INTRODUCTION TO RELATIONAL DATABASE SYSTEMS

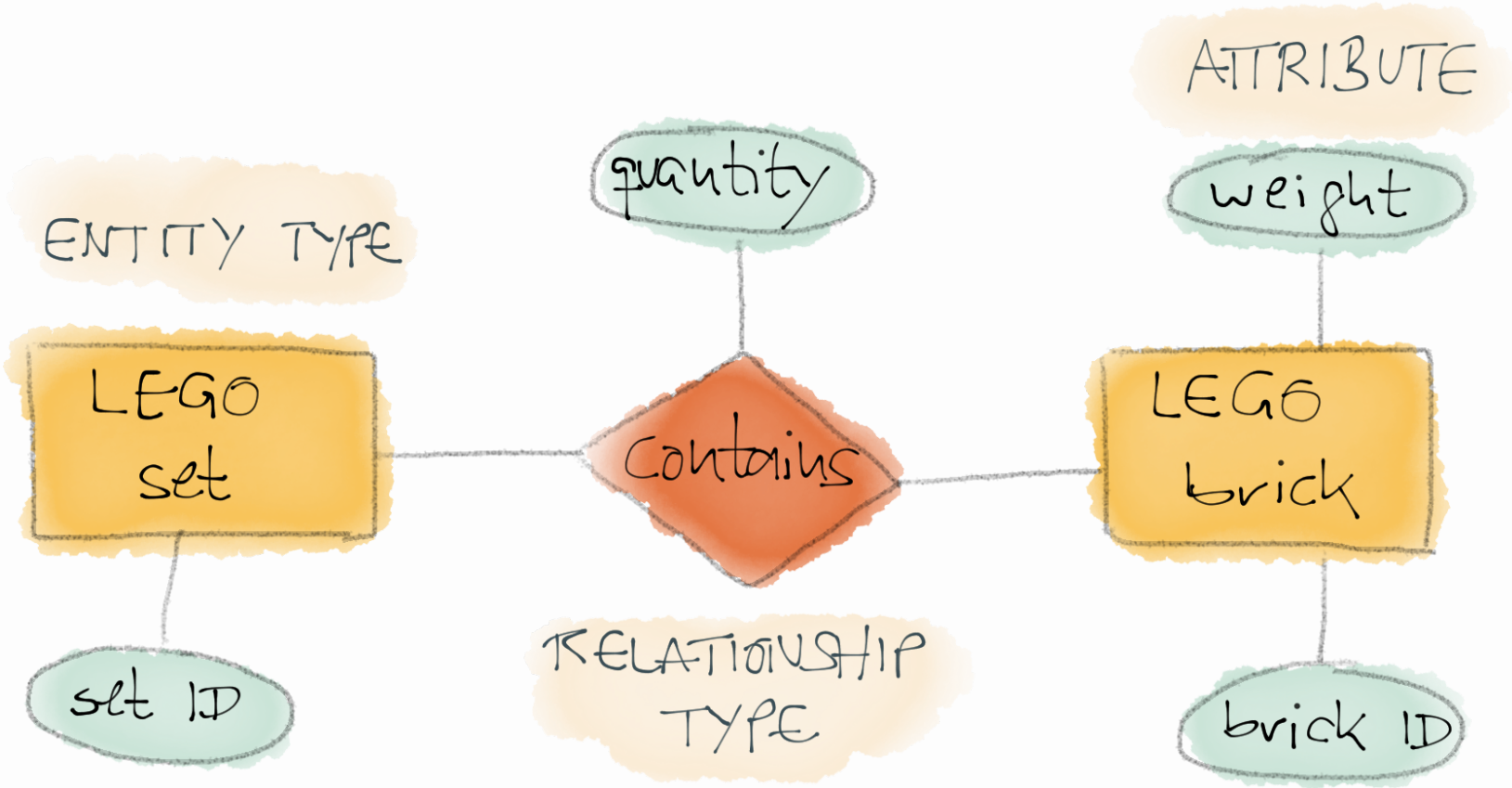
DATENBANKSYSTEME 1 (INF 3131)

Torsten Grust
Universität Tübingen
Winter 2021/22

THE ENTITY–RELATIONSHIP MODEL

- The **Entity–Relationship Model (ER model)** has been defined to concisely describe mini-worlds:
  - ER describes objects **(entities)** and their **relationships.** Objects and relationships carry **attributes** that characterize them further.
  - ER comes with a **graphical notation (ER diagrams)** that helps to establish a quick overview of complex mini-worlds. Also a great way to communicate models to domain experts/DB non-experts/future DB users.
- There are *no* ER model DBMSs. ER is also known as a **semantic data model.** Instead, **ER diagrams are translated** into the constructs of a concrete data model (e.g., relational or JSON) to generate a database schema.
  - While the ER model/diagram is being designed, the focus exclusively remains on mini-world aspects. Details of the implementing data model — e.g., types, constraints — only come into play when the (automatic) translation is performed.

ER: DIAGRAMS



Sample ER Diagram

ER CONSTRUCTS: ENTITIES, ENTITY TYPES

Entity and **Entity Types**

Entities represent relevant mini-world **objects** to be held in the database. Examples: LEGO sets, bricks, earthquakes, calendars.

Each entity has a specific **entity type** that defines its attributes.

The mini-world can contain only a **finite number of objects.** Entities are distinguishable from one another, i.e., entities possess some form of **identity.**

- Notes:

1. Entities do not have to correspond to objects of physical existence but may also represent conceptual objects like, for example, *vacations*.
2. Entity identity is inherent, *not* based on (attribute) values.
   Upon translation, however, entity identity may often be implemented in terms of attribute values (e.g., through LEGO brick IDs, travel agency booking numbers).

ER CONSTRUCTS: RELATIONSHIPS, RELATIONSHIP TYPES

Relationship and **Relationship Type**

Relationships establish a connection between two entities (binary relation). One entity may participate in multiple relationships (or none). See **cardinalities** below.

