Chapter 9. Advanced Data Normalisation

Table of contents

Objectives

At the end of this chapter you should be able to:

- Convert a set of relations to Boyce-Codd normal form.
- Describe the concept of multi-valued dependency, and be able to convert a set of relations to fourth normal form.
- Avoid a number of problems associated with decomposing relations for normalisation.

- Describe how denormalisation can be used to improve the performance response of a database application.

Context

This chapter relates closely to the previous two on database design. It finalises the material on normalisation, demonstrates how a fully normalised design can equally be represented as an entity-relationship model, and addresses the impact that a target DBMS will have on the design process. The issues relating to appropriate choices of DBMS-specific parameters, to ensure the efficient running of the application, relate strongly to the material covered in the chapters on indexing and physical storage. Information in all three chapters can be used in order to develop applications which provide satisfactory response times and make effective use of DBMS resources.

Recap

Introduction

In parallel with this chapter, you should read Chapter 14 of Thomas Connolly and Carolyn Begg, "Database Systems A Practical Approach to Design, Implementation, and Management", (5th edn.).

In this concluding database design unit, we bring together a number of advanced aspects of database application design. It begins by extending the coverage of data normalisation in an earlier chapter, describing Boyce-Codd normal form (a refinement of the original third normal form) and providing a different view of how to generate a set of relations in third normal form. The chapter then looks at a number of important issues to be considered when decomposing relations during the process of normalisation. Finally, the important topic of physical database design is included, which shows the impact that DBMS-specific parameters can have in the development of an application. Many of these considerations have a direct impact on both the flexibility and the performance response of the application.

Before starting work on this chapter

This chapter addresses a number of advanced issues relating to data normalisation. It is very important that you fully understand all the concepts and techniques introduced in the previous chapter, Data Normalisation.

You should not attempt this chapter until you are confident in your understanding and application of the following concepts:

- **Functional dependency**

- Fully functional dependency
- Partial functional dependency
- Direct dependency
- Transitive (indirect) dependency
- **Determinants**
- Determinant
- Determinacy diagrams
- **Normal forms**
- Un-normalised form (UNF)
- First normal form (1NF)
- Second normal form (2NF)
- Third normal form (3NF)

Summary of the first three normal forms

The following is a brief summary of the first three normal forms:

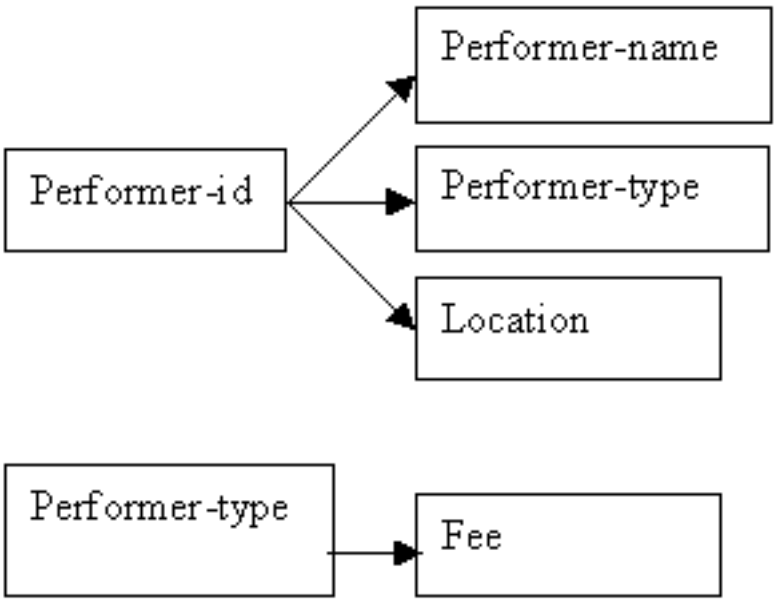
- 1NF — Identify the determinants of data items, and through the removal of any repeating groups, arrange the data items into an initial first normal form relation.
- 2NF — Remove part-key dependencies from the relations in first normal form. I.e. for non-key attributes, remove those attributes that are not fully functionally dependent on the whole of the primary key (and form new entities where these attributes are fully functionally dependent on the whole primary key).
- 3NF — Remove any transitive (indirect) dependencies from the set of relations in second normal form (to produce a set of relations where all attributes are directly dependent on the primary key).

Third normal form determinacy diagrams and relations of Performer

In this chapter we shall be extending the work from the previous chapter on the data for the Performer case study. As a starting point, we shall first present the determinacy diagrams and the third normal form relations developed for this case study.

Case study

Determinacy diagram: Performers and Fees



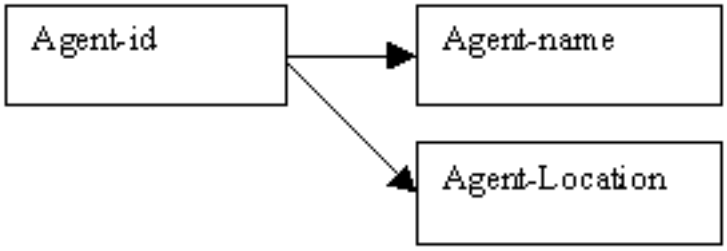
Relation in third normal form: Performers

Performer-id	Performer-name	Performer-code	Performer-location
101	Baron	Singer	York
105	Steed	Singer	Berlin
108	Jones	Actor	Bombay
112	Eagles	Actor	Leeds
118	Markov	Dancer	Moscow
126	Stokes	Comedian	Athens
129	Chong	Actor	Beijing
134	Brass	Singer	London
138	Ng	Singer	Penang
140	Strong	Magician	Rome
141	Gomez	Musician	Lisbon
143	Tan	Singer	Chicago
147	Qureshi	Actor	London
149	Tan	Actor	Taipei
150	Pointer	Magician	Paris
152	Peel	Dancer	London

Relation in third normal form: Fees

Performer-type	Fee
Singer	75
Dancer	60
Actor	85
Comedian	90
Magician	84
Musician	92

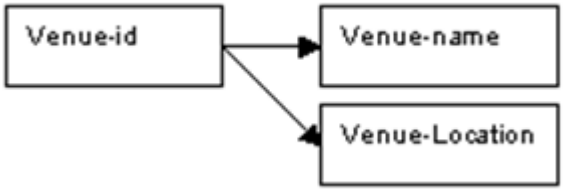
Determinacy diagram: Agents



Relation in third normal form: Agents

Agent-id	Agent-name	Agent-location
1295	Burton	Luton
1435	Nunn	Boston
1504	Lee	Taipei
1682	Tsang	Beijing
1460	Stritch	Rome
1522	Ellis	Madrid
1478	Burns	Leeds
1377	Webb	Sydney
1509	Patel	York
1190	Patel	Hue
1802	Chapel	Bristol
1076	Eccles	Oxford
1409	Arkley	York
1428	Vernon	Cairo

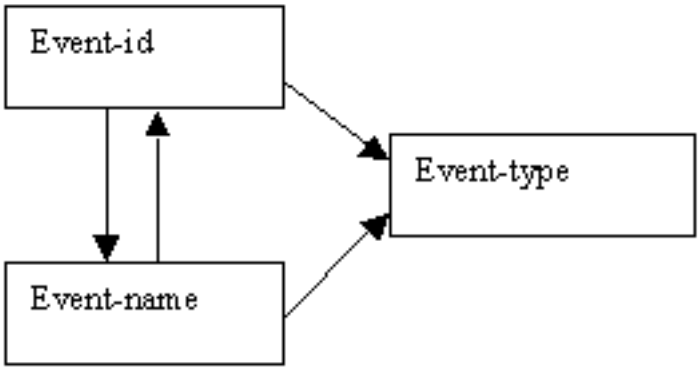
Determinacy diagram: Venues



Relation in third normal form: Venues

Venue-id	Venue-name	Venue-location
59	Atlas	Tokyo
35	Polis	Athens
54	Nation	Lisbon
79	Festive	Rome
46	Royale	Cairo
28	Gratton	Boston
75	Vostok	Kiev
84	State	Kiev
82	Tower	Lima
17	Silbury	Tunis
92	Palace	Milan
62	Shaw	Oxford

