Kurs
Datenbankgrundlagen und Modellierung

Sebastian Maneth, Universität Bremen
maneth@uni-bremen.de

Sommersemester 2023

24.4.2023
Vorlesung 2: ER-Diagrams & Creating Tables

Next week:

Monday 01.05.2023: keine Vorlesung (Tag der Arbeit)
Mittwoch 03.05.2023: keine Übung

Donnerstag 04.05.2023: 10:15—11:45 und
14:15—15:45 “Fragestunde” findet statt!

Next week:

Monday 01.05.2023: keine Vorlesung (Tag der Arbeit)
Mittwoch 03.05.2023: keine Übung

Donnerstag 04.05.2023: 10:15—11:45 und
14:15—15:45 “**Fragestunde**” findet statt!

Topics

— how to **run** `sqlite3`
— how to **create tables** in `sqlite3`
— how to **load a CSV-file** into a table in `sqlite3`

Agenda

1.) Entity Relationship Diagrams (ER Diagrams)

2.) The Relational Model

3.) Creating and Modifying Tables

— PRIMARY KEYs
— FOREIGN KEYs
— NOT NULL constraint
— data types
— adding/deleting rows/tables

1. Entity Relationship Model

→ high-level database model [Peter Chen (MIT) TODS 1, 1976]

→ useful for design before moving to a lower level model (e.g. relational)

— similar to class diagrams (but without methods & data types)

1. Entity Relationship Model

→ high-level database model [Peter Chen (MIT) TODS 1, 1976]

→ useful for design before moving to a lower level model (e.g. relational)

ER Model has

→ Structural part
- — entity types
- — attributes
- — relationship types

→ Integrity constraints
- — primary keys for entity and relationship types
- — multiplicity constraints for relationship types

1. Entity Relationship Model

→ high-level database model [Peter Chen (MIT) TODS 1, 1976]

→ useful for design before moving to a lower level model (e.g. relational)

ER Model has

→ Structural part
- — entity types
- — attributes
- — relationship types

→ Integrity constraints
- — primary keys for entity and relationship types
- — multiplicity constraints for relationship types

ER Diagrams

→ relatively simple
→ user-friendly
→ unified view of data, independent of any *implemented* data model

Next slides from Peter Wood's DB Management Lecture

Entity Types

Entity = a “thing” that exists and can be uniquely identified,
e.g. an individual person

Entity type = collection of similar entities, e.g., a collection of people (rectangle)

Entity Types

In terms of
UML Class Diagrams

Entity = a “thing” that exists and can be uniquely identified, e.g. an individual person

Object

Entity type = collection of similar entities, e.g., a collection of people (rectangle)

Class

Entity Types

In terms of
UML Class Diagrams

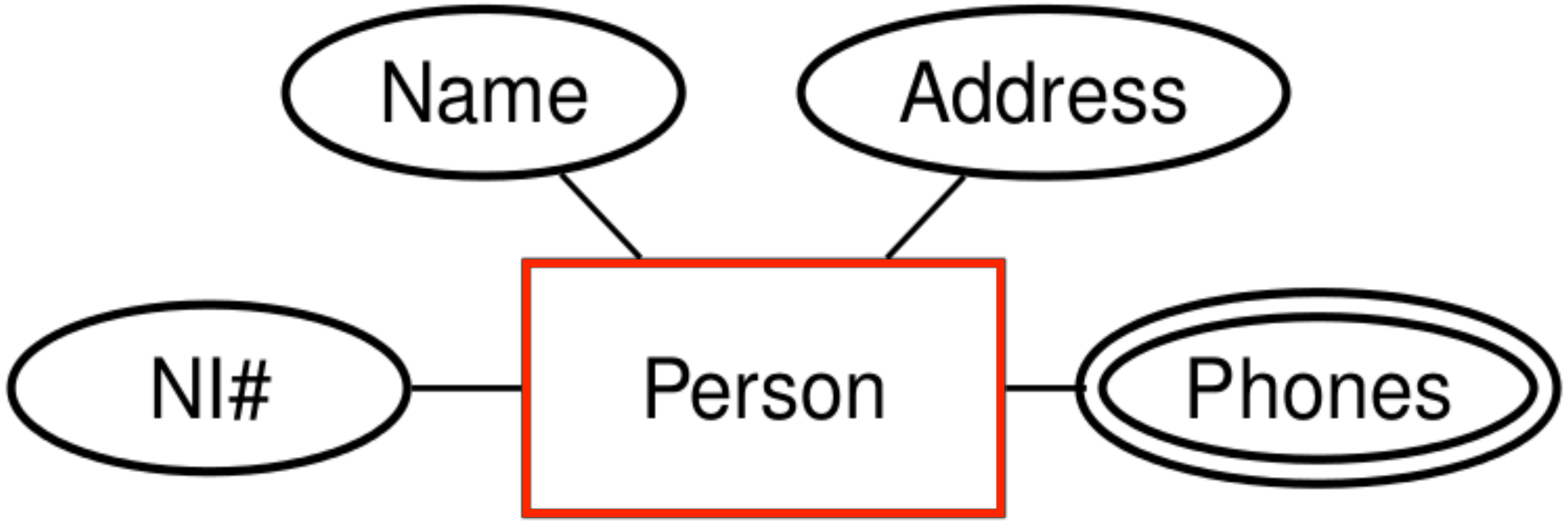
Entity = a "thing" that exists and can be uniquely identified, e.g. an individual person

Object

Entity type = collection of similar entities, e.g., a collection of people (rectangle)

Class

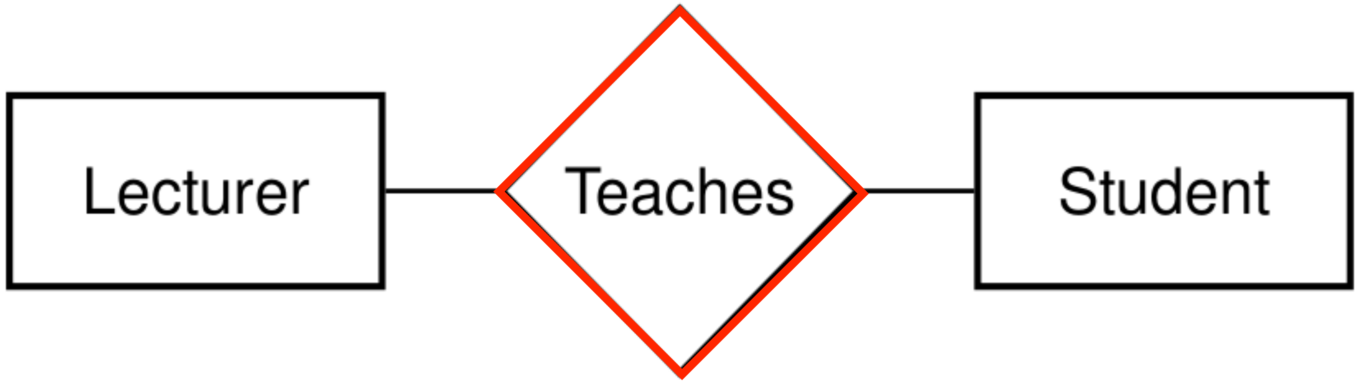
Entity type has attributes (circles), representing properties of the entities.



Each Person has single Name, Address, and Nat. Insurance number (NI#)
Each Person can have zero or more Phones

Relationship Types

Relationship Type = association between two or more entity types. (diamond)



Multiplicity Constraints in Relationship Types

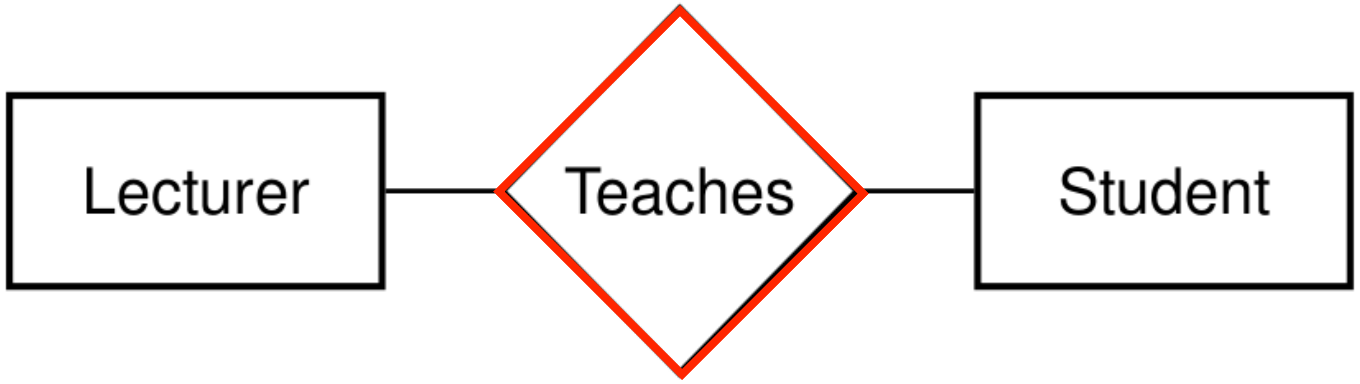
→ Many-to-One (or One-to-Many)

→ One-to-One

→ Many-to-Many

Relationship Types

Relationship Type = association between two or more entity types. (diamond)



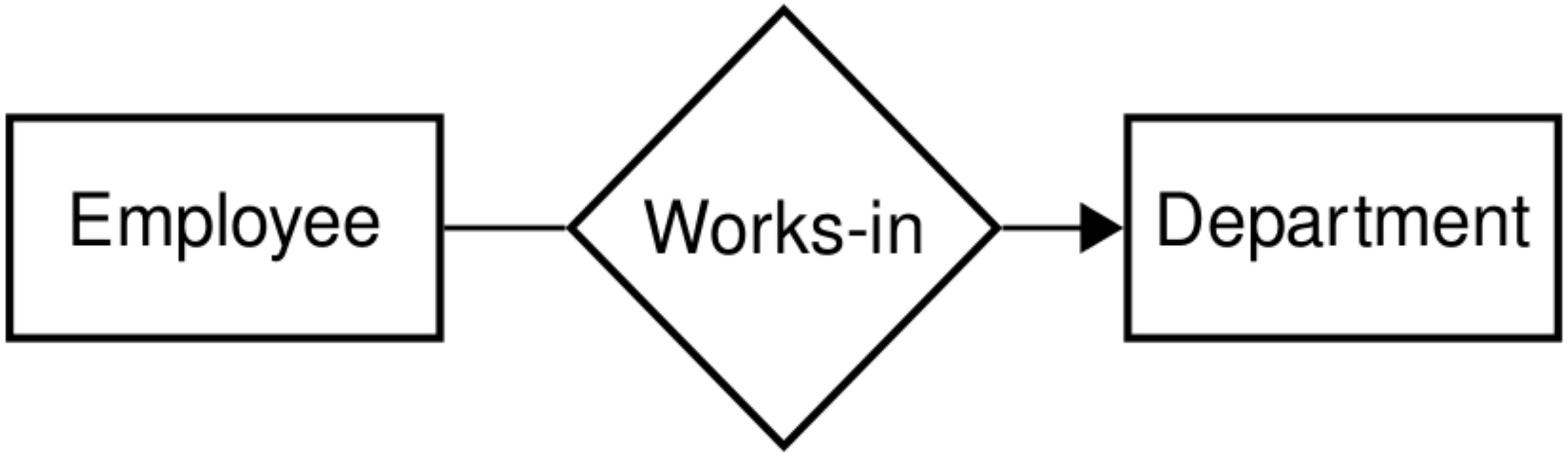
Multiplicity Constraints in Relationship Types

→ Many-to-One (or One-to-Many)
  An Employee Works in one Department.
  A Department has many Employees.

→ One-to-One
  A Manager Heads one Department and vice versa.

→ Many-to-Many
  A Lecturer Teaches many Students and a Student is taught by many Lecturers

Example of Many-to-One Relationship Type

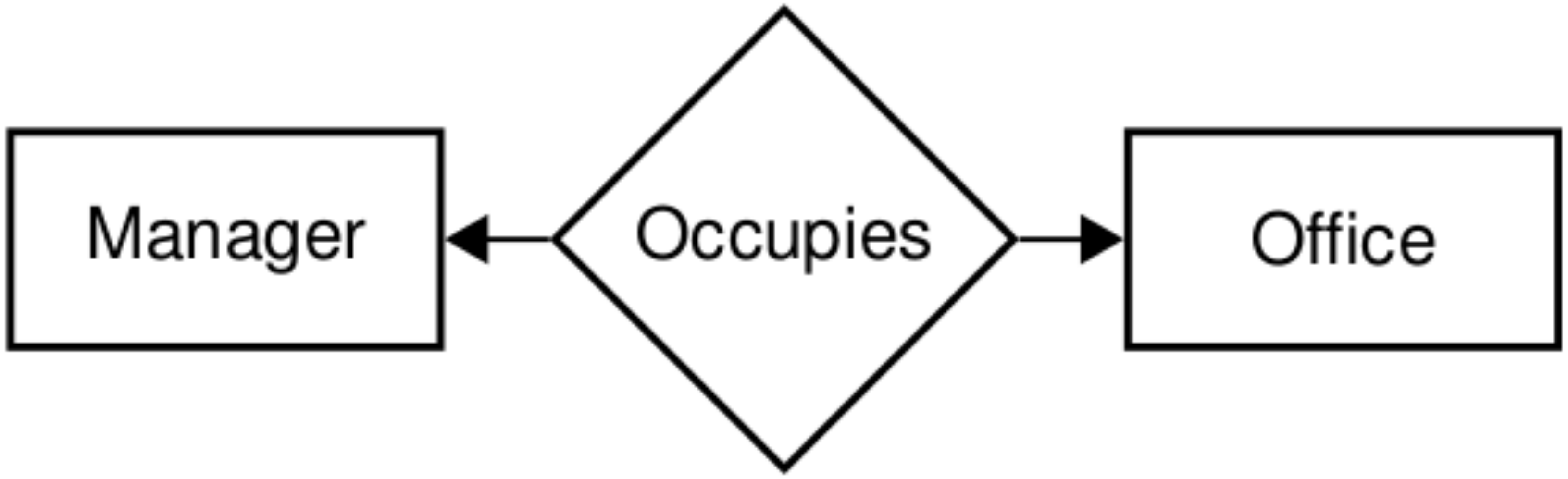


The arrowhead is drawn at the "one" end of rel. type

→ Each Employee Works-in one Department

→ Each Department has many Employees Working in it.

