

1. Introduction to Databases

1.1. Introduction

We live in an information age. By this we mean that, first, we accept the universal fact that information is required in practically all aspects of human enterprise. The term 'enterprise' is used broadly here to mean any organisation of activities to achieve a stated purpose, including socio-economic activities. Second, we recognise further the importance of efficiently providing timely relevant information to an enterprise and of the importance of the proper use of technology to achieve that. Finally, we recognise that the unparalleled development in the technology to handle information has and will continue to change the way we work and live, ie. not only does the technology support existing enterprises but it changes them and makes possible new enterprises that would not have otherwise been viable.

The impact is perhaps most visible in business enterprises where there are strong elements of competition. This is especially so as businesses become more globalised. The ability to coordinate activities, for example, across national borders and time zones clearly depends on the timeliness and quality of information made available. More important perhaps, strategic decisions made by top management in response to perceived opportunities or threats will decide the future viability of an enterprise, one way or the other. In other words, in order to manage a business (or any) enterprise, future development must be properly estimated. Information is a *vital* ingredient in this regard.

Information must therefore be collected and analysed in order to make decisions. It is here that the proper use of technology will prove to be crucial to an enterprise. Up-to-date management techniques should include computers, since they are very powerful tools for processing large quantities of information. Collecting and analysing information using computers is facilitated by current Database Technology, a relatively mature technology which is the subject of this book.

1.2. Information Model

Information stored in computer memory is called *data*. In current computer systems, such data can (persistently) reside on a number of memory devices, most common of which are floppy disks, CD-ROMs, and hard disks.

Data that we store and manipulate using computers are meaningful only to the extent that they are associated with some real world object in a given context. Take, for example, the number '23'. This is a piece of data, but by itself a meaningless quantity. If it was associated with, say, a person and interpreted to denote that person's age (in years), then it begins to be more meaningful. Or, if it was associated with, say, an

organisation that sells electronic goods and interpreted to mean the number of television sets sold in a given month, then again it becomes more meaningful. Notice that in both preceding examples, other pieces of data had to be brought into context - a person, a person's age, a shop, television sets, a given month, etc.

If the data is a collection of related facts about some enterprise (eg. a business, an organisation, an activity, etc), then it is called a database. The data stored need not include every conceivable piece of fact about that enterprise. Usually, only facts relevant to some area of an enterprise are captured and organised, typically to provide information to support decision making at various levels (operational, management, etc). Such a constrained area of focus is also often referred to as the *problem domain* or *domain of interest*, and is typical of databases. In this sense, a database is an *information model* of some (real-world) problem domain.

1.2.1. Entities

Information models operate on so-called entities and entity relationships. In this section we will clarify what an entity is. Entity relationships are described in 1.2.2.

An *entity* is a particular object in the problem domain. For example, we can extend the electronics organisation above to identify three distinct entities: products, customers and sales representatives (see Figure 1-1). They are distinct from one another in the sense that each has characteristic *properties* or *attributes* that distinguish it from the others. Thus a product has properties such as type, function, dimensions, weight, brand name, cost and price; a customer has properties such as name, city of residence, age, credit rating, etc.; and a sales representative has properties such as name, address, sales region, basic salary, etc. Each entity is thus *modelled* in the database as a collection of data items corresponding to its relevant attributes. (Note that we distinguish between entities even if in the real world they are from the same class of objects. For example, a customer and a sales representative are both people, but a customer is a person who purchases goods while a sales representative is one who sells goods. The different 'roles' played distinguishes each from the other)

Note also the point made earlier that an information model captures only what is relevant in the given problem domain. Certainly, there are other entities in the organisation - regional offices, warehouses, store keepers, distributors, etc - but these may be irrelevant in the context of, say, analysing sales transactions and are thus omitted from the information model. Even at the level of entity attributes, not all conceivable properties need be captured as data items. A customer's height, weight, hair colour, hobby, formal qualification, favourite foods, etc, are probably irrelevant and can thus omitted from the model.