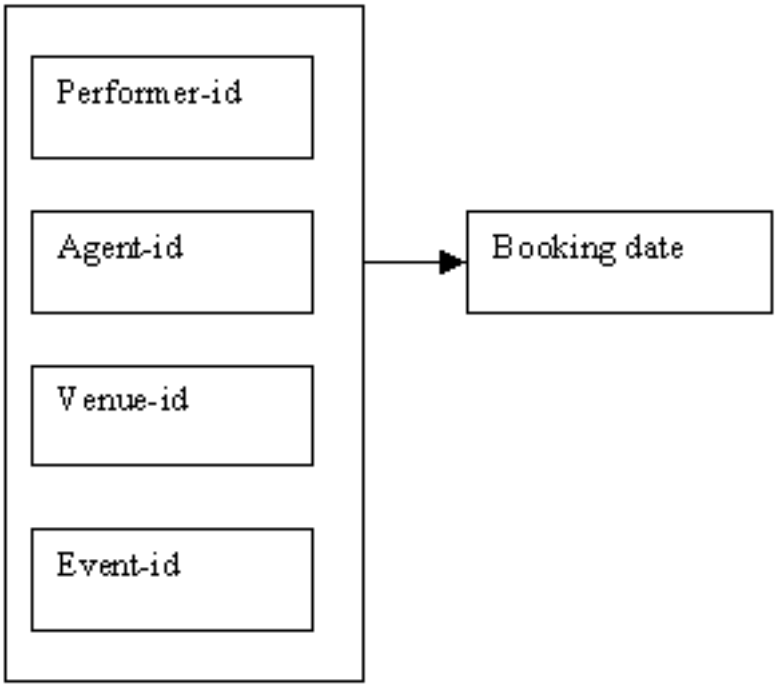
Determinacy diagram: Events

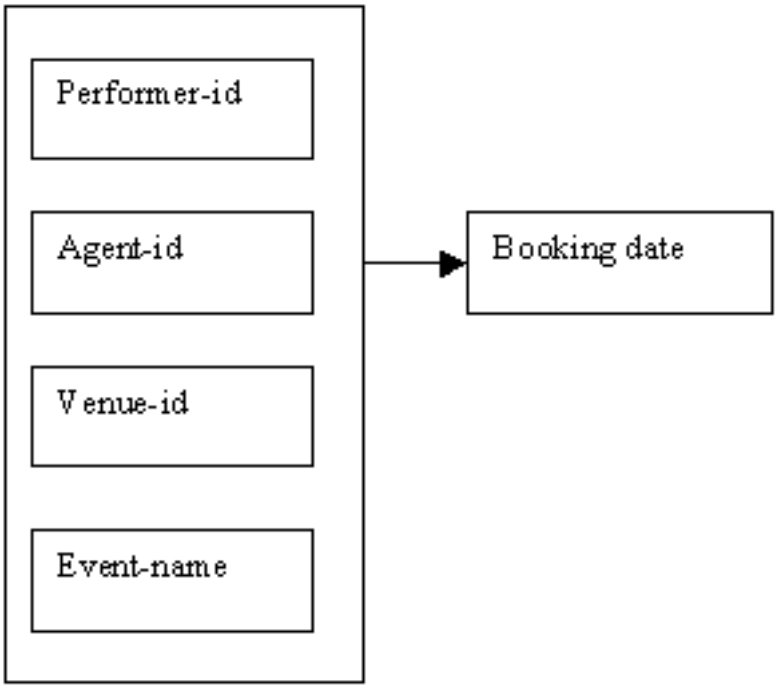


Relation in third normal form: Events

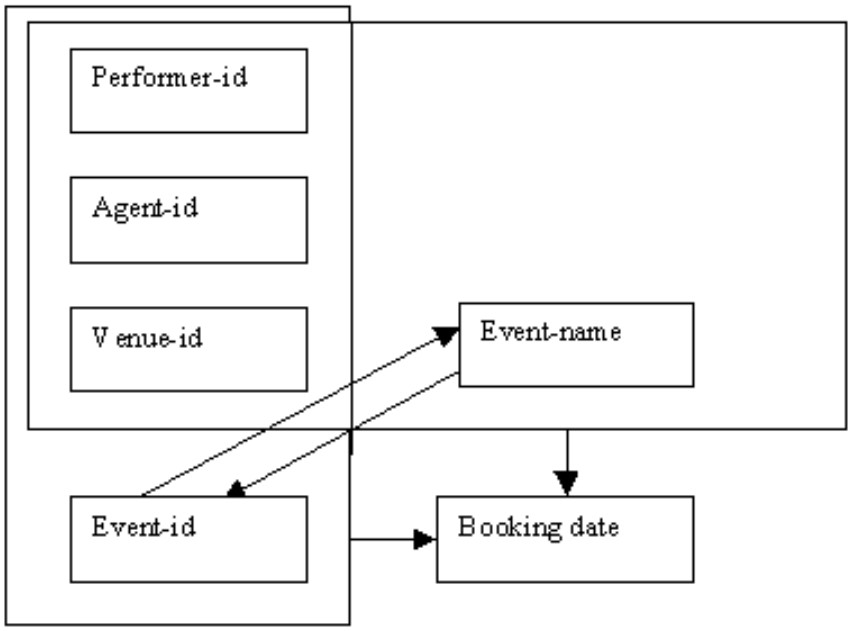
Event-id	Event-name	Event-type
901	The Dark	Drama
907	Elgar 1	Concert
913	What Now?	Drama
921	Silver Shoe	Ballet
926	Next Year	Drama
927	Chanson	Opera
934	Angels	Opera
938	New Dawn	Drama
941	Mahler 1	Concert
942	White Lace	Ballet
945	Trick-Treat	Variety show
952	Gold Days	Drama
957	Quicktime	Musical
959	Show Time	Musical
963	Vanish!	Magic show
964	The Friends	Drama
971	Card Trick	Magic show
978	Swift Step	Dance
981	Birdsong	Musical
988	Secret Tape	Drama

Determinacy diagrams: Bookings


Performer-id
Agent-id
Venue-id
Event-id
Booking date


Performer-id
Agent-id
Venue-id
Event-name
Booking date

The determinacy diagram below combines the previous two determinacy diagrams to show the overlapping keys for the Bookings relation, and illustrates the dependencies between the attributes event-id and event-name:



The details of the Bookings relation are shown below:

Relation in third normal form: Bookings

Performer-id	Agent-id	Venue-id	Event-id	Event-name	BookingDate
101	1295	59	959	Show Time	25-Nov-1999
105	1435	35	921	Silver Shoe	07-Jan-2002
105	1504	54	942	White Lace	10-Feb-2002
108	1682	79	901	The Dark	29-Jul-2003
112	1460	17	926	Next Year	13-Aug-2000
112	1522	46	952	Gold Days	05-May-1999
112	1504	75	952	Gold Days	16-Mar-1999
126	1509	59	945	Trick-Treat	02-Sep-2001
129	1478	79	926	Next Year	22-Jun-2000
134	1504	28	981	Birdsong	18-Sep-2001
138	1509	84	957	Quicktime	18-Aug-1999
140	1478	17	963	Vanish!	18-Aug-1999
141	1478	84	941	Mahler 1	21-Jul-2000
143	1504	79	927	Chanson	21-Nov-2002
147	1076	17	952	Gold Days	30-Apr-2000
147	1409	79	988	Secret Tape	17-Apr-2000
152	1428	59	978	Swift Step	01-Oct-2001

Motivation for normalising beyond third normal form

Why go beyond third normal form?

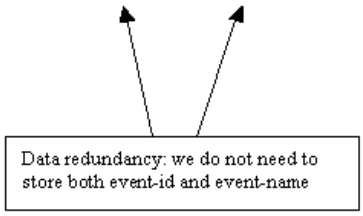
As we shall explore in this section, under certain circumstances there are anomalies that can occur for data that meets all the requirements for third normal form. Once these anomalies were identified and understood, database researchers developed the further normal forms we shall explore in this chapter.

Insertion anomalies of third normal form

There are no true insertion anomalies in the Bookings relation in third normal form; the details about each performer, agent, venue and event are also held in separate relations specifically for those entities, but there is data redundancy.

