INF3703
Databases II

October 15, 2024

Contents

Chapter 10

Database Development Process

10.1 The Information System

A **database** is a carefully designed and constructed repository of facts. This fact repository is part of a larger whole, known as an **information system**. An information system provides for data collection, storage, and retrieval. It also facilitates the transformation of data into information, and the management of both data and information. A complete information system is composed of people, hardware, software, procedures, the databases, and application programs.

Systems Analysis is the process that establishes the need for, and the scope of, an information system. The process of creating an information system is known as **systems development**.

Within the framework of systems development, applications transform data into the information that forms the basis of decision-making. Every application is composed of two parts: the data, and the code by which the data are transformed into information.

The performance of an information system depends on a triad of factors:

1. Database design and implementation
2. Application design and implementation
3. Administrative procedures

In a broad sense, the term **database development** describes the process of database design and implementation. The primary objective in database design is to create complete, normalised, non-redundant (to the extent possible) and fully integrated conceptual, logical, and physical database models. The implementation phase includes creating the database storage structure, loading data into the database, and providing for data management.

10.2 The Systems Development Life Cycle

Phases of the SDLC

Planning	• Initial assessment • Feasibility study
Analysis	• User requirements • Existing system evaluation • Logical system design
Detailed System Design	• Detailed system specification
Implementation	• Coding, testing and debugging • Installation, fine-tuning
Maintenance	• Evaluation • Maintenance • Enhancement

The SDLC is an iterative rather than a sequential process.

10.2.1 Planning

The SDLC planning phase yields a general overview of the company and its objectives.

Initial Assessment

An initial assessment of the information-flow-and-extent requirements must be made during this discovery portion of the SDLC. This should answer the questions:

- Should the existing system be continued?
- Should the existing system be modified?
- Should the existing system be replaced?

Participants in the SDLC's initial assessment must begin to study and evaluate alternative solutions. If it is decided that a new system is necessary, the next question is whether it is feasible.

Feasibility Study

The feasibility study must address the following:

- The technical aspects of hardware and software requirements
- The system cost
- The operational cost

10.2.2 Analysis

Problems defined during the planning phase are examined in greater detail in the analysis phase. A macroanalysis must be made of both individual and organisational needs, addressing questions such as:

- What are the requirements of the current system's end users?
- Do those requirements fit into the overall information requirements?

The existing hardware and software are also studied during the analysis phase. The result of analysis should be a better understanding of the system's functional areas, actual and potential problems and opportunities.

End users and the system designer(s) must work together to identify processes and to uncover potential problem areas.

Along with a study of user requirements and the existing systems, the analysis phase also includes the creation of a logical systems design. The logical design must specify the appropriate conceptual data model, inputs, processes, and expected output requirements.

The database design's data-modelling activities take place at this point, to discover and describe all entities and their attributes, and the relationships among the entities within the database.

Defining the logical system also yields functional descriptions of the system's components (modules) for each process within the database environment. All data transformations are described and documented using systems analysis tools such as data flow diagrams (DFDs). The conceptual data model is validated against those processes.

10.2.3 Detailed System Design

The designer completes the design of the system's processes. The design includes all necessary technical specifications for the screens, menus, reports, and other devices, that might be used to help make the system a more efficient information generator. The steps are laid out for conversion from the old to the new system. Training principles and methodologies are also planned, and must be submitted for management's approval.

10.2.4 Implementation

During the implementation phase, the hardware, DBMS software, and application programs, are installed, and the database design is implemented. During the initial stages of the implementation phase, the system enters into a cycle of coding, testing, and debugging, until it is ready to be delivered. The actual database is created, and the system is customised by the creation of tables and views, user authorisation, and so on.

The database contents may be loaded interactively or in batch mode, using a variety of methods and devices:

- Customised user programs
- Database interface programs
- Conversion programs that import the data from a different file structure, using batch programs, a database utility, or both.

The system is subjected to exhaustive testing until it is ready for use. Traditionally, the implementation and testing of a new system took 50 to 60 percent of the total development time. However, the advent of sophisticated application generators and debugging tools has substantially decreased coding and testing time.

After testing is concluded, the final documentation is reviewed and printed, and end users are trained. The system is in full operation at the end of this phase, but will be continuously evaluated and fine-tuned.

10.2.5 Maintenance

Almost as soon as the system is operational, end users begin to request changes in it. Those changes generate system maintenance activities, which can be grouped into three types:

- **Corrective Maintenance** In response to system errors.
- **Adaptive Maintenance** Due to changes in the business environment.
- **Perfective Maintenance** To enhance the system

Because every request for structural change requires retracing the SDLC steps, the system is, in a sense, always at some stage of the SDLC.

Every system has a predetermined operational lifespan. The actual operational lifespan of a system depends on its perceived utility. There are several reasons for reducing the operational life of certain systems: some include rapid technological change (especially for systems based on processing speed and expandability) and the cost of maintaining a system.

If the system's maintenance cost is high, its value becomes suspect. **Computer-aided systems engineering (CASE)** technology, such as System Architect or Visio Professional, helps make it possible to product better systems within a reasonable amount of time and at a reasonable cost. In addition, the more structured, better-documented, and *standardised* implementation, of CASE-produced applications tends to prolong the operational life of systems by making them easier and cheaper to update and maintain.

10.3 The Database Life Cycle

10.3.1 The Database Initial Study

If a designer has been called in, it is likely that the current system has failed to perform functions deemed vital by the company. So, in addition to examining the current system's operation within the company, the designer must determine how and why the current system fails.

The overall purpose of the database initial study is to:

- Analyse the company situation
- Define problems and constraints
- Define objectives
- Define scope and boundaries

Phases of the DBLC

Database Initial Study	• Analyse the company situation • Define problems and constraints • Define objectives • Define scope and boundaries
Database Design	• Create the conceptual design • DBMS software selection • Create the logical design • Create the physical design
Implementation and Loading	• Install the DBMS • Create the databases • Load or convert the data
Testing and Evaluation	• Test the database • Fine-tune the database • Evaluate the database and its application programs
Operation	• Produce the required information flow
Maintenance and Evolution	• Introduce changes • Make enhancements

Analyse the Company Situation

The **company situation** describes the general conditions in which a company operates, its organisational structure, and its mission. To analyse the company situation, the database designer must discover what the company's operational components are, how they function, and how they interact.

- What is the organisation's general operating environment, and what is its mission within that environment?
- What is the organisation's structure?

Define Problems and Constraints

The designer has both formal and informal sources of information. If the company has existed for any length of time, it already has some form of system in place. It can be useful to consider how the existing system functions, what its inputs and outputs are, the documents generated, how the documents are used, and who the documents are used by. Remember to consider the differences between the official version of a system's operation, and the more informal, real version.

The problem definition process might initially appear to be unstructured. Company end users are often unable to describe precisely the larger scope of company operations, or to identify the real problems encountered during company operations. Often the managerial view of a company's operation is different from that of the end users who perform the actual routine work.

Finding precise answers is important, especially concerning the operational relationships among business units.

Even the most complete and accurate problem definition does not lead to the perfect solution. The real world usually intrudes to limit the design of even the most elegant database by imposing constraints.

Such constraints include time, budget, personnel, and more. The designer must learn to distinguish between what's perfect and what's possible. Also, the designer must look for the *source* of the problem, rather than designing a system to treat the *symptoms* of the problem.

Define Objectives

A proposed database system must be designed to help solve at least the major problems identified during the problem discovery process. As the list of problems unfolds, several common sources are likely to be discovered.

The initial study phase also yields proposed problem solutions. The designer's job is to ensure that the database system objectives, as seen by the designer, correspond to those envisioned by the end-user(s).

The database designer must begin to address the following questions:

- What is the proposed system's initial objective?
- Will the system interface with other existing or future systems in the company?
- Will the system share the data with other systems or users?

Define Scope and Boundaries

The system's **scope** defines the extent of the design according to operational requirements. For example, will the database design encompass the entire organisation, one or more departments within the organisation, or one or more functions of a single department? Knowing the database design scope helps in defining the required data structures, the type and number of entities, the physical size of the database, and so on.

The proposed system is also subject to limits known as **boundaries**, which are external to the system. Boundaries include time and budget limitations. They are also imposed by existing hardware and software.

10.3.2 Database Design

The second phase focuses on the design of the database model that will support company operations and objectives. This is arguably the most critical DBLC phase: making sure that the final product meets user and system requirements. In the process of database design, you must concentrate on the data characteristics required to build the database model.

At this point, there are two views of the data within the system: the business view of data as a source of information, and the designer's view of the data structure, its access, and the activities required to transform the data into information.

- The process of database design is loosely related to the analysis and design of a larger system. The data component is only one element of a larger information system.
- The systems analysts or systems programmers are in charge of designing the other system components. Their activities create the procedures that will help transform the data within the database into useful information.
- The database design does not constitute a sequential process. Rather, it is an iterative process that provides continuous feedback designed to trace previous steps.

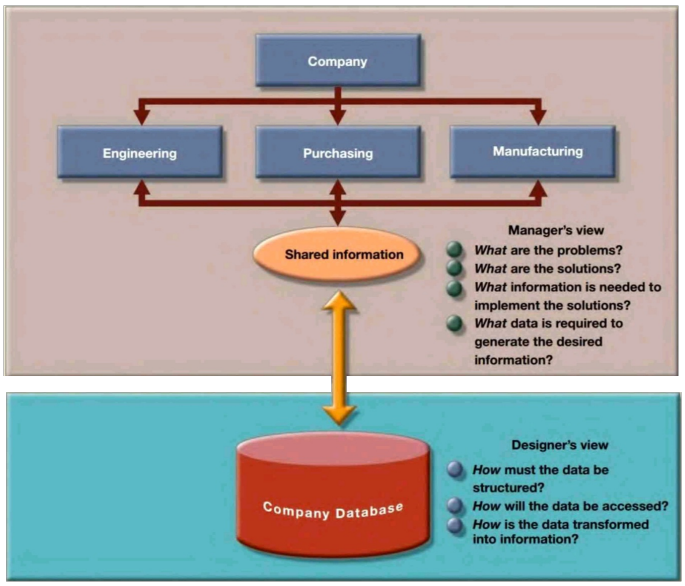


Figure 10.1: Two views of data: business manager and designer

Conceptual Design

In the conceptual design stage, data modelling is used to create an abstract database structure that represents real-world objects in the most realistic way possible. The conceptual model must embody a clear understanding of the business and its functional areas. At this level of abstraction, the type of hardware and/or database model to be used might not yet have been identified. Therefore, the design must be software and hardware independent, so the system can be set up within any hardware and software platform chosen later.

Minimal Data Rule

All that is needed is there, and all that is there is needed.

All data elements required by the database transactions must be defined in the model, and all data elements defined in the model must be used by at least one database transaction.

As you apply the minimal data rule, avoid an excessive short-term bias. Focus not only on the immediate data needs of the business, but also on the future data needs. Thus, the database design must leave room for future modifications and additions.

Steps in Conceptual Design

- Data analysis and requirements
- Entity relationship modelling and normalisation
- Data model verification
- Distributed database design

DBMS Software Selection

The selection of DBMS software is critical to the information system's smooth operation. Therefore, the advantages and disadvantages of the proposed DBMS software should be carefully studied. To avoid false expectations, the end user must be made aware of the limitations of both the DBMS and the database.

Factors Affecting DBMS Software Selection

The factors to consider are:

- **Costs** Purchase, maintenance, operational, licence, installation, training, and conversion costs.
- **DBMS features and tools** Some database software includes a variety of tools that facilitate the application development task. For example, the availability of query by example, screen painters, report generators, application generators, data dictionaries, and so on, helps create a more pleasant work environment for both the end user and the application programmer. Database administrator facilities, query facilities, ease of use, performance, security, concurrency control, transaction processing, and third-party support also influence DBMS software selection.
- **Underlying model** Hierarchical, network, relational, object/relational, or object-oriented.
- **Portability** Across platforms, systems, and languages.
- **DBMS hardware requirements** Processor(s), RAM, disk space, and so on.

Logical Design

The second stage in the database design cycle is known as **logical design**. The aim of the logical design stage is to map the conceptual model into a logical model that can then be implemented on a relational DBMS.

Logical Design Steps

1. Creating the logical data model
2. Validating the logical data model using normalisation
3. Assigning and validating integrity constraints
4. Managing logical models constructed for different parts of the database
5. Reviewing the logical model with the user

The right to use the database is also specified during the logical design phase.

The logical design translates the software-independent conceptual model into a software-dependent model by defining the appropriate domain definitions, the required tables, and the necessary access restrictions.

Physical Design

Physical design is the process of selecting the data storage and data access characteristics of the database. The storage characteristics are a function of the types of devices supported by the hardware, the type of data access methods supported by the system, and the DBMS. Physical design affects not only the location of the data in the storage device(s), but also the performance of the system.

Stages of Physical Design

1. Analyse data volume and database usage
2. Translate each relation identified in the logical data model into tables
3. Determine a suitable file organisation
4. Define indexes
5. Define user views
6. Estimate data storage requirements
7. Determine database security for users

Physical design is a very technical job, more typical of the client/server and mainframe world than of the desktop world. Yet even in the more complex mid-range and mainframe environments, modern database software has assumed much of the burden of the physical portion of the design and its implementation.

In spite of the fact that relational models tend to hide the complexities of the computer's physical characteristics, the performance of relational databases *is* affected by physical-level characteristics. Performance can be affected by the characteristics of the storage media, such as seek time, sector and block (page) size, buffer pool size and number of disk platters and read/write heads. In addition, factors such as the creation of an index can have a considerable effect on the relational database's performance, that is, data access speed and efficiency.

Even the type of data request must be analysed carefully to determine the optimum access method for meeting the application requirements, establishing the data volume to be stored and estimating the performance. Some DBMSs automatically reserve the space required to store the database definition and the user's data in permanent storage devices. This ensures that the data are stored in sequentially adjacent locations, thereby reducing data access time and increasing system performance.

Physical design becomes more complex when data are distributed at different locations, because the performance is affected by the communication media's throughput. Given such complexities, designers favour software that hides as many of the physical-level activities as possible.

Logical and physical design can be carried out in parallel, on a table-by-table (or file-by-file) basis. Logical and physical design can also be carried out in parallel when the designer is working with hierarchical and network models. Such parallel activities require the designer to have a thorough understanding of both software and hardware characteristics.

10.3.3 Implementation and Loading

The output of the design phase is a series of instructions detailing the creation of tables, attributes, domains, views, indexes, security constraints, and storage and performance guidelines. In this phase, you actually implement all of these design specifications.

Install the DBMS

This step is required only when a new dedicated instance of the DBMS is necessary for the system. In many cases, the organisation will have made a particular DBMS the standard to leverage investments in technology and the skills that employees have already developed. The DBMS may be installed on a new server or on existing servers. One current trend is called **virtualisation**.

Virtualisation

A technique that creates logical representations of computing resources that are independent of the underlying physical computing resources.

The technique is used in many areas of computing, such as the creating of virtual services, virtual storage, and virtual private networks.

In a database environment, **database virtualisation** refers to the installation of a new instance of the DBMS on a virtual server running on shared hardware. This is normally a task that involves system and network administrators to create appropriate user groups and services in the server configuration and networks routing.

Another common trend is the use of cloud database services such as Microsoft SQL Database Service or Amazon Relational Database Service. This new generation of services allows users to create databases that can be easily managed, tested, and scaled up as needed.

Create the Database(s)

In most modern relational DBMSs, a new database implementation requires the creation of special storage-related constructs to house the end-user tables. The constructs usually include the storage group (or file groups), the table spaces, and the tables. A storage group can contain more than one table space, and a table space can contain more than one table.

Load or Convert the Data

After the database has been created, the data must be loaded into the database tables. Typically, the data will have to be migrated from the previous version of the system. Often, data to be included in the system must be aggregated from multiple sources. In the best case scenario, all the data will be in a relational database so that it can be readily transferred to a new database. However, in some cases, data may have to be imported from other relational databases, non-relational databases, flat files, legacy systems, or even manual paper-and-pencil systems. If the data format does not support direct importing into the new database, conversion programs may have to be created to reformat the data for importing. In the worst-case scenario, much of the data will have to be manually entered into the database. Once the data has been loaded, the database administrator (DBA) works with the application developers to test and evaluate the database.

Loading existing data into a cloud-based database service can sometimes be expensive. The reason for this is that most cloud services are priced based not only on the volume of the data to be stored, but also on the amount of data that travels over the network.

Example: Implementation of the Logical Design of IBM's DB2

1. The system administrator would create the database storage group. This step is mandatory for such mainframes as DB2. Other DBMS software may create equivalent storage groups automatically when a database is created.
2. The system administrator creates the database within the storage group.
3. The system administrator assigns the rights to use the database to a database administrator.
4. The database administrator creates the table space(s) within the database.
5. The database administrator creates the table(s) within the table space(s).
6. The database administrator assigns access rights to the table spaces and to the tables within specified table spaces. Access rights may be limited to views rather than to whole tables. The creation of views is not required for database access in the relational environment, but views are desirable from a security standpoint.

10.3.4 Database Security

Data stored in the company database must be protected from access by unauthorised users. Any misuse or damage to the data may have a serious impact on the organisation. The most common security goals relate to the integrity, confidentiality, and availability of data. Within database design, it is essential that security measures are developed to meet the security goals, and in doing so, protect the data from any kind of threat. **Threats** are any set of circumstances that have the potential to cause loss, misuse or harm to the system and/or its data.

Threats can include:

- The loss of integrity of data through unauthorised modification.
- The loss of availability of the data.
- The loss of confidentiality of the data.

Threats can occur internally and externally to an organisation, and are of various levels of severity.

Example: Types of Threats and their Effects

Some types of threats and their effects include:

- **Theft and Fraud of Data** Activities such as these are likely to be perpetrated by humans, often by electronic means. Both threat and fraud can occur both inside and outside the organisation and each has to be treated differently.
- **Human error that causes accidental loss of data** This is often caused by humans not following policies and procedures such as user authorisation. However, it is important for an organisation to ensure that it has excellent security policies and procedures in place to begin with. Additionally, data can be lost by poor staff training. If employees do not know the procedures surrounding data security then it will be impossible for them to be followed.

- **Electronic infections** There are four general categories of general infections.
  - **Viruses** A malicious piece of software that is capable of copying itself and spreading across a network. As viruses are usually attached to a program or application, they cannot be 'caught' without human intervention.
  - **Email viruses** This type of virus attaches itself to email messages and replicates itself by automatically mailing itself to all people in the receiver's email address book.
  - **Worms** are also small pieces of software that replicate themselves using any form of telecommunications network or hole in security. They are different from viruses in that they travel between systems without any human intervention, and they can replicate themselves very quickly between networks.
  - **Trojan horses** A computer program that claims to perform one task or action. It remains dormant until run and then begins to do damage such as erase a hard disk.

  The introduction of a virus to a computer network can result in both the loss of integrity of the data and the loss of availability of the system resources, resulting in serious consequences to the business.
- **Natural disasters** such as storms, fires, or floods. These are unpredictable, and not deliberate actions, but would still result in the loss of integrity and availability of data. In addition, data could be corrupted due to power surges, and hardware would become physically damaged.
- **Unauthorised access and modification of data** The phrase often used for gaining unauthorised access is **hacking**. **Hacking** is usually defined as the act of illegally entering a computer system, and making unauthorised changes to files and data contained within. Obtaining unauthorised access to a database may involve a person browsing unauthorised data to gain information that could be used to that person's benefit, or against the organisation. Unauthorised modification could result in the data being changed, or even deleted.
- **Employee sabotage** is concerned with deliberate acts of malice against the organisation. This would include not only any computer system, but also the property, reputation, and safety of a business and its employees. Unauthorised access and modification of data, physically damaging hardware, and theft of data are also covered by this threat.
- **Poor database administration** This could be caused by the database administrator not having enough knowledge through lack of training. One example is the DBA granting excessive privileges to a user who exceeds the requirements of his or her job within the organisation. The user then goes on to abuse these privileges. Another example would be that the DBA has only set up weak authentication schemes, which allow attackers to steal or obtain login information, and then assume the identity of genuine database users.

The above list of threats is not exhaustive. However, it does highlight the need for an organisation to have a comprehensive data security plan. The plan should contain a number of data security measures to protect both the data and the hardware. The DBMS is only part of the computer system infrastructure within an organisation, and will often rely on the security measures used in other parts of the system.

