EER, Relational Model, and Normal Forms

EER Modelling

Super-classes and sub-classes

So far we have covered

CITY UNIVERSITY LONDON

- Entities
- Attributes
- Keys
- Relationships
- Constraints

Enhanced Entity-Relationship (EER) model

CITY UNIVERSITY LONDON
EST 1894

- ER Diagram can't capture Specialisation / Generalisation
  - Bank staff are: Managers, Supervisors, Assistants, …
  - Hospital staff are: Medical staff, Admin staff, Facility staff, etc.
  - Medical staff are: Doctors, Nurses, Technicians, etc.
- Specific staff group have particular attributes relationships
- ER model cannot represent this - EER can!

Enhanced Entity-Relationship (EER) Model

- an ER model with additional semantic concepts
- represented in UML

Entity Type Hierarchy

CITY UNIVERSITY LONDON
EST 1894

- A hierarchy contains of Subclasses and Superclasses
- An entity instance of a Subclass is an entity instance of a Superclass
- Subclass – a distinct sub-grouping of occurrences of an entity type
  - e.g., Manager, Sales Personnel, Secretary is a Staff
- Superclass – an entity type with distinct subclasses (e.g., Staff)
- Subtypes are related to supertypes through the “is a” relationship:
  - “Manager is a Staff”
  - “Sales Personnel is a Staff”
- A subclass may also have one or more subclasses, e.g.:
  - Staff is a supertype of MedicalStaff; MedicalStaff is its subtype
  - MedicalStaff is supertype of Doctor; Doctor is its subtype

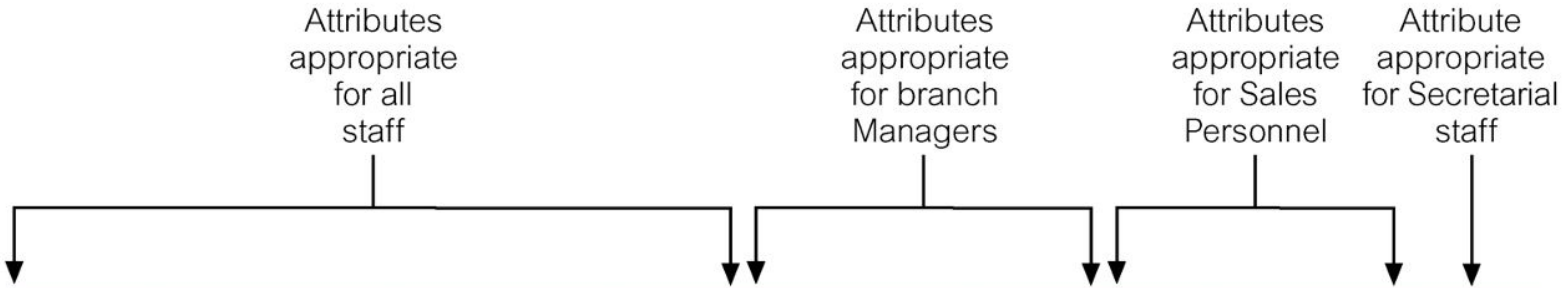
Superclass/Subclass Relationships

CITY UNIVERSITY LONDON
EST 1894

- Each member of subclass is also a member of the superclass
  - All Doctor entities are *MedicalStaff*
- Superclass may contain subclasses that are overlapping or distinct
- Not all members of superclass must be member of a subclass
  - An entity can be *MedicalStaff* without belonging to a subclass

Common and Specific Attributes

CITY UNIVERSITY LONDON
EST 1894

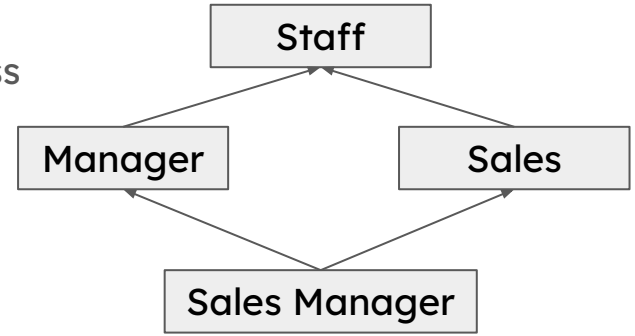


staffNo	name	position	salary	mgrStartDate	bonus	sales Area	car Allowance	typing Speed
SL21	John White	Manager	30000	01/02/95	2000			
SG37	Ann Beech	Assistant	12000					
SG66	Mary Martinez	Sales Manager	27000			SA1A	5000	
SA9	Mary Howe	Assistant	9000					
SL89	Stuart Stern	Secretary	8500					100
SL31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

Superclass/Subclass Attributes

CITY UNIVERSITY LONDON
EST 1894

- Each subclass inherits all attributes of its superclass
  - Staff : {staffNo, name, address, salary}
    - Managers : {above attributes} + {mgrStartDate, Bonus}
    - SalesPersonnel: {above attributes} + {salesArea, carAllowance}
    - Secretarial staff have: {above attributes} + {typing Speed}
- In EER diagram, inherited attributes only appear once
  - they go in the highest superclass
- **Multiple inheritance**
  - a subclass may also have more than one superclass
  - such a subclass is called a **shared subclass**
  - the attributes of the superclasses are inherited by the shared subclass



Specialisation/Generalisation (S/G)

CITY UNIVERSITY LONDON

- **Specialisation**
  - process of maximising differences between members of an entity by identifying their distinguishing characteristics
  - a top-down approach
  - process of identifying differences between entities that we previously grouped together
- **Generalisation**
  - process of minimising differences between entities by identifying their common characteristics
  - a bottom-up approach
  - process of identifying similarities between different entities through common attributes and/or relationships