Example of One-to-One Relationship Type



The arrowhead is drawn at both ends

→ Each Manager Occupies one Office

→ Each Office has one Manager Occupying it

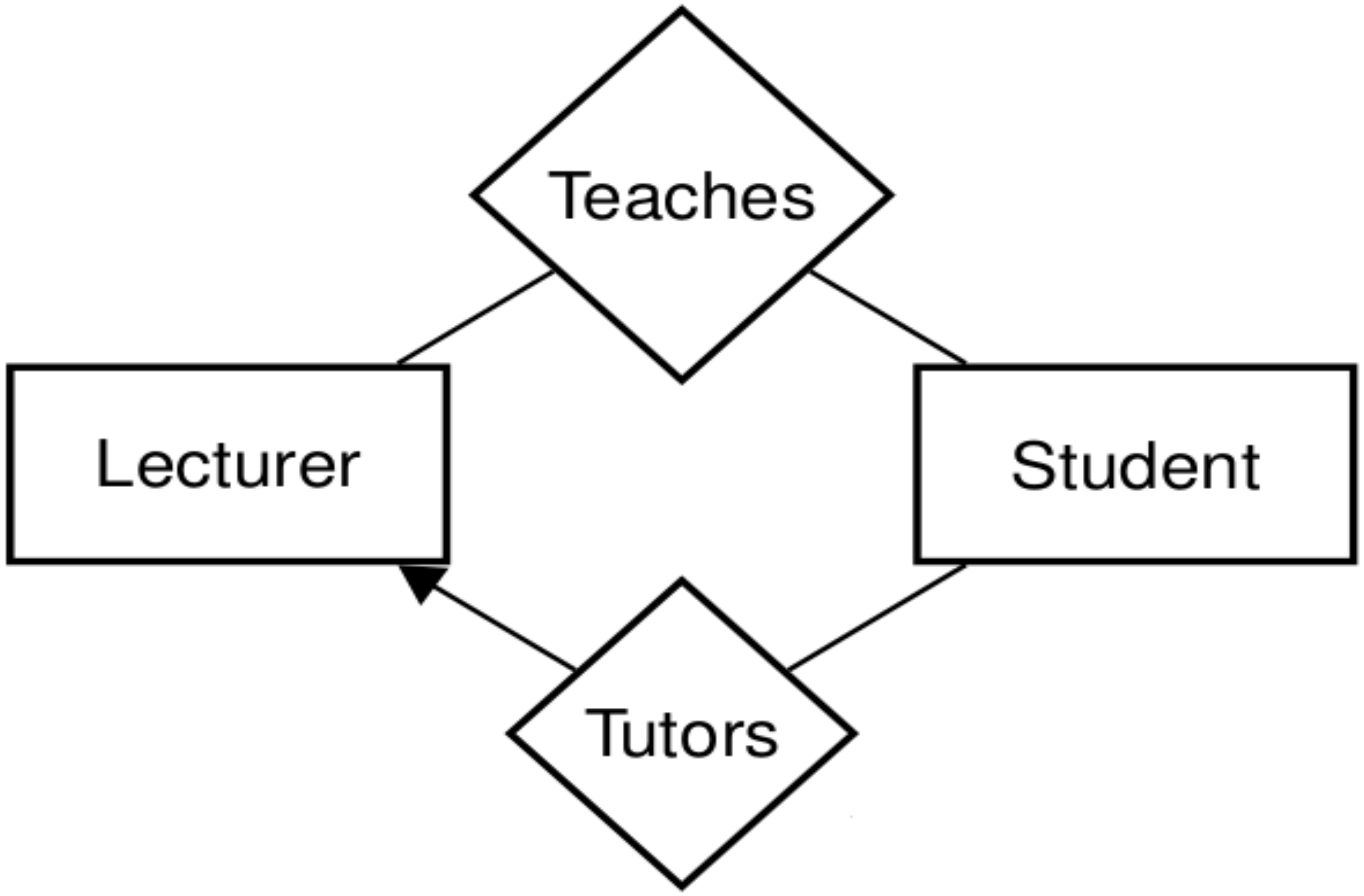
Example of Many-to-Many Relationship Type

No arrowheads

→ Each Lecturer Teaches many Students

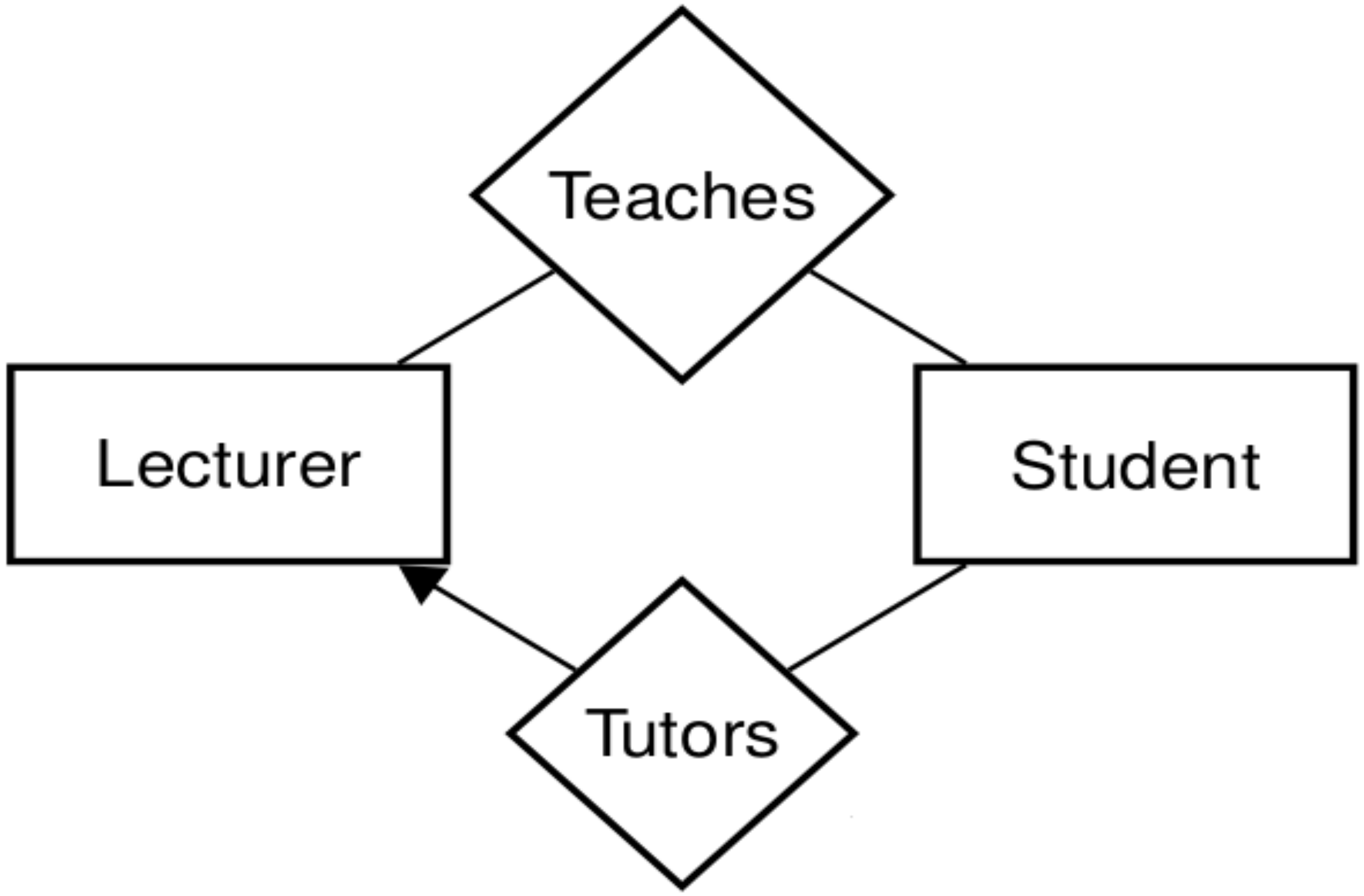
→ Each Student is taught by many Lecturers

Multiple Relationship Types



Explain the bottom part of the diagram.

Multiple Relationship Types



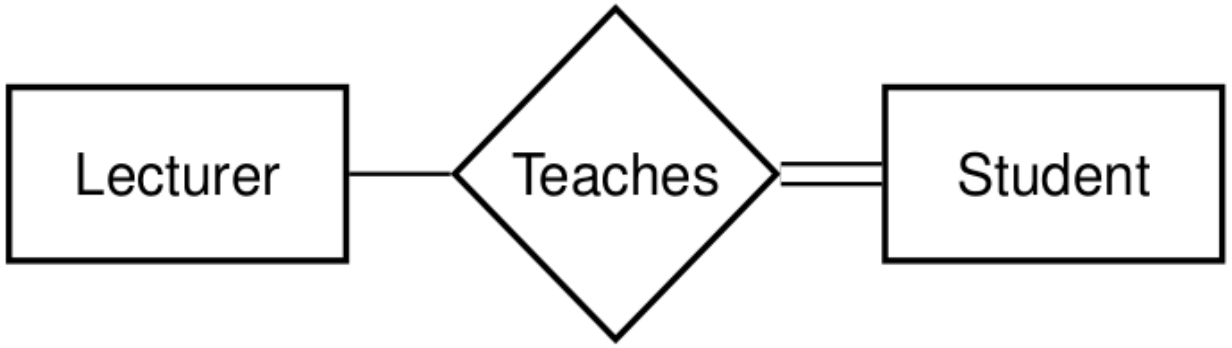
Explain the bottom part of the diagram.

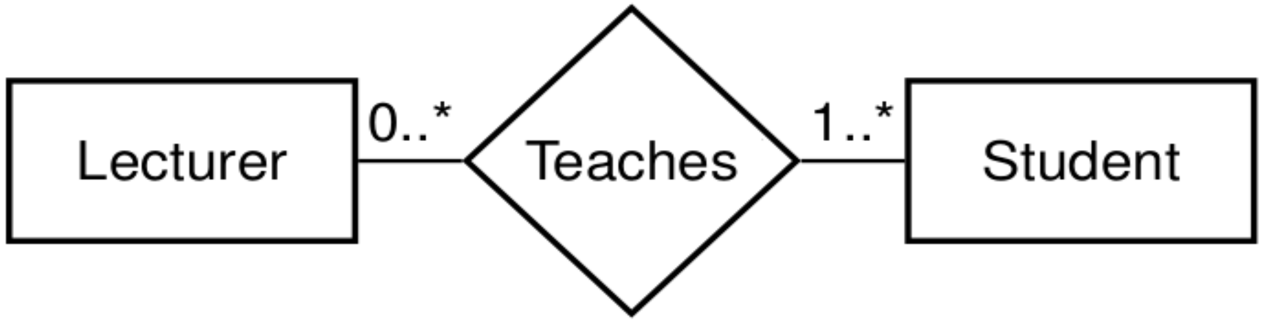
— each Student is tutored by exactly one Lecturer
(UK System: in many colleges / universities
such “personal tutors” exist)

Participation Constraints in Relationships

→ optional (default, sometimes indicated by multiplicity constraint 0..*)
e.g. Employee may or may not be assigned to a Department

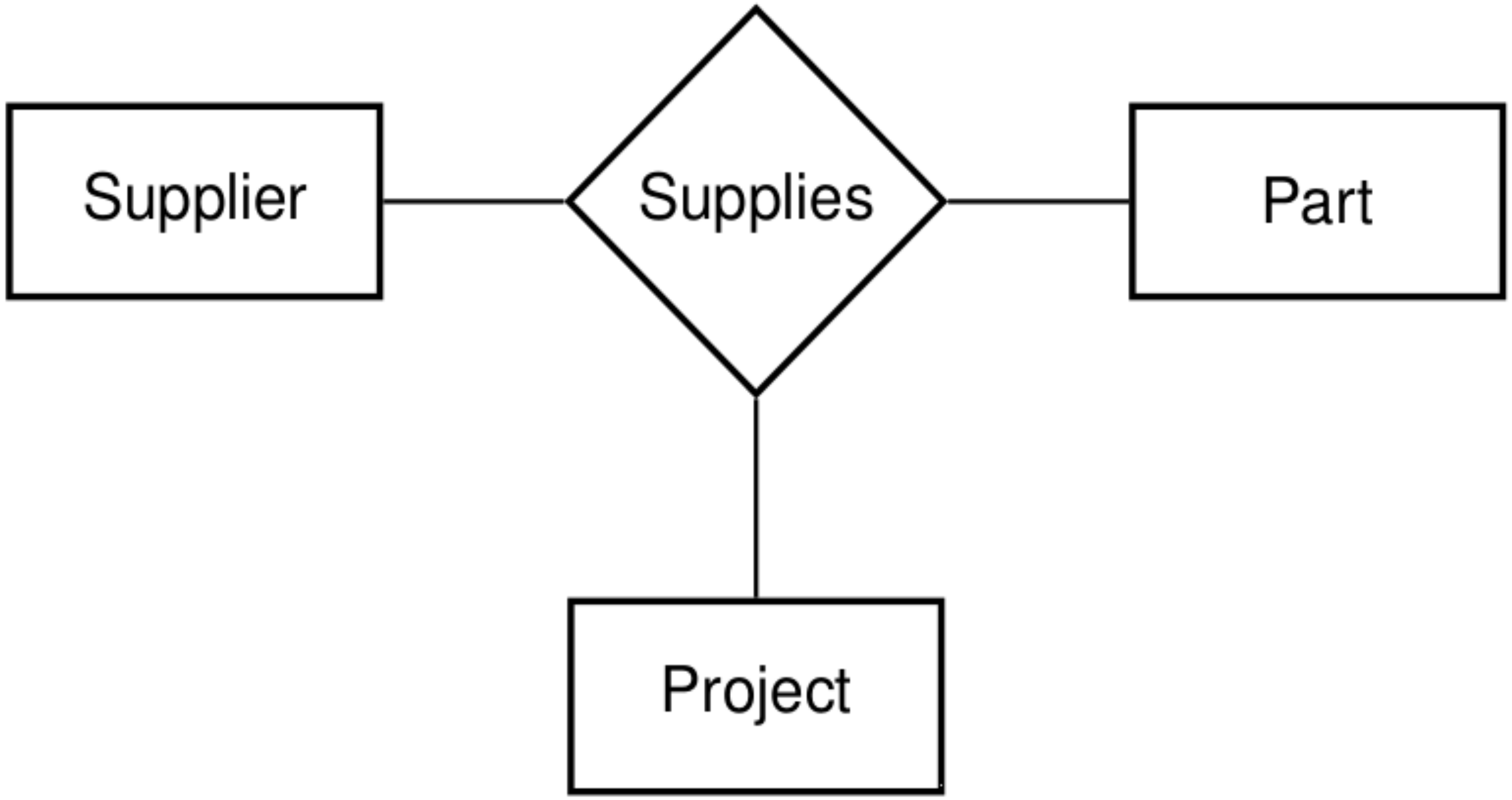
→ mandatory (double lines, or multiplicity constraint 1..*)





→ some Lecturers may not Teach any Students
→ each Student *must* be taught by at least one Lecturer

Multiway Relationship Types



→ each supplier may supply different parts to different projects

Keys and Superkeys

Superkey = Set of attributes of an entity type so that
for each entity e of that type,
the values of the attributes *uniquely* identifies e.

e.g. a Person may be uniquely identified by { Name, NI# }

Key = is a superkey which is *minimal* (aka "Candidate Key")

e.g., a Person is uniquely identified by the key { NI# }

"minimal" = if an attribute is removed,
then not a key anymore

Keys and Superkeys

Superkey = Set of attributes of an entity type so that
for each entity e of that type,
the values of the attributes *uniquely* identifies e.

e.g. a Person may be uniquely identified by { Name, NI# }

Key = is a superkey which is *minimal* (aka "Candidate Key")

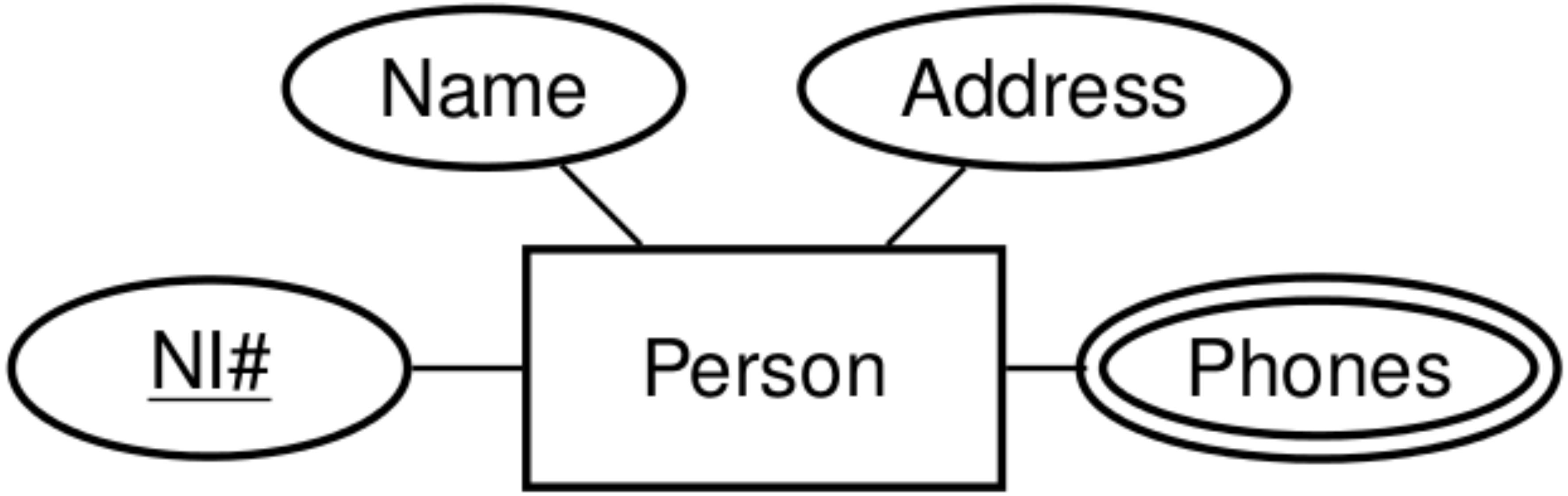
e.g., a Person is uniquely identified by the key { NI# }

