

Chapter 3.6 Databases

3.6 (a) Flat files and relational databases

A database is an organized collection of data for one or more purposes, usually in digital form. The data are typically organized to model relevant aspects of reality (for example, the availability of rooms in hotels), in a way that supports processes requiring this information (for example, finding a hotel with vacancies). This definition is very general, and is independent of the technology used.

Originally all data were held in files. A typical file would consist of a large number of records each of which would consist of a number of fields. Each field would have its own data type and hold a single item of data. Typically a stock file would contain records describing stock. Each record may consist of the following fields.

Field Name	Data Type
Description	String
Cost Price	Currency
Selling Price	Currency
Number in Stock	Integer
Reorder Level	Integer
Supplier Name	String
Supplier Address	String

This led to very large files that were difficult to process. Suppose we want to know which items need to be reordered. This is fairly straightforward, as we only need to sequentially search the file and, if Number in Stock is less than the Reorder Level, make a note of the item and the supplier and output the details.

The problem is when we check the stock the next day, we will create a new order because the stock that has been ordered has not been delivered. To overcome this we could introduce a new field called On Order of type Boolean. This can be set to True when an order has been placed and reset to False when an order has been delivered. Unfortunately it is not that easy.

The original software is expecting the original seven fields not eight fields. This means that the software designed to manipulate the original file must be modified to read the new file layout.

Further ad hoc enquiries are virtually impossible. What happens if management ask for a list of best selling products? The file has not been set up for this and to change it so that such a request can be satisfied in the future involves modifying all existing software. Further, suppose we want to know which products are supplied by Food & Drink Ltd.. In some cases the company's name has been entered as Food & Drink Ltd., sometimes as Food and Drink Ltd. and sometimes the full stop after Ltd has been omitted. This means that a match is very difficult because the data is inconsistent. Another problem is that each time a new product is added to the database both the name and address of the supplier must be entered. This leads to redundant data or data duplication.

The following example, shown in Fig. 3.6.a.1, shows how data can be proliferated when each department keeps its own files.

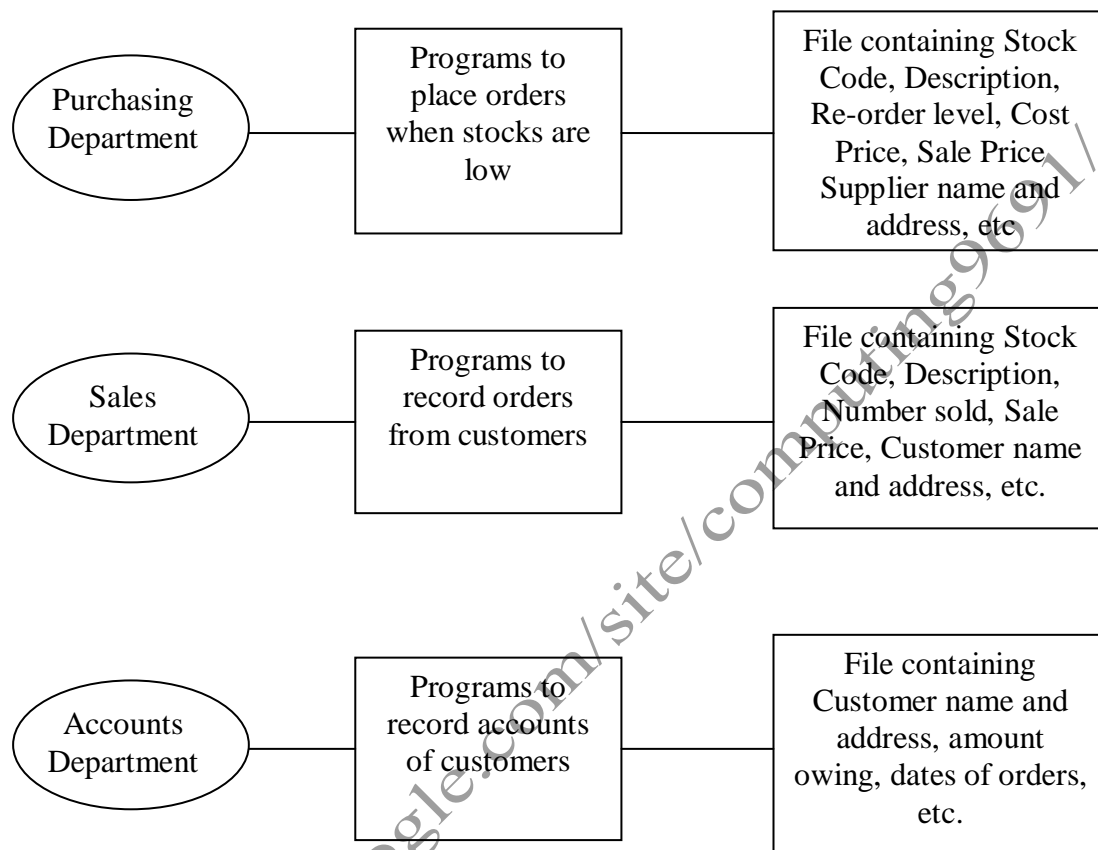


Fig. 3.6.a.1

This method of keeping data uses *flat files*. Flat files have the following limitations.

- Separation and isolation of data

Suppose we wish to know which customers have bought parts produced by a particular supplier. We first need to find the parts supplied by a particular supplier from one file and then use a second file to find which customers have bought those parts. This difficulty can be compounded if data is needed from more than two files.

- Duplication of data

Details of suppliers have to be duplicated if a supplier supplies more than one part. Details of customers are held in two different files.

- Duplication is wasteful as it costs time and money. Data has to be entered more than once, therefore it takes up time and more space.

- Duplication leads to loss of data integrity. What happens if a customer changes his address? The Sales Department may update their files but the Accounts Department may not do this at the same time. Worse still, suppose the Order Department order some parts and there is an increase in price. The Order Department increases the Cost and Sale prices but the Accounts Department do not, there is now a discrepancy.
- Data dependence

Data formats are defined in the application programs. If there is a need to change any of these formats, whole programs have to be changed. Different applications may hold the data in different forms, again causing a problem. Suppose an extra field is needed in a file, again all applications using that file have to be changed, even if they do not use that new item of data.

- Incompatibility of files

Suppose one department writes its applications in COBOL and another in C. Then COBOL files have a completely different structure to C files. C programs cannot read files created by a COBOL program.

- Fixed queries and the proliferation of application programs

File processing was a huge advance on manual processing of queries. This led to end-users wanting more and more information. This means that each time a new query was asked for, a new program had to be written. Often, the data needed to answer the query were in more than one file, some of which were incompatible.

To try to overcome the search problems of sequential files, relational database management systems were introduced.

A database management system (DBMS) is a software package with computer programs that control the creation, maintenance, and the use of a database. It allows organizations to conveniently develop databases for various applications. A database is an integrated collection of data records, files, and other database objects. A DBMS allows different user application programs to concurrently access the same database. DBMSs may use a variety of database models, such as the relational model, to conveniently describe and support applications.

