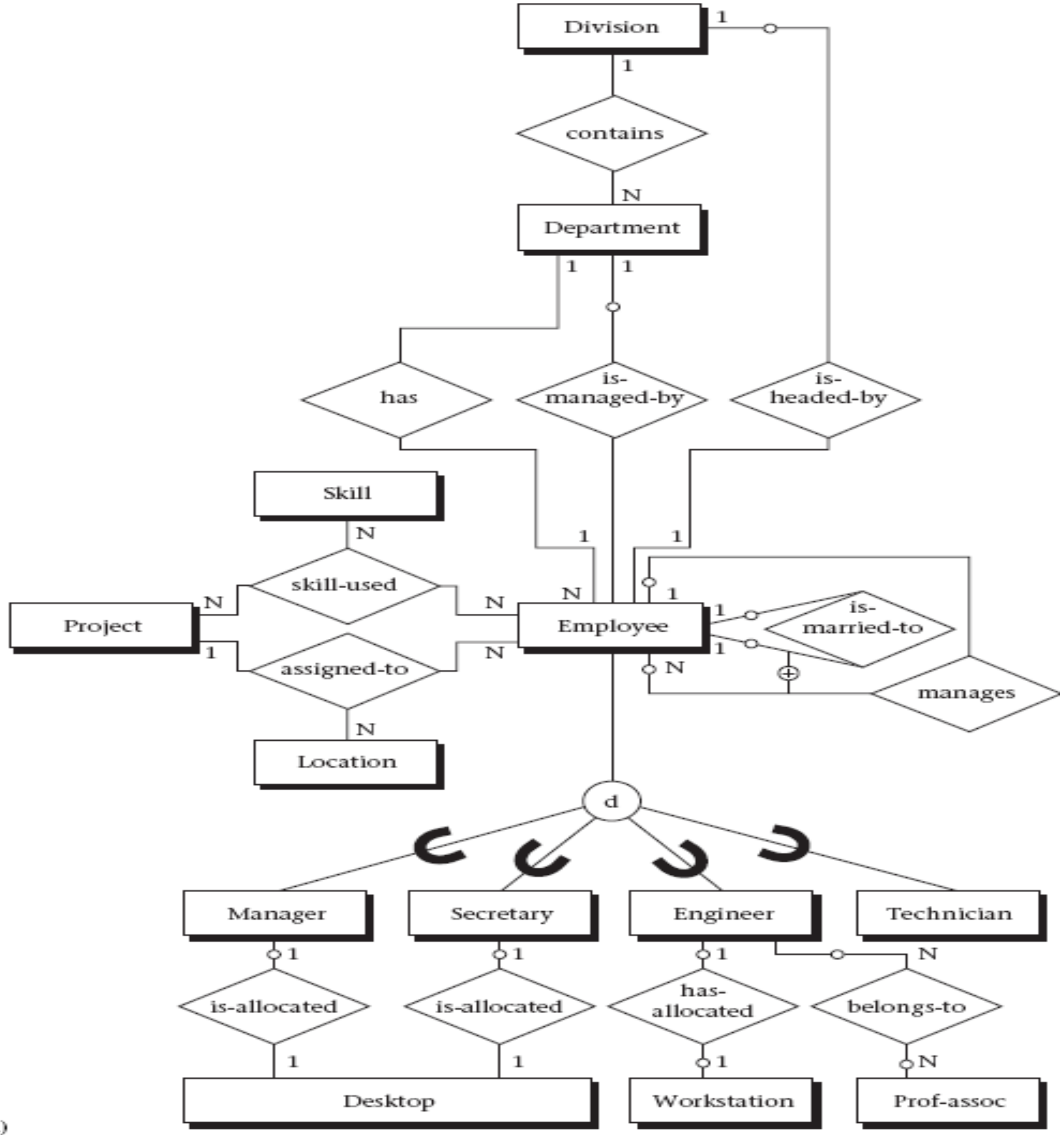
5. Transforming the Conceptual Data Model to SQL

