Relational Schema Design

Using ER Methodology to Design Relational Database Schemas

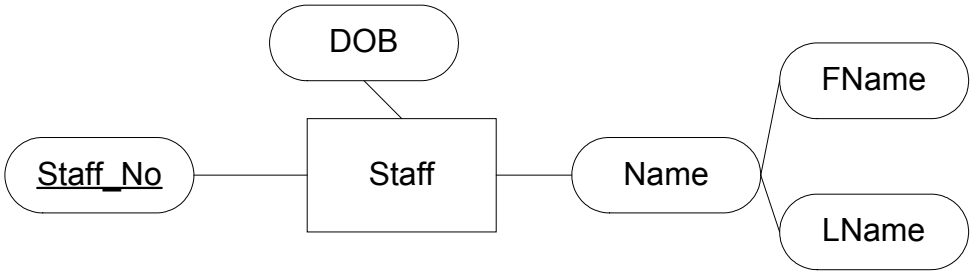
The Development Process

- Collect requirements. Analyze the requirements.
- Conceptually design the data (e.g., draw an ER diagram).
- Logically design the data (e.g., choose relation names and schemas).

ER Diagrams → Relations

Rule 1

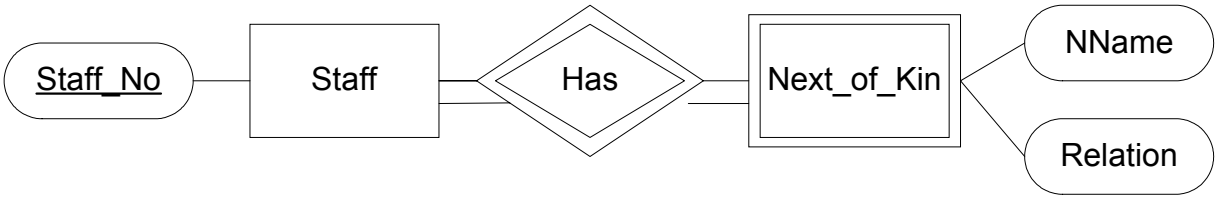
- A ***strong entity E*** becomes a relation.
- Create relation R that includes all simple attributes of E and simple component attributes of composite attributes of E.
- The key of the entity becomes the primary key of the relation schema.



Staff (Staff_No, DOB, Fname, LName)

Rule 2

- A **weak entity W** with owner entity type E becomes a relation R.
- The relation schema contains the entity's properties and the key of the entity E on which the weak entity depends.
  - Create relation R including all simple attributes and simple component attributes of W.
  - The *full* key of the entity becomes the primary key of the schema.
- The "weak" relationship "disappears," If this relationship has any properties, they become attributes of the "weak" relation's schema.