Relation in third normal form: Bookings

Performer-id	Agent-id	Venue-id	Event-id	Event-name	BookingDate
101	1295	59	959	Show Time	25-Nov-1999
105	1435	35	921	Silver Shoe	07-Jan-2002
105	1504	54	942	White Lace	10-Feb-2002
108	1682	79	901	The Dark	29-Jul-2003
112	1460	17	926	Next Year	13-Aug-2000
112	1522	46	952	Gold Days	05-May-1999
112	1504	75	952	Gold Days	16-Mar-1999
126	1509	59	945	Trick-Treat	02-Sep-2001
129	1478	79	926	Next Year	22-Jun-2000
134	1504	28	981	Birdsong	18-Sep-2001
138	1509	84	957	Quicktime	18-Aug-1999
140	1478	17	963	Vanish!	18-Aug-1999
141	1478	84	941	Mahler 1	21-Jul-2000
143	1504	79	927	Chanson	21-Nov-2002
147	1076	17	952	Gold Days	30-Apr-2000
147	1409	79	988	Secret Tape	17-Apr-2000
152	1428	59	978	Swift Step	01-Oct-2001



We can see that there is data redundancy in the Bookings relation, as every time a particular event is involved in a booking, both the event-id and the event-name need to be inserted into the Bookings relation.

Strictly speaking, we do not need to have both event-id and event-name in the Bookings relation, as each determines the other. If a mistake were to be made while inserting a new tuple, so that the event-id and the event-name did not match, this would cause problems of inconsistency within the database. The solution is to decide on one of the two determinants from the Events relation as part of the composite key for the Bookings relation.

We have noted that the event-id and the event-name determine each other within the Events relation, and this in turn creates overlapping keys in the Bookings relation. If the relationship between event-id and event-name were to break down, and a new event happened to have the same name as another event with a different event-id, this could create problems in the Bookings relation.

Performer Scenario 2

We can refer to a slightly altered database design as 'Performer Scenario 2', in

order to demonstrate the effects of overlapping keys.

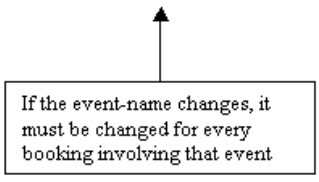
An issue that we need to examine is in the context of this slightly different database. In the example we having been using, the Events relation contains event-id, event-name and event-type. We can see that the performer-type in the Performers relation matches the event-type in the Events relation (e.g. actors performing in dramas, singers performing in musicals). If we now consider that the database holds only event-id and event-name as details about each event, this would affect the structure of the database.

Now, if we were to attempt to insert details about an event which had not yet been booked, we would not be able to do so as we would have an incomplete key in the Bookings relation. An event which has not been booked would have an event-id and an event-name, but no other attributes would have a value as there has been no booking.

Amendment anomalies of third normal form

If there were a change to the name of a particular event, this would need to be reflected in every booking involving that event. Some events may be booked many times, and if the change to the name of an event is not updated in each case, we would again find problems with maintaining consistent information in the database.

Performer-id	Agent-id	Venue-id	Event-id	Event-name	Booking Date
112	1522	46	952	Gold Days	05-May-1999
112	1504	75	952	Gold Days	16-Mar-1999
147	1076	17	952	Gold Days	30-Apr-2000



Here too, the solution is to identify either event-id or event-name as the determinant from the Events relation, so that the other of these two attributes is stored once only in the Events relation.

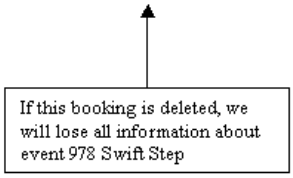
Deletion anomalies of third normal form

There are no deletion anomalies in the example we have been using. If we consider Scenario 2, however, we will find that deletion anomalies do exist.

Performer Scenario 2

We know that in Performer Scenario 2, there is no separate Events relation. If a booking is cancelled, we will want to delete the relevant tuple from the Bookings relation. This means that if we delete a tuple which contained details of an event that had no other booking, we would lose all information about that event.

Performer-id	Agent-id	Venue-id	Event-id	Event-name	BookingDate
152	1428	59	978	Swift Step	01-Oct-2001



Boyce-Codd and fourth normal form

Beyond third normal form

In this section we introduce two new normal forms that are more 'strict' than third normal form. For historical reasons, the simple numbering of first, second and third deviates before getting to fourth. The two new normal forms are called:

- Boyce-Codd normal form
- Fourth normal form

Boyce-Codd normal form

When it comes to identifying the booking, there is an ambiguity, as the booking details could be identified by more than one combination of attributes.

As it is possible to identify details of each event either by the event-id or by the event-name, there are two possible groupings of attributes that could be used to identify a booking: performer-id, agent-id, venue-id and event-id, or performer-id, agent-id, venue-id and event-name.

Important

Boyce-Codd normal form (BCNF)

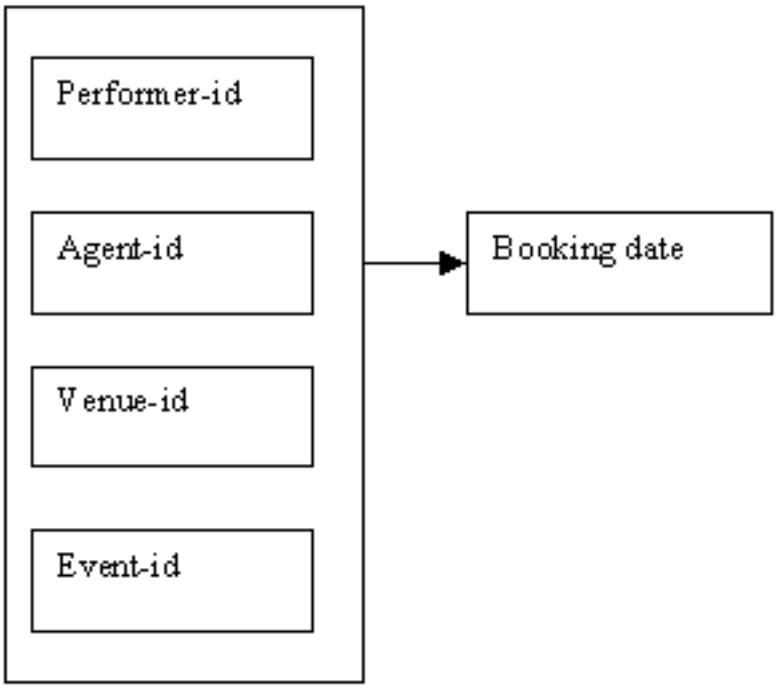
A relation is in Boyce-Codd normal form if all attributes which are determinants are also candidate keys.

Boyce-Codd normal form is stronger than third normal form, and is sometimes known as strong third normal form.

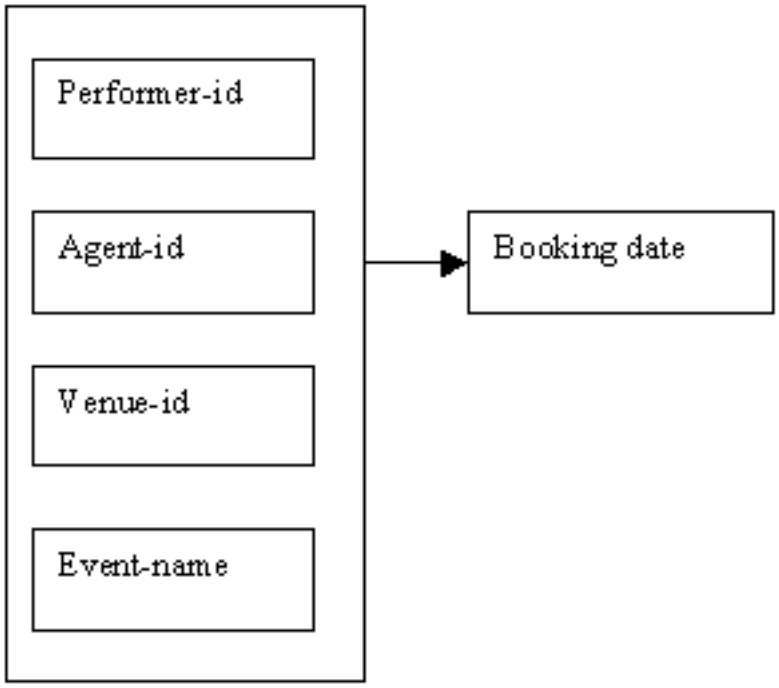
Transformation into Boyce-Codd normal form deals with the problem of overlapping keys.

