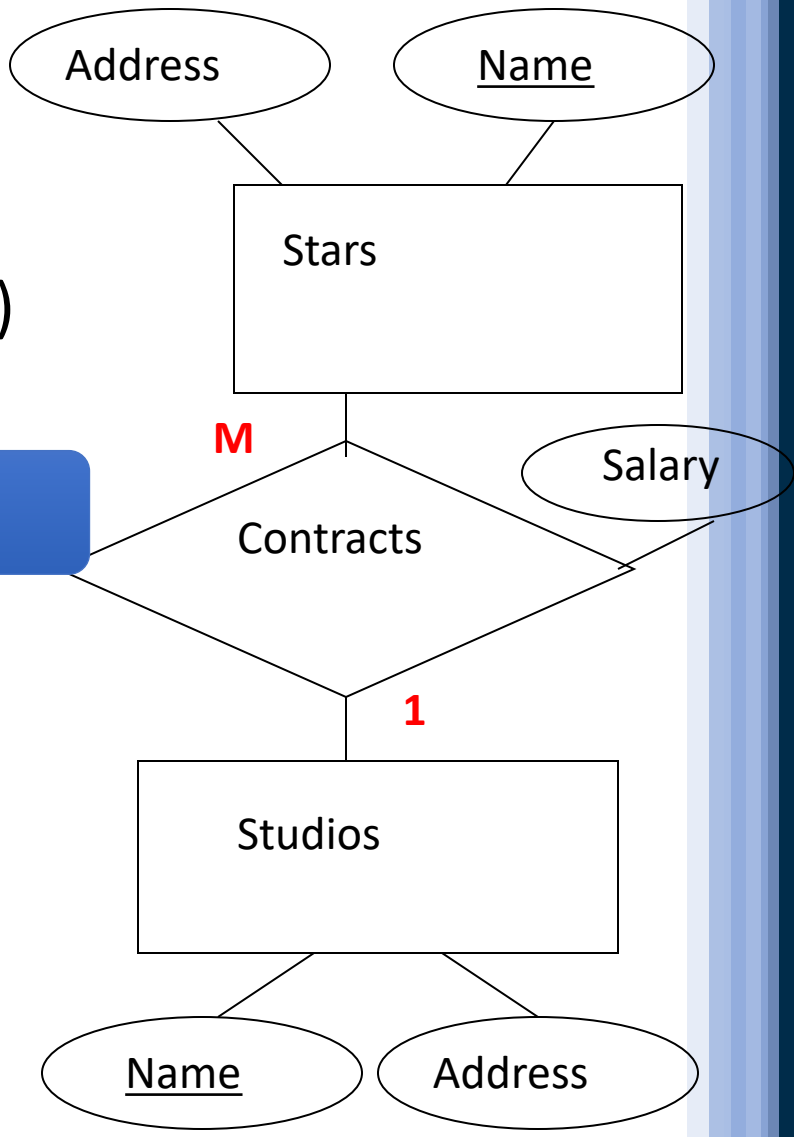
Step 3

- Movie(Title, Type)
- Release(Title, Year, Length)
- Stars(Name, Address, Studio, Salary)
- Studio(Name, Address)

**Foreign key**

1-to-many rule:

*the primary key on the 'one side' of the relationship is added to the 'many side' as a foreign key.*

