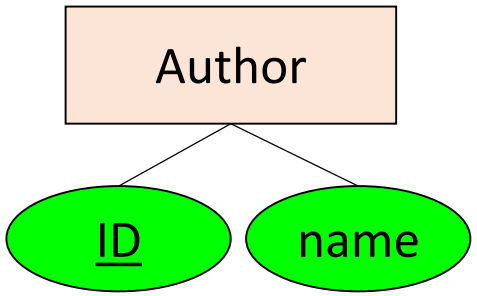
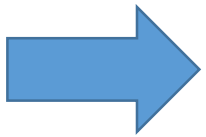
COMP 430
Intro. to Database Systems

ER Implementation as Tables

Slides use ideas from Chris Ré.

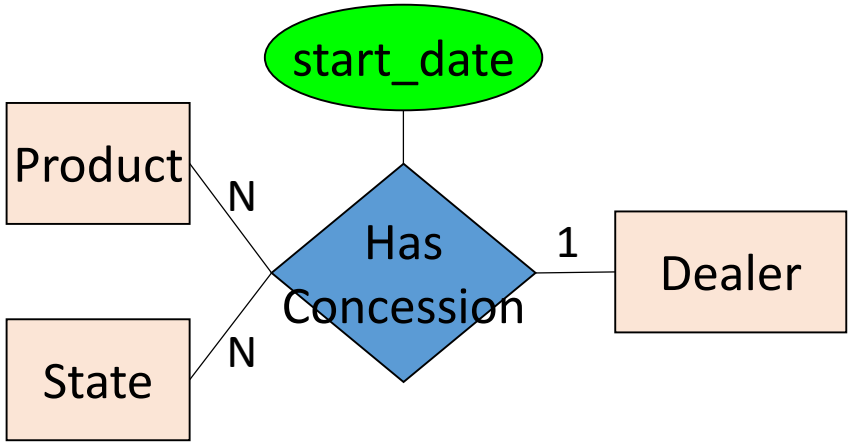
Overview – the general cases

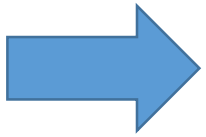




Author

ID	name
…	…





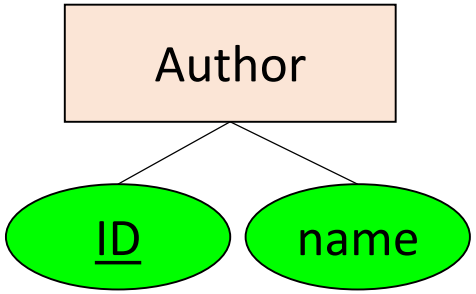
Primary keys =
keys on “many” sides

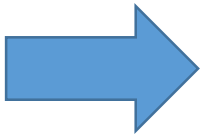
Concession

product_id	state_id	dealer_id	start_date
…	…	…	…

Tables

Entity sets become tables

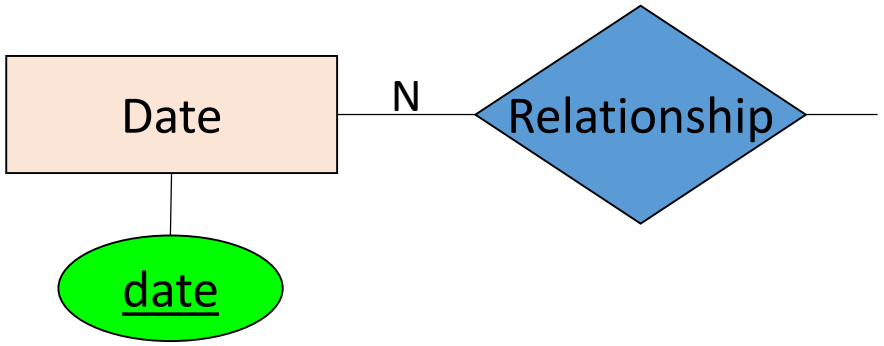




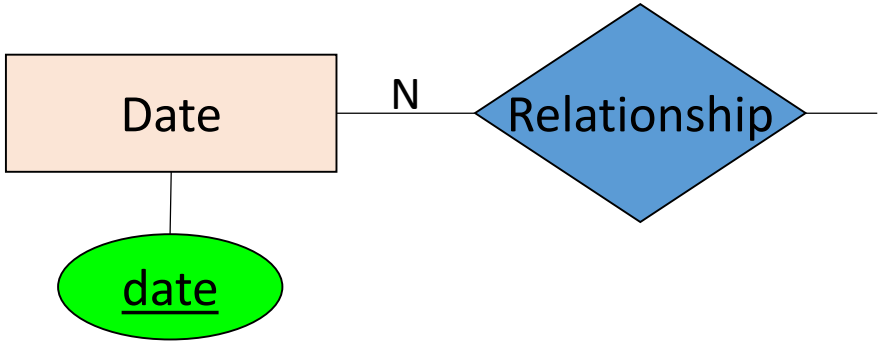
Author

ID	name
…	…

Except when infinite



Add key field to Relationship's table.

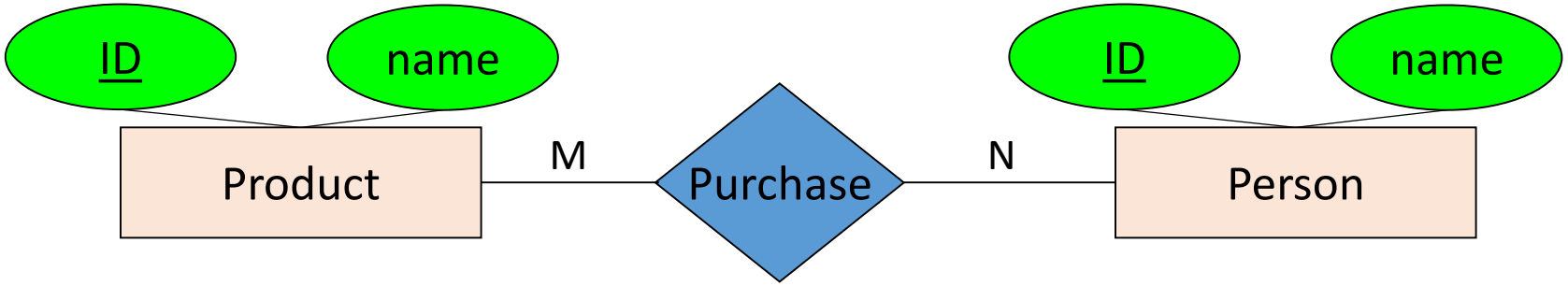


Add non-key field to Relationship's table.

More standard: Not allowing such to be represented as entity sets.