An indirect dependency is resolved by creating a new relation for each entity; these new relations contain the transitively dependent attributes together with the primary key.

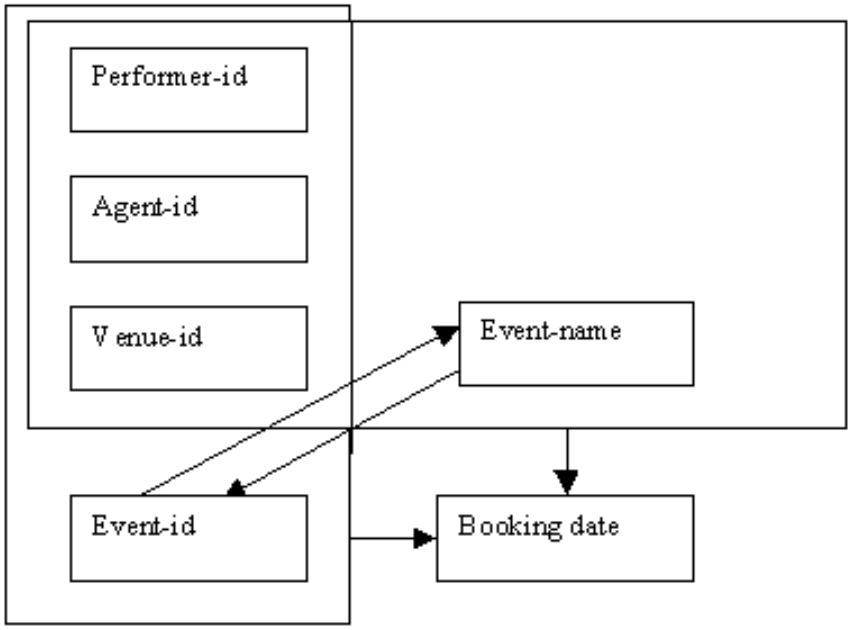
We know that we can identify a booking by means of the attributes performer-id, agent-id, venue-id and event-id, as shown in the determinacy diagram below.



We also know that we can identify a booking by using the attributes performer-id, agent-id, venue-id and event-name, shown in the next determinacy diagram.



When we combine the two determinacy diagrams shown above, we can see that we have an example of overlapping keys:



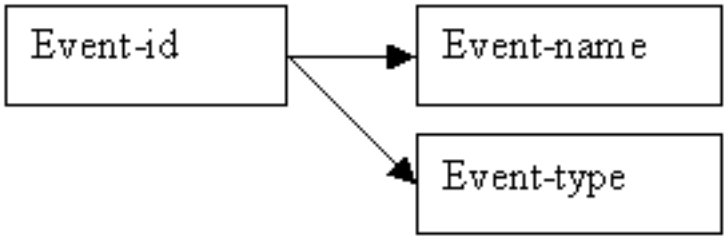
The details of the Bookings relation are shown later in this section.

Although overlapping keys are rare in practice (and some examples may appear rather contrived), we need to be aware that they can occur, and how to deal with them. The solution is simple: we need to decide on a single choice of attributes so that we have only one primary key. We know that event-name would not be an ideal choice of primary key. This is because it can be difficult to get names exactly right (e.g. "Quicktime" is not identical to "Quick Time"), and it may be coincidence rather than a rule that there is a one-to-one relationship between event-id and event-name (the relationship might break down). The choice of attribute to appear in the primary key is therefore event-id rather than event-name.

In Boyce-Codd normal form, we have six relations: Performers, Fees, Agents, Venues, Events and Bookings. The structure of the determinacy diagrams and content of the relations for Performers, Fees, Agents and Venues remain unchanged from third normal forms, and are not repeated here. There are changes to the Events and Bookings relations, which are illustrated below. A summary of the determinacy diagrams and the relations for this example are given in the 'Summary of normalisation rules' section of this chapter, together with an entity-relationship diagram.

Event details The choice of event-id as the primary key for the Bookings relation means that we can show the simpler representation of the determinacy diagram for event details, as we no longer have to consider the attribute event-

name as a possible key.

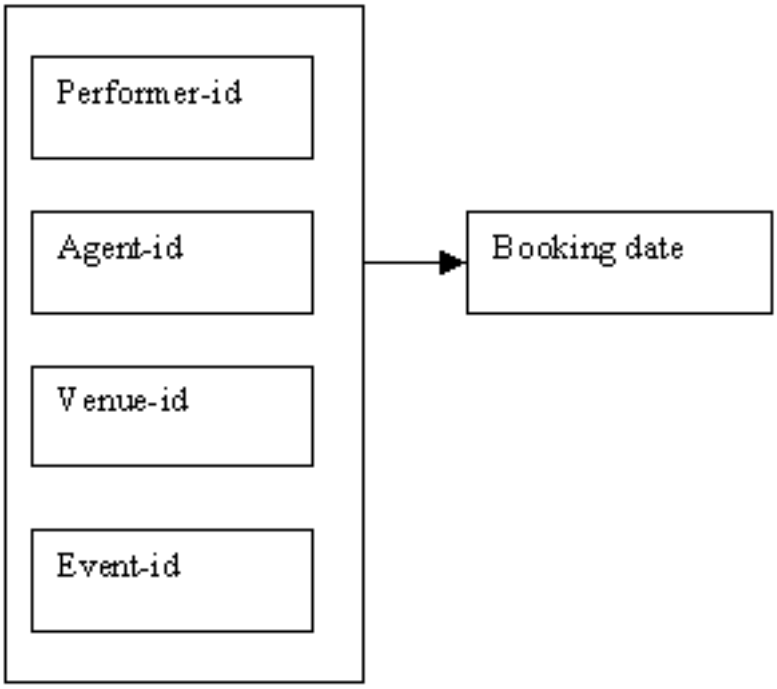


Event-id	Event-name	Event-type
901	The Dark	Drama
907	Elgar 1	Concert
913	What Now?	Drama
921	Silver Shoe	Ballet
926	Next Year	Drama
927	Chanson	Opera
934	Angels	Opera
938	New Dawn	Drama
941	Mahler 1	Concert
942	White Lace	Ballet
945	Trick-Treat	Variety show
952	Gold Days	Drama
957	Quicktime	Musical
959	Show Time	Musical
963	Vanish!	Magic show
964	The Friends	Drama
971	Card Trick	Magic show
978	Swift Step	Dance
981	Birdsong	Musical
988	Secret Tape	Drama

Note that in Scenario 2, where there was no separate Events relation, it would now be necessary to create an Events relation in order to transform the Bookings relation from third normal form into Boyce-Codd normal form.

Booking details Now that we have decided that event-id is the more suitable attribute for use as part of the key for the Bookings relation, we no longer need to store the event-name, which is already held in the Events relation. The problem of the overlapping keys has now been resolved, and the key for the Bookings relation is the combination of the attributes performer-id, agent-id,

venue-id and event-id.



The Bookings relation no longer needs to hold the attribute event-name, as this is already held in the Events relation.

Relation in Boyce-Codd normal form: Bookings

Performer-id	Agent-id	Venue-id	Event-id	BookingDate
101	1295	59	959	25-Nov-1999
105	1435	35	921	07-Jan-2002
105	1504	54	942	10-Feb-2002
108	1682	79	901	29-Jul-2003
112	1460	17	926	13-Aug-2000
112	1522	46	952	05-May-1999
112	1504	75	952	16-Mar-1999
126	1509	59	945	02-Sep-2001
129	1478	79	926	22-Jun-2000
134	1504	28	981	18-Sep-2001
138	1509	84	957	18-Aug-1999
140	1478	17	963	18-Aug-1999
141	1478	84	941	21-Jul-2000
143	1504	79	927	21-Nov-2002
147	1076	17	952	30-Apr-2000
147	1409	79	988	17-Apr-2000
152	1428	59	978	01-Oct-2001

Exercise 1

Define Boyce-Codd normal form.