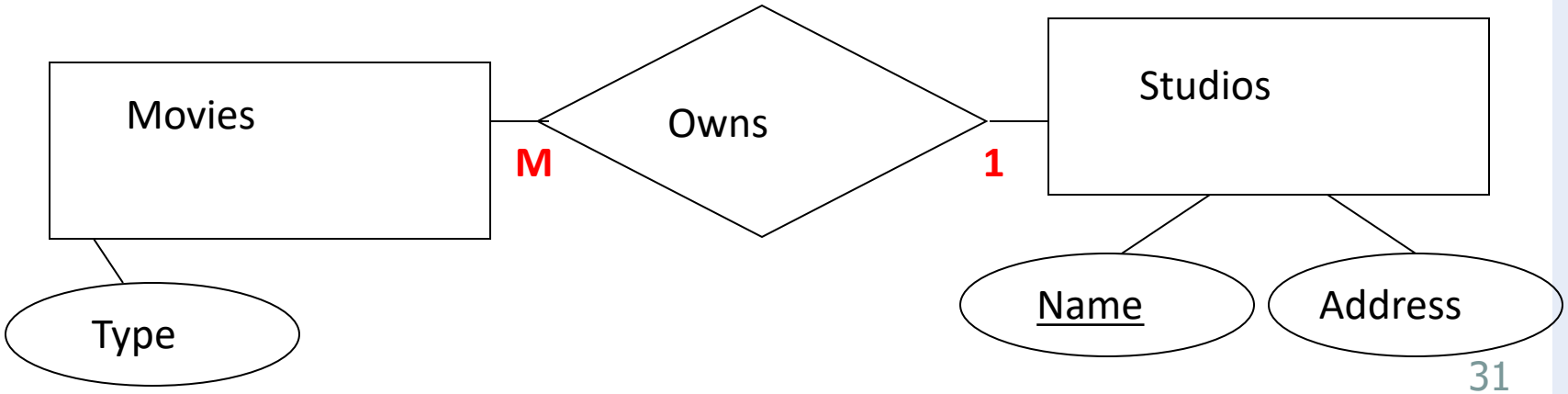


University of Glasgow

# Relationships: 1-to-many

Step 3

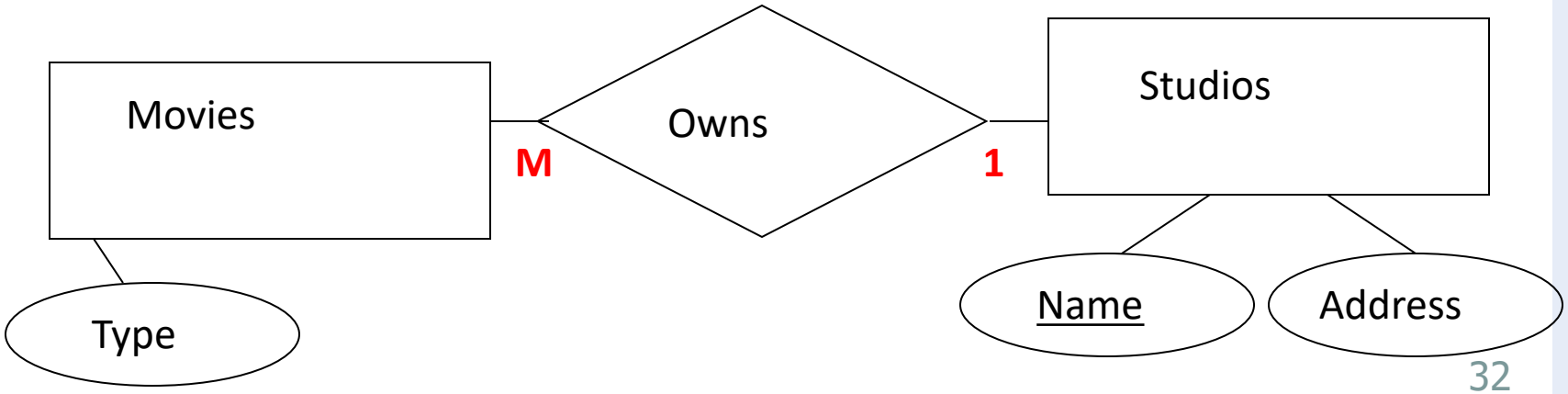
- Movie(Title, Type)
- Release(Title, Year, Length)
- Stars(Name, Address, Studio, Salary)
- Studio(Name, Address)



# Relationships: 1-to-many



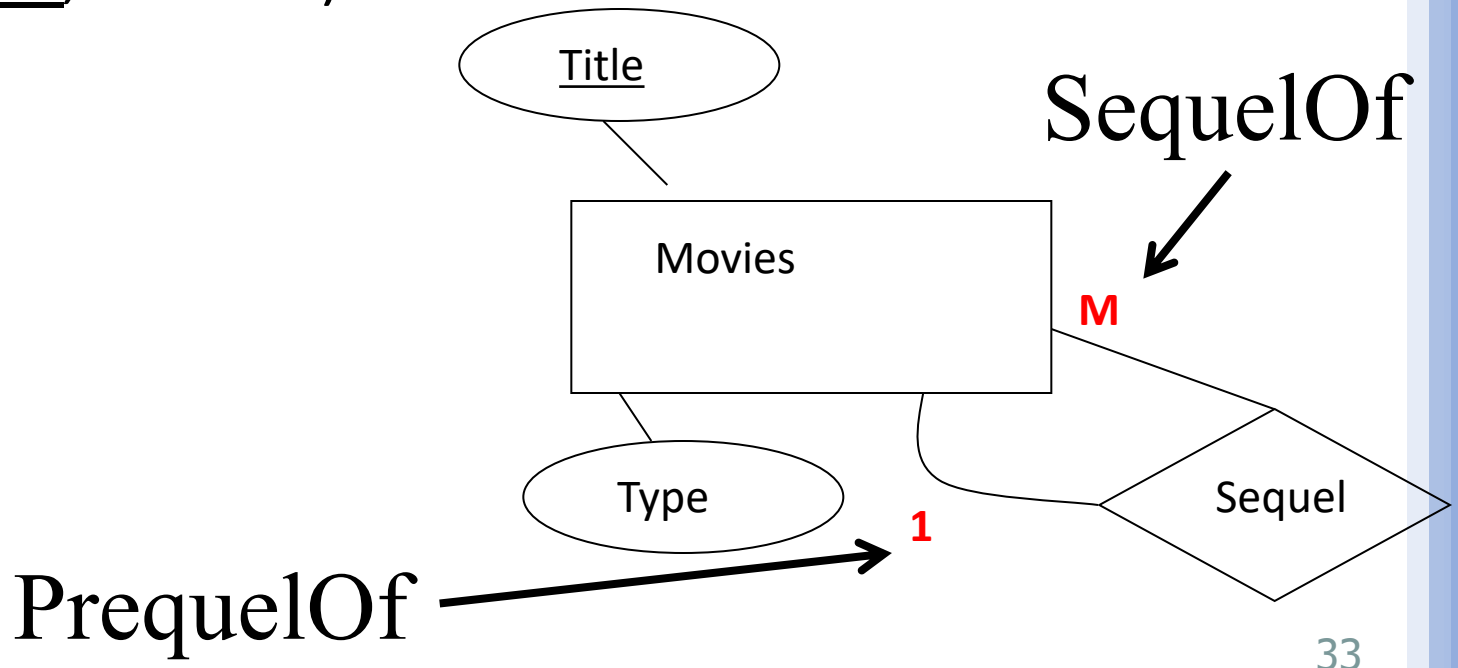
- Movie(Title, Type, Studio)
- Release(Title, Year, Length)
- Stars(Name, Address, Studio, Salary)
- Studio(Name, Address)