Strictly speaking, the objects we referred to above as entities are perhaps more accurately called entity classes because they each denote a set of objects (individual entities), each of which exhibits the properties/attributes described for the class. Thus the entity class 'customer' is made up of individual entities, each of which has attributes 'name', 'city of residence', 'age', etc. Every individual entity will then have these attributes but one individual will differ from another in the *values* (data items)

associated with attributes. For example, one customer entity might have the value 'Smith' as its 'name' attribute, while another might have the value 'Jones'.



Figure 1-1 Problem domain entities and their attributes

Notice now that in our information model an attribute is really a pair: an attribute description or name (such as 'age') and an attribute value (such as '56'), or simply, an 'attribute-value' pair. An individual entity is completely modelled only when all its attribute descriptions have been associated with appropriate attribute values. The collection of attribute-value pairs that model a particular individual entity is termed a *data object*. Figure 1-2 illustrates three data objects in the database, each being a complete model of an individual from its corresponding entity class.

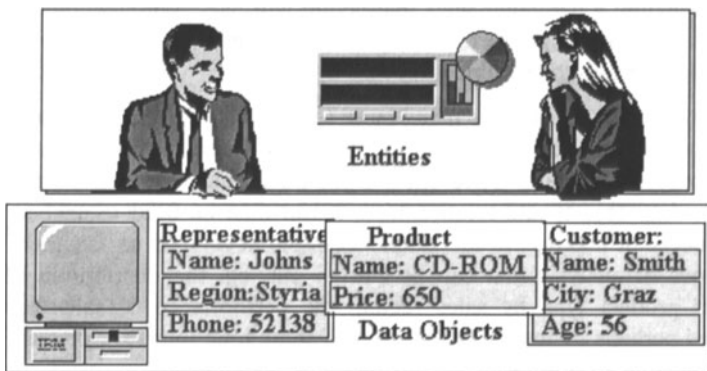


Figure 1-2. Data objects model particular entities in the real world

1.2.2. Entity Relationships

An entity by itself is often not as interesting or as informative as when it relates in some way to some other entity or entities. A particular product, say a CD-ROM drive, itself only tells us about its intrinsic properties as recorded in its associated data

object. A database, however, models more than individual entities in the problem domain. It also models *relationships* between entities.

In the real world, entities do not stand alone. A CD-ROM drive is supplied by a supplier, is stored in a warehouse, is bought by a customer, is sold by a sales representative, is serviced by a technician, etc. Each of these is an example of a relationship, a logical association between two or more entities.

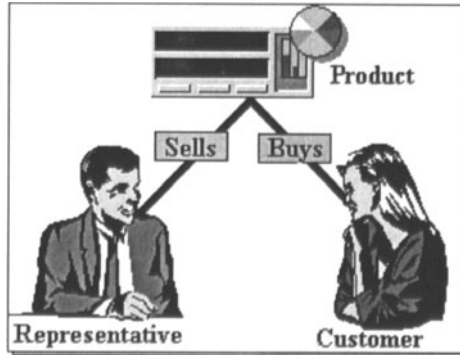


Figure 1-3. Relationships between entities

Figure 1-3 illustrates such relationships by using links labelled with the type of association between entities. In the figure, a representative sells a product and a customer buys a product. Taken together, these links and the entities they link model a sales transaction: a particular sales transaction will have a product data object related (through the 'sells' relation) to a representative data object and (through the 'buys' relation) to a customer data object.

Like entity attributes, there are many more relationships than are typically captured in an information model. The choices are of course based on judgements of relevance given a problem domain. Once choices are made and the database populated with particular data objects and relationships, it can then be used to analyse the data to support decision making. In the simple example developed so far, there are already many types of analysis that can be carried out, eg. the distribution of sales of a particular product type by sales region, the performance of sales representatives (measured perhaps by total value of sales in some time interval), product preferences by customer age groups, etc.