The relational model for database management is a database model that was first formulated and proposed in 1969 by Edgar F. Codd. The purpose of the relational model is to provide a declarative method for specifying data and queries: users directly state in simple sentences that what information the database contains and what information they want from it, and let the database management system software take care of describing data structures for storing the data and retrieval procedures for answering queries.

3.6 (b) Relational Databases and Normalisation

Consider the following delivery note from Easy Fasteners Ltd.

Easy Fasteners Ltd	
Old Park, The Square, Berrington, Midshire BN2 5RG	
To: Bill Jones	No.: 005
London	Date: 14/08/11
England	
Product No.	Description
1	Table
2	Desk
3	Chair

Fig. 3.6. (b)1

In this example, the delivery note has more than one part on it. This is called a repeating group. In the relational database model, each record must be of a fixed length and each field must contain only one item of data. Also, each record must be of a fixed length so a variable number of fields is not allowed. In this example, we cannot say 'let there be three fields for the products as some customers may order more products than this and other fewer products. So, repeating groups are not allowed.

At this stage we should start to use the correct vocabulary for relational databases. Instead of fields we call the columns *attributes* and the rows are called *tuples*. The files are called *relations* (or tables).

We write the details of our delivery note as

DELNOTE(Num, CustName, City, Country, (ProdID, Description))

where DELNOTE is the name of the relation (or table) and Num, CustName, City, Country, ProdID and Description are the attributes. ProdID and Description are put inside parentheses because they form a repeating group. In tabular form the data may be represented by Fig. 3.6 (b)2.

Num	CustName	City	Country	ProdID	Description
005	Bill Jones	London	England	1	Table
				2	Desk
				3	Chair

Fig. 3.6 (b)2

This again shows the repeating group. We say that this is in un-normalised form (UNF). To put it into 1st normal form (1NF) we complete the table and identify a key that will make each tuple unique. This is shown in Fig. Fig. 3.6 (b)3.

Num	CustName	City	Country	ProdID	Description
005	Bill Jones	London	England	1	Table
005	Bill Jones	London	England	2	Desk
005	Bill Jones	London	England	3	Chair

Fig 3.6 (b)3

To make each row unique we need to choose Num together with ProdID as the key. Remember, another delivery note may have the same products on it, so we need to use the combination of Num and ProdID to form the key. We can write this as

DELNOTE(Num, CustName, City, Country, ProdID, Description)

To indicate the key, we simply underline the attributes that make up the key.

Because we have identified a key that uniquely identifies each tuple, we have removed the repeating group.

Definition of 1NF

A relation with repeating groups removed is said to be in First Normal Form (1NF). That is, a relation in which the intersection of each tuple and attribute (row and column) contains one and only one value.

However, the relation DELNOTE still contains redundancy. Do we really need to record the details of the customer for each item on the delivery note? Clearly, the answer is no. Normalisation theory recognises this and allows relations to be converted to Third Normal Form (3NF). This form solves most problems. (Note: Occasionally we need to use Boyce-Codd Normal Form, 4NF and 5NF. This is rare and beyond the scope of this syllabus.)

Let us now see how to move from 1NF to 2NF and on to 3NF.

Definition of 2NF

A relation that is in 1NF and every non-primary key attribute is fully dependent on the primary key is in Second Normal Form (2NF). That is, all the incomplete dependencies have been removed.

In our example, using the data supplied, CustName, City and Country depend only on Num and not on ProdID. Description only depends on ProdID, it does not depend on Num. We say that

Num *determines* CustName, City, Country
ProdID *determines* Description

and write

Num → CustName, City, Country
ProdID → Description

If we do this, we lose the connection that tells us which parts have been delivered to which customer. To maintain this connection we add the dependency

Num, ProdID → 0 (Dummy functional dependency)

We now have three relations.

DELNOTE(Num, CustName, City, Country)
PRODUCT(ProdID, Description)
DEL_PROD(Num, ProdID)

Note the keys (underlined) for each relation. DEL_PROD needs a compound key because a delivery note may contain several parts and similar parts may be on several delivery notes. We now have the relations in 2NF.

Can you see any more data repetitions? The following table of data may help.

Num	CustName	City	Country	ProdID	Description
005	Bill Jones	London	England	1	Table
005	Bill Jones	London	England	2	Desk
005	Bill Jones	London	England	3	Chair
008	Mary Hill	Paris	France	2	Desk
008	Mary Hill	Paris	France	7	Cupboard
014	Anne Smith	New York	USA	5	Cabinet
002	Tom Allen	London	England	7	Cupboard
002	Tom Allen	London	England	1	Table
002	Tom Allen	London	England	2	Desk

Country depends on City not directly on Num. We need to move on to 3NF.

Definition of 3NF

A relation that is in 1NF and 2NF, and in which no non-primary key attribute is transitively dependent on the primary key is in 3NF. That is, all non-key elements are fully dependent on the primary key.

In our example we are saying

$$\text{Num} \rightarrow \text{CustName, City, Country}$$

but it is City that determines Country, that is

$$\text{City} \rightarrow \text{Country}$$

and we can write

$$\begin{aligned} \text{Num} &\rightarrow \text{City} \rightarrow \text{Country} \\ \text{Num} &\rightarrow \text{CustName} \end{aligned}$$

We say that Num transitively functionally determines Country.

Removing this transitive functional determinacy, we have

DELNOTE(Num, CustName, City)
CITY_COUNTRY(City, Country)
PRODUCT(ProdID, Description)
DEL_PROD(Num, ProdID)

Let us now use the data above and see what happens to it as the relations are normalised.

1NF
DELNOTE

Num	CustName	City	Country	ProdID	Description
005	Bill Jones	London	England	1	Table
005	Bill Jones	London	England	2	Desk
005	Bill Jones	London	England	3	Chair
008	Mary Hill	Paris	France	2	Desk
008	Mary Hill	Paris	France	7	Cupboard
014	Anne Smith	New York	USA	5	Cabinet
002	Tom Allen	London	England	7	Cupboard
002	Tom Allen	London	England	1	Table
002	Tom Allen	London	England	2	Desk

Convert to
2NF

DELNOTE

Num	CustName	City	Country
005	Bill Jones	London	England
008	Mary Hill	Paris	France
014	Anne Smith	New York	USA
002	Tom Allen	London	England

PRODUCT

ProdID	Description
1	Table
2	Desk
3	Chair
7	Cupboard
5	Cabinet

DEL_PROD

Num	ProdID
005	1
005	2
005	3
008	2
008	7
014	5
002	7
002	1
002	2

Convert to
3NF

DELNOTE

Num	CustName	City
005	Bill Jones	London
008	Mary Hill	Paris
014	Anne Smith	New York
002	Tom Allen	London

DEL_PROD

Num	ProdID
005	1
005	2
005	3
008	2
008	7
014	5
002	7
002	1
002	2

PRODUCT

ProdID	Description
1	Table
2	Desk
3	Chair
7	Cupboard
5	Cabinet

CITY_COUNTRY

City	Country
London	England
Paris	France
New York	USA

Now we can see that redundancy of data has been removed.

