Chapter 14

Normalization

Chapter 14 - Objectives

- The purpose of normalization.
- How normalization can be used when designing a relational database.
- The potential problems associated with redundant data in base relations.
- The concept of functional dependency, which describes the relationship between attributes.
- The characteristics of functional dependencies used in normalization.

Chapter 14 - Objectives

- How to identify functional dependencies for a given relation.
- How functional dependencies identify the primary key for a relation.
- How to undertake the process of normalization.
- How normalization uses functional dependencies to group attributes into relations that are in a known normal form.

Chapter 14 - Objectives

- How to identify the most commonly used normal forms, namely First Normal Form (1NF), Second Normal Form (2NF), and Third Normal Form (3NF).
- The problems associated with relations that break the rules of 1NF, 2NF, or 3NF.
- How to represent attributes shown on a form as 3NF relations using normalization.

Purpose of Normalization

- **Normalization is a technique for producing a set of suitable relations that support the data requirements of an enterprise.**

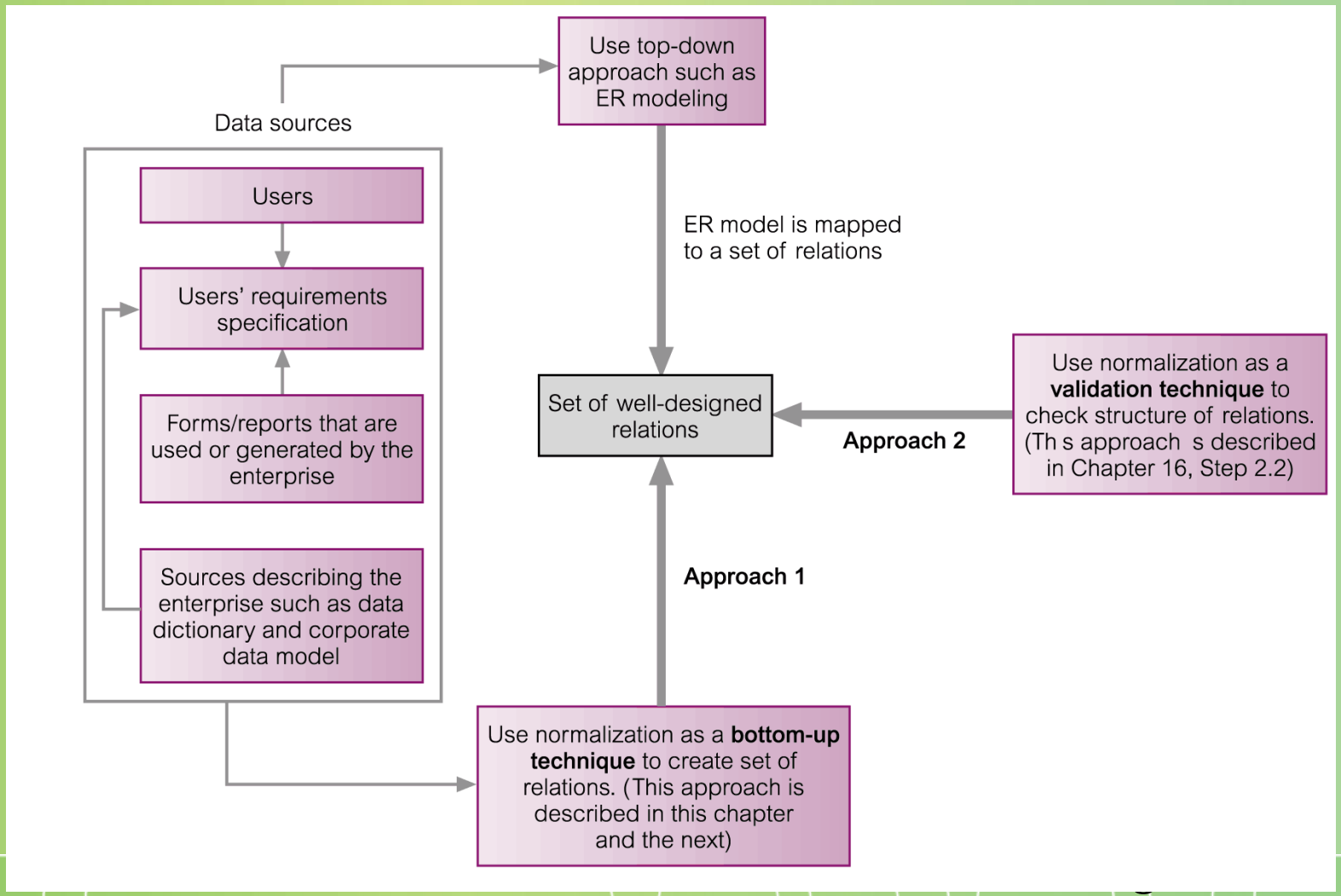
Purpose of Normalization

- Characteristics of a suitable set of relations include:
  - the *minimal* number of attributes necessary to support the data requirements of the enterprise;
  - attributes with a close logical relationship are found in the same relation;
  - *minimal* redundancy with each attribute represented only once with the important exception of attributes that form all or part of foreign keys, which are essential for the joining of related relations..

Purpose of Normalization

- The benefits of using a database that has a suitable set of relations is that the database will be:
  - easier for the user to access and maintain the data;
  - take up minimal storage space on the computer.

How Normalization Supports Database Design



Data Redundancy and Update Anomalies

- **Major aim of relational database design is to group attributes into relations to minimize data redundancy.**

Data Redundancy and Update Anomalies

- **Potential benefits:**
  - **Updates to the data stored in the database are achieved with a minimal number of operations thus reducing the opportunities for data inconsistencies.**
  - **Reduction in the file storage space required by the base relations thus minimizing costs.**

Data Redundancy and Update Anomalies

- **Problems associated with data redundancy are illustrated by comparing the Staff and Branch relations with the *StaffBranch relation.***

Staff	(staffNo, sName, position, salary, branchNo)
Branch	(branchNo, bAddress)

StaffBranch	(staffNo, sName, position, salary, branchNo, bAddress)

Which one is better?