1.3. Database Management

The real world is dynamic. As an organisation goes about its business, entities are created, modified or destroyed. Similarly with entity relationships. This is easy to see even for the simple problem domain above, eg. when a sales is made, the product sold is then logically linked to the customer that bought it and to the representative that sold it. Many sales transactions could take place each day and thus many new logical links created between many individual entities. New entities can also be introduced,

eg. a new customer arrives on the scene, a new product is offered, or a new salesperson is hired. Likewise, some entities may no longer be of concern to the organisation, eg. a product is discontinued, a salesperson quits or is fired, etc (these entities may still exist in the real world but have become irrelevant for the problem domain). Clearly, an information model must also change to reflect the changes in the problem domain that it models.

If the problem domain is small, involving only a few entities and relationship, and the dynamic changes are relatively few or infrequent, manual methods may be sufficient to maintain an accurate record of the state of the business. But if hundreds or thousands of entities are involved and the business is very dynamic, then maintaining accurate records of its state becomes more of a problem. This is when computers with their immense power to handle large amounts of information become crucial to an organisation. Frequently, it is not just a question of efficiency, but of survival, especially in intensely competitive business sectors.

The need to use computers to efficiently and effectively store databases and to keep them current has developed over the years special software packages called *Database Management Systems* (DBMS). A DBMS enables users to create, modify, access and protect their databases (Figure 1-4).

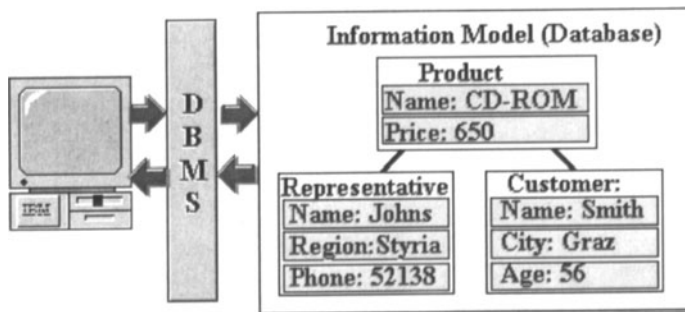


Figure 1-4 A DBMS is a tool to create and use databases

In other words, a DBMS is a *tool* to be applied by users to build an accurate and useful information model of their enterprise.

Conceptually, database management is based on the idea of separating a database *structure* from its *contents*. Quite simply, a database structure is a collection of static descriptions of entity classes and relationships between them. At this point, it is perhaps simplest to think of an entity class description as a collection of attribute labels. Entity contents can then be thought of as the values that get associated with attribute labels, creating data objects. In other words, the distinction between structure and content is little more than the distinction made earlier between attribute label and attribute value.

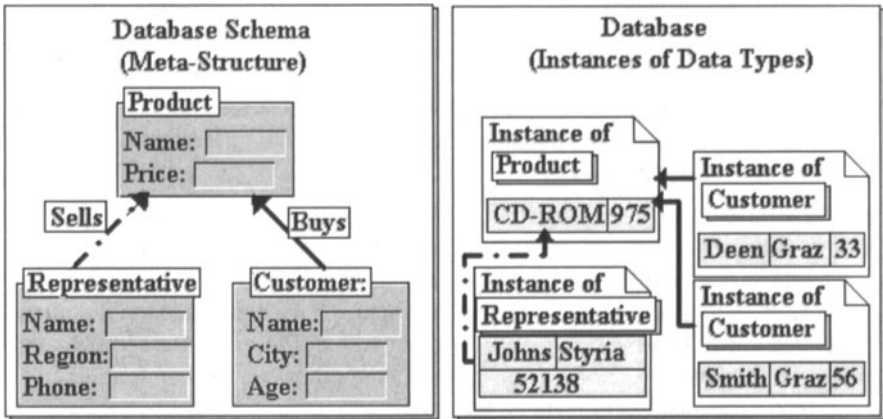


Figure 1-5 Separation of structure from content

Relationship descriptions likewise are simply labelled links between entity descriptions. They specify possible links between data objects, ie. two data objects can be linked only if the database structure describes a link between their respective entity classes. Figure 1-5 illustrates this separation of structure from content.