In tabular form we have

UNF

DELNOTE(Num, CustName, City, Country, (ProdID, Description))

1NF

DELNOTE(Num, CustName, City, Country, ProdID, Description)

2NF

DELNOTE(Num, CustName, City, Country)
PRODUCT(ProdID, Description)
DEL_PROD(Num, ProdID)

3NF

DELNOTE(Num, CustName, City)
CITY_COUNTRY(City, Country)
PRODUCT(ProdID, Description)
DEL_PROD(Num, ProdID)

In this Section we have seen the data presented as tables. These tables give us a *view* of the data. The tables do NOT tell us how the data is stored in the computer, whether it be in memory or on backing store. Tables are used simply because this is how users

view the data. We can create new tables from the ones that hold the data in 3NF. Remember, these tables simply define relations.

Users often require different views of data. For example, a user may wish to find out the countries to which they have sent desks. This is a simple view consisting of one column. We can create this table by using the following relations (tables).

PRODUCT	to find ProdID for Desk
DEL_PROD	to find Num for this ProdID
DELNOTE	to find City corresponding to Num
CITY_COUNTRY	to find Country from City

Here is another example of normalisation.

Films are shown at many cinemas, each of which has a manager. A manager may manage more than one cinema. The takings for each film are recorded for each cinema at which the film was shown.

The following table is in UNF and uses the attribute names

FID	Unique number identifying a film
Title	Film title
CID	Unique string identifying a cinema
Cname	Name of cinema
Loc	Location of cinema
MID	Unique 2-digit string identifying a manager
MName	Manager's name
Takings	Takings for a film

FID	Title	CID	Cname	Loc	MID	MName	Takings
15	Jaws	TF	Odeon	Croyden	01	Smith	£350
		GH	Embassy	Osney	01	Smith	£180
		JK	Palace	Lye	02	Jones	£220
23	Tomb Raider	TF	Odeon	Croyden	01	Smith	£430
		GH	Embassy	Osney	01	Smith	£200
		JK	Palace	Lye	02	Jones	£250
		FB	Classic	Sutton	03	Allen	£300
		NM	Roxy	Longden	03	Allen	£290
45	Cats & Dogs	TF	Odeon	Croyden	01	Smith	£390
		LM	Odeon	Sutton	03	Allen	£310
56	Colditz	TF	Odeon	Croyden	01	Smith	£310
		NM	Roxy	Longden	03	Allen	£250

Converting this to 1NF can be achieved by 'filling in the blanks' to give the relation

FID	Title	CID	Cname	Loc	MID	MName	Takings
15	Jaws	TF	Odeon	Croyden	01	Smith	£350
15	Jaws	GH	Embassy	Osney	01	Smith	£180
15	Jaws	JK	Palace	Lye	02	Jones	£220
23	Tomb Raider	TF	Odeon	Croyden	01	Smith	£430
23	Tomb Raider	GH	Embassy	Osney	01	Smith	£200
23	Tomb Raider	JK	Palace	Lye	02	Jones	£250
23	Tomb Raider	FB	Classic	Sutton	03	Allen	£300
23	Tomb Raider	NM	Roxy	Longden	03	Allen	£290
45	Cats & Dogs	TF	Odeon	Croyden	01	Smith	£390
45	Cats & Dogs	LM	Odeon	Sutton	03	Allen	£310
56	Colditz	TF	Odeon	Croyden	01	Smith	£310
56	Colditz	NM	Roxy	Longden	03	Allen	£250

This is the relation

R(FID, Title, CID, Cname, Loc, MID, MName, Takings)

Title is only dependent on FID

Cname, Loc, MID, MName are only dependent on CID

Takings is dependent on both FID and CID

Therefore 2NF is

FILM(FID, Title)

CINEMA(CID, Cname, Loc, MID, MName)

TAKINGS(FID, CID, Takings)

In Cinema, the non-key attribute MName is dependent on MID. This means that it is transitively dependent on the primary key. So we must move this out to get the 3NF relations

FILM(FID, Title)

CINEMA(CID, Cname, Loc, MID)

TAKINGS(FID, CID, Takings)

MANAGER(MID, MName)

3.6 (c) Entity-Relationship (E-R) Diagrams

Entity-Relationship (E-R) diagrams can be used to illustrate the relationships between entities. In the earlier example we had the four relations

DELNOTE(Num, CustName, City)
CITY_COUNTRY(City, Country)
PRODUCT(ProdID, Description)
DEL_PROD(Num, ProdID)

In an E-R diagram DELNOTE, CITY_COUNTRY, PRODUCT and DEL_PROD are called *entities*. Entities have the same names as relations but we do not usually show the attributes in E-R diagrams.

We now consider the *relationships* between the entities.

Each DELNOTE can be for only one CITY_COUNTRY
because a City only occurs once on DELNOTE

Each CITY_COUNTRY may have many DELNOTE
because a City may occur on more than one DELNOTE

Each DELNOTE will have many DEL_PROD.
Num in DELNOTE could occur more than once in DEL_PROD

Each DEL_PROD will be for only one DELNOTE
because each Num in DEL_PROD can only occur once in DELNOTE

Each PRODUCT will be on many DEL_PROD
PRODUCT can occur more than once in DEL_PROD

Each DEL_PROD will have only one PRODUCT
because each ProdID in DEL_PROD can only occur once in PRODUCT

The statements show two types of relationship. There are in fact four altogether. These are

one-to-one	represented by	_____
one-to-many	represented by	_____<
many-to-one	represented by	>_____
many-to-many	represented by	>_____<

Fig. 3.6 (c)1 is the E-R diagram showing the relationships between DELNOTE, CITY_COUNTRY, PRODUCT and DEL_PROD.

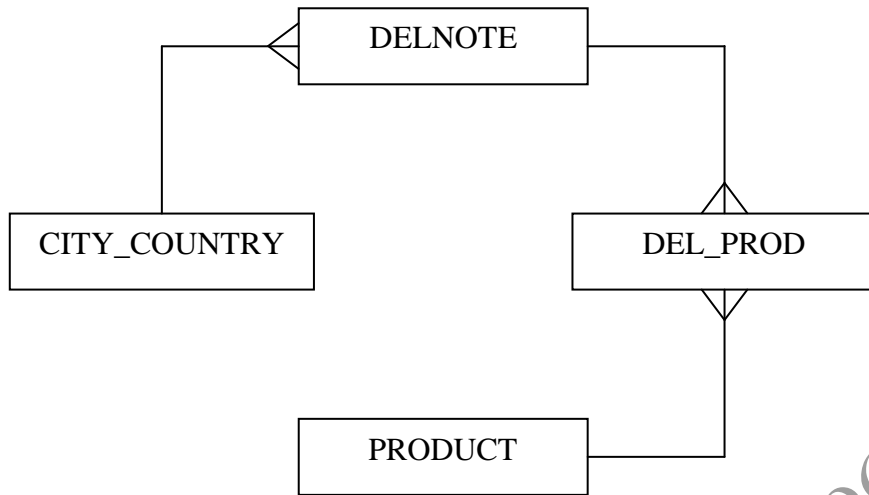


Fig. 3.6 (c)1

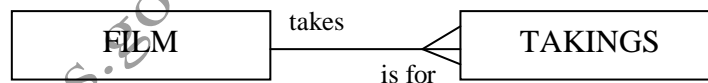
If the relations are in 3NF, the E-R diagram will not contain any many-to-many relationships. If there are any one-to-one relationships, one of the entities can be removed and its attributes added to the entity that is left.