Natural evolution from the ER to a relational schema

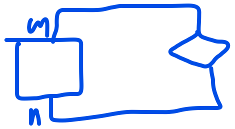
- Conceptual data modeling is an effective early step in relational database development
- Widespread commercialization and use of software design tools that support not only conceptual data modeling but also the automatic conversion of these models to vendor-specific SQL table definitions and integrity constraints
- In this chapter we assume the applications to be Online Transaction Processing (OLTP)

Example


Division
1
1
contains
N
Department
1
1
has
is-managed-by
is-headed-by
Skill
N
skill-used
Project
N
1
N
N
Employee
1
1
1
1
1
N
N
is-married-to
manages
assigned-to
N
Location
d
Manager
Secretary
Engineer
Technician
1
1
1
N
is-allocated
is-allocated
has-allocated
belongs-to
1
1
1
N
Desktop
Workstation
Prof-assoc
i)

SQL table with the same information content as the original entity from which it is derived

- Entities with
  - many-to-many
  - one-to-many on the “one” (parent) side
  - one-to-one on either side
  - recursive relationships that are many-to-many
  - ternary or higher-degree relationship
  - generalization hierarchy



SQL table with the embedded foreign key of the parent entity

- Entities with
  - one-to-many for the entity on the "many" (child) side
  - one-to-one relationships for one of the entities
  - recursive relationship that is one-to-one or one-to-many
- Prompting the user to define a foreign key in the child table that matches a primary key in the parent table

SQL table derived from a relationship, containing the foreign keys of all the entities in the relationship

F.K + F.K → P.K 새로운table

- binary and many-to-many
- recursive and many-to-many
- ternary or higher degree
- A many-to-many relationship can only be defined in terms of a table that contains foreign keys that match the primary keys of the two associated entities
- This new table may also contain attributes of the original relationship
    - Example: A relationship “enrolled-in” between two entities Student and Course might have the attributes “term” and “grade”

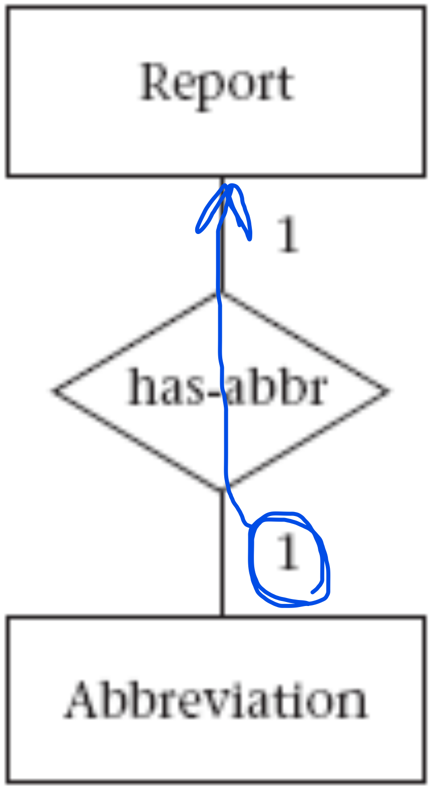
Rules to apply to handling Nulls

default

- Nulls are allowed in an SQL table for foreign keys of associated (referenced) optional entities
- Nulls are not allowed in an SQL table for foreign keys of associated (referenced) mandatory entities
- Nulls are not allowed for any key in an SQL table derived from a many-to-many relationship
  - Because only complete row entries are meaningful in the table

