Advanced Management of Data

Conceptual Database Design - The Enhanced ER Model

Logical Database Design using the Relational Data Model

Phases of Database Design

Database design is made up of three main phases:

Conceptual Design

The process of constructing a model of the data used in an enterprise, independent of **all** physical considerations.

Logical Design

The process of constructing a model of the data used in an enterprise based on a specific data model, but independent of a particular DBMS and other physical considerations.

Physical Design

The process of producing a description of the implementation of the database on secondary storage; it describes the base relations, file organizations, and indexes used to achieve efficient access to the data, and any associated integrity constraints and security measures.

Conceptual Design (2)
- The Enhanced ER Model -

Specialization / Generalization

Aggregation

Composition

The Enhanced ER Model (1)

Superclass

An entity type that includes one or more distinct subgroupings of its occurrences, which must be represented in a data model.

Subclass

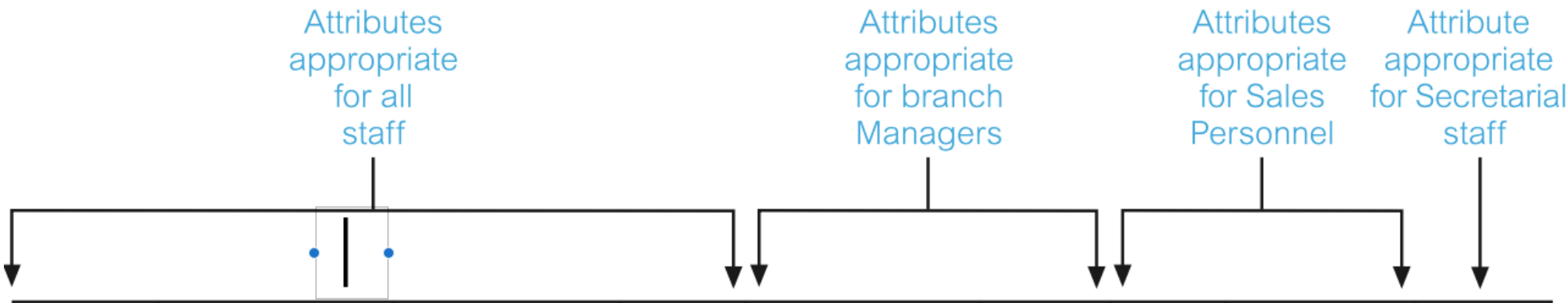
A distinct subgrouping of occurrences of an entity type, which must be represented in a data model.

Superclass / Subclass Relationships

Each member of a subclass is also a member of the superclass, i.e. the entity in the subclass is the same entity in the superclass, but has a distinct role.

The relationship between a superclass and a subclass is one-to-one (1:1) and is called a superclass / subclass relationship

The Enhanced ER Model (2)



staffNo	name	position	salary	mgrStartDate	bonus	sales Area	car Allowance	typing Speed
SL21	John White	Manager	30000	01/02/95	2000			
SG37	Ann Beech	Assistant	12000					
SG66	Mary Martinez	Sales Manager	27000			SA1A	5000	
SA9	Mary Howe	Assistant	9000					
SL89	Stuart Stern	Secretary	8500					100
SL31	Robert Chin	Snr Sales Asst	17000			SA2B	3700	
SG5	Susan Brand	Manager	24000	01/06/91	2350			

[Connolly & Begg]

The Enhanced ER Model (3)

Attribute Inheritance

Since an entity of a subclass represents the same object as in the superclass, it possesses all attributes of the superclass ➔ a subclass inherits all attributes of its superclass automatically.

Additionally, each subclass may possess subclass specific attributes.

Type Hierarchy

A subclass is an entity in its own and may also have one or more subclasses.

The entirety of an entity and all its subclasses (or superclasses) is called type hierarchy.

Multiple Inheritance

A subclass with more than one superclass is called a shared subclass.

The Enhanced ER Model (4)

Specialization

The process of maximizing the differences between members of an entity by identifying their distinguishing characteristics.

Generalization

The process of minimizing the differences between entities by identifying their common characteristics.

Diagrammatic representation of specialization / generalization

The subclasses are attached by lines to a triangle that points toward the superclass.

The label below the specialization / generalization triangle describes the constraints on the relationship between the superclass and its subclasses.

Entity types that represent subclasses must contain only subclass-specific attributes.

The Enhanced ER Model (5)

Participation Constraint

Determines whether every member in the superclass must participate as a member of a subclass.

Disjoint Constraint

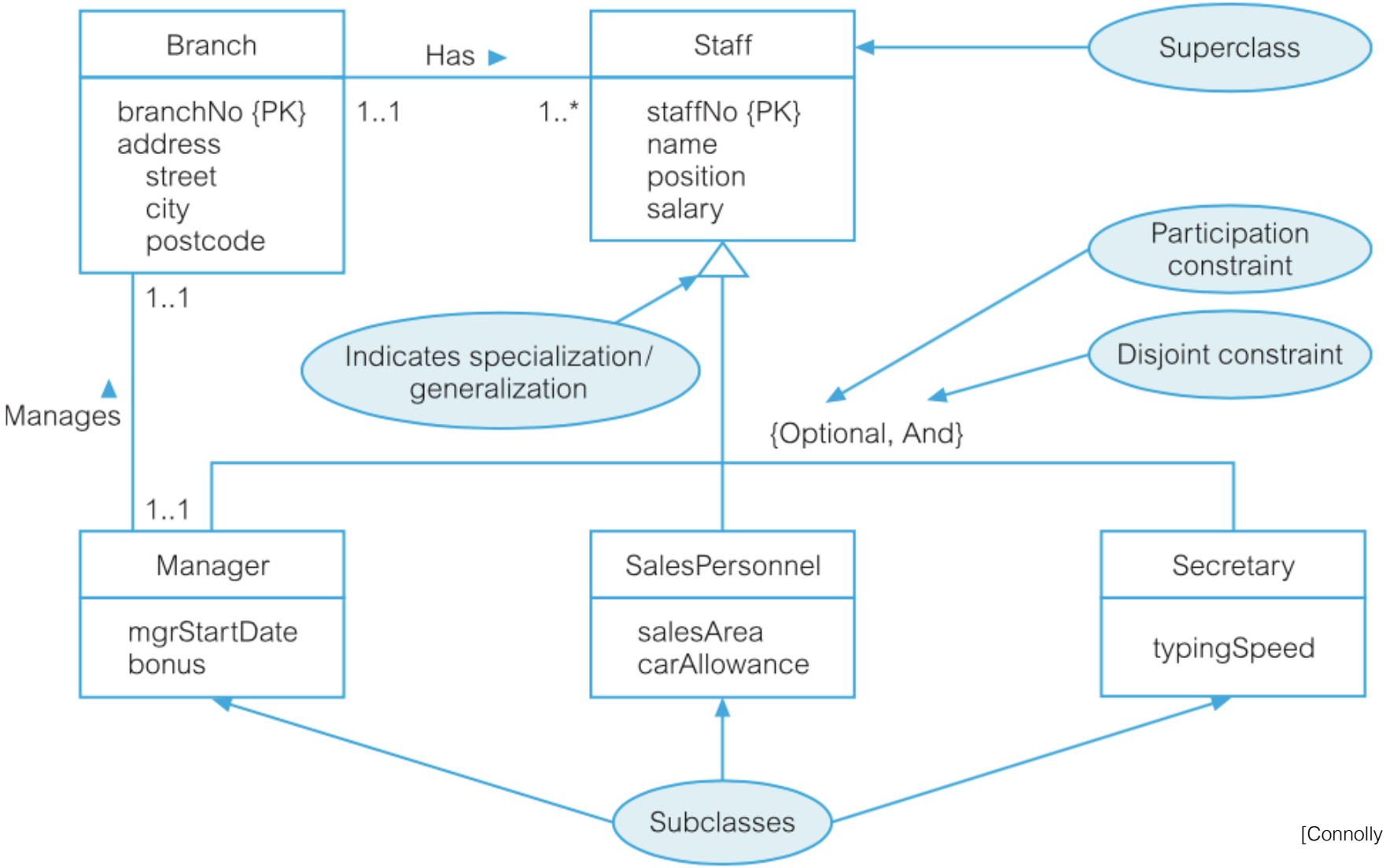
Describes the relationship between members of the subclasses and indicates whether it is possible for a member of a superclass to be a member of one, or more than one, subclass.

