Chapter 17
Logical Database Design for the Relational Model

1

COMP211

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 user 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.

2

COMP211

Logical Database Design Methodology

- The objective is to translate the conceptual data model into a logical data model and then to validate this model to check that it is structurally correct and able to support the required transactions.

Step 2: Build Logical Data Model

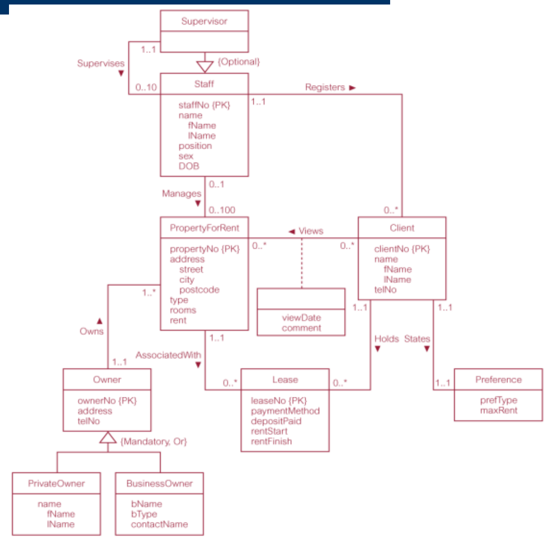
Activities involved:

- 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 Check integrity constraints
- Step 2.5 Review logical data model with user
- Step 2.6 Merge logical data models into global data model (optional)
- Step 2.7 Check for future growth

Step 2.1 Derive Relations for Logical Data Model

- Objective:

  To create relations for the logical data model to represent the entities, relationships, and attributes that have been identified in Fig 17.1.



Step 2.1 Derive Relations for Logical Data Model (cont'd)

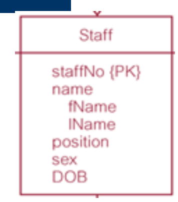
- We describe how relations are derived for the following structures that may occur in a conceptual data model:
  1) Strong entity types;
  2) Weak entity types;
  3) One-to-many (1:*) binary relationship types;
  4) One-to-one (1:1) binary relationship types;
  5) One-to-one(1:1) recursive relationship 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 Strong and Weak entity types

1) Strong entity types:

Create relation that includes all simple attributes;

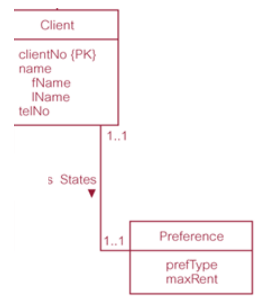
e.g. Staff (<u>staffNo</u>, fName, lName, position, sex, DOB)



2) Weak entity types:

Create relation that includes all simple attributes (primary key has to be identified after the relationship with each owner entity has been mapped)

e.g. **Preference (prefType, maxRent)** with unknown PK yet until after the States relationship has been appropriately mapped.



7

COMP211

Step 2.1 Derive Relations for One-to-many (1:*) binary relationship types

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

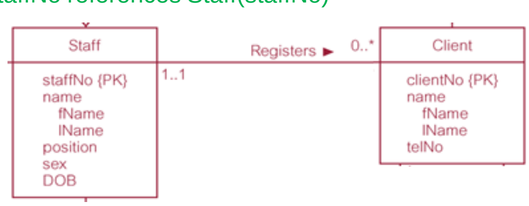
- The entity on the "one side" of the relationship is the parent entity and the entity on the "many side" is the child entity.
- Post primary key of entity on the "one" side to act as foreign key in relation representing entity on the "many" side.

e.g. **Staff registers Client relationship**

Staff (<u>staffNo</u>, fName, lName, position, sex, DOB)

Client (<u>clientNo</u>, fName, lName, telNo, **staffNo**)

FK staffNo references Staff(staffNo)



8

COMP211

Step 2.1 Derive Relations for One-to-many (1:1) binary relationship types

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

4.1 *Mandatory participation on both sides (1..1; 1..1)*
Combine entities into one relation

4.2 *Mandatory participation on one side (1..1; 0..1)*
Post PK of entity on the "mandatory" side to act as FK in relation representing entity on the "optional" side

4.3 *Optional participation on both sides (0..1; 0..1)*

9

Step 2.1 Derive Relations for One-to-many (1:1) binary relationship types

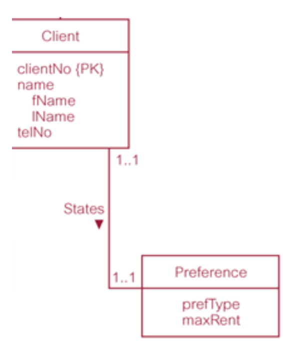
4) One-to-one (1:1) binary relationship types;

4.1 *Mandatory participation on both sides (1..1; 1..1):* Combine entities into one relation

e.g. **Client states Preference relationship** (every client must have Preference)

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

Since every client specifies Preference, can simply merge them into one table.