Ternary → Nulls are not allowed

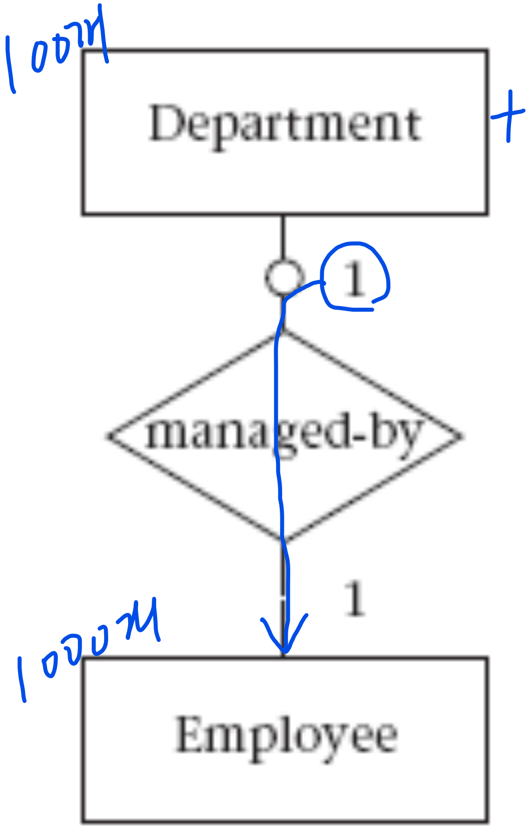
One-to-one, both entities mandatory



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

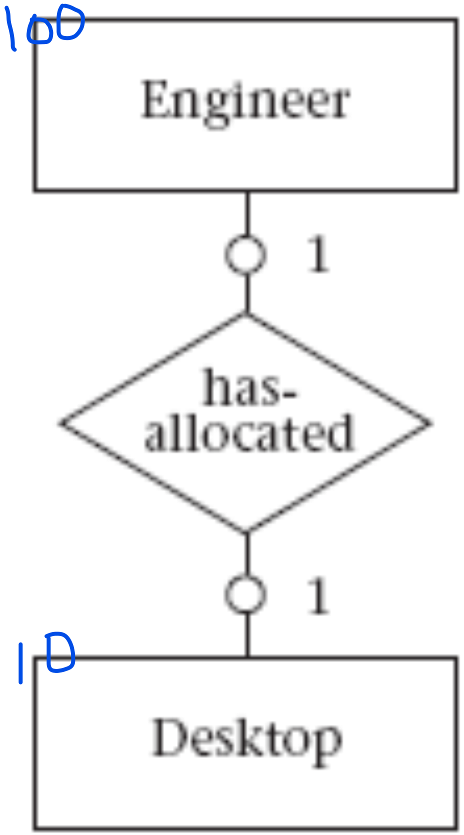
create table **abbreviation** 요약
(abbr_no char(6), mand 1:1
report_no integer not null unique,
primary key (abbr_no),
foreign key (report_no) references report
on delete cascade ~~on update cascade~~);
참조무결성 필수 함께 P.K 바뀌는경우X

One-to-one, one entity optional, one entity mandatory



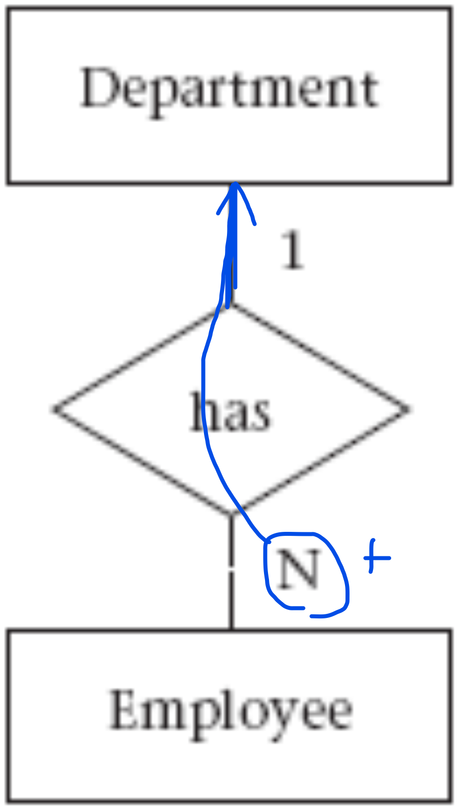
```
create table department
        (dept_no integer,
        dept_name char(20),  M      1:1
        mgr_id char(10) not null unique,
        primary key (dept_no),
        foreign key (mgr_id) references employee(emp_id)
         on delete set default ~~on update cascade~~);
create table employee
        (emp_id char(10),
        emp_name char(20),
        primary key (emp_id));
```