Data Redundancy and Update Anomalies

Staff

staffNo	sName	position	salary	branchNo
SL21	John White	Manager	30000	B005
SG37	Ann Beech	Assistant	12000	B003
SG14	David Ford	Supervisor	18000	B003
SA9	Mary Howe	Assistant	9000	B007
SG5	Susan Brand	Manager	24000	B003
SL41	Julie Lee	Assistant	9000	B005

Branch

branchNo	bAddress
B005	22 Deer Rd, London
B007	16 Argyll St, Aberdeen
B003	163 Main St, Glasgow

Figure 14.2 Staff and Branch relations.

Figure 14.3 StaffBranch relation.

StaffBranch

staffNo	sName	position	salary	branchNo	bAddress
SL21	John White	Manager	30000	B005	22 Deer Rd, London
SG37	Ann Beech	Assistant	12000	B003	163 Main St, Glasgow
SG14	David Ford	Supervisor	18000	B003	163 Main St, Glasgow
SA9	Mary Howe	Assistant	9000	B007	16 Argyll St, Aberdeen
SG5	Susan Brand	Manager	24000	B003	163 Main St, Glasgow
SL41	Julie Lee	Assistant	9000	B005	22 Deer Rd, London

Data Redundancy and Update Anomalies

- StaffBranch relation has redundant data; the details of a branch are repeated for every member of staff.

- In contrast, the branch information appears only once for each branch in the Branch relation and only the branch number (branchNo) is repeated in the Staff relation, to represent where each member of staff is located.

Data Redundancy and Update Anomalies

- Relations that contain redundant information may potentially suffer from update anomalies.

- Types of update anomalies include
  - Insertion
  - Deletion
  - Modification

Data Redundancy and Update Anomalies

- **Insertion**
  - For example to insert the details of new staff located at branch number B007, we must enter the correct details of branch number B007 so that the branch details are consistent with values for branch B007 in other tuples of the **StaffBranch** relation. To insert details of a new branch that currently has no members of staff intothe **StaffBranch** relation, it is necessary to enter nulls into the attributes for staff, such as staffNo. However, as **staffNo** is the primary key for the **StaffBranch** relation, attempting to enter nulls for **staffNo** violates entity integrity, and is not allowed.

Data Redundancy and Update Anomalies

- **Deletion**
  - If we delete a tuple from the StaffBranch relation that represents the last member of staff located at a branch, the details about that branch are also lost from the database.

Data Redundancy and Update Anomalies

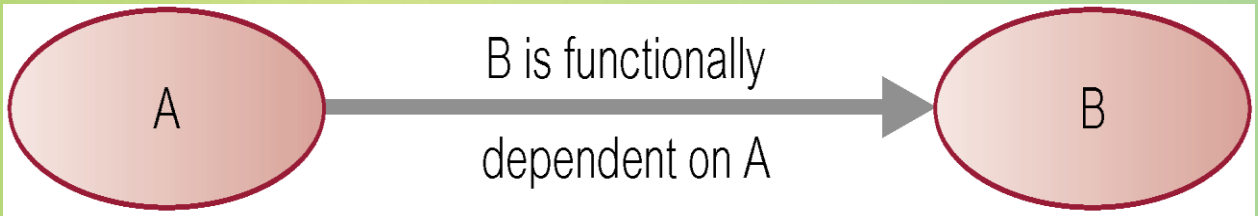
- **Modification**
  - If we want to change the value of one of the attributes of a particular branch in the **StaffBranch** relation—for example, the address for branch number B003—we must update the tuples of all staff located at that branch.
  - If this modification is not carried out on all the appropriate tuples of the **StaffBranch** relation, the database will become inconsistent.
  - In this example, branch number B003 may appear to have different addresses in different staff tuples.
  - This suggests that we can avoid these anomalies by decomposing the original relation into the **Staff** and **Branch** relations.

Lossless-join and Dependency Preservation Properties

- Two important properties of decomposition.
  - Lossless-join property enables us to find any instance of the original relation from corresponding instances in the smaller relations.
  - Dependency preservation property enables us to enforce a constraint on the original relation by enforcing some constraint on each of the smaller relations (Chapter 15 Advanced Normalization).

Functional Dependencies

- Important concept associated with normalization.
- Functional dependency is a property of the meaning or semantics of the attributes in a relation, i.e., it describes relationship between attributes.
- For example, let A and B be attributes of a relation, then B is functionally dependent (or A functionally determines B") on A (denoted A $\rightarrow$ B) if each value of A (in R) is associated with exactly one value of B (in R).
- (A and B may each consist of one or more attributes.)



- The determinant of a functional dependency refers to the attribute or group of attributes on the left-hand side of the arrow.

