Database fundamentals

Introduction

More and more of today's applications need to handle large amounts of data. Some have done this for a long time - the transactional systems that businesses use to manage their processes are obvious examples. Some companies have hundreds of thousands of customers; each customer may interact with the company on many occasions generating millions of sales; over several years, the company might accumulate billions of records that all need to be stored, reported on and occasionally searched.

All social networking sites store the interactions between members in a database. Again, for a popular site, the number of interactions could be astronomical, and they all need to be available at a second's notice when a user's page is accessed.

In fact, any activity which needs to track large amounts of data will do so using a database - think of ticketing operations, airports, sporting venues, cinema scheduling, university administration, mp3 downloading, the electoral register, and so on...

Database technology is not exciting. It is not new or cutting edge, but if you don't know how to use if effectively, you will severely limit your understanding of the way many types of system work. Whether you are a system developer or an end user, this is a technology you need to understand.

This first set of notes goes through some of the basic ideas on which database systems are based. You will notice that there is a lot of terminology associated with the subject, and it is a good idea to learn to use these terms properly. Technical terminology allows you to express complex ideas quickly and precisely. If you use the wrong terms or you use terms incorrectly, there is a risk of misunderstandings and hence errors in the project you are working on, and time is wasted on clarifications later. Another good reason to learn the correct terms is that you needs to show other people that you know what you are talking about. Would you trust a systems analyst or developer who did not know the right terms?

The database approach

A database is a computer-based system to record and maintain information about anything of significance to an organisation. Modern databases almost always store their contents in structures called *tables* like the example below.

Imagine a case where we want to store facts about employees in a company. Such facts could include their name, address, date of birth, and salary. A table looks very much like a spreadsheet page with different employees as the rows, and the facts (e.g. their names) as columns. The table must have a name, and this one could be called EMP (short for employee).

Name	Address	Date of Birth	Salary
Jim Smith	1 Apple Lane	1/3/1991	11000
Jon Greg	5 Pear St	7/9/1992	13000
Bob Roberts	2 Plum Road	3/2/1990	12000

The database contains two types of information:

- The *schema* is the structure of data including the names of tables and columns. The schema defines the container in which real data is stored.
- *Data* are the "facts" that are stored in the databases tables. The schema defines the rules that the data must obey.

From this information the *schema* would define that EMP has four components, "NAME","ADDRESS","DOB","SALARY". As designers we can name the columns however we like, but making them meaningful helps. In addition to the name, we want to try and make sure that people don't accidentally store a name in the DOB column, or some other silly error. Protecting the database against bad data is one of the most important database design steps, and is what much of this course is about. From what we know about the facts, we can say things like:

- NAME is a string, and needs to hold at least 12 characters.
- ADDRESS is a string, and needs to hold at least 12 characters.
- DOB is a date... The company forbids people over 100 years old or younger than 18 years old working for them.
- SALARY is a number. It must be greater than zero.

These rules are called *constraints* and can be enforced by the database management system (*DBMS*). Constraints are identified during the design phase of a database schema. The more constraints there are, the harder it is to enter poor quality data.

