Chapter 11:
Databases

Learning objectives

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

- show understanding of the limitations of using a file-based approach for the storage and retrieval of data
- describe the features of a relational database which address the limitations of a file-based approach
- show understanding of and use the terminology associated with a relational database model
- use an entity–relationship (E–R) diagram to document a database design
- show understanding of the normalisation process
- explain why a given set of database tables are, or are not, in 3NF
- produce a normalised database design for a description of a database, a given set of data, or a given set of tables
- show understanding of the features provided by a Database Management System (DBMS) that address the issues of a file based approach
- show understanding of how software tools found within a DBMS are used in practice
- show understanding that DBMS carries out all creation/modification of the database structure using its Data Definition Language (DDL)
- show understanding that the DBMS carries out all queries and maintenance of data using its DML
- show understanding that the industry standard for both DDL and DML is Structured Query Language (SQL)
- Understand given SQL (DDL) commands and be able to write simple SQL (DDL) commands using a sub-set of commands
- write an SQL script to query or modify data (DML) which are stored in (at most two) database tables.

11.01 Limitations of a file-based approach

Data integrity problems in a single file

Let’s consider a simple scenario. A theatrical agency makes bookings for bands and is setting up a computerised system. Text files are to be used. One of these text files is to store data about individual band members. Each line of the file is to contain the following data for one band member:

- Name
- Contact details
- Banking details
- Band name
- Band agent name
- Band agent contact details

The intention is that this file could be used if the agency needed to contact the band member directly or through the band’s agent. It could also be used after a gig when the band member has to be paid. Ignoring what would constitute contact details or banking details, we can look at a snapshot of some of the data that might be stored for the member’s given name, the member’s family name and the band name. The file might have a thousand or more lines of text. The following is a selection of some of the data that might be contained in various lines in the file:

```
Xiangfei   Jha        ComputerKidz
Mahesh     Ravuru     ITWizz
Dylan      Stoddart
Graham     Vandana    ITWizz
Vandana    Graham     ITWizz
Mahesh     Ravuru     ITWizz
Precious   Olsen      ComputerKidz
Precious   Olsen      ITWizz
```

It is clear that there are problems with this data. It would appear that when the data for Vandana Graham was first entered, her names were inserted in the wrong order. A later correct entry was made without deletion of the original incorrect data. This type of problem is not unique to a file-based system. There is no validation technique that could detect the original error. By contrast, validation should have led to the correction of the missing band name for Dylan Stoddart. The Precious Olsen data are examples of duplication of data and inconsistent data.

There is also possibly an error that is not evident from looking at the file contents. A band name could be entered here when that band doesn’t exist.

The above discussion shows how a file-based approach can lead to data integrity problems in an individual file. The reason is the lack of in-built control when data is entered. The database approach can prevent such problems or, at least, minimise the chances of them happening.

The data privacy issue with a single file

A different problem is a lack of data privacy. The file above was designed so that the finance section could find banking details and the recruitment section could find contact details. The problem is that there cannot be any control of access to part of a file, so for example, staff in the recruitment section would be able to access the banking details of band members. Data privacy would be properly handled by a database system.

Data redundancy and possible inconsistency in multiple files

Mindful of this privacy problem, the agency decides to store data in different files for different departments of the organisation. Table 11.01 summarises the main data to be stored in each department's file.

Department	Data items in the section's file				
Contract	Member names		Band name	Gig details	
Finance	Member names	Bank details		Gig details	
Publicity			Band name	Gig details	
Recruitment	Member names		Band name		Agent details

Table 11.01 Data to be held in the department files

There is now data duplication across the files. This is commonly referred to as **data redundancy**. This does not mean that the data is no longer of use. Rather, it is a recognition that once data has been stored in one file there should be no need for it to be stored again in a different file. Unfortunately, some data redundancy cannot be avoided in file-based systems. This can lead to data inconsistency, either because of errors in the original entry or because of errors in subsequent editing. This is a different cause of data lacking integrity. One of the primary aims of the database approach is the elimination of data redundancy.

Data dependency concerns

The above account has focused on the problems associated with storing the data in the files. We now need to consider the problems that might occur when programs access the files.

Traditionally a programmer wrote a program and at the same time defined the data files that the program would need. For the agency, each department would have its own programs that would access the department's data files. When a programmer creates a program for a department, the programmer has to know how the data is organised in these files, for example, that the fourth item on a line in the file is a band name. This is an example of 'data dependency'.

It is very likely that the files used by one department might have some data which is the same as the data in the files of other departments. However, in the scenario presented above there is no plan for file sharing.

A further issue is that the agency might decide that there is a need for a change in the data stored. For instance, they might see an increasing trend for bands to perform with additional session musicians. Their data will need to be entered into some files. This will require the existing files to be re-written. In turn, this will require the programs to be re-written so that the new files are read correctly. In a database scenario the existing programs could still be run even though additional data was added. The only programming change needed would be the writing of additional programs to use this additional data.