Categories

The disjoint and participation constraints of specialization and generalization are distinct:

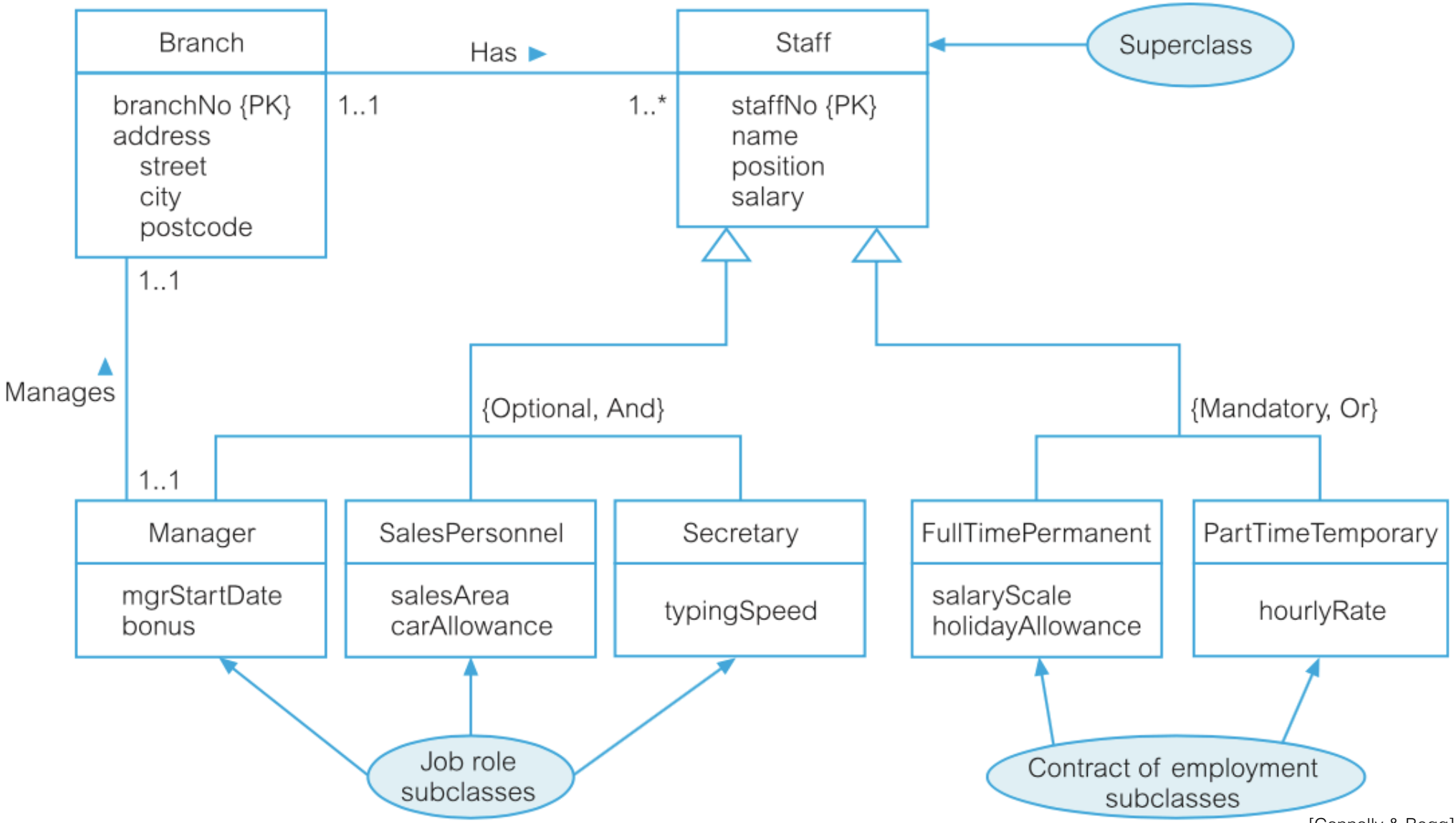
- mandatory and disjoint {Mandatory, Or}
- optional and disjoint {Optional, Or}
- mandatory and nondisjoint {Mandatory, And}
- optional and nondisjoint {Optional, And}

Specialization / Generalization



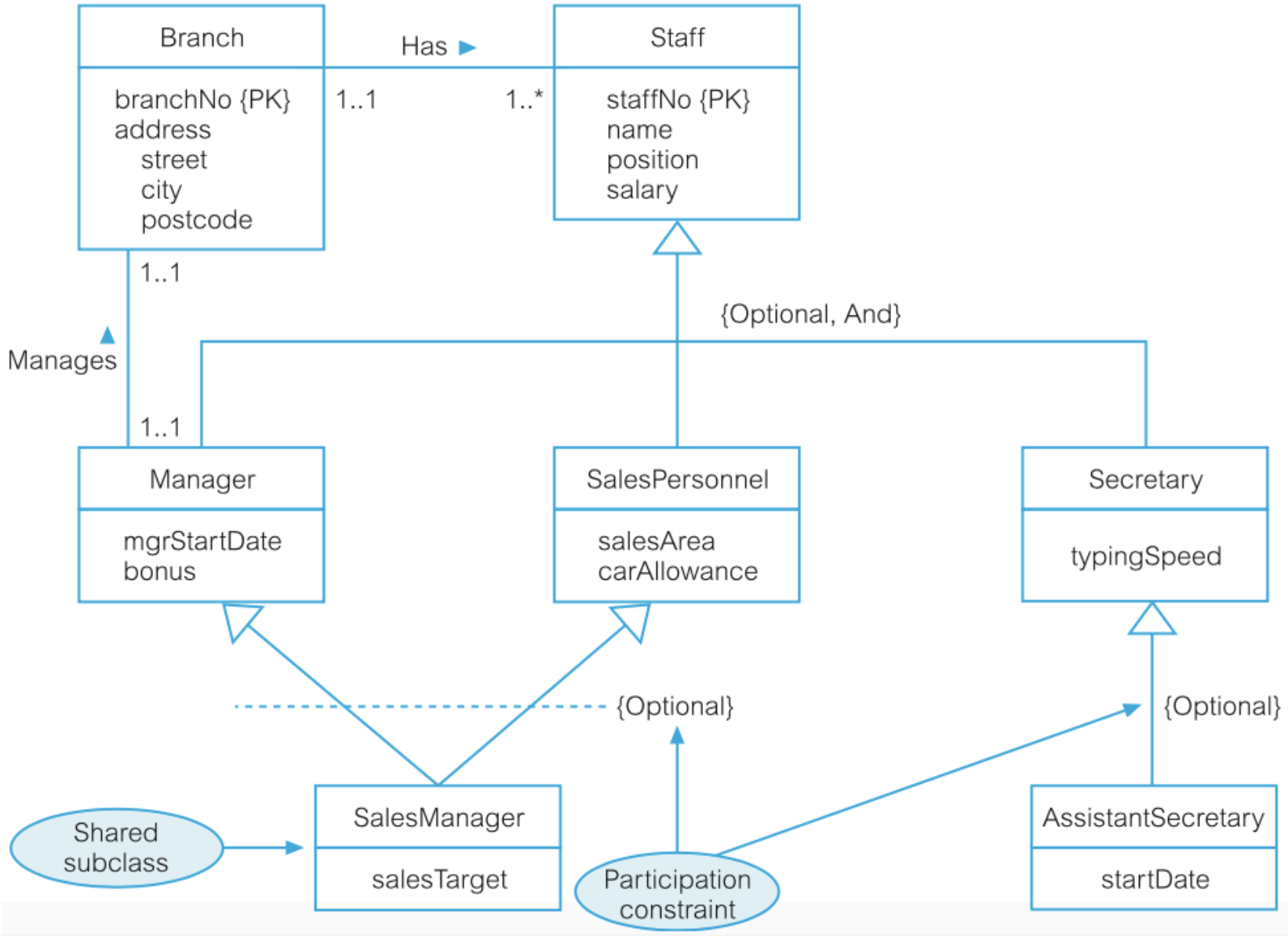
[Connolly & Begg]

Specialization / Generalization



[Connolly & Begg]

Specialization / Generalization



[Connolly & Begg]

