Translation of ER to SQL

Iztok Savnik

FAMNIT

Literature

Toby Teorey, Sam Lightstone, Tom Nadeau, H.V. Jagadish, Database Modeling and Design: Logical Design, 5th Edition, Morgan Kaufmann Pub., Inc. (Elsevier), 2011. (Chapter 5)

Introduction

- Basic rules for translating ER diagrams to relations are given
  - Translating entities and relationships
  - Some integrity constraints can be used to implement features of ER model
    - NULL declarations, foreign keys, UNIQUE constraints, functional dependencies
- Not all features of ER model can be translated into SQL

Translation of entities

- SQL table with the same information contents as the original entity from which it was developed
- Translation is used for entities with binary relationship
  - N-N
  - 1-N on „one“ side
  - 1-1 on both sides
- Entities that are linked with binary recursive relationship of type N-N
- Entities with ternary or, in general, n-ary relationship, or, with the generalization hierarchy

Dependent entities

- SQL table with nested foreign key of the parent entity
- Transformation is done for the entities with binary relationship of type
  - 1-N for the entity on the (child) side
  - 1-1 for one of the entities
  - For all entities with the binary recursive relationship of type 1-1 or 1-N
- This is one of two ways that the design tools treat relationships

Translation of relationships

- SQL table derived from a relationship
  - Contains foreign keys of all entities in relationship
- This translation is used for
  - Binary relationships of type N-N
  - Relationships that are recursive and N-N
  - Relationships that are ternary or higher order
- This is the second most common way that the design tools use to treat relationships in ER and in UML
  - Relationship of type N-N is always defined with a table that contains foreign keys of all entities in relationship
  - This table can include the attributes of the original ER relationship

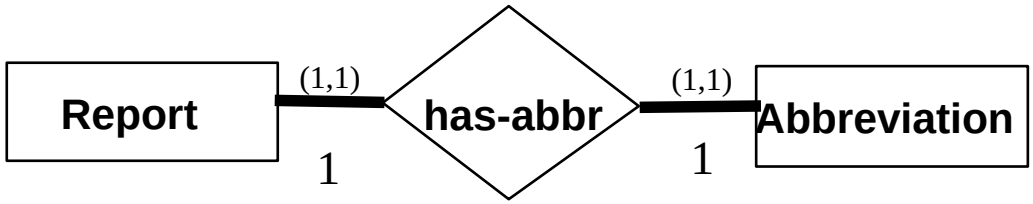
The use of NULL values

- NULL values are allowed for the foreign keys that link optional entities
- NULL values are not allowed in SQL tables for foreign keys that link obligatory entities
- NULL values are not allowed for all foreign keys derived from a binary relationship of type N-N, and, any n-ray relationship where n>2
  - Only the complete rows (with all foreign keys) make sense

Binary relationships

- We will go through the cases of binary relationships based on cardinality of relationships
- How entities are translated? How to translate binary relationships? How to translate optional and mandatory relationships? Which SQL constructs are used?
- We have three types of relationships based on the type of the relationship: 1-1, 1-N in N-N
- Every role in relationship can be either optional or mandatory
- We will present Min-Max and Chen’s cardinality notations !

Relationship 1-1 (2 x mandat.)



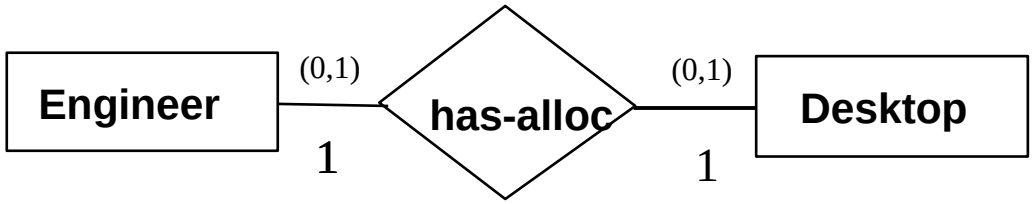

```
create table report
   (report_no integer,
    report_name varchar(256),
    primary key(report_no);

create table abbreviation
    (abbr_no char(6),
     report_no integer not null unique,
     primary key (abbr_no),
     foreign key (report_no) references report
     on delete cascade on update cascade);
```