The other aspect of data dependency is that when file structures have been defined to suit specific programs, they may not be suited to supporting new applications. The agency might feel the need for an information system to analyse the success or otherwise of the gigs they have organised over a number of years. Extracting the data for this from the sort of file-based system described here would be a complex task that would take considerable time to complete.

11.02 The relational database

In the relational database model, each item of data is stored in a **relation** which is a special type of table. The strange choice of name comes from a mathematical theory. A relational database is a collection of relational tables.

When a table is created in a relational database it is first given a name and then the attributes are named. In a database design, a table would be given a name with the **attribute** names listed in brackets after the table name. For example, the design for a database for the theatrical agency might contain the table definitions shown in Figure 11.01.

Member(MemberID, MemberGivenName, MemberFamilyName, BandName, ...)
Band(BandName, AgentID, ...)

Figure 11.01 Two tables in a database design for the theatrical agency

A logical view of some data stored in these tables is given in Table 11.02 and Table 11.03. Each attribute is associated with one column in the table and is in effect a column header. The entries in the rows beneath this column header are attribute values.

MemberID	MemberGivenName	MemberFamilyName	BandName	...
0005	Xiangfei	Jha	ComputerKidz	...
0009	Mahesh	Ravuru	ITWizz	...
0001	Dylan	Stoddart	ComputerKidz	...
0025	Vandana	Graham	ITWizz	...

Table 11.02 Logical view of the Member table in a relational database

BandName	AgentID	...
ComputerKidz	01	...
ITWizz	07	...

Table 11.03 Logical view of the Band table in a relational database

This is described as a logical view because an underlying principle for a relational database is that there is no ordering defined for the attribute columns. At least one database product does allow a view of a table and its contents. However, this is not in keeping with the fundamental relational database concept that a query should be used to inspect the data in a table. Queries are discussed later in the chapter.

A row in a relation should be referred to as a **tuple** but this formal name is not always used. Often a row is called a ‘record’ and the attribute values ‘fields’. The tuple is the collection of data stored for one ‘instance’ of the relation. In Table 11.02, each tuple relates to one individual band member. A fundamental principle of a relational database is that a tuple is a set of ‘atomic’ values; each attribute has one value or no value.

The most important feature of the relational database concept is the **primary key**. A primary key may be a single attribute or a combination of attributes. Every table must have a primary key and each tuple in the table must have a value for the primary key and that value must be unique.

Once a table and its attributes have been defined, the next task is to choose the primary key. In some cases there may be more than one attribute for which unique values are guaranteed. In this case, each one is a **candidate key** and one will be selected as the primary key. A candidate key that is not selected as the primary key is then referred to as a **secondary key**. Often there is no candidate key and so a primary key has to be created. The design in Figure 11.01 illustrates this with the introduction of the attribute MemberID as the primary key for the Member table. Note that the primary key is underlined

in the database design.

The primary key ensures integrity within the table. The DBMS will not allow an attempt to insert a value for a primary key when that value already exists. Therefore, each tuple automatically becomes unique. This is one of the features of the relational model that helps to ensure data integrity. The primary key also provides a unique reference to any attribute value that a query selects.

A database can contain stand-alone tables, but it is more usual for each table to have some relationship to another table. This relationship is implemented by using a **foreign key**.

Let's discuss the use of a foreign key using the database design shown in Figure 11.01. When the database is being created, the Band table is created first. BandName is chosen as the primary key because unique names for bands can be guaranteed. Then the Member table is created. MemberID is defined as the primary key and the attribute BandName is identified as a foreign key referencing the primary key in the Band table. Once this relationship between primary and foreign keys has been established, the DBMS will prevent any entry for BandName in the Member table being made if the corresponding value does not exist in the Band table. This provides **referential integrity** which is another reason why the relational database model helps to ensure data integrity.

Question 11.01

BandName is a primary key for the Band table. Does this mean that as a foreign key in the Member table it must have unique values? Explain your reasoning.

11.03 Entity–relationship modelling

We can use a top-down method called stepwise refinement to break down the process of database design into simple steps (see also Chapter 12, Section 12.08). At each step more detail is added to the design. In database design this approach uses an entity-relationship (E–R) diagram. Typically, this can be created either by a database designer or a systems analyst working with the designer. We introduced the term ‘relationship’ earlier in connection with the use of a foreign key. An entity (strictly speaking an entity type) could be a thing, a type of person, an event, a transaction or an organisation. Most importantly, there must be a number of ‘instances’ of the entity. An entity is something that will become a table in a relational database.

WORKED EXAMPLE 11.01

Creating an entity–relationship diagram for the theatrical agency

Let’s consider a scenario for the theatrical agency which will be sufficient to model a part of the final database they would need. The starting point for a top-down design is a statement of the requirement:

The agency needs a database to handle bookings for bands. Each band has a number of members. Each booking is for a venue. Each booking might be for one or more bands.

Step 1: Choose the entities

You look for the nouns. You ignore ‘agency’ because there is only the one. You choose Booking, Band, Member and Venue. For each of these there will be more than one instance. You are aware that each booking is for a gig at a venue but you ignore this because you think that the Booking entity will be sufficient to hold the required data about a gig.

