Chapter 17

Methodology
Logical Database Design for the Relational Model

UiA

Purpose of Logical Desing

- Translate conceptual model into internal model
- Map objects in conceptual model to relational model
- Design components
  - Tables
  - Indexes
  - Views
  - Transactions

UiA

Chapter 17 - Objectives

- How to derive a set of relations from a conceptual data model.
- How to validate these relations using the technique of normalization.
- How to validate a logical data model to ensure it supports the required transactions.
- How to merge local logical data models based on one or more user views into a global logical data model that represents all user views.
- How to ensure that the final logical data model is a true and accurate representation of the data requirements of the enterprise.

Deliverables and outcome

- Logical database desing must account for every element of a system input and output.
- Normalized relations are the primary deliverable
- Next step (physical database design) is transforming relations into files for target database system.

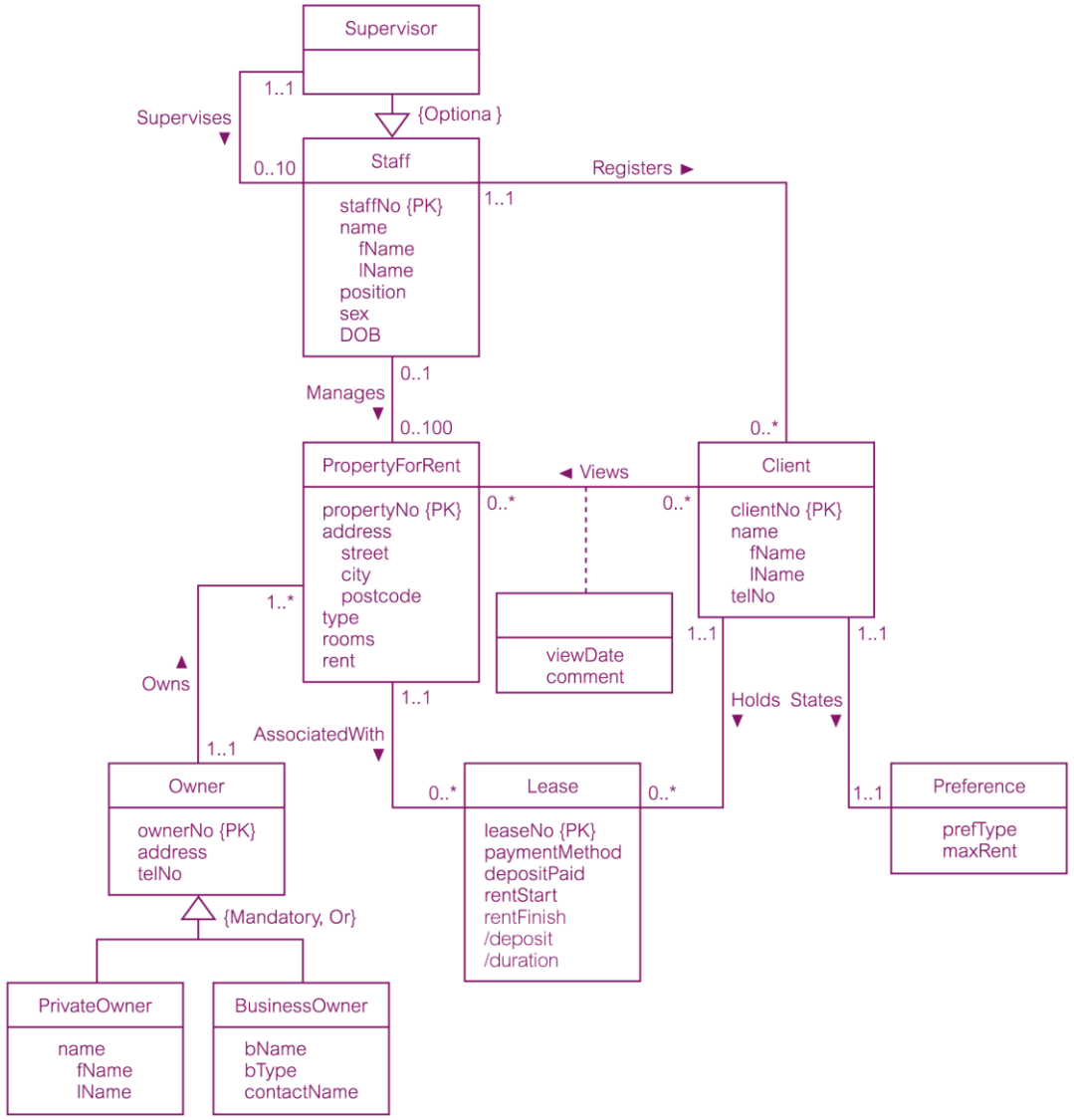
Step 2 Build and Validate Logical Data Model

- **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 Build and Validate Logical Data Model

- **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.**

Conceptual data model for Staff view showing all attributes



Step 2.1 Derive relations for logical data model

- The following structures may occur in a conceptual data model:
  - (1) strong entity types
  - (2) weak entity types
  - (3) one-to-many (1:*) binary relationship types
  - (4) one-to-one (1:1) binary relationship types
  - (5) one-to-one (1:1) recursive relationship types
  - (6) superclass/subclass relationship types
  - (7) many-to-many (*:*) binary relationship types
  - (8) complex relationship types
  - (9) multi-valued attributes.