Constraints on S/G

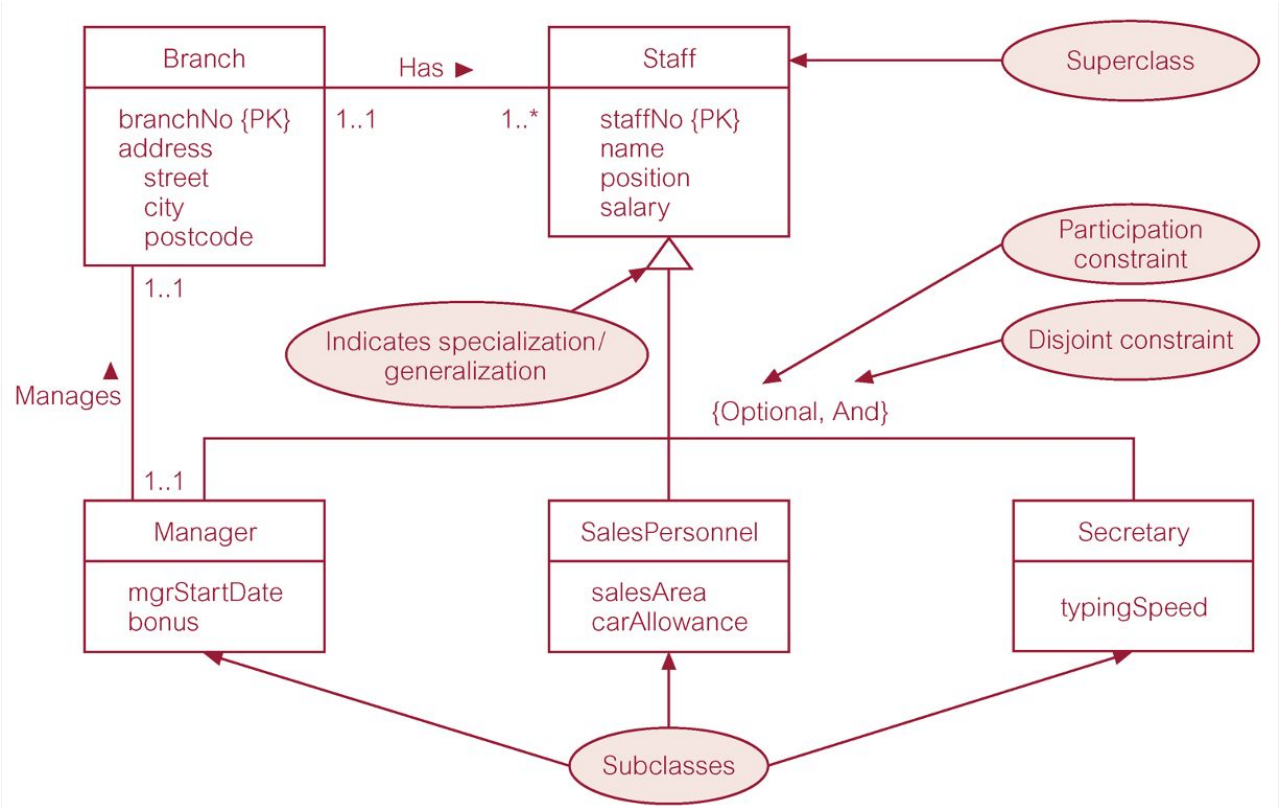
CITY UNIVERSITY LONDON
EST 1894

{Mandatory/Optional, And/Or}

- **Participation constraint**
  - determines whether every member in superclass must participate as a member of subclass: mandatory | optional
  - **mandatory participation**: every member of staff must have a contract of employment
  - **optional participation**: member of staff need not to have an additional job role
- **Disjoint constraint**
  - describes relationship between members of the subclasses and indicates whether a member of a superclass can be a member of one, or more than one, subclass
  - may be *disjoint* {Or} or *non-disjoint* {And}

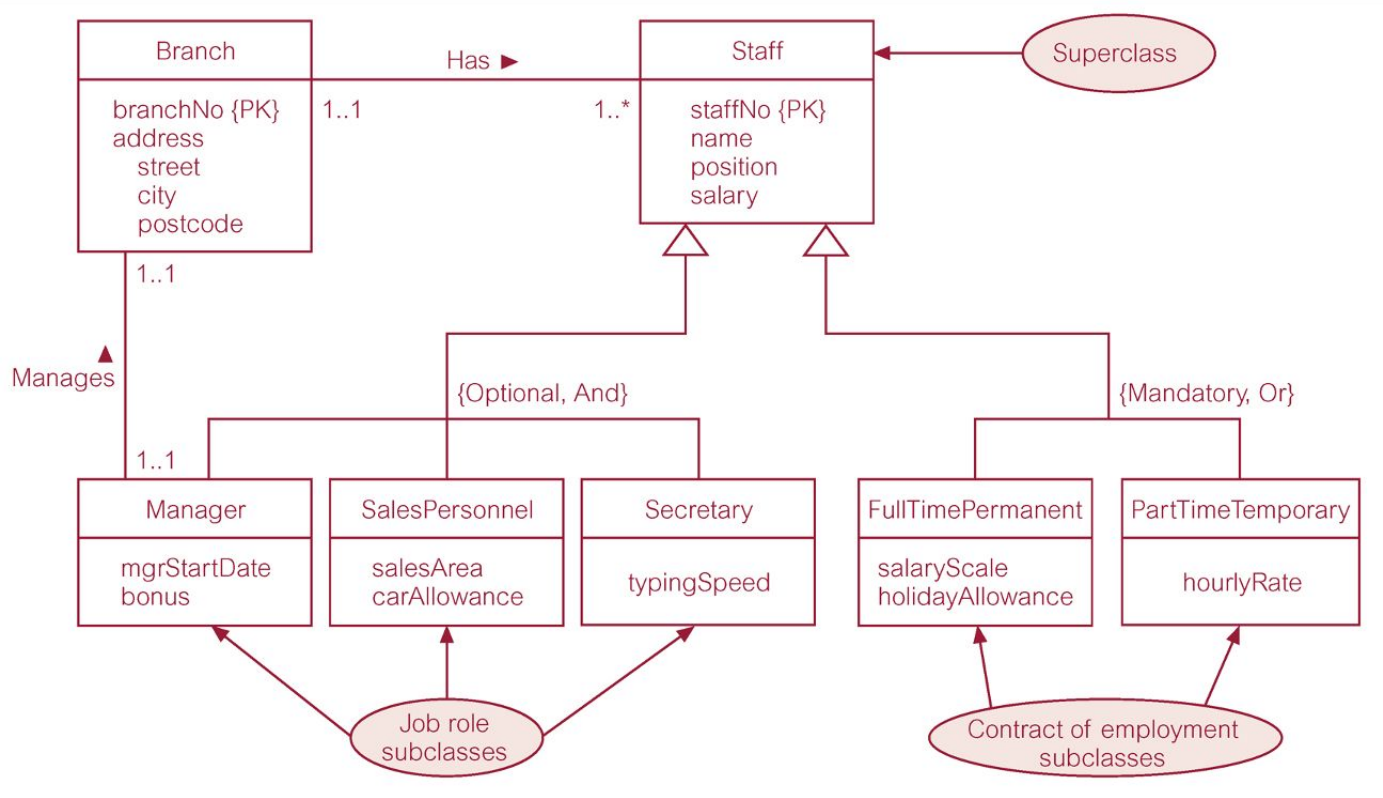
S/G of Staff Entity: Job Role Subclasses