Step 2: Identify the relationships

This requires experience, but the aim is not to define too many. You choose the following three:

- Booking with Venue
- Booking with Band
- Band with Member.

You ignore the fact that there will be, for example, a relationship between Member and Venue because you think that this will be handled through the other relationships that indirectly link them. You can now draw a preliminary E–R diagram as shown in Figure 11.02.

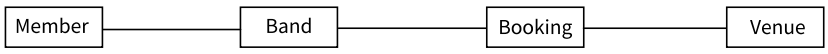


Figure 11.02 A preliminary entity–relationship diagram

Step 3: Decide the cardinalities of the relationships

Now comes the crucial stage of deciding on what are known as the ‘cardinalities’ of the relationships. At present we have a single line connecting each pair of entities. This line actually defines two relationships which might be described as the ‘forward’ one and the ‘backward’ one on the diagram as drawn. However, this only becomes apparent at the final stage of drawing the relationship. First, we have to choose one of the following descriptions for the cardinality of each relation:

- one-to-one or 1:1
- one-to-many or 1:M
- many-to-one or M:1
- many-to-many or M:M.

Let’s consider the relationship between Member and Band. We argue that one Member is a

member of only one Band. (This needs to be confirmed as a fact by the agency.) We then argue that one Band has more than one Member so it has many. Therefore, the relationship between Member and Band is M:1. In its simplest form, this relationship can be drawn as shown in Figure 11.03.

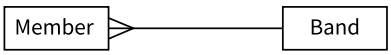


Figure 11.03 The M:1 relationship between Member and Band

This can be given more detail by including the fact that a member must belong to a Band and a Band must have more than one Member. To reflect this, the relationship can be drawn as shown in Figure 11.04.

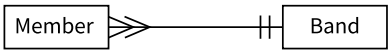


Figure 11.04 The M:1 relationship with more detail

At each end of the relationship there are two symbols. One of the symbols shows the minimum cardinality and the other the maximum cardinality. In this particular case, the minimum and maximum values just happen to be the same. However, using the diagram to document that a Member must belong to a Band is important. It indicates that when the database is created it must not be possible to create a new entry in the Member table unless there is a valid entry for BandName in that table.

For the relationship between Booking and Venue we argue that one Booking is for one Venue (there must be a venue and there cannot be more than one) and that one Venue can be used for many Bookings so the relationship between Booking and Venue is M:1. However, a Venue might exist that has so far never had a booking so the relationship can be drawn as shown in Figure 11.05.

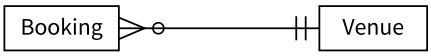


Figure 11.05 The M:1 relationship between Booking and Venue

Finally for the relationship between Band and Booking we argue that one Booking can be for many Bands and that one Band has many Bookings (hopefully!) so the relationship is M:M. However, a new band might not yet have a booking. Also, there might be only one Band for a booking so the relationship can be drawn as shown in Figure 11.06.

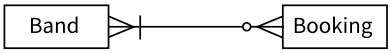


Figure 11.06 The M:M relationship between Band and Booking

Step 4: Create the full E–R diagram

At this stage we should name each relationship. The full E–R diagram for the limited scenario that has been considered is as shown in Figure 11.07.

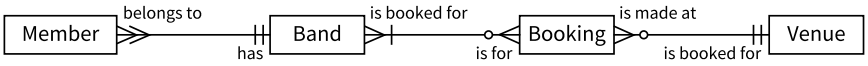


Figure 11.07 The E–R diagram for the theatrical agency's booking database

To illustrate how the information should be read from such a diagram we can look at the part shown in Figure 11.08. Despite the fact that there is a many-to-many relationship, a reading of a relationship always considers just one entity to begin the sentence. So, reading forwards and then backwards, we say that:

One Band is booked for zero or many Bookings

One Booking is for one or many Bands

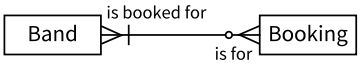


Figure 11.08 Part of the annotated E–R diagram

Question 11.02

If you are deciding on the cardinality of the relationship between two entities does it matter which one is put on the left and which on the right?

TIP

Be careful not to confuse the two completely different terms relation and relationship.

11.04 A logical entity–relationship model

A fully annotated E–R diagram of the type developed in Section 11.03 holds all of the information about the relationships that exist for the data that is to be stored in a system. It can be defined as a conceptual model because it does not relate to any specific way of implementing a system. If the system is to be implemented as a relational database, the E–R diagram has to be converted to a logical model. To do this we can start with a simplified E–R diagram that just identifies cardinalities.

If a relationship is 1:M, no further refinement is needed. The relationship shows that the entity at the many end needs to have a foreign key referencing the primary key of the entity at the one end.

If there were a 1:1 relationship there are options for implementation. However, such relationships are extremely rare and we do not need to consider them here.

The problem relationship is the M:M, where a foreign key cannot be used. A foreign key attribute can only have a single value, so it cannot handle the many references required. Another way of looking at this problem is to argue that a foreign key is required in each entity but neither table could be created first because the other table needed to exist for the foreign key to be defined. The solution for the M:M relationship is to create a link entity. For Band and Booking, the logical entity model will contain the link entity shown in Figure 11.09.