Prime Attribute = attribute that appears **in a key**
Non-Prime Attribute = attribute that appears **in no key**

Primary Keys

Primary Key = a *key* that has been chosen as such by the database designer

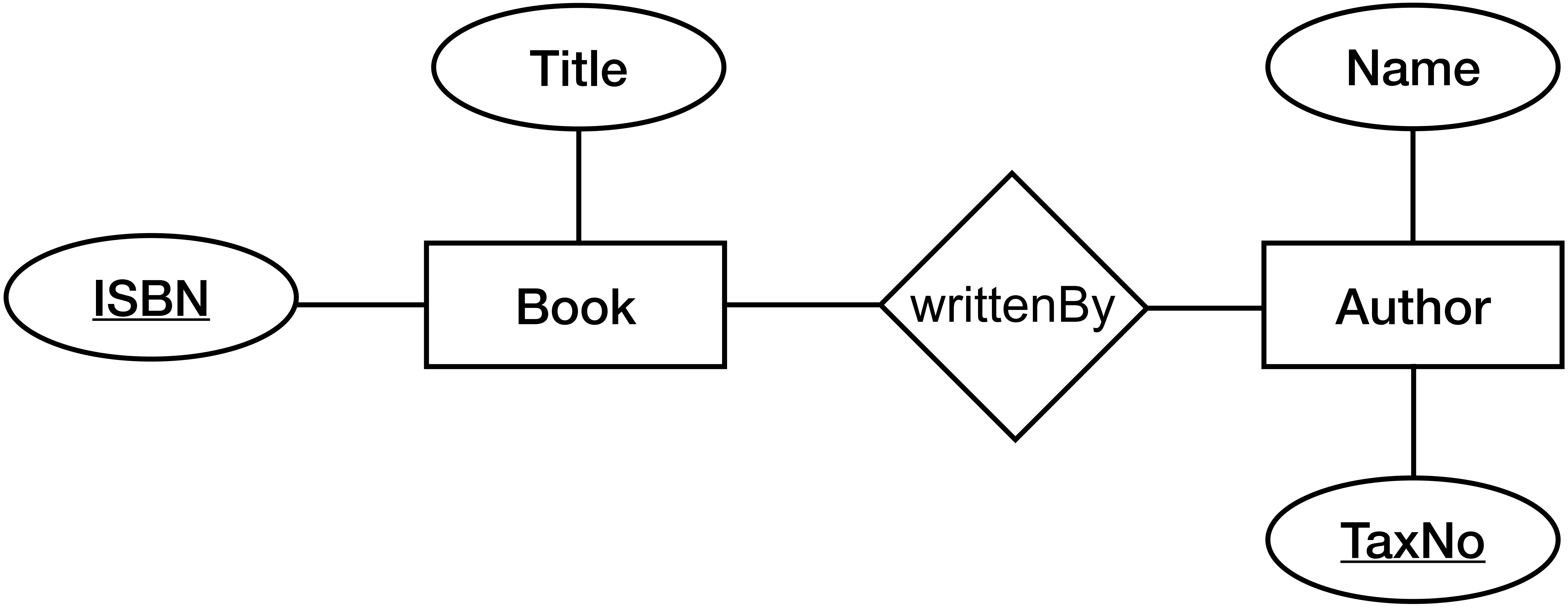
→ primary key guarantees logical access to every entity (attributes of a primary key are underlined)



Primary Keys

Primary Key = a *key* that has been chosen as such by the database designer

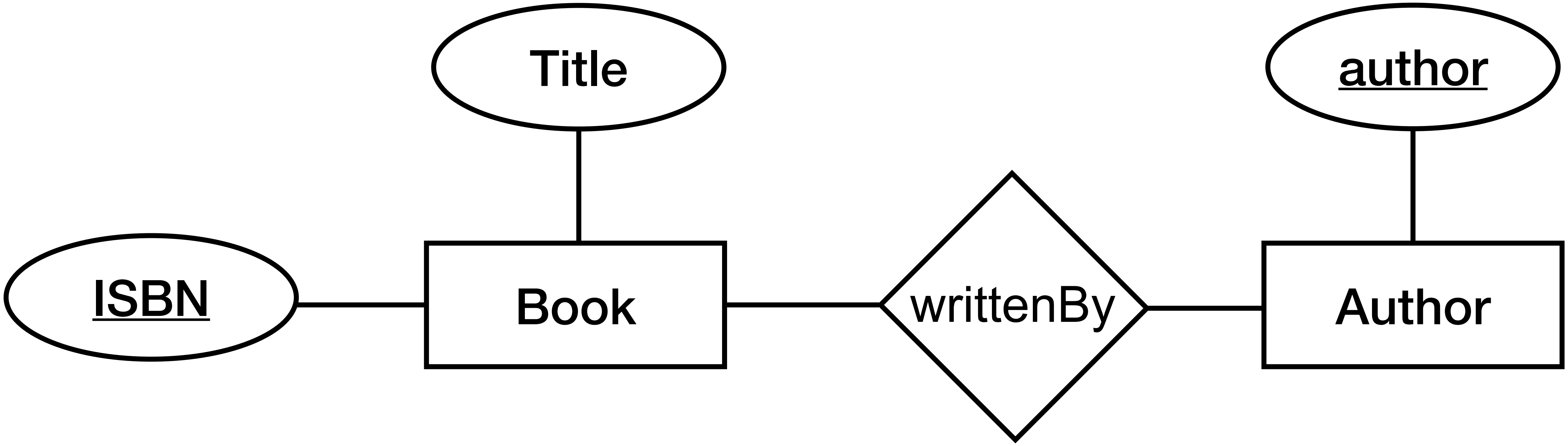
→ primary key guarantees logical access to every entity (attributes of a primary key are underlined)



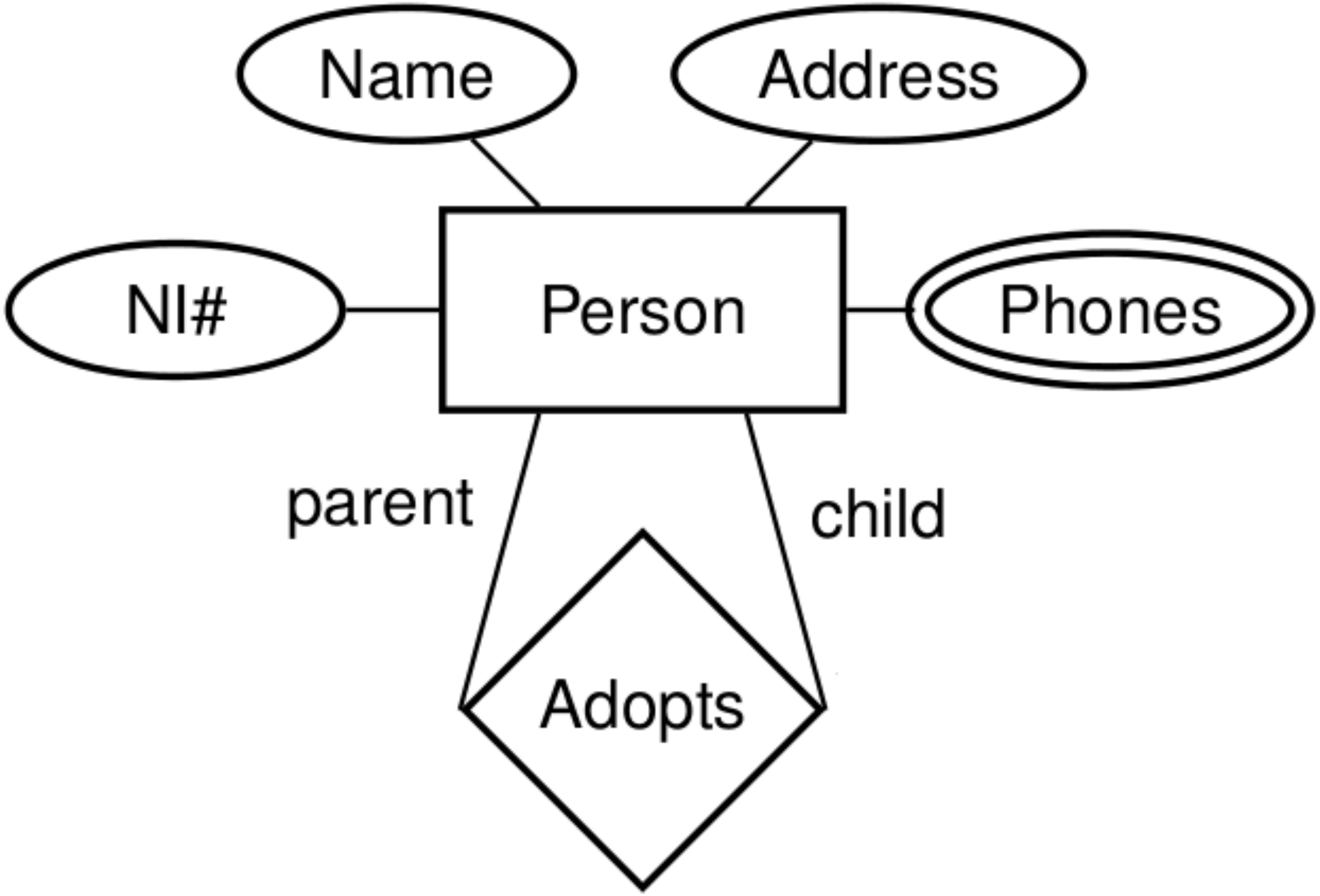
Primary Keys

Primary Key = a *key* that has been chosen as such by the database designer

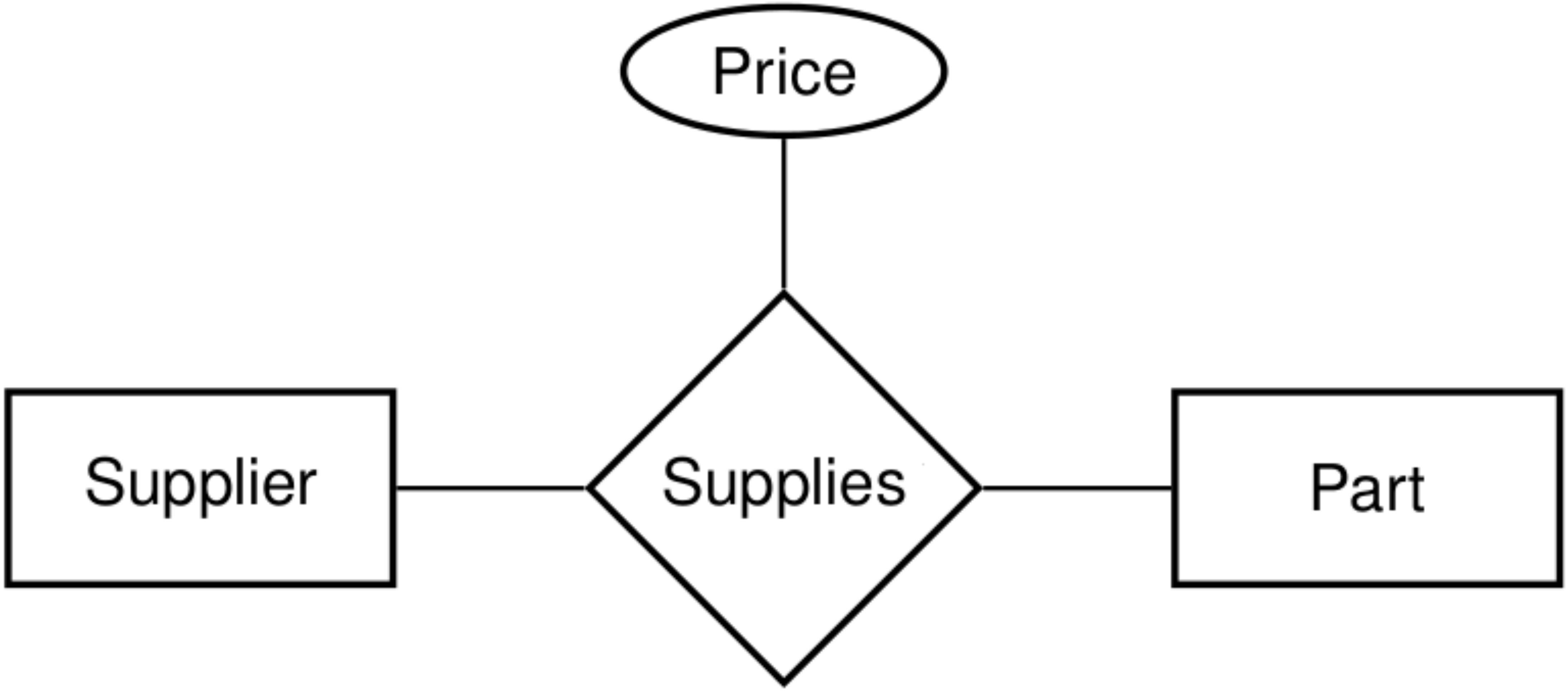
→ primary key guarantees logical access to every entity
(attributes of a primary key are underlined)



Cyclic Relationship Type with Roles



Relationship Type with Attributes



→ Each Supplier Supplies a Part at a certain Price

Weak Entity Types

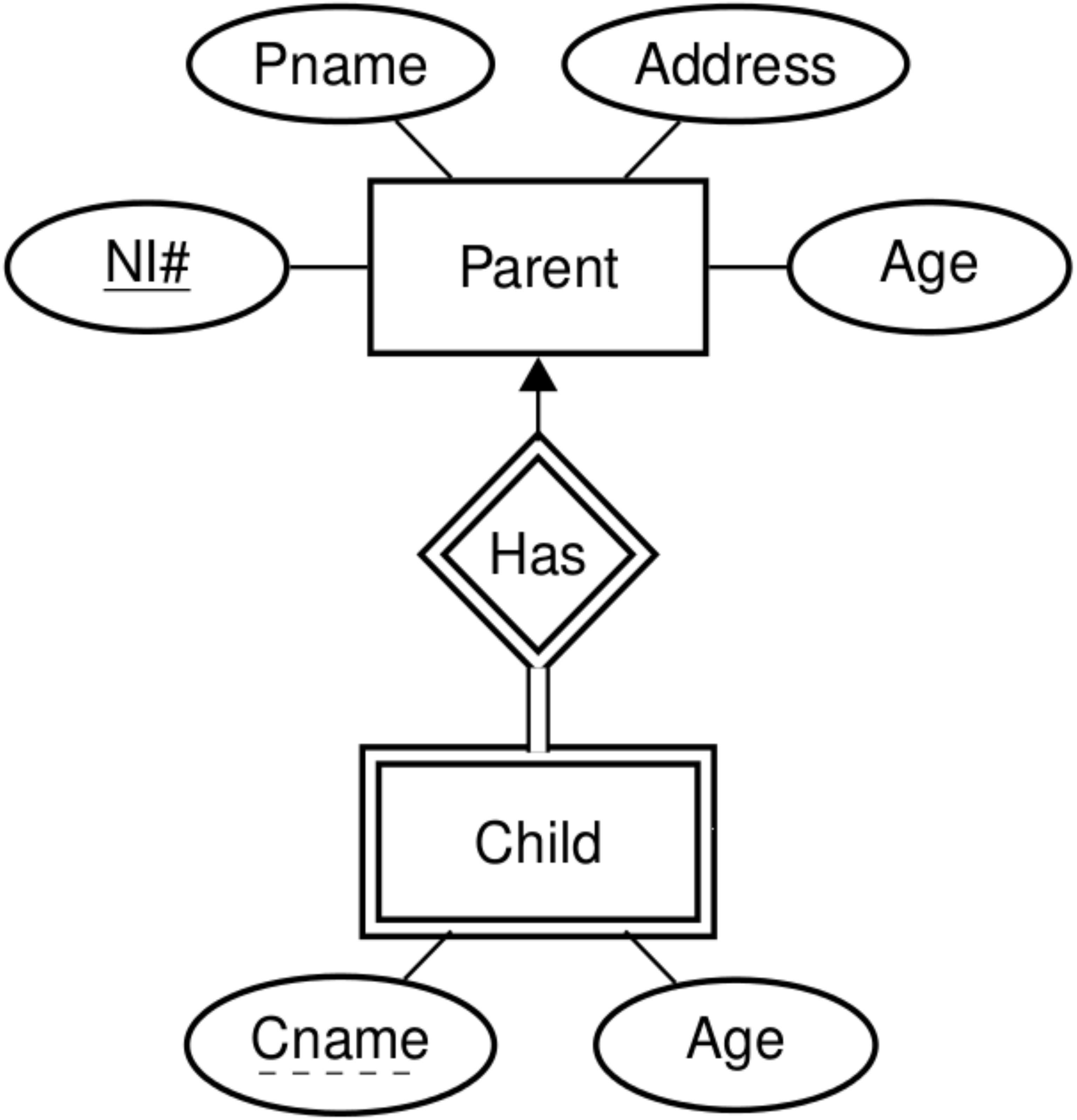
Weak Entity Type = an entity type that does not have sufficient attributes to form a primary key (double rectangle)

→ depends on the existence of an identifying entity type ("owner")
(they have an "identifying (ID) relationship" – double diamond)
→ must have a *discriminator* (dashed underline) for distinguishing its entities

E.g. in an employee database, Child entities exist only if their corresponding Parent employee entity exists.