Data Security Measures

Physical security allows only authorised personnel access to specific areas. Depending on the type of database implementation, however, establishing physical security may not always be practical. Examples of impractical candidates for physical security include a university student research database, and large multi-server microcomputer networks. In terms of guarding against the loss of data and hardware due to a natural disaster, the placement of the hardware in a building could be carefully considered. For example, do not place the hardware in the basement, due to the possibility of floods. Physical access to rooms can be controlled by push-button security controls, swipe cards, or biometric systems.

User authentication is a way of identifying the user, and verifying that the user is allowed to access restricted data or applications. This can be achieved through the use of passwords and access rights.

- **Password Security** allows the assignment of access rights to specific authorised users. Password security is usually enforced at logon time at the operating system level.
- **Access Rights** can be established through the use of database software. The assignment of access rights may restrict operations (CREATE, UPDATE, DELETE) on predetermined objects such as databases, tables, views, queries, and reports.

User authentication is a function of authorisation management, which is part of the DBA's managerial role.

Audit trails are usually provided by the DBMS to check for access violations. Although the audit trail is an after-the-face device, its mere existence can discourage unauthorised use. Audit trails represent the last line of the database defence. Although it would be preferable for the security measures to work, and for an attacker to not gain access to the system, if all else fails, the audit data itself can identify the existence of a violation or unauthorised access after it has occurred. The audit data may then be used to link a violation to a particular user, and may be used to repair the system.

Data encryption can be used to render data useless to unauthorised users who might have violated some of the database security layers or security measures. Data encryption is carried out by an algorithm.

Example: Data Encryption

Suppose a bank wants to encrypt the account numbers of its customers. The first stage would be to alter the code by a secret one-digit number, for example 5. If a person's account number is 32451, then the encrypted value would be 32456. The real value can then be decrypted from the encrypted value by subtracting 5.

The logic of adding a specific number to the real data is known as the **encryption algorithm**. Here, the value 5, which is added by the algorithm, is known as the **encryption key**.

Where only one method is used, the method is referred to as the **one-key** method, or the **data encryption standard (DES)**. Both the sender and the receiver would need to know the key in order to decipher the stored data. With the one-key method, an intruder would need up to ten guesses, whereas for a *two-key* method, up to 100 guesses would be needed. Therefore, the longer the key, the more difficult it is to decipher the data.

In the two-key method, all users who wish to send data have a public key. The encryption algorithm uses this public key to transform the data in the message into an encrypted message. The second key, known as the private key, is used by the encryption algorithm to convert the encrypted message back

to the data in the message. The only person who may hold the private key is the one for whom the original message was destined.

Some DBMS products include encryption routines. For example, Oracle DBMS has a feature known as Transparent Data Encryption (TDE) which allows for columns in a database table to be easily encrypted without the need for writing lots of complex code. When users insert data, the database transparently encrypts and stores it in the column. Similarly, when users select the column, the database automatically decrypts it.

Encryption in the Real World

The most common example of encryption at work is **Secure Electronic Transactions (SET)**. This is an open protocol designed by a large consortium of companies interested in ensuring data privacy in all electronic commerce over the Internet. SET ensures the authenticity of electronic transactions and provides a guarantee that customer's transactions are protected.

A combination of private and public key encryption is used in **Secure Sockets Layer (SSL)** technology on the Internet. SSLs create a secure connection between a user and an external server, over which any amount of data can be sent securely. The use of SSL can be seen when a person purchases goods from an Internet-based store. This is normally indicated by the use of 'https' instead of 'http' before the web address.

User-defined policies and procedures should be put in place by the organisation to ensure that employees know how to implement the data security measures. Such policies and procedures can cover personal controls such as training employees in security aspects and monitoring employees to ensure that they are actually following the procedures themselves. The establishment of policies and procedures is also a responsibility of the DBA.

Backup and recovery strategies should be in place in the event of a disaster occurring. The responsibility ultimately lies with the DBA to ensure the data within the database can always be fully recovered.

Antivirus software is used by organisations to search system hard drives and media devices for any known or potential viruses. Each time a virus is discovered, antivirus software vendors record the virus' unique signature, and then incorporate it into their software database. The antivirus software will check, in real time, all messages entering an organisation's network from any external source, to see if a known virus is trying to enter. This feature is only useful if kept up to date.

Firewalls are systems comprising hardware devices or software applications which act as gatekeepers to an organisation's network. They are used to prevent unauthorised access by allowing you to establish a set of rules or filters to determine which messages should be allowed in or out of an organisation's network. They are most commonly used when an organisation's database can be accessed by Web applications. If a message is flagged as breaking the rules, it is not allowed through.

- **Packet filtering** Each message or packet that contains data is checked against a set of filters. Packets that are accepted are allowed to be sent to the designated system, and all others are discarded.
- **Proxy server** The proxy server manages all communication between the internal network of an organisation and external networks such as the Internet. There are further advantages to using a proxy server, other than security measures. It can also cache the Web pages that have been requested, so that network traffic is reduced if other users request the same page. This also increased response time. In addition, the proxy server can also be used to limit the websites that users may view outside the organisation.

- **Circuit-level gateway** This blocks all incoming messages to any host but itself. Within the organisation, all the client machines will run software to allow them to establish a connection with the circuit-level gateway machine. The proxy server performs all communication with any external network, such as the Internet, so the internal client machines never actually have any contact with the 'outside world'.
- **Diskless workstations** These allow end-users to access the database without being able to download the information from their workstations.

Desirable Attributes

Data are: • Protected • Reconstructable • Auditable • Tamper-proof

Users are: • Identifiable • Authorised • Monitored

10.3.5 Testing and Evaluation

In the design phase, decisions were made to ensure integrity, security, performance and recoverability of the database. During implementation and loading, these plans were put into place. In testing and evaluation, the DBA tests and fine-tunes the database to ensure that it performs as expected. This phase occurs in conjunction with application programming. Programmers use database tools to prototype the applications during coding of the programs. Tools such as report generators, screen painters, and menu generators are especially useful to application programmers.

Test the Database

During this step, the DBA tests the database to ensure that it maintains the integrity and security of the data. Data integrity is enforced by the DBMS through the proper use of primary and foreign key rules. Many DBMSs also support the creation of domain constraints and database triggers. Testing will ensure that these constraints are properly designed and implemented. Data integrity is also the result of properly implemented data management policies, which are part of a comprehensive data administration framework.

Evaluate the Database and its Application Programs

As the database and application programs are created and tested, the system must also be evaluated using a more holistic approach. Testing and evaluation of the individual components should culminate in a variety of broader system tests to ensure that all the components interact properly to meet the needs of the users. Integration issues and deployment plans are refined, user training is conducted, and system documentation is finalised. Once the system receives final approval, it must be a sustainable resource for the organisation. To ensure that the data contained in the databases are protected against loss, backup and recovery plans are tested.

Databases can lose data through unintentional deletions, power outages, and other causes. Data backup and recovery procedures create a safety valve, ensuring the availability of consistent data. Database vendors encourage the use of fault-tolerant components such as uninterruptible power supply (UPS) units, RAID storage devices, clustered servers, and data replication technologies, to ensure the continuous operation of the database in case of a hardware failure.

Database Backup Levels

1. A full backup, or dump, of the entire database. All database objects are backed up in their entirety.
2. A differential backup of the database, in which only the objects that have been updated or modified since the last full backup are backed up.
3. A transaction log backup, which backs up only the transaction log operations that are not reflected in a previous backup copy of the database. In this case, no other database objects are backed up.

The database backup is stored in a secure place, usually in a different building from the database itself, and is protected against dangers such as fire, theft, flood, and other potential calamities. The main purpose of the backup is to guarantee database restoration following a hardware or software failure. Depending on the type and extent of the failure, the recovery process ranges from a minor short-term inconvenience to a major long-term rebuild. Regardless of the extent of the required recovery process, recovery is not possible without a usable backup.

Database recovery generally follows a predictable scenario. First, the type and extent of the required recovery are determined. If the entire database needs to be recovered to a consistent state, the recovery uses the most recent backup copy of the database in a known consistent state. The backup copy is then rolled forward to restore all subsequent transactions by using the transaction log information. If the database needs to be recovered, but the committed portion of the database is still usable, the recovery process uses the transaction log to 'undo' all the transactions that were not committed. At the end of this phase, the database completes an iterative process of testing, evaluation and modification, that continues until the system is certified as ready to enter the operational phase.

Common Sources of Database Failure

Source	Description
Software	Software-induced failures may be due to the operating system, application programs, or viruses and other malware.
Hardware	Hardware-induced failures may include memory chip errors, disk crashes, bad disk sectors, and disk-full errors.
Programming exemptions	Application programs or end-users may roll back transactions when certain conditions are defined. Programming exemptions can also be caused by malicious or improperly tested code that can be exploited by hackers.
Transactions	The system detects deadlocks and aborts one of the transactions.
External factors	Backups are especially important when a system suffers complete destruction from a fire, earthquake, flood, or other natural disaster.

10.3.6 Operation

Once the database has passed the evaluation stage, it is considered operational. At this point, the database, its management, its users, and its application programs constitute a complete information system.

The beginning of the operational phase starts the process of system evolution. As soon as all the targeted end users have entered the operations phase, problems that could not have been foreseen during the testing phase begin to surface. These problems range from serious to minor issues. Either way, this leads to a demand for change, leading to the next phase: maintenance and evolution.

10.3.7 Maintenance and Evolution

Routine Maintenance Activities

- Preventive maintenance (backup)
- Corrective maintenance (recovery)
- Adaptive maintenance (enhancing performance, adding entities and attributes)
- Assignment of access permissions and their maintenance for new and old users
- Generation of database access statistics to improve the efficiency and usefulness of system audits and to monitor system performance.
- Periodic security audits based on the system-generated statistics
- Periodic system-usage summaries for internal billing or budgeting purposes.

The likelihood of new information requirements and the demand for additional reports and new query formats require application changes and possible minor changes in the database components and contents. Those changes can be easily implemented only when the database design is flexible, and when all documentation is updated and online. Eventually, even the best-designed database environment will no longer be capable of incorporating such evolutionary changes; then the whole DBLC process begins anew.

10.3.8 Determine Performance Measures

Physical design becomes more complex when data is distributed at different locations because the performance is affected by the communication media's throughput. Designers favour database software that hides as many of the physical-level activities as possible. Despite the fact that relational models tend to hide the complexities of the computer's physical characteristics, the performance of relational databases is affected by physical storage properties. Performance can be affected by characteristics of the storage media, such as seek time, sector and block (page) size, buffer pool size, and the number of disk platters and read/write heads. In addition, factors such as the creation of an index can have a considerable effect on the relational database's performance. **Physical design performance measurement** deals with fine-tuning the DBMS and queries to ensure they will meet end-user performance requirements.

10.4 Database Design Strategies

There are two classical approaches to database design:

- **Top-down design** starts by identifying the data sets, then defines the data elements for each of those sets. This process involves the identification of different entity types and the definition of each entity's attributes.
- **Bottom-up design** first identifies the data elements (items), then groups them together in data sets. In other words, it first defines attributes, then groups them to form entities.

The selection of a primary emphasis on top-down or bottom-up procedures often depends on the scope of the problem, or on personal preference. Although the two methodologies are complementary rather than mutually exclusive, a primary emphasis on a bottom-up approach may be more productive for small databases, with few entities, attributes, relations, and transactions. For situations in which the number, variety and complexity of entities, relations and transactions is overwhelming, a primarily top-down approach may be more easily managed.

Normalisation and ER models

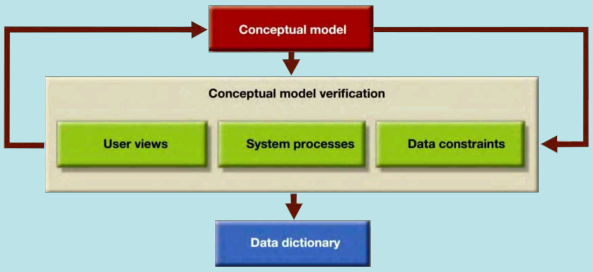
Even when a primarily top-down approach is selected, the normalisation process that revises existing tables is a bottom-up technique. ER models constitute a top-down process even when the selection of attributes and entities can be described as bottom-up. Both normalisation and the ER model form the basis for most designs.

10.5 Centralised vs Decentralised Design

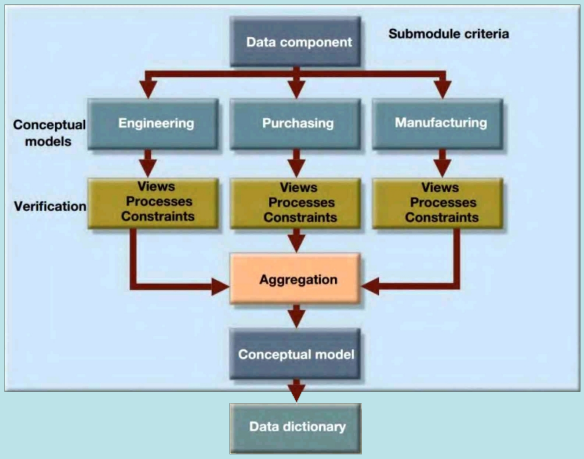
The two approaches discusses above can be influenced by factors such as the scope and size of the system, the company's management style, and the company's structure. Depending on the above factors, database design might focus on either a centralised or decentralised design philosophy.

Centralised Design

Centralised design is productive when the data component is composed of a relatively small number of objects and procedures. The design can be carried out and represented in a fairly simple database. Centralised design is typical of relatively simple and/or small databases and can be successfully done by a single person or a small, informal design team. The company operations and the scope of the problem are sufficiently limited to allow even a single designer to define the problem(s), create the conceptual design, verify the conceptual design with the user views, define system processes and data constraints to ensure the efficacy of the design, and ensure that the design will comply with all the requirements. Although centralised design is typical of small companies, it is not limited to small companies. A single conceptual design is completed and then validated in the centralised design approach.



Decentralised Design



Decentralised design might be used when the data component of the system has a considerable number of entities and complex relations on which very complex operations are performed. Decentralised design is also likely to be employed when the problem itself is spread across several operational sites, and each element is a subset of the entire data set. In large and complex projects, the database design typically cannot be done by only one person. Instead, a carefully selected team of database designers is employed to tackle a complex database project. Within the decentralised design framework, the database design task is divided into several modules. Once the design criteria have been established, the lead designer assigns design subsets or modules to design groups within the team.

As each design group focuses on modelling a subset of the system, the definition of boundaries and the interrelation among data subsets must be very precise. Each design group creates a conceptual data modelling corresponding to the subset being modelled. Each conceptual model is then verified individually against the user views, processes, and constraints for each of the modules. After the verification process has been completed, all modules are integrated into one conceptual model. Because the data dictionary describes the characteristics of all objects within the conceptual data model, it plays a vital role in the integration process. After all the subsets have been aggregated into a larger conceptual model, the lead designer must verify that the combined conceptual model is still able to support all the required transactions.

Aggregation Problems

The aggregation process requires the designer to create a single model in which various aggregation problems must be addressed:

- **Synonyms and homonyms** Different departments might know the same object by different names (**synonyms**), or they might use the same name to address different objects (**homonyms**). The object can be an entity, an attribute, or a relationship.
- **Entity and entity subtypes** An entity subtype might be viewed as a separate entity by one or more departments. The designer must integrate such subtypes into a higher-level entity.
- **Conflicting object definitions** Attributes can be recorded as different types, or different domains can be defined for the same attribute. Constraint definitions can also vary. The designer must remove such conflicts from the model.

10.6 Database Administration

The person responsible for the control of the centralised and shared database is the **database administrator (DBA)**. The size and role of the DBA function varies from company to company, as does its placement within a company's organisation structure. On the organisation chart, the DBA function might be defined as either a staff or line position. Placing the DBA function in a staff position often creates a consulting environment in which the DBA is able to device the data administration strategy, but does not have the ability to enforce it or resolve possible conflicts. The DBA function in a line position has both the responsibility and the authority to plan, define, implement, and enforce the policies, standards, and procedures used in the data administration activity. There is no standard for how the DBA function fits in an organisation's structure. In part, that is because the DBA function itself is probably the most dynamic of any organisation's functions.

The fast-paced changes in DBMS technology dictate changing organisational styles. For example:

- The development of distributed databases can force an organisation to decentralise the data administration further. The distributed database requires the system DBA to define and delegate the responsibilities of each local DBA, thus imposing new and more complex *coordinating* activities on the system DBA.
- The growing use of Internet-ready and object-oriented databases and the growing number of data warehouse applications are likely to add to the DBA's data modelling and design activities, thus expanding and diversifying the DBA's job.
- The increasing sophistication and power of desktop-based DBMS packages provided an easy platform for the development of user-friendly, cost-effective and efficient solutions to specific departmental information needs. But such an environment also invites data duplication, not to mention the problems created by people who lack the technical qualifications to produce good database designs. In short, the new desktop environment requires the DBA to develop a new set of technical and managerial skills.

Although no current standard exists, it is common practice to define the DBA function by dividing the DBA operations according to the DBLC phases. If that approach is used, the DBA function requires personnel to cover the following activities:

- Database planning, including the definition of standards, procedures, and enforcement.
- Database requirements gathering and conceptual design
- Database logical design and transaction design
- Database physical design and implementation
- Database testing and debugging
- Database operations and maintenance, including installation, conversion, and migration
- Database training and support

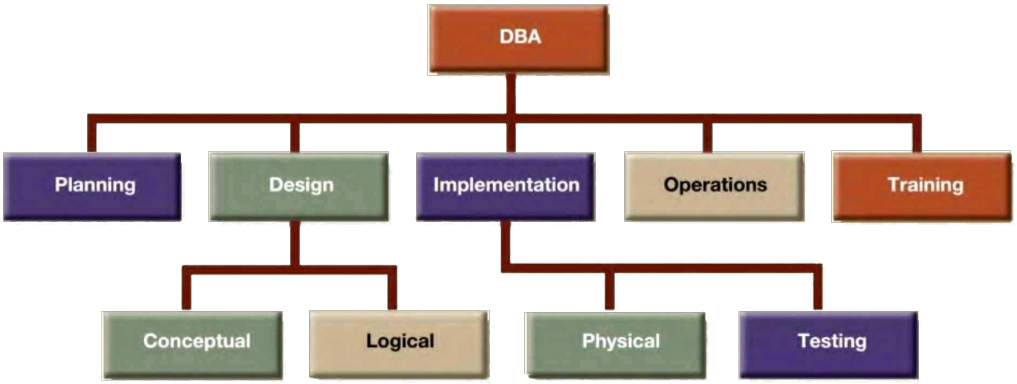


Figure 10.2: A DBA Functional Organisation

A company might have several different and incompatible DBMSs installed to support different operations. For example, it is not uncommon to find corporations with a hierarchical DBMS to support the daily transactions at an operational level, and a relational database to support middle and top management's ad hoc information needs. There may also be a variety of desktop DBMSs installed in the different departments. In such an environment, the company might have one DBA assigned for each DBMS. The general coordinator of all DBAs is sometimes known as the **systems administrator (SYSADM)**.

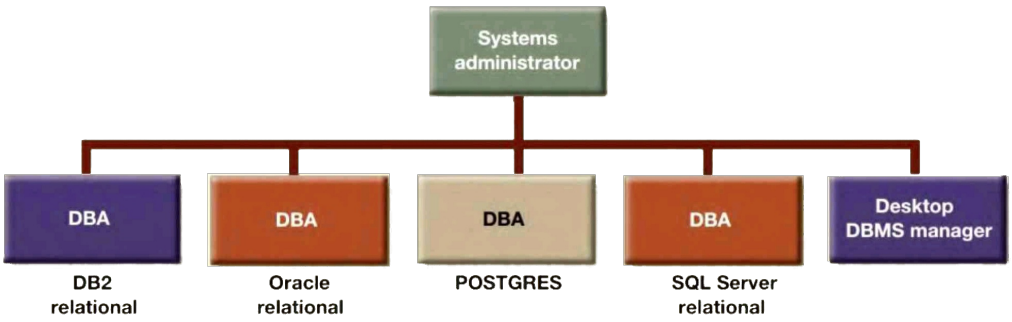


Figure 10.3: Multiple Database Administrators in an Organisation

There is a growing trend towards specialisation in the data management function. For example, the organisation charts used by some of the bigger corporations make a distinction between a DBA and a **data administrator (DA)**. The DA, also known as the **information resource manager (IRM)**, usually reports directly to top management and is given a higher degree of responsibility and authority than the DBA, although the two roles tend to overlap to some extent.