Fourth normal form

The normalisations process so far has produced a set of five relations, which are robust against insertion, amendment and deletion anomalies. If at this stage it were decided to introduce further details into the relations, it would still be possible to do so. Database designers and developers would be well advised to start again with the normalisations process if changes are proposed to the data. However, for this example, we will introduce some new information that only affects the performers.

We are now required to add further details to the Performers relation, to show their knowledge and skills in two other areas: languages and hobbies.

Important

Fourth normal form (4NF) A relation is in fourth normal form if there are no multi-valued dependencies between the attributes.

Multi-valued dependencies occur where there are a number of attributes that depend only on the primary key, but exist independently of each other.

The representation of languages spoken and hobbies would be a simple enough requirement if each performer spoke exactly one language and had only one hobby. However, our performers are multi-talented, and some speak many languages, and others have several hobbies. Furthermore, the languages and hobbies are not related to each other. This presents us with a problem: how can we represent this in the relation? We know we cannot have a group of items under the headings Languages and Hobbies, as this would contravene the rules for first normal form.

The relation below is an attempt at representing some of this information, using a small number of performers as an example.

Relation: Some-Performers-Example 1

Relation: Some-Performers-Example 1

Perf-id	Perf-name	Perf-code	Performer-location	Languages	Hobbies
101	Baron	0348	York	French	
101	Baron	0348	York	English	
101	Baron	0348	York	Italian	
101	Baron	0348	York		Art
105	Steed	0862	Berlin	German	
105	Steed	0862	Berlin	English	
105	Steed	0862	Berlin	Mandarin	
105	Steed	0862	Berlin		Music
105	Steed	0862	Berlin		Poetry
108	Jones	0729	Bombay	Cantonese	

If we look at this relation, while it conforms to the rules for first normal form (there are no repeating groups), there is still some ambiguity in its meaning. If we look at Baron's hobbies, we can see that 'art' has been identified, but that there is no entry for the attribute 'language'. Does this mean that Baron does not speak any other languages? We know this is not true, because there are other entries that demonstrate that Baron speaks three languages. If we take the alternative view, and look at another entry for Baron, we can see that Baron speaks Italian, but from this entry it could appear that Baron has no hobbies. This approach is not the solution to the problem.

Another attempted solution pairs languages and hobbies together, but sometimes there is a language but no hobby (or the other way around).

Relation: Some-Performers-Example 2

Relation: Some-Performers-Example 2

Perf-id	Perf-name	Perf-code	Performer-location	Languages	Hobbies
101	Baron	0348	York	French	Art
101	Baron	0348	York	English	
101	Baron	0348	York	Italian	
105	Steed	0862	Berlin	German	Music
105	Steed	0862	Berlin	English	Poetry
105	Steed	0862	Berlin	Mandarin	
108	Jones	0729	Bombay	Cantonese	

In this new approach, we have entered each hobby against a language. However, we are still faced with problems. If Steed decides to give up poetry as a hobby, we will lose the information that Steed speaks English. If Baron's French gets 'rusty' and is deleted from the relation, we will lose the information that Baron's hobby is art.

Relation: Some-Performers-Example 3

Relation: Some-Performers-Example 3

Perf-id	Perf-name	Perf-code	Performer-location	Languages	Hobbies
101	Baron	0348	York	French	Art
101	Baron	0348	York	English	Art
101	Baron	0348	York	Italian	Art
105	Steed	0862	Berlin	German	Music
105	Steed	0862	Berlin	German	Poetry
105	Steed	0862	Berlin	English	Music
105	Steed	0862	Berlin	English	Poetry
105	Steed	0862	Berlin	Mandarin	Music
105	Steed	0862	Berlin	Mandarin	Poetry
108	Jones	0729	Bombay	Cantonese	

In this next attempt, all languages are paired with all hobbies; this means that there is a great amount of redundancy, as basic data about the performers is repeated each time. We have a problem with Jones, who does not appear to have a hobby, which questions whether this entry is valid. In addition, if Steed learns a new language, it would be necessary to repeat this new language paired with Steed's existing hobbies. This option is also an unsatisfactory method of solution.

The solution to this problem is to divide the information that we are trying to represent into a group of new relations; one containing the basic performer information as before, another showing details of languages spoken, and a third maintaining a record of hobbies.

This transformation deals with the problems of multi-valued facts and associated redundancies in the data; we can now convert the relation into three relations

in fourth normal form.

Naturally, the new relations would hold data for all the performers, although only an extract from each relation is given here.

Relation in fourth normal form: Some-Performers

Perf-id	Perf-name	Perf-code	Performer-location
101	Baron	0348	York
105	Steed	0862	Berlin
108	Jones	0729	Bombay

Relation in fourth normal form: Some-Performers-Languages

Perf-id	Languages
101	French
101	English
101	Italian
105	German
105	English
105	Mandarin
108	Cantonese

Relation in fourth normal form: Some-Performers-Hobbies

Perf-id	Hobbies
101	Art
105	Music
105	Poetry

Exercise 2

Define fourth normal form.

Summary of normalisation rules

The rules used for converting a group of data items which are un-normalised into a collection of normalised relations are summarised in the table below. Remember that in some cases we might choose not to normalise the data completely, if this would lead to inefficiency in the database.

The conversion from fourth normal form to fifth normal form is included for completeness. We will not be examining the definition of fifth normal form in detail; it is concerned with avoiding unnecessary duplication of tuples when new relations are created by joining together existing relations. The cause of this problem is the existence of interdependent multi-valued data items.

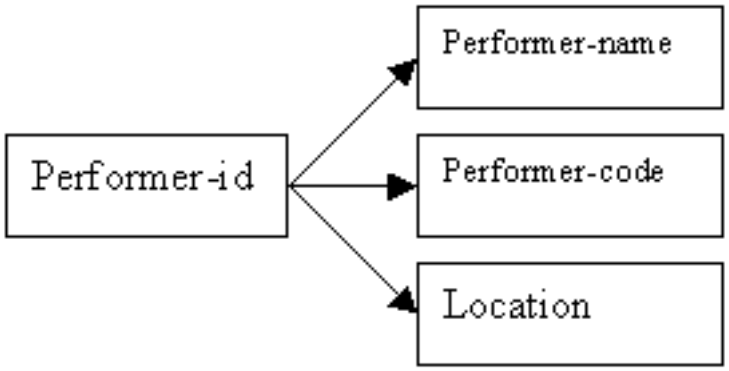
Starting with normal form	Convert to normal form	Abbreviation	Use the rule
Un-normalized data	First	1NF	remove repeating groups of data within a single tuple into new tuples with only one data value for each attribute
First	Second	2NF	Extract non-key partial dependencies (items not fully functionally dependent on the key) into a separate relation
Second	Third	3NF	remove transitive dependencies into a separate relation
Third	Boyce-Codd	BCNF	remove overlapping keys by identifying a single primary key and holding other values in a separate relation
Third Or Boyce-Codd	Fourth	4NF	remove multi-valued dependencies by extracting independent data into a separate relation
Fourth	Fifth	5NF	remove join dependencies caused by interdependent data

Fully normalised relations

We now have five relations which are fully normalised, and can be represented by means of determinacy diagrams, relations, and an entity-relationship diagram. Each relation has a corresponding entity in the entity-relationship diagram.

Performer details

All performers appear in the Performers relation. The primary key is performer-id, and the other attributes are performer-name, performer-code (which identifies the performer-type) and performer-location.