The primary key of a weak entity type is the combination of the primary key of its owner type and its discriminator.

Weak Entity Types



ISA Relationship Types

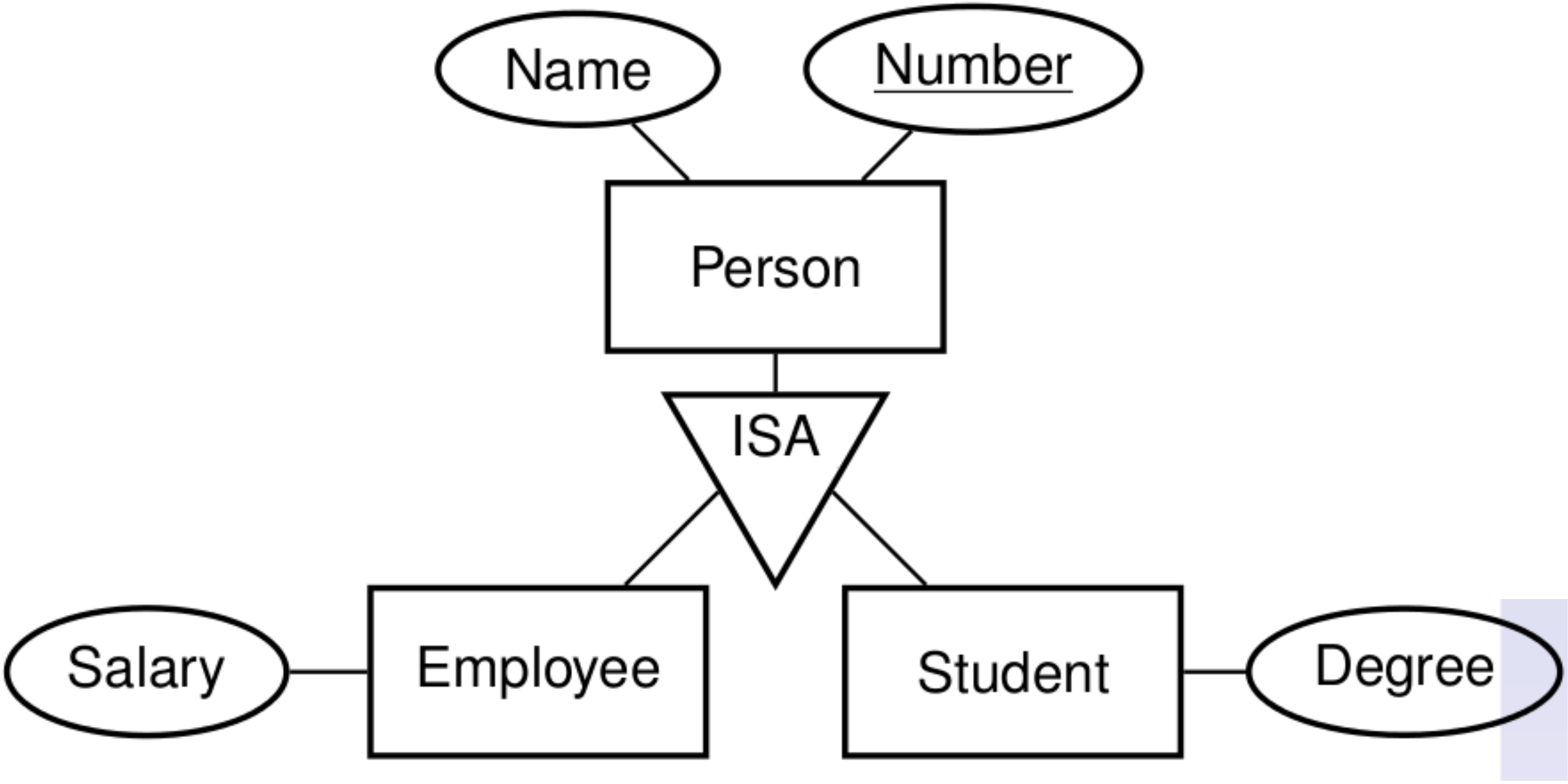
→ If entities of a type have special properties not shared by all entities, then this suggests two entity types with an ISA relationship between them

→ AKA generalization / specialization (supertype / subtype rel.)

E.g. an Employee ISA Person and a Student ISA Person

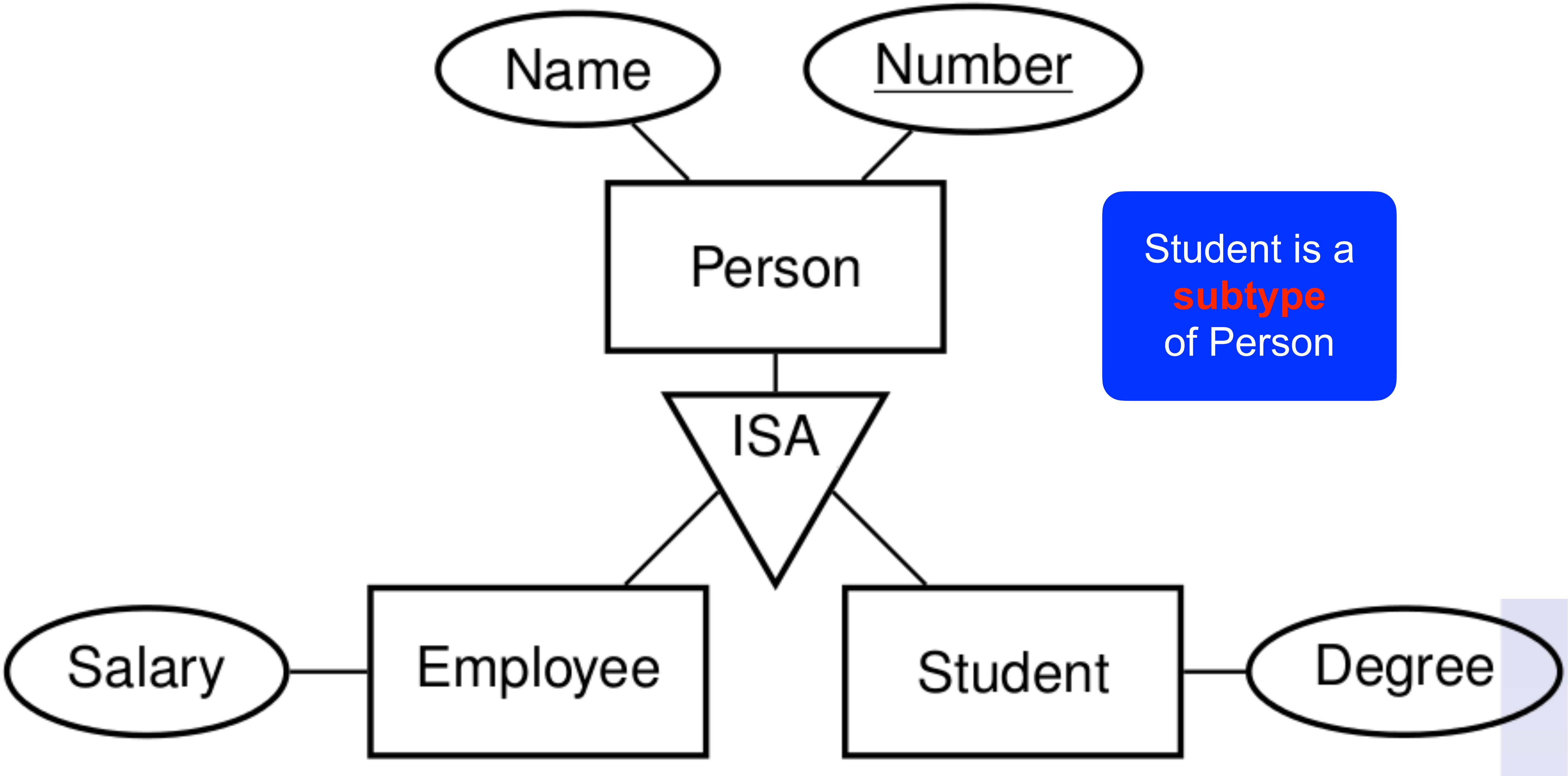
→ If Employee ISA Person, then Employee inherits all attributes of Person.

ISA Relationships



→ Attributes of Employee: Name, Number, and Salary.

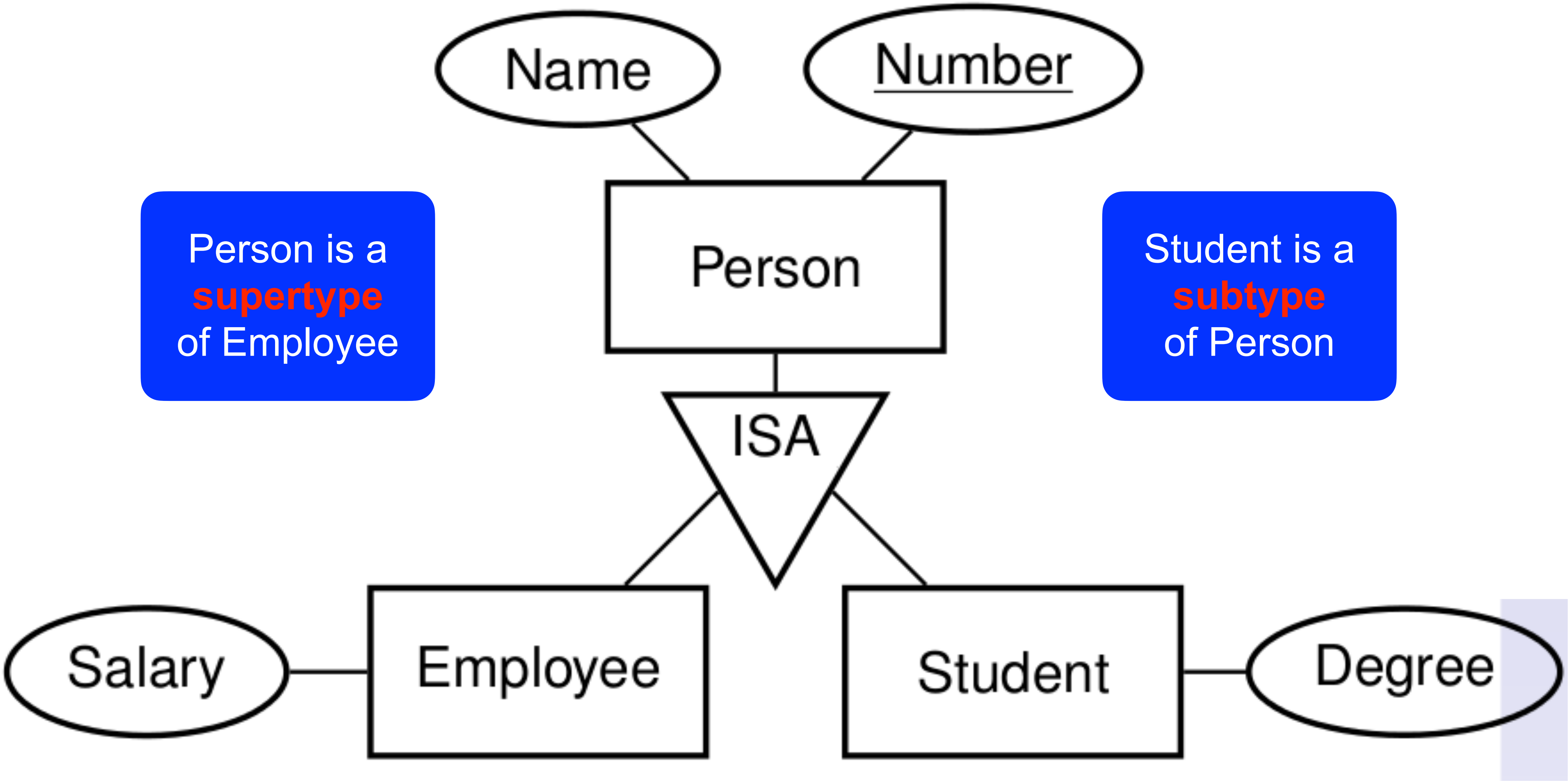
ISA Relationships



Student is a **subtype** of Person

→ Attributes of Employee: Name, Number, and Salary.