Each relationship has a specific **relationship type** that defines its attributes and the two participating entity types.

Examples:
"A LEGO **set** ‹*entity type*› **contains** ‹*relationship type*› one or more LEGO **bricks** ‹*entity type*›." "An **earthquake** ‹*entity type*› **occurs in** ‹*relationship type*› one given geographical **region** ‹*entity type*›."

- Relationship types may be viewed from the angle of both participating entity types: "A LEGO set *contains* one or more LEGO bricks." — "A LEGO brick *is contained in* one or more LEGO sets."

ER CONSTRUCTS: ATTRIBUTES

Attributes

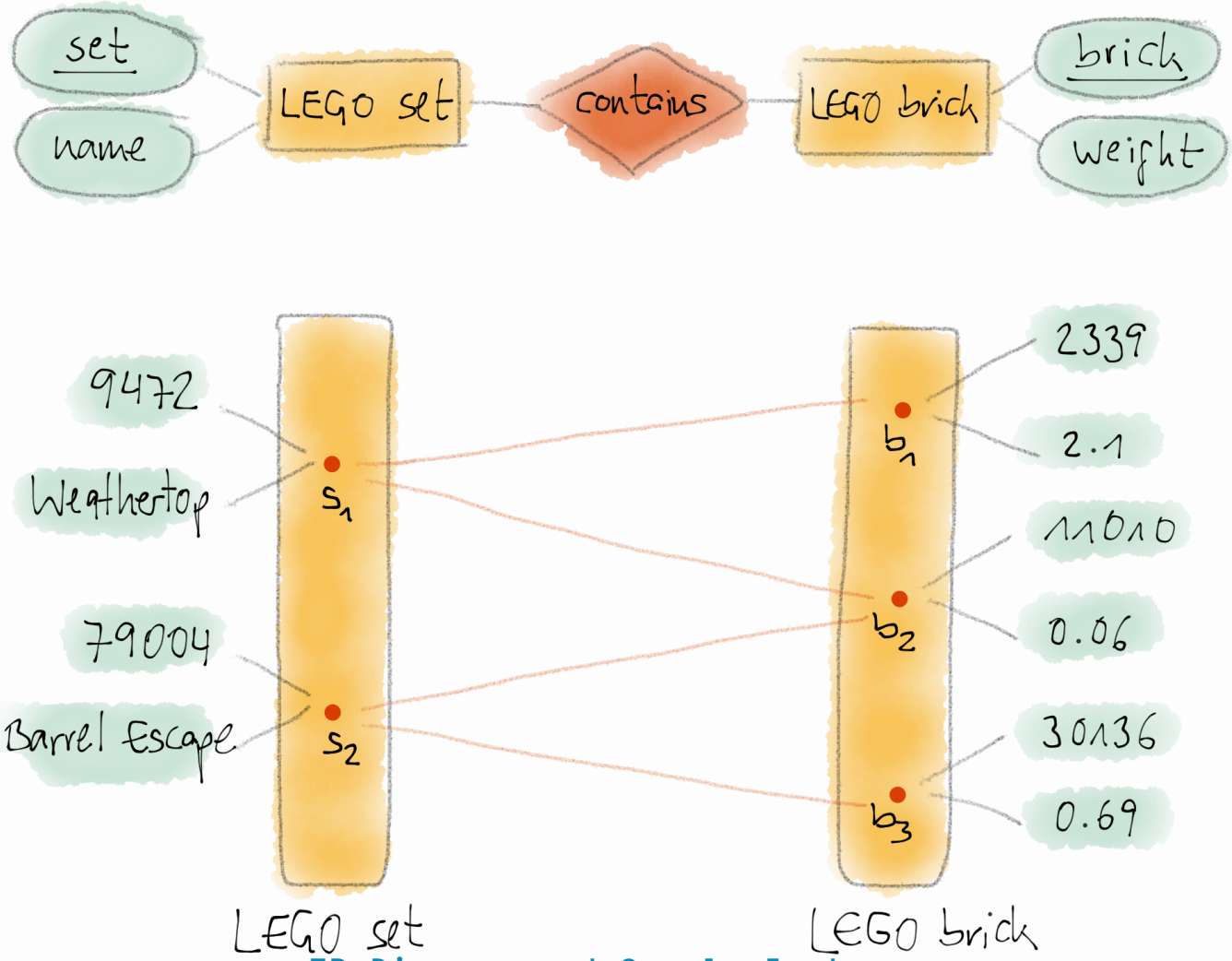
Entity and relationship types may carry **attributes** to represent properties of entities and relationships.

Attribute values may be of arbitrary data types (typically simple/atomic, e.g., string, number, date…). ER is first-order: attributes may *not* have values of type entity or relationship. Upon translation, the target data model deals with the representation of attribute values.

Selected attributes of an entity type may form a **key** (those attributes are underlined in ER diagrams): no two entities of the same entity type may feature the same key values.

- Key attributes exist *in addition* to entity identity and are helpful while translating an ER model into a target data model with value-based keys (e.g., relational).

SAMPLE ER INSTANCE (MINI-WORLD STATE)



ER Diagram and Sample Instance

ER DIAGRAM SEMANTICS

ER Diagram Semantics

The **ER Diagram Semantics** $\mathbb{ER}$ interpret the symbols of an ER diagram by defining

1. a finite set $\mathbb{ER}(e)$ (of entities) for every **entity type** e,
2. a mapping $\mathbb{ER}(a) : \mathbb{ER}(e) \to \mathbb{V}(a)$ for every **attribute** a **of an entity type** e (if a is a key attribute, the mapping is injective),
3. a binary relation $\mathbb{ER}(r) \subseteq \mathbb{ER}(e_1) \times \mathbb{ER}(e_2)$ for every **relationship type** r between entity types e_1 and e_2,
4. a mapping $\mathbb{ER}(a) : \mathbb{ER}(r) \to \mathbb{V}(a)$ for every **attribute** a **of a relationship type** r.