Relationship 1-1 (1 x mandat.)


```
create table department
    (dept_no integer,
     dept_name char(20),
     mgr_id char(10) not null unique,
     primary key (dept_no),
     foreign key (mgr_id) references employee
          on delete set default on update cascade);
```

```
create table employee
    (emp_id char(10),
     emp_name char(20),
     primary key (emp_id));
```

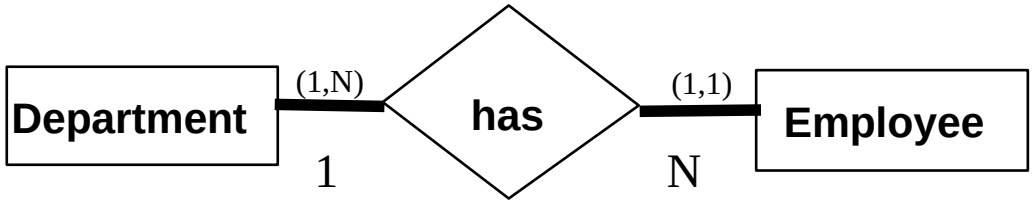
Relationship 1-1 (2 x optional)




```
create table engineer
  (emp_id char(10),
   desktop_no integer,
   primary key (emp_id));
```

```
create table desktop
    (desktop_no integer,
     emp_id char(10),
     primary key (desktop_no),
     foreign key (emp_id) references engineer
     on delete set null on update cascade);
```

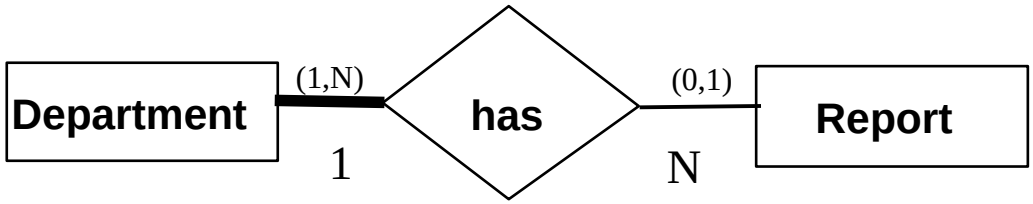
Relationship 1-N (2 x mandat.)




```
create table department
    (dept_no integer,
     dept_name char(20),
     primary key (dept_no));
```

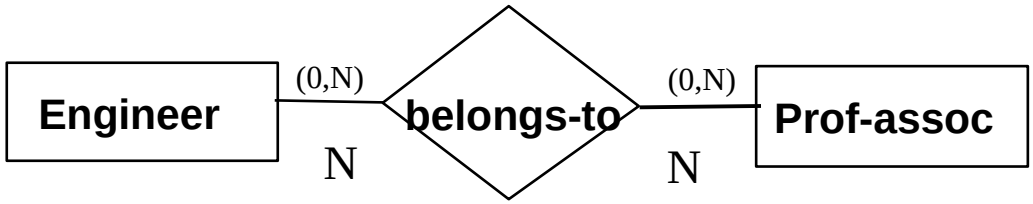
```
create table employee
    (emp_id char(10),
     emp_name char(20),
     dept_no integer not null,
     primary key (emp_id),
     foreign key (dept_no) references department
         on delete set default on update cascade);
```

Relationship 1-N (1 x mandat.)




```
create table department
    (dept_no integer,
     dept_name char(20),
     primary key (dept_no));

create table report
    (report_no integer,
     dept_no integer,
     primary key (report_no),
     foreign key (dept_no) references department
         on delete set null on update cascade);
```

Relationship N-N (2 x optional)

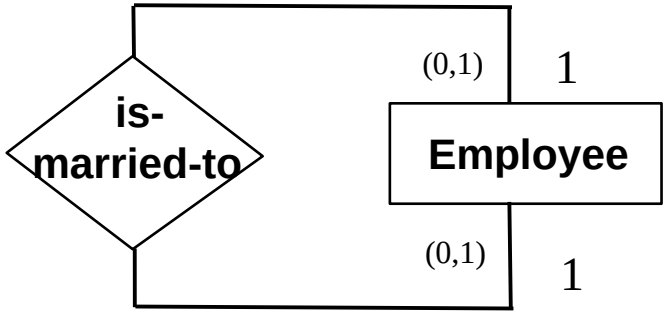



```
create table engineer
    (emp_id char(10),
     primary key (emp_id));

create table prof_assoc
    (assoc_name varchar(256),
     primary key (assoc_name));

create table belongs_to
    (emp_id char(10),
     assoc_name varchar(256),
     primary key (emp_id, assoc_name),
     foreign key (emp_id) references engineer
         on delete cascade on update cascade,
     foreign key (assoc_name) references prof_assoc
         on delete cascade on update cascade);
```