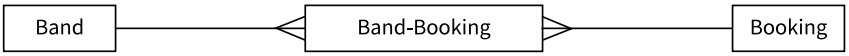


Figure 11.09 A link entity inserted to resolve a M:M relationship

Extension Question 11.01

Is it possible to annotate these relationships?

With the link entity in the model it is now possible to have two foreign keys in the link entity; one referencing the primary key of Band and one referencing the primary key of Booking.

Each entity in the logical E–R diagram will become a table in the relational database. It is therefore possible to choose primary keys and foreign keys for the tables. These can be summarised in a key table. Table 11.04 shows sensible choices for the theatrical agency's booking database.

Table name	Primary key	Foreign key
Member	MemberID	BandName
Band	BandName	
Band-Booking	BandName & BookingID	BandName, BookingID
Booking	BookingID	VenueName
Venue	VenueName	

Table 11.04 A key table for the agency booking database

The decisions about the primary keys are determined by the uniqueness requirement. The link entity cannot use either BandName or BookingID alone but the combination of the two in a compound primary key will work.

TASK 11.01

Consider the following scenario. An organisation books cruises for passengers. Each cruise visits a number of ports. Create a conceptual E–R diagram and convert it to a logical E–R diagram. Create a key table for the database that could be implemented from the design.

11.05 Normalisation

Normalisation is a design technique for constructing a set of table designs from a list of data items. It can also be used to improve on existing table designs.

TIP

Unfortunately, you will be coming across a completely different use of the term normalisation in Chapter 16.

WORKED EXAMPLE 11.02

Normalising data for the theatrical agency

To illustrate the technique, let's consider the document shown in Figure 11.10. This is a booking data sheet that the theatrical company might use.

Booking data sheet: 2016/023

Venue:

Cambridge International Theatre

Camside

CA1

Booking date: 23.06.2016

Bands booked	Number of band members	Headlining
ComputerKidz	5	Y
ITWizz	3	N
DeadlyDuo	2	N

Figure 11.10 Example booking data sheet

The data items on this sheet (ignoring headings) can be listed as a set of attributes:

(BookingID, VenueName, VenueAddress1, VenueAddress2, Date,

(BandName, NumberOfMembers, Headlining))

The list is put inside brackets because we are starting a process of table design. The extra set of brackets around BandName, NumberOfMembers, Headlining is because they represent a **repeating group**. If there is a repeating group, the attributes cannot sensibly be put into one relational table. A table must have single rows and atomic attribute values so the only possibility would be to include tuples such as those shown in Table 11.05. There is now data redundancy here with the duplication of the BookingID, venue data and the date.

Booking ID	Venue Name	Venue Address1	Venue Address2	Date	Band Name	Number Of Members	Headlining
2016/023	Cambridge International Theatre	Camside	CA1	23.06.2016	Computer Kidz	5	Y
2016/023	Cambridge International Theatre	Camside	CA1	23.06.2016	ITWizz	3	N
2016/023	Cambridge International	Camside	CA1	23.06.2016	DeadlyDuo	2	N

	Theatre						

Table 11.05 Data stored in an unnormalised table

Step 1: Conversion to first normal form (1NF)

The conversion to first normal form (1NF) requires splitting the data into two groups. At this stage we represent the data as table definitions. Therefore, we have to choose table names and identify a primary key for each table. One table contains the non-repeating group attributes, the other the repeating group attributes. For the first table a sensible design is:

Booking(BookingID, VenueName, VenueAddress1, VenueAddress2, Date)

The table with the repeating group is not so straightforward. It needs a compound primary key and a foreign key to give a reference to the first table. The sensible design is:

Band-Booking(BandName, BookingID(fk), NumberOfMembers, Headlining)

Again, the primary key is underlined but also the foreign key has been identified, with (fk). Because the repeating groups have been moved to a second table, these two tables could be implemented with no data redundancy in either. This is one aspect of 1NF. Also, we can say that for each table the attributes are dependent on the primary key.

Step 2: Conversion to second normal form (2NF)

For conversion to second normal form (2NF), the process is to examine each non-key attribute and ask if it is dependent on both parts of the compound key. Any attributes that are dependent on only one of the attributes in the compound key must be moved out into a new table. In this case, NumberOfMembers is only dependent on BandName. In 2NF there are now three table definitions:

Booking(BookingID, VenueName, VenueAddress1, VenueAddress2, Date)

Band-Booking(BandName(fk), BookingID(fk), Headlining)

Band(BandName, NumberOfMembers)

Note that the Booking table is unchanged from 1NF. The Booking table is automatically in 2NF; only tables with repeating group attributes have to be converted. The Band-Booking table now has two foreign keys to provide reference to data in the other two tables. The characteristics of a table in 2NF is that it either has a single primary key or it has a compound primary key with any non-key attribute dependent on both components.