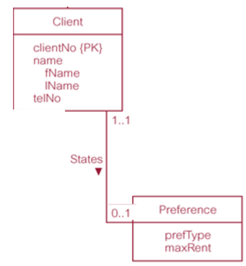
10

Step 2.1 Derive Relations for One-to-many (1:1) binary relationship types

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

4.2 *Mandatory participation on one side (1..1; 0..1):* Post PK of entity on the "mandatory" side to act as FK in relation representing entity on the "optional" side

e.g. **Not every client specifies Preference, *if we merge them into one table, there might be a lot of NULLS.***



For 1:1 relationship with mandatory participation on **Client** side, post **clientNo** into **Preference** to model ***States*** relationship

Client (clientNo, fName, lName, telNo, staffNo)
Primary Key clientNo
Foreign Key staffNo **references** Staff(staffNo)

Preference (clientNo, prefType, maxRent)
Primary Key clientNo
Foreign Key clientNo **references** Client(clientNo)

11

Step 2.1 Derive Relations for One-to-many (1:1) binary relationship types

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

4.3 *Optional participation on both sides (0..1; 0..1): Arbitrary without further information*

12

Step 2.1 Derive Relations for One-to-many (1:1) recursive relationships types

5) One-to-one(1:1) recursive relationship types;

5.1 *Mandatory participation on both sides*
represent the recursive relationship as a single relation with 2 copies of the PK

5.2 *Mandatory participation on one side (1..1; 0..1)*

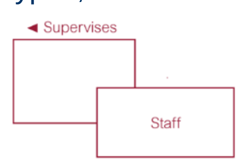
- Option 1: create a single relation with 2 copies of the primary key as described above;
- Option 2: Create a new relation to represent the relationship.

5.3 *Optional participation on both sides (0..1; 0..1)*
create a new relation to represent the relationship

Step 2.1 Derive Relations for One-to-many (1:1) recursive relationships types

5) One-to-one(1:1) recursive relationship types;

5.1 Mandatory participation on both sides:
represent the recursive relationship as a single relation with 2 copies of the PK. One copy of the PK represents a FK and should be renamed to indicated the relationship it represents.



e.g. **Every staff must have a supervisor**

Staff (<u>staffNo</u>, fName, lName, position, sex, DOB, **supervisorNo**)

FK supervisorNo references Staff(staffNo)

Since every staff must have a supervisor, can simply merge them into one table

Step 2.1 Derive Relations for One-to-many (1:1) recursive relationship types

5) One-to-one(1:1) recursive relationship types;

5.2 Mandatory participation on one side:

e.g. **Not every staff has a supervisor**

- Option 1: create a single relation with 2 copies of the primary key as described above;
- Option 2: Create a new relation to represent the relationship. The new relation has 2 attributes, both copies of the PK. As before, one copy of the PK represents a FK and should be renamed to indicated the relationship it represents.

Staff (<u>staffNo</u>, fName, IName, position, sex, DOB)

Supervisor (**<u>staffNo</u>, supervisorNo**)

FK staffNo references Staff(staffNo)

FK supervisorNo references Staff(staffNo)

For example, if you have 1000 staff and only 100 staff has supervisor, option 2 saves storage space.

Step 2.1 Derive Relations for One-to-many (1:1) recursive relationship types

5) One-to-one(1:1) recursive relationship types;

5.2 Mandatory participation on one side – a case study for discussion

Option 1:

Table name: EMPLOYEE_V1

EMP_NUM	EMP_LNAME	EMP_FNAME	EMP_SPOUSE
345	Ramirez	James	347
346	Jones	Anne	349
347	Ramirez	Louise	345
348	Delaney	Robert	
349	Shapiro	Anton	346

First implementation

- Note that the EMPLOYEE_V1 table is likely to yield data anomalies.
- For example, if Anne Jones divorces Anton Shapiro, two records must be updated—by setting the respective EMP_SPOUSE values to null—to properly reflect that change.
- If only one record is updated, inconsistent data occur.
- The problem becomes even worse if several of the divorced employees then marry each other.
- In addition, that implementation also produces undesirable nulls for employees who are *not* married to other employees in the company.