Arity & cardinality: Common cases

Many-to-many



Product

ID	name
…	…

Purchase

prod_ID	person_ID
…	…

Person

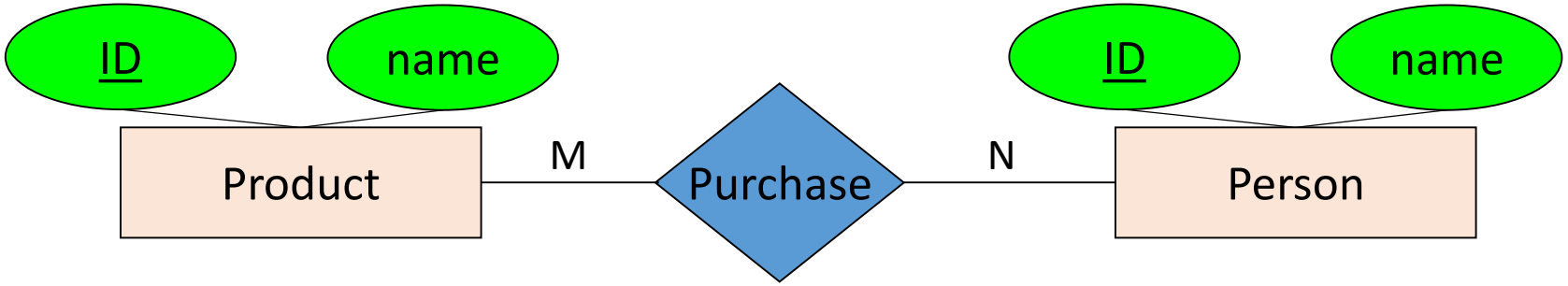
ID	name
…	…

```
CREATE TABLE Product (
    …
    PRIMARY KEY (ID)
);
```

```
CREATE TABLE Purchase (
    …
    PRIMARY KEY (prod_ID, person_ID),
    FOREIGN KEY (prod_ID) REFERENCES Product (ID),
    FOREIGN KEY (person_ID) REFERENCES Person (ID)
);
```

```
CREATE TABLE Person (
    …
    PRIMARY KEY (ID)
);
```

Many-to-many



Purchase

prod_ID	person_ID
…	…

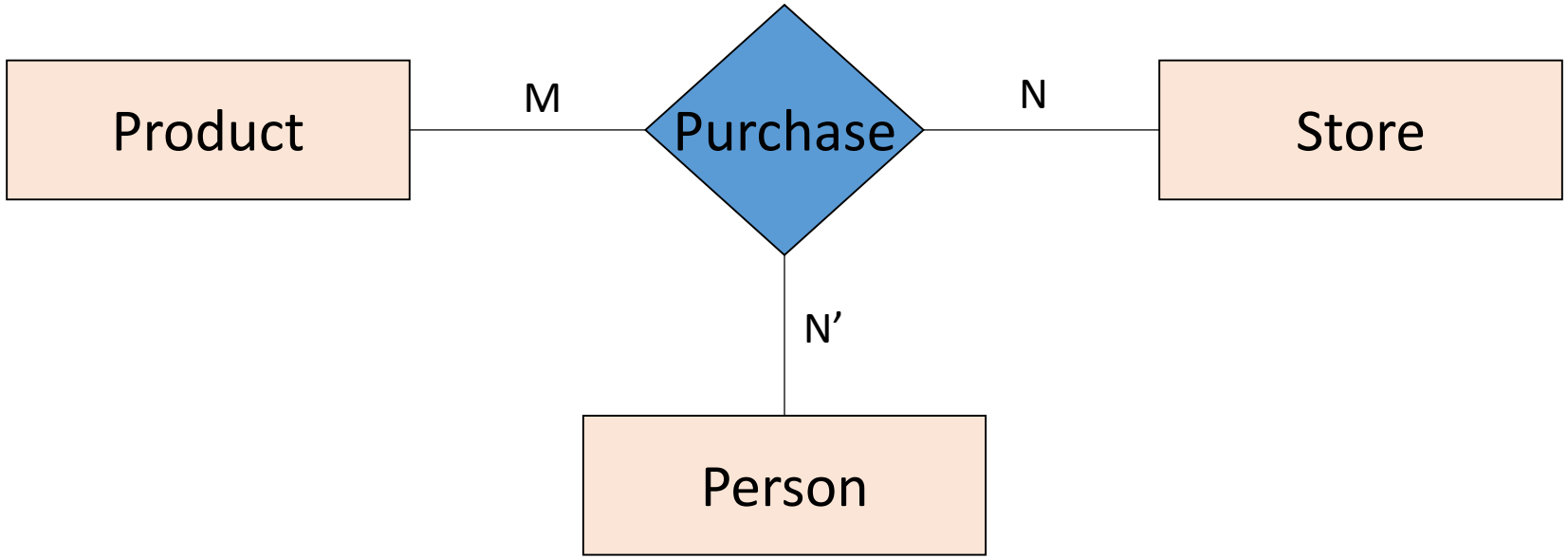
No. E.g., a **Product** never purchased simply never appears in **Purchase** table.

Primary key fields also NOT NULL.

Doesn't that force total participation?

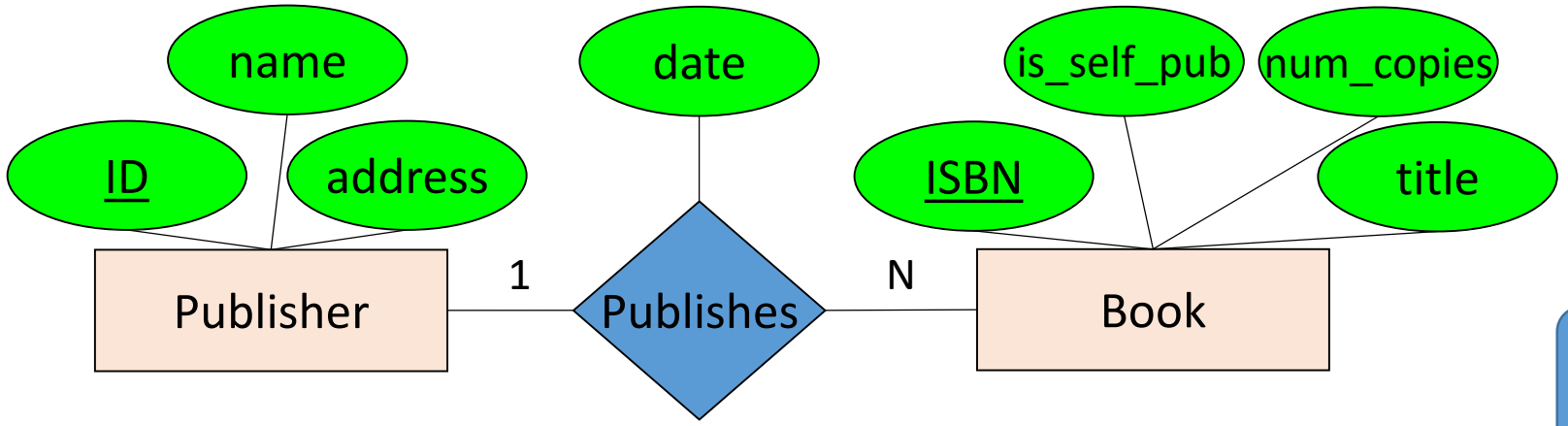
```
CREATE TABLE Purchase (
    …
    PRIMARY KEY (prod_ID, person_ID),
    FOREIGN KEY (prod_ID) REFERENCES Product (ID),
    FOREIGN KEY (person_ID) REFERENCES Person (ID)
);
```

N-ary many-many



Junction table’s primary key = combination of primary keys of all n tables.

One-to-many / many-to-one



Publishes table optimized away.