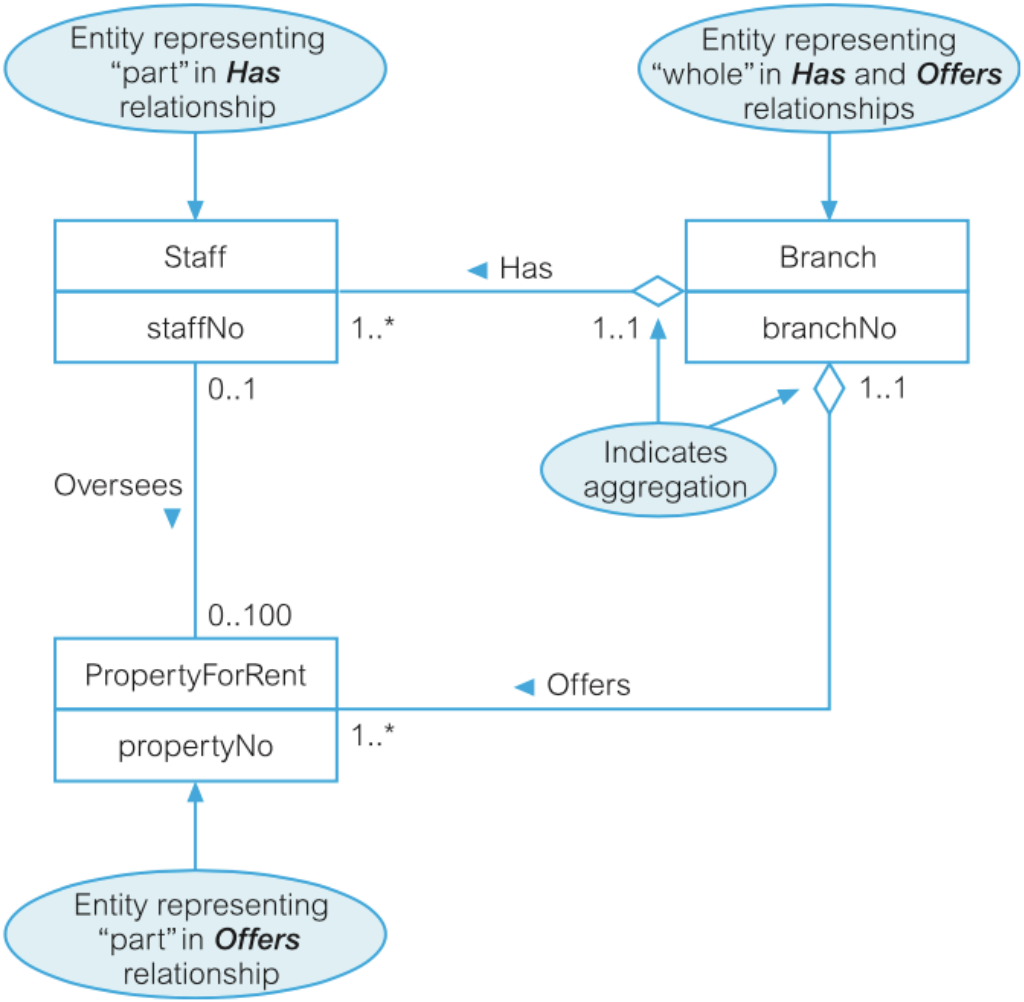
The Enhanced ER Model (6)

Aggregation

Represents a “has-a” or “is-part-of” relationship between entity types, where one represents the “whole” and the other the “part.”

Diagrammatic representation

UML represents aggregation by placing an open diamond shape at one end of the relationship line, next to the entity that represents the “whole.”



[Connolly & Begg]

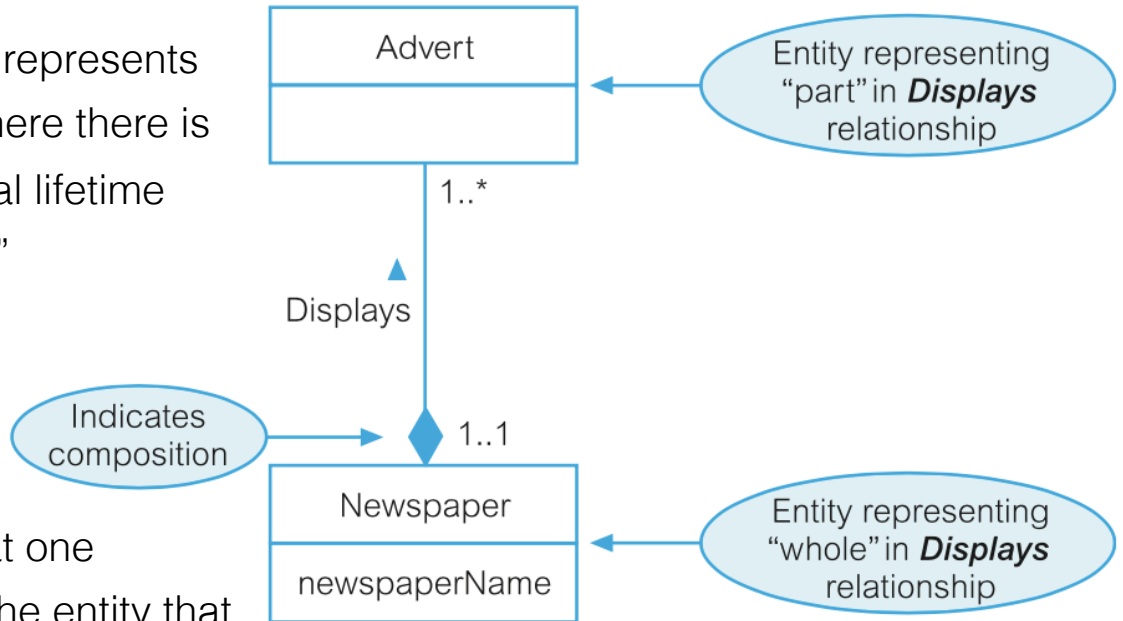
The Enhanced ER Model (7)

Composition

A specific form of aggregation that represents an association between entities, where there is a strong ownership and coincidental lifetime between the “whole” and the “part.”

Diagrammatic representation

UML represents composition by placing a filled-in diamond shape at one end of the relationship line next to the entity that represents the “whole” in the relationship.



[Connolly & Begg]

Phases of Database Design

Database design is made up of three main phases:

Conceptual Design

The process of constructing a model of the data used in an enterprise, independent of **all** physical considerations.

Logical Design

The process of constructing a model of the data used in an enterprise based on a specific data model, but independent of a particular DBMS and other physical considerations.

Physical Design

The process of producing a description of the implementation of the database on secondary storage; it describes the base relations, file organizations, and indexes used to achieve efficient access to the data, and any associated integrity constraints and security measures.

Logical Database Design

Deriving relations from the ER model

Logical Database Design

To build the logical data model we have to map the concepts of the conceptual model onto the concepts of the logical data model (which will be the relational model in most cases).

Since the relational model knows only one concept - the relation - the mapping process requires to derive relations from the conceptual data model.

Note

- The power of expression of the relational model is much less than the conceptual model.
- Strong and weak entity types, binary and complex relationship types, multiplicity, multi-valued attributes, super- and subclasses, all have to be mapped onto relations (only).
- Additionally, attributes of different entity or relationship types may be mixed in one relation.
- Hence, the meaning of a relation may not be clear without the knowledge of the ER model.
- Therefore, we create a conceptual model first, before deriving a logical data model.

Deriving Relations

We describe the composition of each relation using a Database Definition Language for relational databases.

1. We specify the name of the relation

2. We add a list of the relation's simple attributes

3. We identify the primary key and any alternate and/or foreign key(s) of the relation.

   The relationship that an entity has with another entity is represented by the primary key/ foreign key mechanism.
   In deciding where to place the foreign key attribute(s), we must first identify the "parent" and "child" entities involved in the relationship.
   The parent entity refers to the entity that posts a copy of its primary key into the relation that represents the child entity, to act as the foreign key.