Step 2.1 Derive Relations for One-to-many (1:1) recursive relationship types

5) One-to-one(1:1) recursive relationship types;

5.2 Mandatory participation on one side – a case study for discussion

Option 2:

Table name: EMPLOYEE

EMP_NUM	EMP_LNAME	EMP_FNAME
345	Ramirez	James
346	Jones	Anne
347	Ramirez	Louise
348	Delaney	Robert
349	Shapiro	Anton

Table name: MARRIED_V1

EMP_NUM	EMP_SPOUSE
345	347
346	349
347	345
349	346

- Another approach would be to create a new entity shown as MARRIED_V1 in a 1:M relationship with EMPLOYEE.
- This second implementation can eliminate the nulls for employees who are not married to somebody working for the same company.
- However, this approach still yields possible duplicate values.
  - For example, the marriage between employees 345 and 347 may still appear twice, once as 345,347 and once as 347,345.
- This solution is still likely to produce inconsistent data. For example, it is possible to enter employee 345 as married to employee 347 and to enter employee 348 as married to employee 345.

17

COMP211

Step 2.1 Derive Relations for One-to-many (1:1) recursive relationship types

5) One-to-one(1:1) recursive relationship types;

5.2 Mandatory participation on one side – a case study for discussion

SUMMARY:

- As you can see, a recursive 1:1 relationship yields many different solutions with varying degrees of effectiveness and adherence to basic design principles.
- Any of the above solutions would likely involve the creation of program code to help ensure the integrity and consistency of the data.
- Finally, document, document, and document! Put all design activities in writing. Then review what you have written.
- Documentation not only helps you stay on track during the design process, but also enables you (or those following you) to pick up the design thread when the time comes to modify the design.

18

COMP211

Step 2.1 Derive Relations for One-to-many (1:1) recursive relationship types

5) One-to-one(1:1) recursive relationship types;

5.3 Optional participation on both sides:

Create a new relation as described on previous slide.

Step 2.1 Derive Relations for Superclass/subclass relationship types

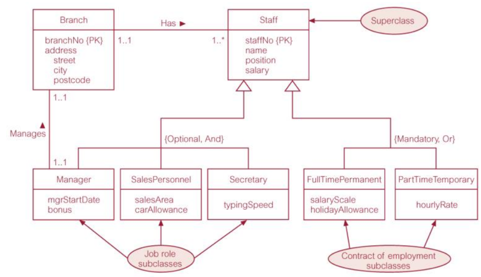
6) Superclass/subclass relationship types;

- Identify superclass as parent entity and subclass entity as child entity.
- There are various options on how to represent such a relationship as one or more relations. Most appropriate option depends on a number of factors such as:
  - disjointness and participation constraints on the superclass/subclass relationship,
  - whether subclasses are involved in distinct relationships,
  - number of participants in superclass/subclass relationship.
- Guidelines for representation of Superclass / Subclass relationship is shown on slide 23.

Step 2.1 Derive Relations for Superclass/subclass relationship types

6) Superclass/subclass relationship types:

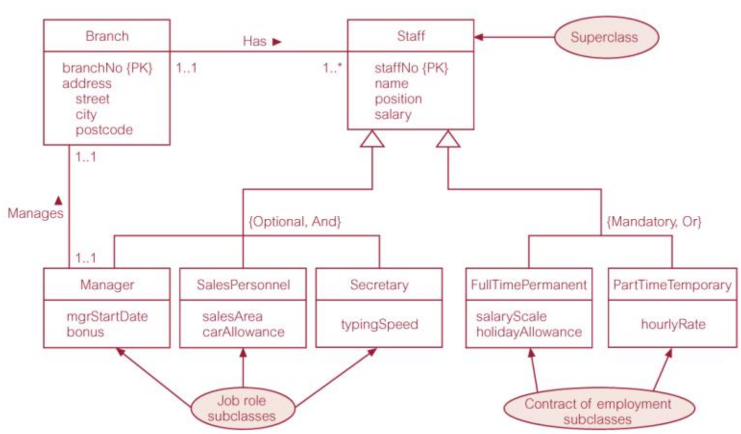
- Disjointness: **And / Or**
  - Determines whether should merge various subclasses together.
    - And: Yes
    - Or: No
- Participation: **Mandatory / Optional**
  - Determines whether should merge superclass & subclass together.
    - Mandatory: Yes
    - Optional: No



Slide 16 of Chapter 13

Step 2.1 Derive Relations for Superclass/subclass relationship types

- As a result, we will have



Step 2.1 Derive Relations for Superclass/subclass relationship types

Table 17.1 Guidelines for the representation of a superclass/subclass relationship based on the participation and disjoint constraints.