Step 3: Conversion to third normal form (3NF)

For conversion to third normal form (3NF) each table has to be examined to see if there are any non-key dependencies; that means we must look for any non-key attribute that is dependent on another non-key attribute. If there is, a new table must be defined.

In our example, VenueAddress1 and VenueAddress2 are dependent on VenueName. With the addition of the fourth table we have the following 3NF definitions:

Band(BandName, NumberOfMembers)

Band-Booking(BandName(fk), BookingID(fk), Headlining)

Booking(BookingID, Date, VenueName(fk))

Venue(VenueName, VenueAddress1, VenueAddress2)

Note that once again a new foreign key has been identified to keep a reference to data in the newly created table. These four table definitions match four of the entities in the logical E–R model for which the keys were identified in Table 11.04. This will not always happen. A logical E–R diagram will describe a 2NF set of entities but not necessarily a 3NF set.

To summarise, if a set of tables are in 3NF it can be said that each non-key attribute is dependent on the key, the whole key and nothing but the key.

Question 11.03

In Step 2 of Worked Example 11.02, why is the Headlining attribute not placed in the Band table?

TASK 11.02

Normalise the data shown in Figure 11.11.

Order no: 07845				**Date:** 25-06-2016
Customer no: 056		**Customer name:**	CUP	
		Address:	Cambridge square	Cambridge
Sales rep no:	2	**Sales Rep name:**	Dylan Stoddart	

Product no	Description	Quantity	Price / unit	Total
327	Inkjet cartridges	24	$30	$720
563	Laser toner	5	$25	$125
		Total Price		$835

Figure 11.11 An order form

11.06 The Database Management System (DBMS)

The database approach

It is vital to understand that a database is not just a collection of data. A database is an implementation according to the rules of a theoretical model. The basic concept was proposed some 40 years ago by ANSI (American National Standards Institute) in its three-level model. The three levels are:

- the external level
- the conceptual level
- the internal level.

The architecture is illustrated in Figure 11.12 in the context of a database to be set up for our theatrical agency.

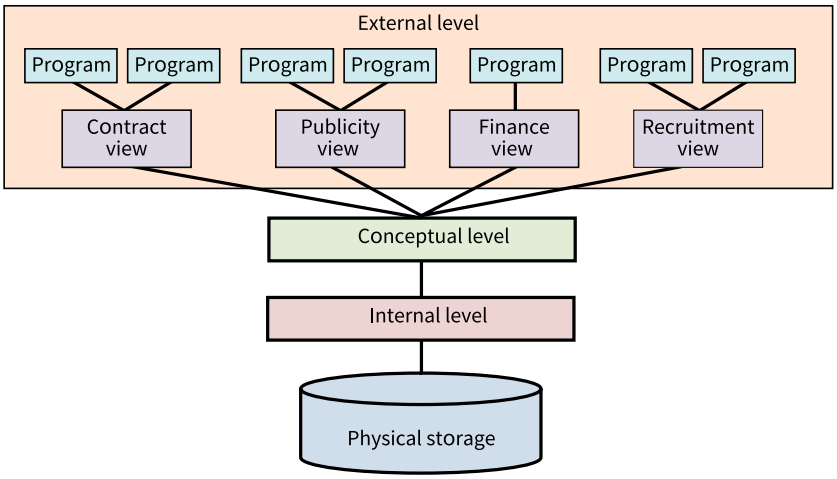


Figure 11.12 The ANSI three-level architecture for the theatrical agency database

The physical storage of the data is represented here as being on disk. The details of the storage (the internal schema) are known only at the internal level, the lowest level in the ANSI architecture. This is controlled by the **database management system (DBMS)** software.

The programmers who wrote this software are the only ones who know the structure for the storage of the data on disk. The software will accommodate any changes that might be needed in the storage medium.

At the next level, the conceptual level, there is a single universal view of the database. This is controlled by the **database administrator (DBA)** who has access to the DBMS. In the ANSI architecture the conceptual level has a conceptual schema describing the organisation of the data as perceived by a user or programmer. This may also be described as a logical schema.

At the external level there are individual user and programmer views. Each view has an external schema describing which parts of the database are accessible. A view can support a number of user programs.

An important aspect of the provision of views is that they can be used by the DBA as a mechanism for ensuring security. Individual users or groups of users can be given appropriate access rights to control what actions are allowed for that view. For example, a user may be allowed to read data but not to amend data. Alternatively, there may only be access to a limited number of the tables in the database.

The facilities provided by a DBMS

You need to remember that databases come in a variety of forms ranging from a simple system created for one individual through to the central database for some large organisation. Some of the facilities provided by a DBMS are only relevant for large organisations, when their use will be controlled by a DBA.

Whatever the size of the database, one option for its creation is to use the special-purpose language SQL which is discussed in the next section of this chapter. There are alternatives to SQL for most types of DBMS. The DBMS provides software tools through a **developer interface**. These allow for tables to be created and attributes to be defined together with their data types. In addition, the DBMS provides facilities for a programmer to develop a user interface. It also provides a **query processor** that allows a query to be created and processed. The **query** is the mechanism for extracting and manipulating data from the database. The other feature likely to be provided by the DBMS is the capability for creating a report to present formatted output. A programmer can incorporate access to queries and reports in the user interface.