# Relationships: 1-to-many



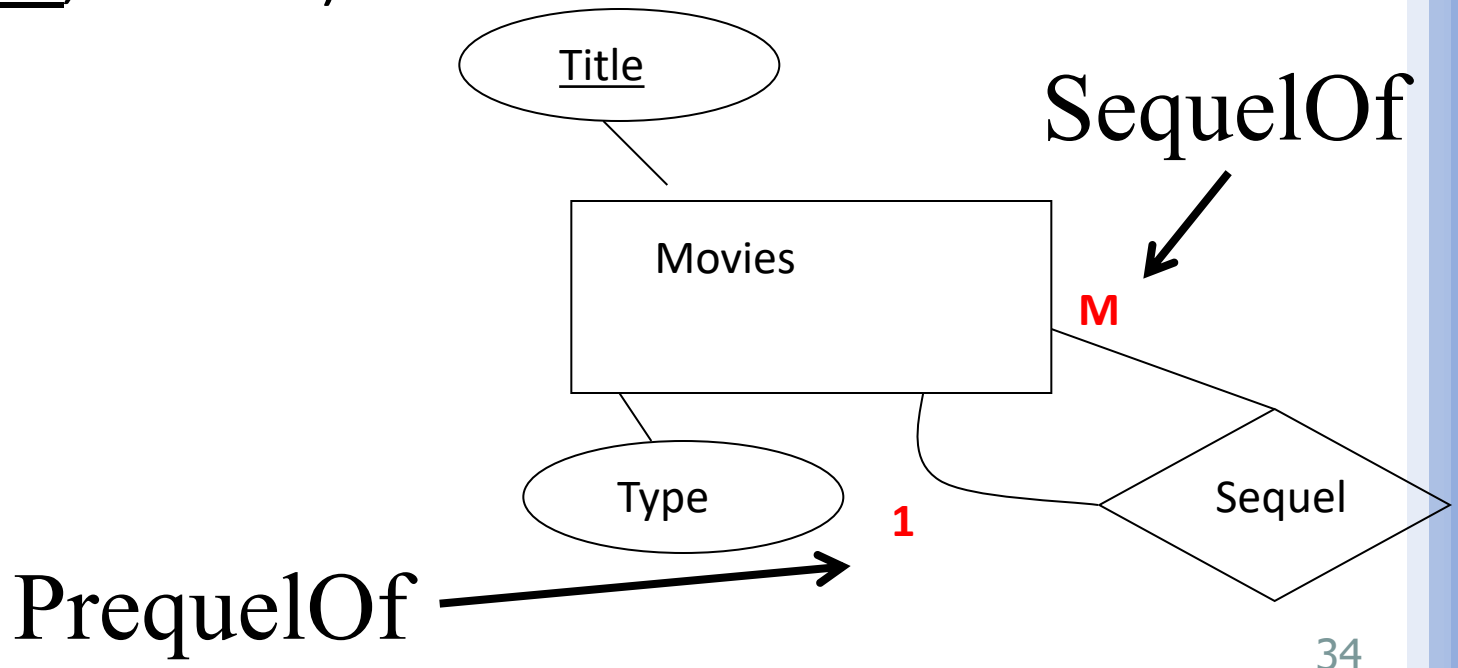
- Movie(Title, Type, Studio)
- Release(Title, Year, Length)
- Stars(Name, Address, Studio, Salary)
- Studio(Name, Address)



# Relationships: 1-to-many

Step 3

- Movie(Title, Type, Studio, SequelOf)
- Release(Title, Year, Length)
- Stars(Name, Address, Studio, Salary)
- Studio(Name, Address)