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

Step 2.1 Derive Relations for Superclass/subclass relationship types

6) Superclass/subclass relationship types;
   - E.g. the various options to represent the Owner superclass/subclass relationship are as follows:

Option 1 – Mandatory, nondisjoint

AllOwner (ownerNo, address, telNo, fName, lName, bName, bType, contactName, pOwnerFlag, bOwnerF ag)
Primary Key ownerNo

Option 2 – Optional, nondisjoint

Owner (ownerNo, address, telNo)
Primary Key ownerNo

OwnerDetails (ownerNo, fName lName, bName, bType, contactName, pOwnerF ag, bOwnerFlag)
Primary Key ownerNo
Foreign Key ownerNo **references** Owner(ownerNo)

Option 3 – Mandatory, disjoint

PrivateOwner (ownerNo, fName, lName, address, te No)
Primary Key ownerNo

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

Option 4 – Optional, disjoint

Owner (ownerNo, address, telNo)
Primary Key ownerNo

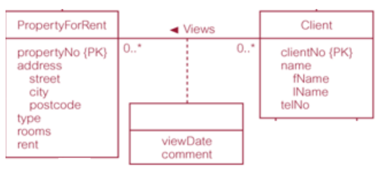
PrivateOwner (ownerNo, fName, lName)
Primary Key ownerNo
Foreign Key ownerNo **references** Owner(ownerNo)

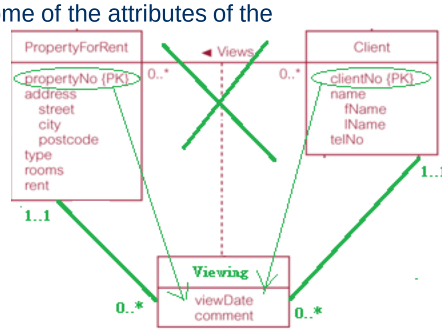
BusinessOwner (ownerNo, bName, bType, contactName)
Primary Key ownerNo
Foreign Key ownerNo **references** Owner(ownerNo)

Step 2.1 Derive Relations for Many-to-many (*:*) relationship types

7) Many-to-many (*:*) binary relationship types;

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





Step 2.1 Derive Relations for Many-to-many (*:*) relationship types

7) Many-to-many (*:*) binary relationship types;

- E.g. **Consider the Client Views PropertyForRent**

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)

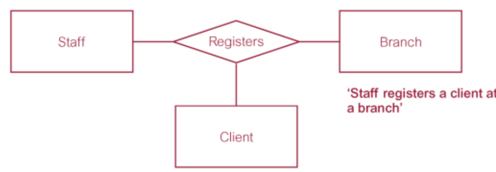
Primary Key propertyNo

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 Complex relationship types

8) Complex relationship types;

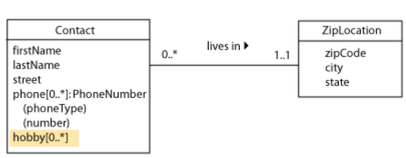
- Create relation to represent relationship and include any attributes that are part of the relationship.
- Post copy of primary key attribute(s) of entities that participate in the complex relationship into new relation, to act as foreign keys.
- Any foreign keys that represent a ‘many’ relationship (for example, 1..*, 0..*) generally will also form the primary key of new relation, possibly in combination with some of the attributes of the relationship.



Step 2.1 Derive Relations for Multi-valued attributes: Example 1

9) Multi-valued attributes – Example 1.

- A multi-valued attribute is a single attribute with more than one distinct data values entered for that attribute.
- According to the properties of a relation that we have discussed in Chapter 4, no attribute is permitted to have multiple data values.
- Look at the following example



Contact hobbies

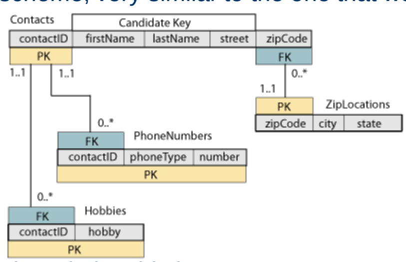
contactid	firstname	lastname	hobbies
1639	George	Barnes	reading
5629	Susan	Noble	hiking, movies
3388	Erwin	Star	hockey, skiing
5772	Alice	Buck	
1911	Frank	Borders	photography, travel, art
4848	Hanna	Diedrich	gourmet cooking

- In this case, the hobby attribute is a **multivalued attribute**.
- The problem with doing it is that it is now difficult (but possible) to search the table for any particular hobby that a person might have, and it is impossible to create a query that will individually list the hobbies that are shown in the table.

