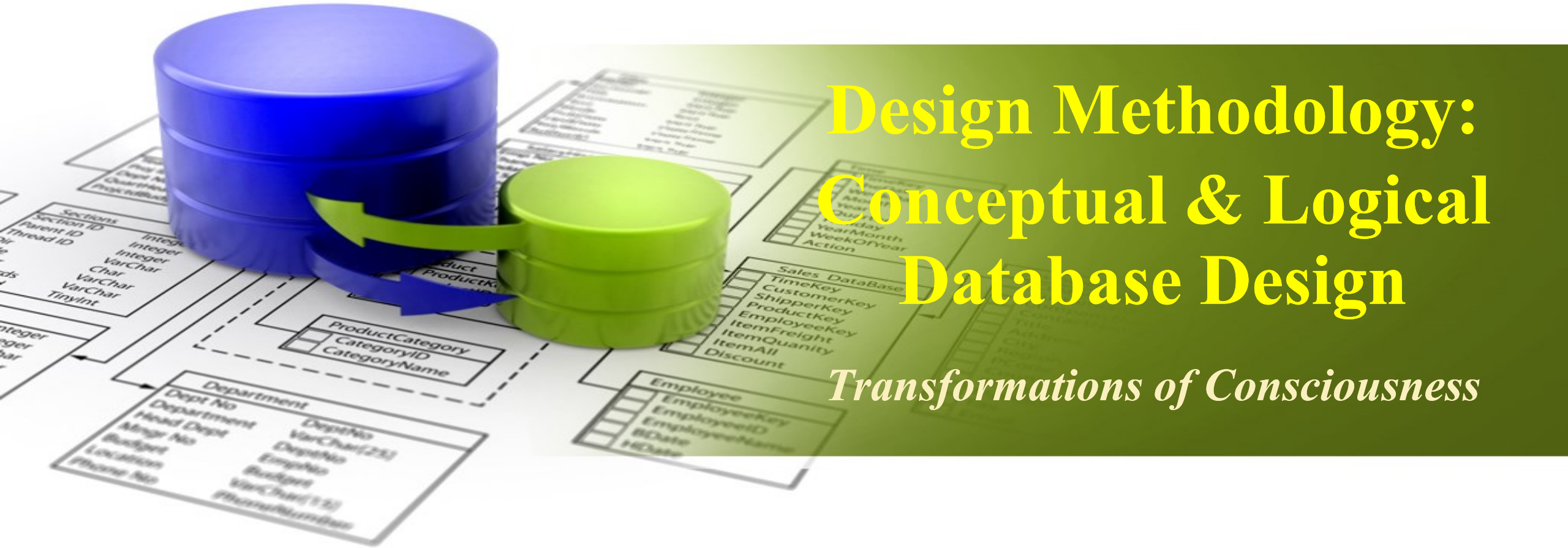
CS 422 - DBMS

Lesson 7 - Chapter 16 & 17



Design Methodology: Conceptual & Logical Database Design

Transformations of Consciousness

WHOLENESS OF THE LESSON

Logical database design starts with the conceptual model of the database and creates a set of relations which are based on the entities, attributes and their relationships that are described in the conceptual model. Science & Technology of Consciousness: The TM technique starts at the grossest level of thinking and brings the awareness to subtler and more universal levels.

Design Methodology

- Design Methodology is a structured approach that uses procedures, techniques, tools, and documentation aids to support and facilitate the process of design.
- **Three main phases of Database Design Methodology:**
  - **Conceptual database design**
    - Create the ER model
  - **Logical database design**
    - Translate ER model to tables
    - Normalize the tables
  - **Physical database design**
    - Translate logical database design to specific DBMS

Conceptual Database Design

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

- ER modelling is conceptual design. It is not designed by keeping tables in mind. Its designed only for capturing the needs of enterprise.

Logical Database Design

- The process of constructing a model of the data used in an enterprise based on a specific data model (e.g. relational), but independent of a particular DBMS and other physical considerations.
  - A data model is a collection of concepts that can be used to describe the structure of the database.

Physical Database 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 design used to achieve efficient access to the data, and any associated integrity constraints and security measures.

Overview of DB Design Methodology

Conceptual database design

1. Build conceptual data model

- Step 1.1 Identify entity types
- Step 1.2 Identify relationship types
- Step 1.3 Identify and associate attributes with entity or relationship types
- Step 1.4 Determine attribute domains
- Step 1.5 Determine candidate, primary, and alternate key attributes
- Step 1.6 Consider use of enhanced modeling concepts (optional step)
- Step 1.7 Check model for redundancy
- Step 1.8 Validate conceptual model against user transactions
- Step 1.9 Review conceptual data model with user

Overview of DB Design Methodology

Logical database design for the relational model

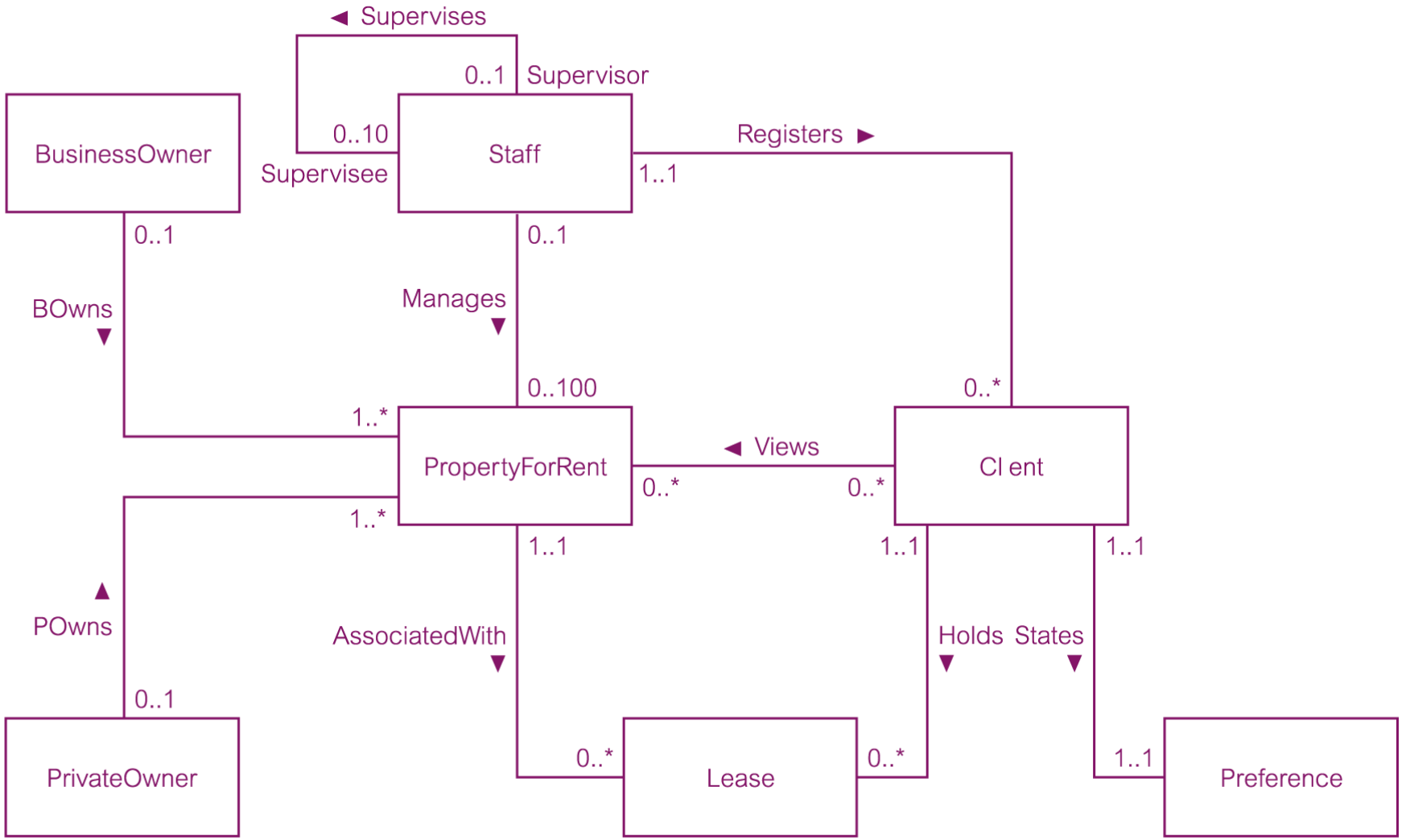
2. Build and validate logical data model

- Step 2.1 Derive relations for logical data model
- Step 2.2 Validate relations using normalization
- Step 2.3 Validate relations against user transactions
- Step 2.4 Define integrity constraints
- Step 2.5 Review logical data model with user
- Step 2.6 Merge logical data models into global model (optional step)
- Step 2.7 Check for future growth

DB Design Methodology

- Database design is an iterative process that has a starting point and an almost endless procession of refinements.
- Although the steps of the methodology are presented here as a procedural process, it must be emphasized that this does not imply that it should be performed in this manner.
- It is likely that knowledge gained in one step may alter decisions made in a previous step.
- Similarly, it may be useful to look briefly at a later step to help with an earlier step.
- Therefore, the methodology should act as a framework to help guide the designer through database design effectively.

First-cut ER Diagram for Staff User Views of *DreamHome*



Extract From Data Dictionary For Staff User Views Of *DreamHome* Showing Description Of Entities

Entity name	*Description*	*Aliases*	*Occurrence*
Staff	General term descr bing al staff employed by *DreamHome*.	Employee	Each member of staff works at one particular branch.
PropertyForRent	General term descr bing all property for rent.	Property	Each property has a s ngle owner and is available at one specific branch, where the property is managed by one member of staff. A property is viewed by many clients and rented by a single client, at any one time.

Extract from Data Dictionary for Staff User Views of *DreamHome* showing Description Of Relationships

Entity name	*Multiplicity*	*Relationship*	*Multiplicity*	*Entity name*
Staff	0..1 0..1	*Manages* *Supervises*	0..100 0..10	**PropertyForRent** **Staff**
PropertyForRent	1..1	*AssociatedWith*	0..*	**Lease**

Extract from Data Dictionary for Staff User Views of *DreamHome* showing Description of Attributes

Entity name	*Attributes*	*Description*	*Data Type & Length*	*Nulls*	*Multi-valued*	. . .
Staff	**staffNo**	Unique y identifies a member of staff	5 variable characters	No	No	
	name					
	fName	First name of staff	15 variable characters	No	No	
	lName	Last name of staff	15 variable characters	No	No	
	position	Job title of member of staff	10 variable characters	No	No	
	sex	Gender of member of staff	1 character (M or F)	Yes	No	
	DOB	Date of birth of member of staff	Date	Yes	No	
PropertyForRent	**propertyNo**	Unique y identifies a property for rent	5 variable characters	No	No	

Figure 15.4 Extract from the data dictionary for the Staff user views of *DreamHome* showing a description of attributes.