CITY UNIVERSITY LONDON
EST 1894

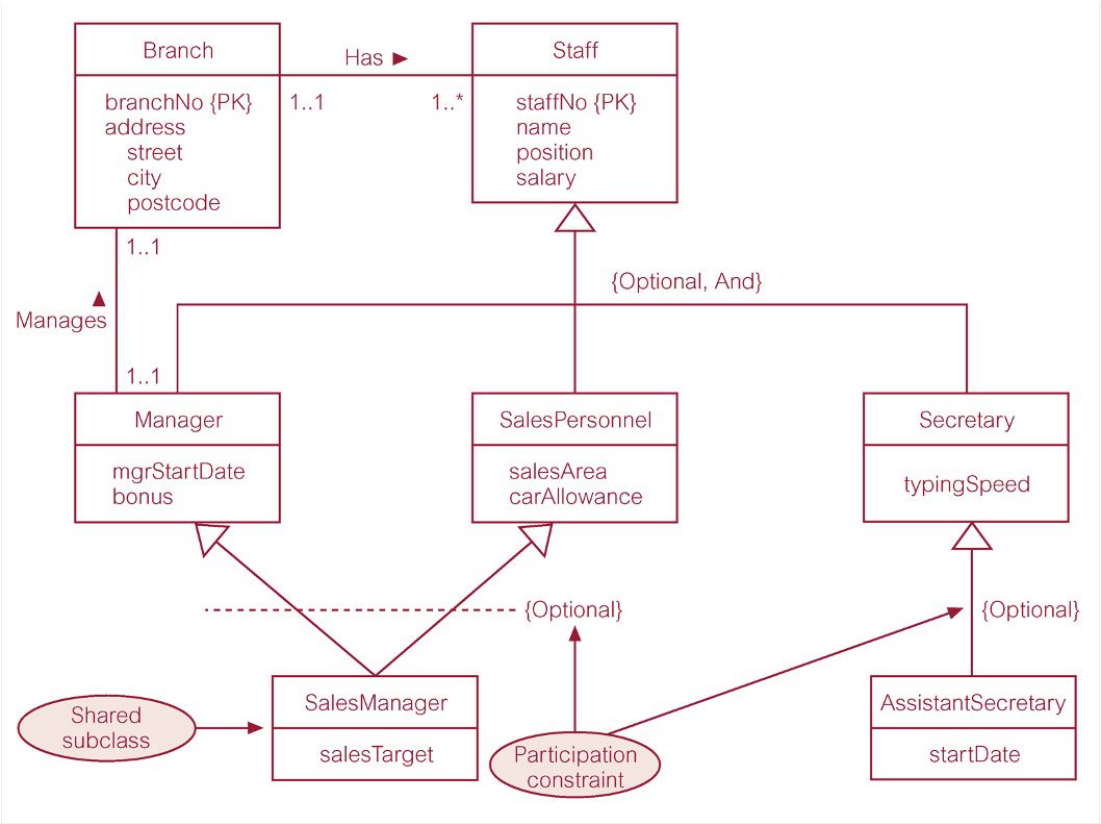


S/G of Staff Entity: Job Roles & Contracts

CITY UNIVERSITY LONDON
EST 1894



More on Subclasses



- **shared Subclass**
- **subclass with its own Subclass**

Super-classes and sub-classes. Example 2

CITY UNIVERSITY LONDON

All staff have a staff number and we record their names, addresses, their department name, and pay grade. Some staff are managers, they get an additional responsibility allowance. Staff in the sales department get a car; identified by its registration number. Some managers work in the sales department.

- Some attributes apply to all staff
- Some attributes only apply to defined subsets

Super-classes and sub-classes. Example 2

CITY UNIVERSITY LONDON

- The attributes common to all staff are allocated to a superclass entity, Staff
  - staff_no, name, address, pay grade.
  - What should be the primary key?
- Attributes common to a sub-group are allocated to one or more sub-class entities as appropriate
  - Manager; allowance
  - Sales_staff; reg_no

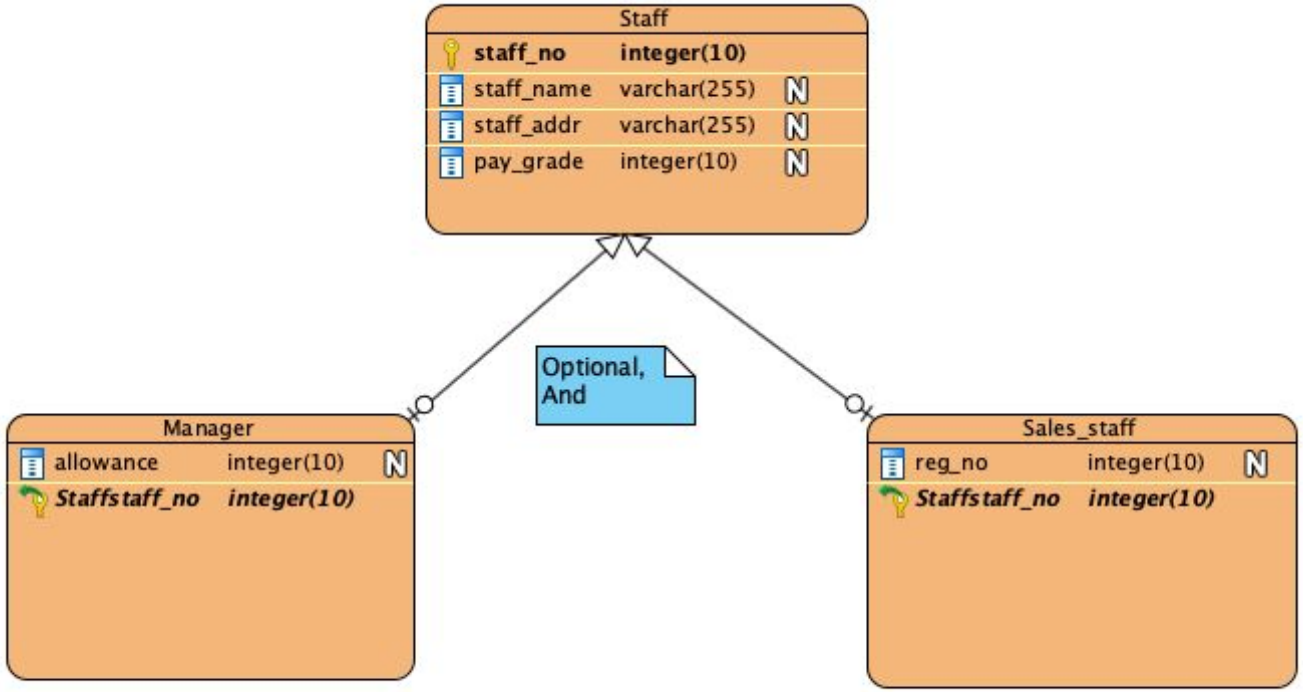
Super-classes and sub-classes. Example 2

CITY UNIVERSITY LONDON

- Sub-classes inherit the primary key of the super-class
  - Staff; staff_no, name, address, pay_grade
  - Manager; staff_no, allowance
  - Sales_staff; staff_no, reg_no
- What is the relationship between:
  - Staff and Manager?
  - Staff and Sales_staff?
- What are the relationship cardinalities? What are the constraints? Are they identifying or non-identifying?