One-to-one, both entities optional



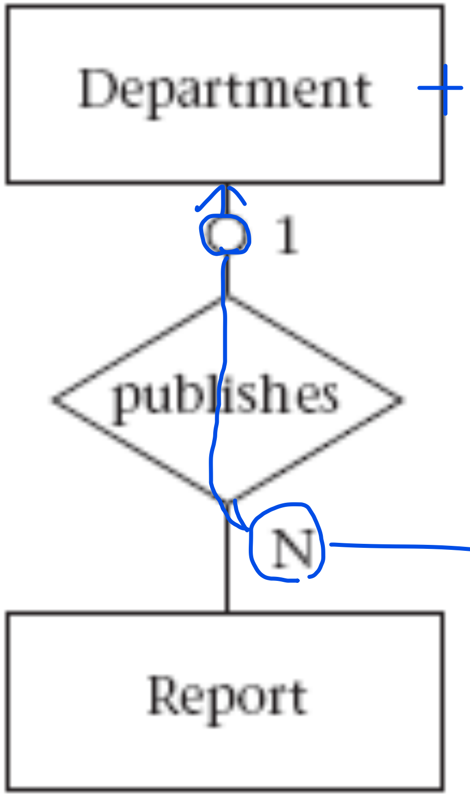
```
create table engineer
        (emp_id char(10),
        desktop_no integer,
        primary key (emp_id));
create table desktop
        (desktop_no integer, op
        emp_id char(10),
        primary key (desktop_no),
        foreign key (emp_id) references engineer
        on delete set null ~~on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

One-to-many, both entities mandatory



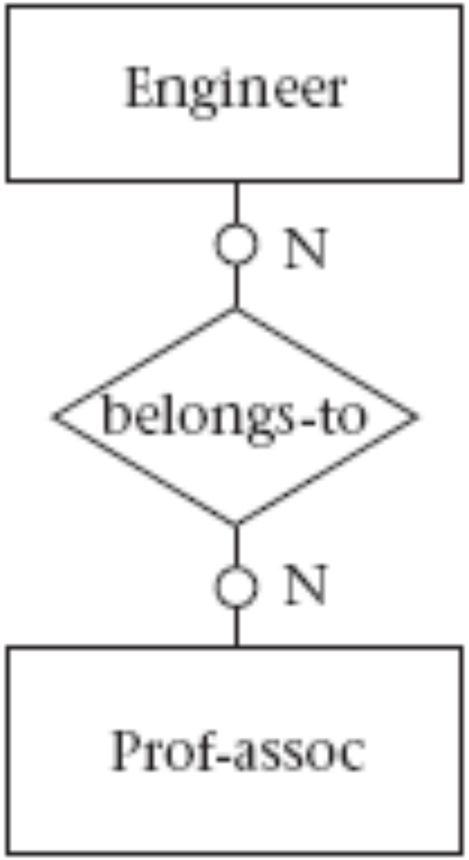
create table **department**
(dept_no integer,
dept_name char(20),
primary key (dept_no));
create table **employee**
(emp_id char(10),
emp_name char(20),
dept_no integer not null,
primary key (emp_id),
foreign key (dept_no) references **department**
on delete set default ~~on update cascade~~);

One-to-many, one entity optional, one entity mandatory



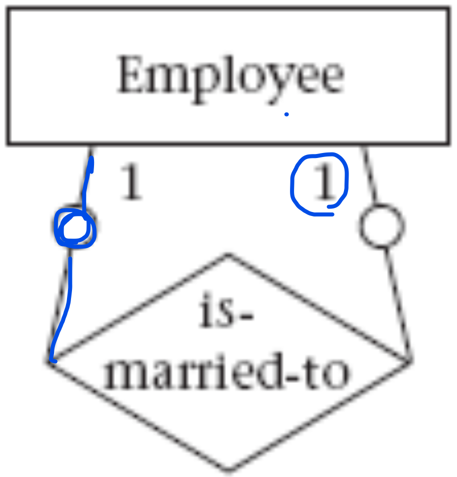
```
create table department
            (dept_no integer,
            dept_name char(20),
            primary key (dept_no));
create table report
            (report_no integer,
            dept_no integer,      op
            primary key (report_no),
            foreign key (dept_no) references department
             on delete set null ~~on update cascade~~);
```

many One-to-many, one entity optional, one entity mandatory



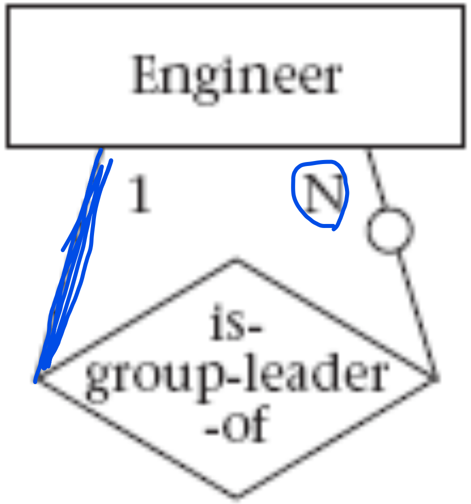
```
create table engineer
          (emp_id char(10),
          primary key (emp_id));
create table prof_assoc
          (assoc_name varchar(256),
          primary key (assoc_name));
create table belongs_to
          (emp_id char(10), +
          assoc_name varchar(256),
          primary key (emp_id,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