The DA is responsible for controlling the overall corporate data resources, both computerised and manual. Thus, the DA's job description covers a larger area of operations than that of the DBA because the DA is in charge of controlling not only the computerised data, but also the data outside the scope of the DBMS. The placement of the DBA within the expanded organisational structure may vary from company to company. Depending on the structure's components, the DBA might report to the DA, the IRM, the IS manager, or directly to the company's CEO.

There are two distinct roles that the DBA must perform: the managerial role, and the technical role. The DBA's **managerial role** is focused on personnel management and on interactions with the end-user community. The DBA's **technical role** involves the use of the DBMS – database design, development, and implementation – as well as the production, development, and use of application programs.

Desired DBA Skills

Managerial	Technical
Broad business understanding	Broad data-processing background
Coordination skills	Systems development life cycle knowledge
Analytical skills	Structured methodologies: • Data flow diagrams • Structure charts • Programming languages
Communication skills (oral and written)	Database modelling and design skills: • Conceptual • Logical • Physical
Negotiation skills	Operation skills: • Database implementation • Data dictionary management • Security • etc.

10.6.1 The Managerial Role of the DBA

As a manager, the DBA must concentrate on the control and planning dimensions of database administration. Therefore, the DBA is responsible for:

- Coordinating, monitoring and allocating database administration resources: people and data.
- Defining goals and formulating strategic plans for the database administration function.

DBA Activities and Services

DBA Activity	DBA Service
Planning	End-user support
Organising	Policies, procedures, and standards
Testing	Data security, privacy, and integrity
Monitoring	Data backup and recovery
Delivering	Data distribution and use

End-User Support

The DBA interacts with the end-user by providing data and information support services to the organisation's departments. Because end-users usually have dissimilar backgrounds, end-user support services usually include:

- **Gathering user requirements** The DBA must work with the end-user community to help gather the data required to identify and describe the end-user's problems. The DBA's communication skills are very important at this stage, because the DBA works closely with people who tend to have different computer backgrounds and communication styles. The gathering of user requirements requires the DBA to develop a precise understanding of the user's views and needs, and to identify present and future information needs.
- **Building end-user confidence** Finding adequate solutions to end-user's problems increases end-user trust and confidence in the DBA function.
- **Resolving conflicts and problems** Finding solutions to end-user's problems in one department might trigger conflicts with other departments. End-users are typically concerned with their own specific data needs rather than those of others, and they are not likely to consider how their data affect other departments within the organisation. When data/information conflicts arise, the DBA function has the responsibility and authority to resolve them.
- **Finding solutions to information needs** The ability and authority to resolve data conflicts enable the DBA to develop solutions that will properly fit within the existing data management framework. The DBA's primary objective is to provide solutions to the end-user's information needs. Given the growing importance of the Internet, those solutions are likely to require the development and management of Web browsers to interface with the databases. In fact, the explosive growth of e-commerce requires the use of *dynamic* interfaces to facilitate interactive product queries and product sales.
- **Ensuring quality and integrity of applications and data** Once the right solution has been found, it must be properly implemented and used. Therefore, the DBA must work with both application programmers and end-users to teach them the database standards and procedures required for data access and manipulation. The DBA must also make sure that the database transactions do not adversely affect the database's data quality. Certifying the quality of the application programs that access the database is a crucial DBA function. Special attention must be given to the DBMS Internet interfaces, because those interfaces do not provide the transaction management features that are typically found in the DBMS-managed database environment.
- **Managing the training and support of DBMS users** One of the most time-consuming DBA activities is teaching end-users how to use the database properly. The DBA must ensure that all users accessing the database have a basic understanding of the functions and use of the DBMS software. The DBA coordinates and monitors all activities concerning end-user education.

Policies, Standard, and Procedures

A prime component of a successful data administration strategy is the continuous enforcement of the policies, procedures, and standards for correct data creation, usage, distribution, and deletion within the database. The DBA must define, document, and communicate the policies, procedures, and standards, before they can be enforced.

Policies

General statements of direction or action that communicate and support DBA goals.

Example

- All users must have passwords.
- Passwords must be changed every six months.

Standards

More detailed and specific than policies, and describe the minimum requirements of a given DBA activity. In effect, standards are rules that are used to evaluate the quality of the activity. For example, standards define the structure of application programs, and the naming conventions programmers must use.

Example

- A password must have a minimum of five characters.
- A password may have a maximum of 12 characters.
- ID numbers, names, and birth-dates cannot be used as passwords.

Procedures

Written instructions that describe a series of steps to be followed during the performance of a given activity. Procedures must be developed within existing working conditions, and they must support and enhance that environment.

Example To create a password,

1. the end-user sends the DBA a written request for the creation of an account;
2. the DBA approves the request and forwards it to the computer operator;
3. the computer operator creates the account, assigns a temporary password, and sends the account information to the end-user;
4. a copy of the account information is sent to the DBA;
5. the user changes the temporary password to a permanent one.

Standards and procedures defined by the DBA are used by all end-users who want to benefit from the database. Standards and procedures must complement each other, and must constitute an extension of data administration policies. Procedures must facilitate the work of end users and the DBA.

The DBA must define, communicate and enforce procedures that cover areas such as:

- **End-user database requirements gathering** Which documentation is required? Which forms should be used?
- **Database design and modelling** Which database design methodology to use; which tools to use.
- **Documentation and naming conventions** Which documentation to use in the definition of all data elements, sets, and programs that access the database.
- **Design, coding, and testing of database application programs** The DBA must define the standards for application program coding, documentation, and testing. The DBA standards and procedures are given to the application programmers, and the DBA must enforce those standards.
- **Database software selection** The selection of the DBMS package and any other software related to the database must be properly managed. For example, the DBA might require that software be properly interfaced with existing software, that it has the features required by the organisation, and that it provides a positive return on investment. In today's Internet environment, the DBA must also work with Web administrators to find proper Web-to-database connectivity solutions.
- **Database security and integrity** The DBA must define the policies governing security and integrity. Database security is especially crucial. Security standards must be clearly defined and strictly enforced. Security procedures must be designed to handle a multitude of security scenarios to ensure that security problems are minimised. Although no system can ever be completely secure, security procedures must be designed to meet critical standards. The growing use of internet interfaces to databases opens the door to new security threats that are far more complex and difficult to manage than those encountered with more traditional internally generated and controlled interfaces. Therefore, the DBA must work closely with Internet security specialists to ensure that the databases are properly protected from attacks launched inadvertently, or attacks launched deliberately by unauthorised users.
- **Database backup and recovery** Database backup and recovery policies must include the information necessary to guarantee proper execution and management of the backups.
- **Database maintenance and operation** The DBMS's daily operations must be clearly documented. Operators must keep job logs, and they must write operator instructions and notes. Such notes are helpful in pinpointing the causes and solutions of the problems. Operational procedures must also include precise information concerning backup and recovery procedures.
- **End-user training** A full-featured training program must be established within the organisation, and procedures governing the training must be clearly specified. The objective is to indicate clearly who does what, when, and how. Each end-user must be aware of the type and extent of the available training methodology.

Procedures and standards must be revised at least annually to keep them up to date and to ensure that the organisation can adapt quickly to changes in the work environment. The introduction of new DBMS software, the discovery of security or integrity violations, the reorganisation of the company, and similar changes, require revision of the procedures and standards.

Data Security, Privacy, and Integrity

The security, privacy, and integrity of the data in the database are of great concern to DBAs who manage current DBMS installations. Technology has pointed the way to greater productivity through information management. Technology has also resulted in the distribution of data across multiple sites, thus making it more difficult to maintain data control, security, and integrity. The multiple-site data configuration has made it imperative that the DBA use the security and integrity mechanisms provided by the DBMS to enforce the database administration policies. In addition, DBAs must team up with security experts to build firewalls, proxy services, and other security mechanisms to safeguard data from possible attacks.

Protecting the security and privacy of the data in the database is a function of authorisation management. **Authorisation management** defines procedures to protect and guarantee database security and integrity. Those procedures include, but are not limited to, user access management, view definition, DBMS access control and DBMS usage monitoring.

- **User access management** This function is designed to limit access to the database and likely includes at least the following procedures:
  - **Define each user to the database** This is achieved at two levels: at the operating system level and at the DBMS level. At the OS level, the DBA can request the creation of a logon user ID that allows the end user to log on to the computer system. At the DBMS level, the DBA can either create a different user ID or employ the same user ID to authorise end-user access to the DBMS.
  - **Assign passwords to each user** This can also be done at the OS and DBMS level. The database passwords can be assigned with predetermined expiration dates. The use of expiration dates enables the DBA to screen end-users periodically, and to remind users to change their passwords periodically, thus making unauthorised access less probable.
  - **Define user groups** Classifying users into user groups according to common access needs facilitates the DBA's job of controlling and managing the access privileges of individual users.
  - **Assign access privileges** The DBA assigns access privileges or access rights to specific users to access specified databases. An access privilege describes the type of authorised access. Access privileges in relational databases are assigned through `SQL GRANT` and `REVOKE` commands.
  - **Control physical access** Physical security can prevent unauthorised users from directly accessing the DBMS installation and facilities. Some common physical security practices found in large database installations include secured entrances, password-protected workstations, electronic personnel badges, closed-circuit video, voice recognition, and biometric technology.
- **View definition** The DBA must define data views to protect and control the scope of the data that are accessible by an authorised user. The DBMS must provide the tools that allow the definition of views that are composed of one or more tables and the assignment of access rights to a user or a group of users. The SQL command `CREATE VIEW` is used in relational databases to define views.
- **DBMS access control** Database access can be controlled by placing limits on the use of the DBMS's query and reporting tools. The DBA must make sure that those tools are used properly, and only by authorised personnel.

- **DBMS usage monitoring** The DBA must also audit the use of the data in the database. Several DBMS packages contain features that allow the creation of an **audit log**, which automatically records a brief description of the database operations performed by all users. Such audit trails enable the DBA to pinpoint access violations. The audit trails can be tailored to record all database accesses, or just failed database accesses.

Security breaches can yield a database whose integrity is either preserved or corrupted:

- **Preserved:** Action is required to avoid the repetition of similar security problems, but data recovery may not be necessary. As a matter of fact, most security violations are produced by unauthorised and unnoticed access for information purposes, but such snooping does not disrupt the database.
- **Corrupted:** Action is required to avoid the repetition of similar security problems, and the database must be recovered to a consistent state. Corrupting security breaches include database access by computer viruses and by hackers whose actions are designed to alter or destroy data.

The integrity of a database might be lost because of external factors beyond the DBA's control. Whatever the reason, the possibility of data corruption or destruction makes backup and recovery procedures crucial to any DBA.

Data Backup and Recovery

When data are not readily available, companies face potentially ruinous losses. Therefore, data backup and recovery procedures are critical in all database installations. The DBA must also ensure that the data in the database can be fully recovered in case of physical data loss or loss of database integrity.

Data loss can be partial or total. A **partial loss** can be caused when a physical loss of part of the database has occurred or when part of the database has lost integrity. A **total loss** might mean that the database continues to exist, but its integrity is entirely lost, or the entire database is physically lost. Either way, backup and recovery procedures are the cheapest database insurance.

The management of database security, integrity, backup and recovery is so crucial that many DBA departments have created a position staffed by the **database security officer (DSO)**. The DSO's sole job is to ensure database security and integrity. In large database shops, the DSO's activities are often classified as *disaster management*.

Disaster management includes all the DBA activities designed to secure data availability following a physical disaster or a database integrity failure. Disaster management includes all planning, organising, and testing of database contingency plans and recovery procedures. The backup and recovery measures must include at least:

- **Periodic data and applications backups** Some DBMSs include tools to ensure backup and recovery of the data in the database. The DBA should use those tools to render the backup and recovery tasks automatic. Products such as IBM's DB2 allow the creation of different backup types: full, incremental, and concurrent. A **full backup**, also known as a database dump, produces a complete copy of the entire database. An **incremental backup** produces a backup of all data since the last backup date; a **concurrent backup** takes place while the user is working on the database.
- **Proper backup identification** Backups must be clearly identified through detailed descriptions and date information, thus enabling the DBA to ensure that the correct backups are used to recover the database. While cloud-based backups are fast replacing tape backups, many organisations still use tapes. As tapes require physical storage, it is vital that the storage and labelling of tapes be done diligently by the computer operators, and the DBA must keep track of tape currency and

location. However, organisations that are large enough to hire a DBA do not typically use tapes for enterprise backup. Other emerging solutions include optical and disk-based backup devices. Such backup solutions use a layered backup approach in which the data are first backed up to fast disk media for intermediate storage and fast restoration. Later, the data is transferred to tape for archival storage.

- **Convenient and safe backup storage** There must be multiple backups of the same data, and each backup copy must be stored in a different location. The storage locations must include sites both inside and outside the organisation. The storage locations must be properly prepared and may include fire-safe and earthquake-proof vaults, as well as humidity and temperature controls. The DBA must establish a policy to determine where backups should be stored, and for how long.
- **Physical protection of both hardware and software** Protection might include the use of closed installations with restricted access, as well as preparation of the computer sites to provide air conditioning, backup power, and fire protection. Physical protection also includes the provision of a backup computer and DBMS for use in case of emergency.
- **Personal access control to the software of a database installation** Multilevel passwords and privileges, and hardware and software challenge/response tokens, can properly identify authorised users of resources.
- **Insurance coverage for the data in the database** The DBA or security officer must ensure an insurance policy to provide financial protection in the event of a database failure. The insurance may be expensive, but it is less expensive than the disaster created by massive data loss.

Two additional points:

- Data recovery and contingency plans must be thoroughly tested and evaluated, and they must be practised frequently. So-called fire drills are not to be disparaged, and they require top-level management's support and enforcement.
- A backup and recovery program is not likely to cover all components of an information system. Therefore, it is appropriate to stablish priorities concerning the nature and extent of the data recovery process.

Data Distribution and Use

Data are useful only when they reach the right users at the right time. The DBA is responsible for ensuring that the data are distributed to the right people, at the right time, and in the right format. The DBA's data distribution and use tasks can become very time-consuming, especially when the data delivery capacity is based on a typical applications programming environment, where users depend on programmers to deliver the programs to access the data in the database. Although the Internet and its intranet and extranet extensions have opened databases to corporate users, their use has also created a new set of challenges for the DBA.

Current data distribution philosophy makes it easy for *authorised* end-users to access the database. One way to accomplish this task is to facilitate the use of a new generation of more sophisticated query tools and the Internet Web front-ends. They enable the DBA to educate end-users to produce the required information without being dependent on application programmers. The DBA must ensure that appropriate standards and procedures are adhered to.

This distribution philosophy is common today, and it is likely that it will become more common as database technology marches on. Such an environment is more flexible for the end-user. Enabling end-users to become relatively self-sufficient in the acquisition and use of data can lead to more

efficient use of data in the decision process. Yet this 'data democracy' can also produce some troublesome side effects. Letting end users micromanage their data subsets could inadvertently sever the connection between those users and the data administration function. The DBA's job under those circumstances might become sufficiently complicated to compromise the efficiency of the data administration function. Data duplication might flourish again without checks at the organisational level to ensure the uniqueness of data elements. Thus, end-users who do not completely understand the nature and sources of data might make improper use of the data elements.

10.6.2 The Technical Role of the DBA

The DBA's technical role requires a broad understanding of DBMS functions, configuration, programming languages, data modelling, design methodologies, and other DBMS-related issues. The DBA's technical activities include the selection, installation, operation, maintenance, and upgrading of the DBMS and utility software, as well as the design, development, implementation, and maintenance of the application programs that interact with the database.

Many of the DBA's technical activities are a logical extension of the DBA's managerial activities.

The technical aspects of the DBA's job are rooted in the following areas of operation:

- Evaluating, selecting, and installing the DBMS and related utilities
- Designing and implementing databases and applications
- Testing and evaluating databases and applications
- Operating the DBMS, utilities, and applications
- Training and supporting users
- Maintaining the DBMS, utilities, and applications

Evaluating, Selecting, and Installing the DBMS and Utilities

One of the DBA's first and most important technical responsibilities is selecting the database management system, utility software, and supporting hardware for use in the organisation. Therefore, the DBA must develop and execute a plan for evaluating and selecting the DBMS, utilities, and hardware. The plan must be based primarily on the organisation's needs, rather than on specific software and hardware features. The DBA must recognise that the search is for solutions to problems, rather than for a computer or DBMS software. A DBMS is a management tool, and not a technological toy.

The first and most important step of the evaluation and acquisition plan is to determine company needs. To establish a clear picture of those needs, the DBA must make sure that the entire end-user community, including top- and mid-level managers, is involved in the process. Once the needs are identified, the objectives of the data administration function can be established, and the DBMS features and selection criteria can be defined.

Pros and cons of several alternative solutions must be evaluated during the selection process. Available alternatives are often restricted because software must be compatible with the organisation's existing computer system. A DBMS is just part of the solution: it requires support from other hardware, application software, and utility programs.

DBMS Features Checklist

- **DBMS Model** Are the company's needs better served by a relational, object-oriented, or object/relational DBMS? If a data warehouse application is required, should a relational or multidimensional DBMS be used?
- **DBMS storage capacity** What maximum disk and database size is required? How many disk packages must be supported? How many tape units or what other storage capacity are needed?
- **Application development support** Which third- and fourth-generation languages are supported? Which application development tools (database schema design, data dictionary, performance monitoring, screen and menu painters) are available? Are end-user query tools provided? Does the DBMS provide Web front-end access?
- **Security and integrity** Does the DBMS support referential and entity integrity rules, access rights, and so on? Does the DBMS support the use of audit trails to spot errors and security violations? Can the audit trail size be modified?
- **Backup and recovery** Does the DBMS provide some automated backup and recovery tools? Does the DBMS support tape, optical disk, cloud, or network-based backups? Does the DBMS automatically back up the transaction logs?
- **Concurrency control** Does the DBMS support multiple users? What levels of isolation (table, page, row) does the DBMS offer? How much coding is needed in the application programs?
- **Performance** How many transactions per second does the DBMS support? Are additional transaction processors needed?
- **Database administration tools** Does the DBMS offer some type of DBA management interface? What type of information does the DBA interface provide? Does the DBMS provide alerts to the DBA when errors or security violations occur?
- **Interoperability and data distribution** Can the DBMS work with other DBMS types in the same environment? Which coexistence or interoperability level is achieved? Does the DBMS support `READ` and `WRITE` operations to and from other DBMS packages? Does the DBMS support a client/server architecture?
- **Portability and standards** Can the DBMS run on different operating systems and platforms? Can the DBMS run on mainframes, mid-range computers, and desktop computers? Can the DBMS applications run without modification on all platforms? Which national and industry standards does the DBMS follow?
- **Hardware** Which hardware does the DBMS require?
- **Data dictionary** Does the DBMS have a data dictionary? If so, what information is kept in it? Does the DBMS interface with any data dictionary tool? Does the DBMS support any CASE tools?
- **Vendor training and support** Does the vendor offer in-house training? What type and level of support does the vendor provide? Is the DBMS documentation easy to read and helpful? What is the vendor's upgrade policy?

- **Available third-party tools** Which additional tools are offered by third-party vendors (query tools, data dictionary, access management and control, and storage allocation management tools)?
- **Cost** What costs are involved in the acquisition of the software and hardware? How many additional personnel are required, and what level of expertise is required of them? What are the recurring costs? What is the expected payback period?

The selection process must also consider the site's preparation costs. The DBA must include both one-time and recurring expenditures involved in the required preparation and maintenance.

The DBA must supervise the installation of all software and hardware designated to support the data administration strategy; must have a thorough understanding of the components being installed; and must be familiar with the installation, configuration and startup procedures of each of the components. The installation procedures include details such as the location of backup and transaction log files, network configuration information, and physical storage details. Installation and configuration details are DBMS-dependent.

Designing and Implementing Databases and Applications

The DBA function also provides data modelling and design services to the end-user community. Such services are often coordinated with an application development group within the data-processing department. Therefore, one of the primary activities of a DBA is to determine and enforce standards and procedures to be used. Once the appropriate standards and procedures framework is in place, the DBA must ensure that the database modelling and design activities are performed within the framework. The DBA then provides the necessary assistance and support during the design of the database at the conceptual, logical, and physical levels.