Let us now look at our solution to the cinema problem which contained the relations

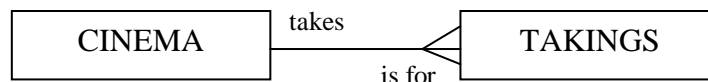
FILM(FID, Title)
 CINEMA(CID, Cname, Loc, MID)
 TAKINGS(FID, CID, Takings)
 MANAGER(MID, MName)

in 3NF.

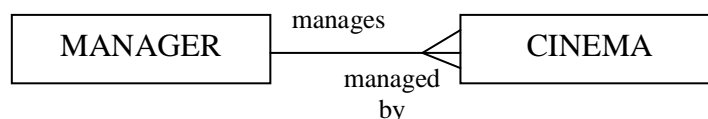
We have the following relationships.



connected by FID



connected by CID



connected by MID

These produce the ERD shown in Fig. 3.6 (c)2.

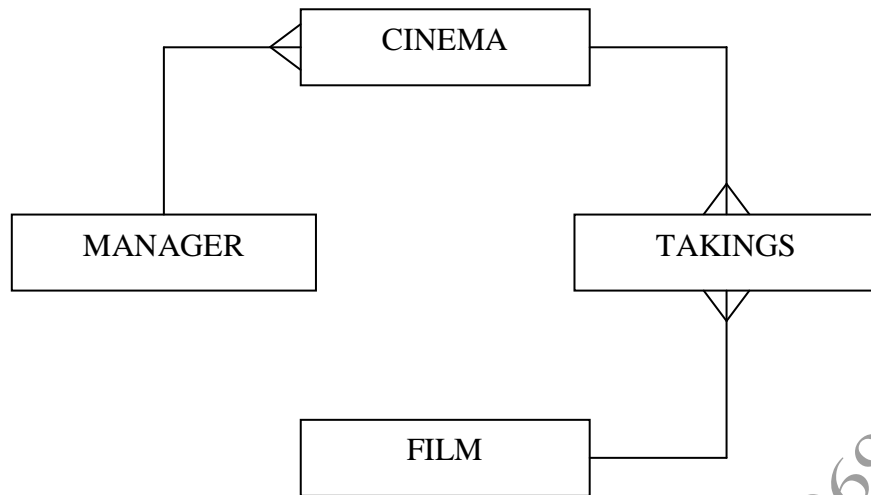


Fig. 3.6 (c)2

In this problem we actually have the relationship

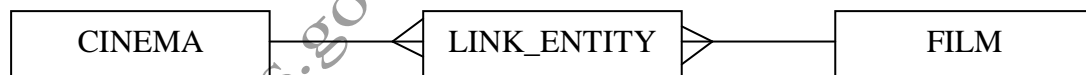
CINEMA shows many FILMs

FILM is shown at many CINEMAs

That is



But this cannot be normalised to 3NF because it is a many-to-many relationship. Many-to-many relationships are removed by using a *link entity* as shown here.



If you now look at Fig. 3.6.c.2, you will see that the link entity is TAKINGS.

Form Design

Section 2.1 (c) discussed the design of screens and forms. All that was said in that section applies to designing forms for data entry, data amendments and for queries. The main thing to remember when designing screen layouts is not to fill the screen too full. You should also make sure that the sequence of entering data is logical and that, if there is more than one screen, it is easy to move between them.

Let us consider a form that will allow us to create a new entry in DELNOTE which has the attributes Num, CustName, City. Num is the key and, therefore, it should be created by the database system. Fig. 3.6 (c)3 shows a suitable form.

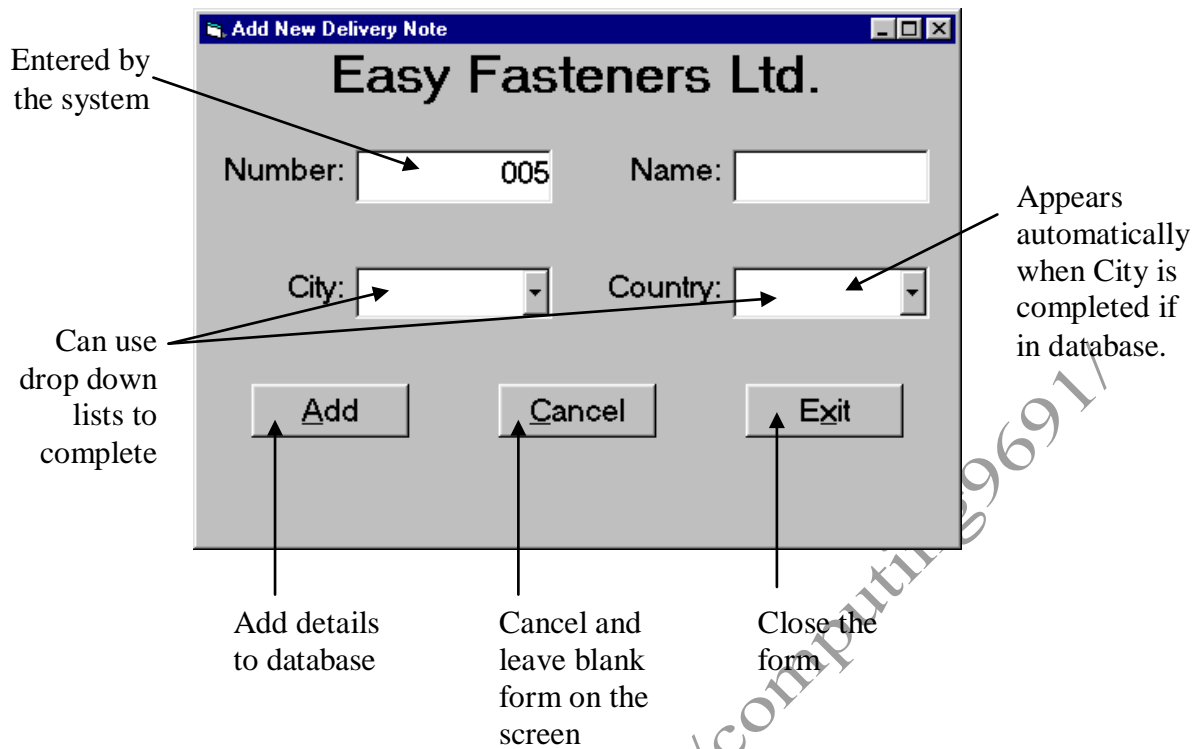


Fig. 3.6 (c)3

With this form, if a new City is input the user can input the Country and the City_Country table will be updated. If the City exists in the database, then Country will appear automatically.

Now let us design a form to allow a user to input a customer's order. In this case we shall need to identify the customer before entering the order details. This is best done by entering the customer's ID. However, this is not always known. An alternative, in this case, is to enter the customer's name. The data entry form should allow us to enter either of these pieces of data and the rest of the details should appear automatically as a verification that the correct customer has been chosen. Fig. 3.6(c)4 shows a form that is in two parts. The upper part is used to identify the customer and the lower part allows us to enter each item of data that is on the customer's order.