ER Diagram for Staff User Views of *DreamHome* with Primary Keys added

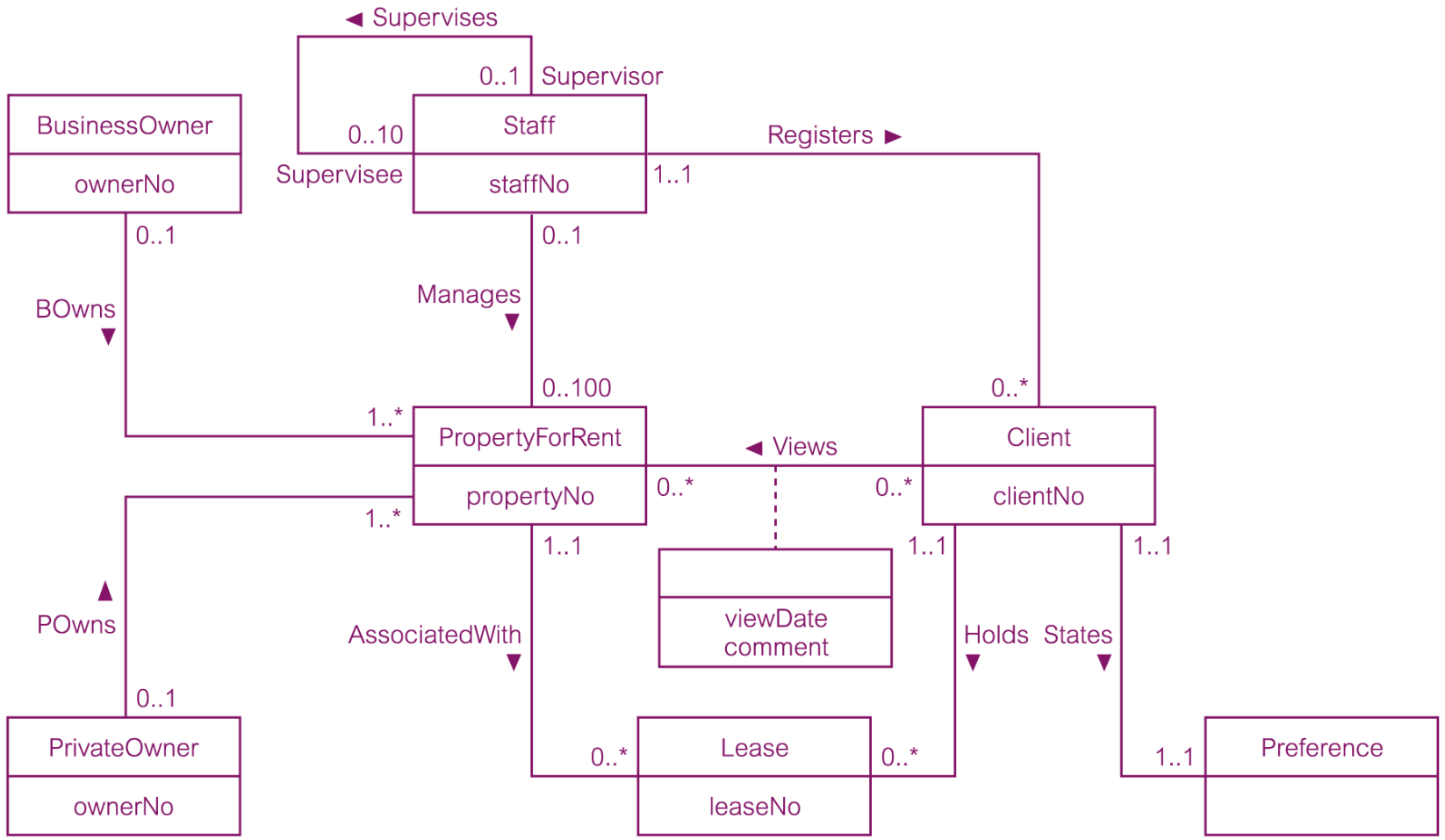


Figure 15.5 ER diagram for the Staff user views of *DreamHome* with primary keys added.

Step 2 Build and Validate Logical Data Model

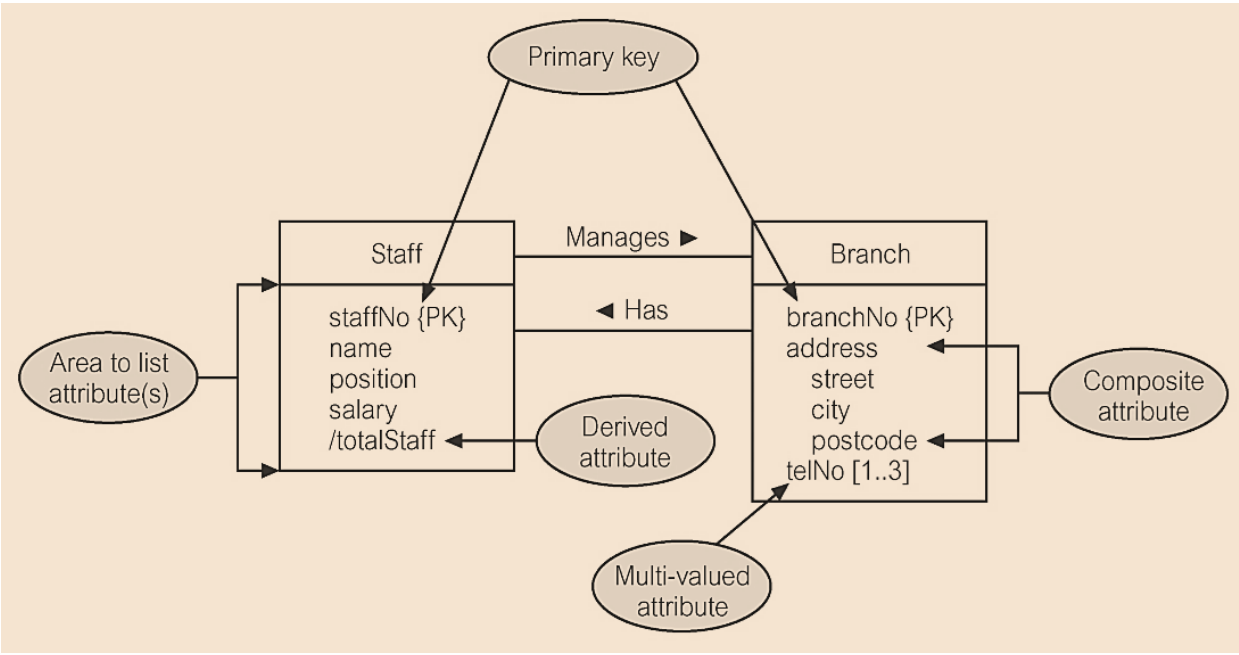
- To translate the conceptual data model into a logical data model and then to validate this model to check that it is structurally correct using normalization and supports the required transactions.

Step 2.1 Derive relations for logical data model

- To create relations for the logical data model to represent the entities, relationships, and attributes that have been identified.

How to Represent Entities?

- **(1) 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.
  - Where possible, identify the PK in each table.
- Example:



Staff (staffNo, fName, lName, position, salary)

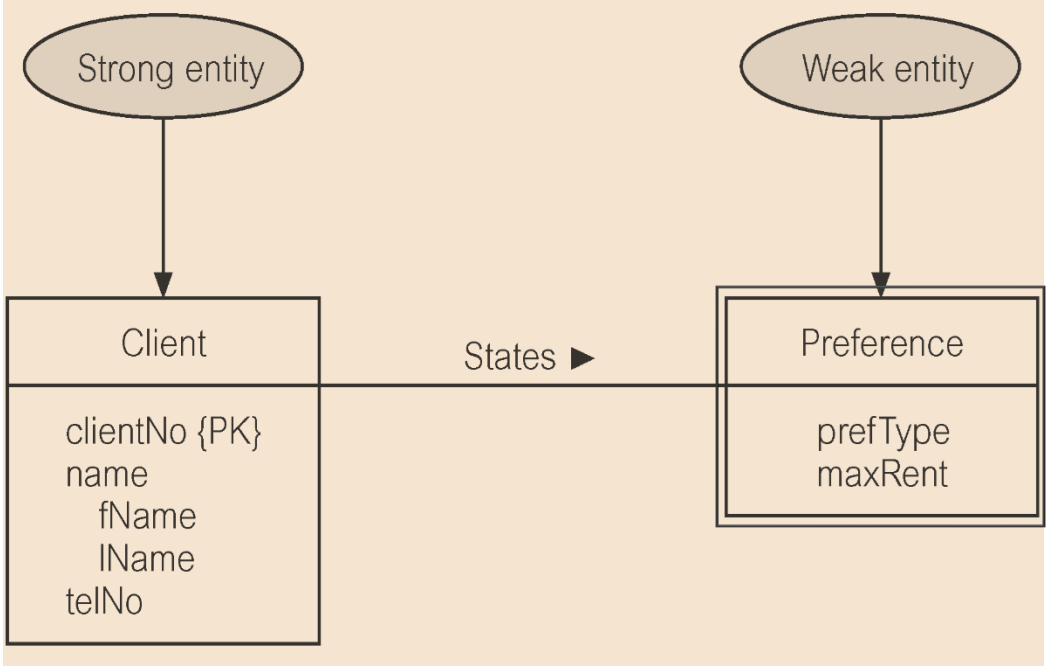
Primary Key: staffNo

How to Represent Entities? Contd..

- **(2) Weak entity types**
  - For each weak entity, create a relation that includes all the simple attributes of that entity.
  - The PK of a weak entity is partially or fully derived from each owner entity and so the identification of the PK of a weak entity cannot be made until after all the relationships with the owner entities have been mapped.
- Example:

Preference(prefType, maxRent)

Primary Key: None (at present)



How to Represent Relationships?

- Use primary key / foreign key mechanism.
- In deciding where to post (or place) the foreign key attribute(s), you 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 table that represents the child entity, to act as the foreign key.