Advantages of databases

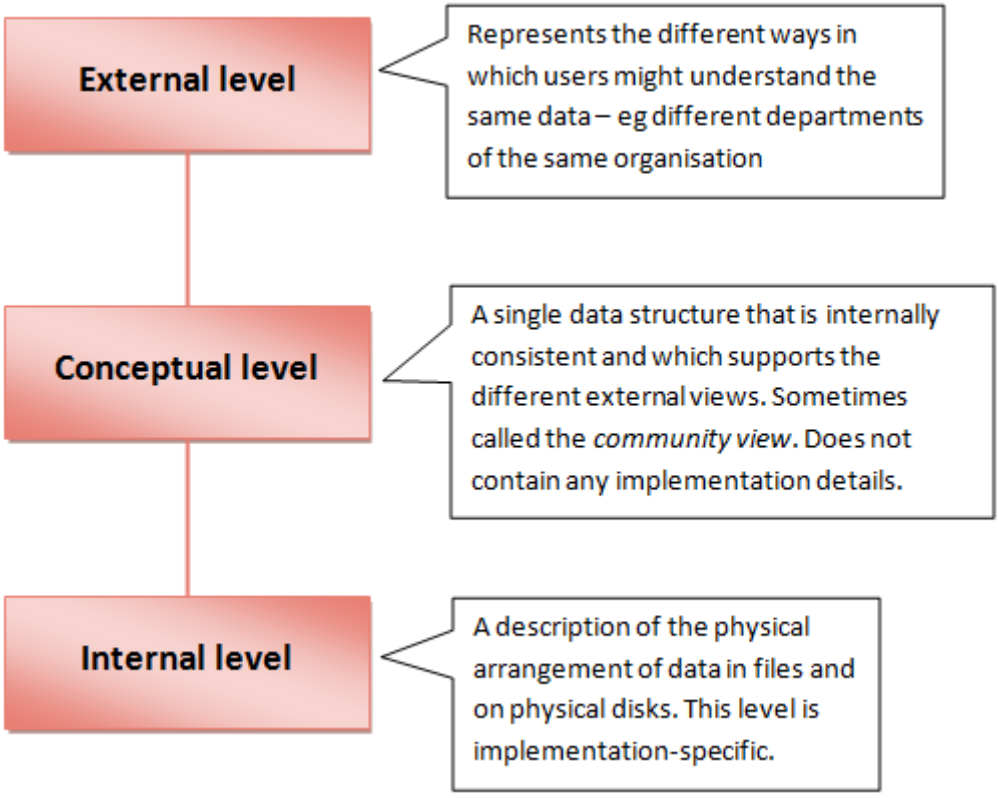
The facilities offered by DBMS vary a great deal, depending on their level of sophistication. In general, however, a good DBMS should provide the following advantages over alternative systems:

Advantage	Description
Data independence	Both the database and the user program can be altered independently of each other thus saving time and money which would be required to retain consistency.
Multi-user access	Many user views may be supported by a single logically consistent structure so that communications delays and transmission errors can be eliminated
Data integration	Data redundancy is eliminated by storing data in a single place
Data integrity	Structural errors are eliminated by the definition and enforcement of rules about the kind of data that is allowed and what can be done with it
Enforcement of standards	With central control of the database, the DBA can ensure that standards are followed in the representation of data. These may be internal to an organisation, or may be required by external legislation, for example.
Security	Having control over the database the DBA can ensure that access to the database is through proper channels and can define the access rights of any user to any data items or defined subset of the database. The security system can prevent corruption of the existing data either accidently or maliciously.
Performance	Data from different sources within an organisation can be combined on demand. The physical operation of the database can be optimised by the DBA.

3-level architecture

The term architecture refers to a high-level view of the structure of a system. It can be used at different levels of abstraction - for example, we can talk about a client-server architecture when thinking about the structure of an application. A client-server application typically includes a database which in some cases may account for the entire server component; however, at a lower level, we can also use the term when thinking about the structure of the database itself.

The accepted standard architecture for most relational databases is the three-level ANSI-SPARC model shown in the diagram below. ANSI-SPARC stands for the American National Standards Institute Standard Planning and Requirement Committee.



External level

A user is anyone who needs to access the data including end users (business users), application programmers and database administrators. One group of users may require only a subset of the information; another group may need information to be presented in a particular way. For example, a university accommodation service might need access to limited information about students to check their eligibility and to lists of available rooms and flats. Academic tutors require different information about students and are concerned about modules and programmes rather than accommodation. All information would be stored in a comprehensive university database, but different users require different *views*.

Conceptual level

Different user needs must be supported by an overall database design which constitutes a model of the real world, and which must also be efficient and internally consistent. The job of the database designer is to abstract this model from information collected from users, and to ensure that it is correctly structured. This is the level where we will concentrate out efforts in this module.

Internal level

With very few exceptions, all data in computer systems is stored in files on disks, and the way in which data is spread across files and disks can have a significant impact on the efficiency of a database. The internal view is concerned with these low-level storage issues and describes what files exist and how they are managed. This level is of more concern to the database administrator (DBA) than to the end-user or application programmer.