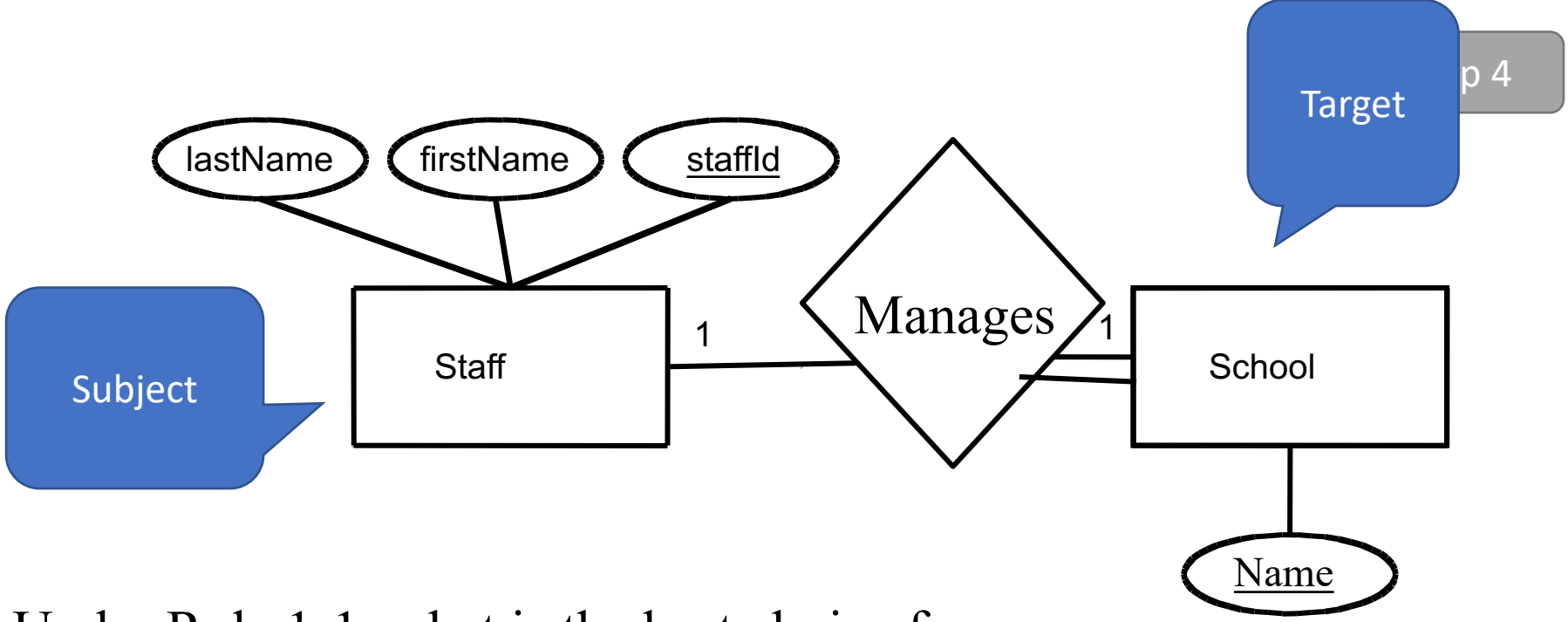


# One-to-one relationships (and their attributes)

Step 4

- The foreign key attributes may be added to either schema

- ***Rule 1-1:*** *For each one-to-one relationship type between two entity types, choose one entity type to be the subject and one to be the target type*
  - *add the key attributes of the subject class to the target schema as foreign key attributes*
  - *add the attributes of the relationship to the target schema*




Under Rule 1-1, what is the best choice for "subject" and "target"?

Set this to be "NOT NULL"

- Staff(staffId, firstName,lastName)
- School(Name, headOfSchoolStaffId)

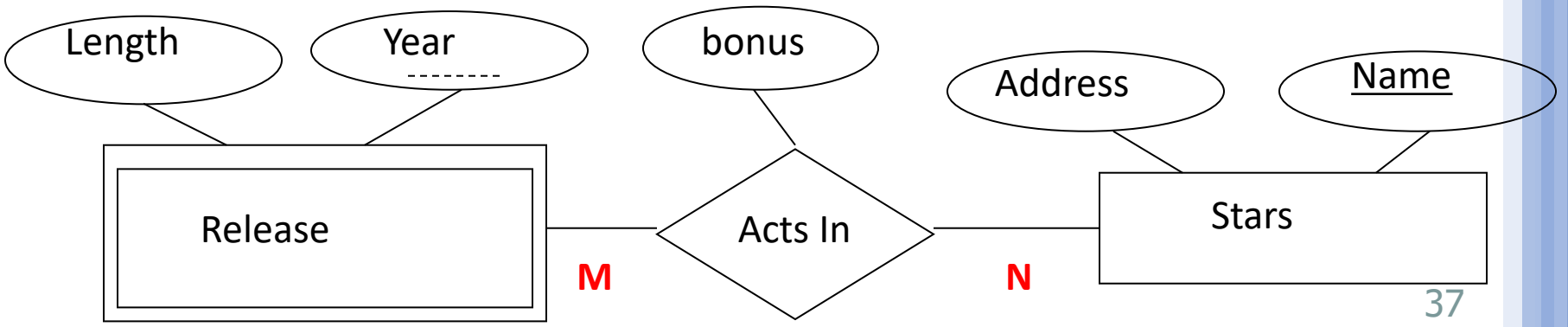
*"If the relationship is mandatory for one entity but not the other, the put foreign key into the table for which participation is mandatory"*