How to Represent Relationships? Contd..

- Consider how to represent the following relationships:
  - one-to-many (1:*) binary relationships;
  - one-to-many (1:*) recursive relationships;
  - one-to-one (1:1) binary relationships;
  - one-to-one (1:1) recursive relationships;
  - many-to-many (*:*) binary relationships;
  - complex relationships;
- Also, consider multi-valued attributes.

1:* Binary Relationships

- Entity on '*one side*' of relationship is the parent entity and entity on '*many side*' is child entity.
- To represent this relationship, a copy of primary key of parent entity is placed into table representing the child entity, to act as a foreign key.
- Example: Staff *Registers* Client relationship

Staff(*staffNo*, fName, lName, position, sex, DOB)

Primary Key: staffNo

Client(clientNo, fName, lName, telNo, *staffNo*)

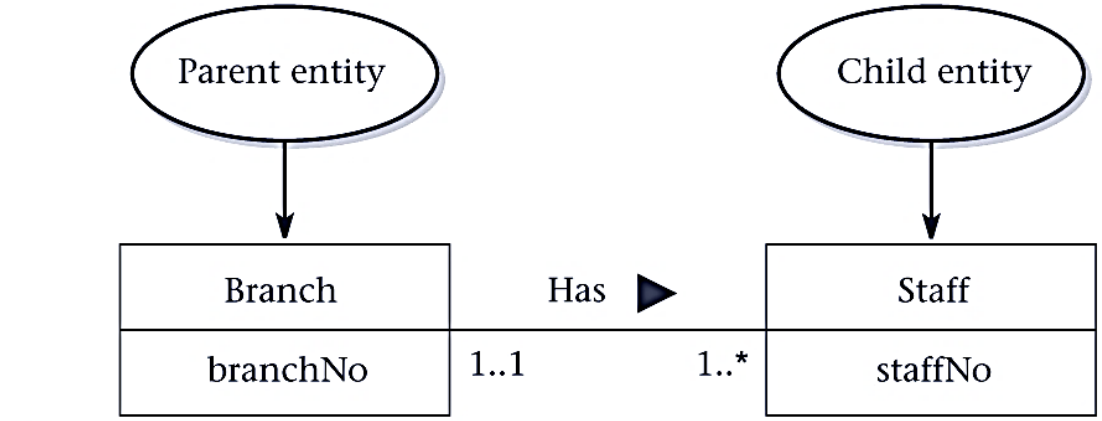
Primary Key: clientNo

Alternate Key: telNo

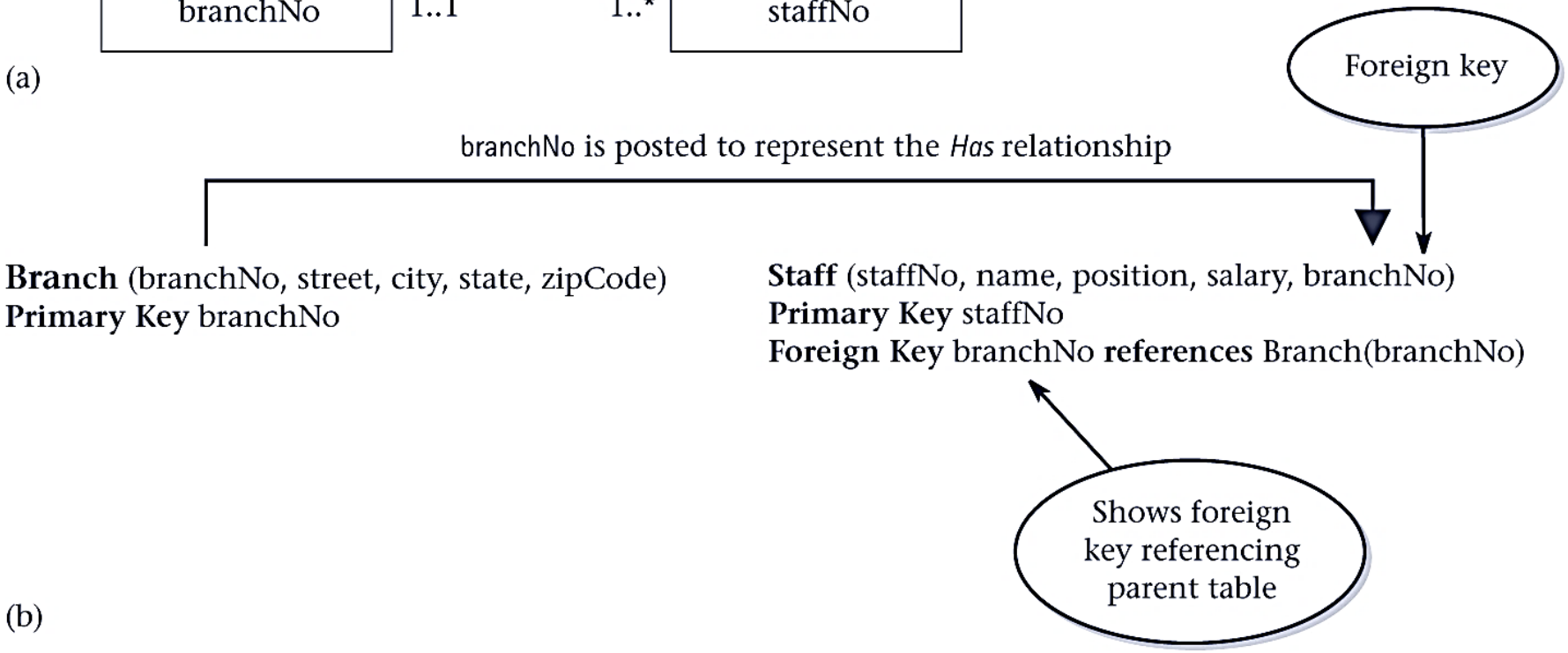
Foreign Key: staffNo **references** Staff(staffNo)

Cardinality of the relationship is used to identify parent and child entities.

1:* Binary Relationship Example 2



(a)

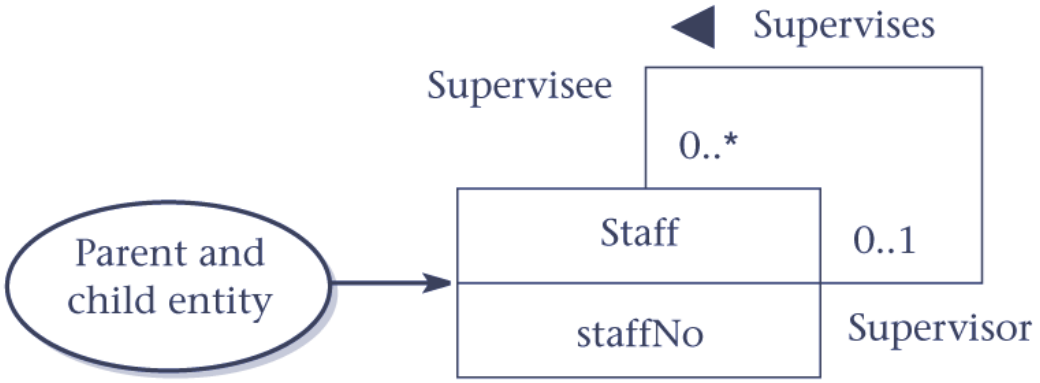


(b)

1:* Recursive Relationships

- The representation of a 1:* recursive relationship is similar to 1:* binary relationship.
- However, in this case, both the parent and child entity is the same entity.

1:* Recursive Relationship Example



(a)

staffNo is posted to represent the *Supervises* relationship and renamed supervisorStaffNo

Staff (staffNo, name, position, salary, branchNo, supervisorStaffNo)
Primary Key staffNo
Foreign Key branchNo **references** Branch(branchNo)
Foreign Key supervisorStaffNo **references** Staff(staffNo)

branchNo is posted to represent Branch *Has* Staff relationship

(b)

1:1 Binary Relationships

- Cannot use cardinality to help identify the parent and child entities.
- Instead, use participation to help decide whether it's best to represent the relationship by combining the entities involved into one table or by creating two tables and posting a copy of the primary key from one table to the other.

1:1 Binary Relationships contd..

- **Consider how to create tables 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

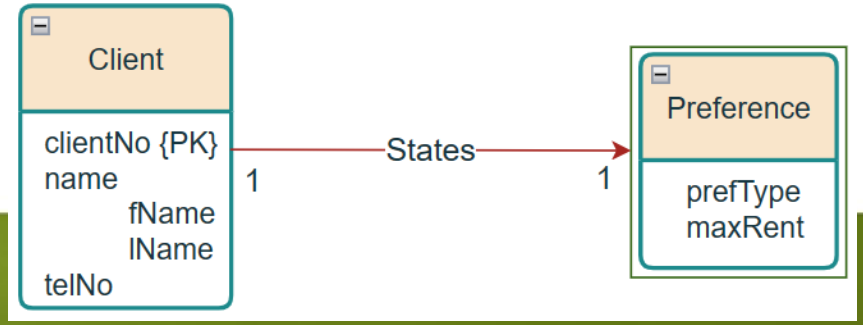
Mandatory Participation on *Both* Sides of 1:1 Relationship

- Combine the involved entities into one table and choose one of the primary keys of the original entities to be the primary key of the new table, while the other is used as an alternate key.
- Example: Client *States* Preference

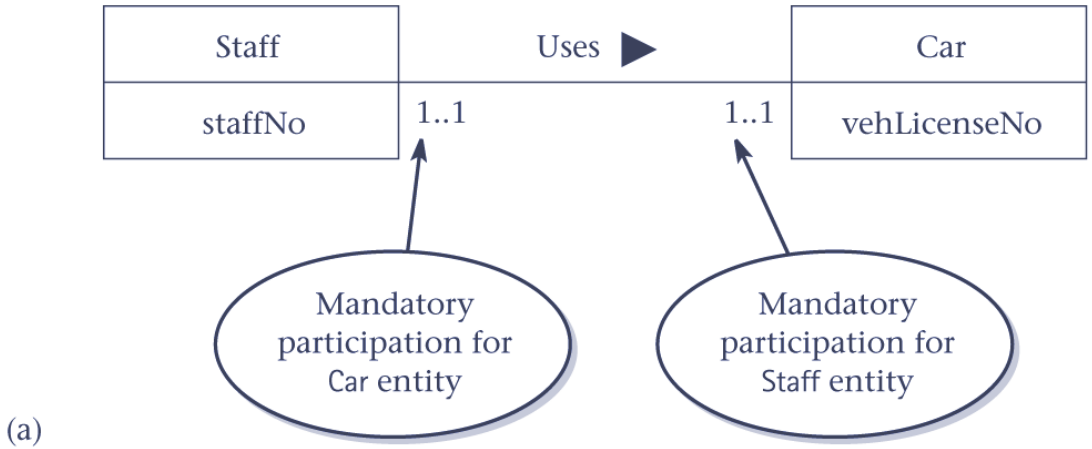
Client(clientNo, fName, lName, telNo, prefType, maxRent, staffNo)

Primary Key: clientNo

Foreign Key: staffNo **references** Staff(staffNo)



Mandatory participation on *both* sides of 1:1 relationship Example 2



(a)

Chosen as primary key of new table

Columns of Staff

Columns of Car

StaffCar (staffNo, name, position, salary, branchNo, supervisorStaffNo, vehLicenseNo, make, model)
Primary Key staffNo
Alternate Key vehLicenseNo
Foreign Key branchNo **references** Branch(branchNo)
Foreign Key supervisorStaffNo **references** Staff(staffNo)

Becomes an alternate key

(b)

Mandatory Participation on *One* Side of a 1:1 Relationship

- **Entity with optional participation is parent entity, & entity with mandatory participation is child entity.**
- A copy of primary key of parent entity is placed in the table 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.