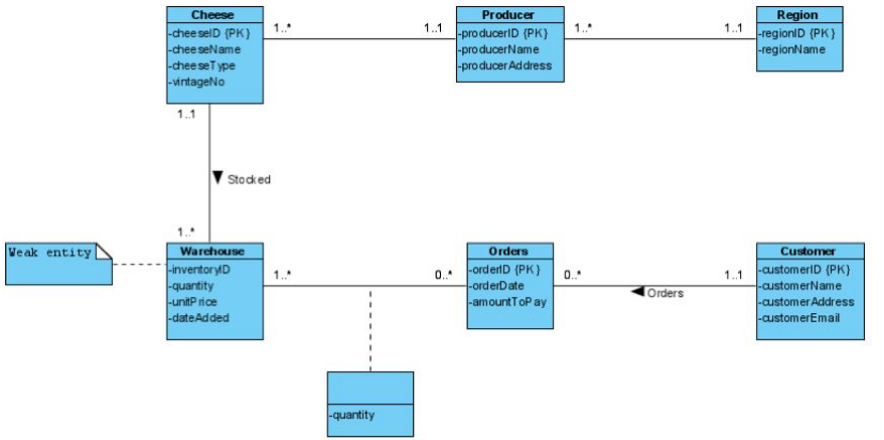
Super-classes and sub-classes. Example 2

CITY UNIVERSITY LONDON
EST 1894



From ER to Relational Model

Logical Design – Map ER to RM



Region (**regionID**, regionName)

Producer (**producerID**, producerName, producerAddress, regionID)
FOREIGN KEY regionID REFERENCES Region (regionID)

Cheese (**cheeseID**, cheeseName, cheeseType, vintageNo, producerID)
FOREIGN KEY producerID REFERENCES Producer (producerID)

Customer (**customerID**, customerName, customerAddress, customerEmail)

Order (**orderID**, orderDate, amountToPay, customerID)
FOREIGN KEY customerID REFERENCES Customer (customerID)

Warehouse (**cheeseID**, **inventoryID**, quantity, unitPrice, dateAdded)
FOREIGN KEY cheeseID REFERENCES Cheese (cheeseID)

WarehouseOrder (**cheeseID**, **inventoryID**, **orderID**, quantity)
FOREIGN KEY (cheeseID, inventoryID) REFERENCES Warehouse (cheeseID, inventoryID)
FOREIGN KEY orderID REFERENCES Order (orderID)

General Rules

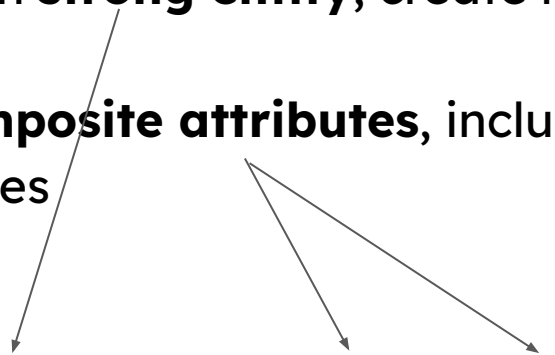
CITY UNIVERSITY LONDON

- Entities
  - an entity type becomes a relation with the same name
  - weak entity types – a relation with all simple attributes
  - primary key comes from the relation assigned to their defining relationship
- Attributes
  - simple attributes are copied directly into the relation
  - primary key is copied directly into the relation
  - composite attributes are split into a separate single-valued attributes in the entity
  - e.g., name → fName and lName

Strong Entities

CITY UNIVERSITY LONDON
EST 1894

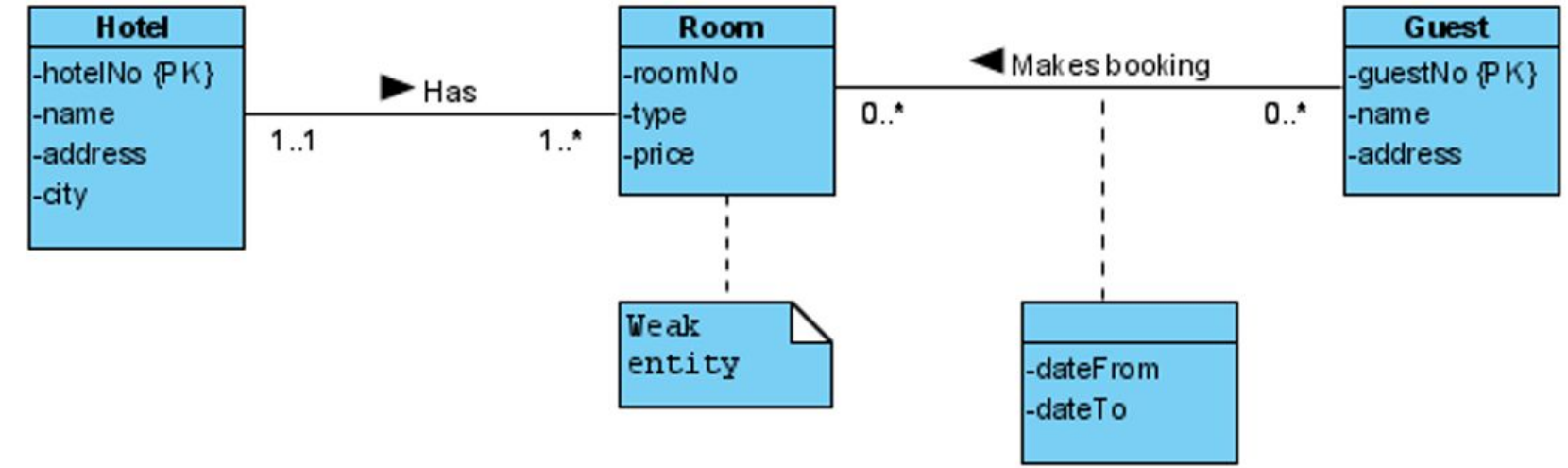
- For each **strong entity**, create relation with all simple attributes of that entity
- For **composite attributes**, include only the constituent simple attributes



- e.g., Staff (staffNo, fName, lName, position, sex, DOB)