Step 2.1 Derive relations for logical data model

(1)Strong entity types

Definition: an entity type that is not existence-dependent on some other another entity

- 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.
- Ex Staff
  - Staff(staffNo, position, sex, DOB)
  - Staff(staffNo, fName, lName, position, sex, DOB)
  - Primary key: staffNo

UiA

Step 2.1 Derive relations for logical data model

(2) Weak entity types
Definition: an entity type that is existence-dependent on some other entity type

- For each weak entity in the data model, create a relation that includes all the simple attributes of that entity.
  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.
- Ex: Preference
- Preference(prefType, maxRent)
- Primary key: None (can not be identified at the moment)

UiA

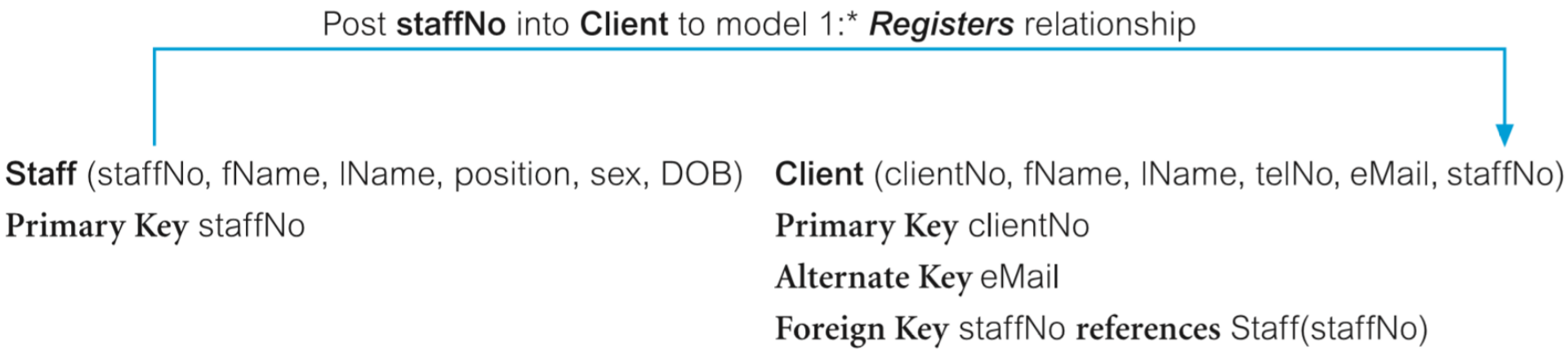
Step 2.1 Derive relations for logical data model

- **(3) 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, 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.
  - “Add the primary key (attribute or attributes) of the entity on one side of the relationship as the foreign key on the many side”.
  - The one side migrates to the many side.
  - Example next slide

One-to-many (1:*) binary relationship types

- Staff 1:* Client



Step 2.1 Derive relations for logical data model

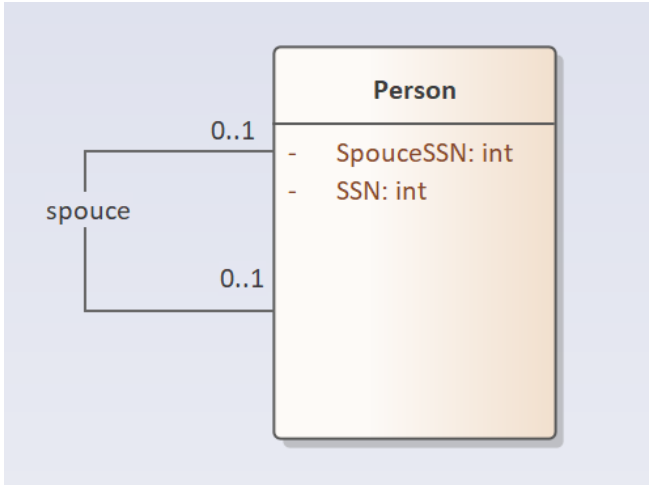
- **(4) One-to-one (1:1) binary relationship types**
  - Creating relations to represent a 1:1 relationship is more complex as the cardinality cannot be used to identify the parent and child entities in a relationship.
    Instead, the participation constraints are used to decide whether it is best to 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.
  - Consider the following
    - *(a) mandatory* participation on *both* sides of 1:1 relationship;
    - *(b) mandatory* participation on *one* side of 1:1 relationship;
    - *(c) optional* participation on *both* sides of 1:1 relationship.

Step 2.1 Derive relations for logical data model

- **(a) *Mandatory* participation on *both* sides of 1:1 relationship**
  - Combine entities involved into one relation and choose one of the primary keys of original entities to be primary key of the new relation, while the other (if one exists) is used as an alternate key.

One-to-one (1:1) binary relationship types

- Example:
  - Client 1:1 Prefenrece
  - a) Both sides: spouce



  - a) Both sides but duplicating data:

Client (clientNo, fName, lName, telNo, eMail, prefType, maxRent, staffNo)
Primary Key clientNo
Alternate Key eMail
Foreign Key staffNo **references** Staff(staffNo)