One-to-one, both sides optional



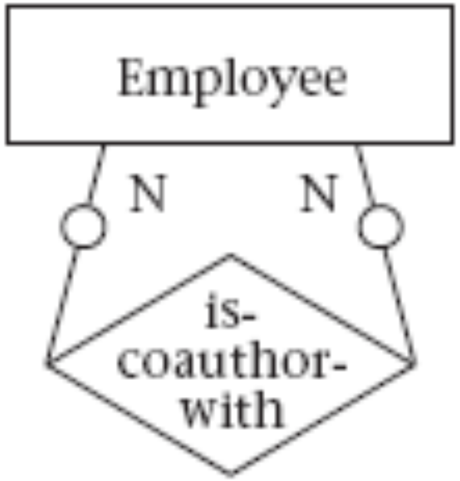
```
create table employee
          (emp_id char(10),
          emp_name char(20),
          spouse_id char(10),
          primary key (emp_id),
          foreign key (spouse_id) references employee (emp_id)
            on delete set null on update cascade);
```

One-to-many, one side mandatory, many side optional



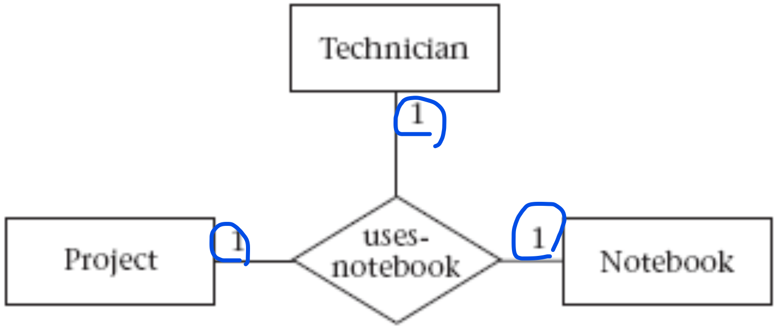
```
create table engineer
          (emp_id char(10),
          leader_id char(10) not null,
          primary key (emp_id),
          foreign key (leader_id) references engineer
            on delete set default on update cascade);
```

Many-to-many, both sides 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);
```

1-1-1



Functional dependencies 후보키

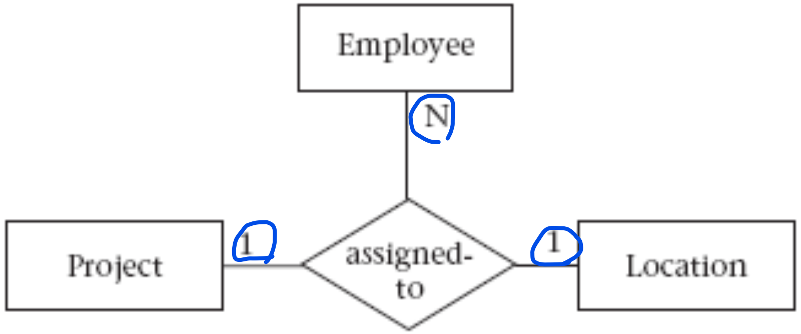
P.K — emp_id, project_name → notebook_no

대체키 emp_id, notebook_no → project_name

project_name, notebook_no → emp_id