The database structure is also called a *schema* (or *meta-structure* - because it describes the structure of data objects). It predefines all possible states of a database, in the sense that no state of the database can contain a data object that is not the result of instantiating an entity schema, and likewise no state can contain a link between two data objects unless such a link was defined in the schema.

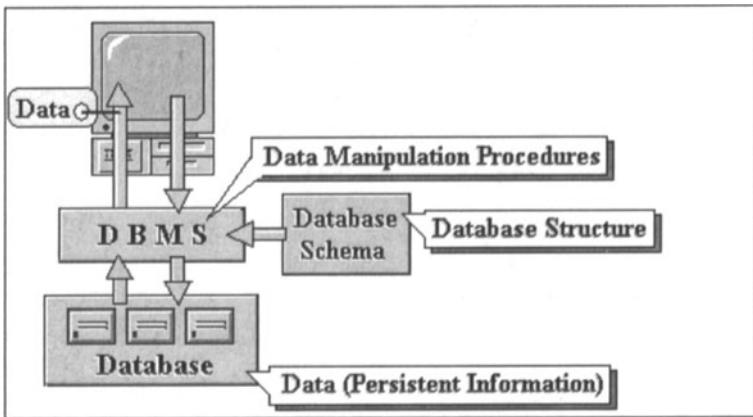


Figure 1-6 Architecture of database systems

Moreover, data manipulation procedures can be separated from the data as well! Thus the architecture of database systems may be portrayed as in Figure 1-6.

1.4. Database Languages

We see from the foregoing that to build a database, a user must

1. Define the Database Schema
2. Apply a collection of operators supported by the DBMS to create, store, retrieve and modify data of interest

A typical DBMS would provide tools to facilitate the above tasks. At the heart of these tools, a DBMS typically maintains two closely related *languages*:

1. A Data Description Language (DDL), which is used to define database schemas, and
2. A Data Manipulation Language (DML), which allows the user to manipulate data objects and relationships between them in the context of given database schemas

These languages may vary from one DBMS to another, in their underlying data model, complexity, functionality, and ease of use (user interface).

So far, we have talked about ‘users’ as if they were all equal in interacting with a DBMS. In actual fact, though, there may be several types of users distinguished by their role (a division of labour, often necessary because of highly technical aspects of DBMSs). For example, an organisation that uses a DBMS will normally have a Database Administrator (DBA) whose job is to create and maintain a consistent set of database schemas to satisfy the needs of different parts of the organisation. The DBA is the principal user of the DDL. Then there are application developers who develop specific functions around the database (eg. product inventory, customer information, point-of-sale transaction handling, etc). They are the principal users of the DML. And finally, there are the end-users who use the applications developed to support their work in the organisation. They normally don’t see (and don’t care to know about!) the DDL or the DML.

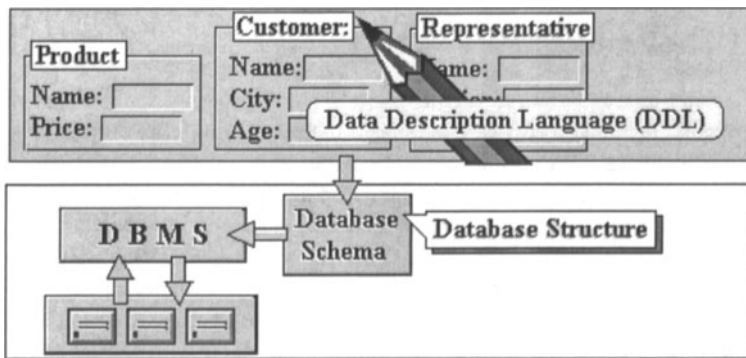


Figure 1-7 (notional) DDL definition

The DDL is a collection of statements for the description of data types. The DBA must define the target database structure in terms of these data types.