Publisher

ID	name	address
…	…	…

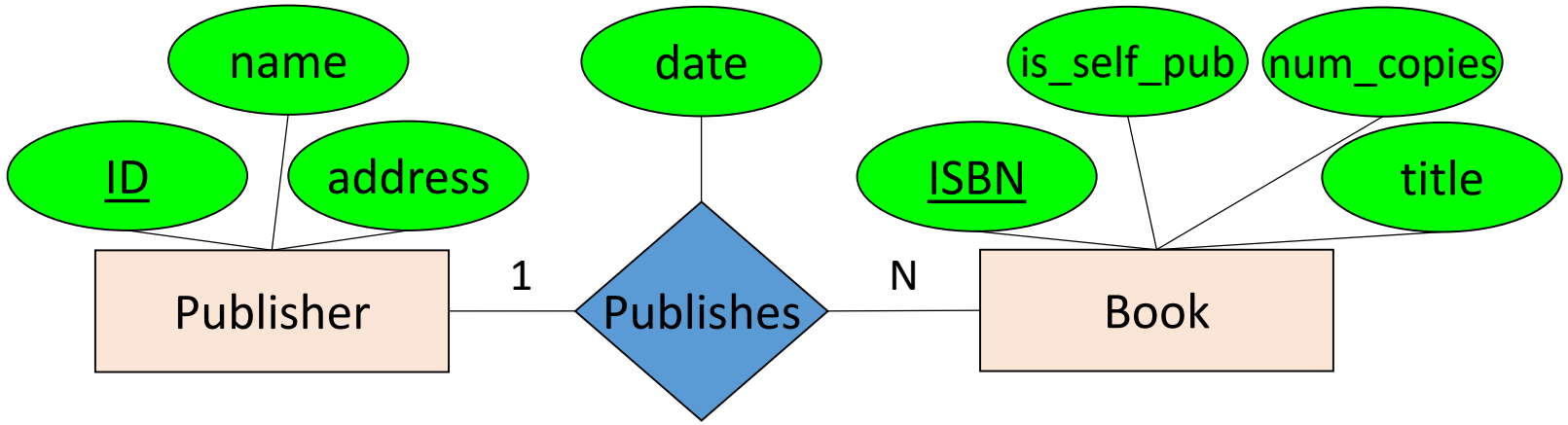
Book

ISBN	title	num_copies	is_self_pub	pub_ID	pub_date
…	…	…	…	…	

```
CREATE TABLE Publisher (
    …
    PRIMARY KEY (ID)
);
```

```
CREATE TABLE Book (
    …
    PRIMARY KEY (ISBN),
    FOREIGN KEY (pub_ID) REFERENCES Publisher (ID)
);
```

One-to-many / many-to-one



Publisher

ID	name	address
…	…	…

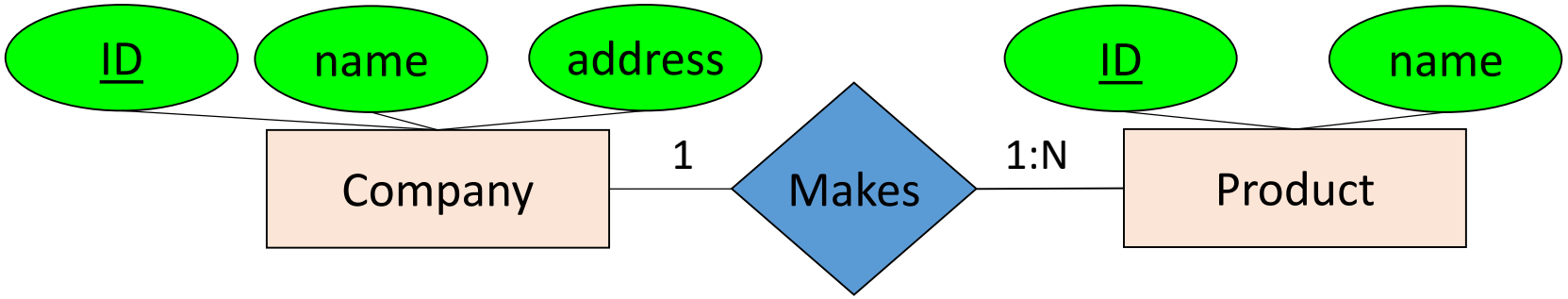
Book

ISBN	title	num_copies	is_self_pub	pub_ID	pub_date
…	…	…	…	…	

Way to remember where the additional field is placed:

- **Book** has one publisher, so it can be a field.
- **Publisher** has many books, so it can't be a field.

Minimum cardinality



Company

ID	name	address
…	…	…

Product

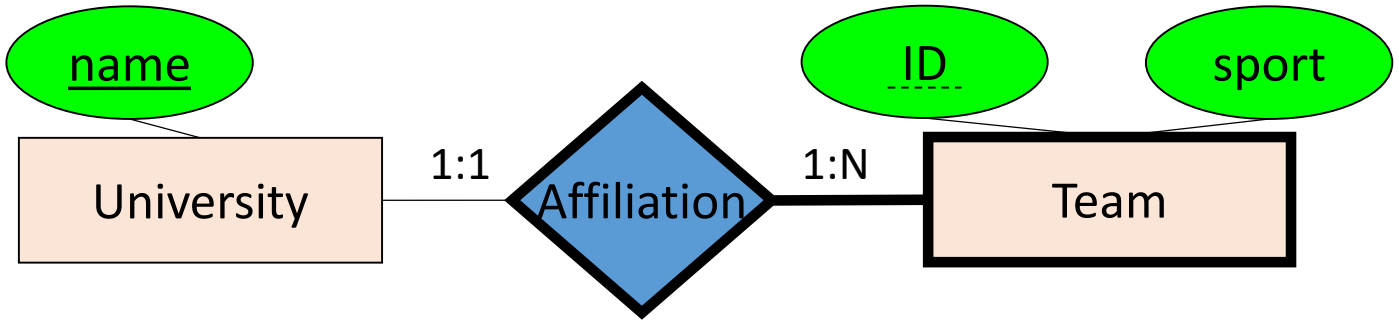
ID	name	co_ID
…	…	…

Foreign key must have a value.

```
CREATE TABLE Company (
    …
    PRIMARY KEY (ID)
);
```

```
CREATE TABLE Product (
    …
    co_ID  VARCHAR(50) NOT NULL,
    PRIMARY KEY (ID),
    FOREIGN KEY (co_ID) REFERENCES Company (ID)
);
```

Weak entity



University

name
…

Team

univ_name	ID	sport
…	…	…

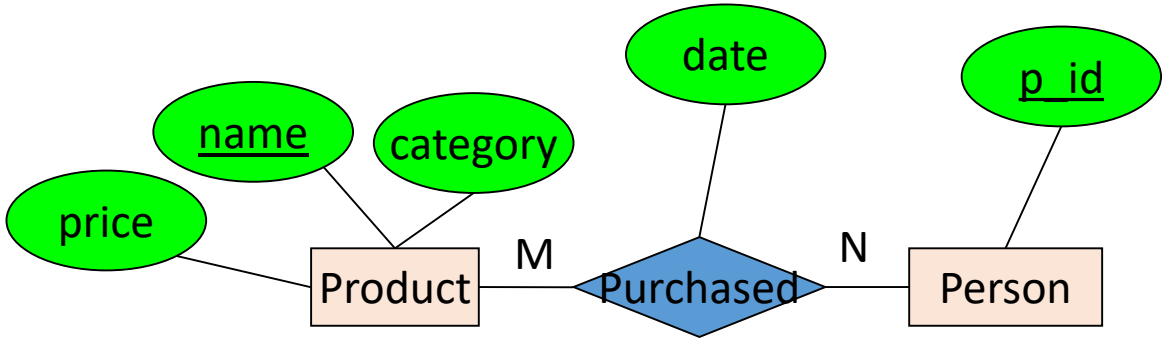
```
CREATE TABLE University (
    …
    PRIMARY KEY (name)
);
```