Step 2.1 Derive Relations for Multi-valued attributes – Example 1 (cont'd)

9) Multi-valued attributes – Example 1.

- Instead, we will remove the old hobbies attribute and create a new scheme, very similar to the one that we created for the phone numbers.



Hobbies

contactid	hobby
1639	reading
5629	hiking
5629	movies
3388	hockey
3388	skiing
1911	photography
1911	travel
1911	art
4848	gourmet cooking

- The relationship between Contacts and Hobbies is one-to-many.
- The Hobbies relation has only one descriptive attribute, the hobby name.
- To uniquely identify each row of the table, we need to know which contact a hobby belongs to and which hobby it is—so both attributes form the composite primary key of the Hobbies relation.

Step 2.1 Derive Relations for Multi-valued attributes – Example 1 Summary

9) Multi-valued attributes – Example 1 - SUMMARY.

- 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, primary key of the new relation is a combination of the multivalued attribute and the primary key of the entity.

Step 2.1 Derive Relations for Multi-valued attributes – Example 2

9) Multi-valued attributes – Example 2.

- **A single branch has up to three telephone numbers**, and the telNo attribute of the Branch entity has been defined as a multi-valued attribute.

Post **branchNo** into **Telephone**

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

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

- An alternative solution is to split the multivalued attributes into its components and keep these components in the same entity.

  Branch(branchNo, street, city, postcode, tel1, tel2, tel3)

- But keep in mind that the amount of storage space that would be wasted if most branches only have one telephone number.

Step 2.1 Derive Relations for Multi-valued attributes – SUMMARY

9) Multi-valued attributes – SUMMARY.

- If you can resolve a multivalued attribute by adding one or two additional attributes to an existing entity, then do so.
- Otherwise, resolve multivalued attributes by creating a new entity.
- Creating a new entity in a 1:* relationship with the original entity creates a more flexible and expandable solution!

Step 2.1 Derive Relations for Logical Data Model (cont'd)

- At the end of Step 2.1, document the composition of the relations derived for the logical data model.

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 Client(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)	

33

Step 2.4 Check integrity constraints

- Consider the following types of integrity constraints:
  - required data,
  - entity and referential integrity,
  - attribute domain constraints, and
  - general constraints
- For referential integrity, specify existence constraints that define conditions under which a candidate key or foreign key may be inserted, updated, or deleted.

34

Step 2.4 Check integrity constraints (cont'd)

For the 1:* Staff Manages PropertyForRent relationship, consider the following cases:

PropertyForRent

propertyNo	street	city	postcode	type	rooms	rent	ownerNo	staffNo	branchNo
PA14	16 Holhead	Aberdeen	AB7 5SU	House	6	650	CO46	SA9	B007
PL94	6 Argyll St	London	NW2	Flat	4	400	CO87	SL41	B005
PG4	6 Lawrence St	Glasgow	G11 9QX	Flat	3	350	CO40		B003
PG36	2 Manor Rd	Glasgow	G32 4QX	Flat	3	375	CO93	SG37	B003
PG21	18 Dale Rd	Glasgow	G12	House	5	600	CO87	SG37	B003
PG16	5 Novar Dr	Glasgow	G12 9AX	Flat	4	450	CO93	SG14	B003

- Case 1: Insert tuple into child relation (PropertyForRent)
  - Check that the FK attribute staffNo of PropertyForRent is set to null or to a value of an existing Staff tuple.

Step 2.4 Check integrity constraints (cont'd)

- Case 2: Delete tuple from child relation
  - Referential integrity is unaffected.
- Case 3: Update foreign key of child tuple
  - Check that the FK attribute staffNo of the updated PropertyForRent is set to null or to a value of an existing Staff tuple.