Participation constraints are used to identify parent and child entities.

Mandatory Participation on *One* Side of a 1:1 Relationship Example 1

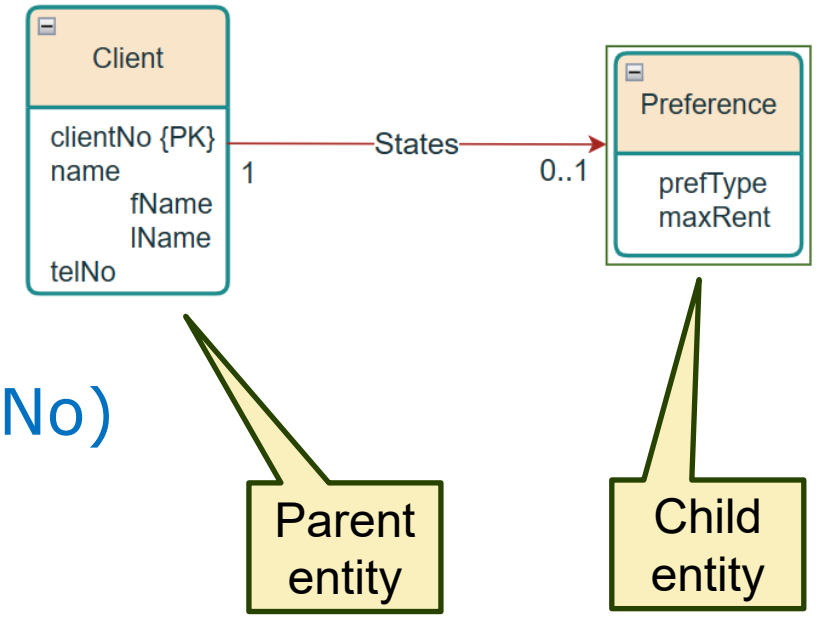
- Example: If not every client specifies preferences, then

Client(clientNo, fName, lName, telNo, staffNo)

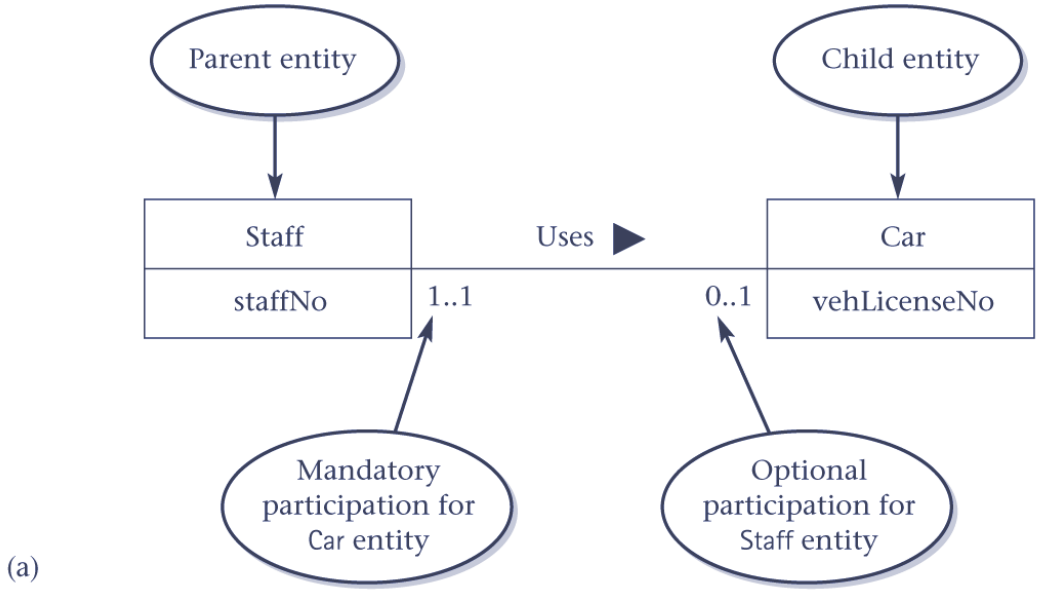
Preference(clientNo, prefType, maxRent)

Primary Key: clientNo

Foreign Key: clientNo **references** Client(clientNo)



Mandatory Participation on *One* Side of a 1:1 Relationship Example 2



(a)

staffNo is posted to represent the *Uses* relationship

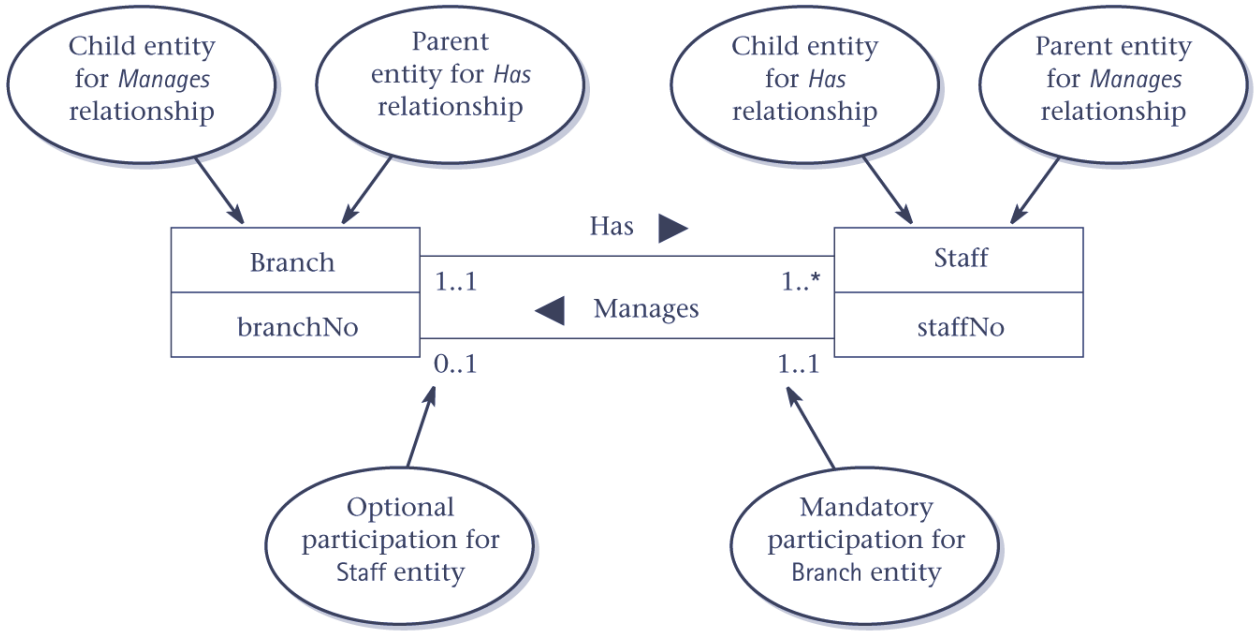
Staff (staffNo, name, position, salary, branchNo, supervisorStaffNo)
Primary Key staffNo
Foreign Key branchNo **references** Branch(branchNo)
Foreign Key supervisorStaffNo **references** Staff(staffNo)

Car (vehLicenseNo, make, model, staffNo)
Primary Key vehLicenseNo
Alternate Key staffNo
Foreign Key staffNo **references** Staff(staffNo)

staffNo becomes foreign key and alternate key

(b)

Mandatory Participation on *One* Side of a 1:1 Relationship Example 3



branchNo is posted to represent the *Has* relationship

staffNo is posted to represent the *Manages* relationship and renamed mgrStaffNo

Branch (branchNo, street, city, state, zipCode, mgrStaffNo)
Primary Key branchNo
Foreign Key mgrStaffNo **references** Staff(staffNo)