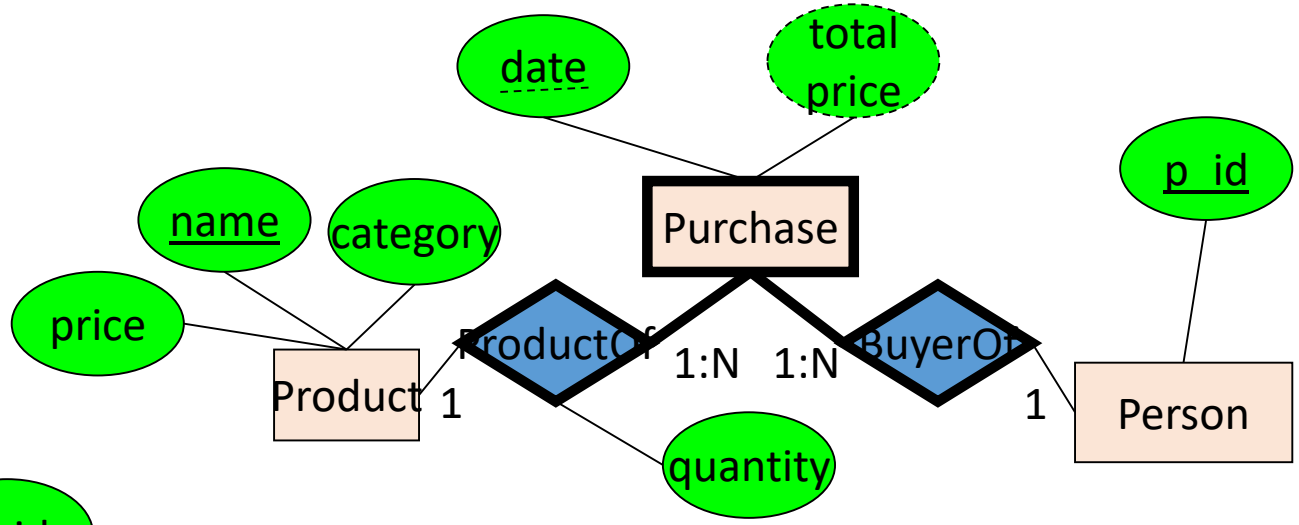
```
CREATE TABLE Team (
    …
    PRIMARY KEY (univ_name, ID),
    FOREIGN KEY (univ_name) REFERENCES University (name)
);
```

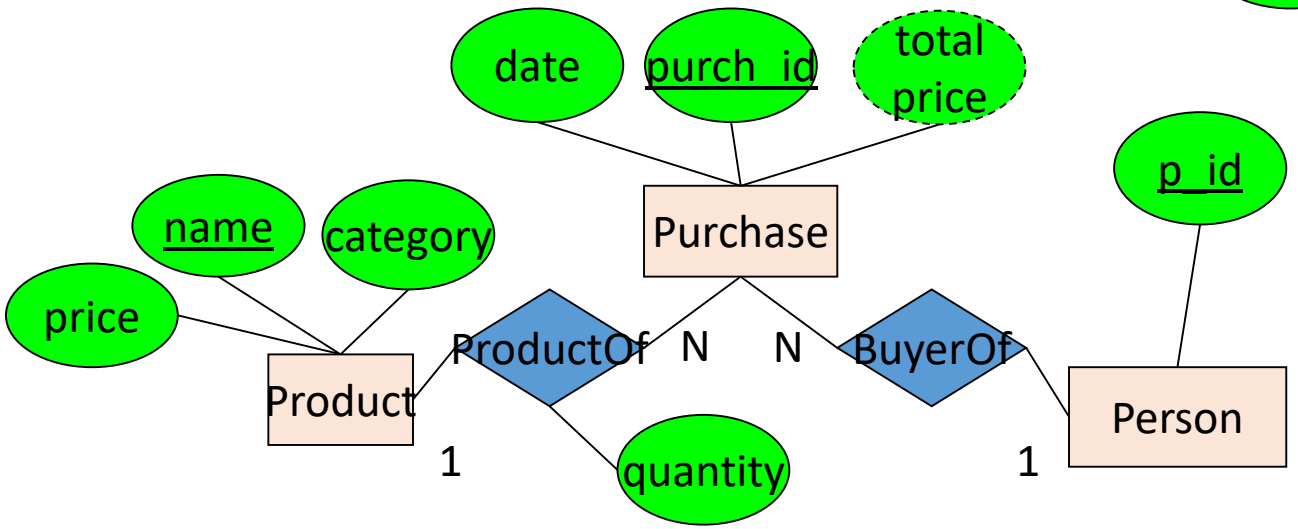
Primary key = keys of both entities. Must have value.

Foreign key = key of related table. Must have value.

Activity: Implement three similar ideas

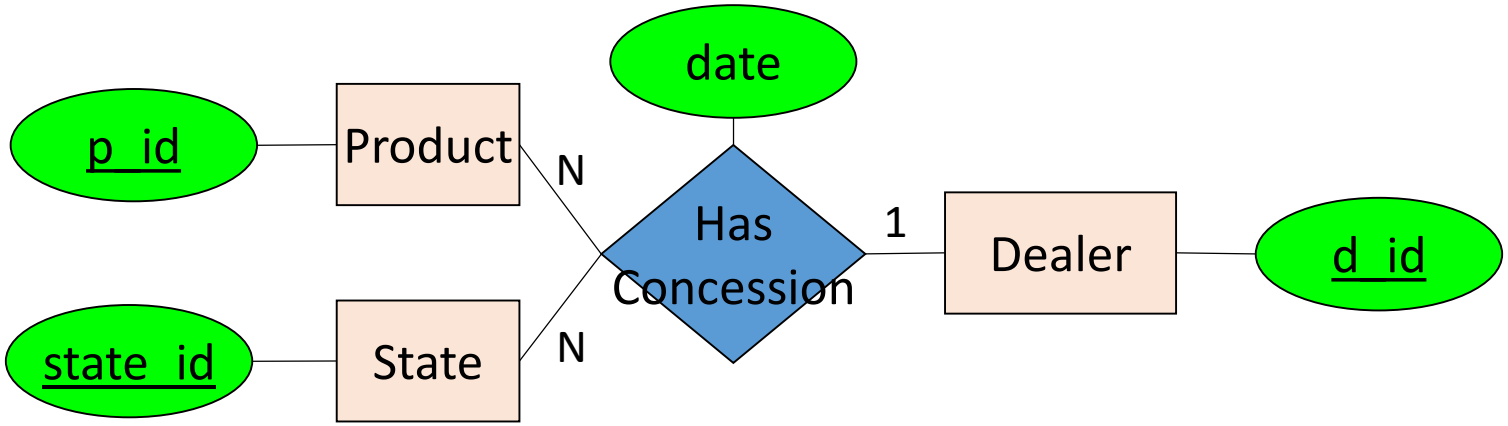






Each results in a **Purchase** junction table. Differences?