ISA Relationships

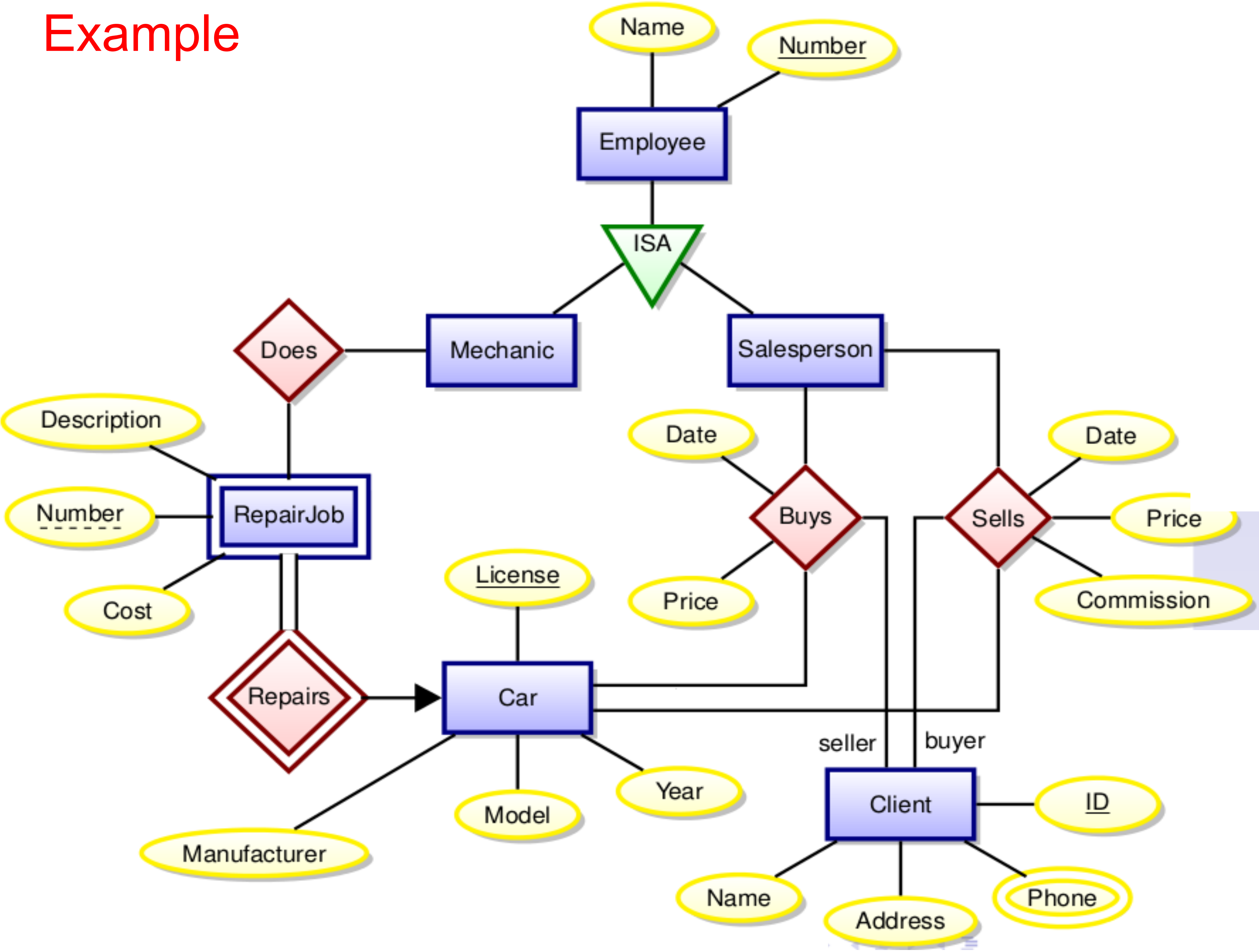


→ Attributes of Employee: Name, Number, and Salary.

Informal Methods for ERD Construction

1. Identify the entity types (including weak entity types) of the application.
2. Identify the relationship (including ISA and ID) types.
3. Classify each relationship type identified in step 2 according to its multiplicity, i.e. if it is a one-to-one, many-to-one or many-to-many.
4. Determine the participation constraints for each entity type in each relationship type.
5. Draw an ERD with the entity types and the relationship types between them.
6. Identify the attributes of entity and relationship types and their underlying domains
7. Identify a primary key for each entity type.
8. Add the attributes and primary keys to the ERD drawn in step 5.

Example


Name
Number
Employee
ISA
Does
Mechanic
Salesperson
Description
RepairJob
Number
Cost
Date
Buys
Price
Date
Sells
Price
Commission
License
Repairs
Car
seller
buyer
Client
ID
Manufacturer
Model
Year
Name
Address
Phone

Question

… once you have constructed a satisfactory Entity Relationship Diagram

— how do you construct relational tables that “fit” the Diagram?

2. The Relational Model

Poll: do you know what a Cartesian Product is?

2. The Relational Model

Poll: do you know what a Cartesian Product is?

A = { 1,2,3,4 }
B = { x,y,z }

The Cartesian product $A \times B$ of two sets A and B contains for every a in A and every b in B the pair (a,b).

2. The Relational Model

Poll: do you know what a Cartesian Product is?

A = { 1,2,3,4 }
B = { x,y,z }

The Cartesian product $A \times B$ of two sets A and B contains for every a in A and every b in B the pair (a,b).

A x B = { (1,x), (1,y), (1,z),
(2,x), (2,y), (2,z),
(3,x), (3,y), (3,z),
(4,x), (4,y), (4,z) }

2. The Relational Model

Poll: do you know what a Cartesian Product is?

A = { 1,2,3,4 }
B = { x,y,z }

The Cartesian product $A \times B$ of two sets A and B contains for every a in A and every b in B the pair (a,b).

A x B = { (1,x), (1,y), (1,z),
(2,x), (2,y), (2,z),
(3,x), (3,y), (3,z),
(4,x), (4,y), (4,z) }

The size | A x B | of the Cartesian product is |A| * |B|.

2. The Relational Model

Poll: do you know what a Cartesian Product is?

A = { 1,2,3,4 }
B = { x,y,z }

The Cartesian product $A \times B$ of two sets A and B contains for every a in A and every b in B the pair (a,b).

A x B = { (1,x), (1,y), (1,z),
(2,x), (2,y), (2,z),
(3,x), (3,y), (3,z),
(4,x), (4,y), (4,z) }

The size | A x B | of the Cartesian product is |A| * |B|.

A x (B x C) = (A x B) x C

2. The Relational Model

Let A_1, A_2, …, A_k be sets.
In mathematics, a relation r over A_1,…, A_k
is a subset of the cartesian product $A_1 \times A_2 \times \cdots \times A_k$

2. The Relational Model

Let $A_1, A_2, \ldots, A_k$ be sets.
In mathematics, a relation r over $A_1, \ldots, A_k$
is a subset of the cartesian product $A_1 \times A_2 \times \cdots \times A_k$

E.g.
A = { 1,2,3 }
B = { d,e,f }
C = { 5,6,7,8,9 }

r = { (1,e,9), (3,f,5) }

A **relation**
over
A,B, and C

2. The Relational Model

Let A_1, A_2, …, A_k be sets.
In mathematics, a relation r over A_1,…, A_k
is a subset of the cartesian product $A_1 \times A_2 \times \cdots \times A_k$

E.g.
A = { 1,2,3 }
B = { d,e,f }
C = { 5,6,7,8,9 }

r = { (1,e,9), (3,f,5) }

A **relation** over A,B, and C

Or
Z = { set of all ASCI strings }

r = { (“Donald Knuth”, “The Art of Computer Programming”), (“J. K. Rowling, “Harry Potter”) }

A **relation** over Z and Z

2. The Relational Model

An element of a relation is called a **tuple**.

E.g.
A = { 1,2,3 }
B = { d,e,f }
C = { 5,6,7,8,9 }

r = { (1,e,9), (3,f,5) }

tuple

Or
Z = { set of all ASCI strings }

r = { (“Donald Knuth”, “The Art of Computer Programming”),
(“J. K. Rowling, “Harry Potter”) }

tuple

2. The Relational Model

From a **mathematical point of view**, we can model a relational database D as follows:

2. The Relational Model

From a **mathematical point of view**, we can model a relational database D as follows:

D = (R, sch, dom, val)

A database consists of **four things**!

2. The Relational Model

From a **mathematical point of view**, we can model a relational database D as follows:

D = (R, sch, dom, val)

A database consists of **four things**!

— R is a finite set of relation names

2. The Relational Model

From a **mathematical point of view**, we can model a relational database D as follows:

D = (R, sch, dom, val)

A database consists of **four things**!

— R is a finite set of relation names

— for every r in R, sch(r) is a list of (pairwise distinct) attributes (“the **schema** of r”)

2. The Relational Model

From a **mathematical point of view**, we can model a relational database D as follows:

D = (R, sch, dom, val)

A database consists of **four things**!

— R is a finite set of relation names

— for every r in R, sch(r) is a list of (pairwise distinct) attributes (“the **schema** of r”)

— If sch(r)=(A_1,…,A_k) then dom(r, A_i) is a set of values for every i=1…k (“the **domain** of A_i in r”)

2. The Relational Model

From a **mathematical point of view**, we can model a relational database D as follows:

D = (R, sch, dom, val)

A database consists of **four things**!

— R is a finite set of relation names

— for every r in R, sch(r) is a list of (pairwise distinct) attributes ("the **schema** of r")

— If sch(r)=(A_1,…,A_k) then dom(r, A_i) is a set of values for every i=1…k ("the **domain** of A_i in r")

and val(r) is a **relation** over dom(r,A_1), …, dom(r,A_k)

2. The Relational Model

From a **mathematical point of view**, we can model a relational database D as follows:

A database consists of **four things**!

D = (R, sch, dom, val)

— R is a finite set of relation names

— for every r in R, sch(r) is a list of (pairwise distinct) attributes (“the **schema** of r”)

— If sch(r)=(A_1,…,A_k) then dom(r, A_i) is a set of values for every i=1…k (“the **domain** of A_i in r”)

and val(r) is a **relation** over dom(r,A_1), …, dom(r,A_k)

$$\text{i.e., } \mathrm{val}(r) \subseteq \mathrm{dom}(r, A_1) \times \cdots \times \mathrm{dom}(r, A_k).$$

3. Creating Tables

3. Creating Tables

Ingredients			
Name	Alcohol	InStock	Price
Orange Juice	0.0	12	2.99
Campari	25.0	5	12.95
Bacardi	37.5	3	16.98

relation or table

schema

record, row, or tuple

field, column, or attribute

Tables of RDBMSs (e.g., MySQL, SQLite3, PostgreSQL) **can** represent relational databases according to the Relational Model.

3. Creating Tables

Ingredients			
Name	Alcohol	InStock	Price
Orange Juice	0.0	12	2.99
Campari	25.0	5	12.95
Bacardi	37.5	3	16.98

relation or table

schema

record, row, or tuple

field, column, or attribute

Tables of RDBMSs (e.g., MySQL, SQLite3, PostgreSQL)
can represent relational databases according to the Relational Model.

R = { Ingredients }
sch(Ingredients) = (Name, Alcohol, InStock, Price)
dom(Ingredients, Name) = `VARCHAR(50)`
dom(Ingredients,Alcohol) = `DECIMAL(3,1)`
dom(Ingredients,InStock) = `INT`
dom(Ingredients,Price) = `DECIMAL(6,2)`
val(Ingredient) = { (‘Orange Juice’, 0.0, 12, 2.99), (‘Campari’, 25.0, 5, 12.95) … }

smaneth — screen ▸ sqlite3 — 88×25

```
bash-3.2$ sqlite3
SQLite version 3.34.0 2020-12-01 16:14:00
Enter ".help" for usage hints.
Connected to a transient in-memory database.
Use ".open FILENAME" to reopen on a persistent database.
sqlite>
```

smaneth — screen ▸ sqlite3 — 88×25

```
bash-3.2$ sqlite3
SQLite version 3.34.0 2020-12-01 16:14:00
Enter ".help" for usage hints.
Connected to a transient in-memory database.
Use ".open FILENAME" to reopen on a persistent database.
sqlite> CREATE TABLE Ingredients(Name VARCHAR(50), Alcohol DECIMAL(3,1), InStock INT, Pr
ice DECIMAL(6,2));
sqlite>
```

smaneth — screen ▸ sqlite3 — 88×25

```
bash-3.2$ sqlite3
SQLite version 3.34.0 2020-12-01 16:14:00
Enter ".help" for usage hints.
Connected to a transient in-memory database.
Use ".open FILENAME" to reopen on a persistent database.
sqlite> CREATE TABLE Ingredients(Name VARCHAR(50), Alcohol DECIMAL(3,1), InStock INT, Pr
ice DECIMAL(6,2));
sqlite> INSERT INTO Ingredients VALUES('Orange Juice',0.0,12,2.99);
sqlite> 
```

smaneth — screen ▸ sqlite3 — 88×25

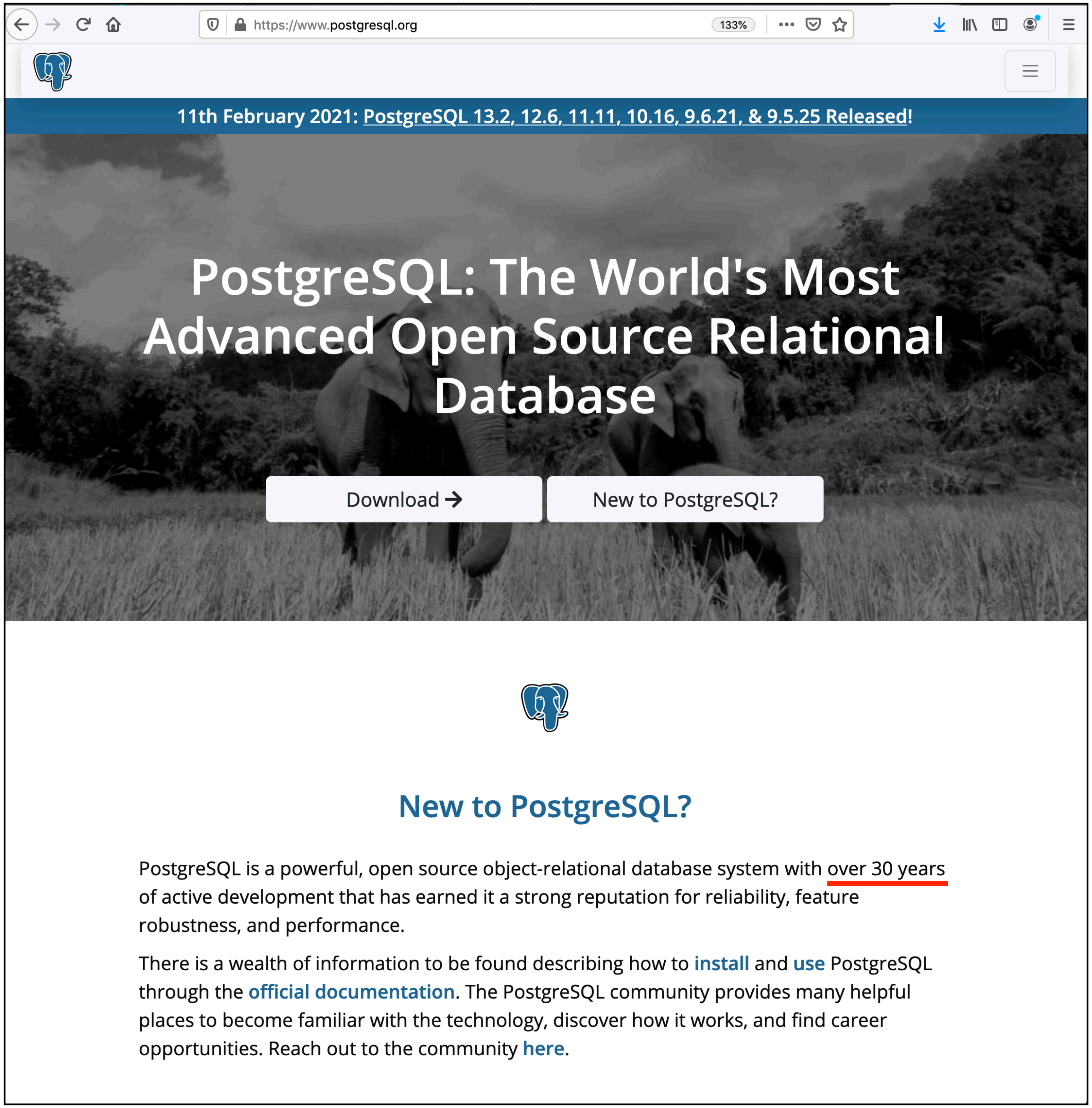
```
bash-3.2$ sqlite3
SQLite version 3.34.0 2020-12-01 16:14:00
Enter ".help" for usage hints.
Connected to a transient in-memory database.
Use ".open FILENAME" to reopen on a persistent database.
sqlite> CREATE TABLE Ingredients(Name VARCHAR(50), Alcohol DECIMAL(3,1), InStock INT, Pr
ice DECIMAL(6,2));
sqlite> INSERT INTO Ingredients VALUES('Orange Juice',0.0,12,2.99);
sqlite> select * from Ingredients;
Orange Juice|0|12|2.99
sqlite> .mode table
sqlite> select * from Ingredients;
+--------------+---------+---------+-------+
|     Name     | Alcohol | InStock | Price |
+--------------+---------+---------+-------+
| Orange Juice | 0       | 12      | 2.99  |
+--------------+---------+---------+-------+
sqlite>
```

smaneth — screen ▸ sqlite3 — 88×25

```
SQLite version 3.34.0 2020-12-01 16:14:00
Enter ".help" for usage hints.
Connected to a transient in-memory database.
Use ".open FILENAME" to reopen on a persistent database.
sqlite> CREATE TABLE Ingredients(Name VARCHAR(50), Alcohol DECIMAL(3,1), InStock INT, Pr
ice DECIMAL(6,2));
sqlite> INSERT INTO Ingredients VALUES('Orange Juice',0.0,12,2.99);
sqlite> select * from Ingredients;
Orange Juice|0|12|2.99
sqlite> .mode table
sqlite> select * from Ingredients;
+--------------+---------+---------+-------+
|     Name     | Alcohol | InStock | Price |
+--------------+---------+---------+-------+
| Orange Juice | 0       | 12      | 2.99  |
+--------------+---------+---------+-------+
sqlite> INSERT INTO Ingredients VALUES(10,'abc','def','');
sqlite> select * from Ingredients;
+--------------+---------+---------+-------+
|     Name     | Alcohol | InStock | Price |
+--------------+---------+---------+-------+
| Orange Juice | 0       | 12      | 2.99  |
| 10           | abc     | def     |       |
+--------------+---------+---------+-------+
sqlite>
```

Attention!
SQLite3 does **not** check types!