Mappings

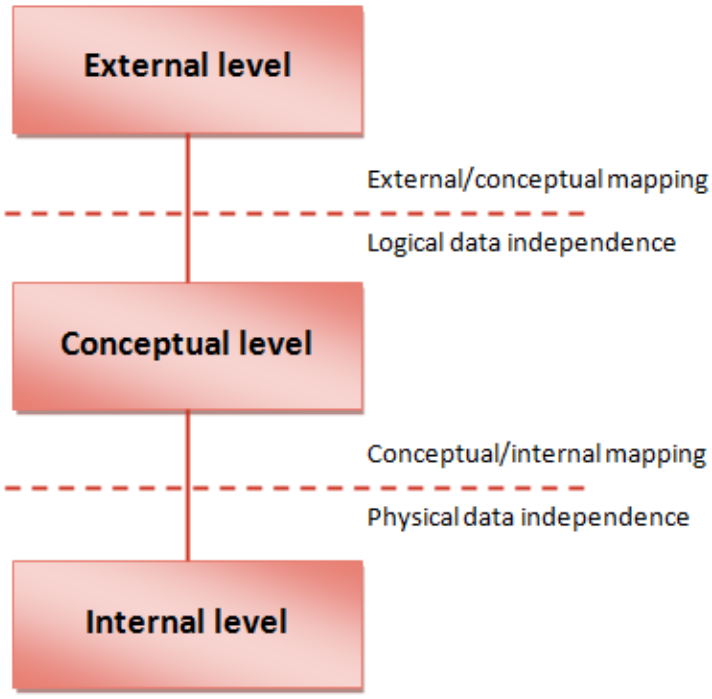
When a user (of any kind) interacts with the database, they will do so using their own view of the data. Any view-specific terms they use need to be mapped onto the shared conceptual structure. References to objects at the conceptual level must be resolved into file access requests at the internal level in order of the user's initial request to succeed. The task of mapping from one level to another is carried out transparently by the database management system (DBMS). Changes at one level - such as a file reorganisation by the DBA - entails a change to the mapping from the modified level to the next; however, the structure of the next level does not change - this is the basis of data independence.

Data independence

The structures at each level of the ANSI-SPARC architecture are independent of structures at other levels. Thus one level may change without having an impact on any other level - the thing that does need to change is the mapping from the modified level to a related level, and that is handled transparently by the DBMS.

For example, the way a department processes some of the data may change in such a way that requires the data to be presented differently. The data is the same, but the presentation is different; therefore, there is no change required at the conceptual level. Likewise, a DBA may reorganise the files in which the data is stored, but again does not require any changes at the conceptual level.

The ANSI-SPARC architecture therefore delivers two types of data independence as illustrated in the diagram below.