Recursive relationship

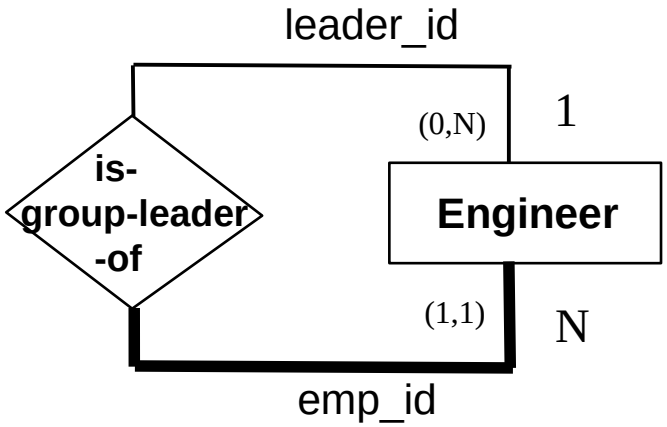
- Binary relationship defined on a single entity
  - Is the set of pairs composed of primary (foreign) keys of the given entity
  - Enrolment of entities from the either side of the relationship can be optional or mandatory
- Two possibe translations into SQL
  - Set of pairs is stored in a separate SQL table
  - Set of pairs can be represented by an additional column of SQL table derived from the given entity

Recursive relationship 1-1 (2 x optional)



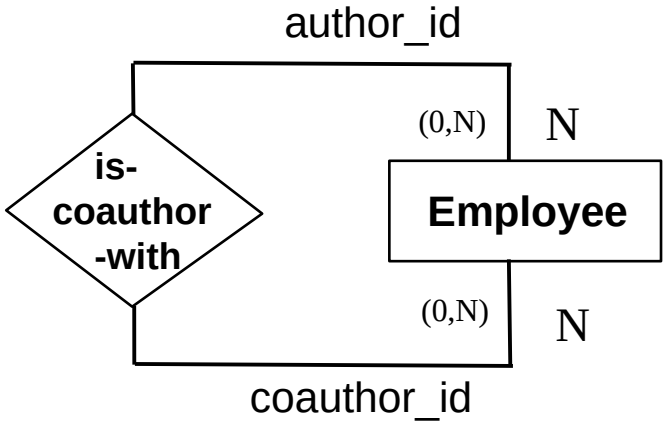

```
create table employee
    (emp_id char(10),
      emp_name char(20),
      spouse_id char(10),
      primary key (emp_id),
      foreign key (spouse_id) references employee
      on delete set null on update cascade);
```

Recursive relationship 1-N (1 x optional)




```
create table engineer
    (emp_id char(10),
     leader_id char(10) not null,
     primary key (emp_id),
     foreign key (leader_id) references engineer
         on delete set default on update cascade);
```

Recursive relationship N-N (2 x optional)

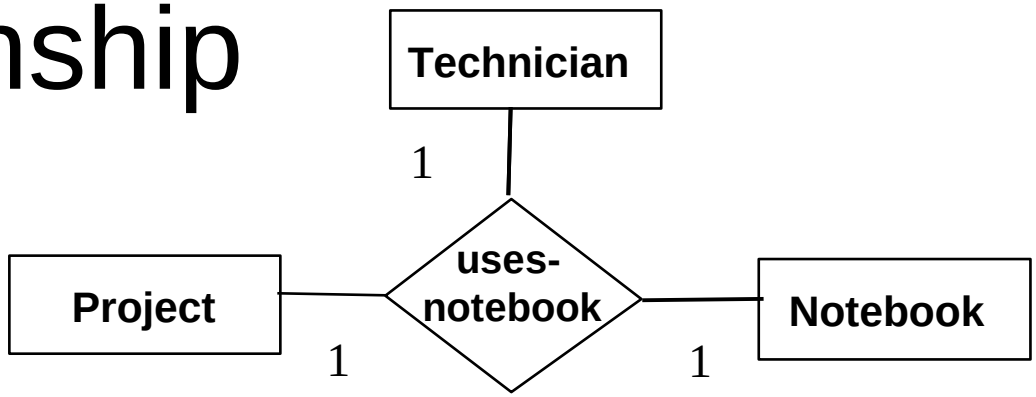



```
create table employee
    (emp_id char(10),
     emp_name char(20),
     primary key (emp_id));

create table coauthor
     (author_id char(10),
      coauthor_id char(10),
      primary key (author_id, coauthor_id),
      foreign key (author_id) references employee
          on delete cascade on update cascade,
      foreign key (coauthor_id) reference employee
          on delete cascade on update cascade);
```