The DBA function usually requires that several people be dedicated to database modelling and design activities. These people might be grouped according to the organisational areas covered by the application. The DBA schedules the design jobs to coordinate the data design and modelling activities. That coordination may require reassignment of available resources based on externally determined priorities.

The DBA also works with application programmers to ensure the quality and integrity of database design and transactions. Such support services include reviewing the database application software to ensure that transactions are:

- **Correct** The transactions mirror real world events.
- **Efficient** The transactions do not overload the DBMS.
- **Compliant** The transactions are compliant with integrity and standards.

The implementation of the applications requires the implementation of the physical database. Therefore, the DBA must provide assistance and oversight during any physical design, including storage space determination and creation, data loading, conversion, and database migration services. The DBA's implementation tasks also include the generation, compilation, and storage of the application's access plan. An **access plan** is a set of instructions generated at application completion time that predetermines how the application will access the database at run time. To be able to create and validate the access plan, the user must have the required rights to access the database.

Before an application comes online, the DBA must develop, test, and implement the operational

procedures required by the new system. Such operational procedures include utilising training, security, and backup and recovery plans, as well as assigning responsibility for database control and maintenance. Finally, the DBA must authorise application users to access the database from which the applications draw the required data. The addition of a new database may require the fine-tuning and/or reconfiguring of the DBMS. The DBMS assists all applications by managing the shared corporate data repository. Therefore, when data structured are added or modified, the DBMS may require the assignment of additional resources to service the new and original users with equal efficiency.

Testing and Evaluating Databases and Applications

The DBA must also provide testing and evaluation services for all the database and end-user applications. These services are the logical extension of the design, development, and implementation services described previously. Testing procedures and standards must already be in place before any application can be approved for use in the company.

Although testing and evaluation services are closely related to database design and implementation services, they are usually maintained independently. This separation is because application programmers and designers are often too close to the problem being studied to detect errors and omissions.

Testing usually starts with the loading of the **test-bed database**. That database contains test data for the applications, and its purpose is to check the data definition and integrity rules of the database and application programs.

The testing and evaluation of a database application cover all aspects of the system – from the simple collection and creation of data to its use and retirement. The evaluation process covers:

- Technical aspects of both the applications and the database. Backup and recovery, security and integrity, use of SQL, and application performance, must be evaluated.
- Evaluation of the written documentation to ensure that the documentation and procedures are accurate and easy to follow.
- Observance of standards for naming, documenting, and coding.
- Data duplication conflicts with existing data.
- The enforcement of all data validation rules.

Following the thorough testing of all applications, the database, and the procedures, the system is declared operational, and can be made available to end-users.

Operating the DBMS, Utilities, and Applications

DBMS operations can be divided into four main areas:

- System support
- Performance monitoring and tuning
- Backup and recovery
- Security auditing and monitoring

System Support

System support activities cover all tasks directly related to the day-to-day operations of the DBMS and its applications.

These activities range form filling out job logs to changing tape, to checking and verifying the status of computer hardware, disk packages, and emergency power sources. System-related activities include periodic, occasional tasks such as running special programs and resource configurations for new and/or upgraded versions of database applications.

Performance Monitoring and Tuning

Activities that are designed to ensure that the DBMS, utilities and applications maintain satisfactory performance levels.

To carry out performance-monitoring and tuning tasks, the DBA must:

- Establish DBMS performance goals
- Monitor the DBMS to evaluate whether the performance objectives are being met
- Isolate the problem and find alternative solutions (if performance objectives are not met)
- Implement the selected performance solutions

DBMSs often include performance-monitoring tools that allow the DBA to query database usage information. If the DBMS does not include performance-monitoring tools, they are available from many different sources. DBMS utilities are provided by third-party vendors, or they may be included in the operating system utilities, or transaction processor facilities. Most of the performance-monitoring tools allow the DBA to focus on selected system bottlenecks. The most common bottlenecks in DBMS performance tuning are related to the use of indexes, query-optimisation algorithms, and management of storage resources.

Because improper index selection can have a negative effect on system performance, most DBMS installations adhere to a carefully defined index creation and usage plan. Such a plan is especially important in a relational database environment.

To produce satisfactory performance, the DBA is likely to spend much time trying to educate programmers and end-users on the proper use of SQL statements. Typically, DBMS programmers' manuals and administration manuals contain useful performance guidelines and examples that demonstrate the proper use of SQL statements, both in the command-line and within application programs. Since relational systems do not give the user an index choice within a query, the DBMS makes the index selection for the user. Therefore, the DBA should create indexes that can improve system performance.

Query-optimisation routines are usually integrated into the DBMS package, thereby allowing few tuning options. Query-optimisation routines are orientated to improving concurrent access to the database. Several database packages let the DBA specify parameters for determining the desired level of concurrency. Concurrency is also affected by the types of locks used by the DBMS and requested by the applications. Because the concurrency issue is important to the efficient operation of the system, the DBA must be familiar with factors that influence concurrency.

Available storage resources, in terms of both primary and secondary memory, must also be considered during DBMS performance tuning. The allocation of storage resources is determined when the DBMS is configured. Storage configuration parameters can be used to determine:

- The number of databases that may be opened concurrently
- The number of application programs or users supported concurrently
- The amount of primary memory (buffer pool size) assigned to each database and each database process
- The size and location of the log files. (These are used to recover the database. The log files can be located in a separate volume to reduce the disk's head movement, and to increase performance.)

Performance monitoring issues are DBMS-specific. Therefore, the DBA must become familiar with the DBMS manuals, to learn the technical details involved in the performance-monitoring task.

Backup and Recovery Activities

Since data loss is likely to be devastating to the organisation, these activities are of primary concern during the DBMS operation. The DBA must establish a schedule for backing up databases and log files at appropriate intervals. Backup frequency is dependent on the application type and on the relative importance of the data. All critical system components – the database, the database applications, and the transaction logs – must be backed up periodically.

Most DBMS packages include utilities that schedule automated database backups, be they full or incremental. Although incremental backups are faster than full backups, an incremental backup requires the existence of a periodic full backup to be useful for recovery purposes.

Database recovery after a media or systems failure requires application of the transaction log to the correct database copy. The DBA must plan, implement, test, and enforce a 'bulletproof' backup and recovery procedure.

Security Auditing and Monitoring

Security auditing and monitoring assumes the appropriate assignment of access rights and the proper use of access privileges by programmers and end-users. The technical aspects of security auditing and monitoring involve creating users, assigning access rights, using SQL commands to grant and revoke access rights to users and database objects, and creating audit trails to discover security violations or attempted violations. The DBA must periodically generate an audit trail report to determine whether there have been actual or attempted security violations – and, if so, from which locations, and, if possible, by whom.

Training and Supporting Users

Training people to use the DBMS and its tools is included in the DBA's technical activities. In addition, the DBA provides or secures technical training in the use of the DBMS and its utilities for application programmers. Application programmer training covers the use of the DBMS tools as well as the procedures and standards required for database programming.

Unscheduled, on-demand technical support for end-users and programmers is also included in the DBA's activities. A technical troubleshooting procedure can be developed to facilitate such support. The technical procedure might include the development of a technical database used to find solutions to common technical problems.

Part of the support is provided by interaction with DBMS vendors. Establishing good relationships with software suppliers is one way to ensure that the company has a good external support source. Vendors are the source for up-to-date information concerning new products and personnel retraining. Good vendor-company relations are also likely to give organisations an edge in determining the future direction of database development.

Maintaining the DBMS, Utilities, and Applications

The maintenance activities of the DBA are an extension of the operational activities. Maintenance activities are dedicated to the preservation of the DBMS environment.

Periodic DBMS maintenance includes management of the physical or secondary storage devices. One of the most common maintenance activities is reorganising the physical location of data in the database. This is usually done as part of the DBMS fine-tuning activities. The reorganisation of a database might be designed to allocate contiguous disk-page locations to the DBMS to increase performance. The reorganisation process might also free the space allocated to deleted data, thus providing more disk space for new data.

Maintenance activities also include upgrading the DBMS and utility software. The upgrade might require the installation of a new version of the DBMS software, or an Internet front-end tool. Or it might create an additional DBMS gateway to allow access to a host DBMS running on a different host computer. DBMS gateway services are common in distributed DBMS applications running in a client/server environment. Also, new-generation databases include features such as spatial data support, data warehousing, and star query support, and support for Java programming interfaces for Internet access.

Quite often, companies are faced with the need to exchange data in dissimilar formats, or between databases. The maintenance efforts of the DBA include migration and conversion services for data in incompatible formats or for different DBMS software. Such conditions are common when the system is upgraded from one version to another, or when the existing DBMS is replaced by an entirely new DBMS. Database conversion services also include downloading data from the host DBMS to an end-user's desktop computer to allow that user to perform a variety of activities – spreadsheet analysis, charting, statistical modelling, and so on. Migration and conversion services can be done at the logical (DBMS- or software-specific) level, or at the physical (storage-media- or operating-system-specific) level.

10.6.3 Developing a Data Administration Strategy

For a company to succeed, its activities must be committed to its main objectives or mission. Therefore, regardless of a company's size, a critical step for any organisation is to ensure that its information system supports its strategic plans for each of its business areas.

The database administration strategy must not conflict with the information systems plans. After all, the information systems plans are derived from a detailed analysis of the company's goals, its condition or situation, and its business needs. Several methodologies are available to ensure the compatibility of data administration and information systems plans, and to guide the strategic plan development. The most commonly used method is known as **information engineering**.

Information Engineering (IE)

Information engineering allows for the translation of the company's strategic goals into the data and applications that will help the company achieve those goals. IE focuses on the description of the corporate data instead of the processes. The IE rationale is simple: business data types tend to remain fairly stable and do not change much during their existence. In contrast, processes change often and thus require the frequent modification of existing systems. By placing the emphasis on data, IE helps decrease the impact on systems when processing changes.

The output of the IE process is an **information systems architecture (ISA)** that serves as the basis for planning, development, and control of future information systems.

Implementing IE methodologies in an organisation is a costly process that involves planning, a commitment of resources, management liability, well-defined objectives, identification of critical factors, and control. An ISA provides a framework that includes the use of computerised, automated, and integrated tools such as a DBMS and CASE tools.

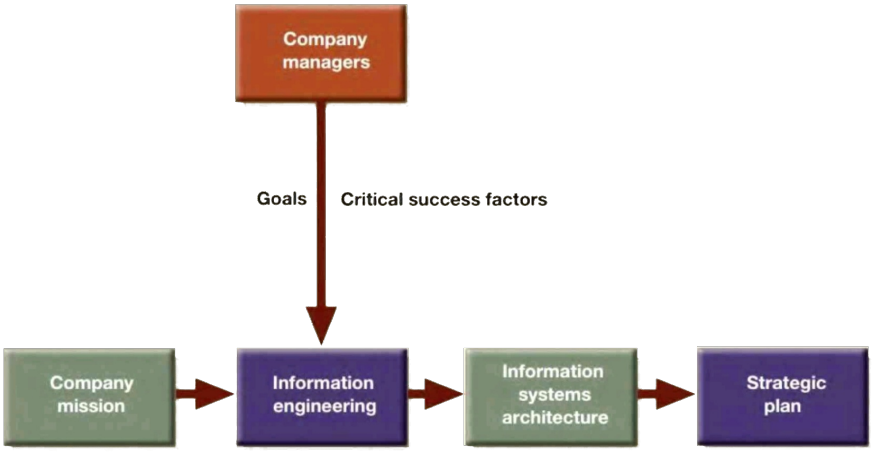


Figure 10.4: Forces affecting the development of the ISA

The success of the overall information systems strategy and, therefore, of the data administration strategy, depends on several critical success factors. Understanding the critical success factors helps the DBA develop a successful corporate data administration strategy. Critical success factors include managerial, technological, and corporate culture issues, such as:

- **Management commitment** Top-level management commitment is necessary to enforce the use of standards, procedures, planning, and controls. The example must be set at the top.
- **Thorough company situation analysis** The current situation of the corporate data administration must be analysed to understand the company's position, and to have a clear vision of what must be done. For example, how are database analysis, design, documentation, implementation, standards, codification, and other issues handled? Needs and problems should be identified first, then prioritised.
- **End-user involvement** Another aspect critical to the success of the data administration strategy. What is the degree of organisational change involved? Successful organisational change requires that people are able to adapt to the change. Users should be given an open communication channel to upper-level management to ensure success of the implementation. Good communication channels are key to the overall process.
- **Defined standards** Analysts and programmers must be familiar with appropriate methodologies, procedures, and standards. If analysts and programmers lack familiarity, they may need to be trained in the use of the procedures and standards.
- **Training** The vendor must strain the DBA personnel in the use of the DBMS and other tools. End-users must be trained to use the tools, standards, and procedures to obtain and demonstrate the maximum benefit, thereby increasing end-user confidence. Key personnel should be trained first, so they can train others later.
- **A small pilot project** A small project is recommended to ensure that the DBMS will work in the company, that the output is what was expected, and that the personnel have been trained properly.

This list of factors is not and cannot be comprehensive. Nevertheless, it does provide the initial framework for the development of a successful strategy. However, no matter how comprehensive the list of success factors is, it must be based on the notion that development and implementation of a successful data administration strategy are tightly integrated with the overall information systems planning activity of the organisation.

Chapter 11

Conceptual, Logical, and Physical Database Design

- **Conceptual Database Design** Create the conceptual representation of the database by producing a data model that identifies the relevant entities and relationships within our system.
    - Data analysis and requirements
    - Entity relationship modelling and normalisation
    - Data model verification
    - Distributed database design
- **Logical Database Design** Design relations based on each entity, and define integrity rules to ensure there are no redundant relationships within our database.
    - Creating the logical data model
    - Validating the logical data model using normalisation
    - Assigning and validating integrity constraints
    - Merging logical models constructed for different parts of the database
    - Reviewing the logical data model with the user
- **Physical Database Design** Implement the physical database in the target DBMS. Consider how each relation is stored, and how the data is accessed.
    - Translate each relation identified in the logical data model into tables
    - Determine a suitable file organisation
    - Define indexes
    - Define user views
    - Estimate data storage requirements
    - Determine database security for users

A relational DBMS design should maintain logical and physical data independence. **Logical design** is concerned with what the database looks like to the user. **Physical design** is concerned with how the logical design maps to the physical storage of the database in secondary storage.

- If the logical structure of the database should change, then the way that the user views the database should not change.
- If the physical methods of storing and retrieving data change, then the user interface should not be affected in any way.

11.1 Conceptual Design

Data modelling is used to create an abstract database structure that represents real-world objects in the most realistic way possible. The conceptual model must embody a clear understanding of the business and its functional areas. The design must be software and hardware independent.

Minimal Data Rule

All that is needed is there, and all that is there is needed.

Focus not only on the immediate needs of the business, but also on the future data needs. Leave room for future modifications and additions.

11.1.1 Data Analysis and Requirements

The first step in conceptual design is to discover the characteristics of the data elements. Appropriate data element characteristics are those that can be transformed into appropriate information.

The designer is focused on:

- **Information Needs** What output must be generated by the system, what information does the current system generate, to what extent is that information adequate?
- **Information Users** Who will use the information? How will the information be used? What are the different end-user data views?
- **Information Sources** Where is the information to be found? How is the information to be extracted once it is found?
- **Information Constitution** What data elements are needed to produce the information? What are the data attributes? What relationships exist among the data? What is the data volume? How frequently are the data used? What data transformations are to be used to generate the required information?

Sources

- Developing and gathering end-user data views
- Directly observing the current system: existing and desired output
- Interfacing with the systems design group

Business Rule

A brief and precise narrative description of a policy, procedure, or principle within a specific organisation's environment.

These help to create and enforce actions within that organisation's environment. When business rules are written properly, they define entities, attributes, relationships, multiplicities, and constraints.

They define the main and distinguishing characteristics of the data as *viewed by the company*.

Business rules must be easy to understand, and widely disseminated to ensure that every person in the organisation shares a common interpretation of the rules. Ideally, business rules are derived from a formal **description of operations**.

Description of Operations

A document that provides a precise, detailed, up-to-date, and thoroughly reviewed description of the activities that define an organisation's operating environment.

The operating environment is both the data sources and the data users. This environment is also dependent on the organisation's mission.

The main sources of information for the description of operations are company managers, policymakers, department managers, and written documentation such as company procedures, standards, and operation manuals. Interviewing end-users can be effective, but they can be a more unreliable source for business rules. End-users may have different perspectives, and it is the database designer's job to reconcile the differences, and verify the results of the reconciliation to ensure the business rules are appropriate and accurate.

Benefits of Business Rules in the Design of New Systems

- They help standardise the company's view of the data.
- They constitute a communications tool between users and designers.
- They allow the designer to understand the nature, role, and scope of the data.
- They allow the designer to understand business processes.
- They allow the designer to develop appropriate relationship participation rules and foreign key constraints.

11.1.2 Entity Relationship Modelling and Normalisation

Before creating the ER model, the designer must communicate and enforce appropriate standards to be used in the documentation of the design.

Steps for Developing the Conceptual Model using ER Diagrams

1. Identify, analyse, and refine the business rules.
2. Identify the main entities, using the results of the previous step.
3. Define the relationships among the entities, using the results of the previous steps.
4. Define the attributes, primary keys, and foreign keys for each of the entities.
5. Normalise the entities. (Remember the entities are implemented as tables in an RDBMS.)
6. Complete the initial ER diagram.
7. Have the main end users verify the model against the data, information and processing requirements.
8. Modify the ER diagram, using the results of the previous step.

All objects (entities, attributes, relations, views, etc.) are defined in a **data dictionary**, which is used together with the normalisation process to help eliminate data anomalies and redundancy problems.

Designer's Tasks during ER Modelling

- Define entities, attributes, primary keys, and foreign keys.
- Make decisions about adding new primary key attributes to satisfy end-user and/or processing requirements.
- Make decisions about the treatment of multivalued attributes.
- Make decisions about the placement of foreign keys in 1:1 relationships.
- Avoid unnecessary ternary relationships.
- Draw the corresponding ER diagram.
- Normalise the entities.
- Include all data element definitions in the data dictionary.
- Make decisions about standard naming conventions.

Naming Conventions

- Use descriptive entity and attribute names wherever possible.
- Composite entities are usually assigned a name that describes the relationship they represent.
- An attribute name should be descriptive, and it should contain a prefix that helps identify the table in which it is found. The advantage of this naming convention is that it immediately identifies a table's foreign keys.

It is not always possible to adhere strictly to the naming conventions.

11.1.3 Data Model Verification

The ER model must be verified against the proposed system processes in order to confirm that the intended purposes can be supported by the database model. Verification requires that the model be run through a series of tests against:

- End-user data views, and their required transactions. Such transactions include the data manipulation commands `SELECT`, `INSERT`, `UPDATE`, and `DELETE`.
- Access paths and security.
- Business-imposed data requirements and constraints.

Revision of the database design starts with a careful re-evaluation of the entities, followed by a detailed examination of the attributes that describe those entities. This process serves several important purposes:

- The emergence of the attribute details may lead to a revision of the entities themselves. Perhaps some of the components first believed to be entities will turn out to be attributes within other entities. Or what was originally considered to be an attribute might contain a significant number of subcomponents, and so should rather be an entity.
- The focus on attribute details can provide clues about the nature of relationships as they are defined by the primary and foreign keys. Improperly defined relationships lead to implementation problems first, and to application development problems later.

- To satisfy processing and/or end-user requirements, it might be useful to create a new primary key to replace an existing primary key.
- Unless the entity details are precisely defined, it is difficult to evaluate the extent of the design's normalisation. Knowledge of the normalisation level helps guard against undesirable redundancies.
- A careful review of the rough database design blueprint is likely to lead to revisions. Those revisions will help ensure that the design is capable of meeting end-user requirements.

Because real-world database design is generally done by teams, the design's major components should be organised into **modules**.

Module

An information system component that handles a specific function, such as inventory, orders, payroll, and so on. At the design level, a module is an ER segment that is an integrated part of the overall ER model.

Creating and using modules accomplishes several important ends:

- The modules can be delegated to design groups within teams, which speeds up the development work.
- The modules simplify the design work, by containing a smaller, more manageable number of entities.
- The modules can be prototyped quickly. Implementation and application programming trouble spots can be identified quicker.
- Even if the entire system cannot be brought online quickly, the implementation of one or more modules will demonstrate that progress is being made.

While modules are useful, they represent ER model fragments. Fragmentation creates a potential problem: the fragments may not include all the ER model's components, and may, therefore, not be able to support all the required processes. To avoid that problem, the modules must be verified against the complete ER model.