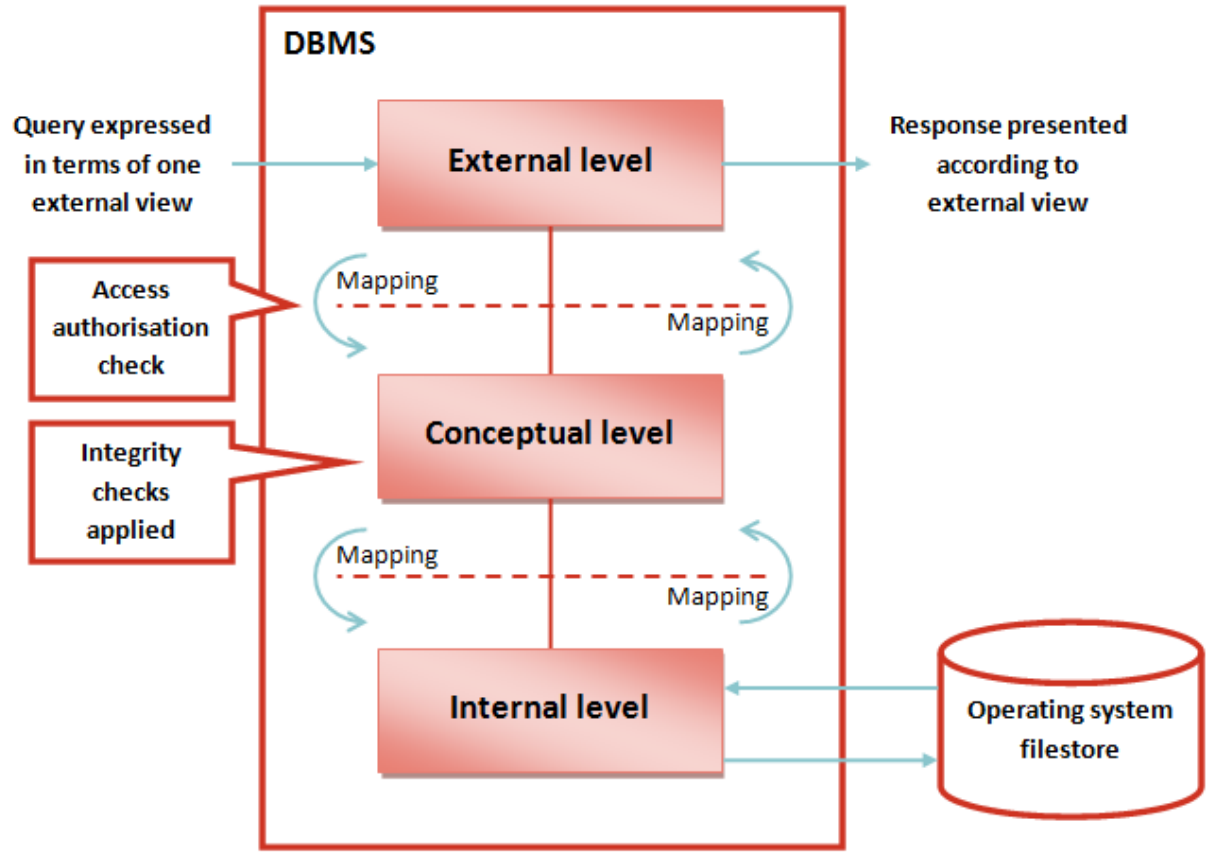
Data independence means that the extent of any change in the database structure is limited and can be assigned to someone in an appropriate role with the appropriate knowledge. Applications that do not exhibit data independence have no such clean divisions, and so a simple conceptual change - changing the length of a stored data item, for example - might require many individual updates throughout the application. Changes to the way in which data is physically stored could affect logical structures and user views. Maintenance is therefore inefficient because the entire application must be checked through even for the simplest modification.

This module is mainly - but not exclusively - concerned with the conceptual level.

The DBMS

The database management system (DBMS) is the software that

- handles all access to the database
- performs the mapping from one level of the ANSI-SPARC architecture to the next
- is responsible for applying the authorisation checks and validation procedures including integrity checks



Data redundancy

In non-database systems each application has its own private files. This can often lead to multiple copies of the same data stored in different places. If one copy is changed, the overall picture becomes inconsistent. Even if there are comprehensive checking procedures in place to ensure that all data stores are updated at the same time, this entails a large management overhead.

Thus data redundancy is inefficient and can lead to inconsistencies in the data (loss of data integrity)

The database may be thought of as a unification of several otherwise distinct data files, with any redundancy among those files partially or wholly eliminated. Some of the techniques that will be covered in this module are specifically to do with taking a structure that contains redundancy and resolving them through the application of procedural rules.

Redundancy is

- *direct* if a value is a copy of another
- *indirect* if the value can be derived from other values

Data integration is generally regarded as an important characteristic of a database. The rule of thumb is that a single piece of data should only be stored once, and if it can be derived from other data, it should not be stored at all. However, there are some cases where this guideline may be broken to speed up a particular database operation, or simplify a user view. Identifying the need for such exceptions is an advanced topic, however, and requires a detailed understanding of database technology.

Data integrity

At the conceptual level, a database provides a model of the real world using four simple components:

Component	Description	Examples
Entities	The important things in the real world that need to be modelled	People, places, objects, events, etc.
Attributes	Individual items of data associated with an entity	Name, national insurance number, weight, date of manufacture
Relationships	Ways in which entities are connected	A is part of B, A lives in B, A produces B, A takes place in B, etc.
Constraints	Rules which place limits on the data that is allowed	Every A must have a B, Only future dates are allowed, etc.