An Example Functional Dependency

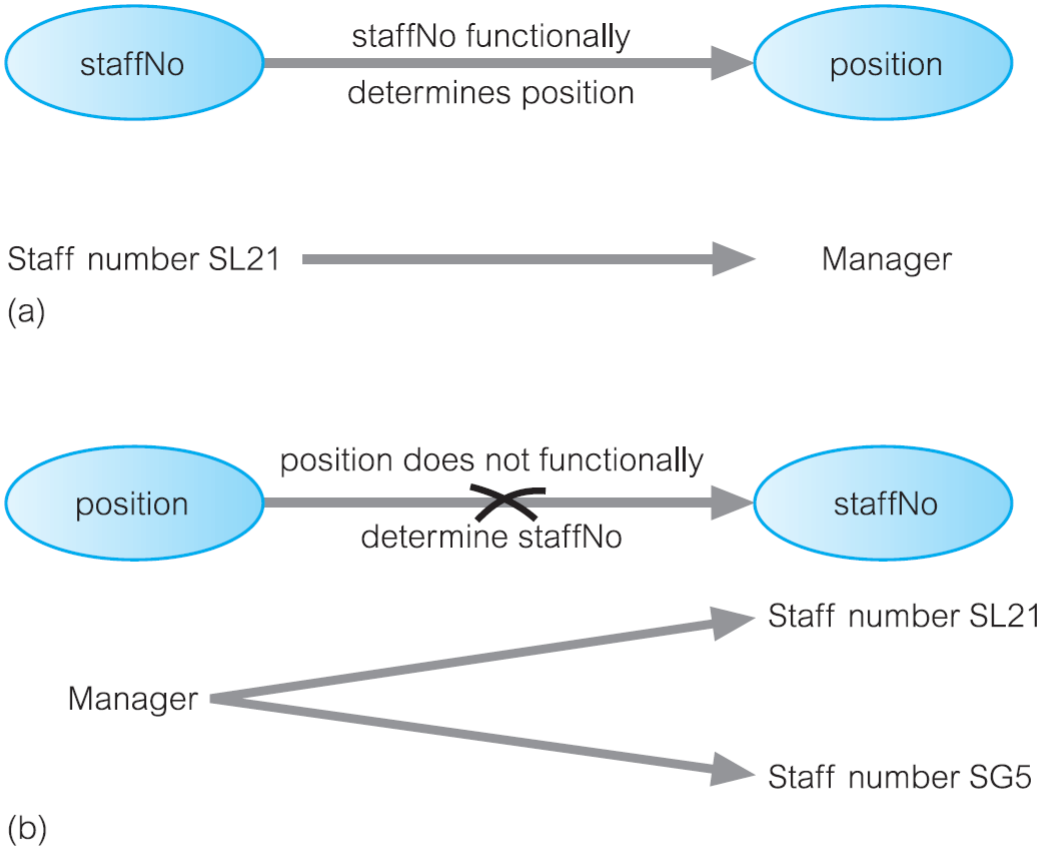


Figure 14.5 (a) staffNo functionally determines position (staffNo ® position); (b) position does *not* functionally determine staffNo (position ⊗® staffNo).

Example Functional Dependency that holds for all Time

- Consider the values shown in staffNo and sName attributes of the Staff relation (see Slide 12).

- Based on sample data, the following functional dependencies appear to hold.

staffNo → sName

sName → staffNo

Example Functional Dependency that holds for all Time

- However, the only functional dependency that remains true for all possible values for the staffNo and sName attributes of the Staff relation is (there might be members of staff with the same name):

  staffNo → sName

Characteristics of Functional Dependencies

- **Determinants should have the minimal number of attributes necessary to maintain the functional dependency with the attribute(s) on the right hand-side.**

- **This requirement is called *full functional dependency.***

Characteristics of Functional Dependencies

- **Full functional dependency indicates that if A and B are attributes of a relation, B is fully functionally dependent on A, if B is functionally dependent on A, but not on any proper subset of A.**

Example Full Functional Dependency

- Consider the following functional dependency that exists in the Staff relation of Figure 14.2:

staffNo, sName → branchNo

  - It is correct that each value of **(staffNo, sName)** is associated with a single value of **branchNo**. However, it is not a full functional dependency, because branchNo is also functionally dependent on a subset of **(staffNo, sName)**, namely **staffNo**.
  - In other words, the functional dependency shown in the example is an example of a partial dependency.
  - The type of functional dependency that we are interested in identifying is a full functional dependency as shown here:

staffNo→ branchNo

Characteristics of Functional Dependencies

- **Main characteristics of functional dependencies used in normalization:**
  - **There is a *one-to-one* relationship between the attribute(s) on the left-hand side (determinant) and those on the right-hand side of a functional dependency.**
  - **Holds for *all* time.**
- **The determinant has the *minimal* number of attributes necessary to maintain the dependency with the attribute(s) on the right hand-side, i.e., there must be a full functional dependency between the attribute(s) on the left-hand and right-hand sides of the dependency.**

Transitive Dependencies

- Important to recognize a transitive dependency because its existence in a relation can potentially cause update anomalies.

- Transitive dependency describes a condition where A, B, and C are attributes of a relation such that if A → B and B → C, then C is transitively dependent on A via B (provided that A is not functionally dependent on B or C).

Example Transitive Dependency

- Consider the following functional dependencies in the StaffBranch relation (see Slide 12):

staffNo → sName, position, salary, branchNo, bAddress

branchNo → bAddress

- The transitive dependency branchNo → bAddress exists on staffNo via branchNo.
- In other words, the staffNo attribute functionally determines the bAddress via the branchNo attribute and neither branchNo nor bAddress functionally determines staffNo.

The Process of Normalization

- Formal technique for analyzing a relation based on its primary key and the functional dependencies between the attributes of that relation.

- Often executed as a series of steps. Each step corresponds to a specific normal form, which has known properties.

Identifying Functional Dependencies

- Identifying all functional dependencies between a set of attributes is relatively simple if the meaning of each attribute and the relationships between the attributes are well understood.

- This information should be provided by the enterprise in the form of discussions with users and/or documentation such as the users' requirements specification.

Identifying Functional Dependencies

- However, if the users are unavailable for consultation and/or the documentation is incomplete then depending on the database application it may be necessary for the database designer to use their common sense and/or experience to provide the missing information.

Example - Identifying a set of functional dependencies for the StaffBranch relation

- Examine semantics of attributes in StaffBranch relation.
- Assume that position held and branch determine a member of staff's salary.

StaffBranch

staffNo	sName	position	salary	branchNo	bAddress
SL21	John White	Manager	30000	B005	22 Deer Rd, London
SG37	Ann Beech	Assistant	12000	B003	163 Main St, Glasgow
SG14	David Ford	Supervisor	18000	B003	163 Main St, Glasgow
SA9	Mary Howe	Assistant	9000	B007	16 Argyll St, Aberdeen
SG5	Susan Brand	Manager	24000	B003	163 Main St, Glasgow
SL41	Julie Lee	Assistant	9000	B005	22 Deer Rd, London

Example - Identifying a set of functional dependencies for the StaffBranch relation

- **With sufficient information available, we can simply identify five functional dependencies for the StaffBranch relation:**

staffNo → sName, position, salary, branchNo, bAddress

branchNo → bAddress

bAddress → branchNo

branchNo, position → salary

bAddress, position → salary

Using sample data to identify functional dependencies.

- As a contrast to previous example, we now consider the situation where functional dependencies are to be identified in the absence of appropriate information about the meaning of attributes and their relationships.
- In this case, it may be possible to identify functional dependencies if sample data is available that is a true representation of *all* possible data values that the database may hold.

Example - Using sample data to identify functional dependencies.

- Consider the data for attributes denoted A, B, C, D, and E in the *Sample* relation (see Slide 36).
- It is important to establish that sample data values shown in relation are representative of all possible values that can be held by attributes A, B, C, D, and E.
- Let us assume that to be true despite the relatively small amount of data shown in this relation.