DBMS functions likely to be used by a DBA

The DBA is responsible for setting up the user and programmer views and for defining the appropriate, specific access rights.

An important feature of the DBMS is the data dictionary which is part of the database that is hidden from view from everyone except the DBA. It contains metadata about the data. This includes details of all the definitions of tables, attributes and so on but also of how the physical storage is organised.

There are a number of features that can improve performance. Of special note is the capability to create an **index** for a table. This is needed if the table contains a large number of attributes and a large number of tuples. An index is a secondary table that is associated with an attribute that has unique values. The index table contains the attribute values and pointers to the corresponding tuples in the original table. The index can be on the primary key or on a secondary key. Searching an index table is much quicker than searching the full table.

The integrity of the data in the database is a key concern. One potential cause of problems occurs when a transaction is started but a system problem prevents its completion. The result would be a database in an undefined state. The DBMS should have a built-in feature that prevents this from happening. As with all systems, regular backup is a requirement. The DBA will be responsible for backup of the stored data.

Discussion Point:

How many of the above concepts are recognisable in your experience of using a database?

11.07 Structured Query Language (SQL)

SQL is the programming language provided by a DBMS to support all of the operations associated with a relational database. Even when a database package offers high-level software tools for user interaction, they create an implementation using SQL.

Data Definition Language (DDL)

Data Definition Language (DDL) is the part of SQL provided for creating or altering tables. These commands only create the structure. They do not put any data into the database.

The following are some examples of DDL that could be used in creating the database designed in Worked example 11.02 for the theatrical agency:

```
CREATE DATABASE BandBooking;
CREATE TABLE Band (
     BandName varchar(25),
     NumberOfMembers integer);
ALTER TABLE Band ADD PRIMARY KEY (BandName);
ALTER TABLE Band-Booking ADD FOREIGN KEY (BandName REFERENCES
Band(BandName);
```

These examples illustrate a number of general points regarding the writing of SQL.

- The SQL consists of a sequence of commands.
- Each command is terminated by;
- A command can occupy more than one line.
- There is no case sensitivity.
- There has been a decision made here to use upper case for the commands and lower case for table names, attribute names and datatypes.
- When a command contains a list of items these are separated by a comma.
- For the CREATE TABLE command this list is enclosed in parentheses

These examples show that once the database has been created, the tables can be created and the attributes defined. It is possible to define a primary key and a foreign key within the `CREATE TABLE` command but the `ALTER TABLE` command can be used as shown (it can also be used to add extra attributes).

When an attribute is defined, its data type must be specified. As with procedural languages there can be different data types or different names for data types depending on which DBMS is being used. One feature common to all databases is that the number of characters allowed for an attribute can be defined by including the number in brackets. In the above example `BandName varchar(25)` allows up to 25 characters for the band name.

The following list shows some of the names that might be used to define a data type: character, varchar, boolean, integer, real, date, time. In this chapter these will be written in lower case, but you might see them written in upper case in other sources.

TASK 11.03

For the database defined in Worked Example 11.02, complete the DDL for creating the four tables. Use varchar(8) for BookingID, integer for NumberOfMembers, date for Date, character for Headlining and varchar(25) for all other data.

Data Manipulation Language (DML)

There are three categories of use for Data Manipulation Language (DML)

- The insertion of data into the tables when the database is created
- The modification or removal of data in the database
- The reading of data stored in the database

The following illustrate the two possible ways that SQL can be written to populate a table with data:

```
INSERT INTO Band (‘ComputerKidz’, 5);
INSERT INTO Band-Booking (BandName, BookingID)
VALUES (‘ComputerKidz’,‘2016/023’);
```

The first example shows a simpler version that can be used if the order of the attributes is known. The second shows the safer method; the attributes are defined then the values are listed. The following are some points to note.

- Parentheses are used in both versions.
- A separate INSERT command has to be used for each tuple in the table.
- There is an order defined for the attributes.
- Although the SQL will have a list of INSERT commands the subsequent use of the table has no concept of the tuples being ordered.

The main use of DML is to obtain data from a database using a query. A query always starts with the `SELECT` command.

The simplest form for a query has the attributes for which values are to be listed as output identified after SELECT and the table name identified after FROM. For example:

```
SELECT BandName FROM Band;
```

Note that the components of the query are separated by spaces.

The Band table only has two attributes. To list the values for both there are two options:

```
SELECT BandName, NumberOfMembers
FROM Band;
```

or

```
SELECT * FROM Band;
```

which uses * to indicate all attributes. Note that in the first example the attributes are separated by commas but no parentheses are needed.