The ER Model Verification Process

1. Identify the ER Model's central entity.
2. Identify each module and its components.
3. Identify each module's transaction requirements:
   - **Internal** Updates, Inserts, Deletes, Queries, Reports
   - **External** Module Interfaces
4. Verify all processes against the ER model.
5. Make all necessary changes suggested in the previous step.
6. Repeat Steps 2–5 for all modules.

The verification process requires the continuous verification of business transactions as well as system and user requirements.

The process starts with selecting the central (most important) entity. This entity is defined in terms of its participation in most of the model's relationships, and it is the focus of most of the system's operations. That is, the entity involved in the greatest number of relationships.

Then one identifies the module or subsystem to which the central entity belongs, and defines that module's boundaries and scope. The entity belongs to the module that uses is most frequently. Once each module is defined, the central entity is placed within the module's framework, to let you focus your attention on the module's details.

Within the central entity/module framework, a designer needs to:

- **Ensure the module's cohesivity** The term **cohesivity** describes the strength of the relationships found among the module's entities. A module must display **high cohesivity** – that is, the entities must be strongly related, and the module must be complete and self-sufficient.
- **Analyse each module's relationships with other modules to address module coupling** Modules need to be independent of each other. **Module Coupling** describes the extent to which modules are independent. Modules must display **low coupling**, indicating that they are independent of other modules. Low coupling decreases unnecessary inter-module dependencies, thereby allowing the creation of a truly modular system and eliminating unnecessary relationships among entities.

One of the design challenges is to achieve the right balance between cohesion and coupling. If one seeks to have highly cohesive modules, one might create more modules that are dependent on each other, which means there is high coupling.

Process Classification

Processes may be classified according to their:

- Frequency (daily, weekly, monthly, yearly, or exceptions)
- Operational type (`INSERT` or `ADD`, `UPDATE` or `CHANGE`, `DELETE`, queries and reports, batches, maintenance and backups)

All identified processes must be verified against the ER model. If necessary, appropriate changes are implemented. The process verification is repeated for all the model's modules.

11.2 Logical Database Design

11.2.1 Creating the Logical Data Model

The first stage of logical database design is to translate the conceptual design into a set of relational database constructs. This involves converting the ER model from into a set of relations using a set of rules. A relation must be created for each entity and relationships and attributes must be created while meeting the required integrity constraints. Usually, relations with no dependents (not containing any foreign keys) are created first.

Creating a Relation

The name of the relation is specified along with its associated attributes enclosed in brackets. Next, the primary keys are identified, followed by any foreign keys. Primary keys are underlined, foreign keys are written with an asterisk (*) after.

Example: DVD Relation

DVD(DVD_ID, DVD_TITLE, DVD_COPIES, DVD_CHARGE)

Relations for Weak Entities

A **weak entity** is one that is existence-dependent – it cannot exist without the entity with which it has a relationship. A weak entity has a primary key that is totally or partially derived from the parent entity in the relationship. For each weak entity, a new relation must be created that includes all attributes from the entity.

The primary key of the relation is then determined from each owner of the entity. It cannot be established until all the foreign key relationships with the owning entities have been identified. So, the primary key of the owner identity is included in the new relation as a foreign key attribute. The primary key of the new relation then becomes a composite key through combining the primary key of the owner entity and the partial identifier of the weak entity.

Mapping Binary Relations

One-to-Many Relationships

Create the relations for each of the two entities participating in the relationship. To create the foreign key on the 'many' side, include the primary key attribute from the 'one' side. The 'one' side is referred to as the **parent table**, and the 'many' as the **dependent table**.

One-to-One Relationships

- If both entities are in a mandatory participation in a relationship, and they do not participate in other relationships, it is most likely that the two entities should be part of the same entity.
- If there is mandatory participation on one side of the relationship, then the entity that has the optional participation becomes the parent entity and the entity that has the mandatory relationship becomes the dependent entity. The relation corresponding to the parent entity should contain the foreign key of the dependent entity.
- If both entities are in an optional participation, the database designer should choose which should be the parent entity.

Many-to-Many Relationships

Create the relations for each of the two entities participating in the relationship. Then create a third relation to represent the actual relationship. The third relation will contain the foreign keys of the two original entities that participated in the original 1:1 relationship.

11.3 Physical Database Design

Physical database design requires the definition of specific storage or access methods that will be used by the database. We translate the logical model into a set of DBMS specifications for storing and accessing data.

Required Info for Physical Database Design

- Set of normalised relations devised from the ER model and the normalisation process.
- Estimate of the volume of data which will be stored in each database table, and the usage statistics.
- Estimate of the physical storage requirements for each field (attribute) within the database.
- Physical storage characteristics of the DBMS being used.

11.3.1 Analyse Data Volume and Database Usage

Gio Wiederhold

20 percent of queries requested by users account for 80 percent of data accesses.

The steps for carrying out this phase are:

1. Identify the most frequent and critical transactions.
2. Analyse the critical transactions to determine which relations in the database participate in these transactions.

Data volume and data usage statistics are shown on a simplified version of the ERD, known as a **composite usage map**, or **transaction usage map**.

11.3.2 Determine a Suitable File Organisation

Techniques for physically arranging relations onto secondary storage are known as **file organisation techniques**. Selecting the most suitable file organisation is important to ensure data is stored efficiently, and can be retrieved as quickly as possible. There are three categories of file organisations:

- **Heap Files** Contain randomly ordered records.
- **Index Files** Sorted by one or more fields, such as file organisations based on indexes.
- **Hash Files** Files hashed on one or more fields.

Heap File Organisations

The most basic file organisation is that of a **heap file**, where records are unordered. Records are inserted into a file as they come. Used only when a large quantity of data needs to be inserted into a table for the first time. The input sequence is used to automatically generate a primary key for each row. The only way to access a record in this type of file is to search every row in the file.

Sequential File Organisations

In a **sequential file** organisation, the records are stored in a sequence based on the value of one or more fields, which is often the primary key. In order to locate a specific record, the whole file must be searched, and every record in the file must be read until the required record is located. It can be faster if records are ordered based on the primary key value, but this is not always the case. Inserting or modifying records usually results in rewriting the whole file. The deletion of records leads to storage space being wasted.

Indexed File Organisations

Used to access a record directly instead of searching through the entire file.

Records in a file supporting this type of file organisation can be stored in a sorted or unsorted sequence, and an index is created to locate specific records quickly. An **index** is an ordered set of values that contains the index key and pointers. The pointers are the row IDs for actual table rows.

An index scan is more efficient than a full table scan, because the index data are already ordered, and the amount of data is usually a magnitude of scale smaller.

Indexes are logically and physically independent of the data in the associated table. They require their own storage space. How much space depends on the type of index that is applied.

Main Types of Indexes

- **Primary Index** Placed on unique fields, such as the primary key. Used to locate a specific record pointed to by the index. A file can have at most one primary index, but can have several secondary indexes.
- **Secondary Index** Indexes places on any field in the file that is unordered.
- **Multilevel Index** Used when one index becomes too large, and is split into a number of separate indexes to reduce the search.

Each index can be defined as being sparse or dense. When using a sparse index, index pointers are created only for some of the records, whereas with a dense index, an index pointer appears for every search key value in the file. Dense indexes are faster, but sparse indexes require less storage space.

B-Tree Index

Within a DBMS, indexes are often stored in a data structure known as a *tree*. These are more efficient at storing indexes, as they reduce the time of the search compared with other data structures.

These trees are called **Balanced (B) trees**, and are used to maintain an ordered set of indexes or data. They consist of a set of hierarchy of nodes that contains a set of pointers that link the nodes of the tree together. Each B-tree that is created is said to be of the order n, where n is the maximum number of children allowed for each parent node.

When a node does not have any children, it is called a **leaf node**.

B-tree indexes are mainly used when you know that a query refers to a column which is indexed, and will retrieve only a few rows.

Bitmap Indexes

Often used on multidimensional data held in data warehouses.

Bitmap indexes are usually applied to attributes that are sparse in a given domain. A 2D array is constructed. One column is generated for every row in the table we want to index, with each column representing a distinct value within the bitmapped index.

Bitmaps are more compact than B-trees and take up less storage space. They are usually used when:

- A column in the table has low cardinality – for Oracle, less than 100 distinct values.
- The table is not used for data manipulation activities. Updating bitmapped indexes takes a lot of time, so if a table is often updated, this can be resource intensive. Bitmapped indexes are most suitable for large, read-only tables.
- Specific SQL queries reference a number of low cardinality values in their `WHERE` clauses.

Join Index

Used mainly in data warehousing. Applies to columns from two or more tables whose values come from the same domain. Often referred to as a **bitmap join index**, and is a way of saving space by reducing the volume of data that must be joined. The bitmap join stores the ROWIDs of corresponding rows in a separate table.

This type of index is useful when dealing with large quantities of data that are typically found in data warehouses. They are unsuitable when there are high-volume updates. The queries that reference these indexes may also not reference any fields in the `WHERE` clause which are not in the join index.

Hashed File Organisations

A **hashed file** organisation uses a hashing algorithm to map a primary key value onto a specific record address in the file. Records are stored in a random order throughout the file. Files that follow hashed organisation are called *random* or *direct* files. The aim of a hashing algorithm is to distribute records evenly within the data storage area. The algorithm reduces the primary key value to a shorter identifier, called a **hash**.

The main weakness with hashing algorithms is that there is no guarantee that a unique address is generated – if the algorithm generates the same hash for two different primary keys, it is known as a **collision**.

Clusters

Tables can be clustered together, if there are fields in them that are accessed frequently together. The **cluster key** is a field, or set of fields, that the clustered tables have in common, which is usually defined through the table join. This key is determined when the cluster is created.

11.3.3 Define Indexes

Each table typically has a **primary index** created for the primary key of the table. **Secondary indexes** are usually placed on additional fields that are used regularly in queries to increase the speed of data retrieval.

Index selectivity is a measure of how likely it is that an index will be used in query processing. Indexes should have high selectivity.

11.3.4 Database Security

System privileges authorise a user account to execute DDL commands. **Object privileges** allow a user account to execute DML commands.

Users of a database can be grouped together depending on the type of privileges they require, and a database role can be assigned to each group. A **role** is simply a collection of privileges referred to under a single name.

Chapter 12

Managing Transactions and Concurrency

12.1 What is a Transaction?

A **transaction** is any action that reads from and/or writes to a database. It is a *logical* unit of work that must be entirely completed or entirely aborted; no intermediate states are acceptable. A successful transaction changes the database from one consistent state to another. A **consistent database state** is one in which all data integrity constraints are satisfied. To ensure consistency of the database, every transaction must begin with the database in a known consistent state.

A **database request** is the equivalent of a single SQL statement in an application program or transaction. Each database request generates several I/O operations that read from or write to physical storage media.

12.1.1 Evaluating Transaction Results

If the DBMS supports transaction management, if it encounters an inconsistent state, it will roll back the database to a previous consistent state. Although the DBMS is designed to recover a database to a previous consistent state when an interruption prevents the completion of a transaction, the transaction itself is defined by the end user or programmer, and must be semantically correct.

The DBMS cannot guarantee that the semantic meaning of the transaction truly represents the real-world event.

Improper or incomplete transactions can seriously affect database integrity. Some DBMSs provide means by which the user can define enforceable constraints based on business rules. Other integrity rules, such as those governing referential and entity integrity, are enforced automatically by the DBMS when the table structures are properly defined.

12.1.2 Transaction Properties

All transactions must display *atomicity*, *consistency*, *isolation*, *durability*, and *serialisability*, which collectively are called the **ACIDS test**.

ACIDS Test

- **Atomicity** Requires that *all* operations of a transaction be completed; if not, the transaction is aborted. A transaction is treated as a single, indivisible, logical unit of work.
- **Consistency** Indicates the permanence of the database's consistent state. A transaction takes a database from one consistent state to another. When a transaction is completed, the database reaches a consistent state. If any of the transaction parts violate an integrity constraint, the entire transaction is aborted.
- **Isolation** The data used during the execution of a transaction cannot be used in a second transaction until the first one is completed. This is useful in multi-user database environments, because several different users can access and update the database at the same time.
- **Durability** Ensures that once transaction changes are done, they cannot be undone or lost, even in the event of a system failure.
- **Serialisability** Ensures that concurrent execution of several transactions yields consistent results. If there are multiple transactions, the results of the transactions appear to have been executed one after another. This is important in multi-user and distributed databases, where multiple transactions are likely to be executed concurrently.

A single-user database system automatically ensures serialisability and isolation of the database, because only one transaction is executed at a time. The atomicity and the durability of the transactions must be guaranteed by the single-user DBMS.

Multi-user databases are typically subject to multiple concurrent transactions. Therefore, the multi-user DBMS must implement controls to ensure serialisability and isolation of transactions – in addition to atomicity and durability – to guard the database's consistency and integrity.

12.1.3 Transaction Management with SQL

Transaction support is provided by two SQL statements: `COMMIT` and `ROLLBACK`. ANSI standards require that, when a transaction sequence is initiated by a user or application program, the sequence must continue through all succeeding SQL statements until one of the following four events:

1. A `COMMIT` statement is reached, in which case all changes are permanently recorded within the database. The `COMMIT` statement automatically ends the SQL transaction.
2. A `ROLLBACK` statement is reached, in which case all changes are aborted, and the database is rolled back to its previous consistent state.
3. The end of a program is successfully reached, in which case all changes are permanently recorded in the database. This action is equivalent to `COMMIT`.
4. The program is abnormally terminated, in which case the changes made in the database are aborted, and the database is rolled back to its previous consistent state. This action is equivalent to `ROLLBACK`.

12.1.4 The Transaction Log

A DBMS uses a **transaction log** to keep track of all transactions that update the database. The information stored in this log is used by the DBMS for a recovery requirement triggered by a rollback statement, a program's abnormal termination, or a system failure.

While the DBMS executes transactions that modify the database, it also automatically updates the transaction log. The transaction log stores:

- A record for the beginning of the transaction.
- For each transaction component (SQL statement):
  - The type of operation (update, delete, insert)
  - The names of the objects affected by the transaction
  - The 'before' and 'after' values for the fields being updated.
  - Pointers to the previous and next transaction log entries for the same transaction.
- The ending (`COMMIT`) of the transaction.

If a `ROLLBACK` is issued before the termination of a transaction, the DBMS will restore the database only for that particular transaction, rather than for all transaction, to maintain the *durability* of the previous transactions. That is, committed transactions are not rolled back.

The transaction log is itself a database, and is managed by the DBMS like any other database.

12.2 Concurrency Control

Concurrency control is the coordination of the simultaneous execution of transactions in a multi-user database system. The objective of concurrency control is to ensure the serialisability of transactions in a multi-user database environment. Concurrency control is important because the simultaneous execution of transactions over a shaded database can create several data integrity and consistency problems. The three main problems are lost updates, uncommitted data, and inconsistent retrievals.

12.2.1 Lost Updates

The **lost update** problem happens when two concurrent transactions are updating the same data element, and one of the updates is lost (overwritten by the other transaction).

12.2.2 Uncommitted Data

The **uncommitted data** problem occurs when two transactions, T_1 and T_2, are executed concurrently, and the first transaction (T_1) is rolled back after the second transaction (T_2) has already accessed the uncommitted data – thus violating the *isolation* property of transactions.

12.2.3 Inconsistent Retrievals

Inconsistent retrievals occur when a transaction accesses data before and after one or more other transactions finish working with such data. The problem is that the transaction might read some data before they are changed, and other data *after* they are changed, which leads to inconsistent results.

12.2.4 The Scheduler

Database consistency can only be ensured before and after the execution of transactions (not during). A database always moves through an unavoidable temporary state of inconsistency during a transaction's execution. That temporary inconsistency exists because a computer cannot execute two operations at the same time, and must execute them serially. During this serial process, the isolation property of transactions prevents them from accessing the data not yet released by other transactions. The order that transactions occur in affects the results of the transactions.

The **scheduler** is a special DBMS program that establishes the order in which the operations within concurrent transactions are executed. The scheduler *interleaves* the execution of database operations to ensure serialisability and isolation of transactions. To determine the appropriate order, the scheduler bases its actions on concurrency control algorithms.

The scheduler also makes sure the CPU is used efficiently, and facilitates isolation to ensure that two transactions do not update the same data element at the same time.

Several methods have been proposed to schedule the execution of conflicting operations in concurrent operations. Those methods have been classified as locking, time stamping, and optimistic. Locking methods are used the most frequently.

12.3 Concurrency Control with Locking Methods

A **lock** guarantees exclusive use of a data item to a current transaction. That is, transaction T_2 does not have access to a data item that is currently being used by transaction T_1. A transaction acquires a lock prior to data access; the lock is released (unlocked) when the transaction is complete, so that another transaction can lock the data item for its exclusive use.

This series of locking actions assumes that concurrent transactions might attempt to manipulate the same data at the same time. The use of locks is based on the assumption that conflict between transactions is likely. This is known as **pessimistic locking**.

Data consistency cannot be guaranteed *during* a transaction; the database may be in a temporary inconsistent state when several updates are executed. Therefore, locks are required to prevent another transaction from reading inconsistent data.

Most multi-user DBMSs automatically initiate and enforce locking procedures. All lock information is managed by a **lock manager**, which is responsible for assigning and policing the locks used by the transactions.

12.3.1 Lock Granularity

Lock granularity indicates the level of lock use. Locking can take place at the following levels: database, table, page, row, field.

Database Level

In a **database-level lock**, the entire database is locked, thus preventing the use of any tables in the database by transaction T_2 while transaction T_1 is executing. Transactions T_1 and T_2 cannot access the same database concurrently, even when they use different tables.

This level of locking is good for batch processes, but is unsuitable for online multi-user DBMSs.

Table Level

In a **table-level lock**, the entire table is locked, preventing access to any row by transaction T_2 while transaction T_1 is using the table. If a transaction requires access to several tables, each table may be locked. However, two transactions can access the same database as long as they access different tables.

Table-level locks, while less restrictive than database-level locks, cause traffic jams when many transactions are waiting to access the same table. If different transactions require access to different parts of the same table (i.e. they won't interfere with each other), this can be frustrating. So, table-level locks are not suitable for multi-user DBMSs.

Page Level

In a **page-level lock**, the DBMS locks an entire diskpage. A **diskpage** or **page** is the equivalent of a *diskblock*, which can be described as a directly addressable section of a disk. A page has a fixed size, such as 4K, 8K, or 16K. For example, if you want to write only 73 bytes to a 4K page, the entire 4K page must be read from disk, updated in memory, and written back to disk. A table can span several pages, and a page can contain several rows of one or more tables.

Page-level locks are currently the most frequently used multi-user DBMS locking method.

Row Level

A **row-level lock** is much less restrictive than the earlier locks. The DBMS allows concurrent transactions to access different rows of the same table, even when the rows are located on the same page.

Although the row-level locking approach improves the availability of data, its management requires high overhead (a lock needs to exist for each row in each table in the database).

Field Level

The **field-level lock** allows concurrent transactions to access the same row as long as they require the use of different fields (attributes) within that row.

Although field-level locking yields the most flexible multi-user data access, it is rarely done because it requires an extremely high level of computer overhead.

12.3.2 Lock Types

Regardless of the level of locking, the DBMS may use different lock types: binary of shared/exclusive.

Binary Lock

Has only two states: locked (1) or unlocked (0).

If an object is locked by a transaction, no other transaction can use that object. If an object is unlocked, any transaction can lock the object for its use. Every database operation requires that the affected object be locked. As a rule, a transaction must unlock the object after its termination. Therefore, every transaction requires a lock and unlock operation for each data item that is accessed. Such operations are automatically managed and scheduled by the DBMS.

The lock and unlock features eliminate the lost update problem. However, binary locks are now considered too restrictive to yield optimal concurrency conditions. For example, the DBMS will not allow two transactions to read the same database object, even though neither transaction updates the database (and therefore, no concurrency problems can occur).

Concurrency conflicts only occur when two transactions execute concurrently, *and* one of them updates the database.

Shared/Exclusive Locks

Exclusive Lock

An **exclusive lock** exists when access is reserved specifically for the transaction that locked the object. The exclusive lock must be used when the potential for conflict exists.

Issued when a transaction wants to update (write) a data item, and no locks are currently held on that data item by any other transaction.

Shared Lock

A **shared lock** exists when concurrent transactions are granted Read access on the basis of a common lock. A shared lock produces no conflict as long as all the concurrent transactions are read-only.