4. The relation containing the referenced primary key is given.

5. Any derived attributes are also listed, along with how each one is calculated.

Deriving Relations

We have to describe how relations are derived for the following structures that may occur in a conceptual data model:

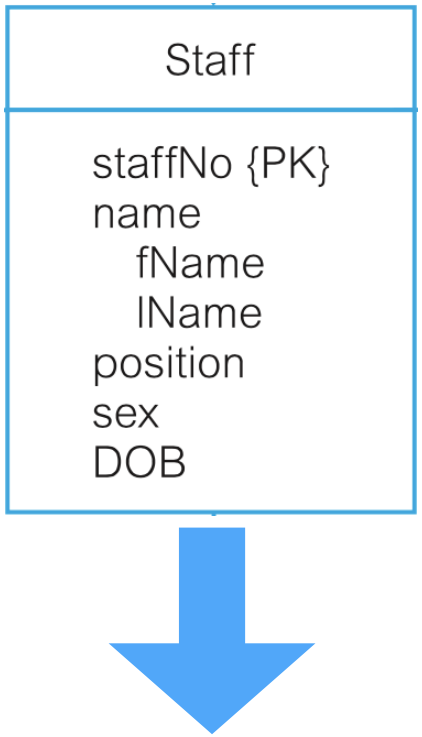
- strong entity types
- weak entity types
- one-to-many (1:*) binary relationship types
- one-to-one (1:1) binary relationship types
- one-to-one (1:1) recursive relationship types
- many-to-many (*:*) binary relationship types
- complex relationship types
- multi-valued attributes
- Superclass / Subclass relationship types

Deriving Relations

Strong Entity Types

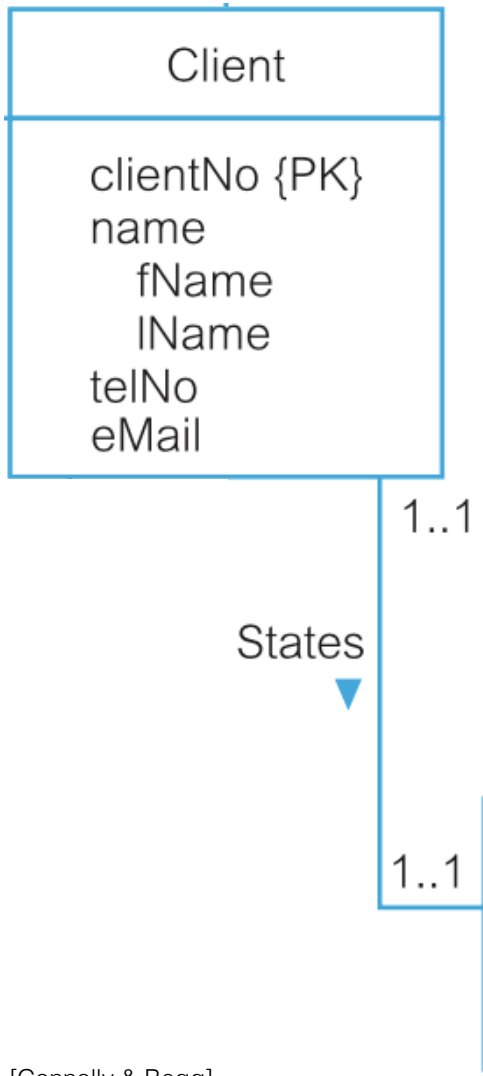
- For each strong entity in the data model, create a relation that includes all the simple attributes of that entity.
- For composite attributes, include only the constituent simple attributes.
- Mark the primary key, which should correspond to the primary key of the entity type.

  Within a relation primary key attributes can be easily visualized by underlining.



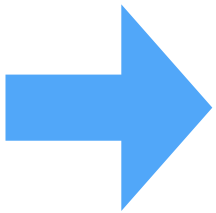
Staff					
staffNo	fName	lName	position	sex	DOB

Deriving Relations



Weak Entity Types

- For each weak entity in the data model, create a relation that includes all the simple attributes of that entity.
- For composite attributes, include only the constituent simple attributes
- The primary key of a weak entity is partially or fully derived from each owner entity and so the identification of the primary key of a weak entity cannot be made until after all the relationships with the owner entities have been mapped.



Preference	
prefType	maxRent

[Connolly & Begg]

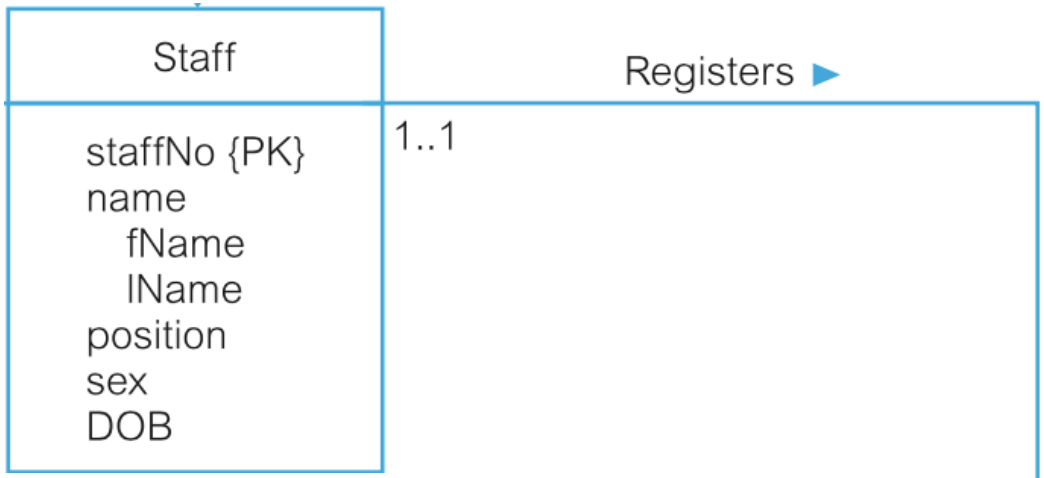
Deriving Relations

One-to-Many (1:*) Binary Relationship Types

- For each 1:* binary relationship, the entity on the “one side” of the relationship is designated as the parent entity and the entity on the “many side” is designated as the child entity.
- To represent this relationship, we post a copy of the primary key attribute(s) of the parent entity into the relation representing the child entity, to act as a foreign key.
- In the case where a 1:* relationship has one or more attributes, these attributes should follow the posting of the primary key to the child relation.

  Foreign keys can be visualized by a bold type face.

Example



Alternate Key: eMail

Client					
clientNo	fName	lName	telNo	eMail	**staffNo**

foreign key *staffNo* references primary Key *staffNo* of *Staff*

Staff					
staffNo	fName	lName	position	sex	DOB

[Connolly & Begg]

Deriving Relations

One-to-One (1:1) Binary Relationship Types

Creating relations to represent a 1:1 relationship is slightly more complex, as the cardinality cannot be used to help identify the parent and child entities in a relationship.

Instead, the participation constraints are used to help decide whether it is best to represent the relationship by combining the entities involved into one relation or by creating two relations and posting a copy of the primary key from one relation to the other.

We consider how to create relations to represent the following participation constraints:

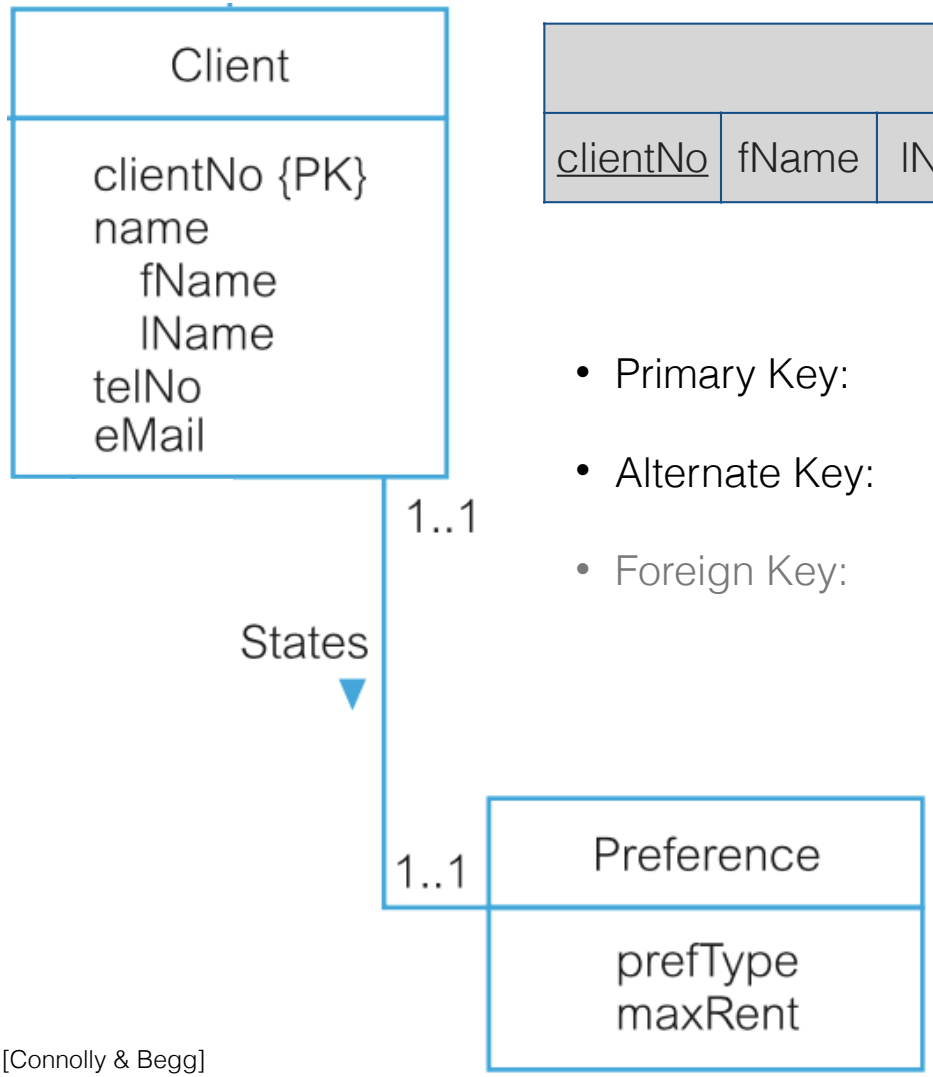
- mandatory participation on both sides of 1:1 relationship
- mandatory participation on one side of 1:1 relationship
- optional participation on both sides of 1:1 relationship

Deriving Relations

Mandatory participation on both sides of 1:1 relationship

- We should combine the entities involved into one relation and choose one of the primary keys of the original entities to be the primary key of the new relation, while the other (if one exists) is used as an alternate key.
- In the case where a 1:1 relationship with mandatory participation on both sides has one or more attributes, these attributes should also be included in the merged relation.
- Note that it is possible to merge two entities into one relation only when there are no other direct relationships between these two entities that would prevent this, such as a 1:* relationship.

Example



Client							
clientNo	fName	lName	telNo	eMail	prefType	maxRent	**staffNo**

- Primary Key: clientNo
- Alternate Key: eMail
- Foreign Key: staffNo references staffNo from Staff (from a different relationship type)

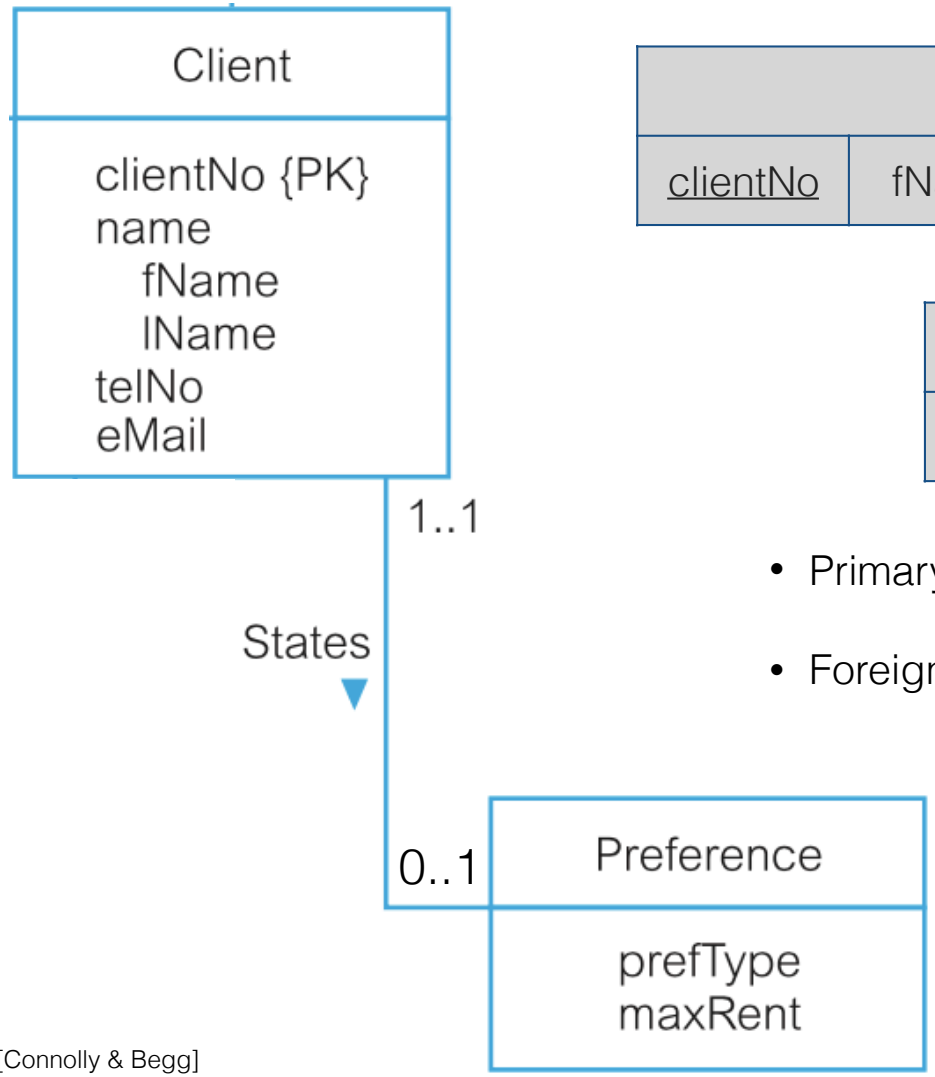
[Connolly & Begg]

Deriving Relations

Mandatory participation on one side of a 1:1 relationship

- In this case we are able to identify the parent and child entities for the 1:1 relationship using the participation constraints.
- The entity that has optional participation in the relationship is designated as the parent entity, and the entity that has mandatory participation in the relationship is designated as the child entity.
- A copy of the primary key of the parent entity is placed in the relation representing the child entity.
- If the relationship has one or more attributes, these attributes should follow the posting of the primary key to the child relation.

Example



Client					
clientNo	fName	lName	telNo	eMail	staffNo

Preference		
clientNo	prefType	maxRent

- Primary Key: *clientNo*
- Foreign Key: *clientNo* references *clientNo* of *Client*

[Connolly & Begg]

Deriving Relations

Optional participation on both sides of a 1:1 relationship

- Unless if we can find out more about the relationship the designation of the parent and child entities is arbitrary.
- If we can assess whether one side of the relationship is closer to mandatory than the other, we should choose this entity as child entity and post a copy of the primary key of the parent entity in the the child entity relation.
- Alternatively, we can create a new relation, which contains only copies of the primary key as attributes. In this case, we avoid Nulls.

Deriving Relations

One-to-One (1:1) Recursive Relationships

- mandatory participation on both sides:

  We represent the recursive relationship as a single relation with two copies of the primary key. As before, one copy of the primary key represents a foreign key and should be renamed to indicate the relationship it represents.

- mandatory participation on only one side:

  We have the option to create a single relation with two copies of the primary key as described, or to create a new relation to represent the relationship. The new relation would have only two attributes, both copies of the primary key. As before, the copies of the primary keys act as foreign keys and have to be renamed to indicate the purpose of each in the relation.

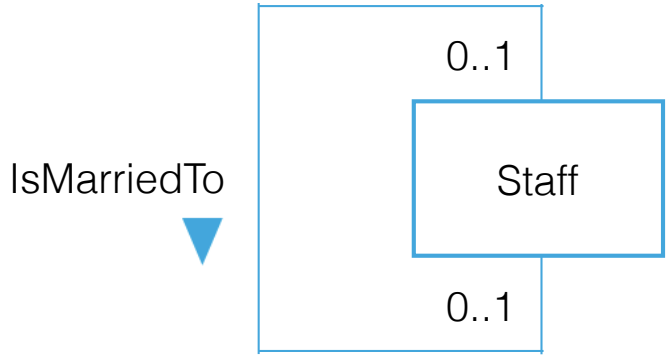
- optional participation on both sides:

  we create a new relation as described before.