It is possible to include instructions in the SQL to control the presentation of the output. The following uses ORDER BY to ensure that the output is sorted to show the data with the band names in alphabetical order.

```
SELECT BandName, NumberOfMembers
FROM Band
ORDER BY BandName;
```

In this query there is no question of duplicate entries because BandName is the primary key of the BandName table. However, in the Band-Booking table an individual value for BandName will occur many times. If a query were being used to find which bands already had a booking there would be repeated names in the output. This can be prevented by the use of GROUP BY as shown here:

```
SELECT BandName
FROM Band-Booking
GROUP BY BandName;
```

An extension of the control of the output from a query is to include a condition to limit the selected data. This is provided by a WHERE clause. The following are examples:

```
SELECT BandName
FROM Band-Booking
WHERE Headlining = ‘Y’
GROUP BY BandName;
```

which produces a single output for each band that has headlined. Note how a query can have several

component parts which are best presented on separate lines.

```
SELECT BandName, NumberOfMembers
FROM Band
WHERE NumberOfMembers > 2
ORDER BY BandName;
```

which excludes any duo bands.

It is possible to qualify the SELECT statement by using a function. SUM, COUNT and AVG are examples of functions that work on data held in several tuples for a particular attribute and return one value. For this reason, these functions are called aggregate functions. As an example, the following code displays the number of members in a band:

```
SELECT Count(*)
FROM Band;
```

This is a special case because there is no need to specify the attribute. An example using a specific attribute would be:

```
SELECT AVG(NumberOfMembers)
FROM Band;
```

another example is:

```
SELECT SUM(NumberOfMembers)
FROM Band;
```

A query can be based on a 'join condition' between data in two tables. The most frequently used is an inner join which is illustrated by:

```
SELECT VenueName, Date
FROM Booking
WHERE Band-Booking.BookingID = Booking.BookingID
AND Band-Booking.BandName = 'ComputerKidz';
```

The SQL uses the full definitive name for each attribute with the table name and attribute name separated by a dot. The query contains two conditions. The way that the query works is as follows.

- The Band-Booking table is searched for instances where the BandName is ComputerKidz.
- For each instance the BookingID is noted.
- Then there is a search of the Booking table to find the examples of tuples having this value for BookingID.
- For each one found the VenueName and Date are presented in the output.

Some versions of SQL require the explicit use of INNER JOIN. The following is a possible generic syntax:

```
SELECT table1.column1, table2.column2...
FROM table1
INNER JOIN table2
ON table1.common_field = table2.common_field;
```

The other use of DML is to modify the data stored in the database. The UPDATE command is used to change the data. If the band ComputerKidz recruited an extra member the following SQL would make the change needed.

```
UPDATE Band
SET NumberOfMembers = 6
WHERE BandName = 'ComputerKidz';
```

Note the use of the WHERE clause. If you forgot to include this the UPDATE command would change the number of band members to 6 for all of the bands.

The DELETE command is used to remove data from the database. This has to be done with care. If the ITWizz band decided to disband the following SQL would remove the name from the database.

```
DELETE FROM Band-Booking
WHERE BandName = 'ITWizz';
DELETE FROM Band
```

```
WHERE BandName = 'ITWizz';
```

Note that if an attempt was made to carry out the deletion from Band first there would be an error. This is because BandName is a foreign key in Band-Booking. Any entry for BandName in Band-Booking must have a corresponding value in Band.

Reflection Point:

Did you find normalisation difficult? It would be surprising if you didn't. Are you going to get as much practice as possible? There are many questions from previous exam papers that contain examples to try.

Summary

- A database offers improved methods for ensuring data integrity compared to a file-based approach.
- A relational database comprises tables of a special type; each table has a primary key and may contain foreign keys.
- Entity–relationship modelling is a top-down approach to database design.
- Normalisation is a database design method that starts with a collection of attributes and converts them into first normal form then into second normal form and, finally, into third normal form.
- A database architecture provides, for the user, a conceptual level interface to the stored data.
- Features provided by a database management system (DBMS) include: a data dictionary, indexing capability, control of user access rights and backup procedures.
- Structured Query Language (SQL) includes data definition language (DDL) commands for establishing a database and data manipulation language (DML) commands for creating queries.

Exam-style Questions

1 **a** A relational database has been created to store data about subjects that students are studying. The following is a selection of some data stored in one of the tables. The data represents the student's name, the personal tutor group, the personal tutor, the subject studied, the level of study and the subject teacher but there are some data missing:

Xiangfei	3	MUB	Computing	A	DER
Xiangfei	3	MUB	Maths	A	BNN
Xiangfei	3	MUB	Physics	AS	DAB
Mahesh	2	BAR	History	AS	IJM
Mahesh	2	BAR	Geography	AS	CAB

i Define the terms used to describe the components in a relational database table using examples from this table. [2]

ii If this represented all of the data, it would have been impossible to create this table. Identify what has not been shown here and must have been defined to allow the creation as a relational database table? Explain your answer and suggest a solution to the problem. [4]

iii Is this table in first normal form (1NF)? Explain your reason. [2]

b It has been suggested that the database design could be improved. The design suggested contains the following two tables:

Student(StudentName, TutorGroup, Tutor)

StudentSubject(StudentName, Subject, Level, SubjectTeacher)

i Identify features of this design which are characteristic of a relational database. [3]

ii Explain why the use of StudentName here is a potential problem. [2]

iii Explain why the Student table is not in third normal form (3NF). [2]

2 Consider the following scenario:

A company provides catering services for clients who need special-occasion, celebratory dinners. For each dinner, a number of dishes are to be offered. The dinner will be held at a venue. The company will provide staff to serve the meals at the venue.