PropertyForRent

propertyNo	street	city	postcode	type	rooms	rent	ownerNo	staffNo	branchNo
PA14	16 Holhead	Aberdeen	AB7 5SU	House	6	650	CO46	SA9	B007
PL94	6 Argyll St	London	NW2	Flat	4	400	CO87	SL41	B005
PG4	6 Lawrence St	Glasgow	G11 9QX	Flat	3	350	CO40		B003
PG36	2 Manor Rd	Glasgow	G32 4QX	Flat	3	375	CO93	SG37	B003
PG21	18 Dale Rd	Glasgow	G12	House	5	600	CO87	SG37	B003
PG16	5 Novar Dr	Glasgow	G12 9AX	Flat	4	450	CO93	SG14	B003

Step 2.4 Check integrity constraints (cont'd)

- Case 4: Insert tuple into parent relation (Staff)
  - This action does not affect referential integrity as it simply becomes a parent without any children.

Staff

Relation

staffNo	fName	lName	position	sex	DOB	salary	branchNo
SL21	John	White	Manager	M	1-Oct-45	30000	B005
SG37	Ann	Beech	Assistant	F	10-Nov-60	12000	B003
SG14	David	Ford	Supervisor	M	24-Mar-58	18000	B003
SA9	Mary	Howe	Assistant	F	19-Feb-70	9000	B007
SG5	Susan	Brand	Manager	F	3-Jun-40	24000	B003
SL41	Julie	Lee	Assistant	F	13-Jun-65	9000	B005

Step 2.4 Check integrity constraints (cont'd)

- Case 5: Delete tuple from parent relation
  - This action might cause referential integrity to be lost if there exists a child tuple referencing the deleted parent tuple.
  - There are several strategies to consider:
    - NO ACTION: Prevent such a deletion to cause the loss of referential integrity
    - CASCADE: When the parent tuple is deleted, automatically delete any referenced child tuples.
    - SET NULL: When the parent tuple is deleted, the FK values in all corresponding child tuples are automatically set to null.
    - SET DEFAULT: When the parent tuple is deleted, the FK values in all corresponding child tuples are automatically set to their default values.
    - NO CHECK: do nothing to ensure referential integrity is maintained.

Step 2.4 Check integrity constraints (cont'd)

- Case 6: Update primary key of parent tuple
  - This action might cause referential integrity to be lost if there exists a child tuple referencing the old primary key value.
  - To ensure referential integrity, the strategies described in Case 5 can be used.

Step 2.4 Check integrity constraints

Referential Integrity Constraints for Relations in Staff View 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 2.6 Merge logical data models into global data model (optional)

- 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

Step 2.6 Merge logical data models into global data model (optional) (cont'd)

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.

Global Logical Data Model

Table 17.3 A comparison of the names of entities/relations and their candidate keys in the Branch and Staff views.

Branch view		Staff view	
Entity type	Candidate keys	Entity type	Candidate keys
Branch	branchNo postcode		
Telephone	telNo		
Staff	**staffNo**	**Staff**	**staffNo**
Manager	staffNo		
PrivateOwner	**ownerNo**	**PrivateOwner**	**ownerNo**
BusinessOwner	**bName** **telNo**	**BusinessOwner**	**bName** **telNo** ownerNo
Client	**clientNo**	**Client**	**clientNo**
PropertyForRent	**propertyNo**	**PropertyForRent**	**propertyNo**
		Viewing	clientNo, propertyNo
Lease	**leaseNo** **propertyNo, rentStart** **clientNo, rentStart**	**Lease**	**leaseNo** **propertyNo, rentStart** **clientNo, rentStart**
Registration	clientNo		
Newspaper	newpaperName telNo		
Advert	(propertyNo, newspaperName, dateAdvert)		

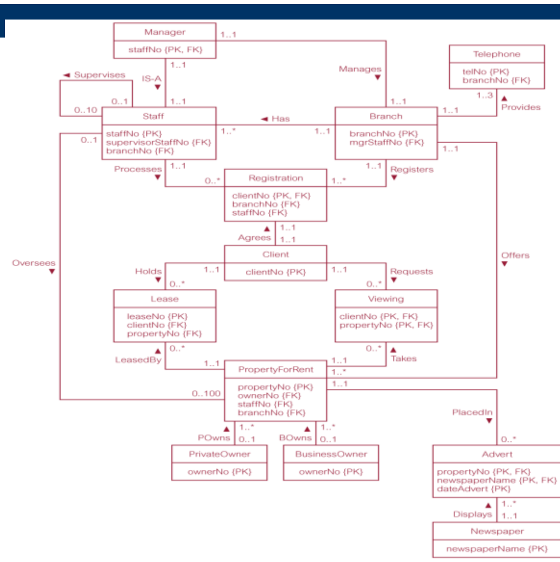
43

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** (clientNo, 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, 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)	**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)	

44

Global Relation Diagram for *DreamHome*



45