**File-based** A collection of application programs that perform services for the system end-users such as the production of reports. Each program defines and manages its own data.

File-based systems were an early attempt to computerize the manual filing system that we are all familiar with.

**Example**, in an organization a manual file is set up to hold all external and internal correspondence relating to a project, product, task, client, or employee. Typically, there are many such files, and for safety they are labeled and stored in one or more cabinets. For security, the cabinets may have locks or may be located in secure areas of the building.

**Example**, In our own home, we probably have some sort of filing system which contains receipts, guarantees, invoices, bank statements, and such like. When we need to look something up, we go to the filing system and search through the system starting from the first entry until we find what we want.

Alternatively, we may have an **indexing system** that helps locate what we want more quickly.

**Example**, we may have divisions in the filing system or separate folders for different types of item that are in some way logically related. A typical real estate agent’s office might have a separate file for each property for sale or rent, each potential buyer and renter, and each member of staff. Consider the effort that would be required to answer the following questions:

What three-bedroom properties do you have for sale with a garden and garage?

What flats do you have for rent within three miles of the city center?

What is the average rent for a two-bedroom flat?

What is the total annual salary bill for staff?

How does last month’s turnover compare with the projected figure for this month?

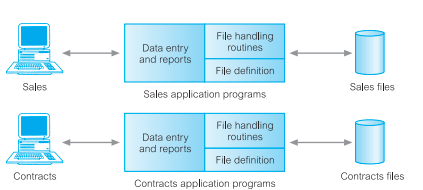
What is the expected monthly turnover for the next financial year?

The file based system was developed in response to the needs of industry for more efficient data access. However, rather than establish a **centralized store** for the organization’s operational data, a decentralized approach was taken, where each department, with the assistance of Data Processing (DP) staff, stored and controlled its own data.

**Limitations of the File-Based Approach** This brief description of traditional file-based systems should be sufficient to discuss the limitations of this approach. We list five problems in Table 1.1.

**Separation and isolation of data** When data is isolated in separate files, it is more difficult to access data that should be available. **For example**, if we want to produce a list of all houses that match the requirements of clients, we first need to create a temporary file of those clients who have ‘house’ as the preferred type. We then search the Property For Rent file for those properties where the property type is ‘house’ and the rent is less than the client’s maximum rent. With file systems, such processing is difficult. The application developer must synchronize the processing of two files to ensure the correct data is extracted. This difficulty is compounded if we require data from more than two files.

**Duplication of data** Due to the decentralized approach taken by each department, the file-based approach encouraged, if not necessitated, the uncontrolled duplication of data.



We can clearly see that there is duplication of both property and client details in the Sales and Contracts Departments. Uncontrolled duplication of data is undesirable for several reasons.

* Duplication is wasteful. It costs time and money to enter the data more than once.
* It takes up additional storage space, again with associated costs. Often, the duplication of data can be avoided by sharing data files.
* Perhaps more importantly, duplication can lead to loss of data integrity; in other words, the data is no longer consistent.

**Data dependence** As we have already mentioned, the physical structure and storage of the data files and records are defined in the application code. This means that changes to an existing structure are difficult to make. For example, increasing the size of the PropertyForRent address field from 40 to 41 characters sounds like a simple change, but it requires the creation of a one-off program (that is, a program that is run only once and can then be discarded) that converts the PropertyForRent file to the new format.

**Incompatible file formats** Because the structure of files is embedded in the application programs, the structures are dependent on the application programming language.

**For example**, the structure of a file generated by a COBOL program may be different from the structure of a file generated by a ‘C’ program. The direct incompatibility of such files makes them difficult to process jointly.

**For example**, suppose that the Contracts Department wants to find the names and addresses of all owners whose property is currently rented out. Unfortunately, Contracts does not hold the details of property owners; only the Sales Department holds these. However, Contracts has the property number (propertyNo), which can be used to find the corresponding property number in the Sales Department’s PropertyForRent file. This file holds the owner number (ownerNo), which can be used to find the owner details in the PrivateOwner file. The Contracts Department programs in COBOL and the Sales Department programs in ‘C’. Therefore, to match propertyNo fields in the two PropertyForRent files requires an application developer to write software to convert the files to some common format to facilitate processing. Again, this can be time-consuming and expensive.