```
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), ①
          project_name char(20),①      +
          notebook_no integer not null, ①
          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-N



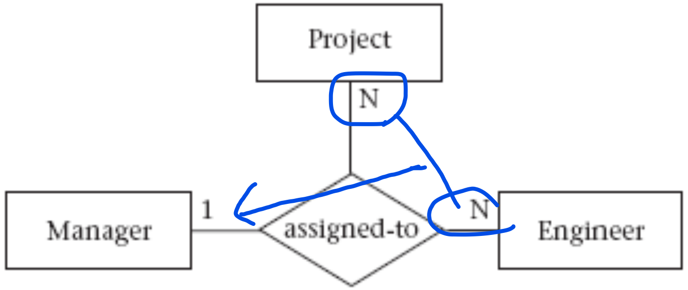
Functional dependencies

대체 – emp_id, loc_name → project_name

P.K – emp_id, project_name → loc_name

```
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), 1
        project_name char(20), N  +
        loc_name char(15) not null, 1
        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-N-N

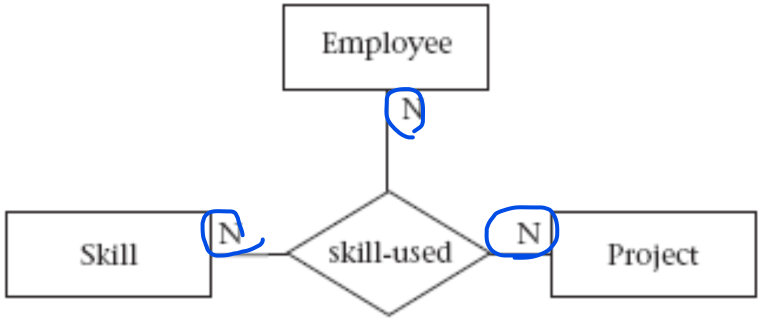


Functional dependency

project_name, emp_id → mgr_id

```
create table project (project_name char(20),
            primary key (project_name));
create table manager (mgr_id char(10),
            primary key (mgr_id));
create table engineer (emp_id char(10),
            primary key (emp_id));
create table manages (project_name char(20), N
            mgr_id char(10) not null, 1          +
            emp_id char(10), N
            primary key (project_name, emp_id),
            foreign key (project_name) references project
              on delete cascade on update cascade,
            foreign key (mgr_id) references manager
              on delete cascade on update cascade,
            foreign key (emp_id) references engineer
              on delete cascade on update cascade);
```

N-N-N

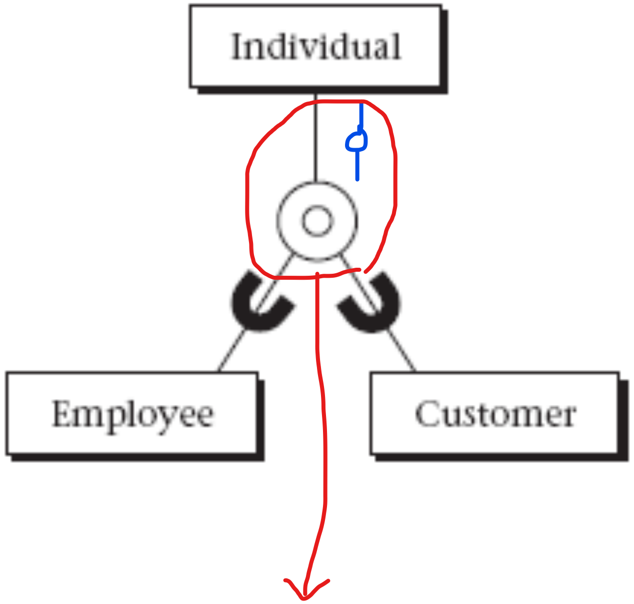