Enter either the customer's number OR the customer's name. The other three boxes will then be completed automatically by the system.

The screenshot shows a window titled "Order Entry" for "Easy Fasteners Ltd.". The form is divided into two main sections: "Customer details" and "Order details".

Customer details: Includes fields for "Number:", "Name:", "City:", and "Country:". Annotations indicate that either the "Number" or "Name" field is entered by the user, and the other three fields are automatically completed by the system.

Order details: Includes fields for "Part Number:" and "Description:". Annotations indicate that the "Part Number" is entered by the user, and the "Description" is entered by the system.

Buttons: "Add", "Clear", and "Exit".

- Add:** Add item to order and clear the order detail part of the form
- Clear:** Clear all boxes ready for a new customer
- Exit:** Close the Order Entry form

Fig. 3.6 (c)4

Notice how certain boxes are automatically completed. Also, because the form requires a customer ID (Number), orders can only be taken for customers whose details are on the database. This ensures the entry of customer details before an order can be entered. In order to be consistent, the positions of the boxes for customer details is the same on the Order Entry form as on the Add New Delivery Note form. It is usual for both these forms to be password protected. This ensures that only authorised personnel can enter data into the database.

This is a very simple example. Suppose the customer's ID is not known. We have seen one way of overcoming this which satisfies the needs of the problem given. Some systems allow the post code to be entered in order to identify the address. In this case, the street, town and county details are displayed and the user is asked for the house number. Other systems allow the user to enter a dummy ID such as 0000 and then a list of customers appears from which the user can choose a name. Alternatively, part of the name can be entered and then a short list of possible names is displayed. Again the user can choose from this list.

Deletion and modification screens are similar, but must be password protected as before so that only authorised personnel can change the database.

A query screen should not allow the user to change the database. Also, users should only be allowed to see what they are entitled to see. To see how this may work, let us consider a query requesting the details of all the cinemas in our second example. The view presented to the users will give details of cinema names, locations, manager names and film names as shown in Fig. 3.6 (c)5.

Cinema	Location	Manager	Film
Odeon	Croyden	Smith	Jaws
Embassy	Osney	Smith	Jaws
Palace	Lye	Jones	Jaws
Odeon	Croyden	Smith	Tomb Raider
Embassy	Osney	Smith	Tomb Raider
Palace	Lye	Jones	Tomb Raider
Classic	Sutton	Allen	Tomb Raider
Roxy	Longden	Allen	Tomb Raider
Odeon	Croyden	Smith	Cats & Dogs
Odeon	Sutton	Allen	Cats & Dogs
Odeon	Croyden	Smith	Colditz
Roxy	Longden	Allen	Colditz

Fig. 3.6 (c)5

However, another user may be given the view shown in Fig. 3.6 (c)6.

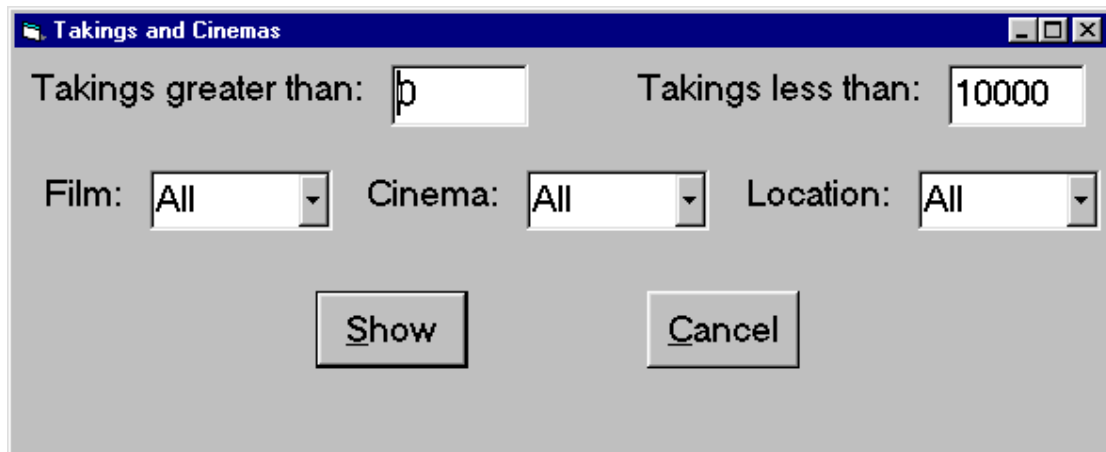
Cinema	Location	Manager	Film	Takings
Odeon	Croyden	Smith	Jaws	£350
Embassy	Osney	Smith	Jaws	£180
Palace	Lye	Jones	Jaws	£220
Odeon	Croyden	Smith	Tomb Raider	£430
Embassy	Osney	Smith	Tomb Raider	£200
Palace	Lye	Jones	Tomb Raider	£250
Classic	Sutton	Allen	Tomb Raider	£300
Roxy	Longden	Allen	Tomb Raider	£290
Odeon	Croyden	Smith	Cats & Dogs	£390
Odeon	Sutton	Allen	Cats & Dogs	£310
Odeon	Croyden	Smith	Colditz	£310
Roxy	Longden	Allen	Colditz	£250

Fig. 3.6 (c)6

Another user may be given all the details, including the cinema and manager IDs. Notice that the columns do not have to have the same names as the attributes in the database. This means that these names can be made more user friendly.

In order to create the query a user will normally be presented with a data entry form. This form may contain default values, as shown in Fig. 3.6 (c)7, which allows a user

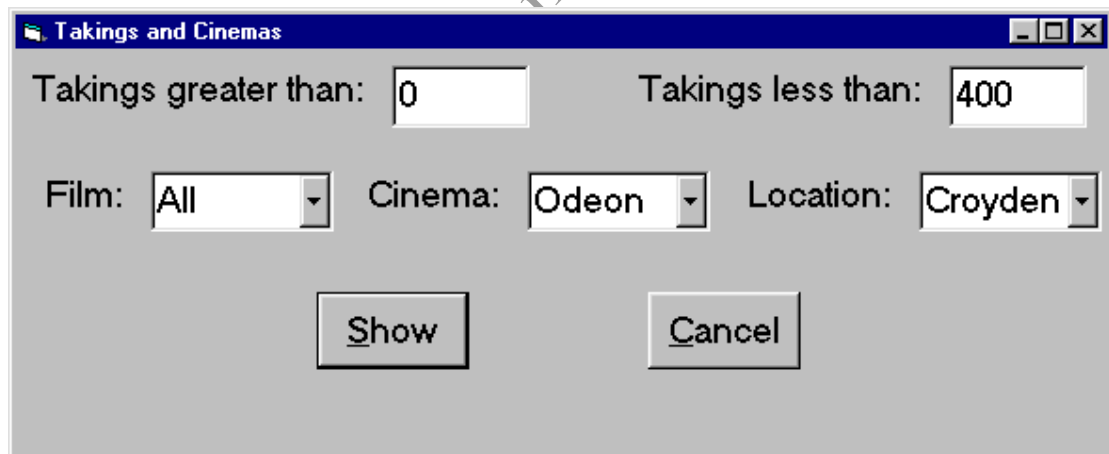
to list cinemas that have takings for films between set limits. This form allows users to choose all the films, all the cinemas and all locations or to be more selective by choosing from drop down lists. When the user clicks the OK button a table, such as those given above, will appear.