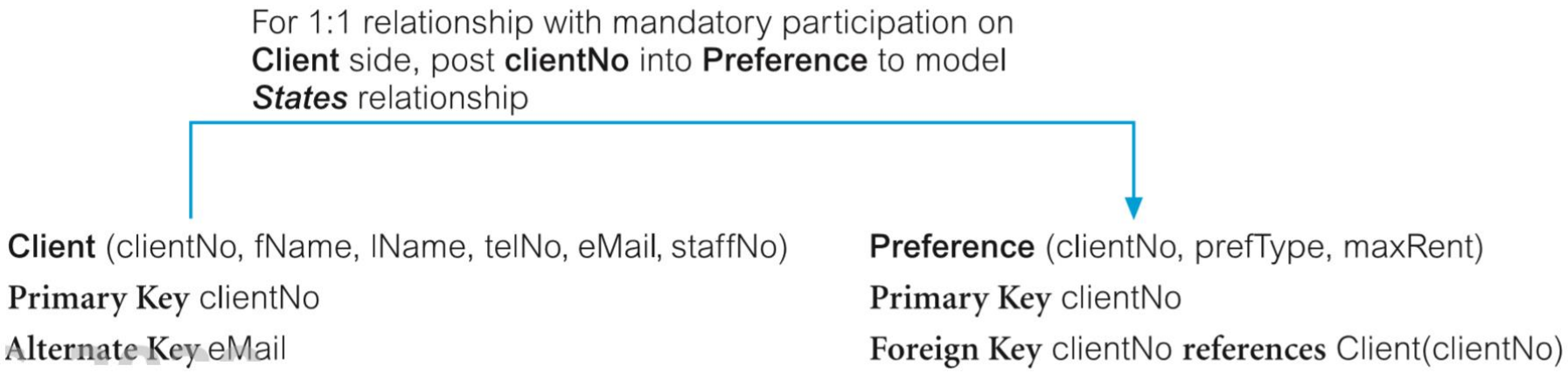
```
Preference(clientNo, prefType, maxRent)
Primary Key clientNo
Alternate Key None
Foreign Key clientNo references Client(clientNo)
```

Step 2.1 Derive relations for logical data model

- **(b) *Mandatory* participation on *one* side of a 1:1 relationship**
  - Identify parent and child entities using participation constraints. Entity with optional participation in relationship is designated as parent entity, and entity with mandatory participation is designated as child entity.
    A copy of 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.

One-to-one (1:1) binary relationship types

- Example
- b)



Step 2.1 Derive relations for logical data model

- **(c) *Optional* participation on *both* sides of a 1:1 relationship**
  - In this case, the designation of the parent and child entities is arbitrary unless we can find out more about the relationship that can help a decision to be made one way or the other.

Step 2.1 Derive relations for logical data model

- **(5) One-to-one (1:1) recursive relationships**
  - For a 1:1 recursive relationship, follow the rules for participation as described above for a 1:1 relationship.
    - mandatory participation on both sides, represent the recursive relationship as a single relation with two copies of the primary key.
    - mandatory participation on only one side, option to create a single relation with two copies of the primary key, or to create a new relation to represent the relationship. The new relation would only have 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, again create a new relation as described above.

Step 2.1 Derive relations for logical data model

- **(6) Superclass/subclass relationship types**
  - Identify superclass entity as parent entity and 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.
  - Disjoint – when a superclass has more than one subclass

Guidelines for representation of superclass / subclass relationship

Participation constraint	Disjoint constraint	Relations required
Mandatory	Nondisjoint {And}	Single relation (with one or more discriminators to distinguish the type of each tuple)
Optional	Nondisjoint {And}	Two relations: one relation for superclass and one relation for all subclasses (with one or more discriminators to distinguish the type of each tuple)
Mandatory	Disjoint {Or}	Many relations: one relation for each combined superclass/subclass
Optional	Disjoint {Or}	Many relations: one relation for superclass and one for each subclass

Representation of superclass / subclass relationship based on participation and disjointness

Option 1 – Mandatory, nondisjoint

AllOwner (ownerNo, address, telNo, fName, lName, bName, bType, contactName, pOwnerFlag, bOwnerFlag)

Primary Key ownerNo

Option 2 – Optional, nondisjoint

Owner (ownerNo, address, telNo)

Primary Key ownerNo

OwnerDetails (ownerNo, fName lName, bName, bType, contactName, pOwnerFlag, bOwnerFlag)

Primary Key ownerNo

Foreign Key ownerNo **references** Owner(ownerNo)

Option 3 – Mandatory, disjoint

PrivateOwner (ownerNo, fName, Name, address, telNo)

Primary Key ownerNo

BusinessOwner (ownerNo, bName, bType, contactName, address, telNo)

Primary Key ownerNo

Option 4 – Optional, disjoint

Owner (ownerNo, address, telNo)

Primary Key ownerNo

PrivateOwner (ownerNo, fName, Name)