Example - Using sample data to identify functional dependencies.

- To identify the functional dependencies that exist between attributes A, B, C, D, and E, examine the Sample relation and identify when values in one column are consistent with the presence of particular values in other columns.
- Begin with the first column on the left-hand side and work our way over to the right-hand side of the relation and then we look at combinations of columns; in other words, where values in two or more columns are consistent with the appearance of values in other columns.

Sample Relation

A	B	C	D	E
a	b	z	w	q
e	b	r	w	p
a	d	z	w	t
e	d	r	w	q
a	f	z	s	t
e	f	r	s	t

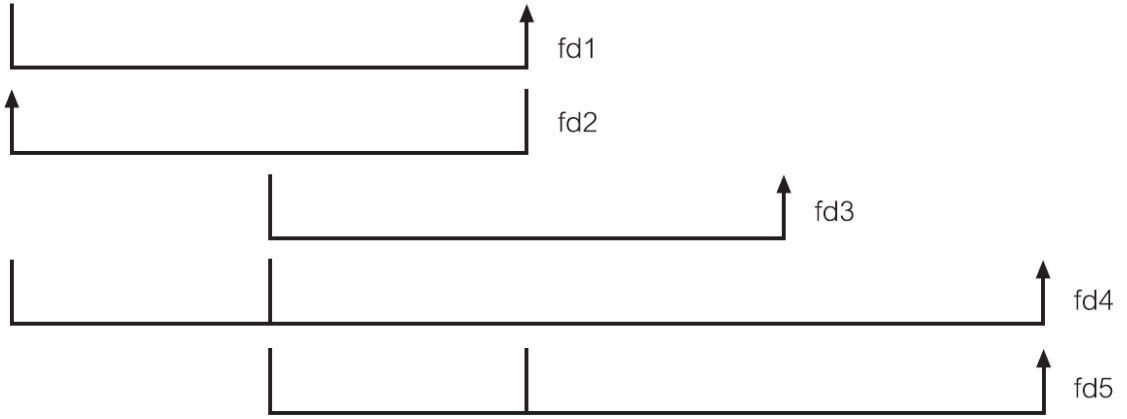


Figure 14.6 The Sample relation displaying data for attributes A, B, C, D, and E and the functional dependencies (fd1 to fd5) that exist between these attributes.

Example - Using sample data to identify functional dependencies.

- **Function dependencies between in the Sample relation.**

$A \rightarrow C$	(fd1)
$C \rightarrow A$	(fd2)
$B \rightarrow D$	(fd3)
$A, B \rightarrow E$	(fd4)
$B, C \rightarrow E$	(fd5)

Identifying the Primary Key for a Relation using Functional Dependencies

- Main purpose of identifying a set of functional dependencies for a relation is to specify the set of integrity constraints that must hold on a relation.

- An important integrity constraint to consider first is the identification of candidate keys, one of which is selected to be the primary key for the relation.

Example - Identify Primary Key for StaffBranch Relation

- **StaffBranch relation has five functional dependencies:**

  staffNo → sName, position, salary, branchNo, bAddress,

  branchNo → bAddress

  bAddress → branchNo

  branchNo, position → salary

  bAddress, position → salary

- **The determinants are staffNo, branchNo, bAddress, (branchNo, position), and (bAddress, position).**
- **To identify all candidate key(s), identify the attribute (or group of attributes) that uniquely identifies each tuple in this relation.**

Example - Identifying Primary Key for StaffBranch Relation

- All attributes that are not part of a candidate key should be functionally dependent on the key.

- The only candidate key and therefore primary key for StaffBranch relation, is staffNo, as *all* other attributes of the relation are functionally dependent on staffNo.

Example - Identifying Primary Key for Sample Relation

- Sample relation has four functional dependencies:

$A \rightarrow C$	(fd1)
$C \rightarrow A$	(fd2)
$B \rightarrow D$	(fd3)
$A, B \rightarrow E$	(fd4)
$B, C \rightarrow E$	(fd5)

- A suitable determinant must functionally determine the other attributes in the relation.
- The determinants in the Sample relation are A, B, C, (A, B), and (B, C).
- However, the only determinants that determine all the other attributes of the relation are (A, B) and (B, C).

Example - Identifying Primary Key for Sample Relation

- **Sample relation has four functional dependencies:**

A → C	**(fd1)**
C → A	**(fd2)**
B → D	**(fd3)**
A, B → E	**(fd4)**
B, C → E	**(fd5)**

- In the case of (A, B), A functionally determines C, and B functionally determines D, and (A, B) functionally determines E, i.e, the attributes that make up the determinant (A, B) can determine all the other attributes either separately as A or B or together as (A, B).
- Hence, we see that an essential characteristic for a candidate key of a relation is that the attributes of a determinant either **individually** or **working together** must be able to functionally determine *all* the other attributes in the relation.

Example - Identifying Primary Key for Sample Relation

- **Sample relation has four functional dependencies:**

$A \rightarrow C$	(fd1)
$C \rightarrow A$	(fd2)
$B \rightarrow D$	(fd3)
$A, B \rightarrow E$	(fd4)
$B, C \rightarrow E$	(fd5)

- This is also true for the determinant (B, C), but is not a characteristic of the others in the Sample relation (namely A, B, or C), as in each case they can determine only one other attribute in the relation.
- In conclusion, there are two candidate keys for the Sample relation; namely, (A, B) and (B, C), and as each has similar characteristics (such as number of attributes), the selection of primary key for the Sample relation is arbitrary. The candidate key not selected to be the primary key is referred to as the alternate key for the Sample relation.

The Process of Normalization

- Normalization is often executed as a series of steps.
- Each step corresponds to a specific normal form that has known properties.
- As normalization proceeds, the relations become progressively more restricted (stronger) in format and also less vulnerable to update anomalies.
- For the relational data model, it is important to recognize that it is only First Normal Form (1NF) that is critical in creating relations; all subsequent normal forms are optional.
- However, to avoid the update anomalies discussed previously, it is generally recommended that we proceed to at least Third Normal Form (3NF).