Issued when a transaction wants to read data from the database and no exclusive lock is held on that data item.

Using the shared/exclusive locking concept, a lock can have three possible states: unlocked, shared (Read), and exclusive (Write).

Because multiple Read transactions can be safely executed at once, shared locks allow several Read transactions to read the same data item concurrently.

Mutually Exclusive Rule

Only one transaction at a time can own an exclusive lock on the same object.

Example If transaction T_2 updates data item X, an exclusive lock is required by T_2 over data item X. *The exclusive lock is granted iff no other locks are held on the data item*. Therefore, if a shared or exclusive lock is already held on data item X by transaction T_1, an exclusive lock cannot be granted to T_2, and T_2 must wait until T_1 commits before beginning.

Although the possibility of shared locks renders data access more efficient, a shared/exclusive lock schema increases the lock manager's overhead:

- The type of lock held must be known before a lock can be granted.
- Three lock operations exist: `READ_LOCK` (to check the type of lock), `WRITE_LOCK` (to issue the lock), and `UNLOCK` (to release the lock).
- The schema has been enhanced to allow a lock upgrade (from shared to exclusive) and a lock downgrade (from exclusive to shared).

Although locks prevent serious data inconsistencies, they can lead to two major problems:

- The resulting transaction schedule may not be serialisable.
- The schedule may create deadlocks. A database **deadlock** is caused when two transactions wait for each other to unlock data.

12.3.3 Two-Phase Locking to Ensure Serialisability

Two-phase locking defines how transactions acquire and relinquish locks. Two-phase locking guarantees serialisability, but does not prevent deadlocks. The two phases are:

1. A **growing phase**, in which a transaction acquires all required locks without unlocking any data. Once all locks have been acquired, the transaction is in its locked point.
2. A **shrinking phase**, in which a transaction releases all locks and cannot obtain any new lock.

The two-phase locking protocol is governed by the following rules:

- Two transactions cannot have conflicting locks.
- No unlock operations can predate a lock operation in the same transaction.
- No data are affected until all locks are obtained – that is, until the transaction is in its locked point.

Two-phase locking increases the transaction processing cost, and may cause additional undesirable effects, such as the possibility of creating deadlocks.

12.3.4 Deadlocks

A deadlock occurs when two transactions wait for each other to unlock data.

Example: Deadlock

Two transactions T_1 and T_2 exist in the following mode:

$\boldsymbol{T_1}$ Access data items X and Y.
$\boldsymbol{T_2}$ Access data items Y and X.

If T_1 has not unlocked Y, T_2 cannot begin. If T_2 has not unlocked X, T_1 cannot continue. Consequently, T_1 and T_2 will wait indefinitely, each waiting for the other to unlock the required data item. Such a deadlock state is known as a **deadly embrace**.

Deadlocks are only possible when one of the transactions wants to obtain an exclusive lock on a data item; no deadlock condition can exist among *shared* locks.

Techniques to Control Deadlock

- **Deadlock Prevention** A transaction requesting a new lock is aborted when there is the possibility that a deadlock can occur. If the transaction is aborted, all changes made by this transaction are rolled back, and all locks obtained by the transaction are released. The transaction is then rescheduled for execution. Deadlock prevention works because it avoids the conditions that lead to deadlocking.
- **Deadlock Detection** The DBMS periodically tests the database for deadlocks. If a deadlock is found, one of the transactions (the 'victim') is aborted (rolled back and restarted), and the other transaction continues.
- **Deadlock Avoidance** The transaction must obtain all of the locks it needs before it can be executed. This technique avoids the rollback of conflicting transactions by requiring that locks by obtained in succession. However, the serial lock assignment required in deadlock avoidance increases action response times.

The choice of the best deadlock control method depends on the database environment. If the probability of deadlocks is low, deadlock detection is recommended. If the probability is high, deadlock prevention is recommended. If response time is not a priority, deadlock avoidance might be used.

12.4 Concurrency Control with Time Stamping Methods

The **time stamping** approach to scheduling concurrent transactions assigns a global, unique time stamp to each transaction. The time stamp value produces an explicit order in which transactions are submitted to the DBMS. Time stamps must have two properties: uniqueness and monotonicity.

- **Uniqueness** Ensures that no equal time stamp values can exist.
- **Monotonicity** Ensures that time stamp values always increase.

All database operations within the same transaction must have the same time stamp. The DBMS executes conflicting operations in time stamp order, thereby ensuring serialisability of the transactions. If two transactions conflict, one is stopped, rolled back, rescheduled and assigned a new time stamp value.

The disadvantage of the time stamping approach is that each value stored in the database requires two additional time stamp fields: one for the last time the field was read, and one for the last update. Time stamping thus increases memory needs and the database's processing overhead. Time stamping tends to demand considerable system resources because many transactions have to be stopped, rescheduled, and re-stamped.

12.4.1 Wait/Die and Wound/Wait Schemes

There are two schemes used to decide which transaction is rolled back and which continues executing.

Wait/Die Scheme

The older transaction waits, and the younger is rolled back and rescheduled.

- If the transaction requesting the lock is the older of the two transactions, it will *wait* until the other transaction is completed and the locks are released.
- If the transactions requesting the lock is the younger of the two transactions, it will *die* (roll-back) and is rescheduled using the same time stamp.

Wound/Wait Scheme

The older transaction rolls back the younger transaction and reschedules it.

- If the transaction requesting the lock is the older of the two transactions, it will pre-empt (*wound*) the younger transaction (by rolling it back). T_1 pre-empts T_2 when T_1 rolls back T_2. The younger preempted transaction is rescheduled using the same time stamp.
- If the transaction requesting the lock is the younger of the two transactions, it will wait until the other transaction is completed and the locks are released.

12.5 Concurrency Control with Optimistic Methods

The **optimistic approach** is based on the assumption that the majority of the database operations do not conflict. This approach does not require locking or time stamping techniques. Instead, a transaction is executed without restrictions until it is committed. Using an optimistic approach, each transaction moves through two or three phases:

- **Read** The transaction reads the database, executes the needed computations, and makes the updates to a private copy of the database values. All update operations of the transactions are recorded in a temporary update file, which is not accessed by the remaining transactions.
- **Validation** The transaction is validated to ensure that the changes made will not affect the integrity and consistency of the database. If the validation test is positive, the transaction goes to the Write phase. If the validation test is negative, the transaction is restarted, and the changes are discarded.
- **Write** The changes are permanently applied to the database.

The optimistic approach is acceptable for most read or query database systems that require few update transactions.

12.6 ANSI Levels of Transaction Isolation

Transaction isolation levels refer to the degree to which transaction data is 'protected or isolated' from other concurrent transactions. The isolation levels are described based on which data other transactions can see (read) during execution. More precisely, the transaction isolation levels are described by the type of 'read' that a transaction allows or not. The types of read operations are:

- **Dirty Read** A transaction can read data that is not yet committed.
- **Non-Repeatable Read** A transaction reads a given row at time t_1, and then it reads the same row at time t_2, yielding different results. The original row may have been updated or deleted.
- **Phantom Read** A transaction executes a query at time t_1, and then it runs the same query at time t_2, yielding additional rows that satisfy the query.

There are four levels of transaction isolation:

1 **Read Uncommitted** Will read uncommitted data from other transactions. At this isolation level, the database does not place any locks on the data, which increases transaction performance, but at the cost of data consistency. Allows all three types of read operations.

2 **Read Committed** Forces transactions to read only committed data. The default mode of operation for most databases. The database will use exclusive locks on data, causing other transactions to wait until the original transaction commits. Does not allow uncommitted data reads, but allows repeatable reads, and phantom reads.

3 **Repeatable Read** Ensures that queries return consistent results. Uses shared locks to ensure that other transactions do not update a row after the original query reads it. However, new rows are read (phantom read) as these rows did not exist when the first query ran. Only allows phantom reads, of the three read operations.

4 **Serialisable** The most restrictive level defined by the ANSI SQL standard. Even at this level, deadlocks are still possible. Most databases use a deadlock detection approach to transaction management, and will therefore detect deadlocks during the transaction validation phase, and reschedule the transaction. Doesn't allow any of the three read operations.

The reason for the different levels of isolation is to increase transaction concurrency. The isolation levels go from least restrictive to most restrictive. The higher the isolation level, the more locks (shared and exclusive) are required to improve data consistency, at the expense of transaction concurrency performance.

12.7 Database Recovery Management

Database recovery restores a database from a given state (usually inconsistent) to a previously consistent state. Recovery techniques are based on the **atomic transaction property**: all portions of the transaction must be treated as a single, logical unit of work in which all operations are applied and completed to produce a consistent database. If, for some reason, any transaction operation cannot be completed, the transaction must be aborted, and any changes to be database must be rolled back.

Examples of Critical Events

- Hardware/software failures.
- Human-caused incidents, both intentional and unintentional.
- Natural disasters.

12.7.1 Transaction Recovery

Database transaction recovery focuses on the different methods used to recover a database from an inconsistent to a consistent state by using the data in the transaction log.

Concepts Affecting the Recovery Process

- **Write-Ahead Log Protocol** This protocol ensures that logs are always written *before* any database data are actually updated. It ensures that, in case of a failure, the database can later be recovered to a consistent state, using the data in the transaction log.
- **Redundant Transaction Logs** Most DBMSs keep several copies of the transaction log to ensure that a physical disk failure will not impair the DBMS's ability to recover data.
- **Database Buffers** A **buffer** is a temporary storage area in primary memory used to speed up disk operations. To improve processing time, the DBMS software reads the data from the physical disk and stores a copy of it on a 'buffer' in primary memory. When a transaction updates data, it actually updates the copy of the data in the buffer, because that process is much faster. Later on, all buffers that contain updated data are written to a physical disk during a single operation, thereby saving significant processing time.
- **Database Checkpoints** A **database checkpoint** is an operation in which the DBMS writes all of its updated buffers to disk. While this is happening, the DBMS does not execute any other requests. A checkpoint operation is also registered in the transaction log. As a result of this operation, the physical database and the transaction log will be 'in sync'. This synchronisation is required because update operations update the copy of the data in the buffers and not in the physical database. Checkpoints are automatically scheduled by the DBMS several times per hour.

Transaction recovery procedures generally make use of deferred-write and write-through techniques.

Deferred Write

When the recovery procedure uses **deferred write** or **deferred update**, the transaction operations do not immediately update the physical database. Instead, only the transaction log is updated. The database is physically updated only after the transaction reaches its commit point, using the transaction log information. If the transaction aborts before it reaches its commit point, no changes need to be made to the database, because the database was never updated.

Deferred Write Recovery Process

The recovery process for all started and committed transactions follow these steps:

1. Identify the last checkpoint in the transaction log. This is the last time transaction data was physically saved to disk.
2. For a transaction that started and committed before the last checkpoint, nothing needs to be done, because the data are already saved.
3. For a transaction that performed a `COMMIT` operation after the last checkpoint, the DBMS uses the transaction log records to redo the transaction and to update the database, using the 'after' values in the transaction log. The changes are made in ascending order, from oldest to newest.
4. For any transaction that had a `ROLLBACK` operation after the last checkpoint, or that was left active before the failure occurred, nothing needs to be done, because the database was never updated.

Write Through (Immediate Update)

The database is immediately updated by transaction operations during the transaction's execution, even before the transaction reaches its commit point. It the transaction aborts before it reaches its commit point, a `ROLLBACK` or undo operation needs to be done to restore the database to a consistent state. In that case, the `ROLLBACK` uses the transaction log 'before' values.

Immediate Update Recovery Process

The recovery process follows these steps:

1. Identify the last checkpoint in the transaction log. This is the last time transaction data was physically saved to disk.
2. For a transaction that started and committed before the last checkpoint, nothing needs to be done, because the data are already saved.
3. For a transaction that performed a `COMMIT` operation after the last checkpoint, the DBMS uses the transaction log records to redo the transaction and to update the database, using the 'after' values in the transaction log. The changes are made in ascending order, from oldest to newest.
4. For any transaction that had a `ROLLBACK` operation after the last checkpoint, or that was left active before the failure occurred, the DBMS uses the transaction log records to `ROLLBACK` or undo the operations, using the 'before' values in the transaction log. Changes are applied in reverse order, from newest to oldest.

Chapter 13

Managing Database and SQL Performance

13.1 Database Performance-Tuning Concepts

One of the main functions of a database system is to provide answers to end users when they are required. End users and the DBMS interact through the use of queries, using the following sequence:

1. The end-user (client-end) application generates a query.
2. The query is sent to the DBMS (server-end).
3. The DBMS (server-end) executes the query.
4. The DBMS sends the resulting data set to the end-user (client-end) application.

The goal of database performance is to execute queries as fast as possible. **Database performance tuning** refers to a set of activities and procedures designed to reduce the response time of the database system, that is, to try to ensure that an end-user query is processed by the DBMS in the minimum amount of time.

The performance of a typical DBMS is constrained by three main factors: CPU processing power, available RAM, and I/O throughput. The system performs best when its hardware and software are optimised. Hardware and software should be optimised to obtain the best throughput possible with existing (and often limited) resources.

Fine-tuning the performance of a system requires a holistic approach: *all* factors must be checked to ensure that each one operates at its optimum level and has sufficient resources to minimise the occurrence of bottlenecks.

13.1.1 Performance Tuning: Client and Server

- **Client Side – SQL Performance Tuning** The objective is to generate a SQL query that returns the correct answer in the least amount of time, using the minimum amount of resources at the server end.
- **Server Side – DBMS Performance Tuning** The DBMS environment must be properly configured to respond to clients' requests in the fastest way possible, while making optimum use of existing resources.

13.1.2 DBMS Architecture

The architecture of a DBMS is represented by the processes and structures (in memory and in permanent storage) used to manage a database.

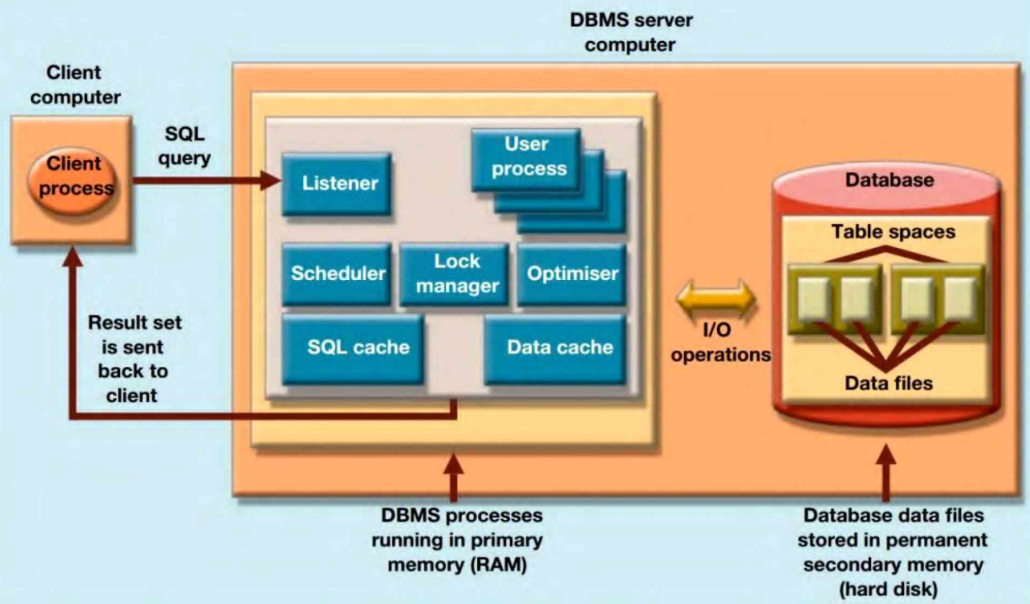


Figure 13.1: Basic DBMS Architecture

- All data in a database are stored in **data files**. A data file can contain rows from one single table, or from many different ones. A DBA determines the initial size of the data files that make up the database; but data files can automatically expand in predefined increments known as **extends**.
- Data files are grouped in file groups creating table spaces. A **table space** or **file group** is a logical grouping of several data files that store data with similar characteristics. Each time you create a new database, the DBMS automatically creates a minimum set of table spaces.
- To work with the data, the DBMS must retrieve the data from permanent storage (data files) and place it in RAM (data cache).
- The **data cache** or **buffer cache** is a shared, reserved memory area that stores the most recently accessed data blocks in RAM. This is where the data read from the database files are stored after the data have been read, or before data are written to the database data files. This cache also caches system catalogue data and the contents of the indexes.
- The **SQL cache** or **procedure cache** is a shared, reserved memory area that stores the most recently executed SQL statements or PL/SQL procedures, including triggers and functions. The SQL cache does not store the end-user written SQL, but rather a 'processed' version of the SQL that is ready for execution by the DBMS.
- To move data from the data files to the data cache, the DBMS issues I/O requests and waits for the replies. An **input/output (I/O) request** is a low-level data access operation to or from computer devices. The purpose of the I/O operation is to move data to and from different computer components or devices. An I/O disk operation retrieves an entire physical disk block from permanent storage to the data cache, even if you only use one attribute from one row. The physical block size depends on the operating system. Depending on the situation, a DBMS might issue a single-block read request, or a multi-block read request.

- Working with data in the data cache is many times faster than working with data in the data files, because the DBMS doesn't have to wait for the hard disk to retrieve the data. (That's because no I/O operations are needed to work within the data cache.)
- The majority of performance-tuning activities focus on minimising the number of I/O operations, because using I/O operations is many times slower than reading data from the data cache.

Processes involved in a DBMS include:

- **Listener** Listens for clients' requests and hands the processing of the SQL requests to other DBMS processes. Once a request is received, the listener passes the request to the appropriate user process.
- **User** The DBMS creates a user process to manage each client session. Therefore, when you log on to the DBMS, you are assigned a user process. This process will handle all requests you submit to the server. There are many user processes, at least one per logged-in client.
- **Scheduler** Schedules the concurrent execution of SQL requests.
- **Lock Manager** Manages all locks placed on database objects.
- **Optimiser** Analyses SQL queries and finds the most efficient way to access the data.

13.1.3 Database Query Optimisation Modes

Most of the algorithms for query optimisation are based on two principles:

- The selection of the optimum execution order, and
- The selection of sites to be accessed to minimise communication costs.

With those two principles, a query optimisation algorithm can be evaluated based on its operation mode or the timing of its optimisation. Optimisation modes can be classified as manual or automatic:

- **Automatic Query Optimisation** The DBMS finds the most cost-effective access path without user intervention.
- **Manual Query Optimisation** Requires that the optimisation be selected and scheduled by the end user or programmer.

Automatic query optimisation is more desirable from the end-user's point of view, but the cost is the increased overhead that it imposes on the DBMS.

Query optimisation algorithms can also be classified according to when the optimisation is done:

- **Static Query Optimisation** Takes place at compilation time. The best optimisation strategy is selected when the query is compiled by the DBMS. This approach is common when SQL statements are embedded in procedural programming languages. When the program is submitted to the DBMS for compilation, it creates the plan necessary to access the database. When the program is executed, the DBMS uses that plan to access the database.
- **Dynamic Query Optimisation** Takes place at execution time. The database access strategy is defined when the program is executed. So, the access strategy is dynamically determined by the DBMS at run time, using the most up-to-date information about the database. Although dynamic query optimisation is efficient, its cost is measured by run-time processing overhead. The best strategy is determined every time the query is executed, which could happen several times in the same program.

Query optimisation techniques can be classified according to the type of information that is used to optimise the query:

- **Statistically Based Query Optimisation** Uses statistical information about the database. The statistics provide information about database characteristics such as size, number of records, average access time, number of requests serviced, and number of users with access rights. These statistics are then used by the DBMS to determine the best access strategy. With statistically based optimisers, some DBMSs allow setting a goal to specify that the optimiser should attempt to minimise the time to retrieve the first row or the last row.

  Minimising the time to retrieve the first row is often used in transaction systems and interactive client environments. In these cases, the goal is to present the first several rows to the user as quickly as possible. Then, while the DBMS waits for the user to scroll through the data, it can fetch the other rows for the query.

  Setting the optimiser goal to minimise retrieval of the last row is typically done in embedded SQL and inside stored procedures. In these cases, the control will not pass back to the calling application until all of the data have been retrieved; therefore, it is important to retrieve all of the data to the last row as quickly as possible so control can be returned.