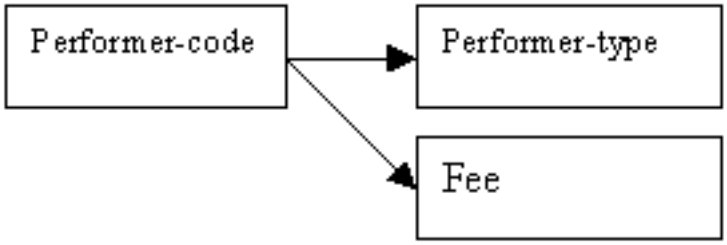


Fully normalised relation: Performers

Performer-id	Performer-name	Performer-code	Performer-location
101	Baron	0348	York
105	Steed	0862	Berlin
108	Jones	0729	Bombay
112	Eagles	0729	Leeds
118	Markov	0862	Moscow
126	Stokes	0244	Athens
129	Chong	0729	Beijing
134	Brass	0348	London
138	Ng	0348	Penang
140	Strong	0360	Rome
141	Gomez	0915	Lisbon
143	Tan	0348	Chicago
147	Qureshi	0729	London
149	Tan	0729	Taipei
150	Pointer	0360	Paris
152	Peel	0862	London

Fee details

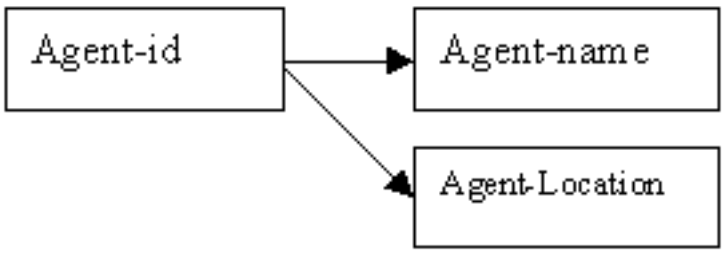
Each performer is paid a fee depending on the performer-type. The rates of pay for each performer-type are stored in the Fees relation, together with a performer-code, which is the primary key.



Performer-code	Performer-type	Fee
0348	Singer	75
0862	Dancer	60
0729	Actor	85
0244	Comedian	90
0360	Magician	84
0915	Musician	92

Agent details

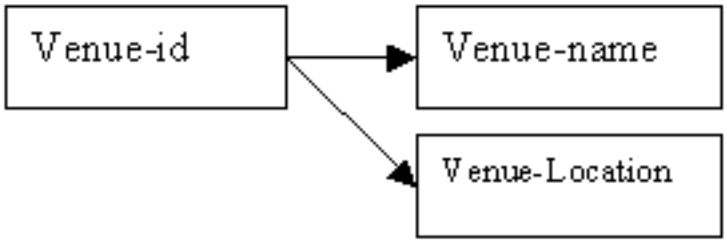
All agents are recorded in the Agents relation, where the primary key is agent-id, and the remaining attributes are agent-name and agent-location.



Agent-id	Agent-name	Agent-location
1295	Burton	Luton
1435	Nunn	Boston
1504	Lee	Taipei
1682	Tsang	Beijing
1460	Stritch	Rome
1522	Ellis	Madrid
1478	Burns	Leeds
1377	Webb	Sydney
1509	Patel	York
1190	Patel	Hue
1802	Chapel	Bristol
1076	Eccles	Oxford
1409	Arkley	York
1428	Vernon	Cairo

Venue details

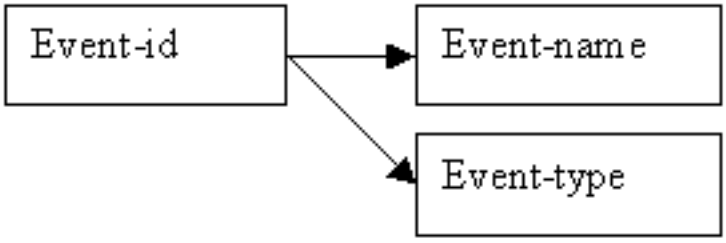
There are a number of venues available for bookings, and these are stored in the Venues relation. The primary key is venue-id, and the other attributes are venue-name and venue-location.



Venue-id	Venue-name	Venue-location
59	Atlas	Tokyo
35	Polis	Athens
54	Nation	Lisbon
79	Festive	Rome
46	Royale	Cairo
28	Gratton	Boston
75	Vostok	Kiev
84	State	Kiev
82	Tower	Lima
17	Silbury	Tunis
92	Palace	Milan
62	Shaw	Oxford

Event details

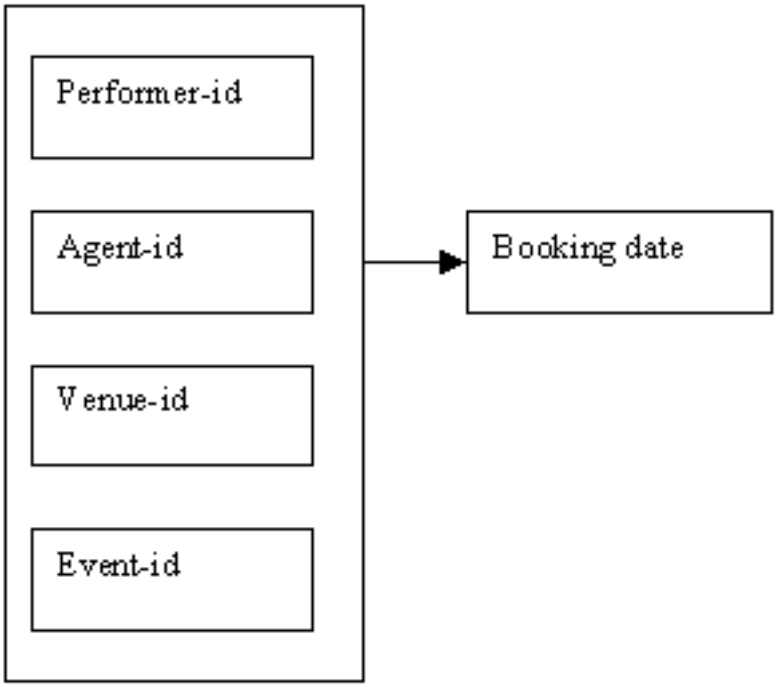
All events which can be booked are listed in the Events relation; the primary key is event-id, and the other attributes are event-name and event-type.



Event-id	Event-name	Event-type
901	The Dark	Drama
907	Elgar 1	Concert
913	What Now?	Drama
921	Silver Shoe	Ballet
926	Next Year	Drama
927	Chanson	Opera
934	Angels	Opera
938	New Dawn	Drama
941	Mahler 1	Concert
942	White Lace	Ballet
945	Trick-Treat	Variety show
952	Gold Days	Drama
957	Quicktime	Musical
959	Show Time	Musical
963	Vanish!	Magic show
964	The Friends	Drama
971	Card Trick	Magic show
978	Swift Step	Dance
981	Birdsong	Musical
988	Secret Tape	Drama

Booking details

Every booking made by an agent, for a performer, at a venue, for an event, is stored in the Bookings relation. The primary key is a combination of performer-id, agent-id, venue-id and event-id; the remaining attribute is booking date.