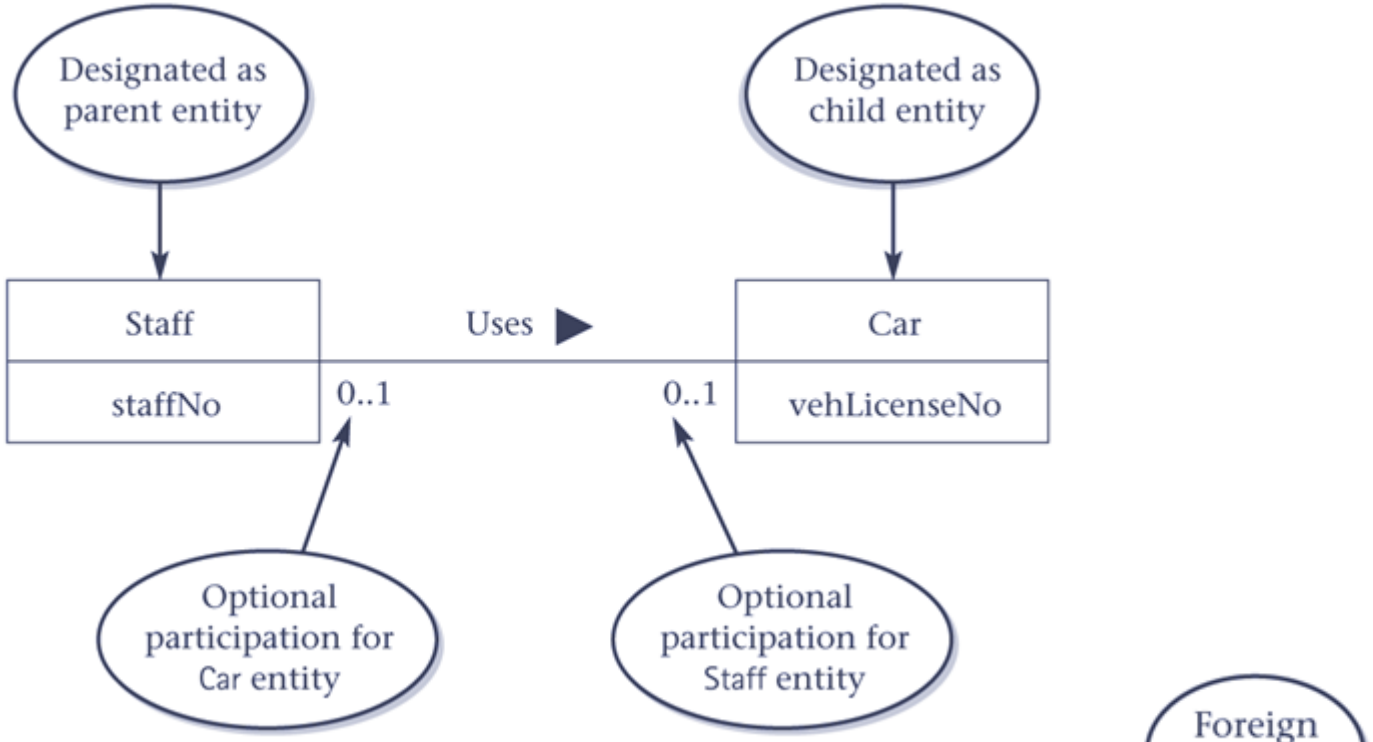
Staff (staffNo, name, position, salary, branchNo, supervisorStaffNo)
Primary Key staffNo
Foreign Key branchNo **references** Branch(branchNo)
Foreign Key supervisorStaffNo **references** Staff(staffNo)

Optional Participation on *Both* Sides of a 1:1 Relationship

- In this case, the designation of the parent and child entities is arbitrary unless you can find out more about the relationship that can help you reach a decision one way or the other.
- Or, create a new table to represent the relationship. New table has two columns, both copies of the primary key acting as foreign keys.

Optional Participation on *Both* Sides of a 1:1 Relationship Example

Assume that the majority of the cars, but not all, are used by staff and that only a minority of staff use cars. The Car entity, although optional, is closer to being mandatory than the Staff entity. We therefore designate Staff as the parent entity and Car as the child entity and post a copy of the PK of the Staff entity (staffNo) into the Car relation.



staffNo is posted to represent the *Uses* relationship

Staff (staffNo, name, position, salary, branchNo, supervisorStaffNo)
Primary Key staffNo
Foreign Key branchNo **references** Branch(branchNo)
Foreign Key supervisorStaffNo **references** Staff(staffNo)

Car (vehLicenseNo, make, model, staffNo)
Primary Key vehLicenseNo
Alternate Key staffNo
Foreign Key staffNo **references** Staff(staffNo)

More about *Mandatory* Participation on *Both* Sides of 1:1 Relationship

- Note that it is not best to merge two entities into one relation **when there exists other entities having direct relationships with these two entities**, such as 1:* relationships.
- If this were the case, we would need to represent the 1:1 relationship using the PK/FK mechanism by designating the parent and child entities the way we do in 1:1 optional on both sides.

1:1 Recursive Relationships

- Follow rules for participation as described for a 1:1 relationship.
- However, in this case, the entity on both sides of the relationship is the same.

1:1 Recursive Relationships with *Mandatory* Participation on *Both* Sides

- Represent as a single table with two copies of the primary key.
- One copy of the primary key represents a foreign key and should be renamed to indicate the relationship it represents.
- Example: Student relation (primary key: ID) with a recursive "Buddy" relationship; Add a new attribute "buddyID" which is a foreign key identifying the buddy of each student.

1:1 Recursive Relationships with *Mandatory* Participation on *One* Side

- Can create a single table with two copies of the primary key, or create a new table to represent the relationship.
- New table has two columns, both copies of the primary key acting as foreign keys. Must be renamed to indicate purpose of each in the table.
- Example: Staff relation with recursive "Supervises" relationship.

  (1) Add a "supervisor" attribute to Staff relation OR

  (2) Create a new relation SuperInfo (staffNo, supervisor) where "supervisor" is a foreign key referencing Staff(staffNo).

1:1 Recursive Relationships with *Optional* Participation on *Both* Sides

- For a 1:1 recursive relationship with optional participation on both sides, create a new table as described on the earlier slide.

: Binary Relationships

- Create a new table to represent the relationship and include any attributes that are part of the relationship.
- Post a copy of the primary key attribute(s) of the entities that participate in the relationship into the new table, to act as foreign keys.
- One or both of the foreign keys will also form the primary key of the new table, possibly in combination with some of the attributes of the relationship.

: Binary Relationship Example

- **Example**: Client *Views* PropertyForRent

Client(clientNo, fName, lName, telNo, prefType, maxRent, staffNo)

PropertyForRent(propertyNo, street, city, postcode, type, rooms, rent)

Viewing(clientNo, propertyNo, dateView, comment)

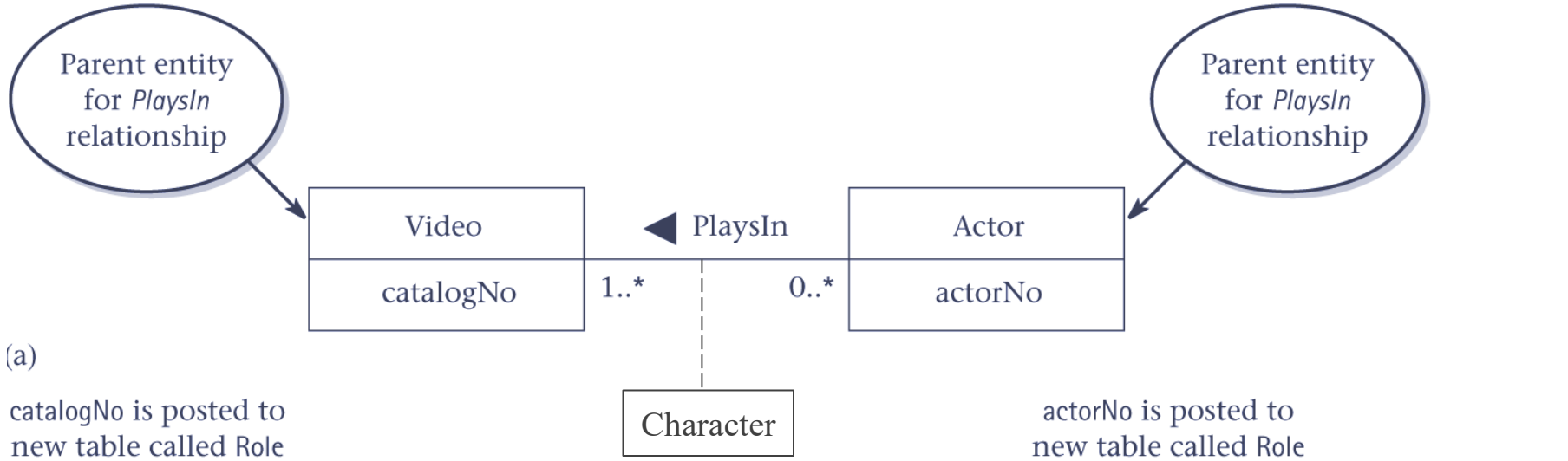
Link Table

Primary Key: clientNo, propertyNo

Foreign Key: clientNo **references** Client(clientNo)

Foreign Key: propertyNo **references** PropertyForRent(propertyNo)

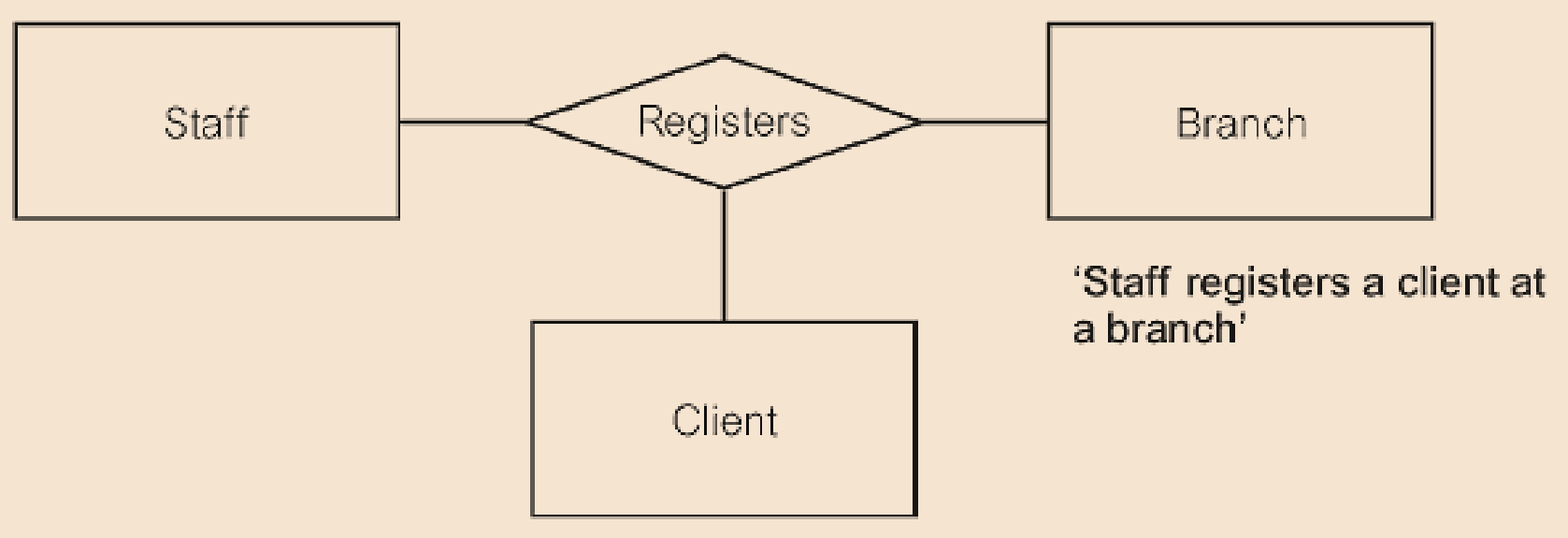
: Binary Relationship Example 2



Complex Relationships

- Create a new table to represent the relationship.
- Post a copy of the primary key attribute(s) of the entities that participate in the complex relationship into the new table, to act as foreign keys, and include any attributes that are associated with the relationship.
- One or more of the foreign keys will also form the primary key of the new table, possibly in combination with some of the attributes of the relationship.