Primary Key ownerNo

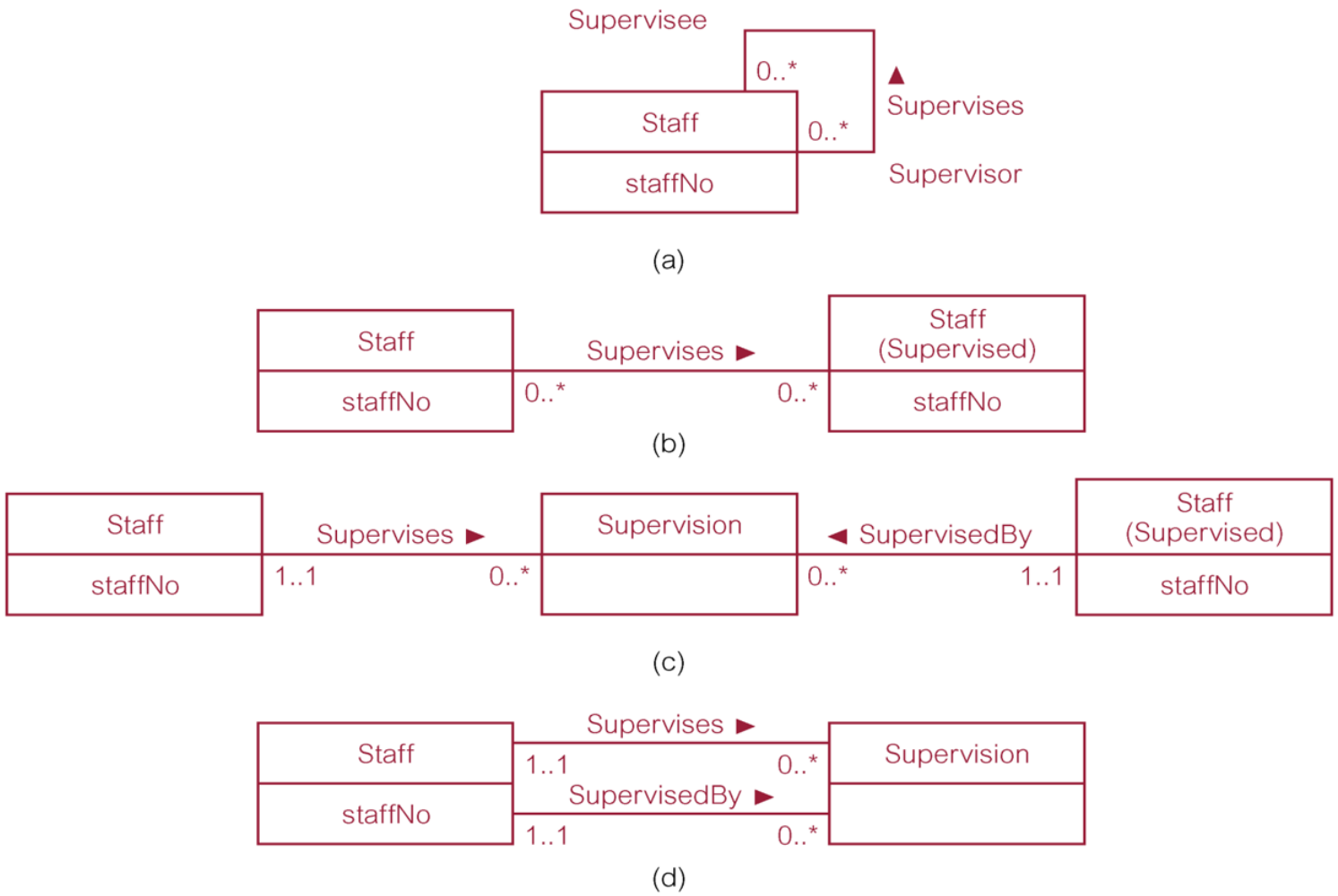
Foreign Key ownerNo **references** Owner(ownerNo)

BusinessOwner (ownerNo, bName, bType, contactName)

Primary Key ownerNo

Foreign Key ownerNo **references** Owner(ownerNo)

Supervisor/Staff superclass/subclass relationship



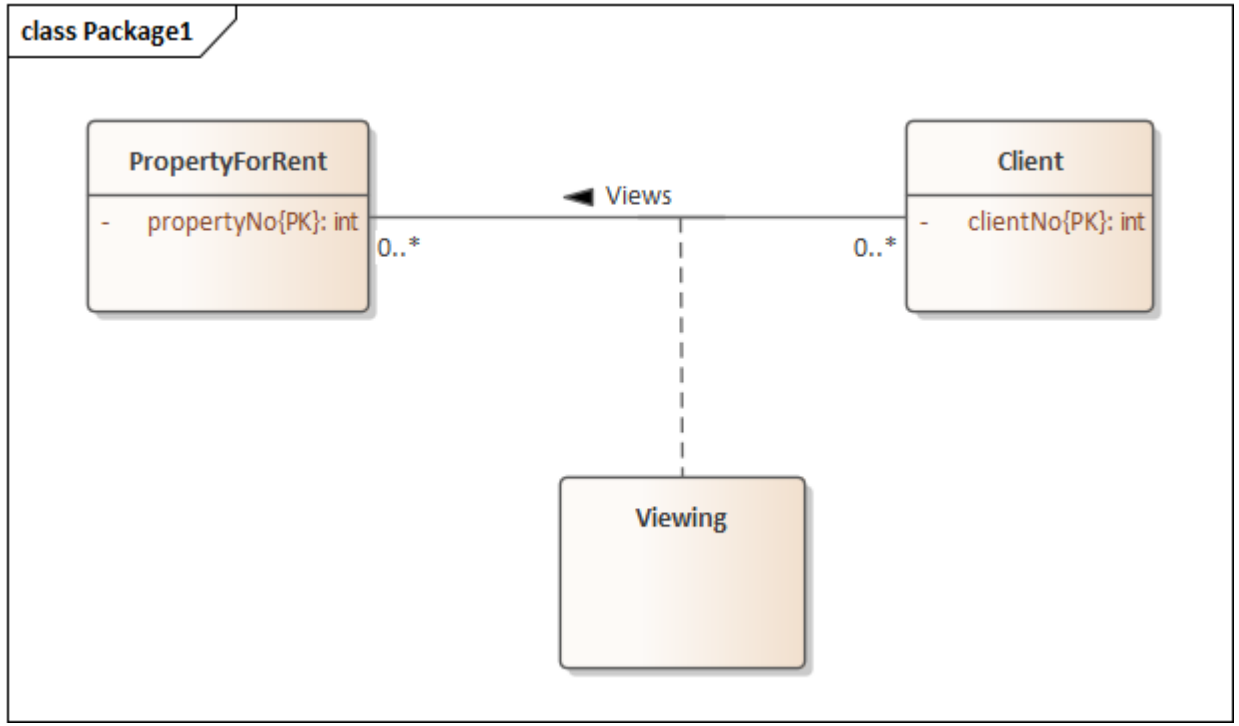
Step 2.1 Derive relations for logical data model