The screenshot shows a dialog box titled "Takings and Cinemas". It contains two text input fields: "Takings greater than:" with the value "0" and "Takings less than:" with the value "10000". Below these are three dropdown menus: "Film:" set to "All", "Cinema:" set to "All", and "Location:" set to "All". At the bottom are two buttons: "Show" and "Cancel".

Fig. 3.6 (c)7

In this Figure, the boxes are initially completed with default values. In this case, if the OK button is clicked, all cinemas and films would be listed. However, suppose we want to know which films at the Odeon, Croyden took less than £400. The user could modify the boxes, using drop down lists, as shown in Fig. 3.6 (c)8.



The screenshot shows the same "Takings and Cinemas" dialog box, but with modified values. The "Takings greater than:" field now contains "0" and the "Takings less than:" field contains "400". The "Film:" dropdown is still "All", but the "Cinema:" dropdown is now "Odeon" and the "Location:" dropdown is now "Croyden". The "Show" and "Cancel" buttons remain at the bottom.

Fig. 3.6 (c)8

When the OK button is clicked, a report like that shown in Fig. 3.6(c)9 would appear together with a button allowing the user to print the results or return to the query form.

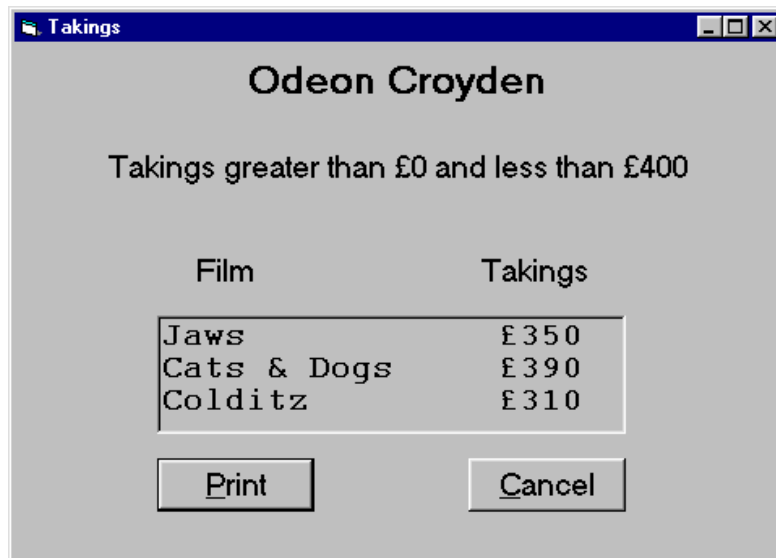


Fig. 3.6 (c)9

<http://sites.google.com/site/computing9691/>

3.6 (d) Advantages of Using a Relational Database (RDB)

Advantage	Notes
Control of data redundancy	Flat files have a great deal of data redundancy that is removed using a RDB.
Consistency of data	There is only one copy of the data so there is less chance of data inconsistencies occurring.
Data sharing	The data belongs to the whole organisation, not to individual departments.
More information	Data sharing by departments means that departments can use other department's data to find information.
Improved data integrity	Data is consistent and valid.
Improved security	The database administrator (DBA) can define data security – who has access to what. This is enforced by the Database Management System (DBMS).
Enforcement of standards	The DBA can set and enforce standards. These may be departmental, organisational, national or international.
Economy of scale	Centralisation means that it is possible to economise on size. One very large computer can be used with dumb terminals or a network of computers can be used.
Improved data accessibility	This is because data is shared.
Increased productivity	The DBMS provides file handling processes instead of each application having to have its own procedures.
Improved maintenance	Changes to the database do not cause applications to be re-written.
Improved back-up and recovery	DBMSs automatically handle back-up and recovery. There is no need for somebody to remember to back-up the database each day, week or month.

Disadvantages are not in the Specification for this Module.

3.6 (e) The Purpose of Keys

We have used keys in all our earlier examples to uniquely identify tuples (rows) in a relation (table). A key may consist of a single attribute or many attributes, in which case it is called a *compound key*.

The key used to uniquely identify a tuple is called the *primary key*.

In some cases more than one attribute, or group of attributes, could act as the primary key. Suppose we have the relation

EMP(EmpID, NINumber, Name, Address)

Clearly, EmpID could act as the primary key. However, NINumber could also act as the primary key as it is unique for each employee. In this case we say that EmpID and NINumber are candidate keys. If we choose EmpID as the primary key, then NINumber is called a *secondary key*.

Now look at these two relations that we saw in Section 4.6.4.

CINEMA(CID, Cname, Loc, MID)
MANAGER(MID, MName)

We see that MID occurs in CINEMA and is the primary key in MANAGER. In CINEMA we say that MID is the *foreign key*.

An attribute is a *foreign key* in a relation if it is the primary key in another relation. Foreign keys are used to link relations.

3.6 (f) Access Rights

Sometimes it must not be possible for a user to access the data in a database. For example, in a banking system, accounts must be updated with the day's transactions. While this is taking place users must not be able to access the database. Thus, at certain times of the day, users will not be able to use a cash point. Another occasion is if two people have a joint account and one of them is withdrawing cash from a cash point. In this case the one user will be able to change the contents of the database while the other will only be allowed to query the database.

Similarly, while a database system is checking stock for re-ordering purposes, the POS terminals will not be able to use the database as each sale would change the stock levels. Incidentally, there are ways in which the POS terminals could still operate. One is to only use the database for querying prices and to create a transaction file of sales which can be used later to update the database.

It is often important that users have restricted views of the database. Consider a large hospital that has a large network of computers. There are terminals in reception, on the wards and in consulting rooms. All the terminals have access to the patient database which contains details of the patients' names and addresses, drugs to be administered and details of patients' illnesses.

It is important that when a patient registers at reception the receptionist can check the patient's name and address. However, the receptionist should not have access to the drugs to be administered nor to the patient's medical history. This can be done by means of passwords. That is, the receptionists' passwords will only allow access to the information to which receptionists are entitled. When a receptionist logs onto the network the DBMS will check the password and will ensure that the receptionist can only access the appropriate data.

Now the terminals on the wards will be used by nurses who will need to see what drugs are to be administered. Therefore nurses should have access to the same data as the receptionists and to the information about the drugs to be given. However, they may not have access to the patients' medical histories. This can be achieved by giving nurses a different password to the receptionists. In this case the DBMS will recognise the different password and give a higher level of access to the nurses than to the receptionists.

Finally, the consultants will want to access all the data. This can be done by giving them another password.

All three categories of user of the database, receptionist, nurse and consultant, must only be allowed to see the data that is needed by them to do their job.

So far we have only mentioned the use of passwords to give levels of security. However, suppose two consultants are discussing a case as they walk through reception. Now suppose they want to see a patient's record. Both consultants have the right to see all the data that is in the database but the terminal is in a public place and patients and receptionists can see the screen. This means that, even if the

consultants enter the correct password, the system should not allow them to access all the data.