Hotel example

CITY UNIVERSITY LONDON
EST 1894



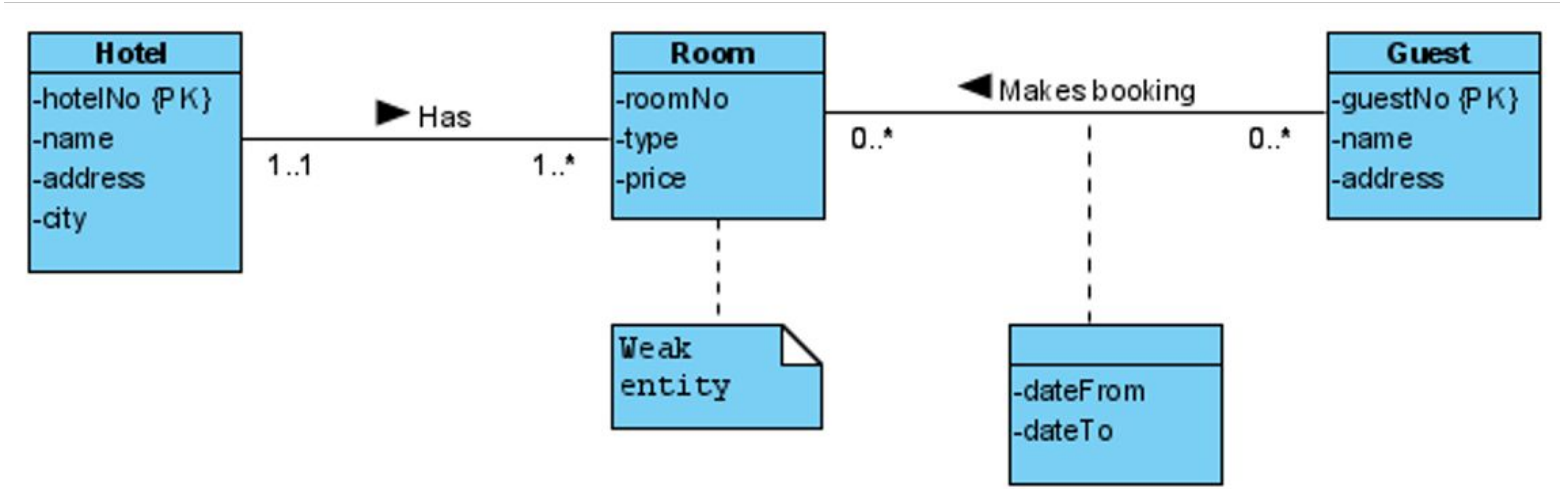
Weak Entities

CITY UNIVERSITY LONDON
EST 1894

- For each weak entity
  - create a relation with all simple attributes of that entity
- Primary key of weak entity is partially or fully derived from (strong) owner entity
  - so identification of primary key of weak entity cannot be made until all relationships with (strong) owner entity have been mapped

- Example: Hotel/Room relationship is strong/weak
  - 1:* relationship
  - Room – the weak entity; Hotel – the strong entity
  - Room (**hotelNo**, **roomNo**, …)

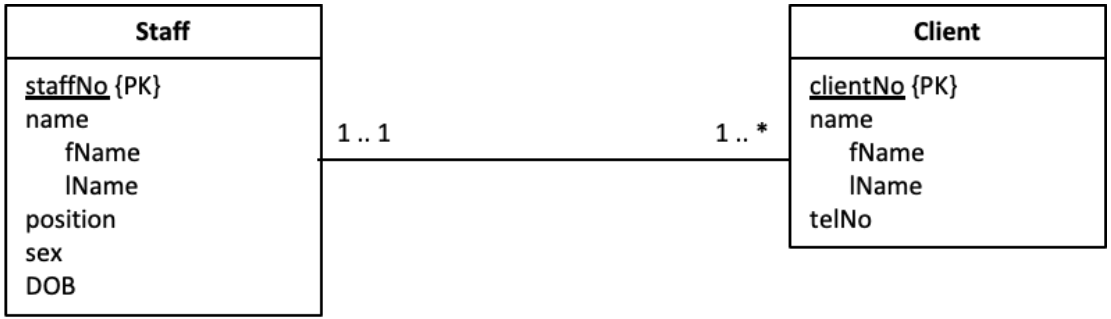
Hotel example



- Relational DBMS:
  - Hotel (**hotelNo**, name, address, city)
  - Room (**roomNo**, **hotelNo**, type, price)
  - Guest (**guestNo**, name, address)

Relationship, One-to-many, 1:*

- For each 1:* binary relationship …
  - the entity on the '**one side**' of the relationship is designated as parent
  - the entity on the '**many side**' is designated as child
- To represent this relationship …
  - post a copy of the primary key attribute(s) of parent entity into the relation representing the child entity, to act as a foreign key
- Example: Staff-Client relationship is one-to-many (1:*)



Relationship, One-to-many, 1:*

CITY UNIVERSITY LONDON
EST 1894

- For each 1:* binary relationship …
  - the entity on the '**one side**' of the relationship is designated as parent
  - the entity on the '**many side**' is designated as child
- To represent this relationship …
  - post a copy of the primary key attribute(s) of parent entity into the relation representing the child entity, to act as a foreign key
- Example: Staff-Client relationship is one-to-many (1:*)
  - PARENT: Staff (**staffNo**, fName, lName, position, sex, DOB)
  - CHILD: Client (**clientNo**, fName, lName, telNo, staffNo)
    staffNo is Foreign Key
    FOREIGN KEY staffNo references Staff (staffNo)

Relationship, One-to-many, 1:*

CITY UNIVERSITY LONDON
EST 1894

Relationships with attributes

- All relationship attributes are posted to the child relation
- As attribute for relationship will have many values

Relationship, One-to-one, 1:1

CITY UNIVERSITY LONDON
EST 1894

There two ways to represent the relation:

- combining the entities involved into one relation, or
- creating two relations and posting copy of PK from one relation to other

Consider the following options:

- 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

Relationship, One-to-one, 1:1. Case 1.

CITY UNIVERSITY LONDON

Mandatory on both sides

- Combine entities involved into one relation
- Choose one of PKs of original entities as PK of new relation
- Other PK (if one exists) is an alternate key

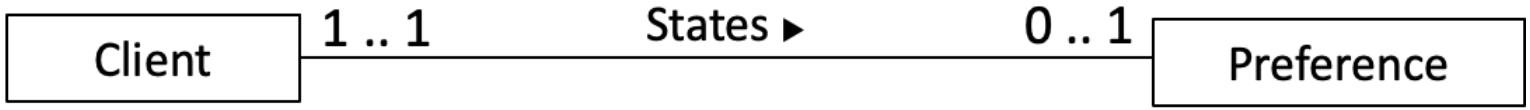
Relationship, One-to-one, 1:1. Case 2.

CITY UNIVERSITY LONDON

Mandatory on one side

- Entity type with optional participation in relationship designated as **parent**
- Entity type with mandatory participation designated as **child**
    - Create two relations
    - Copy PK of parent entity into the relation representing the child entity
    - If the relationship has one or more attributes, these should follow posting of primary key to child relation

Relationship, One-to-one, 1:1. Case 2.