Performer-id
Agent-id
Venue-id
Event-id
Booking date

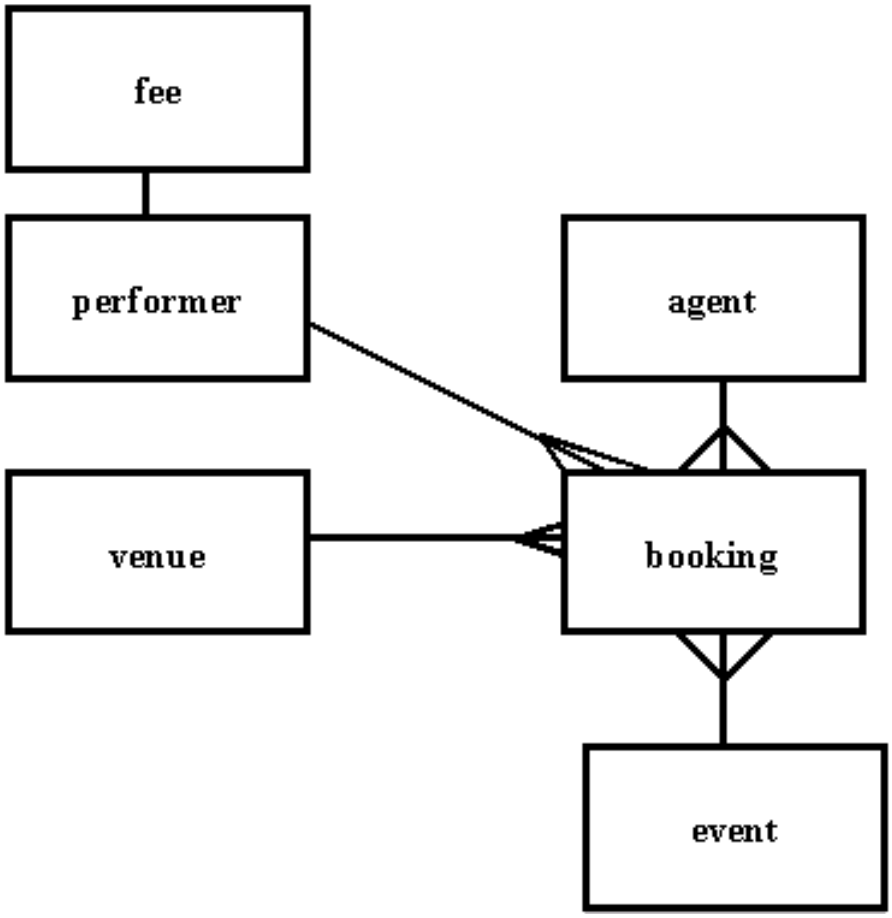
Performer-id	Agent-id	Venue-id	Event-id	Booking date
101	1295	59	959	25-Nov-1999
105	1435	35	921	07-Jan-2002
105	1504	54	942	10-Feb-2002
108	1682	79	901	29-Jul-2003
112	1460	17	926	13-Aug-2000
112	1522	46	952	05-May-1999
112	1504	75	952	16-Mar-1999
126	1509	59	945	02-Sep-2001
129	1478	79	926	22-Jun-2000
134	1504	28	981	18-Sep-2001
138	1509	84	957	18-Aug-1999
140	1478	17	963	18-Aug-1999
141	1478	84	941	21-Jul-2000
143	1504	79	927	21-Nov-2002
147	1076	17	952	30-Apr-2000
147	1409	79	988	17-Apr-2000
152	1428	59	978	01-Oct-2001

Note that this assumes there can only be one booking involving a particular combination of performer, agent, venue and event. This means that we cannot have multiple bookings made involving the same performer, agent, venue and event, as the primary key would be the same for each booking and we would therefore lose unique identification of bookings.

In order to accommodate multiple bookings involving the same entities, we could include the booking date as part of the key, but then we would not be able to distinguish between morning and evening performances on the same date (unless we included time as well as date).

Entity-relationship diagram

The determinacy diagrams and relations, which are now fully normalised, can also be viewed as entities linked by relationships using the data modelling technique described in the chapter on entity-relationship modelling. Each determinacy diagram represents a relation, which in turn corresponds to an entity, as can be seen in the entity-relationship diagram below. The relationships that exist between each entity are summarised below the diagram.



This entity relationship diagram represents the following:

- Each performer may have many bookings, so the relationship between performer and booking is one-to-many.
- A performer earns a fee, the value of which depends on the performer type, so the relationship between performer and fee is one-to-one (a performer can only be of one type).
- An agent may make a number of bookings, so the relationship between agent and booking is one-to-many.
- Any venue may have been booked several times, which makes the relationship between venue and booking one-to-many.
- Each event may be involved in a number of bookings, so this relationship is also one-to-many.

- The relationships that exist between performers, agents, venues and events are shown by their connections through the bookings.

Exercise 3

Why have so many normal forms?

Further issues in decomposing relations

When moving to a higher normal form, we often have a choice about the way in which we can decompose a relation into a number of other relations. This section examines problems that can arise if the wrong decomposition is chosen.

As an example, supposing within a government department responsible for intelligence gathering, we wish to record details of employees in the department, and the levels of their security clearance, which describe the levels of access employees have to secret information. The table might contain the following attributes:

Relation EMPLOYEE (Employee, Security_code, Level)

Where Employee is the primary key and provides some convenient means of identifying each employee, Security_code identifies the security clearance of that employee, and Level identifies the level of access to secret information possessed by anyone having that Security_code.

The determinants in this relation are:

Employee determines Security_code

Security_code determines Level

So we have two functional dependencies, respectively Security_code is functionally dependent on Employee, and Level is functionally dependent on Security_code. We also have a transitive, or indirect dependency, of Level on Employee; that is, an employee's level of security clearance does depend on who that employee is, but only via the value of their Security_code.

This relation is in second normal form; i.e. it contains no repeating groups and no part-key dependencies, but it does contain a transitive dependency.

In order to convert relation EMPLOYEE to third normal form, we need to decompose it to remove the transitive dependency of Level on Employee. Until we make this decomposition, we have the following insert, update and deletion anomalies:

- We cannot create a new Security_code until we have an Employee to whom we wish to allocate it.

- If we change the Level of a Security_code, i.e. change the Level of information employees who hold that code can access, then in relation EMPLOYEE, we would have to propagate the update throughout all the employees who hold that Level.
- If we remove the last Employee holding a particular Security_code, we loose the information about the Level of clearance assigned to that Security_code.

To perform the decomposition, suppose we split relation EMPLOYEE into two relations as follows:

Decomposition A

Relation EMPLOYEE-CLEARANCE (Employee, Level)

Relation SECURITY_LEVEL (Security_code, Level)

There are problems with this decomposition. Supposing we wish to change the security clearance for a given Employee. We can change the value of Level in relation SECURITY_CLEARANCE, but unfortunately, this update is not independent of the data held in relation SECURITY_LEVEL. In order for the change to have taken place in the SECURITY-CLEARANCE relation, one of two things must have arisen. Either the Employee in question has changed his/her Security_code, in which case no update need be made to relation SECURITY_LEVEL, or the Level associated with the Security_code possessed by the Employee has changed, in which case relation SECURITY_LEVEL must be changed to reflect this.

The problem has arisen because the two relations in decomposition A are not independent of one another. There is in fact a functional dependency between them: the fact that Security_code is functionally dependent on Employee. In decomposition A, instead of storing in the same relation those data items that are functionally dependent on one another, we have split the functional dependency of Security_code on Employee across the two relations, preserving the transitive dependency of Level on Employee in relation SECURITY_CLEARANCE. The problems this gives is that we cannot then make updates to one of these relations without considering whether updates are required to the other. As a further example, if we make updates to relation SECURITY_LEVEL, changing the Level of access associated with each Security_code, we must make sure that these updates are propagated to relation SECURITY_CLEARANCE, i.e. that the employees who possess the altered security codes have their Level attribute updated to reflect the changes in the SECURITY_LEVEL relation.

Resolution of the problem

The solution to this problem is to ensure that when making decompositions, we preserve the functional dependencies of data items within the resulting rela-

tions, rather than splitting them between the different relations. For the above example, the correct decomposition would therefore be as follows:

Decomposition B

Relation EMPLOYEE_CODE (Employee, Security_code)

Relation ACCESS_LEVEL (Security_code, Level)

This decomposition allows us to manipulate the level of security granted to an individual employee (in relation EMPLOYEE_CODE) independently of that which specifies in general the level of access associated with security codes (maintained in relation ACCESS_LEVEL).

Denormalisation and over-normalisation

Denormalisation

As the normalisation process progresses, the number of relations required to represent the data of the application being normalised increases. This can lead to performance problems when it comes to performing queries and updates on the implemented system, because the increased number of tables require multiple JOINs to combine data from different tables. These performance problems can be a major issue in larger applications (by larger we mean both in terms of numbers of tables and quantity of data).

To avoid these performance problems, it is often decided not to normalise an application all the way to fourth normal form, or, in the case of an existing application which is performing slowly, to denormalise an existing application. The process of denormalisation consists of reversing (or in the case of a new application, not carrying out in the first place) the steps to fully normalise an application. Whether this is appropriate for any given application depends critically on two factors:

- Whether the size of the application is sufficient that it will run slowly on the hardware/software platform being used.
- Whether failing to carry out certain steps in the normalisation process will compromise the requirements of the applications users.