This can be achieved by the DBMS noting the address of the terminal and, because the terminal is not in the right place, refusing to supply the data requested. This is a hardware method of preventing access. All terminals have a unique address on their network cards. This means that the DBMS can decide which data can be supplied to a terminal.

<http://sites.google.com/site/computing9691/>

3.6 (g) Database Management System (DBMS)

Let us first look at the architecture of a DBMS as shown in Fig. 3.6 (g)1.

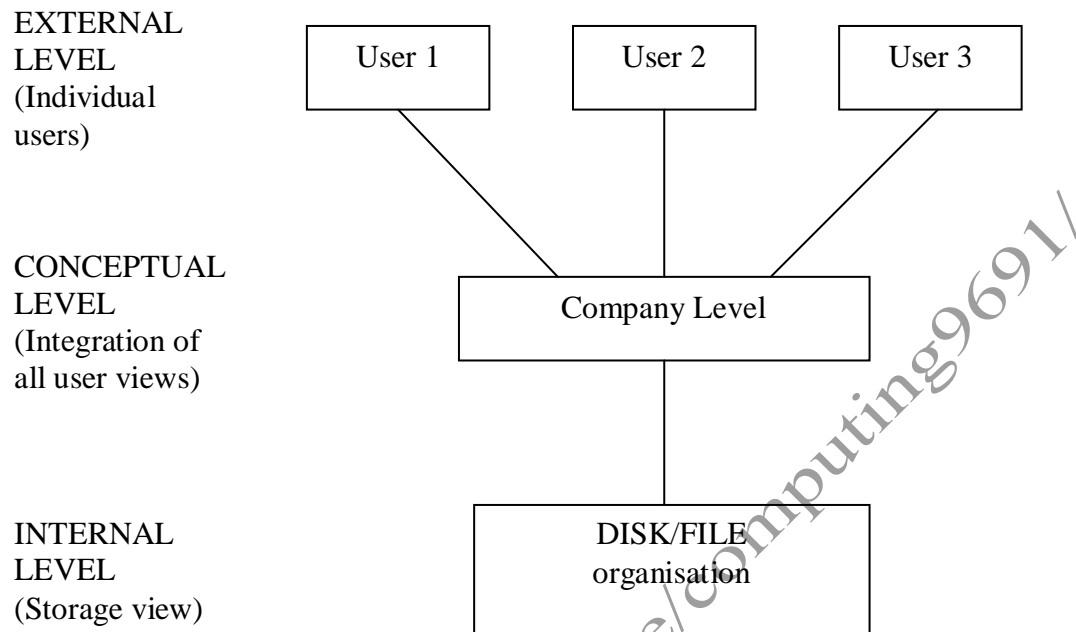


Fig. 3.6 (g)1

At the external level there are many different views of the data. Each view consists of an abstract representation of part of the total database. Application programs will use a data manipulation language (DML) to create these views.

At the conceptual level there is one view of the data. This view is an abstract representation of the whole database.

The internal view of the data occurs at the internal level. This view represents the total database as actually stored. It is at this level that the data is organised into random access, indexed and fully indexed files. This is hidden from the user by the DBMS.

The DBMS is a piece of software that provides the following facilities.

The DBMS contains a data definition language (DDL). The DDL is used, by the database designer, to define the tables of the database. It allows the designer to specify the data types and structures and any constraints on the data. The Structured Query Language (SQL) contains facilities to do this. A DBMS such as Microsoft Access allows the user to avoid direct use of a DDL by presenting the user with a design view in which the tables are defined.

The DDL cannot be used to manipulate the data. When a set of instructions in a DDL are compiled, tables are created that hold data about the data in the database. That is, it holds information about the data types of attributes, the attributes in a relation and

any validation checks that may be required. Data about data is called meta-data. These tables are stored in the data dictionary that can be accessed by the DBMS to validate data when input. The DBMS normally accesses the data dictionary when trying to retrieve data so that it knows what to retrieve. The data dictionary contains tables that are in the same format as a relational database. This means that the data can be queried and manipulated in the same way as any other data in a database.

The other language used is the data manipulation language (DML). This language allows the user to insert, update, delete, modify and retrieve data. SQL includes this language. Again, Access allows a user to avoid directly using the DML by providing query by example (QBE) as was mentioned in Section 3.5 (j).

<http://sites.google.com/site/computing9691/>

Appendix: Designing Databases

Although not stated as part of the syllabus, students may find the following to be of value, particularly when normalising a database.

Consider the following problem.

A company employs engineers who service many products. A customer may own many products but a customer's products are all serviced by the same engineer. When engineers service products they complete a repair form, one form for each product repaired. The form contains details of the customer and the product that has been repaired as well as the Engineer's ID. Each form has a unique reference number. When a repair is complete, the customer is given a copy of the repair form.

The task is to create a database for this problem. In order to do this, you must first analyse the problem to see what entities are involved. The easiest way to do this is to read the scenario again and to highlight the nouns involved. These are usually the entities involved. This is done here.

A company employs engineers who service many products. A customer may own many products but a customer's products are all serviced by the same engineer. When engineers service products they complete a repair form, one form for each product repaired. The form contains details of the customer and the product that has been repaired as well as the engineer's ID. Each form has a unique reference number. When an engineer has repaired a product, the customer is given a copy of the repair form.

This suggests the following entities.

engineer
product
customer
repair form

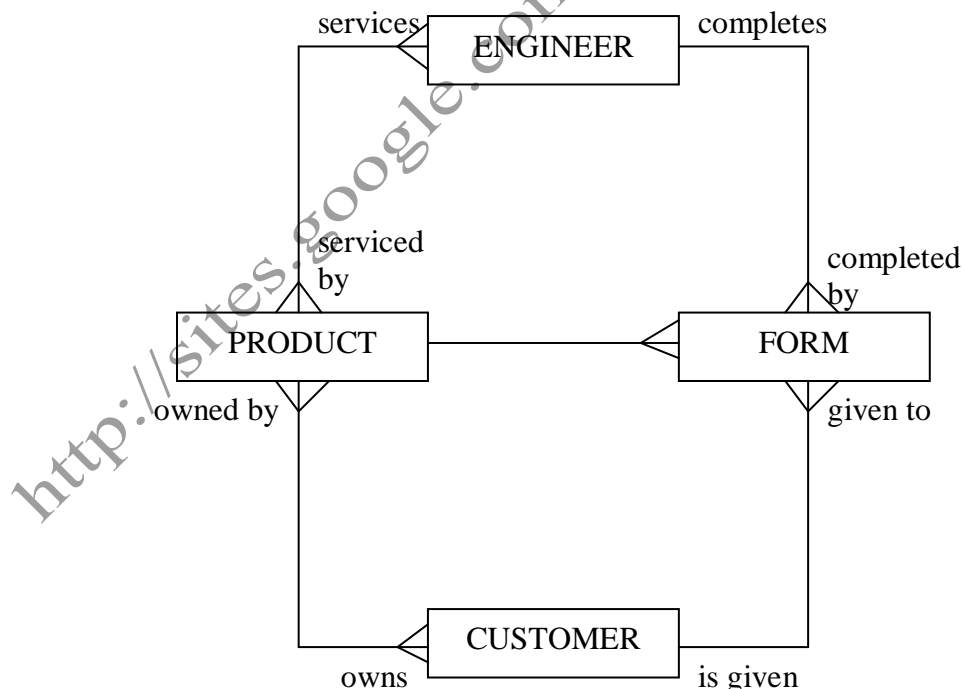
Now we look for the relationships between the entities. These can usually be established by highlighting the verbs as done here.

