Multi-User DBMS Architectures

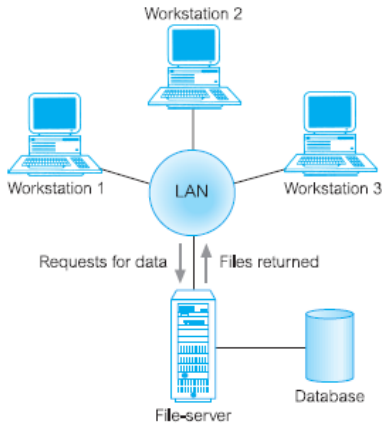
Common architectures that are used to implement multi-user database management systems, namely teleprocessing, file-server, and client–server.

Teleprocessing

- The traditional architecture for multi-user systems was teleprocessing, where there is one computer with a single central processing unit (CPU) and a number of terminals.
- All processing is performed within the boundaries of the same physical computer. User terminals are typically ‘dumb’ ones, incapable of functioning on their own.
- They are cabled to the central computer. The terminals send messages via the communications control subsystem of the operating system to the user’s application program, which in turn uses the services of the DBMS.
- In the same way, messages are routed back to the user’s terminal. Unfortunately, this architecture placed a tremendous burden on the central computer, which not only had to run the application programs and the DBMS, but also had to carry out a significant amount of work on behalf of the terminals (such as formatting data for display on the screen).
- In recent years, there have been significant advances in the development of highperformance personal computers and networks.
- There is now an identifiable trend in industry towards **downsizing**, that is, replacing expensive mainframe computers with more cost-effective networks of personal computers that achieve the same, or even better, results. This trend has given rise to the next two architectures: file-server and client–server.

File-Server Architecture

In a file-server environment, the processing is distributed about the network, typically a local area network (LAN)..
The file-server holds the files required by the applications and the DBMS.
However, the applications and the DBMS run on each workstation, requesting files from the file-server.
In this way, the file-server acts simply as a shared hard disk drive. The DBMS on each workstation sends requests to the file-server for all data that the DBMS requires that is stored on disk.
This approach can generate a significant amount of network traffic, which can lead to performance problems.



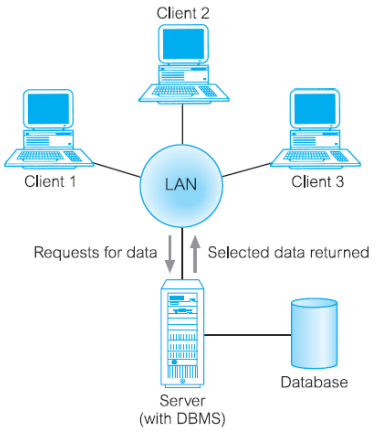
The file-server architecture, therefore, has three main disadvantages:

- There is a large amount of network traffic.
- A full copy of the DBMS is required on each workstation.
- Concurrency, recovery, and integrity control are more complex because there can be multiple DBMSs accessing the same files.

Traditional Two-Tier Client–Server Architecture

To overcome the disadvantages of the first two approaches and accommodate an increasingly decentralized business environment, the client–server architecture was developed.

Client–server refers to the way in which software components interact to form a system.



As the name suggests, there is a **client** process, which requires some resource, and a **server**, which provides the resource. There is no requirement that the client and server must reside on the same machine.

Data-intensive business applications consist of four major components: the database, the transaction logic, the business and data application logic, and the user interface.
The traditional two-tier client–server architecture provides a very basic separation of these components. The client (tier 1) is primarily responsible for the *presentation* of data to the user, and the server (tier 2) is primarily responsible for supplying *data services* to the client.

Presentation services handle user interface actions and the main business and data application logic.