```
create table employee (emp_id char(10),
            emp_name char(20), primary key (emp_id));
create table skill (skill_type char(15),
            primary key (skill_type));
create table project (project_name char(20),
            primary key (project_name));
create table skill_used (emp_id char(10),N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

Functional dependencies

None

Generalization (is-A)



SQL표현X

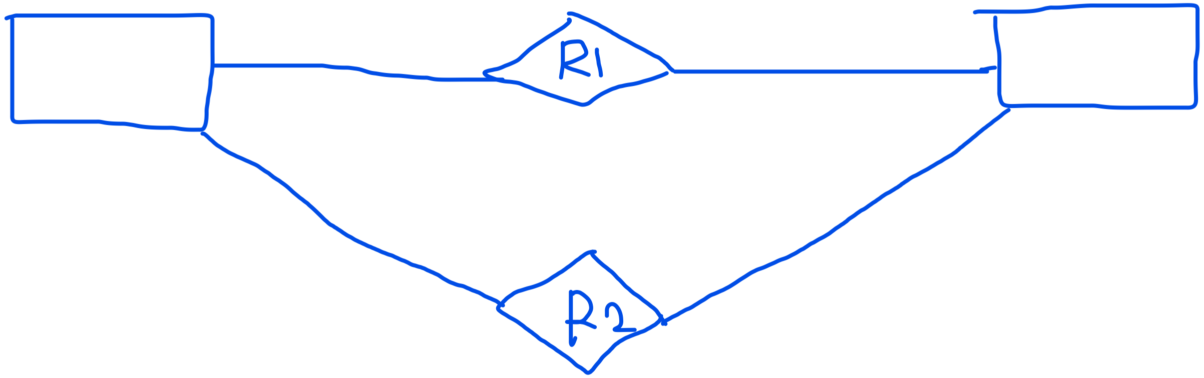
Document로 표현

```
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);
```

Multiple Relationships

- Multiple relationships are always considered to be completely independent

R1, R2 → 독립적으로 생각



R1-1:N → F.K

R2-N:1 → F.K

R1-1:N → F.K

R2-N:M → table추가

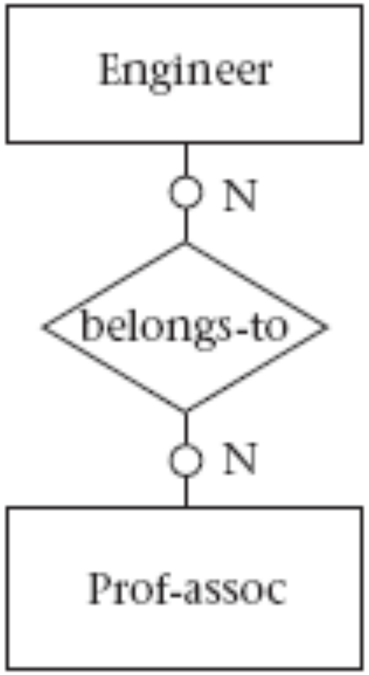
Weak Entities

- Weak entities differ from entities only in their need for keys from other entities to establish their uniqueness

Entity Transformation

- One-to-many relationship between two entities
  - Add the key of the entity on the "one" side (the parent) into the child table as a foreign key
- One-to-one relationship between one entity and another entity
  - Add the key of one of the entities into the table for the other entity and change it to a foreign key
  - Strategy
    - To maintain the most natural parent-child relationship
    - Based on efficiency: add the foreign key to the table with fewer rows
- Generalization hierarchy
  - Every entity in a generalization hierarchy is transformed into a table
  - Each of these tables contains the key of the supertype entity
    - Subtype primary keys are foreign keys as well
- SQL constructs include constraints for not null, unique, and foreign key