- Recall that $\mathbb{V}(a)$ denotes the set of admissable values for attribute a.

ER DIAGRAM SEMANTICS

- **Example:** The ER diagram semantics associated with the ER diagram and instance shown earlier defines:

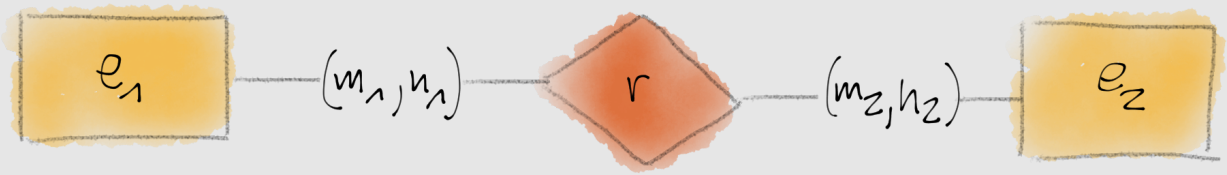
1. $\mathbb{ER}(\text{LEGO set}) = \{s_1, s_2\}$
   $\mathbb{ER}(\text{LEGO brick}) = \{b_1, b_2, b_3\}$
2. $\mathbb{ER}(\text{set}) = f_1$, $\mathbb{ER}(\text{name}) = f_2$, $\mathbb{ER}(\text{brick}) = f_3$, $\mathbb{ER}(\text{weight}) = f_4$
   with

   $f_1(s_1) = 9472$ $\quad$ $f_1(s_2) = 79004$
   $f_2(s_1) = \text{'Weathertop'}$ $\quad$ $f_2(s_2) = \text{'Barrel Escape'}$
   $f_3(b_1) = 2339$ $\quad$ $f_3(b_2) = 11010$ $\quad$ $f_3(b_3) = 30136$
   $f_4(b_1) = 2.1$ $\quad$ $f_4(b_2) = 0.06$ $\quad$ $f_4(b_3) = 0.69$
3. $\mathbb{ER}(\text{contains}) = \{(s_1,b_1), (s_1,b_2), (s_2,b_2), (s_2,b_3)\}$

ER RELATIONSHIP CARDINALITIES

- In general, there is no restriction on **how often** a given entity **participates** in a relationship r.
- Specific **application semantics**, however, may dictate that participation is optional, mandatory, or that a minimum and/or maximum number of participations is required.

Relationship Cardinality (min/max Notation)



$$\forall\, e \in \mathbb{ER}(e_1)\colon m_1 \leqslant |\, \{\, (x,y) \in \mathbb{ER}(r) \mid x = e \,\}\, | \leqslant n_1$$
$$\forall\, e \in \mathbb{ER}(e_2)\colon m_2 \leqslant |\, \{\, (x,y) \in \mathbb{ER}(r) \mid y = e \,\}\, | \leqslant n_2$$

- Notation: $n_i = *$ is interpreted as $n_i = \infty$.

ER DIAGRAM EXAMPLES

- Sketch ER diagrams to model the following mini-worlds. Identify entities and relationships, then restrict relationship cardinalities as required by the application:

1. *"A **man** can be **married to** at most one **woman** and vice versa."*
2. *"An airport lies in exactly one country. A country may have arbitrarily many airports (and maybe none at all)."*
3. *"Orders are placed by customers. Some customers might not have placed an order yet."*
4. *"An order can contain several products."*
5. *"In the Euclidean 2d plane, a line segment connects two points. A simple closed polygon is formed by a list of three or more line segments. The interior of a polygon is colored."*

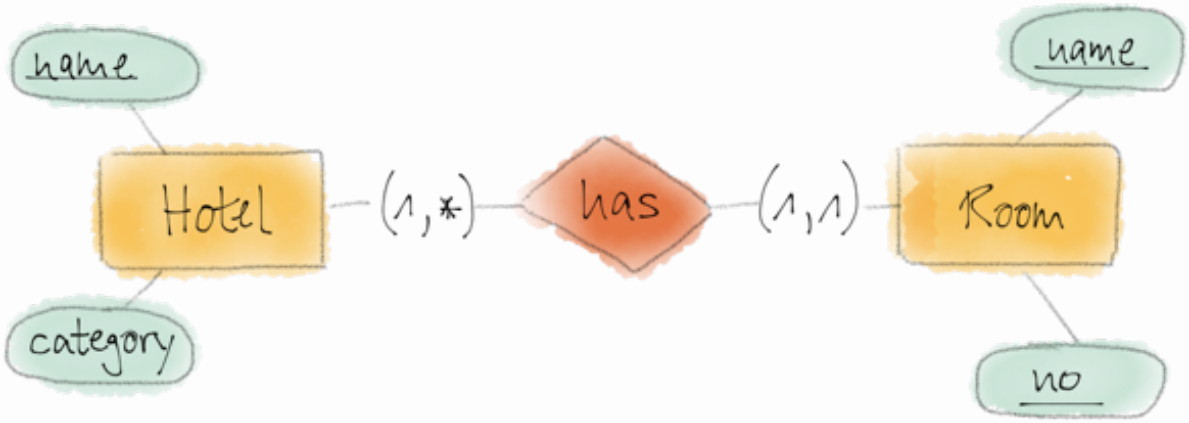
ER: NOTES ON RELATIONSHIP CARDINALITIES

- Cardinality (0,*) represents an unrestricted relationship. A cardinality restriction (m_1,n_1) is **weaker** than (m_2,n_2) if $m_1 \leqslant m_2$ and $n_2 \leqslant n_1$.
- Relationship cardinalities denote **constraints** and have to be translated as such into the target data model.
  - If the target is relational, the important cardinalities (0,1), (1,1), and (0,*) can be directly enforced by relational constraints, e.g., through NOT NULL or keys.
    ⚠ General cardinality constraints (m,n) may not translate (directly), however.
- Conventionally, relationships are categorized by their **maximum cardinalities** on both sides:

$-(m_1,n_1)-\diamond-(m_2,n_2)-$	Relationship Category
$n_1 = 1,\ n_2 = 1$	one-to-one (1:1)
$n_1 = *,\ n_2 = 1$	one-to-many (1:*N*)
$n_1 = 1,\ n_2 = *$	many-to-one (*N*:1)
$n_1 = *,\ n_2 = *$	many-to-many (*M*:*N*)

EXTENDED ER (EER): WEAK ENTITIES

- Common scenario: use an entity type to represent a **detail** of a superordinate **master** entity type. Without the master, the detail would not exist.
- **Example:**



  - “*One hotel has many rooms.*” (General: “*One master has many exclusive details.*”)
  - Detail entity type Room needs to form a **composite key** (incorporating the key of master entity type Hotel) for full identification.
  - Implicit constraint: if detail entity e_2 is in relationship with master entity e_1, both agree in the master’s key (here: attribute name).

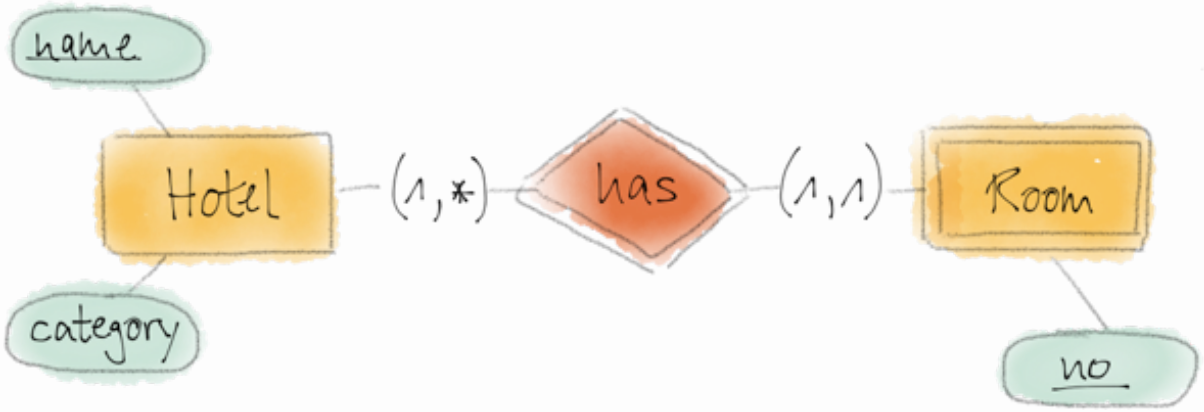
EER: WEAK ENTITIES

Weak Entity Type

In ER diagrams, a **weak entity type** (detail) and its relationship with the master entity type are drawn using **double-stroked lines.**

The weak entity type **implicitly inherits the master key's attributes** (and adds its own key attributes to form a composite key). The **existence** of weak entities **depends** on their master entity (affects translation to the target data model).

- **Example** (continued):



EER: WEAK ENTITIES

- Examples of **master**–detail scenarios in mini-worlds (identify existence dependencies and composite keys):
  1. *"An **invoice** has a number of positions."*
  2. *"A section in a book is identified by a **chapter** and section title."*
  3. *"A web page URI is composed of a **web server** DNS address and a path on that server."*
- Develop an ER diagram to model quizzes (multiple choice tests):

> *"Each quiz is identified by a title, each question within a quiz is numbered, and each possible answer to a given question is referenced by a letter.*
> *For each question and answer, the associated text is stored.*
> *Answers are classified into correct and incorrect ones."*

(What is the complete key for each of the occurring entity types?)

EER: INHERITANCE

Inheritance (Entity Sub/Supertypes)
Entity type e_s is a **subtype** of entity type e if every entity of subtype e_s indeed also is an entity of **supertype** e (i.e., $\mathbb{ER}(e_s) \subseteq \mathbb{ER}(e)$). (**Specialization**)

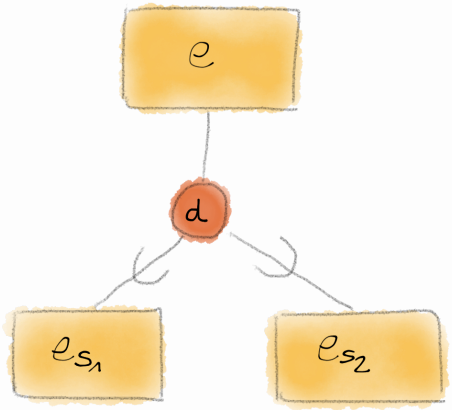
Subtype e_s **inherits all attributes and relationships** from e. On top of these e_s may add further attributes and relationships.

- Supertype e may have multiple subtypes e_{s1}, ..., e_{sn}. Consider the following inheritance scenarios:

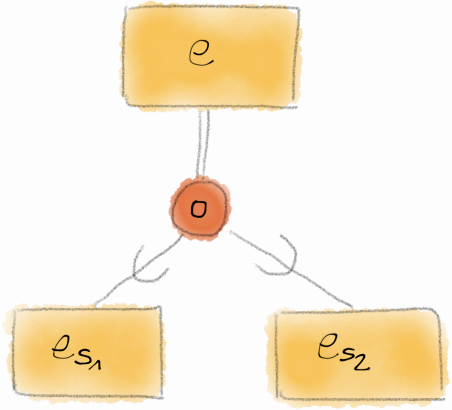
1. **Disjoint** (vs. overlap):
   One entity may belong to a *single* subtype only, i.e., $\forall\ i \neq j$: $\mathbb{ER}(e_{si}) \cap \mathbb{ER}(e_{sj}) = \phi$.
2. **Total** (vs. partial):
   Every entity *must* be a member of a subtype, i.e., $\mathbb{ER}(e) = \mathbb{ER}(e_{s1}) \cup \cdots \cup \mathbb{ER}(e_{sn})$ (cf. with abstract classes in OOPLs).