One-to-many with total participation

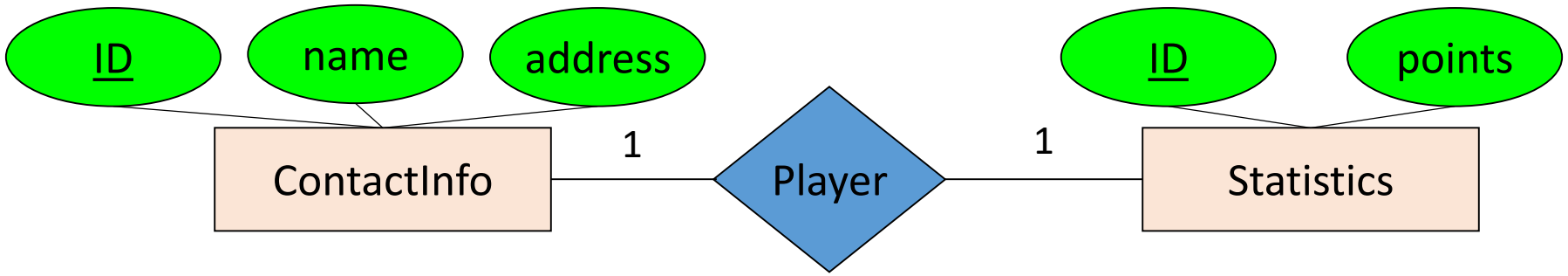


Concession

p_id	state_id	d_id	date
…	…	…	…

```
CREATE TABLE Concession (
    …
    PRIMARY KEY (p_id, state_id),
    FOREIGN KEY (p_id) REFERENCES Product (p_id),
    FOREIGN KEY (state_id) REFERENCES State (state_id),
    FOREIGN KEY (d_id) REFERENCES Dealer (d_id)
);
```

One-to-one with same key

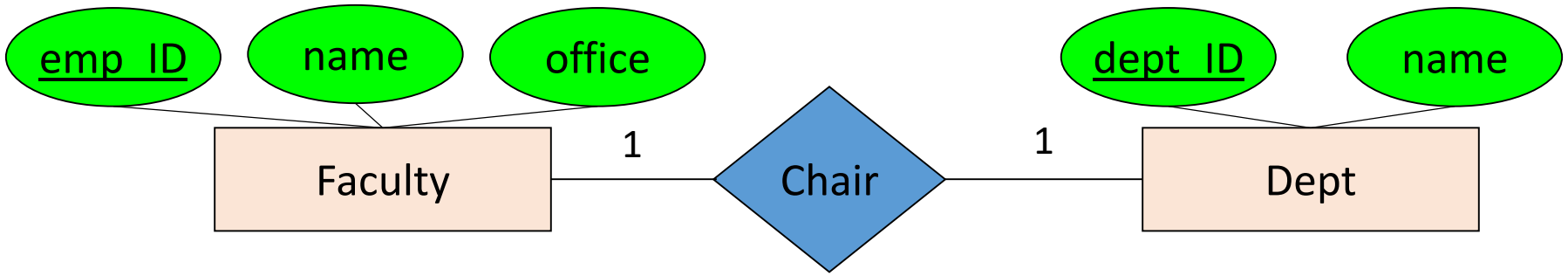


Usually indicates a poor design. Combine into one entity set.

Player

ID	name	address	points
…	…	…	…

One-to-one with different keys



Sometimes indicates a poor design.

Faculty

emp_ID	name	office	dept_id
…	…	…	…

Dept

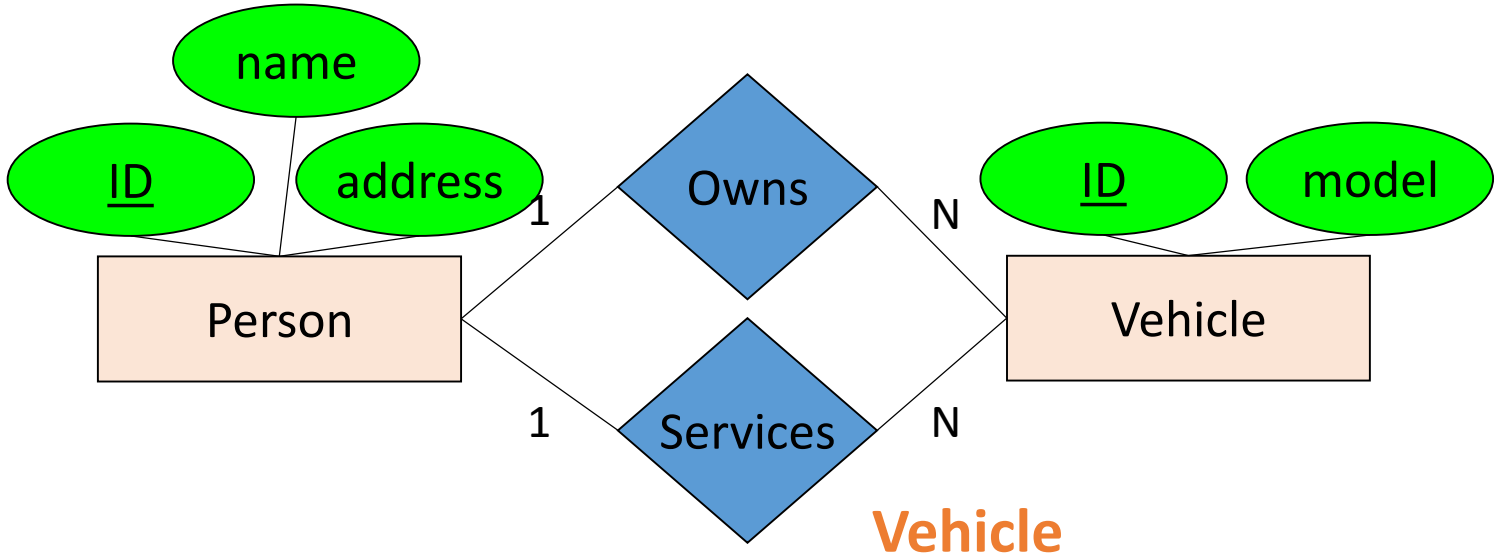
dept_ID	name	chair_id
…	…	…

Same optimization as for one-to-many.

```
CREATE TABLE Faculty (
    …
    PRIMARY KEY (emp_ID),
    FOREIGN KEY (dept_ID) REFERENCES Dept (dept_ID)
);
```

```
CREATE TABLE Department (
    …
    PRIMARY KEY (dept_ID),
    FOREIGN KEY (chair_ID) REFERENCES Faculty (emp_ID)
);
```

Parallel one-to-many/one relationships



Same idea as before, for each relationship.