The Process of Normalization

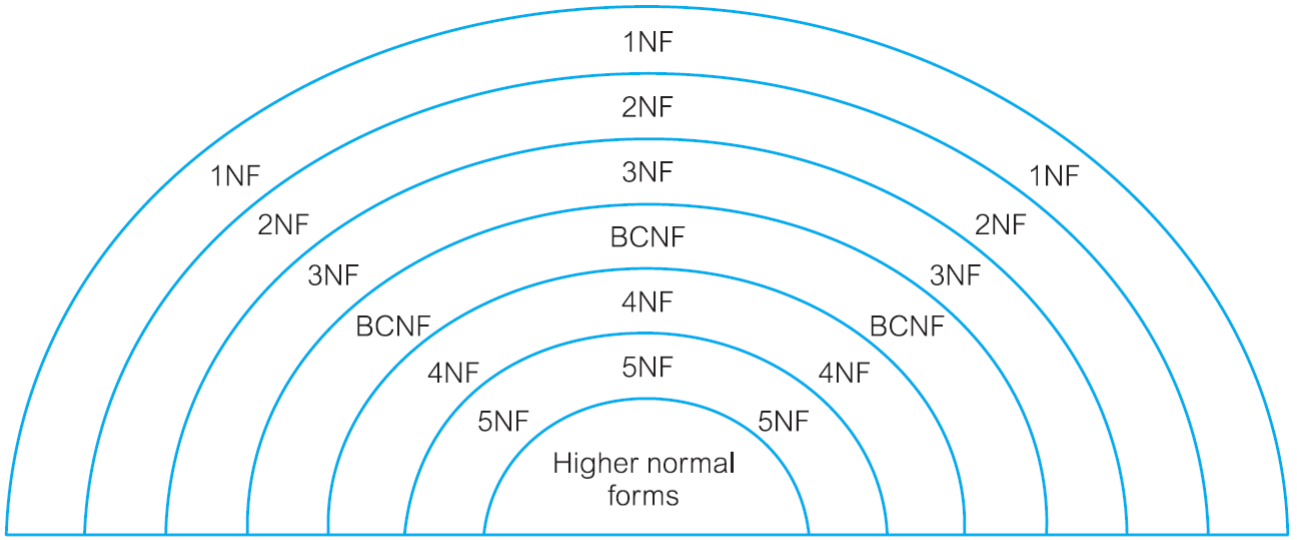


Figure 14.7 Diagrammatic illustration of the relationship between the normal forms.

The Process of Normalization

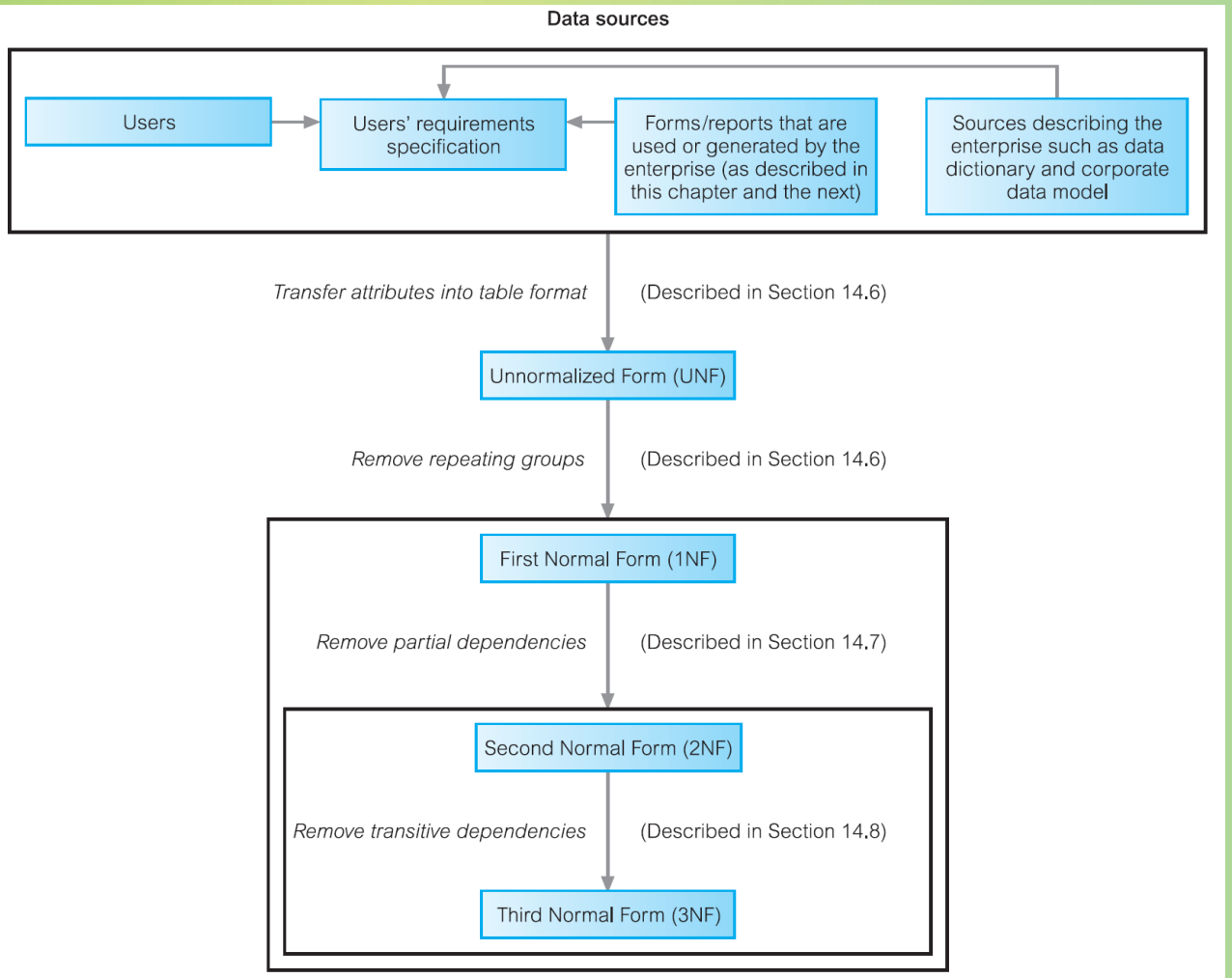


Figure 14.8 Diagrammatic illustration of the process of normalization.