  The statistical information is managed by the DBMS and is generated in one of two different modes:

  - **Dynamic Statistical Generation Mode** DBMS automatically evaluates and updates the statistics after each access.
  - **Manual Statistical Generation Mode** Statistics must be updated periodically through a user-selected utility which is specific to the particular DBMS.
- **Rule-Based Query Optimisation** Based on a set of user-defined rules to determine the best query access strategy. The rules are entered by the end user or database administrator, and are typically general in nature.

13.1.4 Database Statistics

Database statistics refers to a number of measurements about database objects such as tables, indexes, and available resources such as number of processors used, processor speed, and temporary space available. These statistics give a snapshot of database characteristics.

Database statistics can be gathered on request by the DBA or automatically by the DBMS.

Sample Database Statistics Measurements

Table Number of rows, number of disk blocks used, row length, number of columns in each row, number of distinct values in each column, maximum value in each column, minimum value in each column, columns that have indexes.

Indexes Number and name of columns in the index key, number of key values in the index, number of distinct key values in the index key, histogram of key values in an index.

Environment Resources Logical and physical disk block size, location and size of data files, number of extends per data file.

If the object statistics exist, the DBMS uses them in query processing. Although newer DBMSs automatically gather statistics, older ones require the DBA to gather statistics on request, using the syntax:

```
ANALYZE <TABLE/INDEX> object_name COMPUTE STATISTICS;
```

When you generate statistics for a table, all related indexes are also analysed.

Database statistics are stored in the system catalogue in specially designated tables. It is common to periodically regenerate the statistics for database objects, especially those that are subject to frequent change. The more current the statistics, the better the changes are for the DBMS to find the fastest way to execute a given query.

13.2 Query Processing

The DBMS processes queries in three phases:

1 Parsing The DBMS parses the SQL query and chooses the most efficient access/execution plan.

2 Execution The DBMS executes the SQL query, using the chosen execution plan.

3 Fetching The DBMS fetches the data and sends the result set back to the client.

The processing of DDL statements is different from the processing required by DML statements. The different is that a DDL statement actually updates the data dictionary tables or system catalogue, while a DML statement mostly manipulates end-user data.

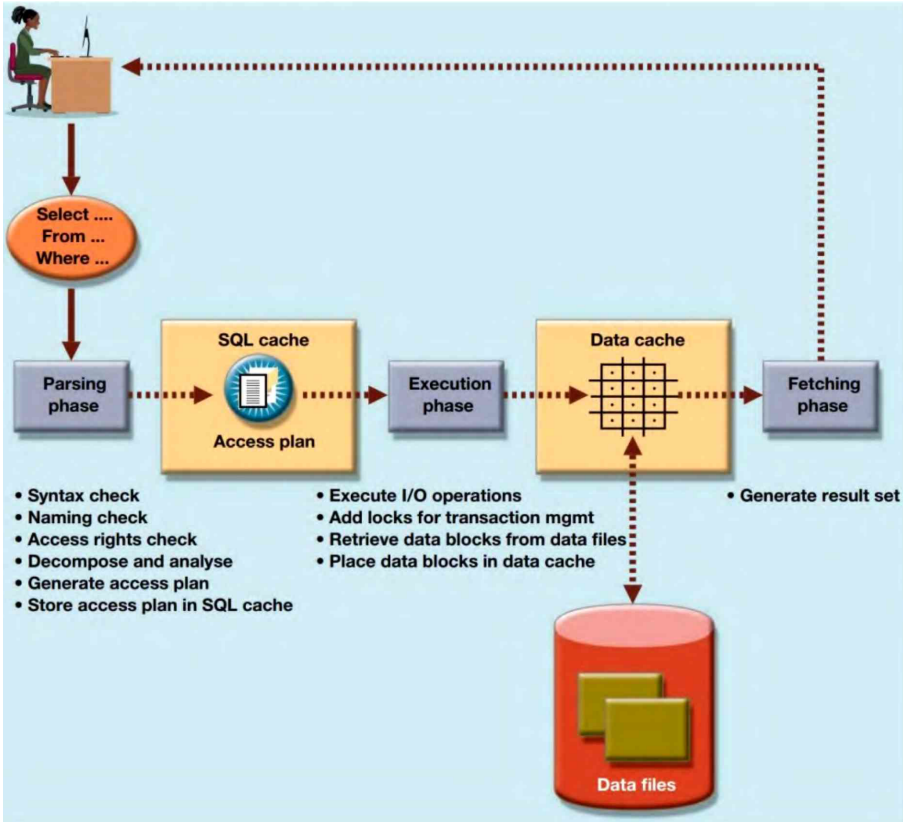


Figure 13.2: Query processing

13.2.1 SQL Parsing Phase

The optimisation process includes breaking down – parsing – the query into smaller units and transforming the original SQL query into a slightly different version of the original SQL code, but one that is fully equivalent and more efficient:

- **Fully equivalent** The optimised query results are always the same as the original query.
- **More efficient** The optimised query will almost always execute faster than the original query.

To determine the most efficient way to execute the query, the DBMS may use the database statistics.

The SQL parsing activities are performed by the query optimiser. The **query optimiser** analyses the SQL query and finds the most efficient way to access the data. This process is the most time-consuming phase in query processing.

Parsing a SQL query requires several steps. The SQL query is

- Validated for syntax compliance
- Validated against the data dictionary to ensure that tables and column names are correct
- Validated against the data dictionary to ensure that the user has proper access rights
- Analysed and decomposed into more atomic components
- Optimised through transformation into a fully equivalent but more efficient SQL query.
- Prepared for execution by determining the most efficient execution or access plan.

Once the SQL statement is transformed, the DBMS creates an access or execution plan. An **access/execution plan** contains the series of steps a DBMS uses to execute the query and return the result set in the most efficient way. First, the DBMS checks to see if an access plan for the query already exists in the SQL cache. If it does, the DBMS reuses the access plan to save time. If it doesn't, the optimiser evaluates different plans and makes decisions about which indexes to use and how best to perform join operations. The chosen access plan for the query is then placed in the SQL cache and made available for use and future reuse.

Access plans are DBMS-specific, and translate the client's SQL query into the series of complex I/O operations required to read the data from the physical data files and generate the result set.

Sample DBMS Access Plan I/O Operations

Although access plans are DBMS specific, there are some commonly found I/O operations.

- **Table Scan (Full)** Reads the entire table sequentially, from the first row to the last row, one row at a time (slowest).
- **Table Access (Row ID)** Reads a table row directly, using the row ID value (fastest).
- **Index Scan (Range)** Reads the index first to obtain the row IDs and then accesses the table rows directly (faster than a full table scan).
- **Index Access (Unique)** Used when a table has a unique index in a column.
- **Nested Loop** Reads and compares a set of values to another set of values, using a nested loop style (slow).
- **Merge** Merges two data sets (slow).
- **Sort** Sorts a data set (slow).

A table access using a row ID is the fastest method. A row ID is a unique identification for every row saved in permanent storage, and can be used to access the row directly.

13.2.2 SQL Execution Phase

In this phase, all I/O operations indicated in the access plan are executed. When the execution plan is run, the proper locks are – if needed – acquired for the data to be accessed, and the data are retrieved from the data files and placed in the DBMS cache. All transaction management commands are processed during the parsing and execution phases of query processing.

13.2.3 SQL Fetching Phase

After the parsing and execution phases are completed, all rows that match the specified condition(s) are retrieved, sorted, grouped, and/or (if required) aggregated. During the fetching phase, the rows of the resulting query set are returned to the client. During this phase, the DBMS mat use temporary table space to store temporary data.

13.2.4 Query Processing Bottlenecks

The main objective of query processing is to execute a given query in the fastest way possible with the fewest resources. The execution of a query requires the DBMS to break down the query into a set of interdependent I/O operations to be executed in a collaborative manner. The more complex a query is, the more complex the operations are, which means that bottlenecks are more likely. The more components a system has, the more interfacing is required among the components, which also increases the likelihood of bottlenecks.

A **query processing bottleneck** is a delay introduced in the processing of an I/O operation that causes the overall system to slow down.

Within a DBMS, five components typically cause bottlenecks:

- **CPU** The CPU processing power of the DBMS should match the system's expected workload. A high CPU utilisation might indicate that the processor is too slow for the amount of work performed. However, heavy CPU utilisation can be caused by other factors, such as a defective component, not enough RAM, a badly written device driver, or a rogue process. A CPU bottleneck will affect not only the DBMS, but all processes running in the system.
- **RAM** The DBMS allocates memory for specific usage, such as data cache and SQL cache. RAM must be shared among all running processes, including the OS and DBMS. If there is not enough RAM available, moving data among components that are competing for sparse RAM can create a bottleneck.
- **Hard Disk** Hard disk space is used for more than just storing end-user data. It is also used for virtual memory, which refers to copying areas of RAM to the hard disk as needed to make room in RAM for more urgent tasks. Therefore, the more hard disk storage space available, and the ability to have faster data transfer rates, reduce the likelihood of bottlenecks.
- **Network** In a database environment, the database server and clients are connected via a network. All networks have a limited amount of bandwidth that is shared among all clients. When many network nodes access the network at the same time, bottlenecks are likely.
- **Application Code** Two of the most common sources of bottlenecks are inferior application code and poorly designed databases. Inferior code can be improved with code optimisation techniques, as long as the underlying database design is sound.

13.3 Optimiser Choices

Query optimisation is the central activity during the parsing phase of query processing. In this phase, the DBMS must choose what indexes to use, how to perform join operations, which table to use first, and so on. Each DBMS has its own algorithms for determining the most efficient way to access the data.

The query optimiser can operate in one of two modes:

- **Rule-Based Optimiser** Uses a set of preset rules and points to determine the best approach to execute a query. The rules assign a 'fixed cost' to each SQL operation; the costs are then added to yield the cost of the execution plan.
- **Cost-Based Optimiser** Uses sophisticated algorithms based on the statistics about the objects being accessed to determine the best approach to execute a query. The optimiser process adds up the processing cost, the I/O costs, and the resource costs to come up with the total cost of a given execution plan.

The objective for the optimiser is to find alternative ways to execute a query, to evaluate the 'cost' of each alternative, and then to choose the one with the lowest cost.

13.3.1 Using Hints to Affect Optimiser Choices

Although the optimiser generally performs well under most circumstances, in some circumstances the optimiser may not choose the best execution plan. The optimiser makes decisions based on the existing statistics. If the statistics are old, the optimiser might not select the best execution plan. Even with current statistics, the optimiser choice may not be the most efficient one.

In some cases, the end user would like to change the optimiser mode for the current SQL statement. In order to do that, you need to use hints. **Optimiser hints** are special instructions for the optimiser that are embedded inside the SQL command text.

Optimiser Hints

- **ALL_ROWS** Instructs the optimiser to minimise the overall execution time, that is, to minimise the time it takes to return all rows in the query result set. Generally used for batch mode processes.

```
SELECT /*+ ALL_ROWS */ * FROM PRODUCT WHERE P_QOH < 10;
```

- **FIRST_ROWS** Instructs the optimiser to minimise the time it takes to process the first set of rows, that is, to minimise the time it takes to return only the first set of rows in the query result set. Generally used for interactive mode processes.

```
SELECT /*+ FIRST_ROWS */ * FROM PRODUCT WHERE P_QOH < 10;
```

- **INDEX(name)** Force the optimiser to use a particular index to process a query.

```
SELECT /*+ INDEX(P_QOH_NDX) */ * FROM PRODUCT WHERE P_QOH < 10;
```

13.4 SQL Performance Tuning

SQL performance tuning is evaluated from the client perspective.

- Most current-generation relational DBMSs perform automatic query optimisation at the server end.
- Most SQL performance optimisation techniques are DBMS-specific and are therefore rarely portable, even across different versions of the same DBMS.

13.4.1 Index Selectivity

Indexes are the most important technique used in SQL performance optimisation. **Index selectivity** is a measure of how likely an index is to be used in query processing. However, you cannot always use an index to improve performance.

Indexes are ignored when you use functions in the table attributes.

You should not create an index for every column in a table, as too many indexes will slow down `INSERT`, `UPDATE` and `DELETE` operations, especially if the table contains many thousands of rows. Furthermore, some query optimisers will choose only one index for a query, even if the query uses conditions in many different indexed columns.

13.4.2 Conditional Expressions

A conditional expression is normally expressed within the `WHERE` or `HAVING` clauses of a SQL statement. A conditional expression restricts the output of a query to only the rows matching the conditional criteria.

- Use simple columns or literals as operands in a conditional expression – avoid the use of conditional expressions with functions whenever possible.
- Numeric field comparisons are faster than character, date, and `NULL` comparisons. As indexes do not store references to null values, `NULL` conditions involve additional processing, and tend to be the slowest of all conditional operands.
- Equality comparisons are faster than inequality comparisons. The slowest (with the exception of `NULL`) of all comparison operators is `LIKE` with wildcard symbols.
- Whenever possible, transform conditional expressions to use literals.
- When using multiple conditional expressions, write the equality conditions first.
- If you use multiple `AND` conditions, write the condition most likely to be false first.
- When using multiple `OR` conditions, write the condition most likely to be true first.
- Whenever possible, try to avoid the use of the `NOT` logical operator.

13.5 DBMS Performance Tuning

DBMS performance tuning includes global tasks such as managing the DBMS processes in primary memory (allocating memory for caching purposes) and the structures in physical storage (allocating space for the data files).

DBMS performance tuning at the server end focusses on setting the parameters used for:

- **Data Cache** The data cache must be set large enough to permit as many data requests to be serviced from the cache as possible. Each DBMS has settings that control the size of the data cache; this cache is shared among all database users. The majority of primary memory resources are allocated to the data cache.
- **SQL Cache** Stores the most recently executed SQL statements (after the SQL statements have been parsed by the optimiser). If different users want to execute the same query, the DBMS only parses the query once, and the requests are then served from the SQL cache, skipping the parsing phase.
- **Sort Cache** Used as a temporary storage area for `ORDER BY` or `GROUP BY` operations, as well as for index-creation functions.
- **Optimiser Mode** Most DBMSs operate in one of two optimisation modes: cost-based or rule-based.

From the performance point of view, it would be optimal to have the entire database stored in primary memory, to minimise costly disk access. This is why in-memory database options exist.

In-Memory Database

Systems that are optimised to store large portions (if not all) of the database in primary (RAM) storage rather than secondary (disk) storage.

These systems are becoming popular because of increased performance demands from modern database applications, diminishing costs, and technology advancements of components. Even though these databases 'eliminate' disk access bottlenecks, they are still subject to query optimisation and performance tuning rules.

Recommendations for Physical Storage of Databases

- Use **I/O accelerators**. This type of device uses flash SSDs to store the database. An SSD does not have any moving parts, and therefore performs I/O operations at a higher speed than traditional disk drives. I/O accelerators deliver high transaction performance rates and reduce contention caused by typical storage drives.
- Use **RAID** to provide balance between performance and fault tolerance.
- Minimise disk contention. Use multiple, independent storage volumes with independent spindles to minimise hard disk cycles.

- A database should have at least the following table spaces:
  - **System Table Space** Stores the data dictionary tables. The most frequently accessed table space, and should be stored in its own volume.
  - **User Data Table Space** Stores end-user data. You should create as many user table spaces and data files as are required.
  - **Index Table Space** Stores indexes. The index table space data files should be stored on a storage volume that is separate from user data files or system data files.
  - **Temporary Table Space** Used as a temporary storage area for merge, sort, or set aggregate operations.
  - **Rollback Segment Table Space** Used for transaction recovery purposes.
- Put high-usage tables in their own table spaces. By doing this, the database minimises conflict with other tables.
- Take advantage of various table storage organisations available in the database.

  In Oracle, consider the use of index-organised tables. An **index-organised table (IOT)** is a table that stores the end-user data and the index data in consecutive locations on permanent storage. This type of storage organisation provides a performance advantage to tables that are commonly accessed though a given index order, because the index contains the index key as well as the data rows. Therefore, the DBMS tends to perform fewer I/O operations.
- Assign separate files in separate storage volumes for the indexes, system and high-usage tables. This ensures that index operations will not conflict with end-user data or data dictionary table access operations.
- Partition tables based on usage.
- Use denormalised tables where appropriate.
- Store computed and aggregated attributes in tables. That is, use derived attributes. Using derived attributes minimises computations in queries and join operations.

Chapter 14

Distributed Databases

A single database can be divided into several fragments. The fragments can be stored on different computers within a network. Processing can also be dispersed among different network sites, or nodes.

The distributed database management system (DDBMS) treats a distributed database as a single logical database. But, the distribution among different sites in a computer network adds complexity.

14.1 The Evolution of Distributed Database Management Systems

A **distributed database management system (DDBMS)** governs the storage and processing of logically related data over interconnected computer systems in which both data and processing functions are distributed among several sites.

The use of a centralised database required that corporate data be stored in a single central site, usually a mainframe or midrange computer. Data access was provided through dumb terminals. The centralised approach worked well to fill the structured information needs of corporations, but fell short when quickly moving events required faster response times and equally quick access to information. What was needed was quick, unstructured access to databases, using ad hoc queries to generate on-the-spot information. Database management systems based on the relational model could provide the environment to do this.

Problems with Centralised Database Management

- **Performance Degradation** Due to a growing number of remote locations over greater distances.
- **High Costs** Associated with maintaining and operating large central (mainframe) database systems.
- **Reliability Problems** Created by dependence on a central site (single point of failure) and the need for data replication.
- **Scalability Problems** Associated with the physical limits imposed by a single location, such as physical space, temperature conditioning, and power consumption.
- **Organisational Rigidity** Imposed by the database, which means it might not support the flexibility and agility required by modern global organisations.

14.2 DDBMS Advantages and Disadvantages

Advantages of a DDBMS

- **Data are located near the greatest demand site** The data in a DDBMS are dispersed to match business requirements.
- **Faster data access** End users often work with only a locally stored subset of the data.
- **Faster data processing** A DDBMS spreads out the system workload by processing data at several sites.
- **Growth Facilitation** New sites can be added to the network without affecting the operations of other sites.
- **Improved Communications** Because local sites are smaller and located closer to customers, local sites foster better communication among departments and between customers and company staff.
- **Reduced Operating Costs** It is more cost-effective to add workstations to a network than to update a mainframe system. Development work is done more cheaply and more quickly on low-cost PCs than on mainframes.
- **User-Friendly Interface** PCs and workstations are usually equipped with an easy-to-use GUI. The GUI simplifies use and training for end users.
- **Less Danger of a Single Point Failure** When one of the computers fails, the workload is picked up by other workstations. Data are also distributed at multiple sites.
- **Processor Independence** The end user is able to access any available copy of the data, and an end user's request is processed by any processor at the data location.

Disadvantages of a DDBMS

- **Complexity of Management and Control** Applications must recognise data location, and they must be able to 'stitch' together data from different sites. DBAs must be able to coordinate database activities to prevent database degradation due to data anomalies.
- **Security** The problem of security lapses increases when data are located at multiple sites. The responsibility of data management will be shared by different people at different sites.
- **Lack of Standards** There are no standard communication protocols *at the database level*. For example, different database vendors employ different and often incompatible techniques to manage the distribution of data and processing in a DDBMS environment.
- **Increased Storage Requirements** Multiple copies of data are required at different sites.
- **Increased Training Costs** Training costs are generally higher in a distributed model than they would be in a centralised model.
- **Higher Costs** Distributed databases require duplicated infrastructure to operate, such as physical location, environment, personnel, software, and licensing.

14.3 Distributed Processing and Distributed Databases

In **distributed processing**, a database's logical processing is shared among two or more physically independent sites that are connected through a network. For example, the data I/O, data selection, and data validation might be performed on one computer, and a report based on that data might be created on another computer. The database is located at a single location, called the **database server**. This system uses only a single-site database, but shares the processing chores among several sites.

A **distributed database** stores a logically related database over two or more physically independent sites. The sites are connected via a computer network. A database is composed of several parts known as **database fragments**. The database fragments are located at different sites, and can be replicated among various sites.

- Distributed processing does not require a distributed database, but a distributed database requires distributed processing.
- Distributed processing may be based on a single location based on a single computer. For the management of distributed data to occur, copies or parts of the database processing functions must be distributed to all data storage sites.
- Both distributed processing and distributed databases require a network of interconnected components.