Person

ID	name	address
…	…	…

```
CREATE TABLE Person (
    …
    PRIMARY KEY (ID)
);
```

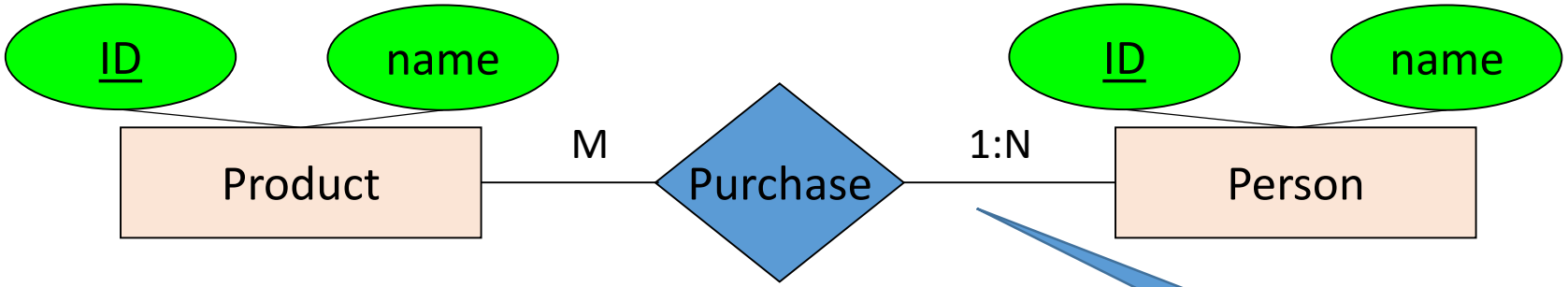
Vehicle

ID	model	owner_ID	mechanic_ID
…	…	…	

```
CREATE TABLE Vehicle (
    …
    PRIMARY KEY (ID),
    FOREIGN KEY (owner_ID) REFERENCES Person (owner_ID),
    FOREIGN KEY (mechanic_ID) REFERENCES Person (owner_ID)
);
```

Cardinality:
Some less common cases

Many-to-many with minimum cardinality



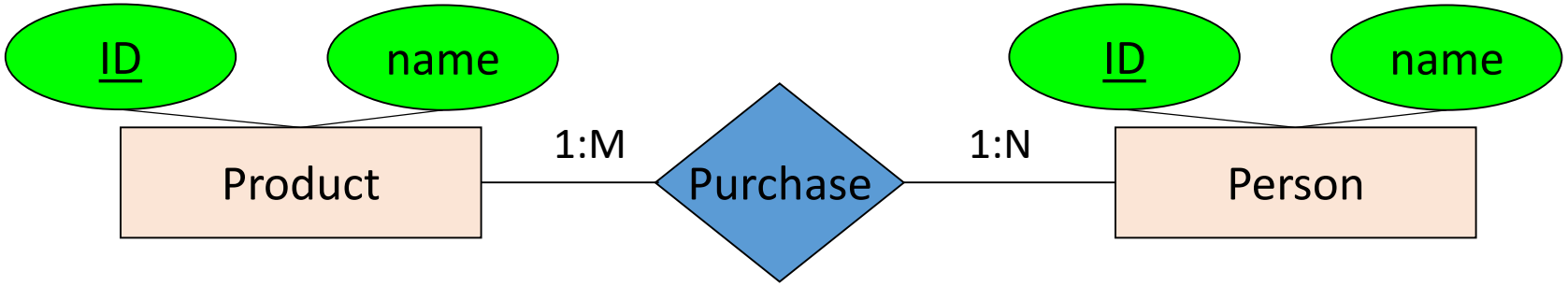
Purchase

prod_ID	person_ID
…	…

Can enforce with a CHECK constraint. See later.

```
CREATE TABLE Purchase (
    …
    PRIMARY KEY (prod_ID, person_ID),
    FOREIGN KEY (prod_ID) REFERENCES Product (ID),
    FOREIGN KEY (person_ID) REFERENCES Person (ID)
);
```

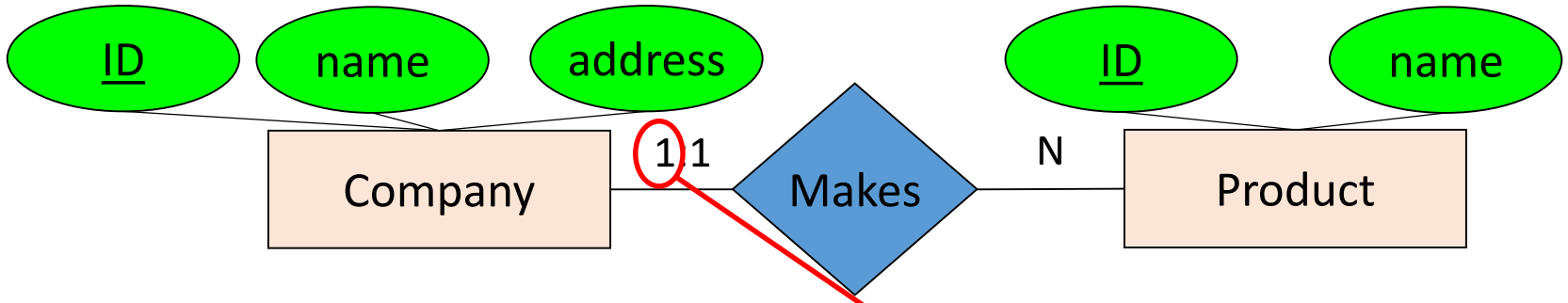
Many-to-many with minimum cardinality



Classic chicken-and-egg problem:

No **Product** can be created without a related **Person**.
No **Person** can be created without a related **Product**.

One-to-many with minimum cardinality



Company

ID	name	address
…	…	…

```
CREATE TABLE Company (
    …
    PRIMARY KEY (ID)
);
```

Product

ID	name
…	…

```
CREATE TABLE Product (
    …
    PRIMARY KEY (ID),
    CHECK chk_made CHECK all_Product_ID_in_Makes()
);
```