Unnormalized Form (UNF)

- **Unnormalized Form (UNF) is a table that contains one or more repeating groups.**

- **To create an unnormalized table**
  - Transform the data from the information source (e.g., a standard data entry form) into table format with columns and rows.
  - In such format, the table is in unnormalized Form and is referred to as an **unnormalized table.**

Unnormalized Form (UNF)- Example

- A collection of (simplified) *DreamHome* leases is shown.
- The lease on top is for a client called John Kay who is leasing a property in Glasgow, which is owned by Tina Murphy.
- For this worked example, we assume that a client rents a given property only once and cannot rent more than one property at any one time.

Figure 14.9 Collection of (simplified) *DreamHome* leases.

Unnormalized Form (UNF)- Example

ClientRental

clientNo	cName	propertyNo	pAddress	rentStart	rentFinish	rent	ownerNo	oName
CR76	John Kay	PG4	6 Lawrence St, Glasgow	1-Jul-12	31-Aug-13	350	CO40	Tina Murphy
		PG16	5 Novar Dr, Glasgow	1-Sep-13	1-Sep-14	50	CO93	Tony Shaw
CR56	Aline Stewart	PG4	6 Lawrence St, Glasgow	1-Sep-11	10-June-12	350	CO40	Tina Murphy
		PG36	2 Manor Rd, Glasgow	10-Oct-12	1-Dec-13	375	CO93	Tony Shaw
		PG16	5 Novar Dr, Glasgow	1-Nov-14	10-Aug-15	450	CO93	Tony Shaw

Figure 14.10 ClientRental unnormalized table.

First Normal Form (1NF)

- A relation in which the intersection of each row and column contains one and only one value.

UNF to 1NF

- To transform the **unnormalized table** to **First Normal Form**, identify and remove repeating groups within the table.
- A repeating group is an attribute, or group of attributes, within a table that occurs with multiple values for a single occurrence of the nominated key attribute(s) for that table.
- Note that in this context, the term “key” refers to the attribute(s) that uniquely identify each row within the **Unnormalized table**.

UNF to 1NF

- There are two common approaches to removing repeating groups from unnormalized tables:
  - Entering appropriate data into the empty columns of rows containing the repeating data ('flattening' the table).
  - Or by
  - Placing the repeating data along with a copy of the original key attribute(s) into a separate relation.

UNF to 1NF-Example

ClientRental

clientNo	cName	propertyNo	pAddress	rentStart	rentFinish	rent	ownerNo	oName
CR76	John Kay	PG4	6 Lawrence St, Glasgow	1-Jul-12	31-Aug-13	350	CO40	Tina Murphy
		PG16	5 Novar Dr, Glasgow	1-Sep-13	1-Sep-14	50	CO93	Tony Shaw
CR56	Aline Stewart	PG4	6 Lawrence St, Glasgow	1-Sep-11	10-June-12	350	CO40	Tina Murphy
		PG36	2 Manor Rd, Glasgow	10-Oct-12	1-Dec-13	375	CO93	Tony Shaw
		PG16	5 Novar Dr, Glasgow	1-Nov-14	10-Aug-15	450	CO93	Tony Shaw

Figure 14.10 ClientRental unnormalized table.

- Repeating Group = (propertyNo, pAddress, rentStart, rentFinish, rent, ownerNo, oName)
- As a consequence, there are multiple values at the intersection of certain rows and columns.

UNF to 1NF-Example

ClientRental

clientNo	cName	propertyNo	pAddress	rentStart	rentFinish	rent	ownerNo	oName
CR76	John Kay	PG4	6 Lawrence St, Glasgow	1-Jul-12	31-Aug-13	350	CO40	Tina Murphy
		PG16	5 Novar Dr, Glasgow	1-Sep-13	1-Sep-14	50	CO93	Tony Shaw
CR56	Aline Stewart	PG4	6 Lawrence St, Glasgow	1-Sep-11	10-June-12	350	CO40	Tina Murphy
		PG36	2 Manor Rd, Glasgow	10-Oct-12	1-Dec-13	375	CO93	Tony Shaw
		PG16	5 Novar Dr, Glasgow	1-Nov-14	10-Aug-15	450	CO93	Tony Shaw

Figure 14.10 ClientRental unnormalized table.

- For example, there are two values for propertyNo (PG4 and PG16) for the client
- named John Kay.
- To transform an unnormalized table into 1NF, we ensure that there is a single value at the intersection of each row and column.
- This is achieved by removing the repeating group.

UNF to 1NF-Example

- With the first approach, we remove the repeating group (property rented details) by entering the appropriate client data into each row.
- The resulting first normal form ClientRental relation is shown in the figure.

ClientRental

clientNo	propertyNo	cName	pAddress	rentStart	rentFinish	rent	ownerNo	oName
CR76	PG4	John Kay	6 Lawrence St, Glasgow	1-Jul-12	31-Aug-13	350	CO40	Tina Murphy
CR76	PG16	John Kay	5 Novar Dr, Glasgow	1-Sep-13	1-Sep-14	450	CO93	Tony Shaw
CR56	PG4	Aline Stewart	6 Lawrence St, Glasgow	1-Sep-11	10-Jun-12	350	CO40	Tina Murphy
CR56	PG36	Aline Stewart	2 Manor Rd, Glasgow	10-Oct-12	1-Dec-13	375	CO93	Tony Shaw
CR56	PG16	Aline Stewart	5 Novar Dr, Glasgow	1-Nov-14	10-Aug-15	450	CO93	Tony Shaw

Figure 14.11 First Normal Form ClientRental relation.

Identifying Candidate Keys Using The functional dependencies

ClientRental

clientNo	propertyNo	cName	pAddress	rentStart	rentFinish	rent	ownerNo	oName
CR76	PG4	John Kay	6 Lawrence St, Glasgow	1-Jul-12	31-Aug-13	350	CO40	Tina Murphy
CR76	PG16	John Kay	5 Novar Dr, Glasgow	1-Sep-13	1-Sep-14	450	CO93	Tony Shaw
CR56	PG4	Aline Stewart	6 Lawrence St, Glasgow	1-Sep-11	10-Jun-12	350	CO40	Tina Murphy
CR56	PG36	Aline Stewart	2 Manor Rd, Glasgow	10-Oct-12	1-Dec-13	375	CO93	Tony Shaw
CR56	PG16	Aline Stewart	5 Novar Dr, Glasgow	1-Nov-14	10-Aug-15	450	CO93	Tony Shaw

Figure 14.11 First Normal Form ClientRental relation.