Ternary relationships

- We will have a look at some examples of ternary relationship
- The constraints are important for the translation of ternary relationship
  - Some can be expressed: participation constraints
  - Some can not be expressed with SQL CREATE statement
  - Keys in ternary relationship are always NOT NULL
  - Keys must be updated and deleted

Ternary relationship (1-1-1)



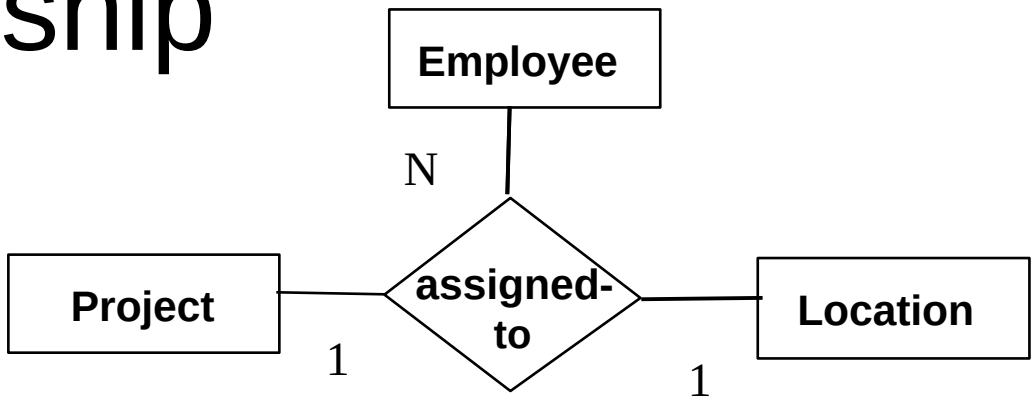

```
create table technician (
    emp_id char(10),
    primary key (emp_id));
create table project (
    project_name char(20),
    primary key (project_name));
create table notebook (
    notebook_no integer,
    primary key (notebook_no));
```

```
create table uses_notebook (
    emp_id char(10),
    project_name char(20),
    notebook_no integer not null,
    primary key (emp_id, project_name),
    foreign key (emp_id) references technician
       on delete cascade on update cascade,
    foreign key (project_name) references project
       on delete cascade on update cascade,
    foreign key (notebook_no) references notebook
        on delete cascade on update cascade,
    unique (emp_id, notebook_no),
    unique (project_name, notebook_no));
```

(1-1-1) – one project, one technican, one notebook

emp_id, project_name → notebook_no
emp_id, notebook_no → project_name
project_name, notebook_no → emp_id

Ternary relationship (1-1-N)




```
create table employee (
   emp_id char(10),
   emp_name char(20),
   primary key (emp_id));
create table project (
   project_name char(20),
   primary key (project_name));
create table location (
   loc_name char(15),
   primary key (loc_name));
```