A company employs engineers who service many products. A customer may own many products but a customer's products are all serviced by the same engineer. When engineers service products they complete a repair form, one form for each product repaired. The form contains details of the customer and the product that has been repaired as well as the Engineer's ID. Each form has a unique reference number. When a repair is complete, the customer is given a copy of the repair form.

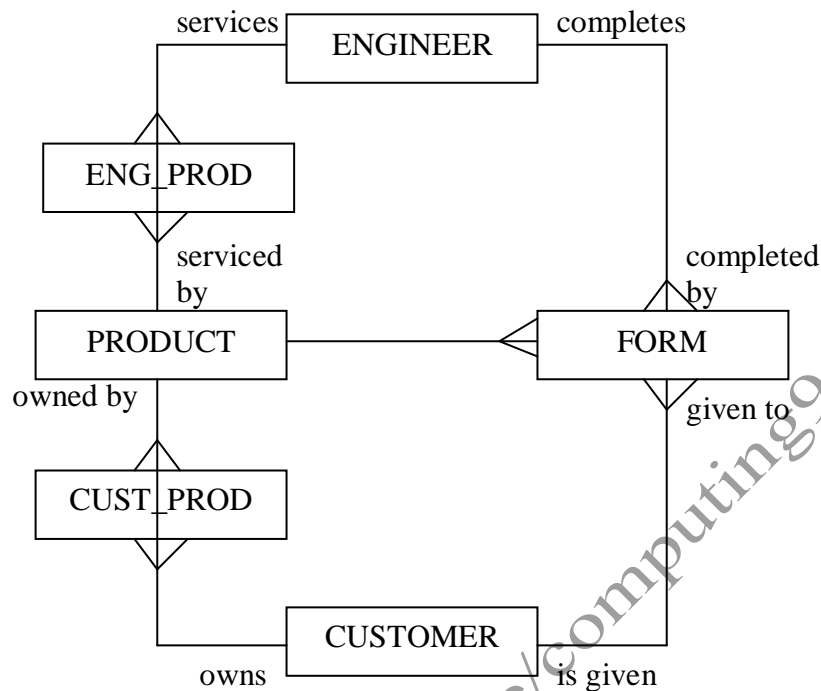
This suggests the following relationships.

Relationship	Type	Notes
engineer services product	many-to-many	An engineer services many products and a product can be serviced by many engineers. For example, many engineers service washing machines.
customer owns product	many-to-many	A customer may own many products and a product can be owned by many customers. For example, many customers own washing machines.
engineer completes form	one-to-many	An engineer completes many forms but a form is completed by only one engineer.
customer is given form	one-to-many	A customer may receive many forms but a form is given to only one customer.
product is on form	one-to-many	Only one product can be on a form but a product may be on many different forms.

Which leads to the entity relationship diagram (ERD).



There are two many-to-many relationships that must be changed to one-to-many relationships by using link entities. This is shown below.

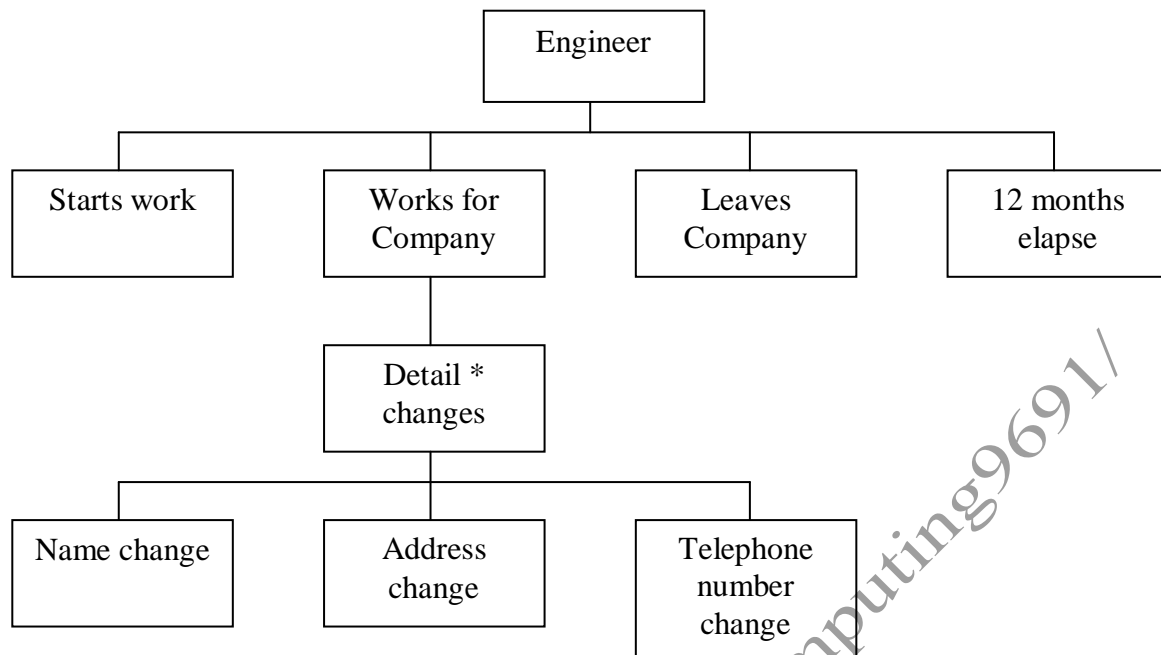


This suggests the following relations (not all the attributes are given).

ENGINEER(EngineerID, Name, ...)
ENG_PROD(EngineerID, ProductID)
PRODUCT(ProductID, Description, ...)
CUST_PROD(CustomerID, ProductID)
CUSTOMER(CustomerID, Name, ...)
FORM(FormID, CustomerID, EngineerID, ...)

These are in 3NF, but you should always check that they are.

Another useful diagram shows the life history of an entity. This simply shows what happens to an entity during its life. An entity life history diagram is similar to a JSP diagram (see Section 3.5 (c)). The next diagram shows the life history of an engineer in our previous problem.



This tells us

1. A new record for an engineer is created when an engineer joins the Company.
2. While the engineer is working for the Company, he/she may change their name, address or telephone number as many times as is necessary (hence the use of the *).
3. When the engineer leaves the Company, the main life-cycle ends and the engineer's record is updated to indicate they no longer work for the Company.
4. 12 months after the engineer has left the Company his/her record is archived and removed from the database.

Similar diagrams can be created for the other entities.

3.6 Example Questions

1. Explain what is meant by a table being in second normal form.
2. Every student in a school belongs to a form. Every form has a form tutor and all the form tutors are members of the teaching body.
Draw an entity relationship diagram to show the relationships between the four entities STUDENT, FORM, TUTOR, TEACHERS (6)
3. Explain what is meant by a foreign key. (3)

<http://sites.google.com/site/computing9691/>