EER: INHERITANCE

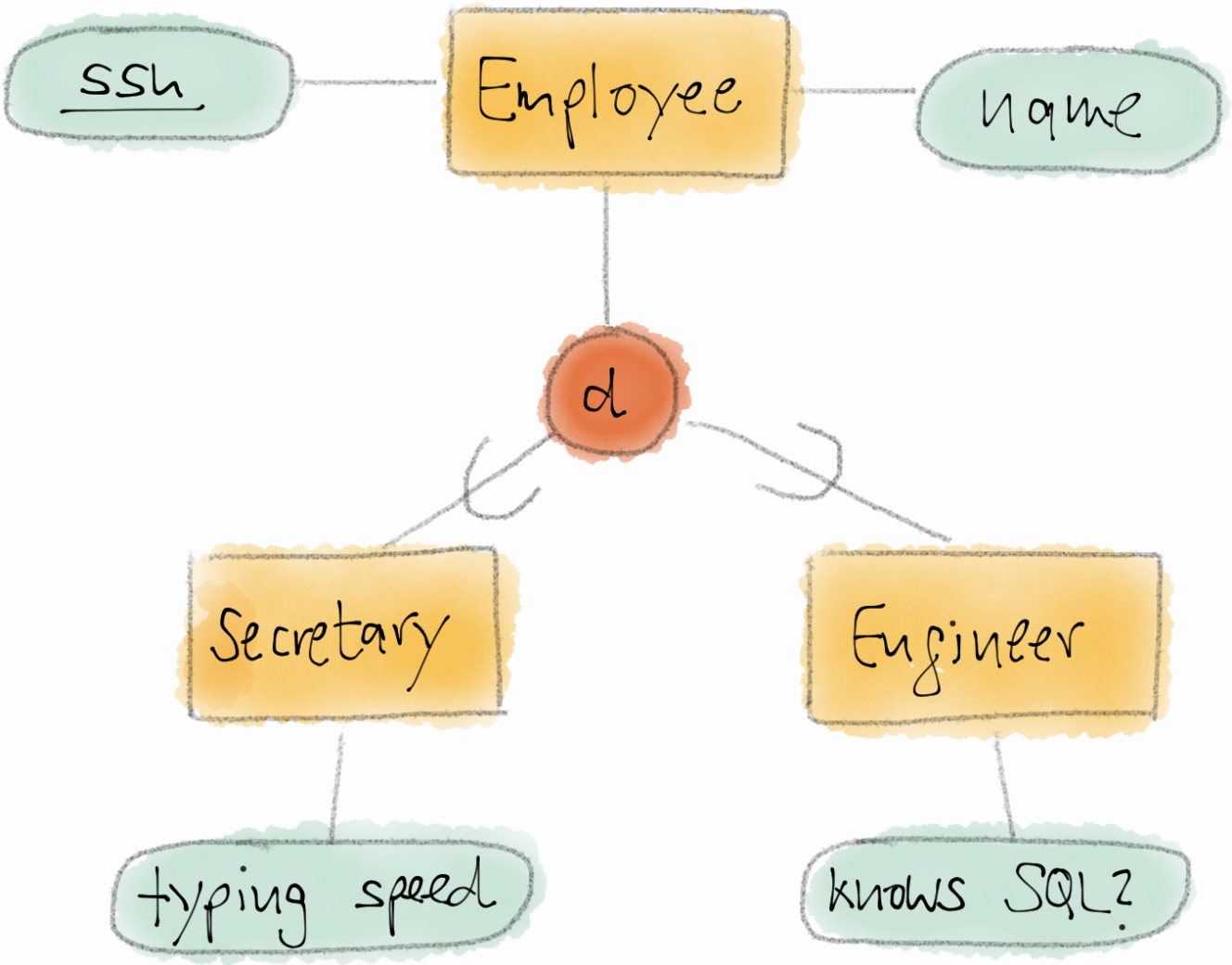
1. EER diagram: partial inheritance (-○), disjoint subtypes (d in ○):