- If a Client might state 0 or 1 preference (which means it is optional for a Client to state a preference)
- Client becomes parent (optional)
- Preference becomes child

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

Preference (**prefNo**, prefType, clientNo)

FOREIGN KEY clientNo references Client(clientNo)

Relationship, One-to-one, 1:1. Case 3.

CITY UNIVERSITY LONDON

Optional on both sides

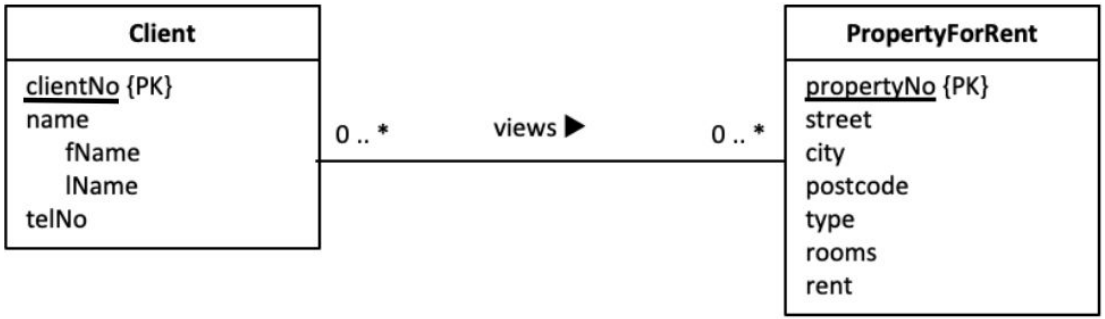
- Follow Case 2
- Create two relations
- Designation of parent-child entities is arbitrary

Relationship, Many-to-Many, *:*

CITY UNIVERSITY LONDON
EST 1894

- Create a relation to represent the relationship
- Include any attributes that are part of the relationship
- Post copy of primary key attribute(s) of participating entities into the new relation, to act as foreign keys.

- Foreign keys are (composite) primary key of new relation
- Possibly in combination with attributes of the relationship



Relationship, Many-to-Many, *:*

CITY UNIVERSITY LONDON
EST 1894

- Create a relation to represent the relationship
- Include any attributes that are part of the relationship
- Post copy of primary key attribute(s) of participating entities into the new relation, to act as foreign keys.

- Foreign keys are (composite) primary key of new relation
- Possibly in combination with attributes of the relationship

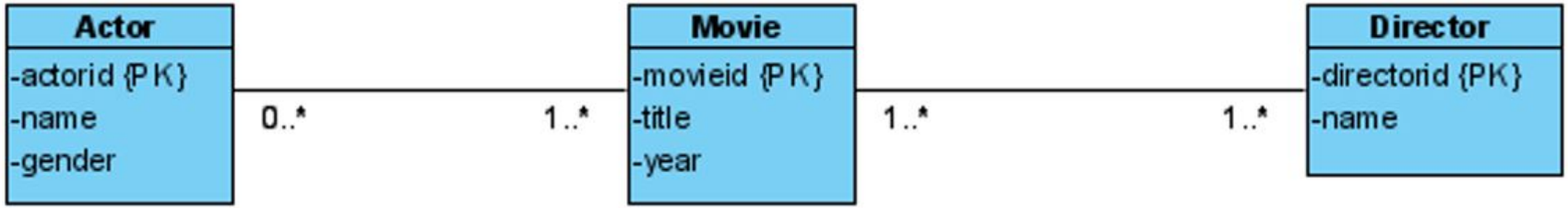
Example: **Client** Views **PropertyForRent**

Viewing (**clientNo**, **propertyNo**, dateView, comment)

Relationship, Many-to-Many, *:*

CITY UNIVERSITY LONDON
EST 1894

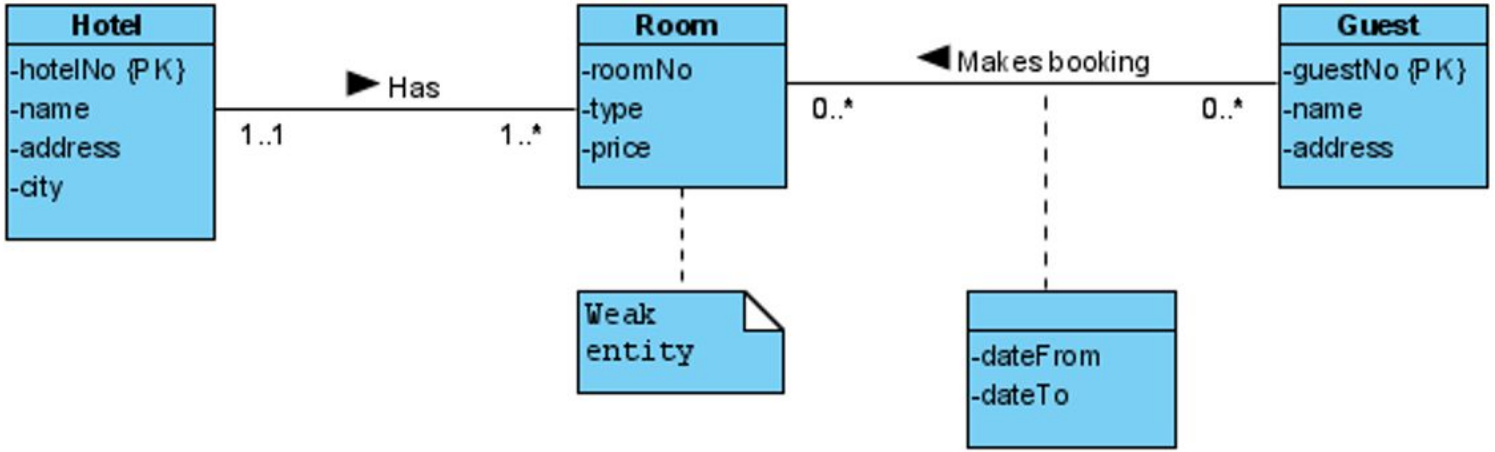
Two *:* binary relationships



- Actor (**actorid**, name, gender)
- Movie (**movieid**, title, year)
- Director (**directorid**, name)

- ActsInMovie (**movieid**, **actorid**)
- DirectsMovie (**movieid**, **directorid**)