The company needs a database to store data related to this business activity.

a An entity–relationship model is to be created as the first step in a database design. Identify a list of entities. [4]

b Identify pairs of entities where there is a direct relationship between them. [4]

c For each pair of entities, draw the relationship and justify the choice of cardinality illustrated by the representation. [6]

3 Consider the following booking form used by a travel agency.

Booking Number 00453

Hotel: Esplanade
Colwyn Bay
North Wales

Rating: ***

Date	Room type	Number of rooms	Room rate
23/06/2016	Front-facing double	2	$80
23/06/2016	Rear-facing double	1	$65
24/06/2016	Front-facing double	2	$80

a Identify an unnormalised list of attributes using the data shown in this form. Make sure that you distinguish between the repeating and non-repeating attributes. [5]

b Demonstrate the conversion of the data to first normal form (1NF). The design of two tables should be defined with the keys identified. [3]

c Identify the appropriate table and demonstrate the conversion of the table to two tables in second normal form (2NF). Explain your choice of table to modify. Explain your identification of the keys for these two new tables. [5]

d Identify which part of your design is not in Third Normal Form (3NF). [2]

4 A small database is to be created with the following three tables:

STUDENT(StudentID, StudentName, StudentOtherName, DateOfbirth)

SUBJECT(SubjectName, SubjectTeacher)

TUTORIAL(StudentID(fk), Subjectname(fk), WeekNumber, Day, PeriodNumber)

a Using the appropriate datatypes from the following list:

CHARACTER, VARCHAR, BOOLEAN, INTEGER, REAL, DATE

Write the SQL scripts to create two of the tables using the CREATE TABLE command. Do not at this stage identify any keys. [5]

b Assuming that all three tables have been created, write the SQL scripts to assign the primary key in the SUBJECT table and the two foreign keys in the TUTORIAL table. [3]

c Write the SQL script that will list all of the student names in age order. [5]

d There is an aspect of the design of the tables that could cause problems. Explain this problem. [2]

5 A school stores a large amount of data. This includes student attendance, qualification and contact details. The school's software uses a file-based approach to store this data.

a The school is considering changing to a DBMS.

i State what DBMS stands for. [1]

ii Describe **two** ways in which the database Administrator (DBA) could use the DBMS software to ensure the security of the student data. [4]

iii A feature of the DBMS software is a query processor.

Describe how the school secretary could use this software. [2]

iv The DBMS has replaced software that used a file-based approach with a relational database.

Describe how using a relational database has overcome the previous problems associated with a file-based approach. [3]

b The database design has three tables to store the classes that students attend.

```
STUDENT (StudentID, FirstName, LastName, Year, TutorGroup)
CLASS (ClassID, Subject)
CLASS -GROUP (StudentID, ClassID)
```

Primary keys are not shown.

There is a one-to-many relationship between CLASS and CLASS –GROUP.

i Describe how this relationship is implemented. [2]

ii Describe the relationship between CLASS –GROUP and STUDENT. [1]

iii Write an SQL script to display the `StudentID` and `FirstName` of all students who are in the tutor group 10B. Display the list in alphabetical order of `LastName`. [4]

iv Write an SQL script to display the `LastName` of all students who attend the class whose `ClasstID` is CS1 [4]

Cambridge International AS & A level Computer Science 9608 paper 11 Q8 June 2016

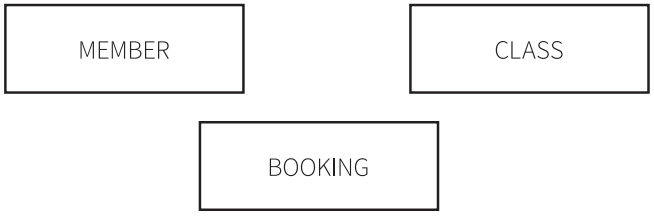
6 A health club offers classes to its members. A member needs to book into each class in advance.

a The health club employs a programmer to update the class booking system. The programmer has to decide how to store the records. The choice is between using a relational database or a file-based approach.

Give **three** reasons why the programmer should use a relational database. [6]

b The programmer decides to use three tables: `MEMBER`, `BOOKING` and `CLASS`.

Complete the entity–relationship (E–R) diagram to show the relationships between these tables.



[2]

c The `Class` table has primary key `Class ID` and stores the following data:

ClassID	Description	StartDate	ClassTime	NoOfSessions	Adultsonly
DAY01	Yoga beginners	12/01/2016	11:00	5	TRUE
EVE02	Yoga beginners	12/01/2016	19:00	5	FALSE
DAY16	Circuits	30/06/2016	10:30	4	FALSE

Write an SQL script to create the `CLASS` table. [6]

Cambridge international AS & A Level Computer Science 9608 paper 12 Q9 November 2016