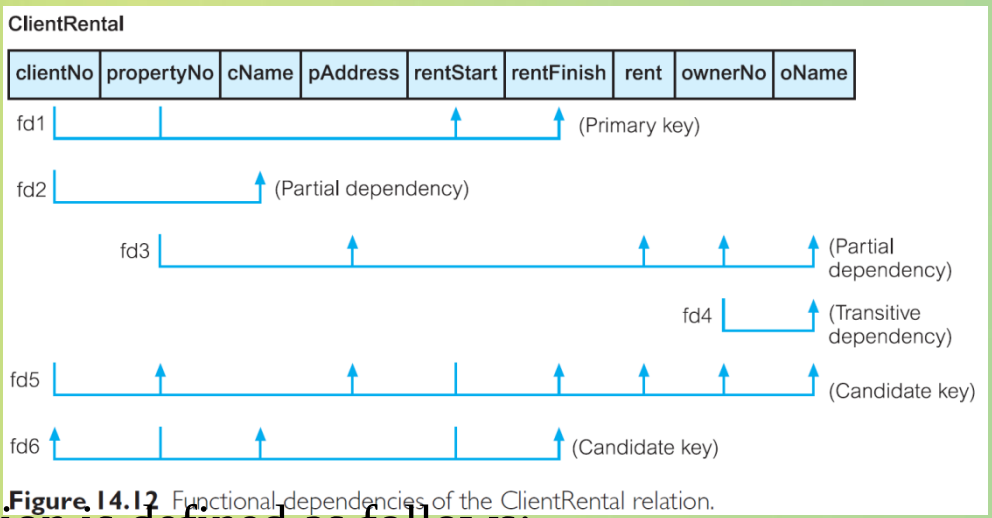


Figure 14.12 Functional dependencies of the ClientRental relation.

The ClientRental relation is defined as follows:

ClientRental (clientNo, propertyNo, cName, pAddress, rentStart, rentFinish, rent, ownerNo, oName)

UNF to 1NF-Example

- With the second approach, we remove the repeating group (property rented details) by placing the repeating data along with a copy of the original key attribute (clientNo) in a separate relation, as shown in the figure.

Client

clientNo	cName
CR76	John Kay
CR56	Aline Stewart

PropertyRentalOwner

clientNo	propertyNo	pAddress	rentStart	rentFinish	rent	ownerNo	oName
CR76	PG4	6 Lawrence St, Glasgow	1-Jul-12	31-Aug-13	350	CO40	Tina Murphy
CR76	PG16	5 Novar Dr, Glasgow	1-Sep-13	1-Sep-14	450	CO93	Tony Shaw
CR56	PG4	6 Lawrence St, Glasgow	1-Sep-11	10-Jun-12	350	CO40	Tina Murphy
CR56	PG36	2 Manor Rd, Glasgow	10-Oct-12	1-Dec-13	375	CO93	Tony Shaw
CR56	PG16	5 Novar Dr, Glasgow	1-Nov-14	10-Aug-15	450	CO93	Tony Shaw

Figure 14.13 Alternative 1NF Client and PropertyRental-Owner relations.

- By using functional dependencies, we identify a primary key for the relations. The format of the resulting 1NF relations are as follows:
  - Client (clientNo, cName)
  - PropertyRentalOwner (clientNo, propertyNo, pAddress, rentStart, rentFinish, rent, ownerNo, oName)

Second Normal Form (2NF)

- Based on the concept of full functional dependency.
- Full functional dependency indicates that if
  - A and B are attributes of a relation: B is fully dependent on A if B is functionally dependent on A but not on any proper subset of A.

Second Normal Form (2NF)

- A relation that is in 1NF and every non-primary-key attribute is fully functionally dependent on the primary key.

1NF to 2NF

- Identify the primary key for the 1NF relation.
- Identify the functional dependencies in the relation.
- If partial dependencies exist on the primary key remove them by placing then in a new relation along with a copy of their determinant.

1NF to 2NF- Example

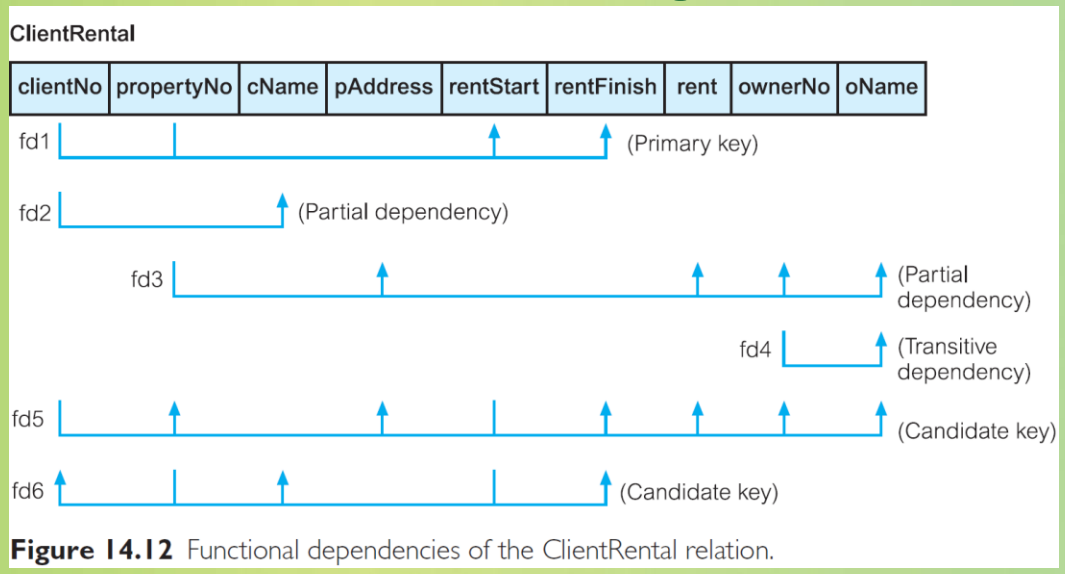


Figure 14.12 Functional dependencies of the ClientRental relation.