User-defined function

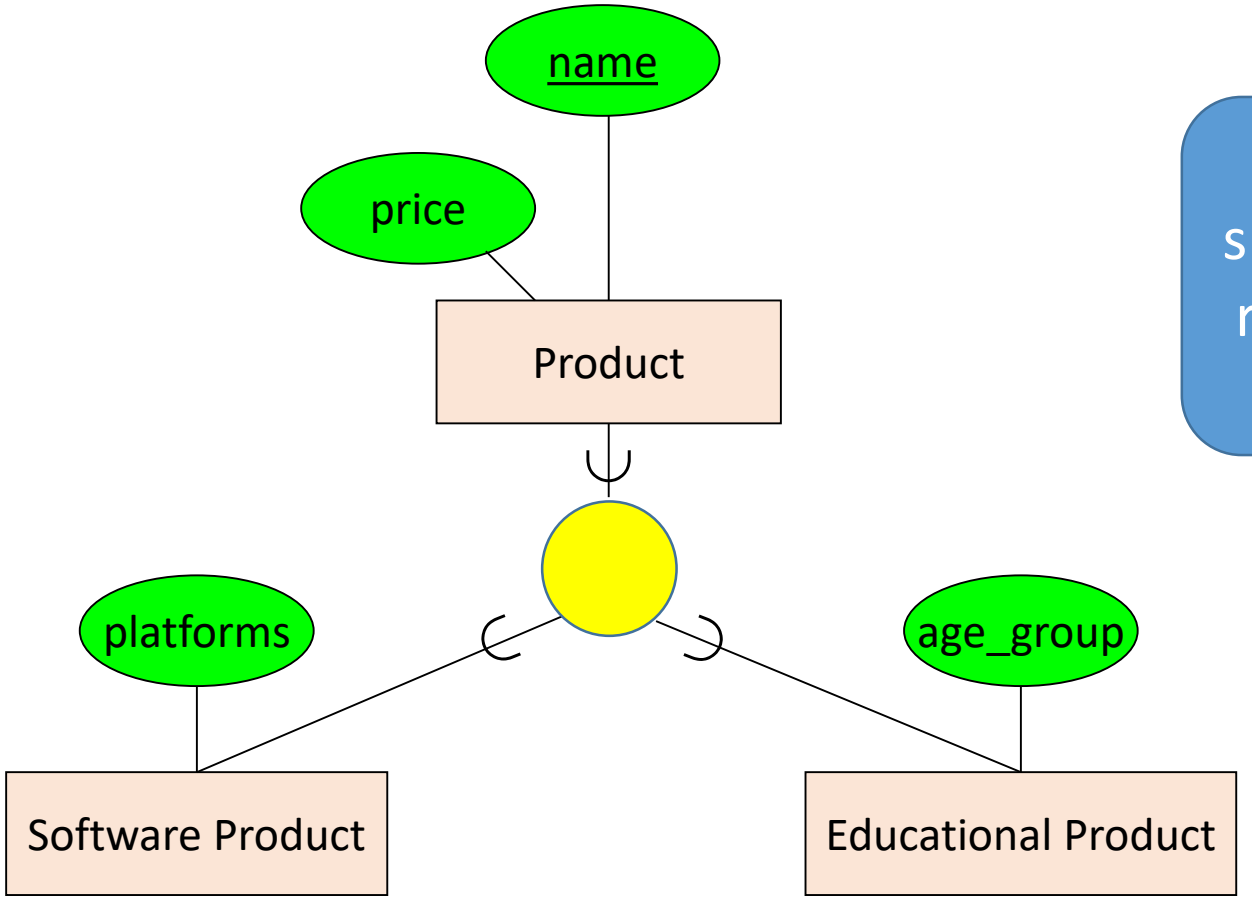
Makes

co_ID	prod_ID
…	…

```
CREATE TABLE Makes (
    …
    PRIMARY KEY (co_ID),
    FOREIGN KEY (co_ID) REFERENCES Company (ID),
    FOREIGN KEY (prod_ID) REFERENCES Product (ID)
);
```

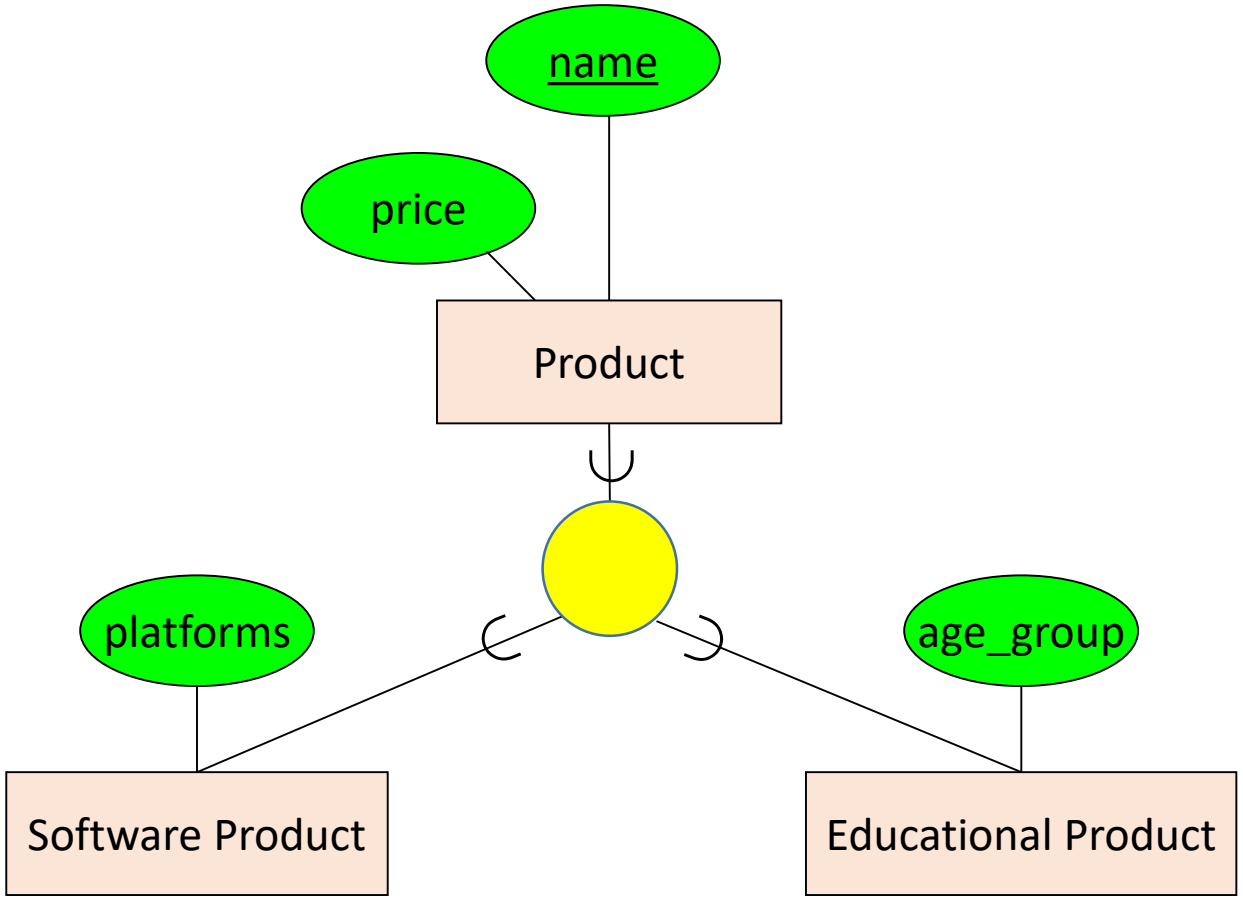
Subclasses, superclasess, unions

Subclasses – Three strategies



Dealing with sub- & superclasses is one of relational database's weaknesses.