# Relationships: many-to-many

- Movie(Title, Type, Studio, SequelOf)
- Release(Title, Year, Length)
- Stars(Name, Address, Studio , Salary)
- Studio(Name, Address)

Many-to-Many Rule:

- A **new relation** is produced which contains the primary keys from both sides of the relationship *as foreign keys*
- These attributes form **a *composite* primary key** for the relation



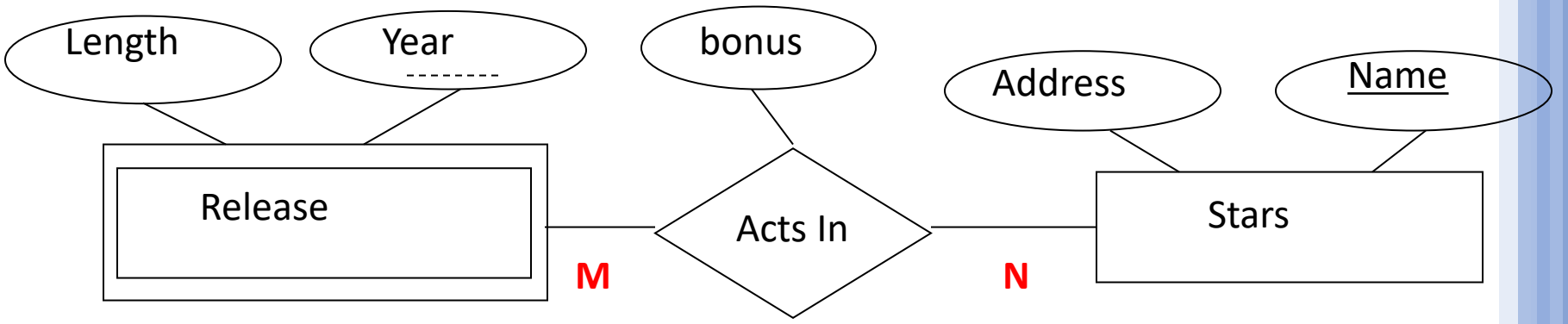
# Relationships: many-to-many

Step 5

- Movie(Title, Type, Studio, SequelOf)
- Release(Title, Year, Length)
- Stars(Name, Address, Studio , Salary)
- Studio(Name, Address)
- ActsIn(Title,Year,StarName, bonus)

Many-to-Many Rule:

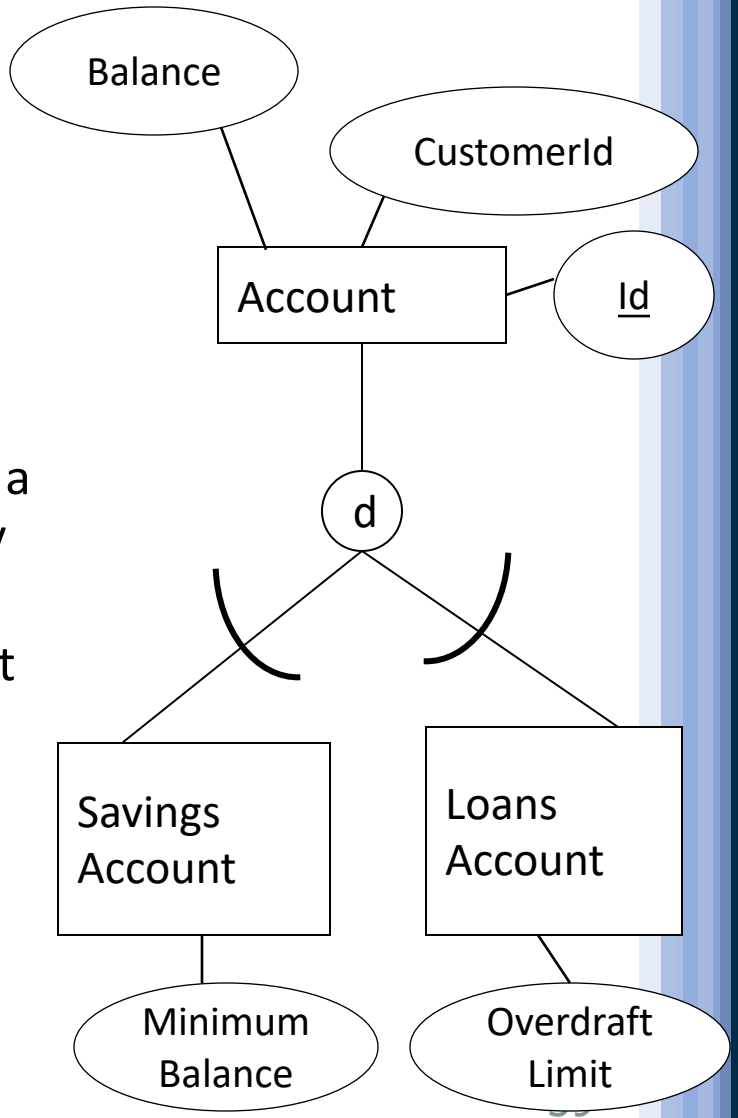
- A **new relation** is produced which contains the primary keys from both sides of the relationship *as foreign keys*
- These attributes form **a composite primary key** for the relation