Example

Situation

- optional participation on both sides



Steps

- we have to create a new relation to represent the relationship
- The new relation would have only two attributes, both copies of the primary key *staffNo*
- the copies of the primary keys act as foreign keys and have to be renamed to indicate the purpose of each in the relation

Staff					
staffNo	fName	Name	position	sex	DOB

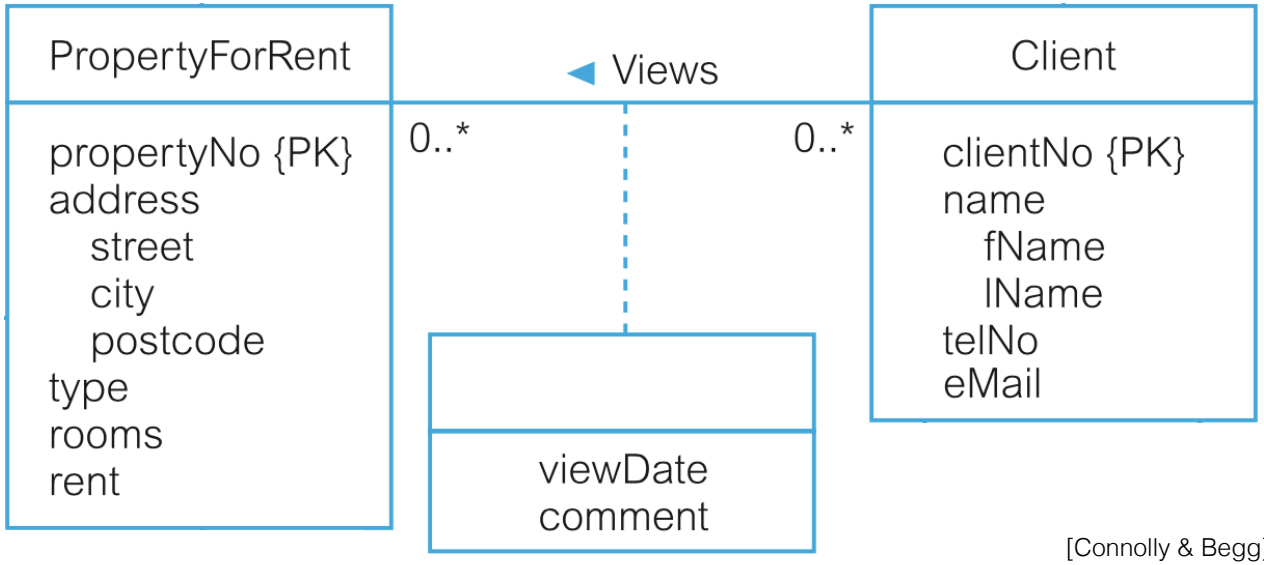
IsMarried	
partner1StaffNo	**partner2StaffNo**

Deriving Relations

Many-to-Many (*:*) Binary Relationship Types

- For each *:* binary relationship, we create a relation to represent the relationship and include any attributes that are part of the relationship.
- We post a copy of the primary key attribute(s) of the entities that participate in the relationship into the new relation, to act as foreign keys.
- One or both of these foreign keys will also form the primary key of the new relation, possibly in combination with one or more of the attributes of the relationship.
- If one or more of the attributes that form the relationship provide uniqueness, then an entity has been omitted from the conceptual data model, although this mapping process resolves this issue.

Example



[Connolly & Begg]

Client							
clientNo	fName	lName	telNo	eMail	prefType	maxRent	**staffNo**

Views			
clientNo	**propertyNo**	viewDate	comment

PropertyForRent						
propertyNo	street	city	postcode	type	rooms	rent

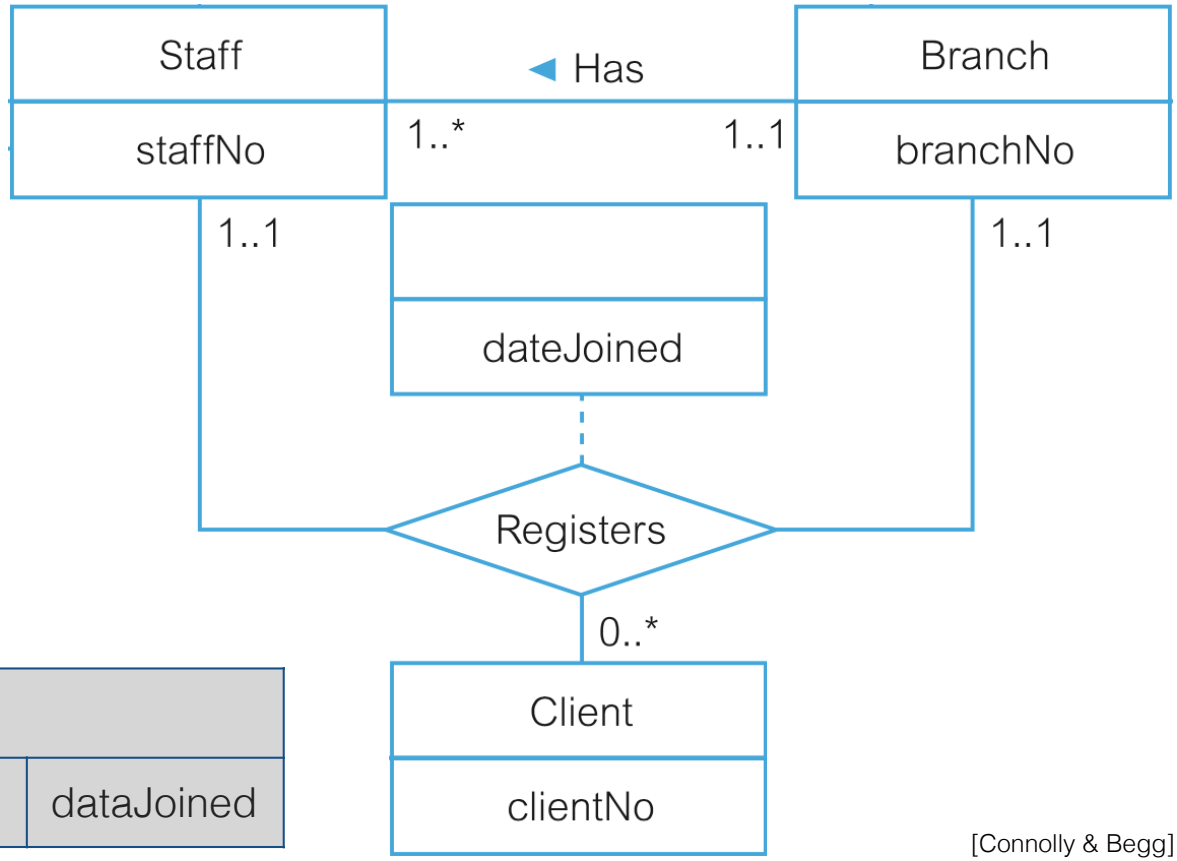
- Primary Key: (clientNo, propertyNo)
- Foreign Key: *clientNo* references *clientNo* of *Client*
- Foreign Key: *propertyNo* references *propertyNo* of *PropertyForRent*

Deriving Relations

Complex relationship types

- For each complex relationship, we create a relation to represent the relationship and include any attributes that are part of the relationship.
- We post a copy of the primary key attribute(s) of the entities that participate in the complex relationship into the new relation, to act as foreign keys.
- Any foreign keys that represent a “many” relationship (for example, 1..*, 0..*) generally will also form the primary key of this new relation, possibly in combination with some of the attributes of the relationship.