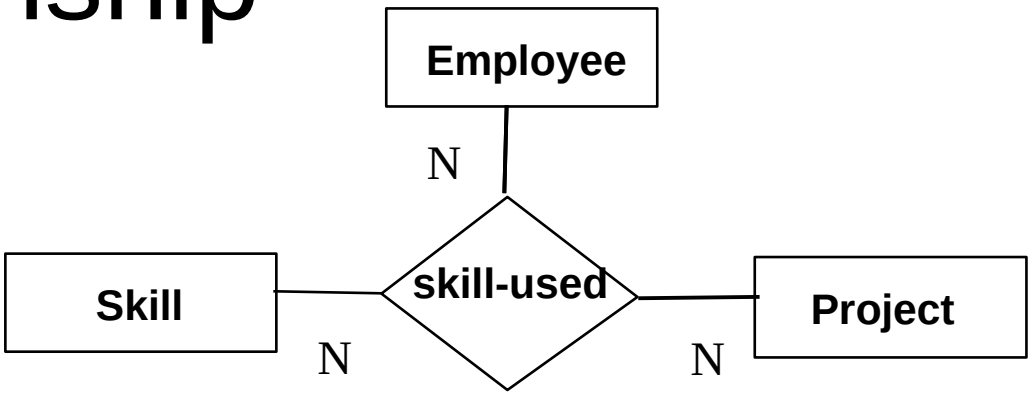
```
create table assigned_to (
   emp_id char(10),
   project_name char(20),
   loc_name char(15) not null,
   primary key (emp_id, project_name),
   foreign key (emp_id) references employee
      on delete cascade on update cascade,
   foreign key (project_name) references project
      on delete cascade on update cascade,
   foreign key (loc_name) references location
      on delete cascade on update cascade,
   unique (emp_id, loc_name));
```

(1-1-N) – one project, one location, more employees

emp_id, loc_name → project_name

emp_id, project_name → loc_name

Ternary relationship (N-N-N)




```
create table employee (
    emp_id char(10),
    emp_name char(20),
    primary key (emp_id));
create table skill (
    skill_type char(15),
    primary key (skill_type));
create table project (
    project_name char(20),
    primary key (project_name));
```

(N-N-N) – skills are arbitrarily linked to employes and projects

```
create table skill_used (
    emp_id char(10),
    skill_type char(15),
    project_name char(20),
    primary key (emp_id, skill_type, project_name),
    foreign key (emp_id) references employee
        on delete cascade on update cascade,
    foreign key (skill_type) references skill
        on delete cascade on update cascade,
    foreign key (project_name) references project
        on delete cascade on update cascade);
```

Generalization

- Translation of a generalization hierarchy composed of a root supertype and subtypes can result in more than one SQL tables
- A table derived from the root supertype contains
  - The key of the supertype entity, and
  - All additional attributes defined for the root supertype entity
- A table derived from the subtype entity contains
  - A key of the supertype
  - Only the attributes from the specific subtype entity

Generalization (2)

- Integrity after insert, update or delete operations is achieved by updating appropriate tables
  - In some cases more than one table has to be updated
  - Cascade behaviour must be used for the foreign keys
  - Updating the key of the root supertype requires the updates in the hierarchy of tables
  - Updating description attribute requires the change in one table only
- Transformation rules are the same for the case of intersecting as well as for the disjunctive subtypes

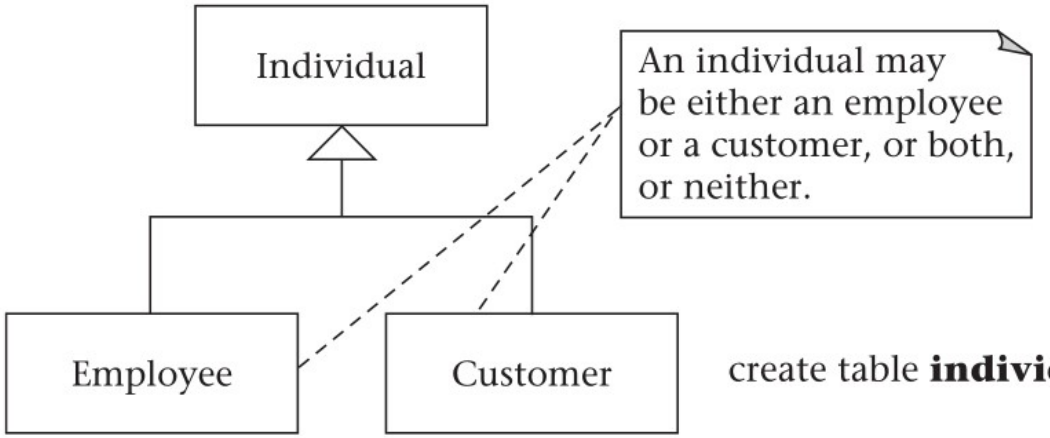
Generalization (3)