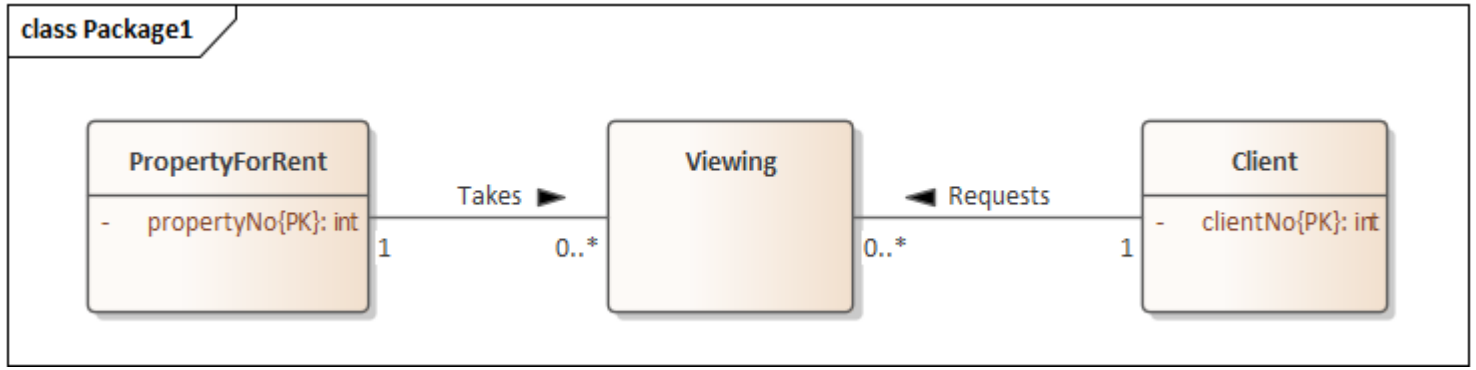
- **(7) Many-to-many (*:*) binary relationship types**
  - Create a relation to represent the relationship and include any attributes that are part of the relationship.

    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.

    These foreign keys will also form the primary key of the new relation, possibly in combination with some of the attributes of the relationship.

Views


class Package1
PropertyForRent
- propertyNo{PK}: int
Views
0..*
0..*
Client
- clientNo{PK}: int
Viewing


class Package1
PropertyForRent
- propertyNo{PK}: int
Takes
1
0..*
Viewing
Requests
0..*
1
Client
- clientNo{PK}: int

Views

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

Primary Key clientNo

Alternate Key eMail

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

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

Primary Key propertyNo

Viewing (clientNo, propertyNo, dateView, comment)

Primary Key clientNo, propertyNo

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

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

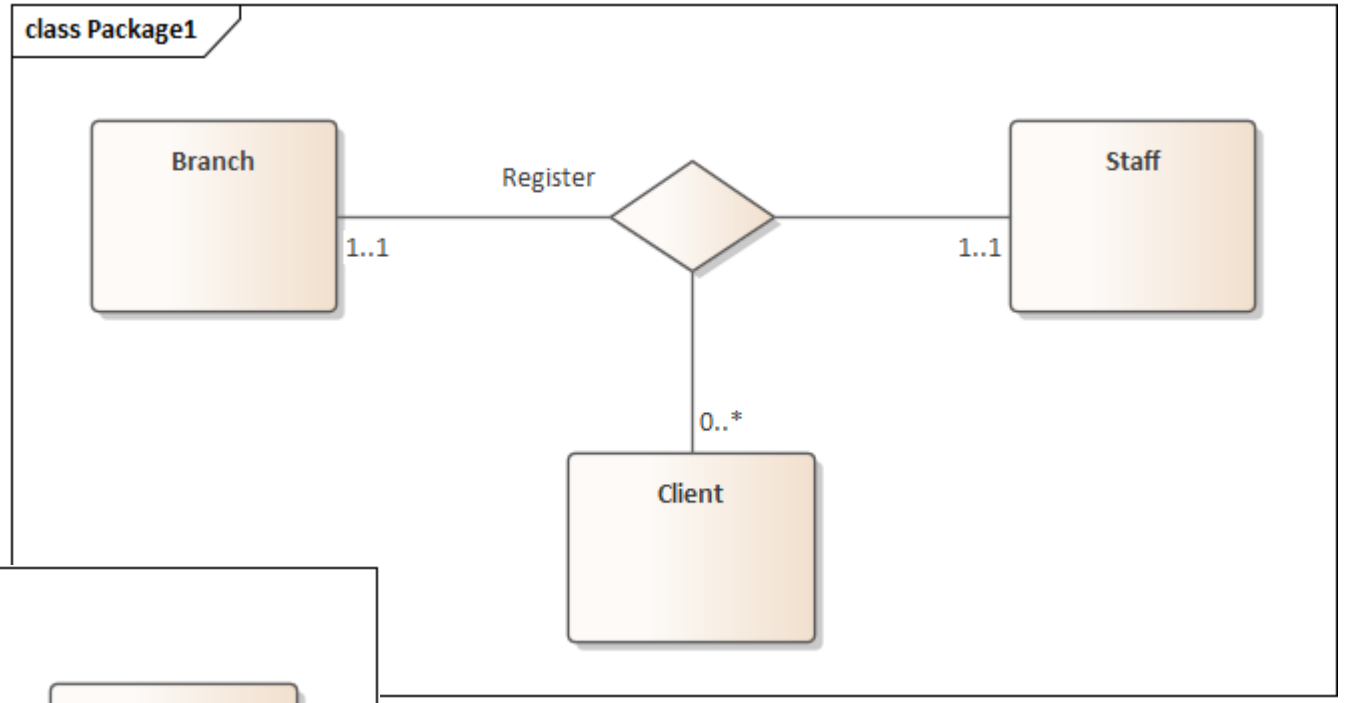
Step 2.1 Derive relations for logical data model