Many-to-Many Binary Relationship Transformation

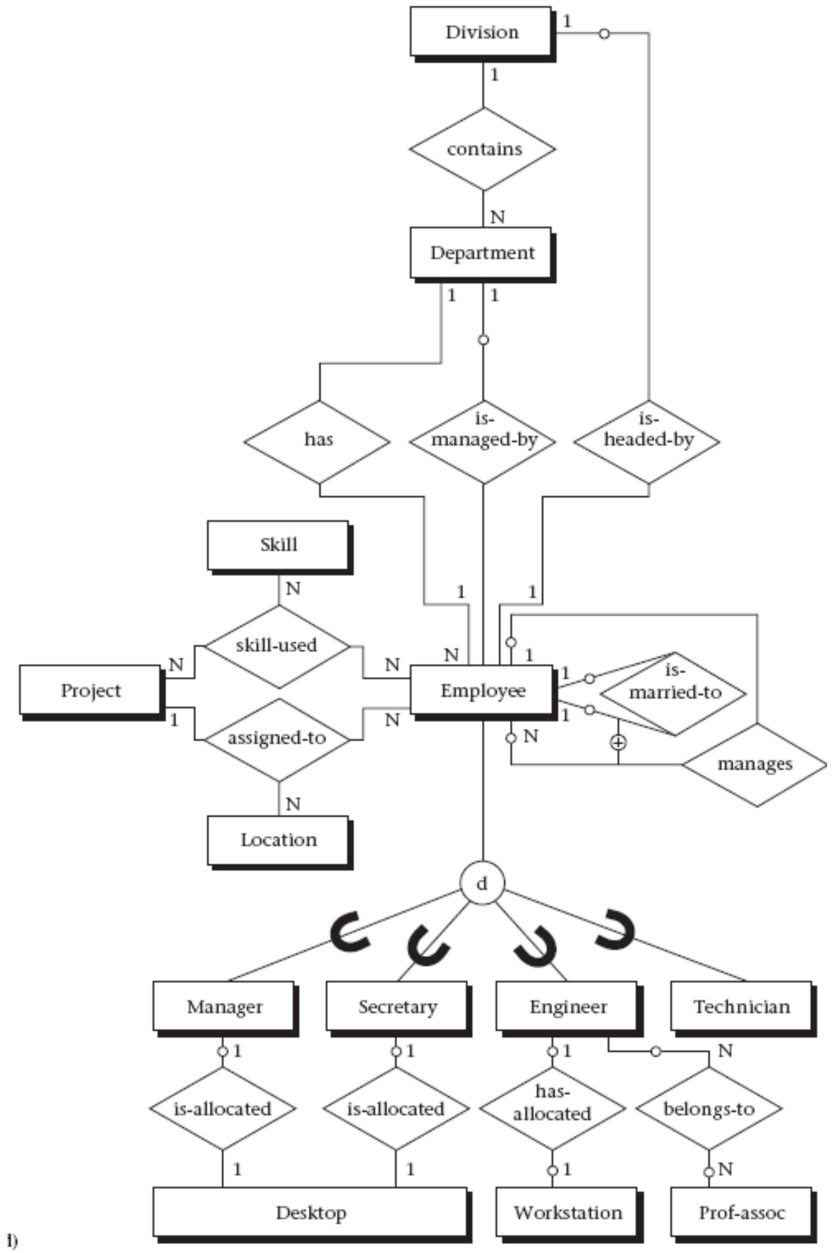


- Every many-to-many binary relationship is transformed into a table containing the keys of the entities and the attributes of the relationship
- SQL constructs for this transformation may include constraints for not null.
- The constraints for primary key and foreign key are required, because a table is defined as containing a composite of the primary keys of the associated entities

Ternary Relationship Transformation

- Every ternary relationship is transformed into a table with 3 foreign keys
- Ternary relationships are defined as a collection of the *3* primary keys in the associated entities in that relationship
  - With possibly some nonkey attributes
  - SQL constructs for this transformation must include constraints for not null, since optionality is not allowed
- The unique clause must also be used to define alternate keys that often occur with ternary relationships

ER to SQL Transformation



- SQL tables derived directly from entities
  - division, secretary, project
  - department, engineer, location
  - employee, technician, prof_assoc
  - manager, skill, desktop, workstation
- SQL tables derived from many-to-many
  - belongs_to
- SQL tables transformed from ternary relationships
  - skill_used, assigned_to