- **As shown in the figure, the *ClientRental* relation has the following functional dependencies:**

fd1	clientNo, propertyNo ® rentStart, rentFinish	(Primary key)
fd2	clientNo ® cName	(Partial dependency)
fd3	propertyNo ® pAddress, rent, ownerNo, oName	(Partial dependency)
fd4	ownerNo ® oName	(Transitive dependency)
fd5	clientNo, rentStart ® propertyNo, pAddress, rentFinish, rent, ownerNo, oName	(Candidate key)
fd6	propertyNo, rentStart ® clientNo, cName, rentFinish	(Candidate key)

1NF to 2NF- Example

fd1	clientNo, propertyNo ® rentStart, rentFinish	(Primary key)
fd2	clientNo ® cName	(Partial dependency)
fd3	propertyNo ® pAddress, rent, ownerNo, oName	(Partial dependency)
fd4	ownerNo ® oName	(Transitive dependency)
fd5	clientNo, rentStart ® propertyNo, pAddress, rentFinish, rent, ownerNo, oName	(Candidate key)
fd6	propertyNo, rentStart ® clientNo, cName, rentFinish	(Candidate key)

- To transform the ClientRental relation into 2NF, it requires the creation of new relations so that the non-primary-key attributes are removed along with a copy of the part of the primary key on which they are fully functionally dependent.
- This results in the creation of three new relations called Client, Rental, and PropertyOwner:
  - Rental (clientNo, propertyNo, rentStart, rentFinish)
  - Client (clientNo, cName)
  - PropertyOwner (propertyNo, pAddress, rent, ownerNo, oName)

1NF to 2NF- Example

Client

clientNo	cName
CR76	John Kay
CR56	Aline Stewart

Rental

clientNo	propertyNo	rentStart	rentFinish
CR76	PG4	1-Jul-12	31-Aug-13
CR76	PG16	1-Sep-13	1-Sep-14
CR56	PG4	1-Sep-11	10-Jun-12
CR56	PG36	10-Oct-12	1-Dec-13
CR56	PG16	1-Nov-14	10-Aug-15

PropertyOwner

propertyNo	pAddress	rent	ownerNo	oName
PG4	6 Lawrence St, Glasgow	350	CO40	Tina Murphy
PG16	5 Novar Dr, Glasgow	450	CO93	Tony Shaw
PG36	2 Manor Rd, Glasgow	375	CO93	Tony Shaw

Figure 14.14 Second normal form relations derived from the ClientRental relation.

Third Normal Form (3NF)

- Based on the concept of transitive dependency.
- Transitive Dependency is a condition where
  - A, B and C are attributes of a relation such that if A $\rightarrow$ B and B $\rightarrow$ C,
  - then C is transitively dependent on A through B. (Provided that A is not functionally dependent on B or C).

Third Normal Form (3NF)

- A relation that is in 1NF and 2NF and in which no non-primary-key attribute is transitively dependent on the primary key.

2NF to 3NF

- Identify the primary key in the 2NF relation.
- Identify functional dependencies in the relation.
- If transitive dependencies exist on the primary key remove them by placing them in a new relation along with a copy of their determinant.

2NF to 3NF-Example

- The functional dependencies for the Client, Rental, and PropertyOwner relations, derived during the conversion into **Second Normal Form (2NF)** , are as follows:

Client		
fd2	clientNo ® cName	(Primary key)
Rental		
fd1	clientNo, propertyNo ® rentStart, rentFinish	(Primary key)
fd5	clientNo, rentStart ® propertyNo, rentFinish	(Candidate key)
fd6	propertyNo, rentStart ® clientNo, rentFinish	(Candidate key)
PropertyOwner		
fd3	propertyNo ® pAddress, rent, ownerNo, oName	(Primary key)
fd4	ownerNo ® oName	(Transitive dependency)

2NF to 3NF-Example

- To transform the PropertyOwner relation into 3NF, we must first remove this transitive dependency by creating two new relations called PropertyForRent and Owner, as shown in the figure.

The PropertyForRent and Owner relations are in 3NF, as there are no further transitive dependencies on the primary key.

PropertyForRent

propertyNo	pAddress	rent	ownerNo
PG4	6 Lawrence St, Glasgow	350	CO40
PG16	5 Novar Dr, Glasgow	450	CO93
PG36	2 Manor Rd, Glasgow	375	CO93

Owner

ownerNo	oName
CO40	Tina Murphy
CO93	Tony Shaw

Figure 14.15 Third normal form relations derived from the PropertyOwner relation.

- The new relations have the following form:
  - PropertyForRent (propertyNo, pAddress, rent, ownerNo)
  - Owner (ownerNo, oName)

2NF to 3NF-Example

- The whole set of relations become in the following form:
  - Client (clientNo, cName)
  - Rental (clientNo, propertyNo, rentStart, rentFinish)
  - PropertyForRent (propertyNo, pAddress, rent, ownerNo)
  - Owner (ownerNo, oName)

Client

clientNo	cName
CR76	John Kay
CR56	Aline Stewart

Rental

clientNo	propertyNo	rentStart	rentFinish
CR76	PG4	1-Jul-12	31-Aug-13
CR76	PG16	1-Sep-13	1-Sep-14
CR56	PG4	1-Sep-11	10-Jun-12
CR56	PG36	10-Oct-12	1-Dec-13
CR56	PG16	1-Nov-14	10-Aug-15

PropertyForRent

propertyNo	pAddress	rent	ownerNo
PG4	6 Lawrence St, Glasgow	350	CO40
PG16	5 Novar Dr, Glasgow	450	CO93
PG36	2 Manor Rd, Glasgow	375	CO93

Owner

ownerNo	oName
CO40	Tina Murphy
CO93	Tony Shaw

General Definitions of 2NF and 3NF

- **Second normal form (2NF)**
  - **A relation that is in first normal form and every non-primary-key attribute is fully functionally dependent on *any candidate key*.**
- **Third normal form (3NF)**
  - **A relation that is in first and second normal form and in which no non-primary-key attribute is transitively dependent on *any candidate key*.**