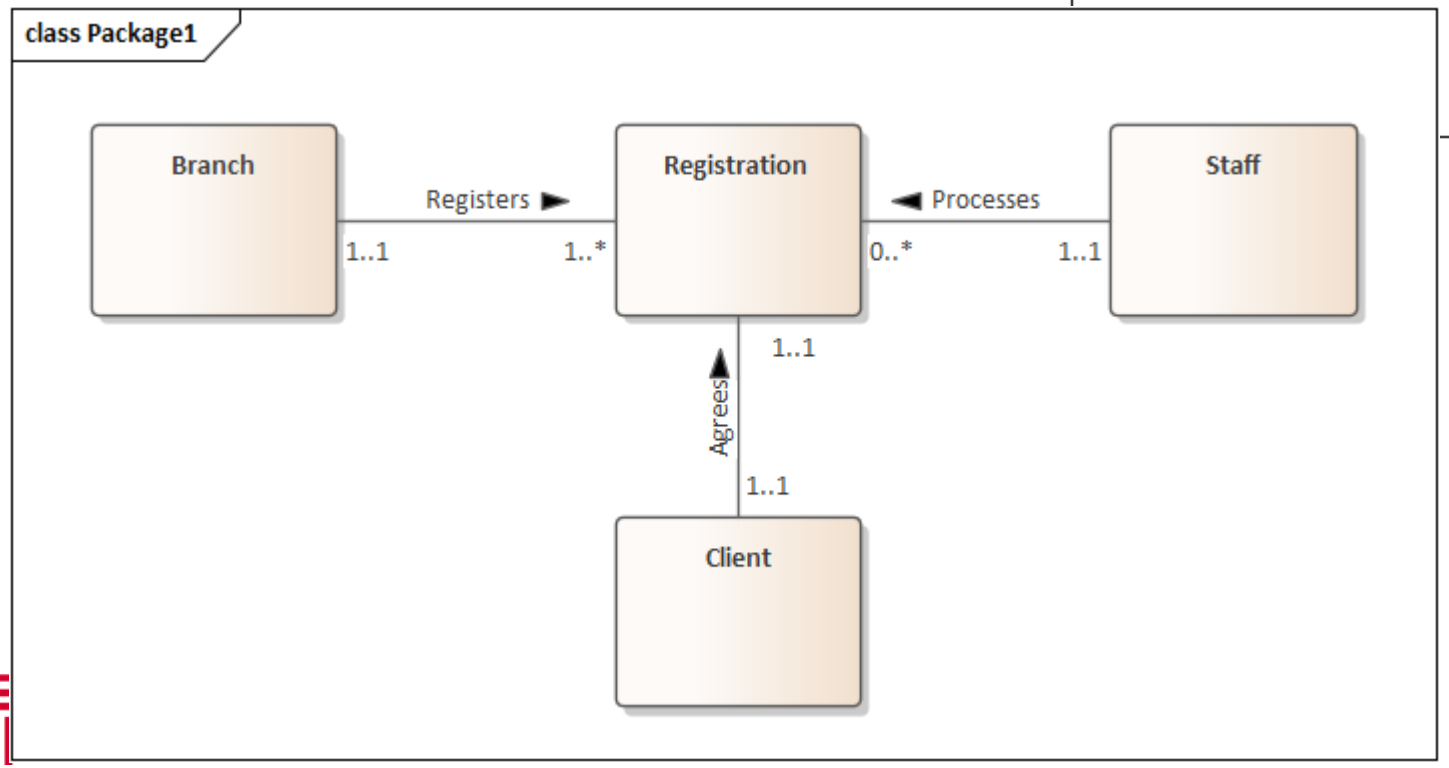
- **(8) Complex relationship types**
  - Create a relation to represent the relationship and include any attributes that are part of the relationship.

    Post a copy of the primary key attribute(s) of the entities that participate in the 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.