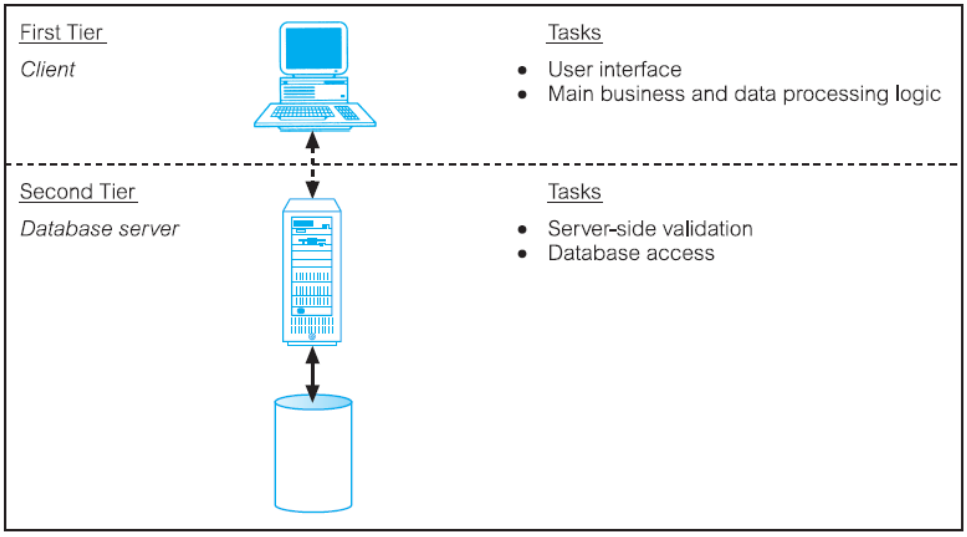
Data services provide limited business application logic, typically validation that the client is unable to carry out due to lack of information, and access to the requested data, independent of its location. The data can come from relational DBMSs, object-relational DBMSs, object-oriented DBMSs, legacy DBMSs, or proprietary data access systems. Typically, the client would run on end-user desktops and interact with a centralized database server over a network.

The client takes the user's request, checks the syntax and generates database requests in SQL or another database language appropriate to the application logic. It then transmits the message to the server, waits for a response, and formats the response for the end-user. The server accepts and processes the database requests, then transmits the results back to the client. The processing involves checking authorization, ensuring integrity, maintaining the system catalog, and performing query and update processing. In addition, it also provides concurrency and recovery control

There are many advantages to this type of architecture. For example:

- It enables wider access to existing databases.
- Increased performance – if the clients and server reside on different computers then different CPUs can be processing applications in parallel. It should also be easier to tune the server machine if its only task is to perform database processing.
- Hardware costs may be reduced – it is only the server that requires storage and processing power sufficient to store and manage the database.
- Communication costs are reduced – applications carry out part of the operations on the client and send only requests for database access across the network, resulting in less data being sent across the network
- Increased consistency – the server can handle integrity checks, so that constraints need be defined and validated only in the one place, rather than having each application program perform its own checking.

- It maps on to open systems architecture quite naturally.



Disadvantages

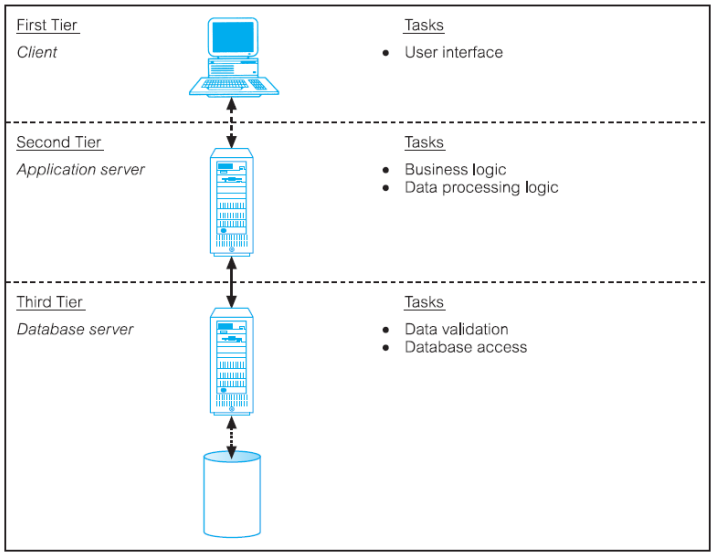
- A ‘fat’ client, requiring considerable resources on the client’s computer to run effectively. This includes disk space, RAM, and CPU power.
- A significant client-side administration overhead.

Three-Tier Client–Server Architecture

This new architecture proposed three layers, each potentially running on a different platform:

- The user interface layer, which runs on the end-user’s computer (the *client*).
- The business logic and data processing layer. This middle tier runs on a server and is often called the *application server*.
- A DBMS, which stores the data required by the middle tier. This tier may run on a separate server called the *database server*.

The client is now responsible only for the application’s user interface and perhaps performing some simple logic processing, such as input validation, thereby providing a ‘thin’ client.
The core business logic of the application now resides in its own layer, physically connected to the client and database server over a local area network (LAN) or wide area network (WAN). O ne application server is designed to serve multiple clients.