Complex Relationships
Ternary relationship called *registers*



Complex Relationships

Ternary relationship called *registers Example*

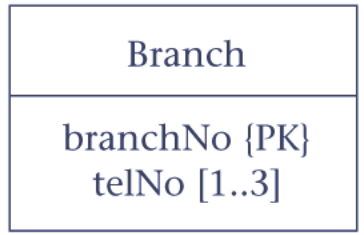
Staff (**staffNo**, fName, lName, position, …)
Primary Key: staffNo
Branch (**branchNo**, street, city, postcode)
Primary Key: branchNo
Client (**clientNo**, fName, lName, …)
Primary Key: clientNo

Registration (**clientNo**, **branchNo**, **staffNo**, dateJoined)
Primary Key: clientNo
Foreign Key: branchNo references Branch(branchNo)
Foreign Key: clientNo references Client(clientNo)
Foreign Key: staffNo references Staff(staffNo)

Multi-valued attributes

- A new table is created to hold the multi-valued attribute and the parent entity posts a copy of its primary key, to act as a foreign key.
- Unless the multi-valued attribute is itself an alternate key of the parent entity, the primary key of the new table is composed of the multi-valued attribute and the original primary key of the parent entity.

Multi-valued attributes – ER diagram and representation as tables



(a)

branchNo is posted to a new table called Telephone to act as a foreign key

telNo is moved to a new table called Telephone

Branch (branchNo, street, city, state, zipCode, telNo [1..3], mgrStaffNo)
Primary Key branchNo
Foreign Key mgrStaffNo **references** Staff(staffNo)

Branch (branchNo, street, city, state, zipCode, mgrStaffNo)
Primary Key branchNo
Foreign Key mgrStaffNo **references** Staff(staffNo)

Telephone (telNo, branchNo)
Primary Key telNo
Foreign Key branchNo **references** Branch(branchNo)

Summary of how to map entities and relationships to relations

Entity/Relationship	Mapping
Strong entity	Create relation that includes all simple attributes.
Weak entity	Create relation that includes all simple attributes (primary key still has to be identified after the relationship with each owner entity has been mapped).
1:* binary relationship	Post primary key of entity on 'one' side to act as foreign key in relation representing entity on 'many' side. Any attributes of relationship are also posted to 'many' side.
1:1 binary relationship:	
(a) Mandatory participation on both sides	Combine entities into one relation.
(b) Mandatory participation on one side	Post primary key of entity on 'optional' side to act as foreign key in relation representing entity on 'mandatory' side.
(c) Optional participation on both sides	Arbitrary without further information.
: binary relationship, complex relationship	Create a relation to represent the relationship and include any attributes of the relationship. Post a copy of the primary keys from each of the owner entities into the new relation to act as foreign keys.
Multi-valued attribute	Create a relation to represent the multi-valued attribute and post a copy of the primary key of the owner entity into the new relation to act as a foreign key.

Relations for the Staff user views of *DreamHome*

Staff (staffNo, fName, lName, position, sex, DOB, supervisorStaffNo) **Primary Key** staffNo **Foreign Key** supervisorStaffNo **references** Staff(staffNo)	**PrivateOwner** (ownerNo, fName, lName, address, telNo) **Primary Key** ownerNo
BusinessOwner (ownerNo, bName, bType, contactName, address, telNo) **Primary Key** ownerNo **Alternate Key** bName **Alternate Key** telNo	**Client** (clientNo, fName, lName, telNo, prefType, maxRent, staffNo) **Primary Key** clientNo **Foreign Key** staffNo **references** Staff(staffNo)
PropertyForRent (propertyNo, street, city, postcode, type, rooms, rent, ownerNo, staffNo) **Primary Key** propertyNo **Foreign Key** ownerNo **references** PrivateOwner(ownerNo) and BusinessOwner(ownerNo) **Foreign Key** staffNo **references** Staff(staffNo)	**Viewing** (clientNo, propertyNo, dateView, comment) **Primary Key** clientNo, propertyNo **Foreign Key** clientNo **references** Cl ent(clientNo) **Foreign Key** propertyNo **references** PropertyForRent(propertyNo)
Lease (leaseNo, paymentMethod, depositPaid, rentStart, rentFinish, clientNo, propertyNo) **Primary Key** leaseNo **Alternate Key** propertyNo, rentStart **Alternate Key** clientNo, rentStart **Foreign Key** clientNo **references** Client(clientNo) **Foreign Key** propertyNo **references** PropertyForRent(propertyNo) **Derived** deposit (PropertyForRent.rent*2) **Derived** duration (rentFinish – rentStart)	

Step 2.2 Validate Relations Using Normalization

- Check composition of each table using the rules of normalization, to avoid unnecessary duplication of data.
- Ensure each table is in at least 3NF.
- If tables are not in 3NF, this may indicate that part of the ER model is incorrect, or that error(s) introduced while creating the tables from the model.
- If necessary, may need to restructure the data model and/or tables.

Referential Integrity Constraints For The Tables

Staff (staffNo, fName, lName, position, sex, DOB, supervisorStaffNo)
Primary Key staffNo
Foreign Key supervisorStaffNo **references** Staff(staffNo) ON UPDATE CASCADE ON DELETE SET NULL

Client (clientNo, fName, lName, telNo, prefType, maxRent, staffNo)
Primary Key clientNo
Foreign Key staffNo **references** Staff(staffNo) ON UPDATE CASCADE ON DELETE NO ACTION

PropertyForRent (propertyNo, street, city, postcode, type, rooms, rent, ownerNo, staffNo)
Primary Key propertyNo
Foreign Key ownerNo **references** PrivateOwner(ownerNo) and BusinessOwner(ownerNo)
ON UPDATE CASCADE ON DELETE NO ACTION
Foreign Key staffNo **references** Staff(staffNo) ON UPDATE CASCADE ON DELETE SET NULL

Viewing (clientNo, propertyNo, dateView, comment)
Primary Key clientNo, propertyNo
Foreign Key clientNo **references** Client(clientNo) ON UPDATE CASCADE ON DELETE NO ACTION
Foreign Key propertyNo **references** PropertyForRent(propertyNo)
ON UPDATE CASCADE ON DELETE CASCADE

Lease (leaseNo, paymentMethod, depositPaid, rentStart, rentFinish, clientNo, propertyNo)
Primary Key leaseNo
Alternate Key propertyNo, rentStart
Alternate Key clientNo, rentStart
Foreign Key clientNo **references** Client(clientNo) ON UPDATE CASCADE ON DELETE NO ACTION
Foreign Key propertyNo **references** PropertyForRent(propertyNo)
ON UPDATE CASCADE ON DELETE NO ACTION

Figure 16.4
Referential integrity constraints for the relations in the Staff user views of *DreamHome*.

Step 2.5 Review Logical Model with Users

- To ensure that the logical database design is a true representation of the data requirements of an organization (or part of the organization) to be supported by the database.

Relations for the Branch user views of DreamHome

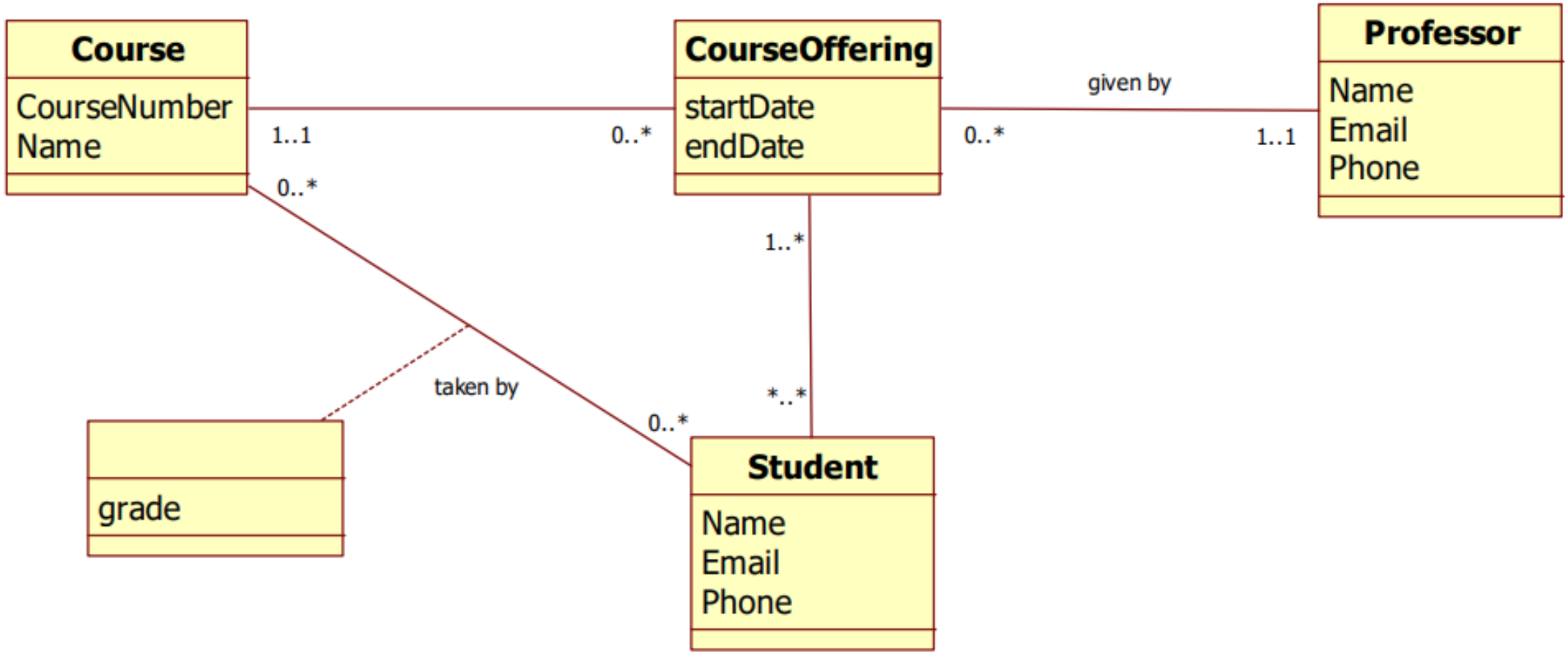
Branch (branchNo, street, c ty, postcode, mgrStaffNo) **Primary Key** branchNo **Alternate Key** postcode **Foreign Key** mgrStaffNo **references** Manager(staffNo)	**Telephone** (telNo, branchNo) **Primary Key** telNo **Foreign Key** branchNo **references** Branch(branchNo)
Staff (staffNo, name, position, salary, supervisorStaffNo, branchNo) **Primary Key** staffNo **Foreign Key** supervisorStaffNo **references** Staff(staffNo) **Foreign Key** branchNo **references** Branch(branchNo)	**Manager** (staffNo, mgrStartDate, bonus) **Primary Key** staffNo **Foreign Key** staffNo **references** Staff(staffNo)
PrivateOwner (ownerNo, name, address, telNo) **Primary Key** ownerNo	**BusinessOwner** (bName, bType, contactName, address, telNo) **Primary Key** bName **Alternate Key** telNo
PropertyForRent (propertyNo, street, city, postcode, type, rooms, rent, ownerNo, staffNo, bName, branchNo) **Primary Key** propertyNo **Foreign Key** ownerNo **references** PrivateOwner(ownerNo) **Foreign Key** bName **references** BusinessOwner(bName) **Foreign Key** staffNo **references** Staff(staffNo) **Foreign Key** branchNo **references** Branch(branchNo)	**Client** (clientNo, name, telNo, prefType, maxRent) **Primary Key** clientNo
Lease (leaseNo, paymentMethod, depositPaid, rentStart, rentFinish, clientNo, propertyNo) **Primary Key** leaseNo **Alternate Key** propertyNo, rentStart **Alternate Key** clientNo, rentStart **Foreign Key** clientNo **references** Client(clientNo) **Foreign Key** propertyNo **references** PropertyForRent(propertyNo) **Derived** deposit (PropertyForRent.rent*2) **Derived** duration (rentFinish – rentStart)	**Registration** (clientNo, branchNo, staffNo, dateJoined) **Primary Key** clientNo **Foreign Key** cl entNo **references** C ient(clientNo) **Foreign Key** branchNo **references** Branch(branchNo) **Foreign Key** staffNo **references** Staff(staffNo)
Advert (propertyNo, newspaperName, dateAdvert, cost) **Primary Key** propertyNo, newspaperName, dateAdvert **Foreign Key** propertyNo **references** PropertyForRent(propertyNo) **Foreign Key** newspaperName **references** Newspaper(newspaperName)	**Newspaper** (newspaperName, address, telNo, contactName) **Primary Key** newspaperName **Alternate Key** telNo