Register


class Package1
Branch
Register
1..1
Staff
1..1
0..*
Client

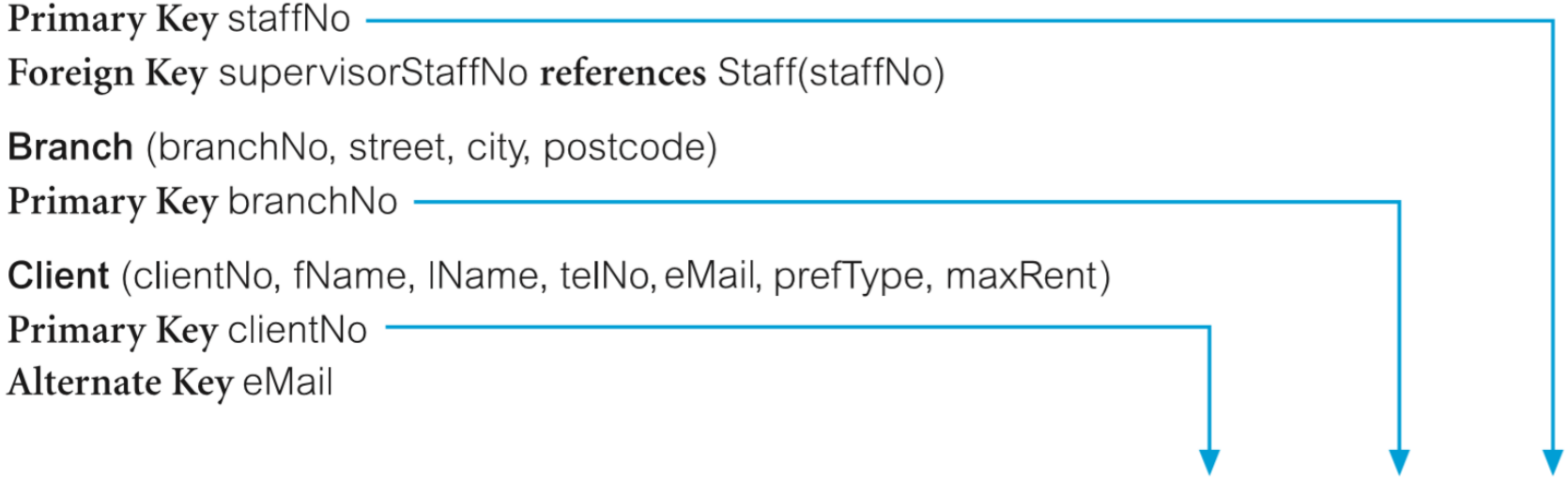

class Package1
Branch
Registers
1..1
1..*
Registration
Processes
0..*
Staff
1..1
1..1
Agrees
1..1
Client

Registration

Staff (staffNo, fName, lName, position, sex, DOB, supervisorStaffNo)
Primary Key staffNo
Foreign Key supervisorStaffNo **references** Staff(staffNo)

Branch (branchNo, street, city, postcode)
Primary Key branchNo

Client (clientNo, fName, lName, telNo, eMail, prefType, maxRent)
Primary Key clientNo
Alternate Key eMail



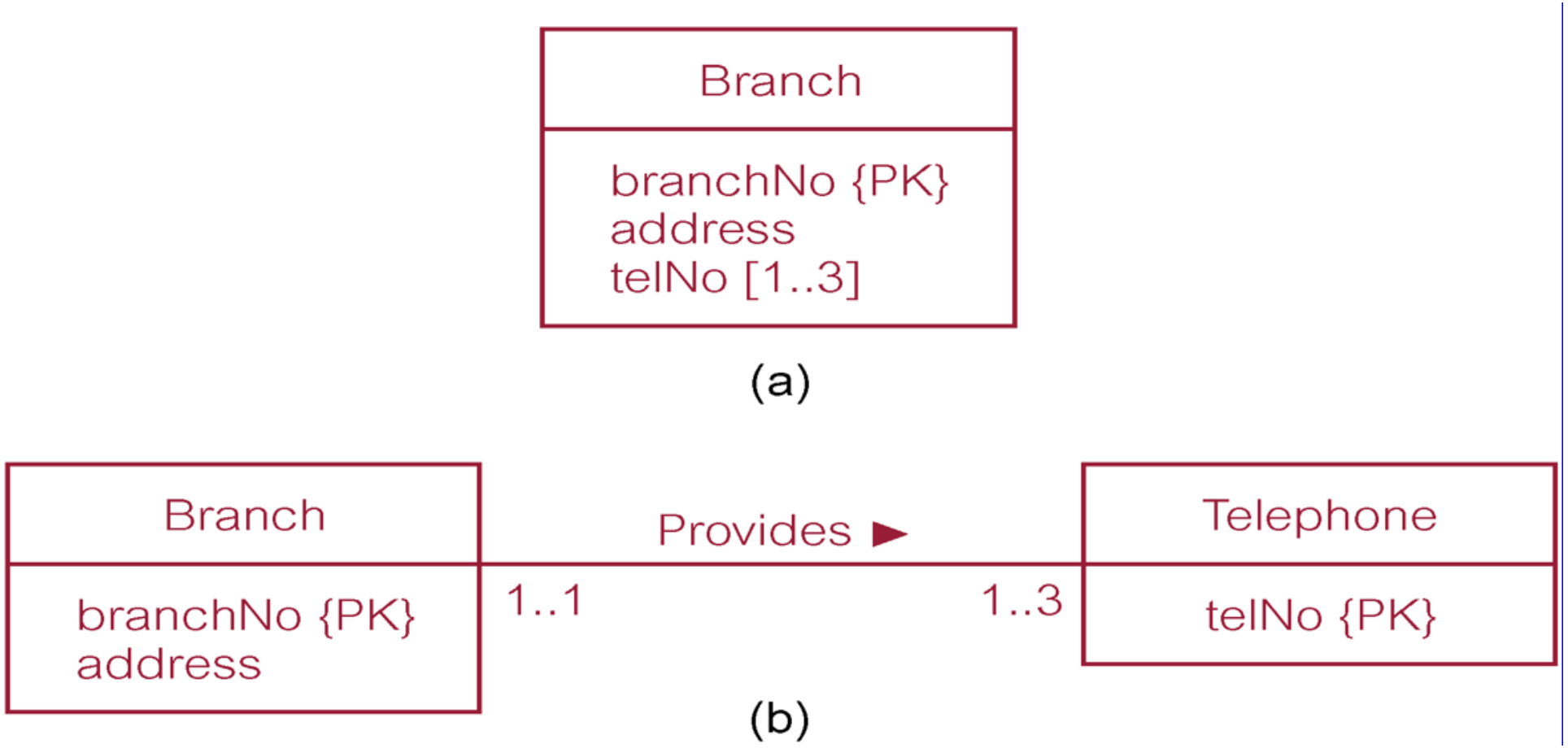
Registration (clientNo, branchNo, staffNo, dateJoined)
Primary Key clientNo, branchNo
Foreign Key branchNo **references** Branch(branchNo)
Foreign Key clientNo **references** Client(clientNo)
Foreign Key staffNo **references** Staff(staffNo)

Step 2.1 Derive relations for logical data model

- **(9) Multi-valued attributes**
  - Create a new relation to represent multi-valued attribute and include primary key of entity in 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.