2. EER diagram: total inheritance (=○), overlapping subtypes (o in ○):

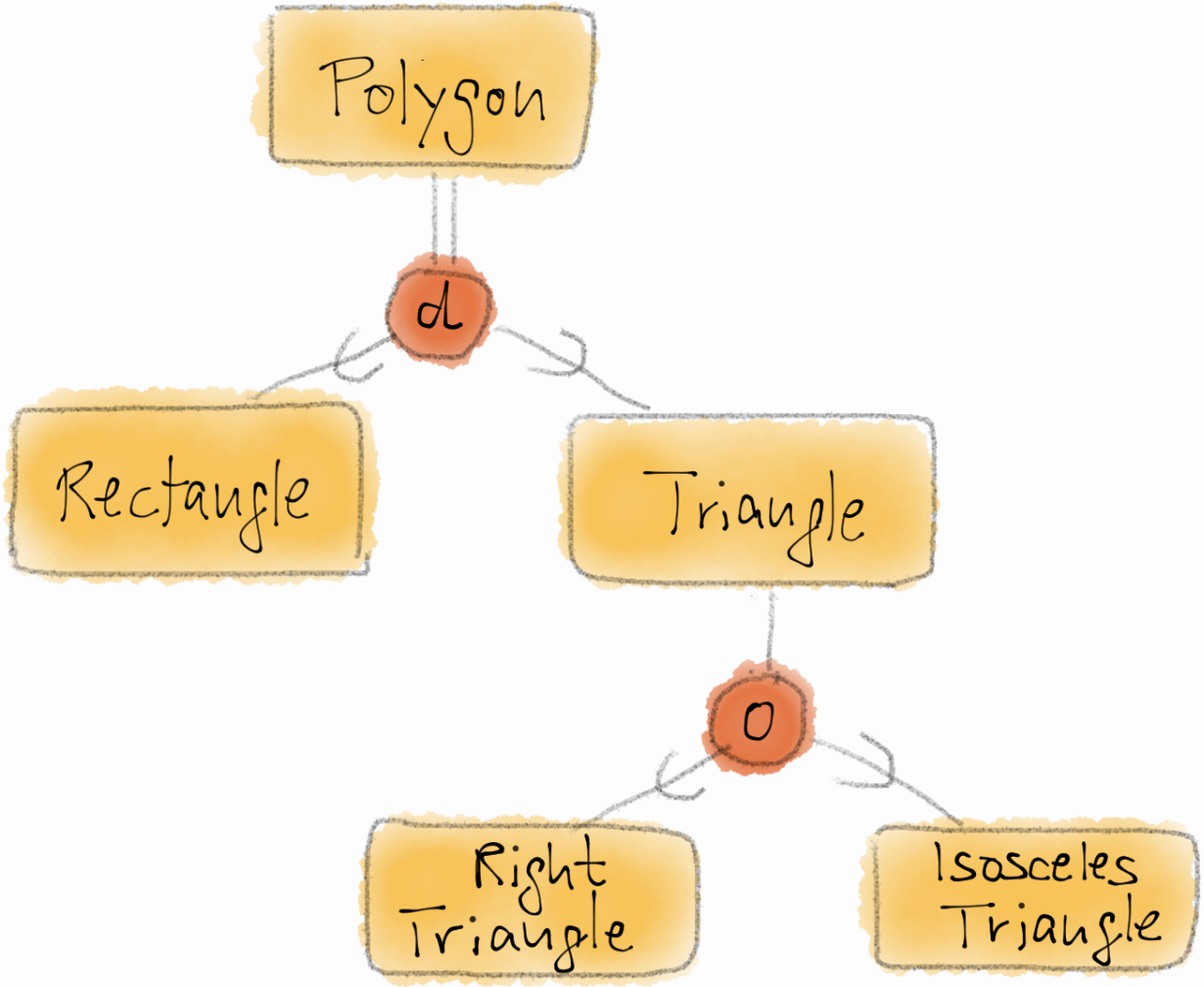


EER: INHERITANCE (EXAMPLES)



EER diagram example

EER: INHERITANCE (EXAMPLES)



EER diagram example (specialization hierarchy)

MAPPING EER TO RELATIONAL TARGET (SQL)

Step #1: Entity Types

- Transforming entity type *e*:

1. CREATE TABLE named *t* = *plural*(*e*)
   (each row of the table will hold one instance of type *e*).
   - Pluralization ("*class*" → "*classes*", "*story*" → "*stories*", "*person*" → "*people*") :
     (e.g., Damian Conway, Monash U, "A pluralization algorithm for English",
     `http://www.csse.monash.edu.au/~damian/papers/HTML/Plurals.html`)
   - Python: `plural()` (in package/module `inflect`)
   - Ruby: `String#pluralize` (see *Ruby on Rails'* ActiveSupport)
2. Each **attribute** *a* of entity type *e* becomes a **column** *a* of table *t*
   (map attribute data type to SQL data type with appropriate value domain).
   If any, list attributes marked as key first.

MAPPING EER TO RELATIONAL TARGET (SQL)

Step #1b: Entity Type Keys

- Transforming entity type *e* (target table *t*):

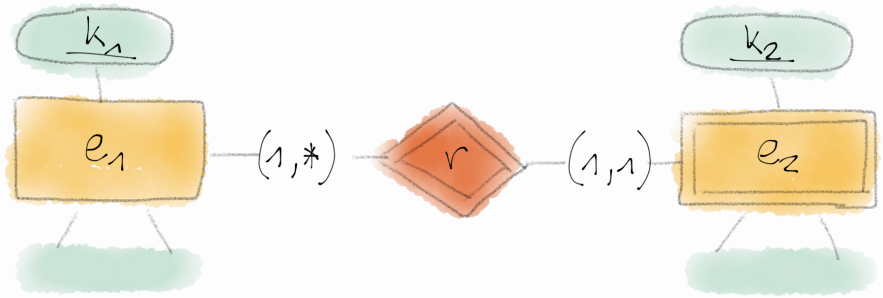
1. If present, a **value-based primary key** of the entity type becomes the **primary key of table** *t*
   (`ALTER TABLE t ADD PRIMARY KEY (…)`)
   - If *e*'s key is composite, so will be the relational key.
2. Otherwise, implement the **inherent entity identity** in terms of an **artificial key column**, say `__id__`, of auto-incrementing `integer` type:

```
CREATE TABLE t (…);

-- establish artificial key column to implement entity identity
ALTER TABLE t ADD COLUMN __id__ integer GENERATED ALWAYS AS IDENTITY;
ALTER TABLE t ADD PRIMARY KEY (__id__);
```

MAPPING EER TO RELATIONAL TARGET (SQL)