Let’s use a better RDBMS: PostgreSQL



smaneth — screen ▸ sqlite3 — 88×25

```
dblp=# CREATE Table Ingredients(Name VARCHAR(50), Alcohol DECIMAL(3,1), InStock INT, Pri
ce DECIMAL(6,2));
CREATE TABLE
dblp=# INSERT INTO Ingredients VALUES('Orange Juice',0.0,12,2.99);
INSERT 0 1
dblp=# select * from Ingredients;
     name     | alcohol | instock | price
--------------+---------+---------+-------
 Orange Juice |     0.0 |      12 |  2.99
(1 row)

dblp=# INSERT INTO Ingredients VALUES(10,'abc','def','');
ERROR:  invalid input syntax for type numeric: "abc"
LINE 1: INSERT INTO Ingredients VALUES(10,'abc','def','');
                                          ^

dblp=#
```

3. Creating Tables

Ingredients			
Name	Alcohol	InStock	Price
Orange Juice	0.0	12	2.99
Campari	25.0	5	12.95
Bacardi	37.5	3	16.98

relation or table

schema

record, row, or tuple

field, column, or attribute

Tables of RDBMSs (e.g., MySQL, SQLite3, PostgreSQL)
can represent relational databases according to the Relational Model.

There are Tables in RDBMSs that
do not correspond to relations in the Relational Model.

— any ideas what those could be?

There are Tables in RDBMSs that
do not correspond to relations in the Relational Model.

For exactly two reasons:

Tables in RDBMSs may contain

1.) **duplicates**

2.) **NULL values**.

smaneth — screen ▸ sqlite3 — 88×25

```
dblp=# SELECT * from Ingredients;
     name     | alcohol | instock | price
--------------+---------+---------+-------
 Orange Juice |     0.0 |      12 |  2.99
(1 row)

dblp=# INSERT INTO Ingredients VALUES('Orange Juice', 0.0, 12, 2.99);
INSERT 0 1
dblp=# SELECT * from Ingredients;
     name     | alcohol | instock | price
--------------+---------+---------+-------
 Orange Juice |     0.0 |      12 |  2.99
 Orange Juice |     0.0 |      12 |  2.99
(2 rows)

dblp=#
```

duplicate
(not possible in a relation)

smaneth — screen ▸ sqlite3 — 88×25

```
--------------+--------+--------+-------
 Orange Juice |    0.0 |      12 |  2.99
(1 row)

dblp=# INSERT INTO Ingredients VALUES('Orange Juice', 0.0, 12, 2.99);
INSERT 0 1
dblp=# SELECT * from Ingredients;
     name     | alcohol | instock | price
--------------+--------+--------+-------
 Orange Juice |    0.0 |      12 |  2.99
 Orange Juice |    0.0 |      12 |  2.99
(2 rows)

dblp=# INSERT INTO Ingredients SELECT * from Ingredients;
INSERT 0 2
dblp=# SELECT * from Ingredients;
     name     | alcohol | instock | price
--------------+--------+--------+-------
 Orange Juice |    0.0 |      12 |  2.99
 Orange Juice |    0.0 |      12 |  2.99
 Orange Juice |    0.0 |      12 |  2.99
 Orange Juice |    0.0 |      12 |  2.99
(4 rows)

dblp=#
```

Instead of VALUES you can INSERT results of Queries!

```
dblp=# INSERT INTO Ingredients SELECT * from Ingredients;
INSERT 0 2
dblp=# SELECT * from Ingredients;
     name     | alcohol | instock | price
--------------+---------+---------+-------
 Orange Juice |     0.0 |      12 |  2.99
 Orange Juice |     0.0 |      12 |  2.99
 Orange Juice |     0.0 |      12 |  2.99
 Orange Juice |     0.0 |      12 |  2.99
(4 rows)

dblp=# INSERT INTO Ingredients VALUES('Orange Juice', NULL, NULL, 3.99);
INSERT 0 1
dblp=# SELECT * from Ingredients;
     name     | alcohol | instock | price
--------------+---------+---------+-------
 Orange Juice |     0.0 |      12 |  2.99
 Orange Juice |     0.0 |      12 |  2.99
 Orange Juice |     0.0 |      12 |  2.99
 Orange Juice |     0.0 |      12 |  2.99
 Orange Juice |         |         |  3.99
(5 rows)

dblp=#
```

NULLs
(not possible in a relation)

smaneth — screen ▸ sqlite3 — 88×25

```
dblp=# INSERT INTO Ingredients SELECT * from Ingredients;
INSERT 0 2
dblp=# SELECT * from Ingredients;
     name     | alcohol | instock | price
--------------+---------+---------+-------
 Orange Juice |     0.0 |      12 |  2.99
 Orange Juice |     0.0 |      12 |  2.99
 Orange Juice |     0.0 |      12 |  2.99
 Orange Juice |     0.0 |      12 |  2.99
(4 rows)

dblp=# INSERT INTO Ingredients VALUES('Orange Juice', NULL, NULL, 3.99);
INSERT 0 1
dblp=# SELECT * from Ingredients;
     name     | alcohol | instock | price
--------------+---------+---------+-------
 Orange Juice |     0.0 |      12 |  2.99
 Orange Juice |     0.0 |      12 |  2.99
 Orange Juice |     0.0 |      12 |  2.99
 Orange Juice |     0.0 |      12 |  2.99
 Orange Juice |         |         |  3.99
(5 rows)

dblp=#
```

NULLs
(not possible in a relation)

Note: in the display NULL and “” (empty string) look the same! (but they are not)

There are Tables in RDBMSs that
do not correspond to relations in the Relational Model.

For exactly two reasons:

Tables in RDBMSs may contain

1.) duplicates

2.) NULL values.

NULL is a condition.
NULL means: there is a value here, but we do not know it!

There are Tables in RDBMSs that
do not correspond to relations in the Relational Model.

For exactly two reasons:

Tables in RDBMSs may contain

1.) duplicates

2.) NULL values.

NULL means: there is a value here, but we do not know it!

— we will **never** design tables that contain duplicates.
— try to avoid NULL values whenever possible!!!

Behavior of Null Values

In operations and predicates, think of **null** as **"unknown"**:

and	true	unknown	false
true	true	unknown	false
unknown	unknown	unknown	false
false	false	false	false

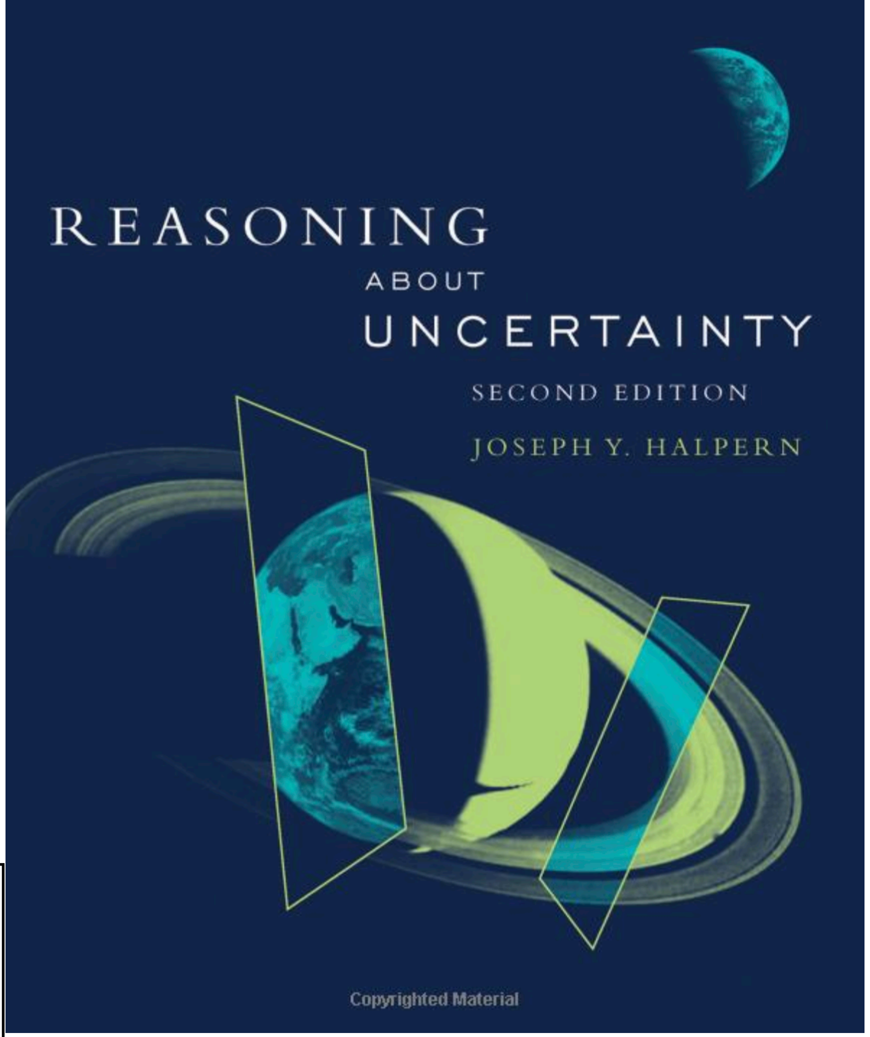
or	true	unknown	false
true	true	true	true
unknown	true	unknown	unknown
false	true	unknown	false

- **Arithmetic operations** with **null** evaluate to **null** ($\textbf{null} + 42 \rightarrow \textbf{null}$).
- **Comparisons** with **null** evaluate to **null** ($\mathit{Semester} < \textbf{null} \rightarrow \textbf{null}$).

— reasoning about NULLs (Uncertainty) quickly gets very complicated!!
— Therefore, avoid NULLs!!

Literature on NULLs:

1) there is a whole area (within logic) on "Reasoning about Uncertainty"



2) check these papers by Libkin et al:

Correctness of SQL Queries on Databases with Nulls

Paolo Guagliardo
School of Informatics
The University of Edinburgh
pguaglia@inf.ed.ac.uk

Leonid Libkin
School of Informatics
The University of Edinburgh
libkin@inf.ed.ac.uk

ABSTRACT

Multiple issues with SQL's handling of nulls have been well documented. Having efficiency as its main goal, SQL disregards the standard notion of correctness on incomplete databases – certain answers – due to its high complexity. As a result, the evaluation of SQL queries on databases with nulls may produce answers that are just plain wrong. However, SQL evaluation can be modified, at least for relational algebra queries, to approximate certain answers, i.e., return only correct answers. We examine recently proposed approximation schemes for certain answers and analyze their complexity, both theoretical bounds and real-life behavior.

1. INTRODUCTION

The way incomplete information is handled in commercial DBMSs, specifically by SQL, has been heavily criticized for producing counter-intuitive and just plain incorrect answers [4, 9]. This is often blamed on SQL's 3-valued logic (3VL), and there are multiple discussions if we deal with relational calculus/algebra queries [2]. On the other hand, SQL evaluation is very efficient; it is in AC^0 (a small parallel complexity class) for the same class of queries, and so it provably cannot compute certain answers.

If SQL provably cannot produce what is assumed to be *the* correct answers, then what kinds of errors can it generate? To understand this, consider the simple database in Figure 1. It shows orders for books, information about customers paying for them, and basic information about customers themselves.

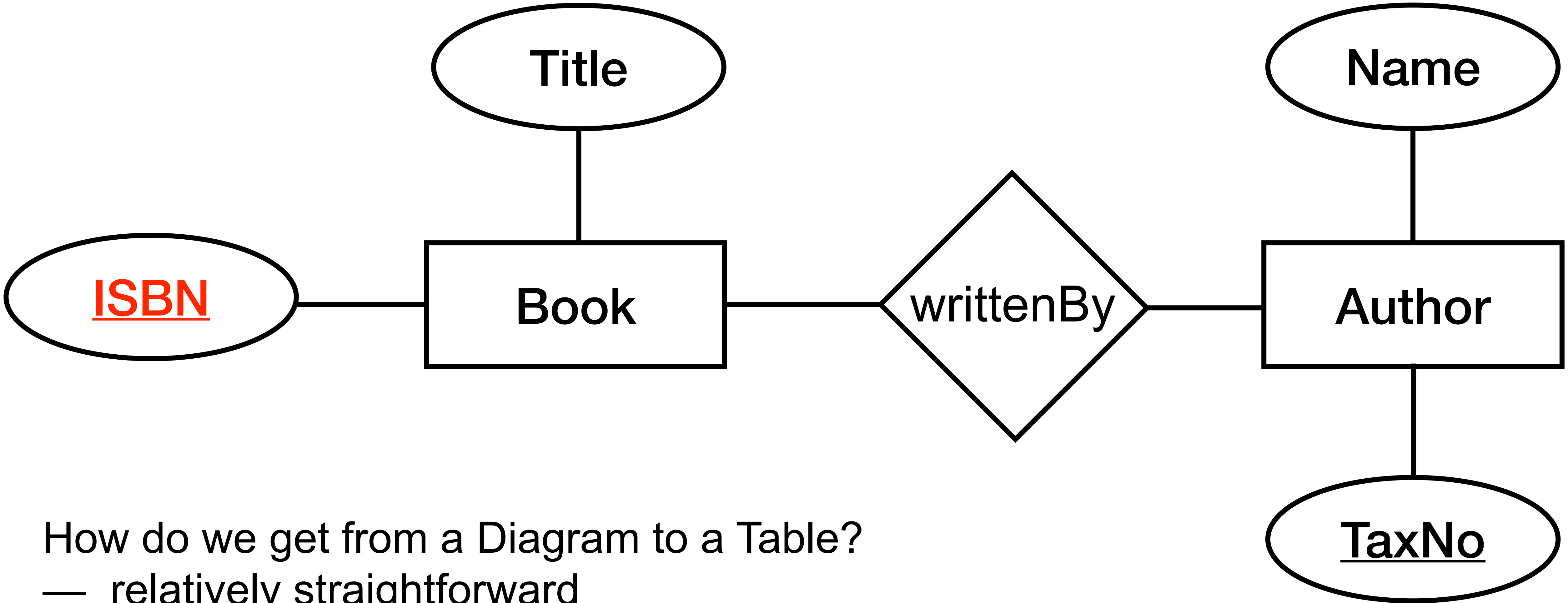
Decision support queries against such a database may include finding *unpaid orders*:

```
SELECT O.order_id FROM Orders O
WHERE  O.order_id NOT IN
     ( SELECT order_id FROM Payments )
```

or finding customers *who have not placed an order*:

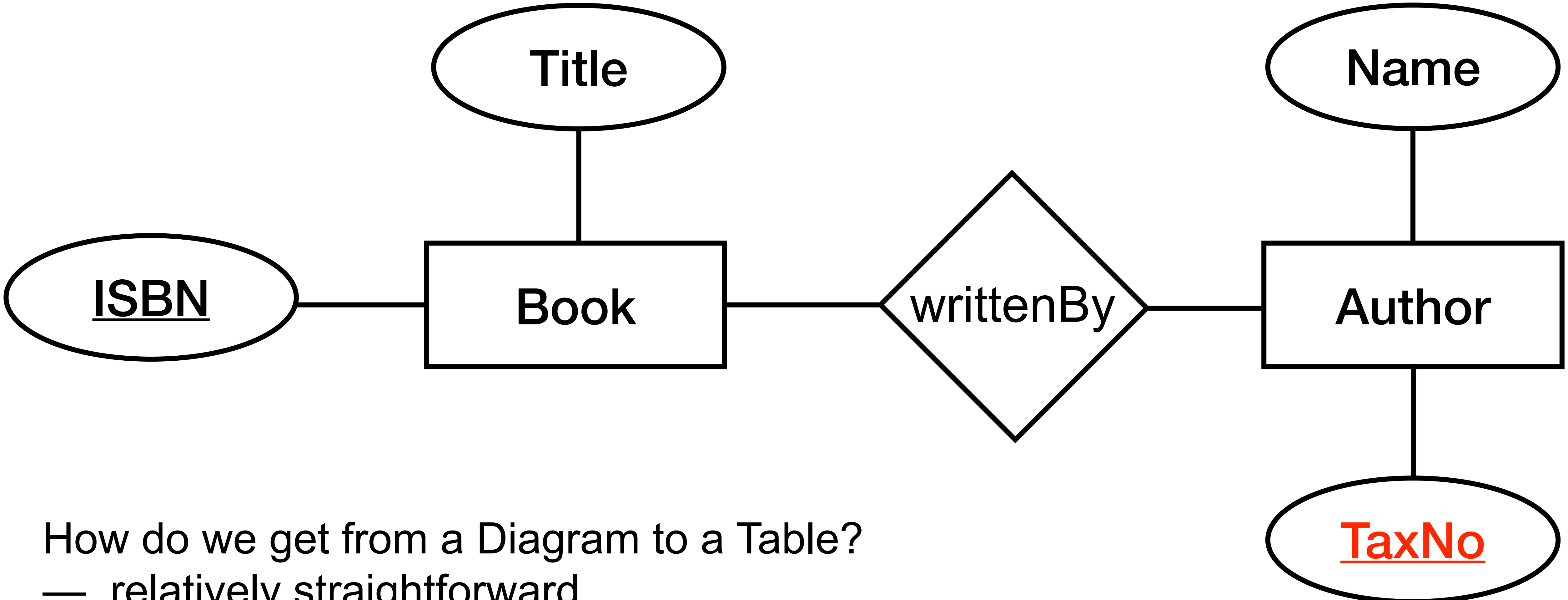
```
SELECT C.cust_id FROM Customers C
WHERE  NOT EXISTS
     ( SELECT * FROM Orders O, Payments P
       WHERE  C.cust_id = P.cust_id
```

Paolo Guagliardo, Leonid Libkin:
Correctness of SQL Queries on Databases with Nulls. SIGMOD Record 46(3): 5-16 (2017)



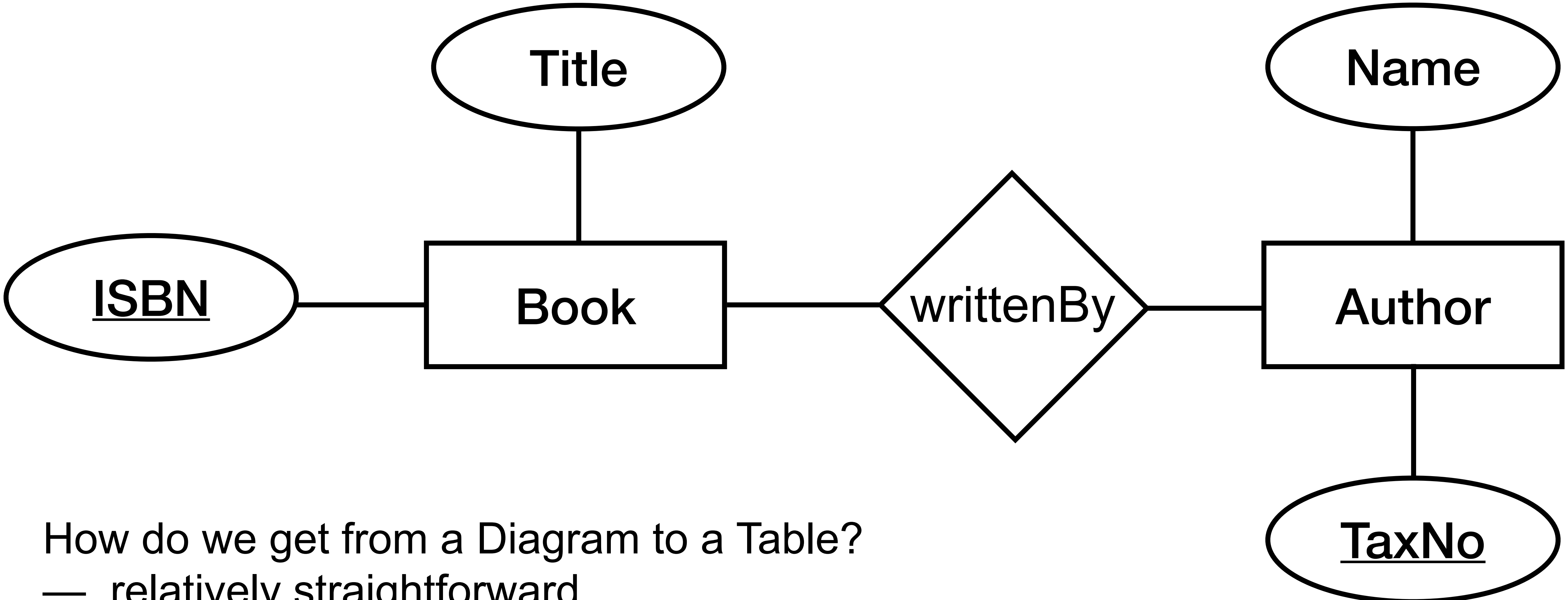
How do we get from a Diagram to a Table?
— relatively straightforward
(see, e.g., https://beginnersbook.com/2015/04/e-r-model-in-dbms/)

```
CREATE Table Book(ISBN VARCHAR(40) PRIMARY KEY,
                  Title VARCHAR(100));
```



How do we get from a Diagram to a Table?
— relatively straightforward
(see, e.g., https://beginnersbook.com/2015/04/e-r-model-in-dbms/)

```
CREATE Table Book(ISBN VARCHAR(40) PRIMARY KEY,
                  Title VARCHAR(100));
CREATE Table Author(Name VARCHAR,
                    TaxNo VARCHAR PRIMARY KEY);
```



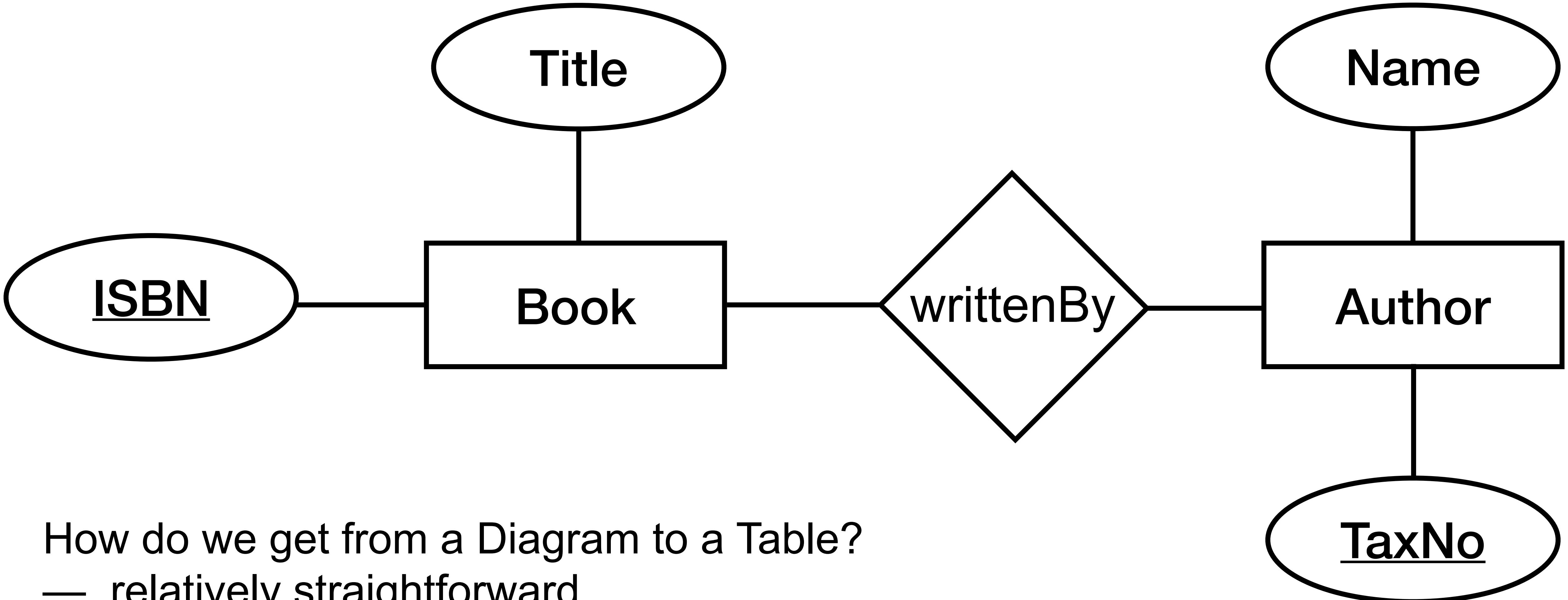
How do we get from a Diagram to a Table?

— relatively straightforward

(see, e.g., https://beginnersbook.com/2015/04/e-r-model-in-dbms/)

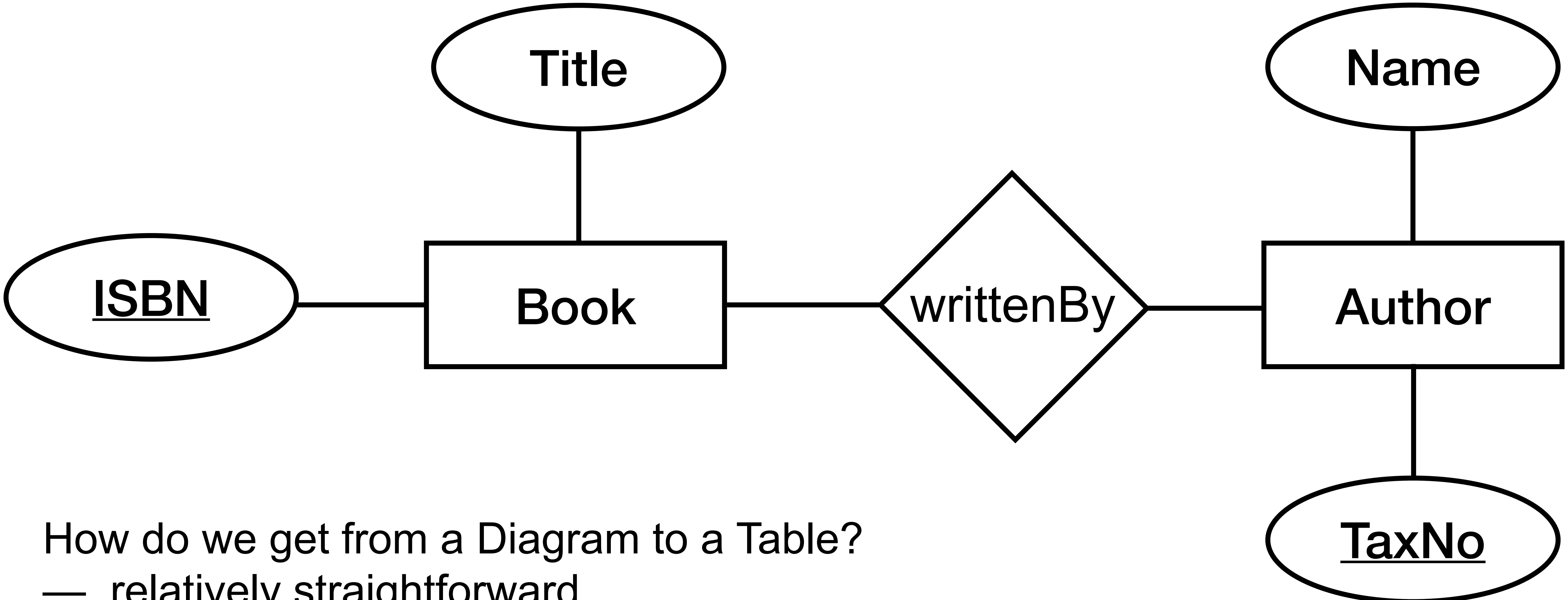
```
CREATE Table Book(ISBN VARCHAR(40) PRIMARY KEY,
                  Title VARCHAR(100));
CREATE Table Author(Name VARCHAR,
                    TaxNo VARCHAR PRIMARY KEY);
```

— how do you create the writtenBy Table??



How do we get from a Diagram to a Table?
— relatively straightforward
(see, e.g., https://beginnersbook.com/2015/04/e-r-model-in-dbms/)

```
CREATE Table Book(ISBN VARCHAR(40) PRIMARY KEY,
                  Title VARCHAR(100));
CREATE Table Author(Name VARCHAR,
                    TaxNo VARCHAR PRIMARY KEY);
CREATE Table writtenBy(ISBN VARCHAR(40) REFERENCES Book,
                       TaxNo VARCHAR REFERENCES Author,
                       PRIMARY KEY (ISBN, TaxNo));
```



How do we get from a Diagram to a Table?
— relatively straightforward
(see, e.g., https://beginnersbook.com/2015/04/e-r-model-in-dbms/)

```
CREATE Table Book(ISBN VARCHAR(40) PRIMARY KEY,
                  Title VARCHAR(100));
CREATE Table Author(Name VARCHAR,
                    TaxNo VARCHAR PRIMARY KEY);
CREATE Table writtenBy(ISBN VARCHAR(40) REFERENCES Book,
                       TaxNo VARCHAR REFERENCES Author,
                       PRIMARY KEY (ISBN, TaxNo));
```

every table should have a primary key!
(this avoids duplicates)

```
CREATE Table Book(ISBN VARCHAR(40) PRIMARY KEY,
                  Title VARCHAR(100));
CREATE Table Author(Name VARCHAR,
                    TaxNo VARCHAR PRIMARY KEY);
CREATE Table writtenBy(ISBN VARCHAR(40) REFERENCES Book,
                       TaxNo VARCHAR REFERENCES Author,
                       PRIMARY KEY (ISBN, TaxNo));
```

`REFERENCES Book`

is a shorthand for ← the primary key of Book is used

`REFERENCES Book(ISBN);`

```
CREATE Table Book(ISBN VARCHAR(40) PRIMARY KEY,
                  Title VARCHAR(100));
CREATE Table Author(Name VARCHAR,
                    TaxNo VARCHAR PRIMARY KEY);
CREATE Table writtenBy(ISBN VARCHAR(40) REFERENCES Book,
                       TaxNo VARCHAR REFERENCES Author,
                       PRIMARY KEY (ISBN, TaxNo));
```

`REFERENCES Book`
is a shorthand for ← the primary key of Book is used
`REFERENCES Book(ISBN);`

other alternative:

```
CREATE Table writtenBy(ISBN VARCHAR(40),
                       TaxNo VARCHAR,
                       PRIMARY KEY (ISBN, TaxNo),
               FOREIGN KEY ISBN REFERENCES Book(ISBN),
               FOREIGN KEY TaxNo REFERENCES Author(TaxNo),
);
```

```
CREATE Table Book(ISBN VARCHAR(40) PRIMARY KEY,
                  Title VARCHAR(100));
CREATE Table Author(Name VARCHAR,
                    TaxNo VARCHAR PRIMARY KEY);
CREATE Table writtenBy(ISBN VARCHAR(40) REFERENCES Book,
                       TaxNo VARCHAR REFERENCES Author,
                       PRIMARY KEY (ISBN, TaxNo));
```

— **before** a pair (isbn, taxno) can be inserted into the writtenBy table, there must be a tuple (isbn, title) in Book and there must be a tuple (name, taxno) in Author.

```sql
CREATE Table Book(ISBN VARCHAR(40) PRIMARY KEY,
                  Title VARCHAR(100));
CREATE Table Author(Name VARCHAR,
                    TaxNo VARCHAR PRIMARY KEY);
CREATE Table writtenBy(ISBN VARCHAR(40) REFERENCES Book,
                       TaxNo VARCHAR REFERENCES Author,
                       PRIMARY KEY (ISBN, TaxNo));
```

— **before** a pair (isbn, taxno) can be inserted into the writtenBy table, there must be a tuple (isbn, title) in Book and there must be a tuple (name, taxno) in Author.