# Subtypes & Supertype

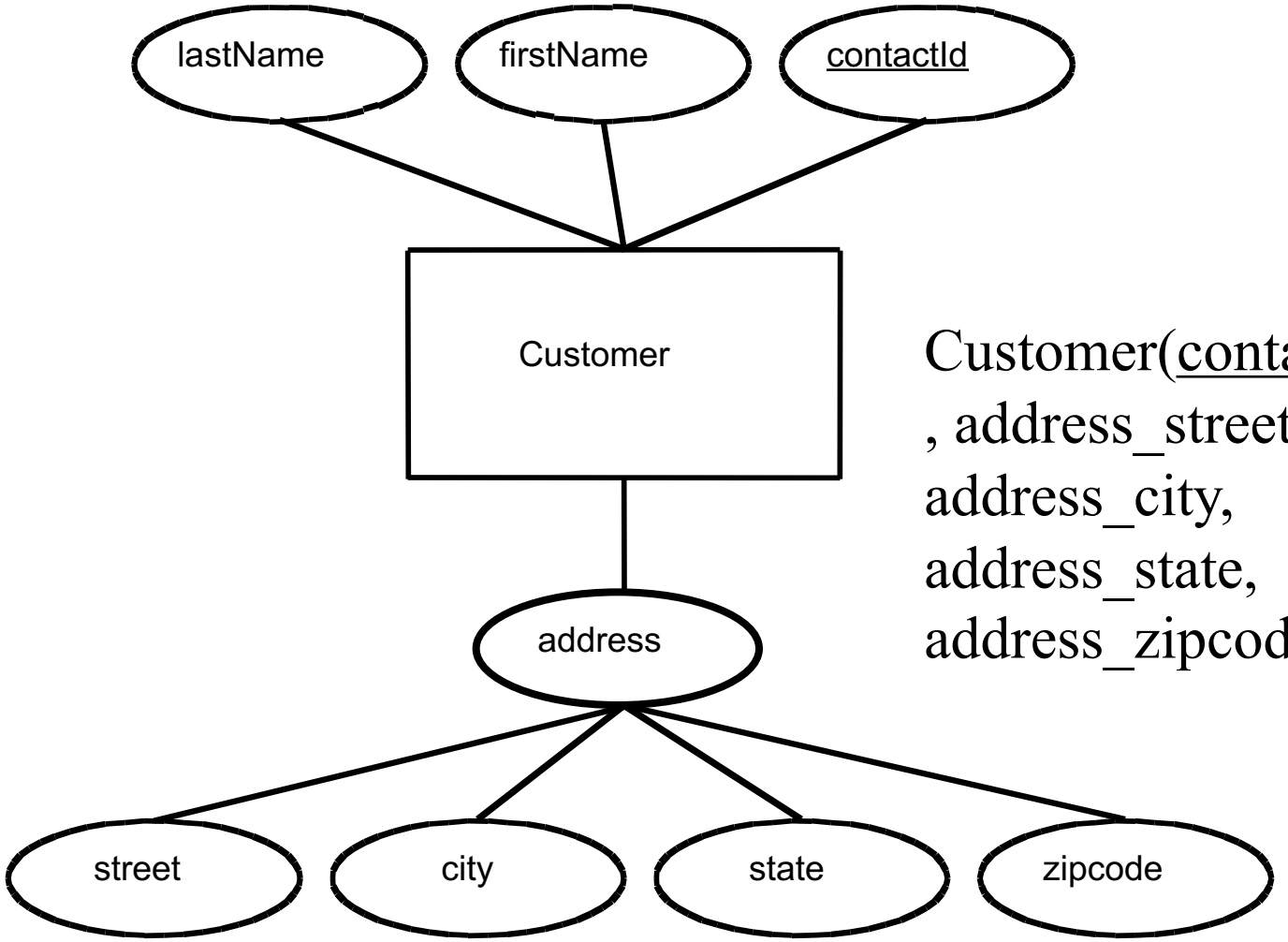
- Not really part of the main relational model, and not directly supported by most standard relational DBMS
- You cannot directly map to a relational schema. Two choices:
  - Model the supertype and all its subtypes as a **single table** (and leave attributes *null* if they don't apply), OR
  - Turn each subtype into its **own table** and set up 1:1 relationships between them and the super entity type.