Supposing, for example, we have a relation in which we wish to store the details of companies, the departments making up the companies and the locations of the departments. We might describe such a relation as follows:

Relation COMPANY (Company, Department, Location)

A row in the relation indicates that a particular Department of a specific Company is based at a particular Location. The primary key of relation COMPANY is the attribute Company. Assuming that Location depends directly on the Department of any particular Company, there is a transitive dependency between

Company and Location, via Department. To convert relation COMPANY to third normal form, we would decompose it into:

Relation DEPARTMENT (Company, Department)

Relation LOCATION (Department, Location)

This would avoid the insert, update and deletion anomalies associated with second normal form relations, giving us the ability to manipulate the information about which departments make up a particular company quite independently of the information about where particular departments are located. This is the additional flexibility provided by taking the step of converting the application to third normal form. However, if we wish to store information about a large number of companies and departments, and we will not need to manipulate the location information about departments independently of the information about which departments make up a company, then we may choose not to proceed to third normal form, but to leave relation COMPANY in second normal form. Thus we'd retain the Company, Department and Location attributes in one relation, where they can be queried and manipulated together, without the need for JOINs.

If we choose to leave relation COMPANY in second normal form, what we will have lost in terms of flexibility is as follows:

- The ability to create new departments at specific locations without allocating them to a specific company.
- The ability to create new departments at specific locations without allocating them to a specific company.

Note that in this particular case, the update anomaly does not arise, as each department is assumed to appear only once in relation COMPANY. Users of the application may feel that the flexibility provided by third normal form is simply not required in this application, in which case we can opt for the second normal form design, with its improved performance.

The same arguments apply when considering whether to take any steps that lead to a higher normal form; there is always a trade-off similar to the above, between the increased flexibility of a more normalised design versus a faster-running application in a less normalised design, due to the smaller number of relations. When Relational database systems first arrived in the early '80s, their performance was generally slow, and this had an influence in slowing down their adoption by some companies. Since that time, a huge amount of research and development has gone into improving the performance of Relational systems. This development work, plus the considerable improvements in the processing speed of hardware, tends to suggest that the need to denormalise applications should be reduced; however, it remains an important option for application designers seeking to develop well-tuned applications.

Over-normalisation

A further technique for improving the performance response of database applications, is that of over-normalisation. This technique is so-called because it results in a further decomposition of the relations of an application, but for different reasons than that of the usual normalisation process. In normalisation, we are seeking to satisfy user requirements for improved application flexibility, and to eliminate data redundancy. In contrast, the decompositions made during over-normalisation are generally done so to improve application performance.

As an example, we shall take the case of a company possessing a large table of customer information, supposing the table contains several thousand rows, and that the customers are more or less equally divided into those based in the home country of the company and overseas.

There are essentially two approaches to over-normalisation of a table — we can divide it up either horizontally or vertically.

Splitting a table horizontally

The most common approach is to split a table horizontally. In the case of the large customer table, we might split it into two tables, one for home-based customers, and the other for overseas customers.

Splitting a table vertically

The alternative approach of vertical partitioning might be used if the columns of a table fell naturally into two or more logical subsets of information; for example, if several columns of the customer table contained data specific to credit-limit assessment, whereas others contained more general contact and customer-profiling information. If this were the case, we might split the table vertically, one partition containing credit-limit assessment information, and the other containing the more general customer details. It is important when performing vertical partitioning in this way that the primary key of the entity involved, in this case, Customer, is retained in both partitions, enabling all of the data for the same entity instance (here for a specific customer) to be re-assembled through a JOIN.

Example of over-normalisation

So, for the process of over-normalisation, tables can be split into a number of horizontal or vertical fragments. For example, if the customers in the above example contained a 'region' attribute, which indicated in which region of the world they are based, then rather than a simple dual split into home-based and overseas customers, we might create a separate partition for each regional grouping of customers.

There are a number of reasons why relations may be fragmented in this way, most of which are directly concerned with improving performance. These objectives are briefly examined below:

- Splitting a large table into a number of smaller tables often reduces the number of rows or columns that need to be scanned by specific query or update transactions for an application. This is particularly true when the partitions created are a good match to different business functions of the application - for example, in the customer table example above, if customers in different regions of the world undergo different types of query and update processing.
- The smaller tables retrieved by queries restricted to one, or even a few, of a number of partitions will take up less space in main memory than the large number of rows fetched by a query on a large table. This often means that the small quantity of data is able to remain in memory, available for further processing if required, rather than being swapped back to disk, as would be likely to happen with a larger data set.

Just as a good match to business functions for the chosen partitions means the over-normalisation process will work well in improving performance, a poor match to transaction requirements could lead to a poorer performance, because the over-normalised design could lead to an increased number of JOINs.

Review questions

Review question 1

Consider the following scenario.

A database is being designed to store details of a hospital's clinics and the doctors who work in them. Each doctor is associated with just one hospital. Each clinic has a two-to-three-hour session during which a doctor who is a specialist in a particular field of medicine, sees patients with problems in that specialist area; for example, a diabetic clinic would be run by a doctor who is a specialist in diabetes. The same clinic may occur in a number of different hospitals; for example, several hospitals may run a diabetes clinic. Doctors may hold a number of different clinics. Clinic within the same hospital are always held by the same doctor. The relation is therefore of the following form:

Relation CLINIC (Hospital, Clinic, Doctor)

A row in the relation signifies that a particular clinic is held by a particular doctor in a specific hospital.

1. What are the determinants in the above relation?
2. Demonstrate that relation CLINIC is not in BCNF. Which normal form is it in?

3. Explain any insertion, update or deletion problems that might arise with relation CLINIC. How might these be resolved?

Review question 2

A library holds a database of the loans and services used by its members. Each member may borrow up to 10 books at a time, and may reserve sessions making use of library facilities, such as time on an Internet PC, a Multimedia PC, booking a ticket for a performance at the library theatre, etc.

Describe how the above scenario could be handled in the process of normalisation.

Review question 3

It is required to develop a database which can provide information about soccer teams: the number of games they have played, won, drawn and lost, and their current position in the league.

Write down your thoughts on the issues involved in supporting the requirement to provide the current league position, and how this is best satisfied.

Review question 4

If BCNF is a stronger definition of 3NF, and provides a more concise definition of the normalisation process, why is it worth understanding the step-by-step processes of moving from an un-normalised design to 3NF? Why is BCNF a stronger normal form than 3NF?

Review question 5

A binary relation is a relation containing just two attributes. Is it true that any binary relation must be in BCNF?

Review question 6

What is the difference between a repeating group and a multi-valued dependency?

Review question 7

True or false: Splitting a functional dependency between two relations when decomposing to a higher normal form is to be preferred to splitting a transitive dependency. Give reasons to justify your assertion.

Review question 8

Explain the difference between denormalisation and over-normalisation. What do the two techniques have in common, and what differences do they have?

Review question 9

A chemical plant produces chemical products in batches. Raw materials are fed through various chemical processes called a production run, which turns the raw materials into a final product. Each batch has a unique number, as

does each product produced by the plant. We can also assume that product names are unique. Each production run results in the production of a quantity of a particular product. We assume that only one product is produced in any given production run, and so different production runs are required for different products. The design of a relation to store the details of production runs could look as follows:

Relation PRODUCTION_RUN (Product_no, Product_name, Batch_no, Quantity)

In which normal form is relation PRODUCTION_RUN? Explain the reasoning behind your assertion.

Resolve any anomalies that could arise in the manipulation of rows in the PRODUCTION_RUN relation.

Review question 10

A company wishes to store details of its employees, their qualifications and hobbies. Each employee has a number of qualifications, and independently of these, a number of hobbies.

Produce a normalised design for storing this information.

Review question 11

We saw earlier the issues surrounding storing some types of derived data; for example, the league position of soccer teams. Supposing we wish to store the number of points accumulated by such teams, given the rules that:

- A win is awarded 3 points
- A draw is awarded 1 point
- There are no points for a defeat

Consider any problems that might be associated with the storage of the number of points obtained by teams in such a league.

Discussion topic

Normalisation has been the second major technique we have examined for use in the design of database applications, the other being entity-relationship modelling. You are encouraged to discuss your feelings about the relative usefulness of these two approaches with respect to the following:

- Learnability. Which of the two techniques have you found easier to learn? Do not settle for merely identifying which technique was easier to learn, but examine what it is about the techniques that makes one or other of them easier to learn.

- Usability. Which of the techniques have you found easier to use so far in the chapters you have worked through, and which would you expect to be more useful in commercial application development? Be sure to back your assertions with an explanation of why you believe them to be true.

Finally, consider what you believe the relative strengths and weaknesses of the two design approaches to be, and consider to what extent these are or are not complementary.