Hotel Example



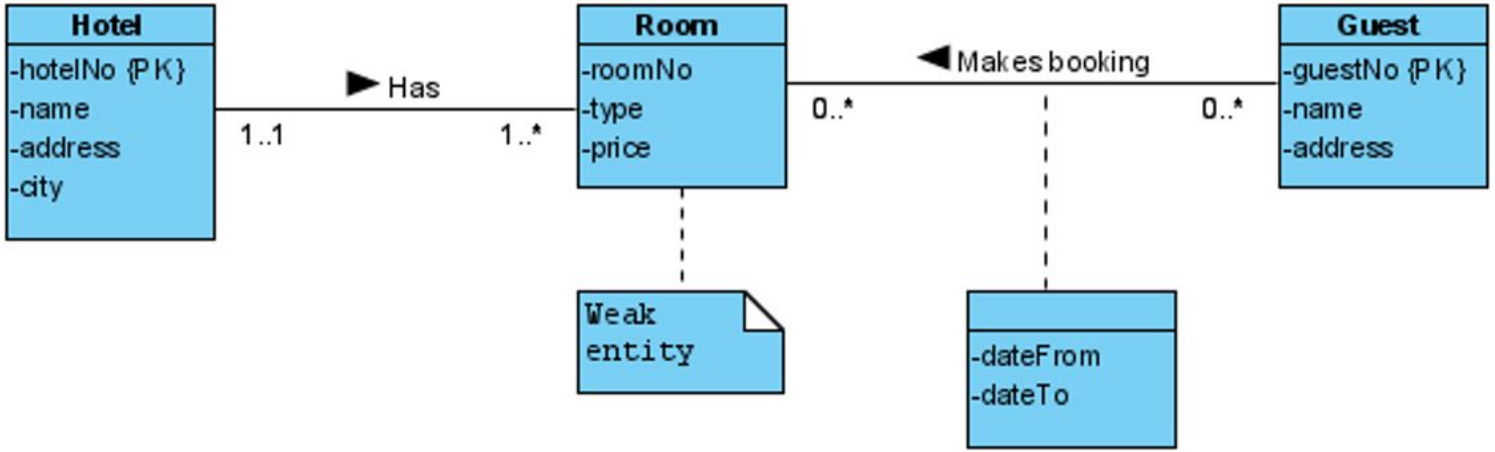
Hotel (**hotelNo**, name, address, city)

Room (**roomNo**, **hotelNo**, type, price)

Guest (**guestNo**, name, address)

Booking (**hotelNo**, **guestNo**, dateFrom, dateTo, **roomNo**)

Hotel Example



Hotel (**hotelNo**, name, address, city)

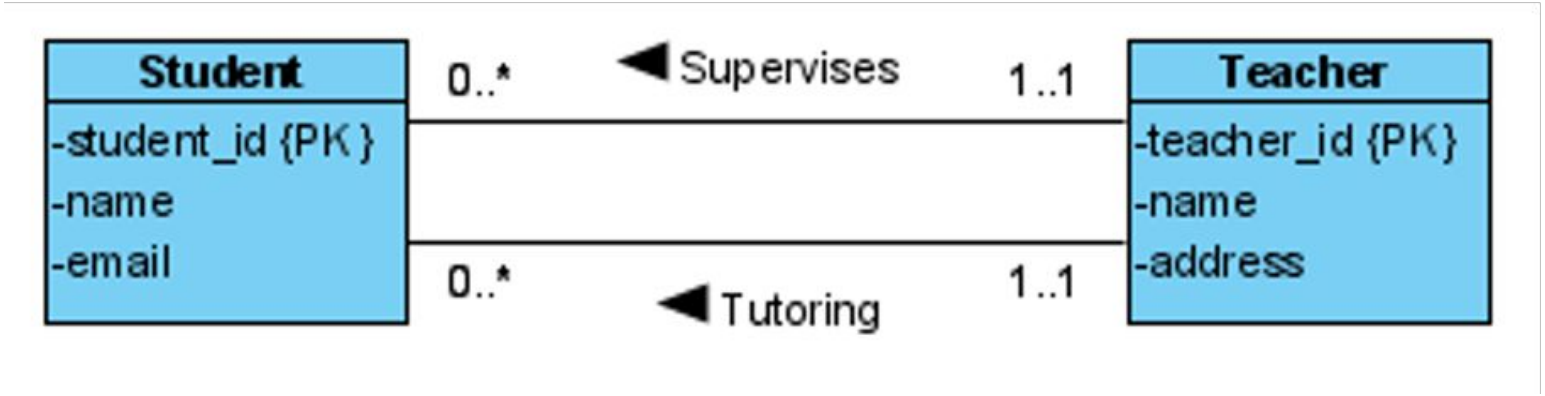
Room (**roomNo**, **hotelNo**, type, price)

Guest (**guestNo**, name, address)

Booking (**hotelNo**, **guestNo**, **dateFrom**, **dateTo**, **roomNo**)

Example. Multiple Relations.

CITY UNIVERSITY LONDON
EST 1894



Teacher (**teacher_id**, name, address)

Student (**student_id**, name, email, supervisor_id, tutor_id)
FOREIGN KEY supervisor_id references Teacher(teacher_id)
FOREIGN KEY tutor_id references Teacher(teacher_id)

S/G Superclass & subclasses

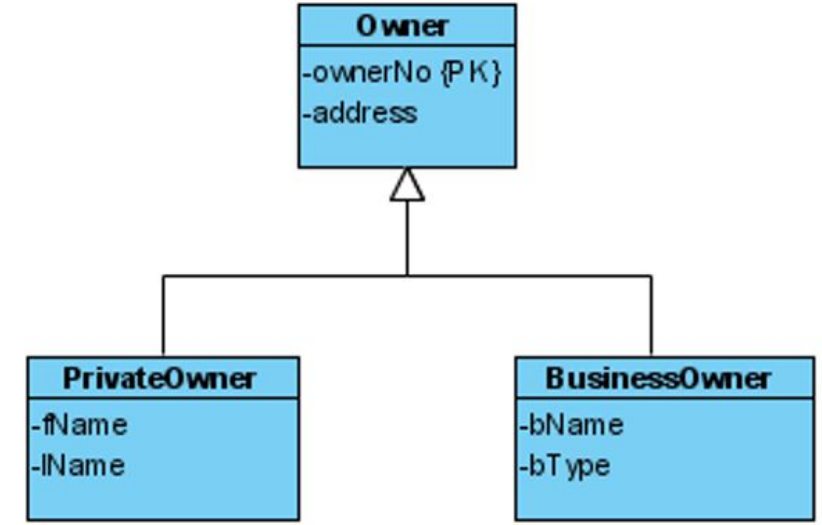
CITY UNIVERSITY LONDON
EST 1894

Many relations – one for superclass, one for each subclass

Owner (**ownerNo**, address)

PrivateOwner (**ownerNo**, fName, lName)
FOREIGN KEY ownerNo references Owner(ownerNo)

BusinessOwner (**ownerNo**, bName, bType)
FOREIGN KEY ownerNo references Owner(ownerNo)