Step #2: Weak Entity Types

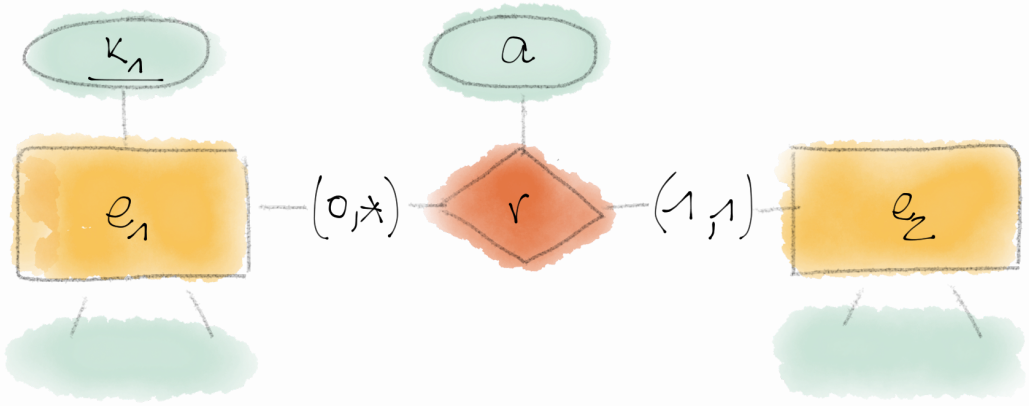


1. Assume table t_i = *plural*(e_i) translates e_i.
2. In t_2, add column r as a **foreign key** referencing table t_1 (**implements relationship** r).
3. Replace key of t_2 with **composite primary key** (r, k_2).
4. Implement existence dependency via an ON DELETE CASCADE action on foreign key r:

```
ALTER TABLE t2 ADD COLUMN r <type of k1> NOT NULL;
ALTER TABLE t2 DROP CONSTRAINT t2_pkey;   --  ◄─┐ replace primary key
ALTER TABLE t2 ADD PRIMARY KEY (r, k2);   --  ◄─┘   of table t2
ALTER TABLE t2 ADD FOREIGN KEY (r) REFERENCES t1(k1) ON DELETE CASCADE;
```

MAPPING EER TO RELATIONAL TARGET (SQL)

Step #3: One-to-Many Relationship Types



1. Assume table t_i = *plural*(e_i) translates e_i.
2. **Implement** r **and** a: add column r to t_2 and establish as a foreign key referencing table t_1 (no dependency, thus no ON DELETE action). Add column a to t_2:

```
-- establish foreign key to implement relationship r
ALTER TABLE t2 ADD COLUMN r <type of k1> NOT NULL;   -- ◄┐ realize (1,_) card
ALTER TABLE t2 ADD COLUMN a <type of a> NOT NULL;    -- ◄┘
ALTER TABLE t2 ADD FOREIGN KEY (r) REFERENCES t1(k1);
```

MAPPING EER TO RELATIONAL TARGET (SQL)

Step #3: One-to-Many Relationship Types (Cardinalities)

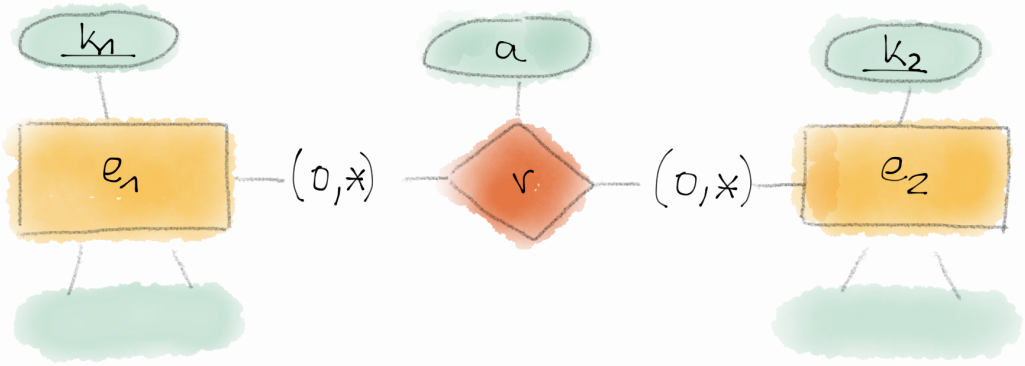
1. –(0,*)–◇–(1,1)–
   The key value of a given e_1 entity does *not* necessarily appear in column r of t_2.
2. –(0,*)–◇–(1,1)–
   The key value of a given e_1 entity may appear *multiple times* in column r of t_2.
3. –(0,*)–◇–(1,1)–
   Value of column r in t_2 is NOT NULL (remove NOT NULL for –(0,*)–◇–(0,1)–).
4. –(0,*)–◇–(1,1)–
   Column r has exactly *one value* in each row of t_2.

⚠ **But:** Relational NOT NULL, key, and foreign key constraints **cannot implement general ER cardinality constraints.** Unimplementable:

- –(m_1,*)–◇–(_,1)– with $m_1 > 0$ and –(0,n_1)–◇–(_,1)– with $n_1 > 0$, $n_1 \neq *$

MAPPING EER TO RELATIONAL TARGET (SQL)

Step #3b: Many-to-Many Relationship Types



1. Assume table t_i = *plural*(e_i) translates e_i.
2. **Implement** r **and** a: CREATE TABLE r with columns k_1, k_2, a to represent the relationship. Establish (k_1, k_2) as **composite key** for table r. In table r, establish k_i as foreign key referencing table t_i with ON DELETE actions.

```
CREATE TABLE r (k1 <type of k1>, k2 <type of k2>, a <type of a>);
ALTER TABLE r ADD PRIMARY KEY (k1, k2);
ALTER TABLE r ADD FOREIGN KEY (k1) REFERENCES t1(k1) ON DELETE CASCADE;
ALTER TABLE r ADD FOREIGN KEY (k2) REFERENCES t2(k2) ON DELETE CASCADE;
```

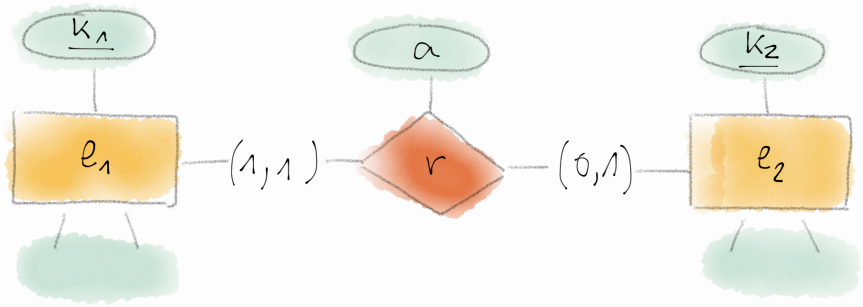
MAPPING EER TO RELATIONAL TARGET (SQL)

Step #3b: Many-to-Many Relationship Types

- **Quiz:**
  1. Explain the choice of *composite* primary key (k_1, k_2) for table r.
  2. What kind of mini-world is implemented if we establish column k_1 (or k_2) as the *only* key column in table r? Is the result useful?
- **Notes on many-to-many relationship cardinalities:**
  - Cardinalities other than –(0,*)–◇–(0,*)– cannot be enforced by the relational model. In particular, –(*m*,*)–◇–(_,*)– with $m > 0$ cannot be translated faithfully.
  - If such scenarios are important in mini-world modelling, the cardinality constraint needs to checked by
    1. the **application program** or
    2. a **general SQL constraint mechanism** (non-relational, in the strict sense).