Branch/Telephone



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.
Superclass/subclass relationship	See Table 16.1.
: 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

- To validate the relations in the logical data model using normalization.
- The logical design may not be the final design. It should represent the database designer's best understanding of the nature and meaning of the data required by the enterprise.
- A normalized design is robust and free of the update anomalies.
- It is sometimes reasonable to implement a design that gains ease of use at the expense of additional processing.

Step 2.3 Validate relations against user transactions

- To ensure that the relations in the logical data model support the required transactions.
- Same check as was done in step 1.8.
- Check whether the relations created in step 2.1 supports the transactions.
- Usually performed manually/on paper.
- Able to perform all required transactions – all good
- Not able to perform all required transactions – not good, must revisit both conceptual and logical model

Step 2.4 Check integrity constraints

- To check integrity constraints are represented in the logical data model. This includes identifying:
  - **Required data – must have value, not null**
  - **Attribute domain constraints – sex, M, F**
  - **Multiplicity – constraint, branch have more than 3 phone numbers, branch has many staff members and staff works at single branch.**
  - **Entity integrity – primary key cannot hold null**
  - **Referential integrity – primary key of foreign key must exist, must be of same type**
  - **General constraints – ex staff not managing more than 100 properties.**

Delete tuple from parent relation

- NO ACTION – prevent the deletion
- CASCADE – delete any referenced child tuples
- SET NULL – when a parent tuple is deleted, set FK to null
- SET DEFAULT – when a parent tuple is deleted, set FK to default value
- NO CHECK – do nothing to ensure referencial integrity.

Referential integrity constraints for relations in Staff user views of *DreamHome*

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

Step 2.5 Review logical data model with user

- **To review the logical data model with the users to ensure that they consider the model to be a true representation of the data requirements of the enterprise.**

Step 2.6 Merge logical data models into global Model (optional step)

- **To merge logical data models into a single global logical data model that represents all user views of a database.**

Step 2.6.1 Merge local logical data models into global model

- **To merge local logical data model into a single global logical data model.**

- **This activities in this step include:**
  - **Step 2.6.1 Merge local logical data models into global model**
  - **Step 2.6.2 Validate global logical data model**
  - **Step 2.6.3 Review global logical data model with users.**

Step 2.6.1 Merge logical data models into a global model

- **Tasks typically includes:**
  - **(1) Review the names and contents of entities/relations and their candidate keys.**
  - **(2) Review the names and contents of relationships/foreign keys.**
  - **(3) Merge entities/relations from the local data models**
  - **(4) Include (without merging) entities/relations unique to each local data model**
  - **(5) Merge relationships/foreign keys from the local data models.**
  - **(6) Include (without merging) relationships/foreign keys unique to each local data model.**
  - **(7) Check for missing entities/relations and relationships/foreign keys.**
  - **(8) Check foreign keys.**
  - **(9) Check Integrity Constraints.**
  - **(10) Draw the global ER/relation diagram**
  - **(11) Update the documentation.**

Step 2.6.2 Validate global logical data model

- **To validate the relations created from the global logical data model using the technique of normalization and to ensure they support the required transactions, if necessary.**

Step 2.6.3 Review global logical data model with users

- **To review the global logical data model with the users to ensure that they consider the model to be a true representation of the data requirements of an enterprise.**

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

Relations that represent the global logical data model for *DreamHome*

Branch (branchNo, street, city, 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, fName, lName, position, sex, DOB, 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, fName, lName, address, telNo) **Primary Key** ownerNo	**BusinessOwner** (ownerNo, bName, bType, contactName, address, telNo) **Primary Key** ownerNo **Alternate Key** bName **Alternate Key** telNo
PropertyForRent (propertyNo, street, city, postcode, type, rooms, rent, ownerNo, staffNo, branchNo) **Primary Key** propertyNo **Foreign Key** ownerNo **references** PrivateOwner(ownerNo) and BusinessOwner(ownerNo) **Foreign Key** staffNo **references** Staff(staffNo) **Foreign Key** branchNo **references** Branch(branchNo)	**Viewing** (c ientNo, propertyNo, dateView, comment) **Primary Key** clientNo, propertyNo **Foreign Key** clientNo **references** Client(clientNo) **Foreign Key** propertyNo **references** PropertyForRent(propertyNo)
Client (clientNo, fName, lName, telNo, prefType, maxRent) **Primary Key** clientNo	**Registration** (clientNo, branchNo, staffNo, dateJoined) **Primary Key** clientNo **Foreign Key** clientNo **references** Client(clientNo) **Foreign Key** branchNo **references** Branch(branchNo) **Foreign Key** staffNo **references** Staff(staffNo)
Lease (leaseNo, paymentMethod, depositPaid, rentStart, rentFinish, cl entNo, propertyNo) **Primary Key** easeNo **Alternate Key** propertyNo, rentStart **Alternate Key** c ientNo, rentStart **Foreign Key** clientNo **references** Client(clientNo) **Foreign Key** propertyNo **references** PropertyForRent(propertyNo) **Derived** depos t (PropertyForRent.rent*2) **Derived** duration (rentFinish – rentStart)	**Newspaper** (newspaperName, address, telNo, contactName) **Primary Key** newspaperName **Alternate Key** telNo
Advert (propertyNo, newspaperName, dateAdvert, cost) **Primary Key** propertyNo, newspaperName, dateAdvert **Foreign Key** propertyNo **references** PropertyForRent(propertyNo) **Foreign Key** newspaperName **references** Newspaper(newspaperName)	

Global relation diagram for *DreamHome*