For instance, the data object, attribute and link mentioned above are data types, and hence may be perceived as a simple DDL. Thus the data structures in Figure 1-5 are notionally DDL descriptions of a database schema, as illustrated in Figure 1-7 ('notional' because the actual language will have specific syntactical constructs that may differ from one DBMS to another).

A DML is a collection of operators that can be applied to valid instances (ie. data objects) of the data types in the schema. As illustrated in Figure 1-8, the DML is used to manipulate instances, including the creation, modification and retrieval of instances. (Like the DDL above, the illustration here is notional; more concrete forms of these languages will be covered in later sections of this book).

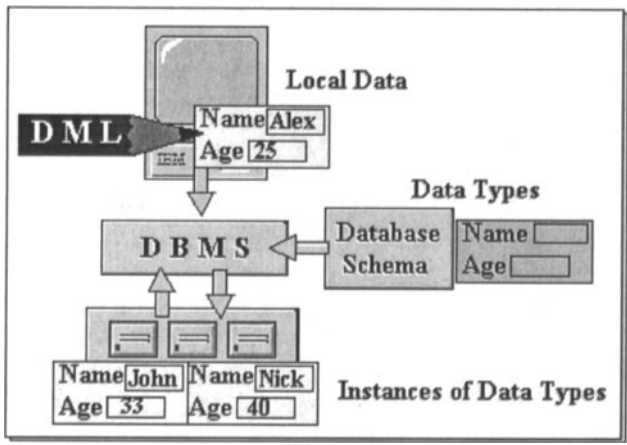


Figure 1-8 DML manipulations of instances

1.5. Data Protection

Databases can be a major investment. There are costs, obviously, associated with the hardware and specialised software such as a DBMS. Less obvious, perhaps, are costs of creating databases. Large and complex databases can require many man-years of analysis and design effort involving specialist skills that may not be present in the organisation. Thus expensive consultants and other technical specialists may have to be brought in. Furthermore, in the long-term, an organisation must also develop internal capability to maintain the databases and deal with changing requirements. This usually means hiring and retaining a group of technical specialists, such as DBAs, who need to be trained (and re-trained to keep up with the technology). End-users too will need to be trained to use the system properly. In other words, there are considerable running costs as well. In all, databases can be *very* expensive.

Aside from the expenses above, databases often are crucial to a business. Imagine what would happen, say, if databases of customer accounts maintained by a bank were

(accidentally or maliciously) destroyed! Because of these actual and potential costs, databases must be deliberately protected against any conceivable harm.

Generally, there are three types of security that must be put in place:

1. *Physical Protection*: these are protective measures to guard against natural disasters (eg. fire, floods, earthquakes, etc), theft, accidental damage to equipment and other threats that can cause the physical loss of data. This is generally the area of physical installation management and is outside the scope of this book.
2. *Operational Protection*: these are measures to minimise or even eliminate the impact of human errors on the databases' integrity. Errors can occur, for example, in assigning values to attributes - a value may be unreasonable (eg. an age of 213!) or of the wrong type (eg. the value 'Smith' assigned to the age attribute). These measures are typically embodied in a set of *integrity constraints* (a set of assertions) that must be enforced (ie. the truth of the assertions must be preserved across all database transactions). An example of an assertion might be 'the price of a product must be a positive number'. Any operation then is invalid if it violates a stated constraint, eg. "Store ... Price= -9.99". These constraints are typically specified by a DBA in the database schema.
3. *Authorisational Protection*: these are measures to ensure that access to the databases are by authorised users only, and then only for specific modes of access (eg. some users may only be allowed to read while others can modify database contents). They are necessary to ensure that confidentiality and correctness of information is preserved. Access control can be applied at various levels in the system. At the installation level, access through computer terminals may be controlled using special access cards or passwords. At successively lower levels, control may be applied to an entire database, to its physical devices (or parts thereof), or to its logical parts (parts of the schema). In extremely sensitive problem domains, access control may even be applied to individual instances or data objects in the database.