- Account(<u>ID</u>, Customer, Balance)
- Savings(<u>AccountID,</u> MinimumBalance)
- Loans(<u>AccountID</u>, OverDraftLimit)



# Composite attributes

Step 6

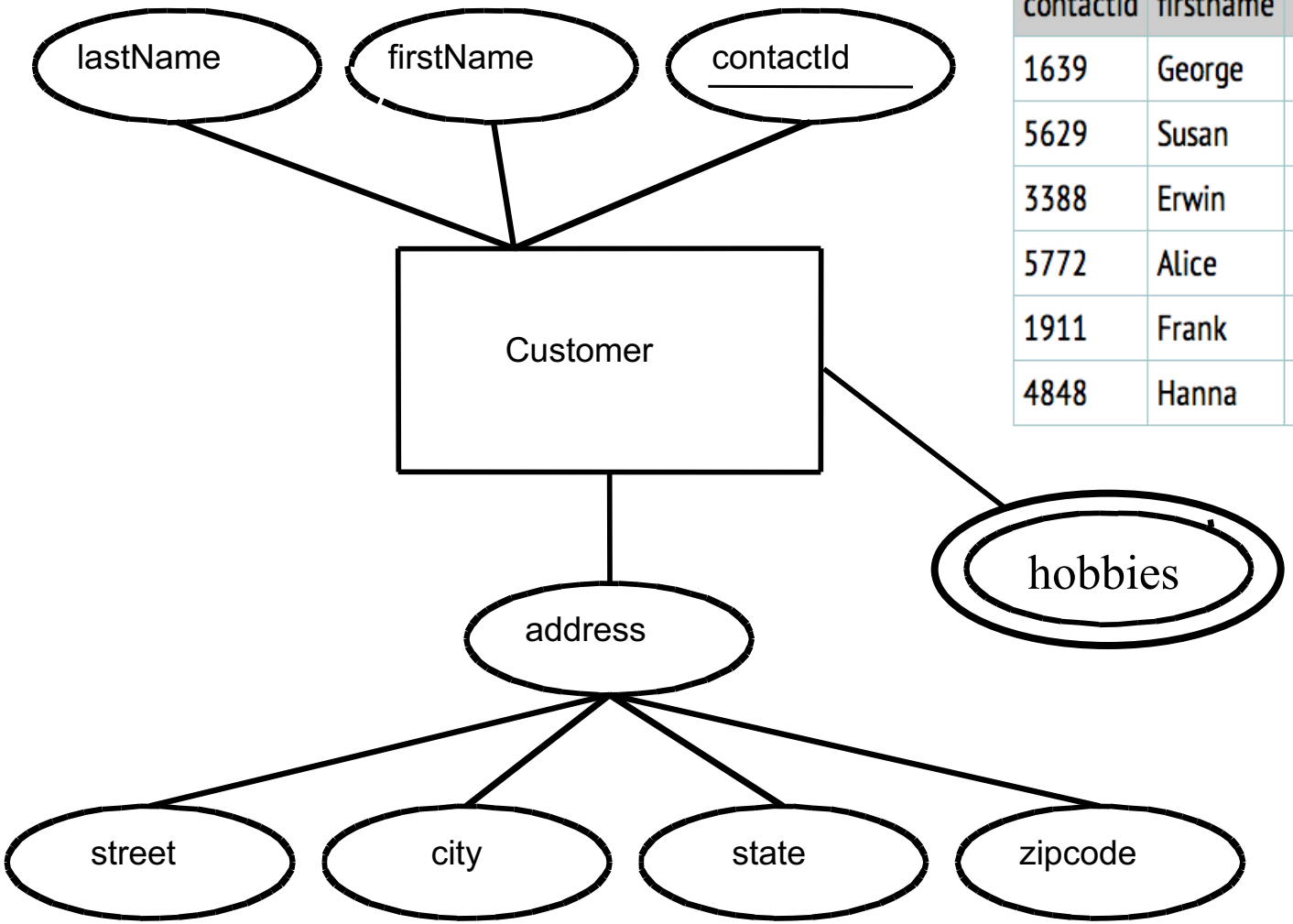



Customer(contactId,… , address_street, address_city, address_state, address_zipcode)

***Create an attribute in the relation schema for each component attribute***
***Use the name of the composite attribute as a prefix for each of the component names***