Critical Success Factors in Database Design

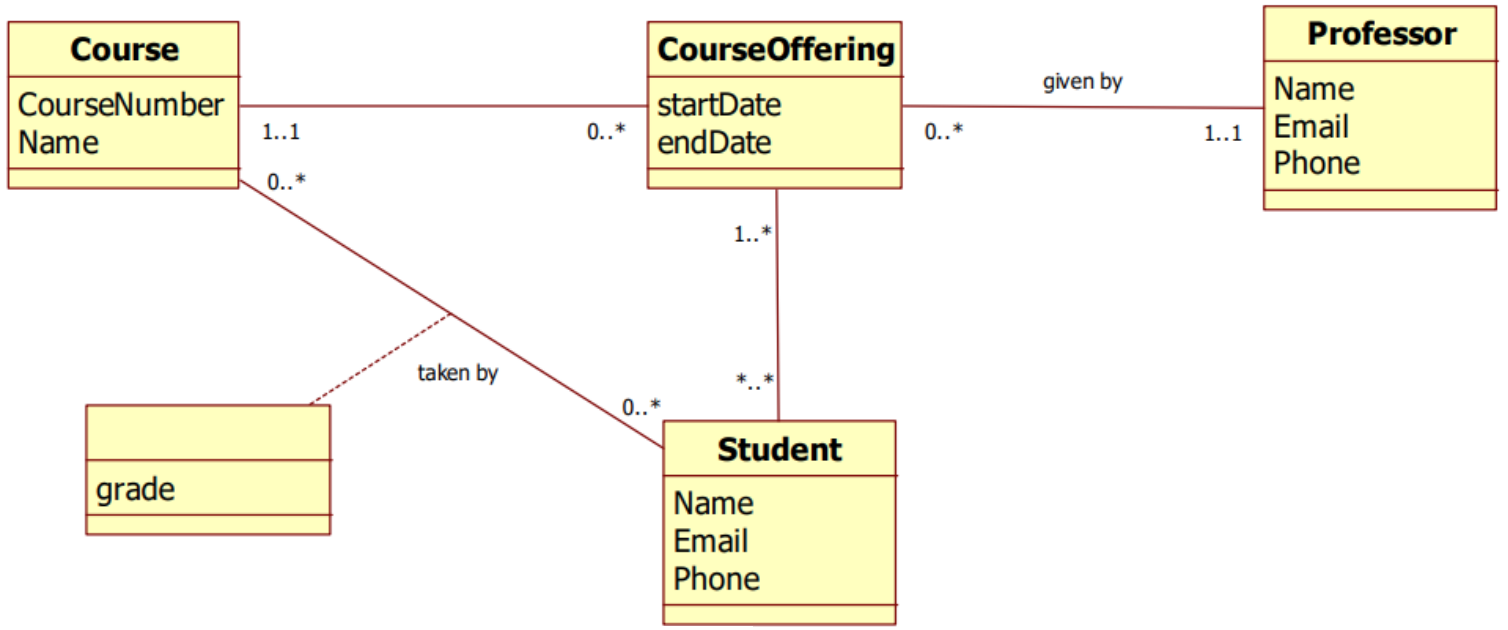
- Work interactively with the users as much as possible.
- Follow a structured methodology throughout the data modeling process.
- Employ a data-driven approach.
- Incorporate structural and integrity considerations into the data models.
- Combine conceptualization, normalization, and transaction validation techniques into the data modeling methodology.
- Use diagrams to represent as much of the data models as possible.
- Use a Database Design Language (DBDL) to represent additional data semantics.
- Build a data dictionary to supplement the data model diagrams.
- Be willing to repeat steps.

In Class Exercise

- Create a logical database design from the given ER diagram.



Entities



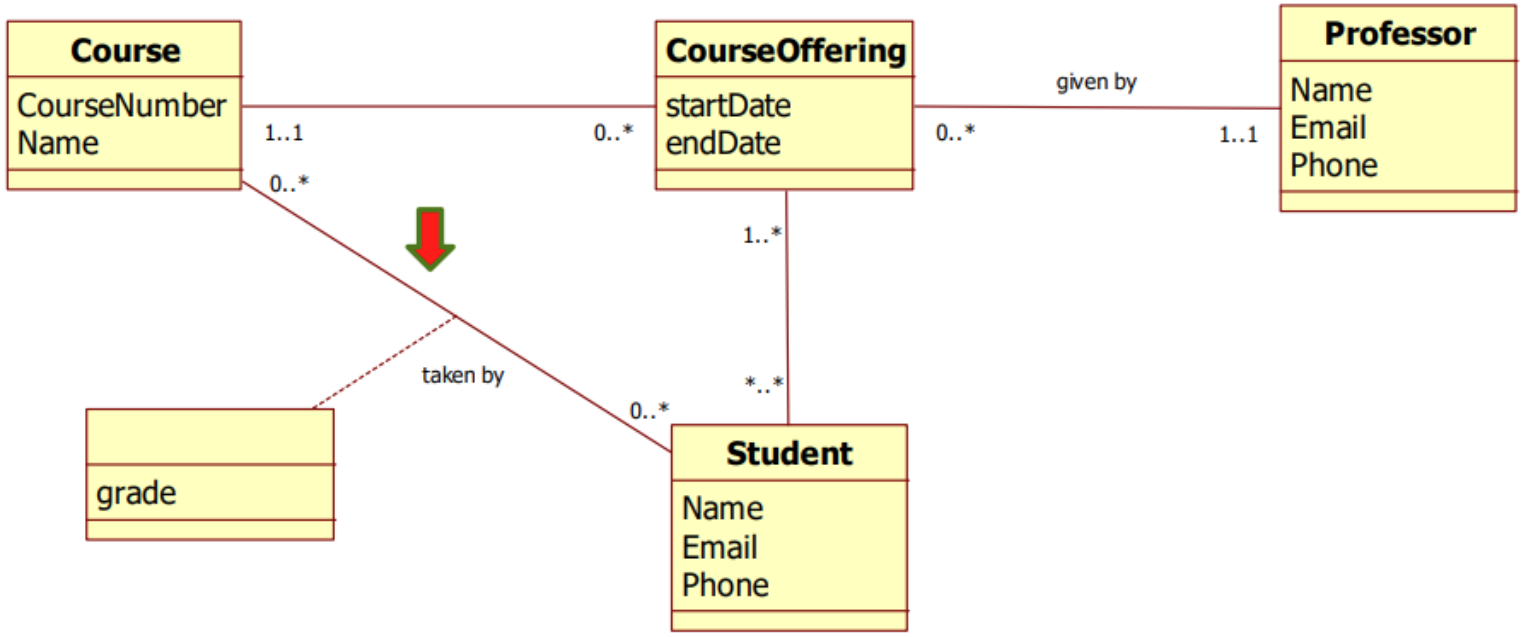
Course(**CourseNumber**, Name)

CourseOffering(**CourseOfferingNumber**, StartDate, EndDate)

Professor(**ProfessorId**, Name, Email, Phone)

Student(**StudentId**, Name, Email, Phone)

Grades



Course(**CourseNumber**, Name)