Implement both subclasses & superclass



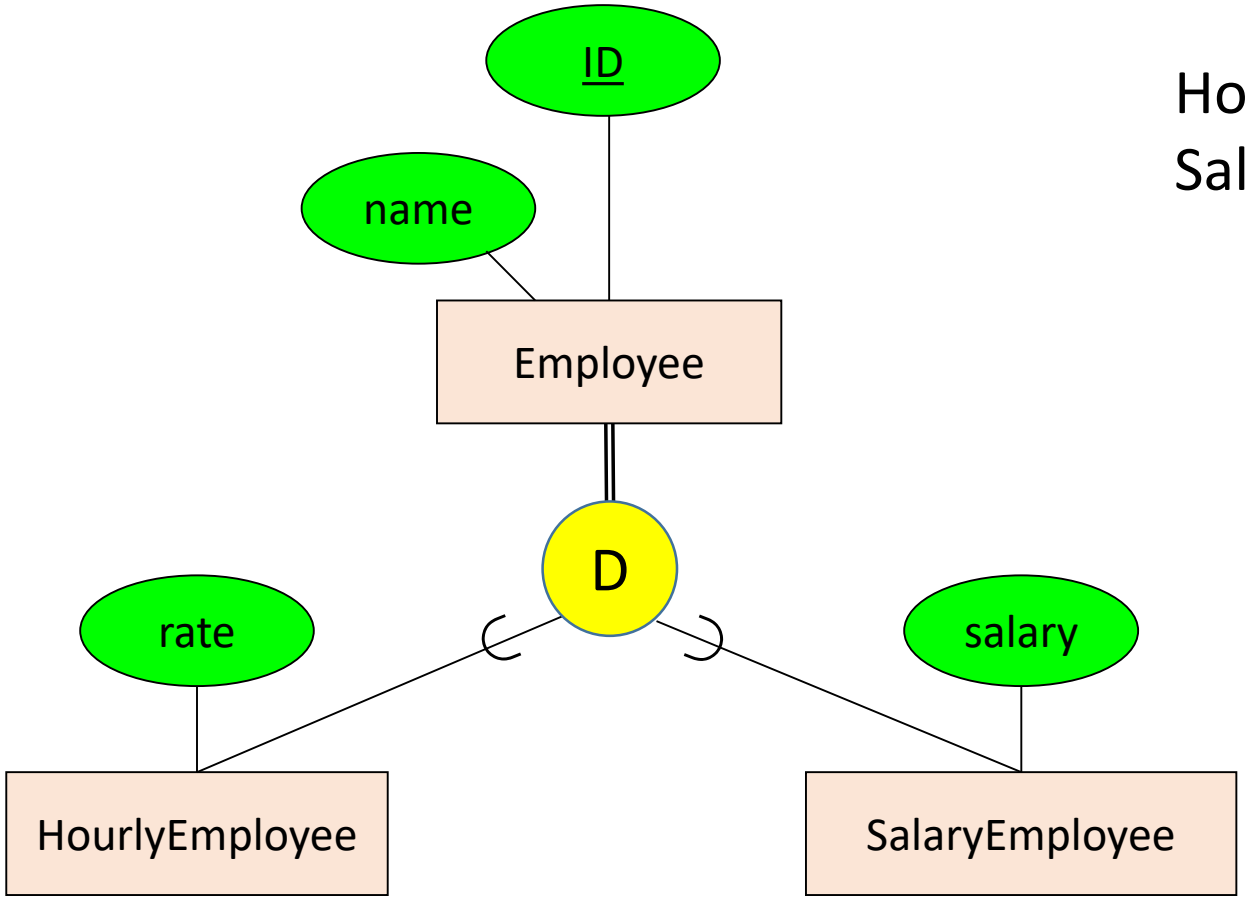
Product(<u>name</u>, price)

SoftwareProduct(<u>name</u>, platforms)
EducationalProduct(<u>name</u>, age_group)

+ Semantically, most accurate representation.

- Typically leads to lots of joins between sub- & superclass

Implement only subclasses

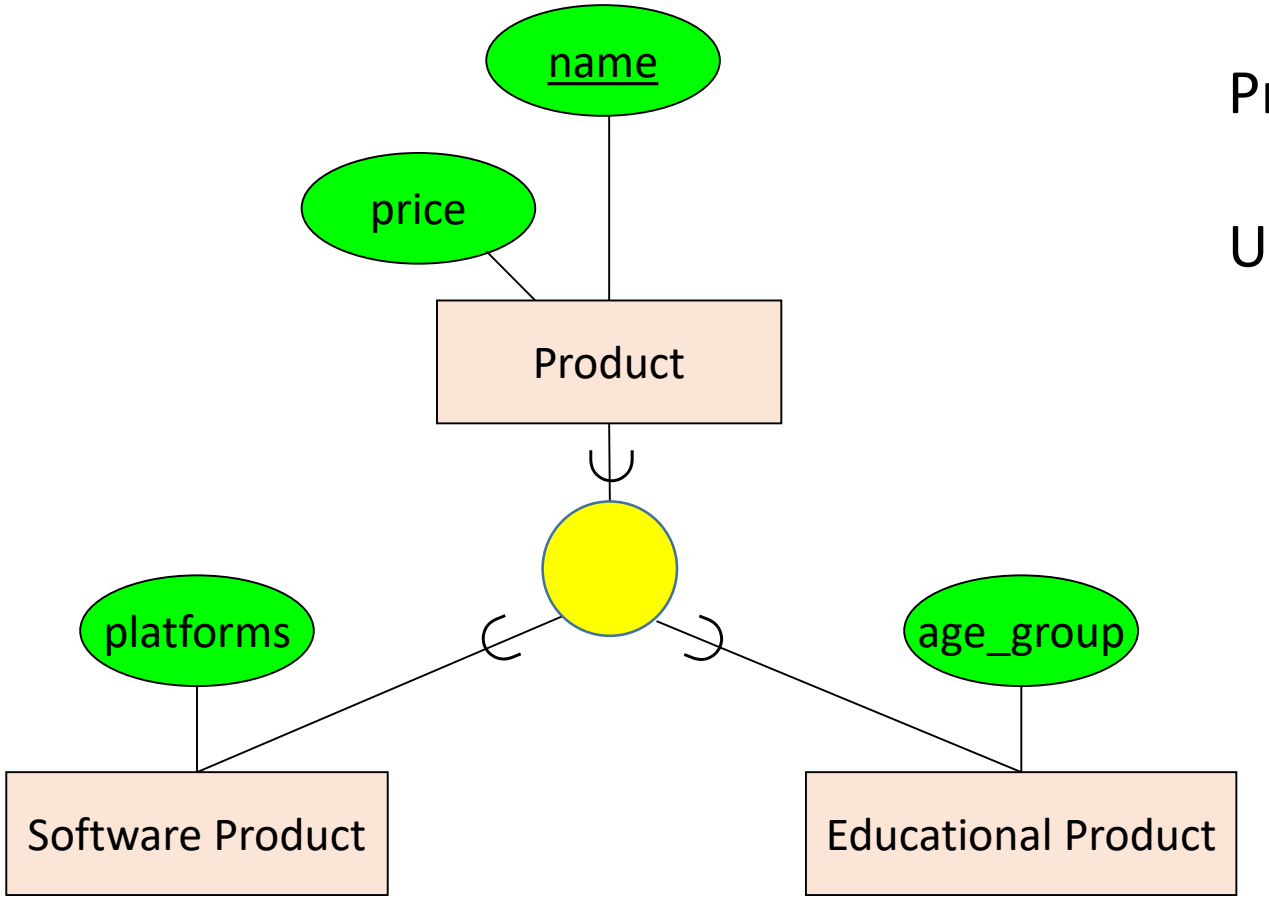


HourlyEmployee(<u>ID</u>, name, rate)
SalaryEmployee(<u>ID</u>, name, salary)

+ Avoids joins between sub- & superclasses.

- Requires disjoint subclasses.
- Poor if superclass is directly related to other entity sets.

Implement only superclass



Product(<u>name</u>, price, platforms, age_group)

Use NULL when attribute not applicable.

+ Avoids joins between sub- & superclasses.

- Joins between subclass & other entity sets now involve more data.

Activity: Implement as Schemas