— **before** a pair (isbn,title) can be deleted from Book, any tuple of the form (isbn, xxx) must be deleted from writtenBy.

smaneth — screen ▸ emacs — 83×23

```
dblp=# select * from book;
 isbn | title
------+-------
(0 rows)

dblp=# select * from author;
 name | taxno
------+-------
(0 rows)

dblp=# select * from authoredby;
 isbn | taxno
------+-------
(0 rows)

dblp=# insert into authoredby VALUES ('1234','676');
ERROR:  insert or update on table "authoredby" violates foreign key constraint "aut
horedby_isbn_fkey"
DETAIL:  Key (isbn)=(1234) is not present in table "book".
dblp=#
```

smaneth — screen ▸ emacs — 83×23

```
dblp=# select * from author;
 name | taxno
------+-------
(0 rows)

dblp=# select * from authoredby;
 isbn | taxno
------+-------
(0 rows)

dblp=# insert into authoredby VALUES ('1234','676');
ERROR:  insert or update on table "authoredby" violates foreign key constraint "aut
horedby_isbn_fkey"
DETAIL:  Key (isbn)=(1234) is not present in table "book".
dblp=# INSERT INTO book VALUES ('1234', 'The Book');
INSERT 0 1
dblp=# INSERT INTO author VALUES ('Joe Doe','676');
INSERT 0 1
dblp=# insert into authoredby VALUES ('1234','676');
INSERT 0 1
dblp=#
dblp=#
dblp=#
```

smaneth — screen ▸ bash — 73×23

```
dblp=# select * from author;
  name    | taxno
---------+-------
 Joe Doe | 676
(1 row)

dblp=# select * from book;
 isbn |  title
------+----------
 1234 | The Book
(1 row)

dblp=# select * from authoredby;
 isbn | taxno
------+-------
 1234 | 676
(1 row)

dblp=# delete from author where taxno='676';
ERROR:  update or delete on table "author" violates foreign key constrain
t "authoredby_taxno_fkey" on table "authoredby"
DETAIL:  Key (taxno)=(676) is still referenced from table "authoredby".
dblp=#
```

ON DELETE CASCADE

```
dblp=# create table authoredBy(ISBN VARCHAR(40),
                       TaxNo VARCHAR,
                       PRIMARY KEY (ISBN, TaxNo), FOREIGN KEY (ISBN) REFERENCES Book(ISB
N), foreign key (TaxNo) references author(TaxNo) ON DELETE CASCADE);
CREATE TABLE
dblp=# select * from author;
  name   | taxno
---------+-------
 Joe Doe | 676
(1 row)

dblp=# select * from authoredby;
 isbn | taxno
------+-------
(0 rows)

dblp=# insert into authoredby values ('1234','676');
INSERT 0 1
dblp=# delete from author where taxno='676';
DELETE 1
dblp=# select * from authoredby;
 isbn | taxno
------+-------
(0 rows)
```

(1234,676)-tuple in authoredBy is autmatically deleted (**CASCADE**)

General Thumb Rules

1.) in every table choose a PRIMARY KEY
(in the ‘worst case’ simply consisting of all attributes)

General Thumb Rules

1.) in every table choose a PRIMARY KEY
(in the ‘worst case’ simply consisting of all attributes)

2.) user FOREIGN KEYs where appropriate

General Thumb Rules

1.) in every table choose a PRIMARY KEY
(in the 'worst case' simply consisting of all attributes)

2.) user FOREIGN KEYs where appropriate

3.) for every attribute, forbid nulls by appending NOT NULL

```
CREATE Table Book(ISBN VARCHAR(40) PRIMARY KEY,
                  Title VARCHAR(100) NOT NULL);
CREATE Table Author(Name VARCHAR NOT NULL,
                    TaxNo VARCHAR PRIMARY KEY);
CREATE Table writtenBy(ISBN VARCHAR(40) REFERENCES Book,
                       TaxNo VARCHAR REFERENCES Author,
                       PRIMARY KEY (ISBN, TaxNo));
```

— attributes of PRIMARY KEYS may not contain NULLs by default

Datatypes

Table Creation

To create a new table, use the **CREATE TABLE** statement:

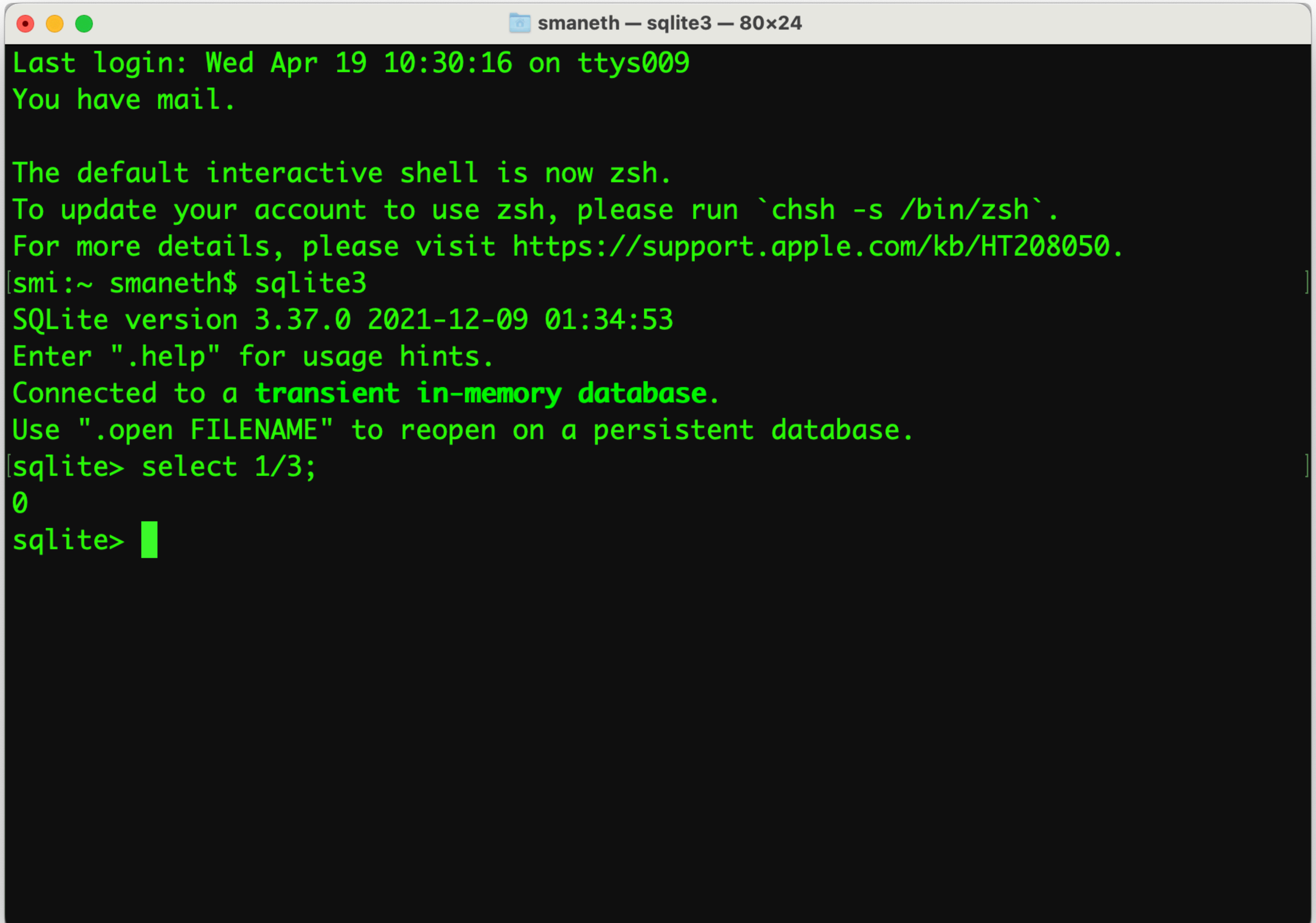
```
CREATE TABLE Ingredients ( IngrID   INTEGER NOT NULL,
                           Name     CHAR(30),
                           Alcohol  DECIMAL(3,1),
                           Flavor   CHAR(20) )
```

Data types (somewhat system-dependent):

- INTEGER, SMALLINT, BIGINT
- DECIMAL(*m*,*n*): *m* digits total, *n* of which are decimals
- CHAR(*n*): fixed-length strings
- VARCHAR(*n*): variable-length strings (up to length *n*)
- DATE, TIME, DATETIME, etc.

Warning

By default there is **rounding**



Warning

By default there is **rounding**

smaneth — sqlite3 — 80×24

```
Last login: Wed Apr 19 10:30:16 on ttys009
You have mail.

The default interactive shell is now zsh.
To update your account to use zsh, please run `chsh -s /bin/zsh`.
For more details, please visit https://support.apple.com/kb/HT208050.
smi:~ smaneth$ sqlite3
SQLite version 3.37.0 2021-12-09 01:34:53
Enter ".help" for usage hints.
Connected to a transient in-memory database.
Use ".open FILENAME" to reopen on a persistent database.
sqlite> select 1/3;
0
sqlite> select 1.0/3;
0.333333333333333
sqlite> select 1/3.0;
0.333333333333333
sqlite>
```

Creating Tables


```
mysql> CREATE TABLE test (a float(3,3), b float(4,2), c float(5,1));
mysql> INSERT INTO test VALUES (100.999, 100.999, 100.999);
Query OK, 1 row affected, 2 warnings (0.01 sec)
mysql> SHOW WARNINGS;
+---------+------+---------------------------------------+
| Level   | Code | Message                               |
+---------+------+---------------------------------------+
| Warning | 1264 | Out of range value for column 'a' at row 1 |
| Warning | 1264 | Out of range value for column 'b' at row 1 |
+---------+------+---------------------------------------+
mysql> INSERT INTO test VALUES (2/3,2/3,2/3);
Query OK, 1 row affected (0.00 sec)

mysql> SELECT * FROM test;
+-------+-------+-------+
| a     | b     | c     |
+-------+-------+-------+
| 0.999 | 99.99 | 101.0 |
| 0.667 |  0.67 |   0.7 |
+-------+-------+-------+
2 rows in set (0.00 sec)
```

Types cont'd

1. **date** - date type

1 warning

```
> CREATE TABLE T1 (col1 date PRIMARY KEY);
> INSERT INTO T1 VALUES ("2005-12-24");
> INSERT INTO T1 VALUES ("01-01-01");
> INSERT INTO T1 VALUES ("05-05-2010");
> SELECT * FROM T1;
+------------+
| col1       |
+------------+
| 2001-01-01 |
| 2005-12-24 |
| 0000-00-00 |
+------------+
```

MySQL

```
> CREATE TABLE T1 (col1 date PRIMARY KEY);
> INSERT INTO T1 VALUES ("01-01-2001");
> INSERT INTO T1 VALUES ("2001-01-01");
> INSERT INTO T1 VALUES ("Feb-2005");
> INSERT INTO T1 VALUES ("2005");
> SELECT * FROM T1;
2005
01-01-2001
2001-01-01
Feb-2005
```

sqlite3

Types cont'd

1. **timestamp**

MySQL

```
> CREATE TABLE T1 (col1 timestamp PRIMARY KEY);
> INSERT INTO T1 VALUES (“2001-12-24 11:18:00”);
> INSERT INTO T1 VALUES (“2001-12-24 23:18:00”);
> SELECT * FROM T1;
+---------------------+
| col1                |
+---------------------+
| 2001-12-24 11:18:00 |
| 2001-12-24 23:18:00 |
+---------------------+
```

date / time / timestamp

→ can be compared for equality and less than (<)

→ if date1 < date2, then date 1 is earlier than date2

Types cont'd

1. **timestamp**

MySQL →

```
> CREATE TABLE T1 (col1 timestamp PRIMARY KEY);
> INSERT INTO T1 VALUES (“2001-12-24 11:18:00”);
> INSERT INTO T1 VALUES (“2001-12-24 23:18:00”);
> SELECT * FROM T1;
+---------------------+
| col1                |
+---------------------+
| 2001-12-24 11:18:00 |
| 2001-12-24 23:18:00 |
+---------------------+
```

```
mysql> SELECT CURTIME(), CURTIME()+0, CURTIME(3)+0;
+-----------+-------------+---------------+
| CURTIME() | CURTIME()+0 | CURTIME(3)+0  |
+-----------+-------------+---------------+
| 09:21:56  |       92156 |     92156.311 |
+-----------+-------------+---------------+
1 row in set (0.00 sec)

mysql> SELECT NOW(), NOW()+0, NOW(3)+0;
+---------------------+----------------+--------------------+
| NOW()               | NOW()+0        | NOW(3)+0           |
+---------------------+----------------+--------------------+
| 2018-07-18 09:22:15 | 20180718092215 | 20180718092215.234 |
+---------------------+----------------+--------------------+
1 row in set (0.00 sec)
```

Tables

→ DELETE FROM T1; - delete all rows from table T1

→ DELETE FROM T1 where c3=1; - delete rows with c3-value equals 1

→ DROP TABLE T1; - remove table T1

→ ALTER TABLE T1 ADD COLUMN col1 int; - adds a column to table T1

→ ALTER TABLE T1 DROP COLUMN col1; - removes a column from table T1

→ DESCRIBE T1; - lists the fields and types of table t1 (**MySQL**, not SQL!)
(in **sqlite3** this is done via ".schema t1" or "PRAGMA table_info(t1)")

→ SHOW tables; - lists tables of your database (**MySQL**, not SQL!)
(in **sqlite3** this is done via ".tables")

— \d lists all tables in PostgreSQL
\d tablename shows schema of the table

Tables

→ default values for some attributes:

```
CREATE TABLE T1 (  <attribute> <type> DEFAULT <value>  )
```

```
> CREATE TABLE T1 (col1 int DEFAULT 0, col2 int);
> INSERT INTO T1 VALUES (1,2);
> INSERT INTO T1 (col2) VALUES (5);
> SELECT * FROM T1;
+------+------+
| col1 | col2 |
+------+------+
|    1 |    2 |
|    0 |    5 |
+------+------+
```

→ ALTER TABLE Movies ADD COLUMN length int DEFAULT 0;

Next week:

Monday 01.05.2023: keine Vorlesung (Tag der Arbeit)
Mittwoch 03.05.2023: keine Übung

Donnerstag 04.05.2023: 10:15—11:45 und
14:15—15:45 “Fragestunde” findet statt!

Next week:

Monday 01.05.2023: keine Vorlesung (Tag der Arbeit)
Mittwoch 03.05.2023: keine Übung

Donnerstag 04.05.2023: 10:15—11:45 und
14:15—15:45 “**Fragestunde**” findet statt!

Topics

— how to **run** `sqlite3`
— how to **create tables** in `sqlite3`
— how to **load a CSV-file** into a table in `sqlite3`

Kurs

Datenbankgrund

und Modell g

End of this lecture

Sebast rsität Bremen

bremen.de

ersemester 2023

24.4.2023

rlesung 2: ER-Diagrams & Creating Tables