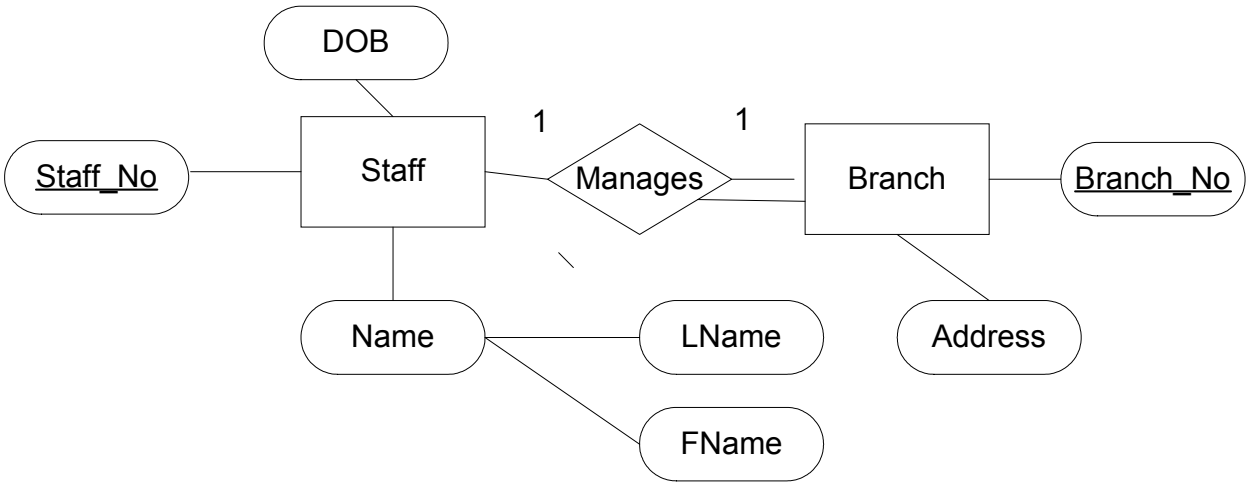


Next_of_Kin (Staff_No, Nname, Relation)

Rule 3

For each **binary 1:1** relationship type R

- identify relations corresponding to participating entities, say T and S
- if there is total participation, make that S
- include primary key of T as a foreign key in S
- include all simple attributes and simple components of composites of R as attributes of S



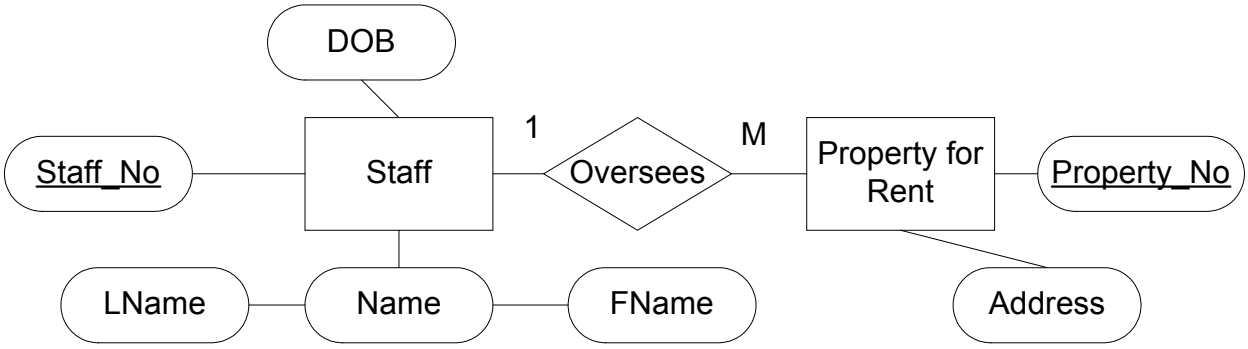
Branch_Managed (Branch_No, Address, Staff_No)
Staff (Staff_No, DOB, Fname, LName)

Rule 4

A **many-to-one** relationship is "absorbed" by the relation corresponding to the entity at the "many" side. This is done by adding to the schema of the relation corresponding to the "many" side the key of the relation corresponding to the "one" side. If the relationship has any properties, these are also added to the schema of the relation corresponding to the "many" side. If the many-to-one relationship connects a weak entity class with the entity class it depends on, this rule has no effect beyond the effect of Rule 2.

For each 1:N binary relationship type R

- identify relations, say S and T, corresponding to participating entity types
- if S is at "N side" of relationship type then include primary key of T in S
- include any simple attributes or simple components of R as attributes of S



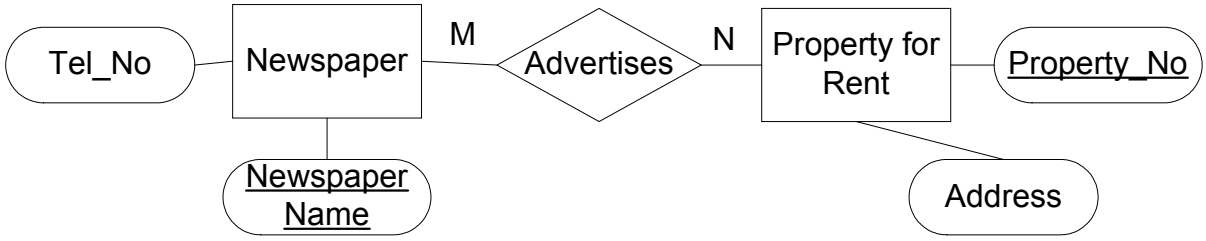
Overseed_property (Property_No, Address, Staff_No)
Staff (Staff_No, DOB, Fname, LName)

Rule 5

A **many-to-many** relationship becomes a relation. The relation schema contains the keys of the entities related by the relationship, plus any properties of the relationship. The union of the keys of the related entities becomes the primary key of the relation schema.