MAPPING EER TO RELATIONAL TARGET (SQL)

Step #3c: One-to-One Relationship Types



1. Assume table t_i = *plural*(e_i) translates e_i.
2. Two options: host **implementation of** r **and** a in t_1 **or** t_2. Here: choose t_1 since **every** e_1 entity participates in r-relationship (–(1,1)–◇⋯): Add column r to t_1 and establish as a foreign key referencing table t_2 (no dependency, thus no ON DELETE action). Add column a to t_1.

```
ALTER TABLE t1 ADD COLUMN r <type of k2> NOT NULL;
ALTER TABLE t1 ADD FOREIGN KEY (r) REFERENCES t2(k2);
ALTER TABLE t1 ADD UNIQUE (r); -- A r becomes candidate key in t1
ALTER TABLE t1 ADD COLUMN a <type of a> NOT NULL;
```

MAPPING EER TO RELATIONAL TARGET (SQL)

Step #3c: One-to-One Relationship Types (Cardinalities)

- Consider how the relational translation implements the ER cardinality constraints:

1. –(1,1)–◇–(0,1)– :

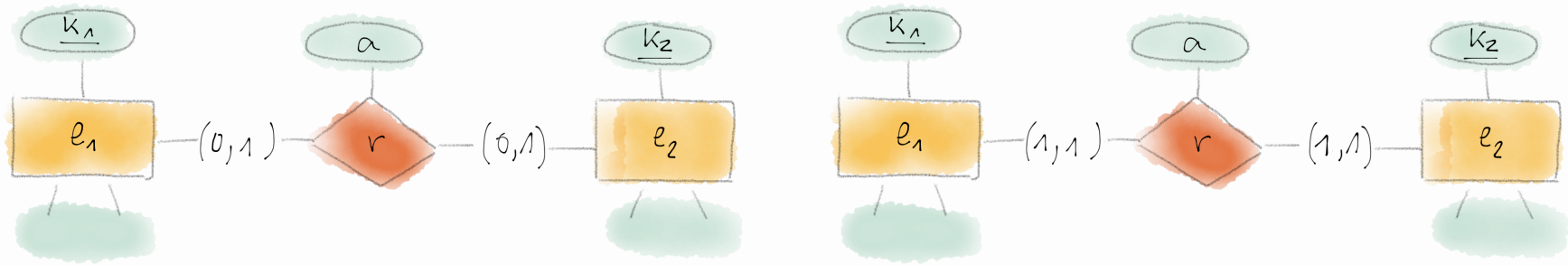
2. –(1,1)–◇–(0,1)– :

3. –(1,1)–◇–(0,1)– :

4. –(1,1)–◇–(0,1)– :

MAPPING EER TO RELATIONAL TARGET (SQL)

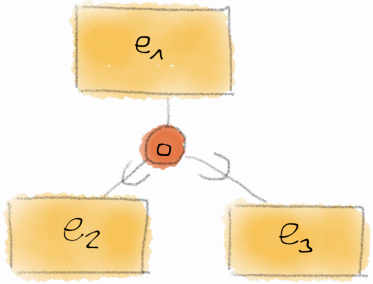
Step #3c: One-to-One Relationship Types (Alternatives)



- –(0,1)–◇–(0,1)– : Two options:
  1. Drop NOT NULL constraint on column r in table t_1.
  2. Represent r (and a) in a **separate table** r with two foreign keys k_i referencing t_i. Both k_i are candidate keys in table r.
- –(1,1)–◇–(1,1)– :
  Merge tables t_1 and t_2 and column a into single table t. No NULL values allowed to enforce minimum cardinalities of 1. Both k_i are candidate keys in table t.

MAPPING EER TO RELATIONAL TARGET (SQL)

Translating Inheritance



1. Assume table t_i = *plural*(e_i) will hold the translation of e_i.
2. **Implement partial, overlapping inheritance:** CREATE TABLEs t_2 and t_3 using SQL's INHERITS declaration to indicate that both inherit from table t_1.

```
CREATE TABLE t1 (…);
CREATE TABLE t2 (…) INHERITS (t1);
CREATE TABLE t3 (…) INHERITS (t1);
```

3. The implementation of **total** or **disjoint** inheritance requires a bit more SQL machinery (e.g., PostgreSQL's CREATE RULE mechanism, see below).

SQL: TABLE INHERITANCE VIA INHERITS

INHERITS (PostgreSQL)

If **subtable** s INHERITS from tables t_1, …, t_n,

```
CREATE TABLE s(…) [ INHERITS (t1 [, …]) ]
```

1. subtable s features all columns of tables t_i *and* those declared locally for s, *i.e.*, $sch(t_i) \subseteq sch(s)$,
2. rows inserted/updated/deleted in s will also affect all tables t_i (but not vice versa), *i.e.*, $inst(s)|sch(t_i) \subseteq inst(t_i)$ ($|C \equiv$ restricted to the columns in set C).

- ⚠️ In PostgreSQL, the subtable does *not* inherit key, foreign key, or UNIQUE constraints of its supertables. Re-declare these for the subtable.
- Note: TABLE t_i produces $inst(t_i)$. TABLE ONLY t_i produces $inst(t_i) \setminus (inst(s)|sch(t_i))$.

THE COMPLETE LEGO DATABASE ER MODEL