Example

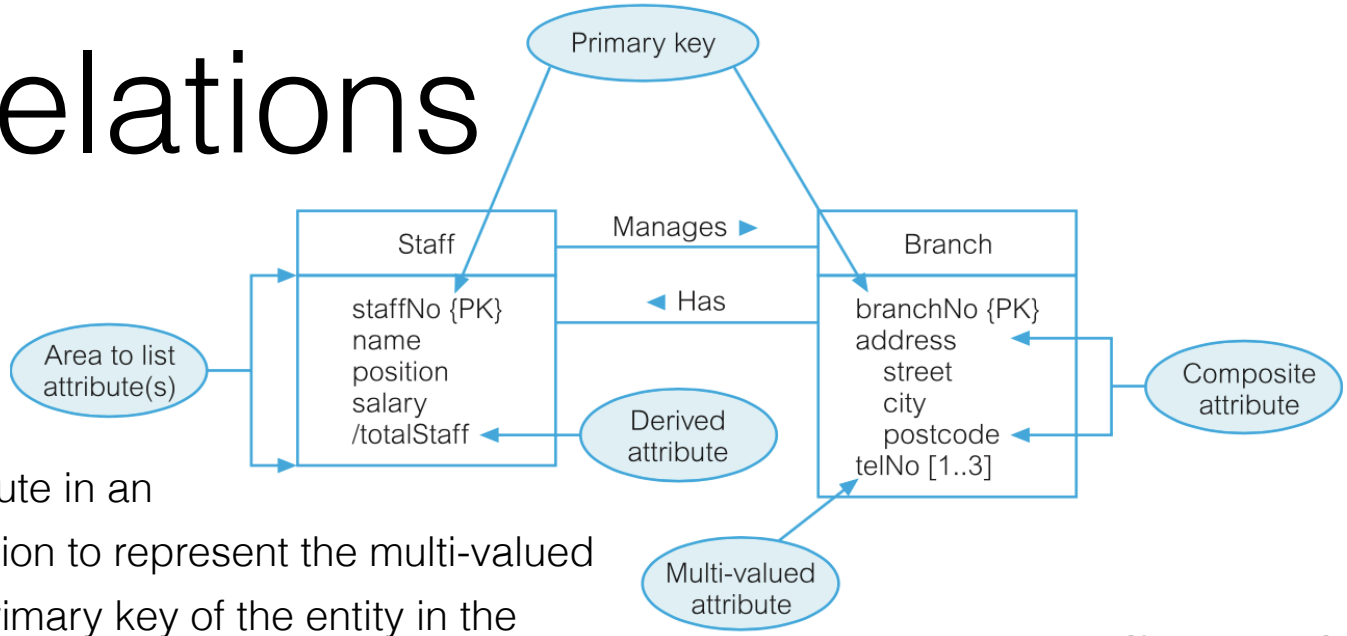


[Connolly & Begg]

Registers			
clientNo	**branchNo**	**staffNo**	dataJoined

- Primary Key: clientNo
- Foreign Key: *clientNo* references *clientNo* of *Client*
- Foreign Key: *branchNo* references *branchNo* of *Branch*
- Foreign Key: *staffNo* references *staffNo* of *Staff*

Deriving Relations



[Connolly & Begg]

Multi-valued attributes

- For each multi-valued attribute in an entity, we create a new relation to represent the multi-valued attribute, and include the primary key of the entity in the new relation to act as a foreign key.

- Unless the multi-valued attribute is itself an alternate key of the entity, the primary key of the new relation is the combination of the multi-valued attribute and the primary key of the entity.

Example

- Primary Key: telNo

- Foreign Key: *branchNo* references *branchNo* of *Branch*

Telephone	
telNo	**branchNo**

Deriving Relations

Superclass / Subclass Relationship Types

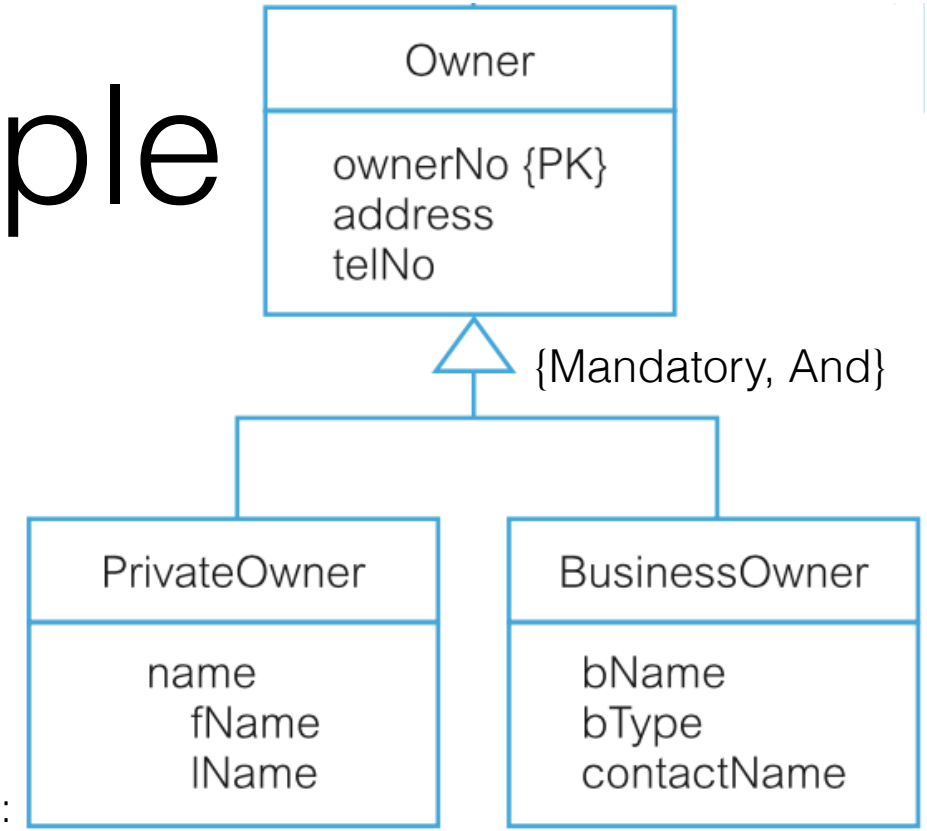
For each superclass/subclass relationship in the conceptual data model, we identify the superclass entity as the parent entity and the subclass entity as the child entity.

There are various options on how to represent such a relationship as one or more relations.

The selection of the most appropriate option is dependent on a number of factors, such as the disjointness and participation constraints on the superclass/subclass relationship, whether the subclasses are involved in distinct relationships, and the number of participants in the superclass / subclass relationship.

Guidelines for the representation of a superclass/subclass relationship based only on the participation and disjoint constraints are shown on the next slides:

Example



Mandatory, non-disjoint {And}

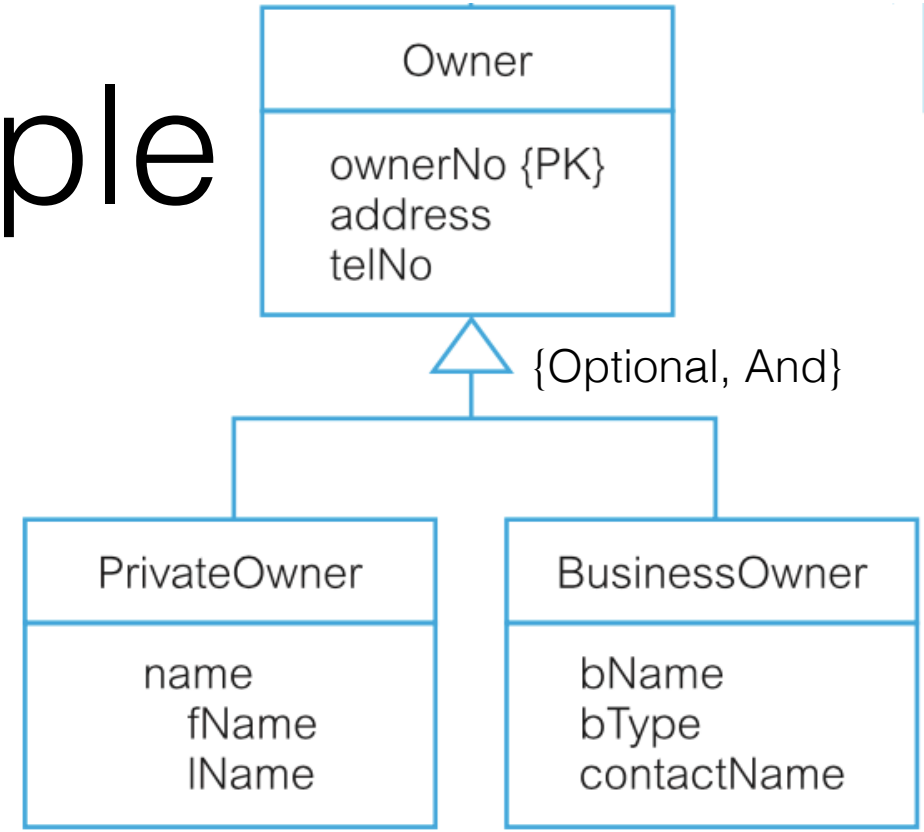
We create a single relation with one or more discriminators to distinguish the type of each tuple:

[Connolly & Begg]

AllOwner									
ownerNo	address	telNo	fName	lName	bName	bType	contactName	pOwnerflag	bOwnerFlag

Primary Key: *ownerNo*

Example



[Connolly & Begg]

Optional, non-disjoint {And}

We create a two relations:

- one relation for superclass (Owner)
- one relation for all subclasses with one or more discriminators to distinguish the type of each tuple (OwnerDetails):

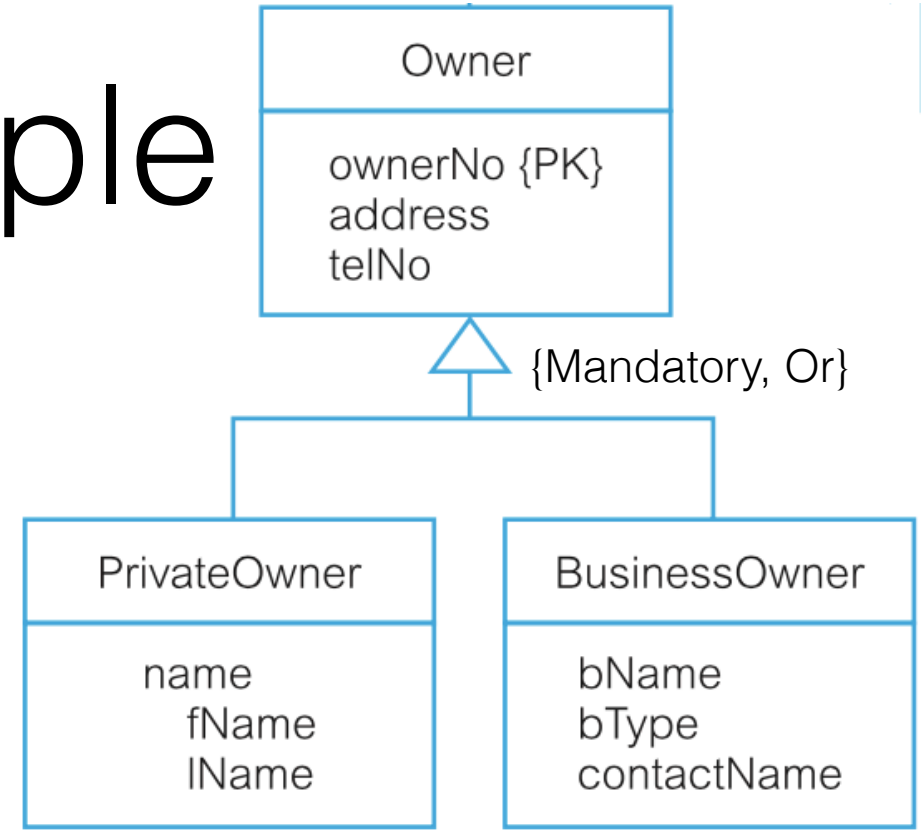
Owner		
ownerNo	address	telNo

OwnerDetails							
ownerNo	fName	lName	bName	bType	contactName	pOwnerflag	bOwnerFlag

Primary Key (for both relations): *ownerNo*

Foreign Key (for *OwnerDetails*): *ownerNo* references *ownerNo* of *Owner*

Example



[Connolly & Begg]

Mandatory, disjoint {Or}

We create many relations:

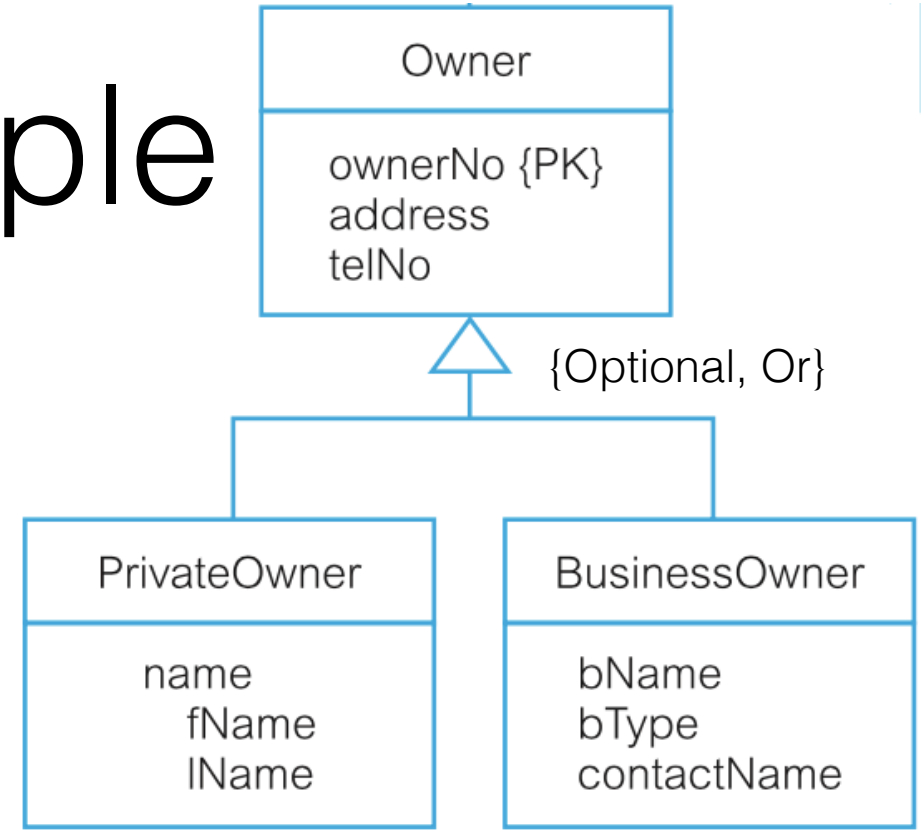
- one relation for each combined superclass / subclass:

PrivateOwner				
ownerNo	address	telNo	fName	lName

BusinessOwner					
ownerNo	address	telNo	bName	bType	contactName

Primary Key (for both relations): *ownerNo*

Example



[Connolly & Begg]

Optional, disjoint {Or}

We create many relations:

- one relation for superclass
- one relation for each subclass:

Owner		
ownerNo	address	telNo

PrivateOwner		
ownerNo	fName	lName

BusinessOwner			
ownerNo	bName	bType	contactName

Primary Key (for all relations): *ownerNo*

Foreign Key (for *PrivateOwner* and *BusinessOwner*): *ownerNo* references *ownerNo* of *Owner*

Logical Database Design - Further Steps (1)

We have derived the relational schema from the ER model, which is the most important step.

There are further steps necessary in logical database design, which we will only sketch briefly:

- the relational schema should be validated using the rules of normalization to ensure that each relation is structurally correct and unnecessary duplication of data is avoided.
- the relational schema has to be validated against user transactions to ensure that the relations in the logical data model support the required transactions
- check whether integrity constraints are represented in the logical data model ➔ next slide

Integrity Constraints

Integrity constraints are the constraints that we wish to impose in order to protect the database from becoming incomplete, inaccurate, or inconsistent.

Integrity constraints have to be documented in the data dictionary.

We consider following types of integrity constraints

- required data (which attributes must always contain values?)
- attribute domain constraints
- multiplicity
- entity integrity (the primary key of an entity cannot hold nulls)
- referential integrity (each value of a foreign key must exist as value of a primary key)
- general constraints

Logical Database Design - Further Steps (2)

We have derived the relational schema from the ER model, which is the most important step.

There are further steps necessary in logical database design, which we will only sketch briefly:

- review the logical data model with the user to ensure that they consider the model to be a true representation of the data requirements of the enterprise
- if several logical data models exist, we have to merge them into a single global logical data model that represents all user views of a database
- check for future growth to determine whether there are any significant changes likely in the foreseeable future and to assess whether the logical data model can accommodate these changes

What we did not repeat

- Essential database features, which we did not repeat, but are very important include
  - Database anomalies and rules of normalization
  - complex SQL
  - Physical Database Design
    - Storage concepts
    - Indexing structures
  - Strategies for query processing and query optimization
  - Transaction processing concepts
  - Concurrency control techniques
  - Database recovery techniques