For each binary N:M relationship type K

- create new relation R to represent K
- if S and T are the relations corresponding to the participating entity types then include their primary keys as foreign keys in R
- combination of foreign keys will be primary key of R
- include any simple attributes or simple components of K as attributes of R

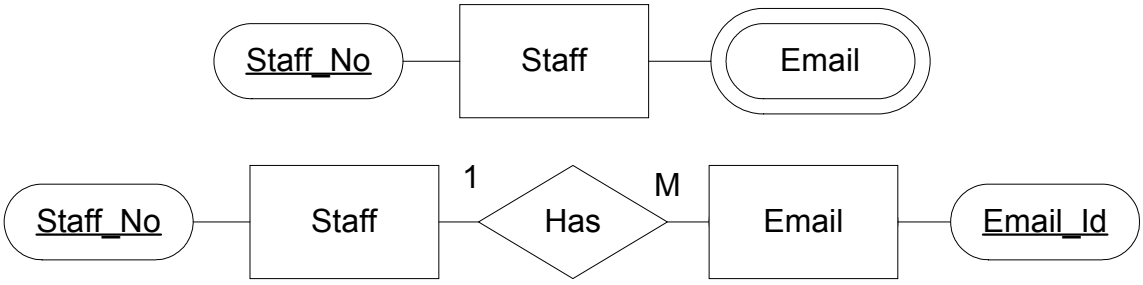


NAP (Newspaper Name, Property_No**)**
Property for Rent (Property_No, Address)
Newspaper (Newspaper Name**,** Tel_No**)**

Rule 6

For each **multi-valued attribute A**

- Create a relation R that contains the attribute A and the primary key of the relation S that corresponds to the entity type that contained A.



Email (Email-Id, Staff_No)
Staff (Staff_No, DOB, Fname, LName)

Rule 7

For each **n-ary relationship** type L (n>2)

- create a new relation R to represent L
- include as foreign keys in R the primary keys of all relations that correspond to the entity types participating in L
- the primary key of R is the combination of all these foreign keys
- include any simple attributes or simple components of L as attributes of R

Rule 8

For each specialization with subclasses **{S_1 , S_2 , …, S_m }** and superclass **C** where attributes of **C** are **{k, a_1 , a_2 , …, a_n }** and **k** is the primary key conversation can be done using one of 4 options:

Option 1

•create relation **L** for **C** where

Attr(L) = {k, a_1 , a_2 , …, a_n} and

PK(L) = k

•create relation **L_i** for each **S_i** where **Attr(L_i) = {k}** U **{attributes of S_i}** and **PK(L_i) = k**

Option 2

•create relation **L_i** for each **S_i** where **Attr(L_i) = {k, a_1 , a_2 , …, a_n}** U **{attributes of S_i}** and **PK(L_i) = k**

Option 3

•for cases where subclasses are disjoint

•create relation **L** where **Attrs(L) = {k, a_1 , a_2 , …, a_n}** U **{attributes of S_1}** U **{attributes of S_2}** U **…** U **{attributes of S_m}** U **{t}** and

PK(L) = k, where **t** is a type (or discriminating) attribute

Option 4

•for cases where subclasses are overlapping

•create relation **L** where **Attrs(L) = {k, a_1 , a_2 , …, a_n}** U **{attributes of S_1}** U **{attributes of S_2}** U **…** U **{attributes of S_m}** U **{t_1, t_2, …, t_m}** and **PK(L)** = k, where **t_i** is a boolean attribute indicating whether a tuple belongs to S_i