Relationships and constraints form a set of rules to which data must conform. Data integrity is the validity and consistency of the database contents with respect to these rules. Data integrity must be maintained; otherwise meaningless data starts to accumulate. For example, if a database contains two entries for the same entity, it would be possible to modify one copy and leave the other as it is. Afterwards there is no way of knowing which copy is "correct", and we have lost some data integrity. The solution would be to enforce a rule that says only one entry is allowed for a single entity.

Integrity checks on data items can be divided into 4 groups:

1. type checks

   e.g. ensuring a numeric field is numeric and not a character - this check should be performed automatically by the DBMS.

2. redundancy checks

   direct or indirect (see data redundancy) - this check is not automatic in most cases and must be added by the database designer

3. range checks

   e.g. to ensure a data item value falls within a specified range of values, such as checking dates so that say (age > 0 AND age < 110).

4. comparison checks

   in this check a function of a set of data item values is compared against a function of another set of data item values. For example, the max salary for a given set of employees must be less than the min salary for the set of employees on a higher salary scale.

A record type may have constraints on the total number of occurrences, or on the insertions and deletions of records. For example in a patient database there may be a limit on the number of x-ray results for each patient or the details of a patients visit to hospital must be kept for a minimum of 5 years before it can be deleted

Centralised control of the database helps maintain integrity, and permits the DBA to define validation procedures to be carried out whenever any update operation is attempted (update covers modification, creation and deletion of data).

Roles

In general there are three broad classes of database user:

1. the *end user*, who accesses the database via a query language or application interface
2. the *application programmer*, responsible for writing programs in some high-level language such as Java, C#, etc.
3. the *database administrator* (DBA), who controls all operations on the database

End users

End users are typically people who work in some non-technical capacity in an organisation. Part of their job is to process information in some way, either by storing it, updating it or using it in the form of reports. Their level of technical skill is typically low, but they are nevertheless the reason for the existence of the database and any applications that use it.

End users have their own small view of the total mass of data in the organisation. Their understanding of the data is completely determined by the priorities of their own role, and part of the job of the database designer is to provide each end user with a view of the data that matches their expectation and needs. For example, an employee in a factory despatch department needs information about orders ready to be shipped and available transport. They need that information presented in a clear and targeted way which will support efficient logistic operations.

This might mean

- filtering out data about shipments not due today
- providing summary information about total weight of several orders, optimised delivery routes, etc.
- using logistics-specific terminology such as "goods" instead of "products", "carriers" rather than "suppliers", etc.

End users typically rely on purpose-built application interfaces or query tools for their interactions with the data. For ad hoc queries they may use query by example (QBE) interfaces, but more technically capable end users may use SQL.

Application programmers

An application programmer may be required to implement a set of use cases using an existing database by constructing a user interface in an appropriate language. It is more likely however that the application programmer will need to design and build the database as well. This requires and investigation into the "world" of the end users so that an adequate shared model can be built.

Whether they are responsible for the database design or not, application programmers need to understand the structure of the database clearly so that they can provide an efficient interface which delivers all of the benefits of the database approach. They need an expert knowledge of SQL as the industry-standard query language, an understanding of how to integrate SQL statements into specific development languages such as Java, and a working knowledge of the methods available for connecting an application to a database.

Database Administrator

The database administrator (DBA) is the person (or group of people) responsible for overall control of the database system. The DBA may be required to design the logical structure of the database; however, this is not always the case. The DBA's main responsibilities include the following:

- setting the storage structure and access strategy, i.e. how the data is to be represented by writing the storage structure definition
- liaising with users - for example to ensure that the data they require is available
- defining authorisation checks and validation procedures.
- defining a strategy for backup and recovery - for example periodic dumping of the database to a backup tape and procedures for reloading the database for backup.
- monitoring performance and responding to changes in requirements - for example adding additional disk storage, archiving old data, adding database-specific performance improvements

Dedicated database administration tools are available, but the DBA requires an expert knowledge of SQL, especially the statements used for creating and modifying database objects known as data definition language (DDL). The DBA also needs to understand the operating system very well so that administrative activities can be performed efficiently.