- The second approach is the definition of a single table that contains the attributes of a root entity as well as the subtype entities
  - Complete hierarchy is in one table
  - NULL values have to be used
- The third approach is the creation of one table for one subtype entity from the leafs of the hierarchy
  - Include all attributes from the entities on the path to the root entity
- There are advantages and disadvantages of each approach

Generalization (4)

- Design tools can implement all three cases
- Practical systems use classification attributes to distinguish among the subtype entities
  - If we have multi-level hierarchy then we use more classification attributes
  - The type hierarchy is implemented with the classification attributes
- The approaches can be combined

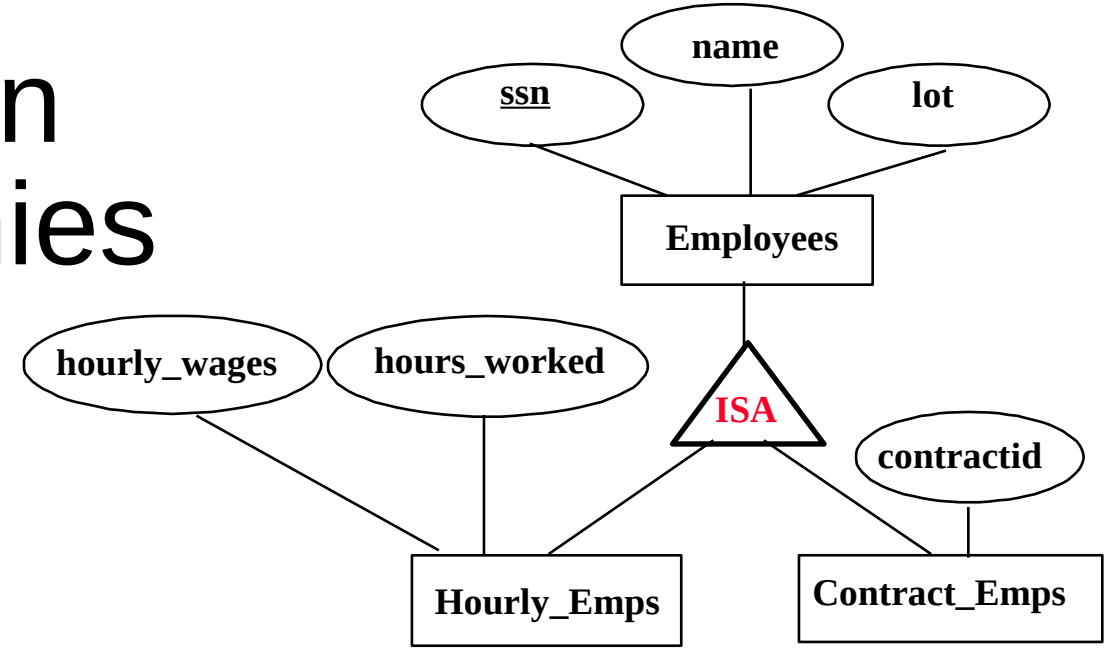
Example: Generalization hierarchy




```
create table individual (indiv_id char(10),
                         indiv_name char(20),
                         indiv_addr char(20),
                         primary key (indiv_id));
create table employee (emp_id char(10),
                       job_title char(15),
                       primary key (emp_id),
                       foreign key (emp_id) references individual
                           on delete cascade on update cascade);
create table customer (cust_no char(10),
                       cust_credit char(12),
                       primary key (cust_no),
                       foreign key (cust_no) references individual
                           on delete cascade on update cascade);
```

Generalization (ISA) Hierarchies

If we declare A ISA B, every A entity is also considered to be a B entity.



- *Overlap constraints*: Can Joe be an Hourly_Emps as well as a Contract_Emps entity? (*Allowed/disallowed*)
- *Covering constraints*: Does every Employees entity also have to be an Hourly_Emps or a Contract_Emps entity? *(Yes/no)*
- Reasons for using ISA:
  - To add descriptive attributes specific to a subclass.
  - To identify entitities that participate in a relationship.

Translating ISA hierarchy

First approach

- 3 tables: Employes, Hourly_Emps, Contract_Emps
  - ON UPDATE|DELETE CASCADE
  - Queries that access attributes of all employes are simple
  - Queries on specific employes are more complex

Third approach:

- *2 tables: Hourly_Emps, Contract_Emps*
  - Updates are allways in a single table