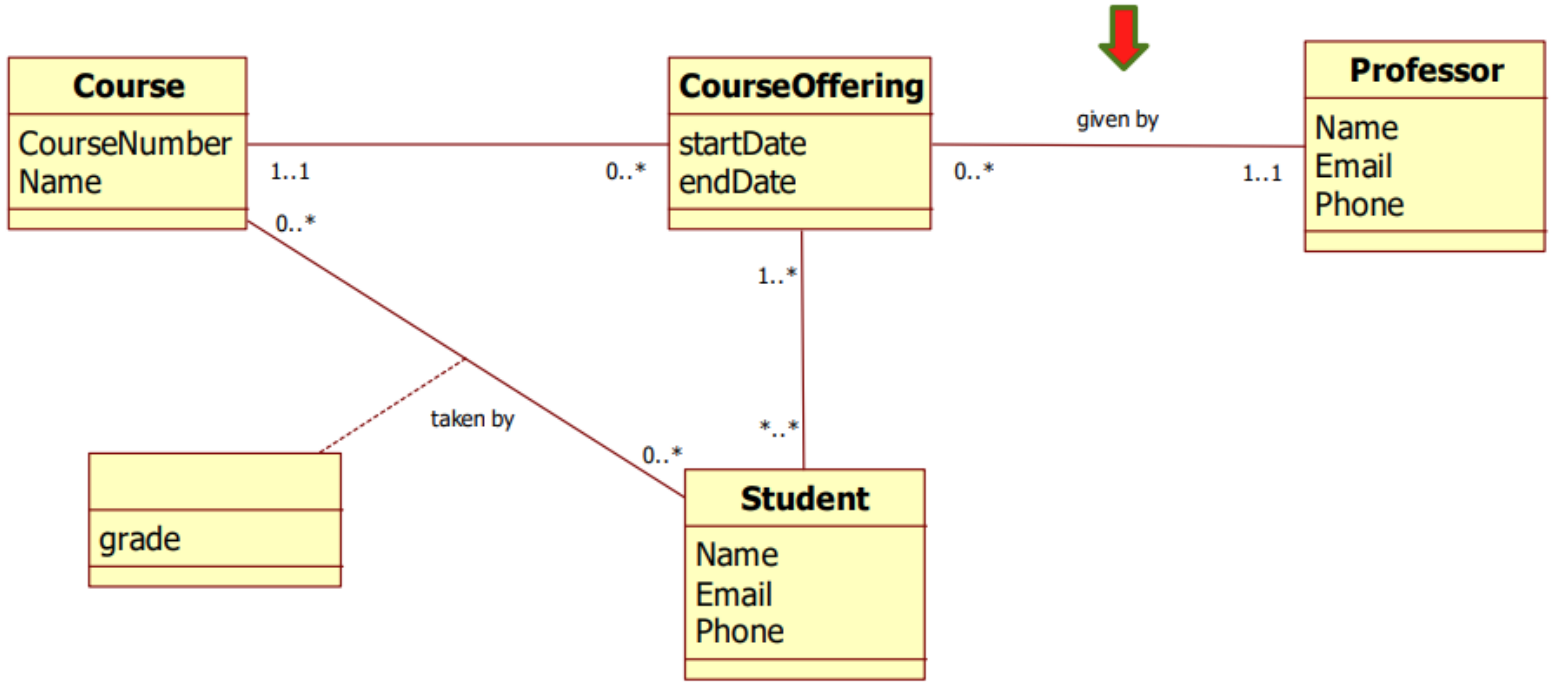
CourseOffering(**CourseOfferingNumber**, StartDate, EndDate)

Professor(**ProfessorId**, Name, Email, Phone)

Student(**StudentId**, Name, Email, Phone)

Grade (**CourseNumber, StudentId,** grade)

Relationships



Course(**CourseNumber**, Name)

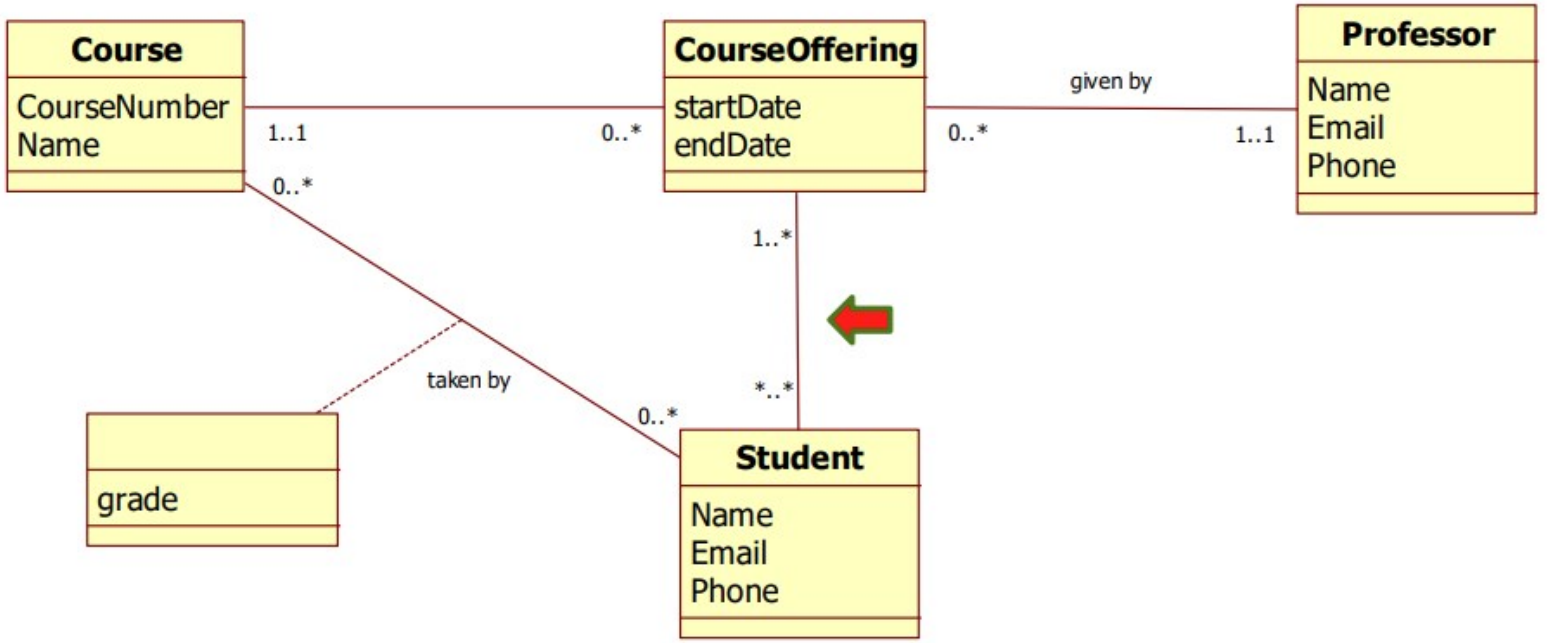
CourseOffering(**CourseOfferingNumber**, StartDate, EndDate, ProfessorId)

Professor(**ProfessorId**, Name, Email, Phone)

Student(**StudentId**, Name, Email, Phone)

Grade (**CourseNumber, StudentId,** grade)

Relationships



Course(**CourseNumber**, Name)

CourseOffering(**CourseOfferingNumber**, StartDate, EndDate, ProfessorId)

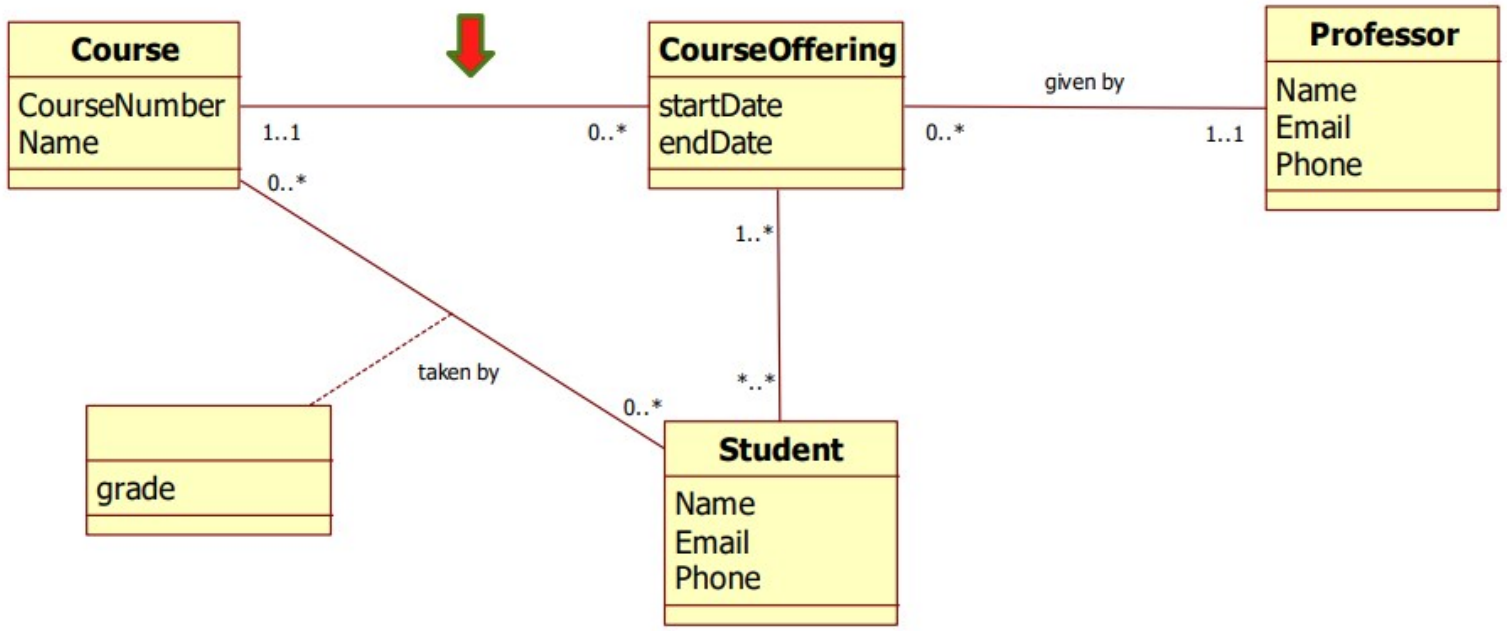
Professor(**ProfessorId**, Name, Email, Phone)

Student(**StudentId**, Name, Email, Phone)

Grade (**CourseNumber, StudentId,** grade)

CourseOffering_Student(**CourseOfferingNumber, StudentId)**

Relationships



Course(**CourseNumber**, Name)

CourseOffering(**CourseOfferingNumber**, StartDate, EndDate, ProfessorId, CourseNumber)

Professor(**ProfessorId**, Name, Email, Phone)

Student(**StudentId**, Name, Email, Phone)

Grade (**CourseNumber, StudentId,** grade)

CourseOffering_Student(**CourseOfferingNumber, StudentId)**