# Multi-valued attributes

Step 7

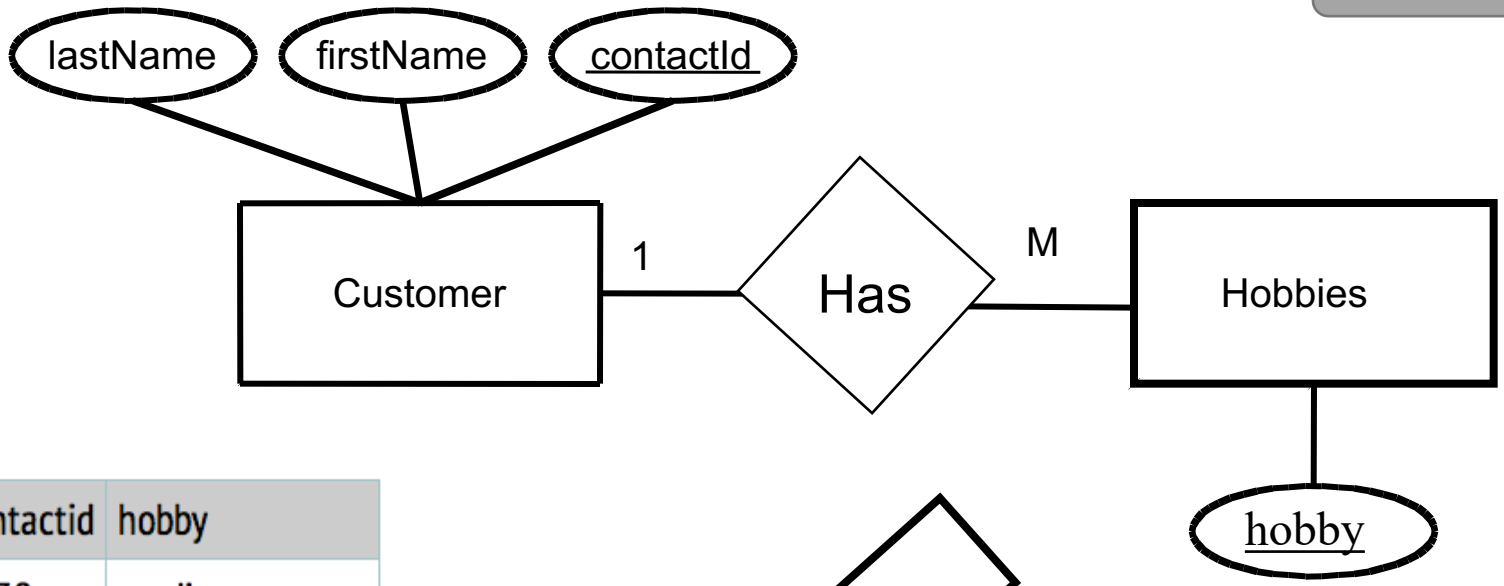



| contactid | firstname | lastname | hobbies |
|---|---|---|---|
| 1639 | George | Barnes | reading |
| 5629 | Susan | Noble | hiking, movies |
| 3388 | Erwin | Star | hockey, skiing |
| 5772 | Alice | Buck | |
| 1911 | Frank | Borders | photography, travel, art |
| 4848 | Hanna | Diedrich | gourmet cooking |

Difficult to search for a particular hobby

**Represent each multi-valued attribute as if it were a weak entity class**

Step 7




| contactid | hobby |
|---|---|
| 1639 | reading |
| 5629 | hiking |
| 5629 | movies |
| 3388 | hockey |
| 3388 | skiing |
| 1911 | photography |
| 1911 | travel |
| 1911 | art |
| 4848 | gourmet cooking |

University of Glasgow

# Summary: ER -> Relations

- Strong entities
  - build a table with columns for each attribute [Step 1]
- Weak entities
  - build a table with columns for each attribute [Step 2]
  - Add the PK of the owner entity
- Relationships
  - 1-N: N side
  - N-M: new relation
  - 1-1: any side
- Sub-types
  1. Collapse to large supertype relation, OR
  2. Compose as 1-to-1 relationships

[Steps 3-5]

University of Glasgow

# Schemas with Domains

- Movie(Title, Type, Studio, SequelOf)
- Release(Title, Year, Length)
- Stars(Name, Address, Studio, Salary)
- Studio(Name, Address)
- ActsIn(Title, Year, StarName, bonus)

VARCHAR(xx)

INT

BIT

University of Glasgow

# Schemas with Domains

- Movie(Title, Type, Studio, SequelOf)
- Release(Title, Year, Length)
- Stars(Name, Address, Studio, Salary)
- Studio(Name, Address)
- ActsIn(Title, Year, StarName, bonus)

VARCHAR(xx)

INT

BIT

University of Glasgow

# Schemas with Domains

- Movie(Title : VARCHAR(50), Type : VARCHAR(50), Studio : VARCHAR(50), SequelOf : VARCHAR(50))
- Release(Title : VARCHAR(50), Year : Int, Length: Int)
- Stars(Name : VARCHAR(50), Address : VARCHAR(100), Studio VARCHAR(50), Salary : Int)
- Studio(Name : VARCHAR(50), Address : VARCHAR(100))
- ActsIn(Title : VARCHAR(50), Year : Int, StarName : VARCHAR(50), bonus : Int)

University of Glasgow

# Some Tips!

- Follow the stepwise guide – it works!
- Write a schema first – then go to the DBMS to build the tables
- Add the entities OWN attributes – then decide what FKs to add
- Be careful to select good data types – they must match when you go to connect PKs and FKs