The three-tier design has many advantages over traditional two-tier or single-tier designs, which include:

- The need for less expensive hardware because the client is ‘thin’.
- Application maintenance is centralized with the transfer of the business logic for many end-users into a single application server. This eliminates the concerns of software distribution that are problematic in the traditional two-tier client–server model.
- The added modularity makes it easier to modify or replace one tier without affecting the other tiers.
- Load balancing is easier with the separation of the core business logic from the database functions.
- An additional advantage is that the three-tier architecture maps quite naturally to the Web environment, with a Web browser acting as the ‘thin’ client, and a Web server acting as the application server.

The three-tier architecture can be extended to *n*-tiers, with additional tiers added to provide more flexibility and scalability. For example, the middle tier of the three-tier architecture could be split into two, with one tier for the Web server and another for the application server.
This three-tier architecture has proved more appropriate for some environments, such as the Internet and corporate intranets where a Web browser can be used as a client. It is also an important architecture for **Transaction Processing Monitors**.

Transaction Processing Monitors

TP Monitor is a program that controls data transfer between clients and servers in order to provide a consistent environment, particularly for online transaction processing (OLTP).

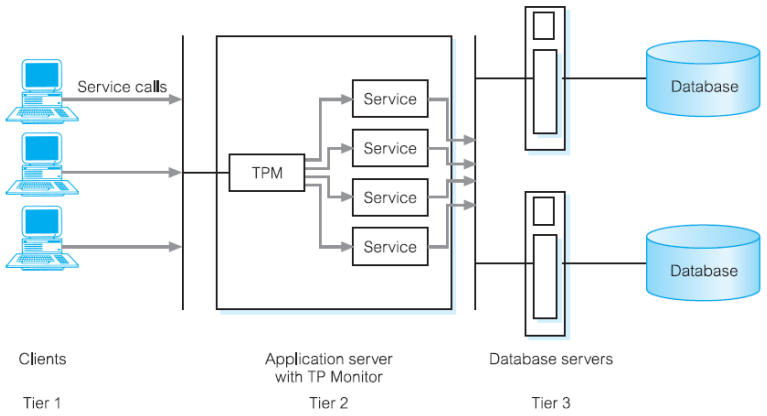


Figure 2.16
Transaction Processing Monitor as the middle tier of a three-tier client–server architecture.

Complex applications are often built on top of several **resource managers** (such as DBMSs,operating systems, user interfaces, and messaging software). A Transaction Processing Monitor, or TP Monitor, is a middleware component that provides access to the services of a number of resource managers and provides a uniform interface for programmers who are developing transactional software. A TP Monitor forms the middle tier of a three-tier Architecture. TP Monitors provide significant advantages, including:

- *Transaction routing* The TP Monitor can increase scalability by directing transactions to specific DBMSs.
- *Managing distributed transactions* The TP Monitor can manage transactions that require access to data held in multiple, possibly heterogeneous, DBMSs. For example, a transaction may require to update data items held in an Oracle DBMS at site 1, an Informix DBMS at site 2, and an IMS DBMS as site 3. TP Monitors normally control
- transactions using the X/Open Distributed Transaction Processing (DTP) standard. A DBMS that supports this standard can function as a resource manager under the control of a TP Monitor acting as a transaction manager.
- *Load balancing* The TP Monitor can balance client requests across multiple DBMSs on one or more computers by directing client service calls to the least loaded server.
- In addition, it can dynamically bring in additional DBMSs as required to provide the necessary performance.
- *Funneling* In environments with a large number of users, it may sometimes be difficult for all users to be logged on simultaneously to the DBMS. In many cases, we would find that users generally do not need continuous access to the DBMS. Instead of each user connecting to the DBMS, the TP Monitor can establish connections with the DBMSs as and when required, and can funnel user requests through these connections. This allows a

larger number of users to access the available DBMSs with a potentially much smaller number of connections, which in turn would mean less resource usage.

- *Increased reliability* The TP Monitor acts as a *transaction manager*, performing the necessary actions to maintain the consistency of the database, with the DBMS acting as a *resource manager*. If the DBMS fails, the TP Monitor may be able to resubmit the transaction to another DBMS or can hold the transaction until the DBMS becomes available again.

TP Monitors are typically used in environments with a very high volume of transactions, where the TP Monitor can be used to offload processes from the DBMS server. Prominent examples of TP Monitors include CICS and Encina from IBM (which are primarily used on IBM AIX or Windows NT and bundled now in the IBM TXSeries) and Tuxedo from
BEA Systems.