Normalization

Purpose of Normalization

CITY UNIVERSITY LONDON
EST 1894

- The minimal number of attributes necessary to support the data requirements
- Attributes with a close logical relationship are in the same relation
- Minimal redundancy, each attribute appears once (except for the foreign keys)

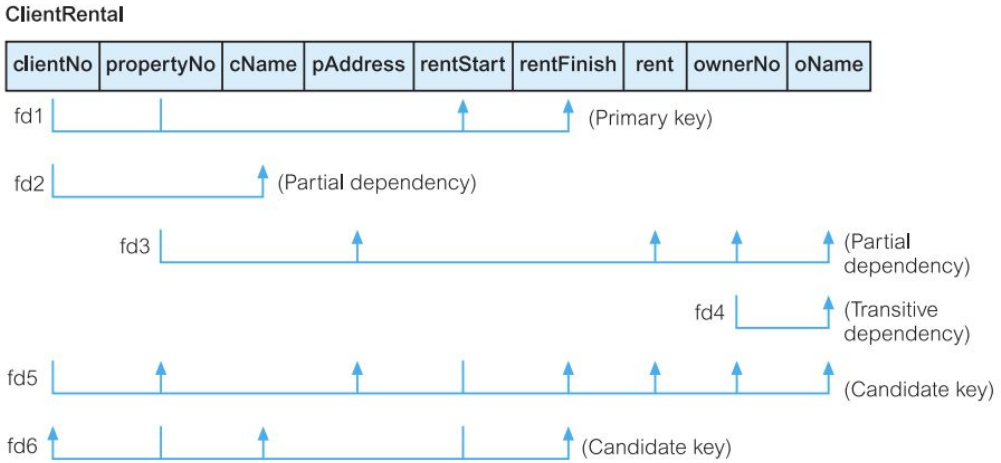
There are lots of normal forms:

- 1, 2, 3 - Normal Forms
- Boyce-Codd Normal Form
- 4, 5 - Normal Forms

Functional Dependencies

CITY UNIVERSITY LONDON
EST 1894

- In one relation, several attributes depend on some of attributes. This is the functional dependency.
- For example, a dependency from primary key. (Which is fine)
- Our main goal for now is to get rid of complex function dependencies
- A→B is full dependency if you cannot throw away any attribute from A



1NF

CITY UNIVERSITY LONDON

- No repeating groups
- All attributes are atomic
- Each relation has a key

1NF

CITY UNIVERSITY LONDON
EST 1894

- No repeating groups
- All attributes are atomic
- Each relation has a key

CId	**Lecturer**	**Phone**	**Phone2**
1	Jason Dykes	11111111	55555555
2	May Elshehaly	12312312	
3	Asad Hafisul	4444444	88888888

1NF

CITY UNIVERSITY LONDON
EST 1894

- No repeating groups
- All attributes are atomic
- Each relation has a key

CId	Lecturer	Phone
1	Jason Dykes	11111111
		55555555
2	May Elshehaly	12312312
3	Asad Hafisul	4444444
		88888888

1NF. Final result

CITY UNIVERSITY LONDON
EST 1894

Two ways:

- Fill in with appropriate data by copying
- Create another relation

CId	Lecturer	Phone
1	Jason Dykes	11111111
1	Jason Dykes	55555555
2	May Elshehaly	12312312
3	Asad Hafisul	4444444
3	Asad Hafisul	88888888

Anomalies

CITY UNIVERSITY LONDON

- It is hard to maintain the consistency in the database if the relations are complicated
- Three types:
  - Insertion anomalies
  - Deletion anomalies
  - Modification anomalies

Insertion Anomaly

CITY UNIVERSITY LONDON

- The function relation stops us from writing a data without ALL data
- We cannot store the lecturer without the course

CId	Lecturer	Phone
1	Jason Dykes	11111111
?	Vladimir Stankovich	12312312

Deletion Anomaly

CITY UNIVERSITY LONDON
EST 1894

- Deletion of one information leads to the deletion of another
- When we delete the last course we lose the telephone number

CId	Lecturer	Phone
1	Jason Dykes	11111111
2	Vladimir Stankovich	12312312

Modification Anomaly

CITY UNIVERSITY LONDON

- The partial change of the information is not propagated everywhere and, then, corrupts the database
- If we change the phone of one lecturer in one place - it is not changed elsewhere

CId	Lecturer	Phone
1	Jason Dykes	11111111
2	Jason Dykes	12312312

Functional Dependencies and 2NF

CITY UNIVERSITY LONDON
EST 1894

CId	**Year**	**Lecturer**	**Exam**
1	2022	Jason Dykes	yes
2	2023	May Elshehaly	no
3	2020	Chris Smart	no
3	2023	Asad Hafisul	no

Which Functional Dependencies we have?

- CId → Exam
- CId Year → Lecturer

What to do: get rid of non-key dependencies that have part of the primary key

2NF. Final result

CITY UNIVERSITY LONDON
EST 1894

CId	Year	Lecturer	Exam
1	2022	Jason Dykes	yes
2	2023	May Elshehaly	no
3	2020	Chris Smart	no
3	2023	Asad Hafisul	no

CId	Exam
1	yes
2	no
3	no

CId	Year	Lecturer
1	2022	Jason Dykes
2	2023	May Elshehaly
3	2020	Chris Smart
3	2023	Asad Hafisul

Functional Dependencies and 2NF

CITY UNIVERSITY LONDON
EST 1894

CId	**Year**	**Lecturer**	**Phone**
1	2022	Jason Dykes	1111111
2	2023	May Elshehaly	2222222
3	2020	Chris Smart	3333333
3	2023	Vladimir Stankovich	444444
1	2023	Jason Dykes	1111111

Which Functional Dependencies we have?

- CId Year → Lecturer
- Lecturer → Phone

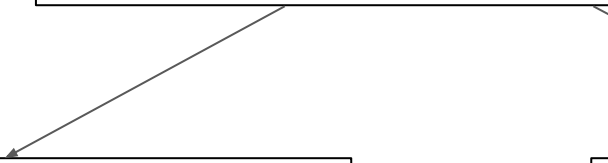
What to do: get rid of transitive dependencies

Algorithm: just decompose over the latest dependency in the order

3NF. Final result

CITY UNIVERSITY LONDON
EST 1894

CId	Year	Lecturer	Phone
1	2022	Jason Dykes	1111111
2	2023	May Elshehaly	2222222
3	2020	Chris Smart	3333333
3	2023	Vladimir Stankovich	444444
1	2023	Jason Dykes	1111111



CId	Year	Lecturer
1	2022	Jason Dykes
2	2023	May Elshehaly
3	2020	Chris Smart
3	2023	Vladimir Stankovich
1	2023	Jason Dykes

Lecturer	Phone
Jason Dykes	1111111
May Elshehaly	2222222
Chris Smart	3333333
Vladimir Stankovich	444444