14.4 Characteristics of Distributed Database Management Systems

A DBMS must have at least the following functions to be classified as distributed:

- **Application Interface** To interact with the end user or application program, and with other DBMSs within the distributed database.
- **Validation** To analyse data requests.
- **Transformation** To determine which data request components are distributed and which are local.
- **Query Optimisation** To find the best access strategy.
- **Mapping** To determine the data location of local and remote fragments.
- **I/O Interface** To read or write data to or from permanent local storage.
- **Formatting** To prepare the data for presentation to the end user or to an application program.
- **Security** To provide data privacy at both local and remote databases.
- **Backup and Recovery** To ensure the availability and recoverability of the database in case of a failure.
- **DB Administration** Features for the data administrator.
- **Concurrency Control** To manage simultaneous data access and to ensure data consistency across database fragments in the DDBMS.
- **Transaction Management** To ensure that data moves from one consistent state to another. This includes the synchronisation of local and remote transactions as well as transactions across multiple distributed segments.

Functions of a DDBMS

A fully distributed DBMS must perform all of the functions of a centralised DBMS:

1. Receive an application's (or an end user's) request.
2. Validate, analyse, and decompose the request.
3. Map the request's logical-to-physical components.
4. Decompose the request into several disk I/O operations.
5. Search for, locate, read, and validate the data.
6. Ensure database consistency, security, and integrity.
7. Validate the data for the conditions, if any, specified by the request.
8. Present the selected data in the required format.

In addition, a DDBMS must handle all necessary functions imposed by the distribution of data and processing. And it must perform those additional functions *transparently* to the end user.

14.5 DDBMS Components

DDBMS Components

The DDBMS must include at least the following components:

- **Computer Workstations** (sites or nodes) that form the network system. The DDBMS must be independent of the computer system hardware.
- **Network Hardware and Cofware Components** Reside in each workstation. The network components allow all sites to interact and exchange data. As the computers are likely to be supplied by different vendors, it is best to ensure that distributed database functions can be run on multiple platforms.
- **Communications Media** Carry the data from one workstation to another. The DDBMS must be communications-media-independent: it must be able to support several types of communications media.
- **Transaction Processor (TP)** The software component found in each computer that requests data. The transaction processor receives and processes the application's data requests (remote and local). The TP is also known as the **application processor (AP)** or the **transaction manager (TM)**.
- **Data Processor (DP)** The software component residing on each computer that stores and retrieves data located at the site. Also known as the **data manager (DM)**. A data processor may even be a centralised DBMS.

Communication among transaction processors and data processors is made possible through a specific set of rules, or **protocols**, used by the DDBMS.

The protocols determine how the distributed database system will:

- Interface with the network to transport data between DPs and TPs.
- Synchronise all data received from DPs (TP side) and route retrieved data to the appropriate TPs (DP side).
- Ensure common database functions in a distributed system. Such functions include security, concurrency control, backup and recovery.

DPs and TPs can be added to the system without affecting the operation of the other components. A TP and DP can reside on the same computer, allowing the end user to access local as well as remote data transparently.

14.6 Levels of Data and Process Distribution

Current database systems can be classified on the basis of how process distribution and data distribution are supported. A DBMS may store data in a single site (centralised DB) or in multiple sites (distributed DB) and may support data processing at a single site or at multiple sites.

A variation of the multiple-site processing, single-site data approach is known as **client/server architecture**. Client/server architecture is similar to that of the network file server, except that all database processing is done at the server site, thus reducing network traffic. Although both the network file server and client/server systems perform multiple-site processing, the latter's processing is distributed. The network file server approach requires the database to be located at a single site, whereas client/server is capable of supporting data at multiple sites.

14.6.1 Multiple-Site Processing, Multiple-Site Data (MPMD)

The **multiple-site processing, multiple-site data (MPMD)** scenario describes a fully distributed DBMS wit with support for multiple data processors and transaction processors at multiple sites. Depending on the level of support for different types of centralised DBMSs, DDBMSs are classified as either homogeneous or heterogeneous.

- **Homogeneous DDBMS** Integrates multiple instances of the same database over a network. Thus, the same DBMS will be running on different mainframes, minicomputers, and microcomputers.
- **Heterogeneous DDBMS** Integrates different types of centralised DBMSs over a network. A **fully heterogeneous DDBMS** will support different DDBMSs that may even support different data models (relational, hierarchical, or network) running over a network.

No DDBMS currently provides full support for the fully heterogeneous environment. Some DDBMS implementations support several platforms, operating systems and networks, and allow remote access to another DBMS. However, such DBMSs still are subject to certain restrictions, for example:

- Remote access is on a read-only basis and does not support write privileges.
- Restrictions are placed on the number of remote tables that may be accessed in a single transaction.
- Restrictions are placed on the number of distinct databases that may be accessed.
- Restrictions are placed on the database model that may be accessed.

14.7 Distributed Database Transparency Features

DDBMS transparency features allow the end user to feel like the database's only user. The user believes they are working with a centralised DBMS; all complexities of a distributed database are hidden, or transparent to the user.

DDBMS Transparency Features

- **Distribution Transparency** Allows a distributed database to be treated as a single logical database. If a DDBMS exhibits distribution transparency, the user does not need to know:
  - That the data are partitioned, meaning the table's rows and columns are split vertically or horizontally, and stored in multiple sites.
  - That the data are geographically dispersed among multiple sites.
  - That the data are replicated among multiple sites.
- **Transaction Transparency** Allows a transaction to update data at several network sites. Ensures that the transaction will be either entirely completed or aborted, thus maintaining database integrity.
- **Failure Transparency** Ensures the system will continue to operate in the event of a node failure. Functions that were lost because of the failure will be picked up by another network node. This is a critical feature, especially in organisations that depend on a Web presence as the backbone for maintaining trust in their business.
- **Performance Transparency** Allows the system to perform as if it were a centralised DBMS. The system will not suffer any performance degradation due to its use of a network or due to the network's platform differences. Performance transparency also ensures that the system will find the most cost-effective path to access remote data. The systems should be able to scale out in a transparent manner or increase performance capacity by adding more transaction or data processing nodes, without affecting the overall performance of the system.
- **Heterogeneity Transparency** Allows the integration of several different local DBMSs under a common, or global, schema. The DDBMS is responsible for translating the data requests from the global schema to the local DBMS schema.

14.8 Distribution Transparency

Distribution transparency allows a physically dispersed database to be managed as though it were a centralised database. The level of transparency supported by the DDBMS varies from system to system. Three levels of distribution transparency are recognised:

- **Fragmentation Transparency** Highest level of transparency. The end user or programmer does not need to know that a database is partitioned. Therefore, neither fragment names not fragment locations are specified prior to data access.
- **Location Transparency** Exists when the end user or programmer must specify the database fragment names, but does not need to specify where those fragments are located.
- **Local Mapping Transparency** Exists when the end user or programmer must specify both the fragment names and their locations.

The **unique fragment** condition indicates that each row is unique, regardless of the fragment in which it is located.

Distribution transparency is supported by a **distributed data dictionary (DDD)** or a **distributed data catalogue (DDC)**. The DDC contains the description of the entire database as seen by the database administrator. The database description, known as the **database global schema**, is the common database scheme used by local TPs to translate user requests into subqueries (remote requests) that are processed by different DPs. The DDC itself is distributed, and it is replicated at the network nodes. Therefore, the DDC must maintain consistency through updating at all sites.

14.9 Transaction Transparency

Transaction transparency ensures that database transactions maintain the distributed database's integrity and consistency. It ensures that transactions are completed only when all database sites involved in the transaction complete their part of the transaction.

14.9.1 Distributed Requests and Distributed Transactions

Whether or not a transaction is distributed, it is formed by one or more database requests. The basic difference between a non-distributed transaction and a distributed transaction is that the latter can update or request data from several different remote sites on a network.

A **remote request** lets a single SQL statement access the data that are to be processed by a single database processor. In other words, the SQL statement (or request) can reference data at only one remote site. Similarly, a **remote transaction**, composed of several requests, accesses data at a single remote site.

- The transaction can reference only one remote DP.
- Each SQL statement (or request) can reference only one (the same) remote DP at a time, and the entire transaction can reference and be executed at only one remote DP.

A **distributed transaction** allows a transaction to reference several different local or remote DP sites. Although each single request can reference only one local or remote DP site, the transaction as a whole can reference multiple DP sites, because each request can reference a different site.

- Each request can access only one remote site at a time.

A **distributed request** lets a single SQL statement reference data located at several different local or remote DP sites. Because each request (SQL statement) can access data from more than one local or remote DP site, a transaction can access several sites. The ability to execute a distributed request provides fully distributed database processing capabilities because of the ability to:

- partition a database into several fragments, and
- reference one or more of those fragments with only one request. In other words, there is fragmentation transparency.

The distributed request feature also allows a single request to reference a physically partitioned table. Full fragmentation transparency support is provided only by a DBMS that supports distributed requests.

14.9.2 Distributed Concurrency Control

Concurrency control becomes especially important in the distributed database environment because multi-site, multiple-process operations are more likely to create data inconsistencies and deadlocked transactions.

Suppose each transaction operation was committed by each local DP, but one of the DPs could not commit the transaction's results. The transaction(s) would yield an inconsistent database, with its integrity problems. The solution to this is a **two-phase commit protocol**.

14.9.3 Two-Phase Commit Protocol

Centralised databases require only one DP. All database operations take place at only one site, and the consequences of database operations are immediately known to the DBMS. In contrast, distributed databases make it possible for a transaction to access data at several sites. A final `COMMIT` must not be issued until all sites have committed their parts of the transaction. The **two-phase commit protocol** guarantees that, if a portion of the transaction operation cannot be committed, all changes made at the other sites participating in the transaction will be undone to maintain a consistent database state.

14.10 Performance and Failure Transparency

Web-based distributed data systems demand high availability, which means not only that data are accessible, but that requests are processed in a timely manner. Performance transparency allows a DDBMS to perform as if it were a centralised database. Failure transparency ensures that the system will continue to operate in the case of a node or network failure. Although these are two separate issues, they are interrelated in that a failing node or congested network path could cause performance problems.

The objective of a query optimisation routine is to minimise the total cost associated with the execution of a request. The costs associated with a request are a function of the

- access time (I/O) cost involved in accessing the physical data stored on disk
- communication cost associated with the transmission of data among nodes in distributed database systems.
- CPU time cost associated with the processing overhead of managing distributed transactions.

Resolving data requests in a distributed environment must consider:

- **Data Distribution** In a DDBMS, query translation is more complicated because the DDBMS must decide which fragment to access. In this case, a TP executing a query must choose what fragments to access, create multiple data requests to the chosen remote DPs, combine the DP responses, and present the data to the application.
- **Data Replication** In addition, the data may also be replicated at several different sites.

  This replication makes the access problem more complex because the database must ensure that all copies of the data are consistent. Therefore, an important characteristic of query optimisation in DDBMSs is that it must provide replica transparency. **Replica transparency** refers to the DDBMS's ability to hide multiple copies of data from the user.

Replica transparency is particularly important with data update operations. If a read-only request is being processed, it can be satisfied by accessing any available remote DP. However, processing a write request also involves synchronising all existing fragments to maintain data consistency. If data are replicated at other sites, the DBMS must also ensure the consistency of all the fragments – all fragments should be mutually consistent. To accomplish this, a DP captures all changes and pushes them to each remote replica. This introduces delays in the system and basically means that not all data changes are immediately seen by all replicas.

- **Network and Node Availability** The response time associated with remote sites cannot be easily predetermined because some nodes finish their part of the query in less time than others and network path performance varies because of the bandwidth and traffic loads. Hence, to achieve performance transparency, the DDBMS should consider issues such as **network latency**, the delay imposed by the amount of time required for a data packet to make a round trip from point A to point B; or **network partitioning**, the delay imposed when nodes suddenly become unavailable due to network failure.

14.11 Distributed Database Design

14.11.1 Data Fragmentation

Data fragmentation allows you to break a single object into two or more segments or fragments. Each fragment can be stored at any site over a computer network. Information about data fragmentation is stored in the distributed data catalogue (DDc), from where it is accessed by the TP to process user requests.

Data fragmentation strategies are based at the table-level and consist of dividing a table into logical fragments. A fragmented table can always be recreated from its fragmented parts by a combination of unions and joins.

There are three types of data fragmentation strategies:

- **Horizontal Fragmentation** Refers to the division of a relation into subsets (fragments) of tuples (rows). Each fragment is stored at a different node, and each fragment has unique rows. However, the unique rows all have the same attributes (columns). Each fragment represents the equivalent of a `SELECT` statement, with the `WHERE` clause on a single attribute.
- **Vertical Fragmentation** Refers to the division of a relation into attribute (column) subsets. Each fragment is stored at a different node, and each fragment has unique columns – with the exception of the key column, which is common to all fragments. Equivalent to the `PROJECT` statement in SQL.
- **Mixed Fragmentation** Refers to a combination of horizontal and vertical strategies. In other words, a table may be divided into several horizontal subsets (rows), each one having a subset of the attributes (columns).

Horizontal Fragmentation

There are various ways to partition a table horizontally:

- **Round-robin partitioning** Rows are assigned to a given fragment in a round-robin fashion (F_1, F_2, F_3, ..., F_n) to ensure an even distribution of rows among all fragments. This is not a good strategy if you require 'location awareness' – the ability to determine which DP node will process a query based on the geospatial location of the requester.
- **Range partitioning based on a partition key** A **partition key** is one or more attributes in a table that determine the fragment in which a row will be stored. For example, if you wanted to provide location awareness, a good partition key would be the customer state field. This is the most common and useful data partitioning strategy.

Vertical Fragmentation

You can also divide a relation into vertical fragments that are composed of a collection of attributes. Each vertical fragment must have the same number of rows, but the inclusion of different attributes depends on the key column.

Mixed Fragmentation

Mixed fragmentation requires a two-step procedure. First, horizontal fragmentation is introduced, which yields the subset of tuples matching certain criteria. Then, vertical fragmentation is used within each horizontal fragment to divide the attributes.

14.11.2 Data Replication

Data replication refers to the storage of data copies at multiple sites served by a computer network. Fragment copies can be stored at several sites to serve specific information requests. Since the existence of fragment copies can enhance data availability and response time, data copies can help to reduce communication and total query costs.

Replicated data are subject to the mutual consistency rule. The **mutual consistency rule** requires that all copies of data fragments be identical. Therefore, to maintain data consistency among the replicas, the DDBMS must ensure that a database update is performed at all sites where replicas exist.

There are two styles of replication:

- **Push Replication** After a data update, the originating DP node sends the changes to the replica nodes to ensure that data are immediately updated. This type of replication focusses on maintaining data consistency. However, it decreases data availability due to the latency involved in ensuring data consistency at all nodes.
- **Pull Replication** After a data update, the originating DP node sends 'messages' to the replica nodes to notify them of the update. The replica nodes decide when to apply the updates to their local fragment. In this type of replication, data updates propagate more slowly to the replicas. The focus is on maintaining data availability. However, this style of replication allows for temporary data inconsistencies.

Although replication has some benefits, it also imposes additional DDBMS processing overhead, because each data copy must be maintained by the system. To illustrate the replica overhead imposed on a DDBMS, consider the processes that the DDBMS must perform to use the database:

- If the database is fragmented, the DDBMS must decompose a query into subqueries to access the appropriate fragments.
- If the database is replicated, the DDBMS must decide which copy to access. A READ operation selects the *nearest copy* to satisfy the transaction. A WRITE operation requires that *all copies* be selected and updated to satisfy the mutual consistency rule.
- The TP sends a data request to each selected DP for execution.
- The DP receives and executes each request and sends the data back to the TP.
- The TP assembles the DP responses.

Three replication scenarios exist:

- **Fully Replicated Database** Stores multiple copies of *each* database fragment at multiple sites. In this case, all database fragments are replicated. A fully replicated database can be impractical due to the amount of overhead it imposes on the system.
- **Partially Replicated Database** Stores multiple copies of *some* database fragments at multiple sites. Most DDBMSs are able to handle the partially replicated database well.
- **Unreplicated Database** Stores each database fragment at a single site. Therefore, there are no duplicate database fragments.

Several factors influence the decision to use data replication:

- **Database Size** The amount of data replicated will have an impact on the storage requirements and the data transmission costs. Replicating large amounts of data requires a window of time and higher network bandwidth that could affect other applications.
- **Usage Frequency** The frequency of data usage determines how frequently the data needs to be updated.
- **Costs** Costs include those for performance, software overhead, and management associated with synchronising transactions and their components versus fault tolerance benefits that are associated with replicated data.

14.11.3 Data Allocation

Data allocation describes the process of deciding where to locate data. Data allocation strategies are:

- **Centralised Data Allocation** The entire database is stored at one site.
- **Partitioned Data Allocation** The database is divided into several disjointed parts (fragments) and stored at several sites.
- **Replicated Data Allocation** Copies of one or more database fragments are stored at several sites.

Data distribution over a computer network is achieved through data partition, data replication, or a combination of both. Data allocation is closely related to the way a database is divided or fragmented. Most data allocation studies focus on one issue: *which* data to locate *where*.

Data allocation algorithms take into consideration:

- Performance and data availability goals
- Size, number of rows, and number of relations that an entity maintains with other entities.
- Types of transactions to be applied to the database and the attributes accessed by each of those transactions.
- Disconnected operation for mobile users.

14.12 The CAP Theorem

- **Consistency** In a distributed database, consistency takes a bigger role. All nodes should see the same data at the same time, which means that the replicas should be immediately updated. However, this involves dealing with latency and network partitioning delays.
- **Availability** A request is always fulfilled by the system. No received request is ever lost.
- **Partition Tolerance** The system continues to operate even in the event of a node failure. The equivalent of failure transparency in distributed databases. The system will fail only if all nodes fail.

As businesses grow and the need for availability increases, database latency becomes a bigger problem. It is more difficult for a highly distributed database to ensure ACIDS transactions without paying a high price in network latency or data contention (delays imposed by concurrent data access). When dealing with highly distributed systems, some companies tend to forfeit the consistency and isolation components of the ACIDS properties to achieve higher availability. This trade-off between consistency and availability has generated a new type of distributed data system in which data are **basically available, soft state, eventually consistent (BASE)**. BASE refers to a data consistency model in which data changes are not immediate, but propagate slowly through the system until all replicas are eventually consistent. For example, **NoSQL** databases provide a highly distributed database with eventual consistency. **NewSQL** databases attempt to merge the best of relational and NoSQL data models. This type of database provides consistency and high availability with relaxed partition tolerance support.

14.13 Distributed Databases within the Cloud

Cloud computing is a new style of delivering applications, data, and resources to users over the Web. It provides an alternative for organisations that do not wish to provide their own information technology infrastructure to host their own databases or software. Instead, they rely on a third party cloud provider that uses a number of interconnected and virtualised computers to supply a range of IT services that are standardised. Each third party will have its own flexible pricing model for each service it provides, called a **service level agreement**.

Benefits to Cloud Infrastructure

- **Cost-Effectiveness** As the third party cloud provider is likely to be hosting services for many organisations, only one IT infrastructure is required, which reduces the cost to the individual organisation. Under the negotiated service level agreement, an organisation will also only pay for what it requires.

- **Latest Software** Most third party cloud providers will ensure their software is always the latest version available, to remain competitive.
- **Scalable Architecture** If the data requirements of the organisation expand, it is easy to increase the database capacity and/or change the underlying data model.
- **Mobile Access** Data and software within the cloud can be accessed generally from anywhere, which allows greater flexibility for employees in terms of where they work.

A traditional DDBMS will face problems when trying to operate in a cloud environment. For example, a DDBMS typically has control over all data requests (through queries) and associated hardware resources to ensure data is consistent. This is in contrast to a DDBMS operating within a cloud where hardware resources are allocated dynamically based upon service requirements at a given time. One solution is to use NoSQL databases to store and manage data within the cloud.

Current NoSQL solutions include column stores and document stores:

- **Column Stores** For large-scale distributed systems that store petabytes of data across hundreds, if not thousands, of servers. An example is Bigtable, which is a parse, distributed persistent multidimensional sorted map where the map is indexed by a row key, column key, and a time stamp.
- **Document Stores** Moves away from storing data in tables. Instead, each document is stored differently depending on its size and format. Document stores are referred to as document-orientated databases. An example is Apache's CouchDB, which is a distributed database system where replica copies of the same database can exist on multiple servers or offline clients.

A further example of a vendor-specific NoSQL database solution is Amazon's SimpleDB, which is a non-relational data store that operates within a cloud environment. Thus, the data can be distributed with replication, which allows high availability. Traditional relational database queries can be used to query data using Web service requests. The underlying data model can be changed quickly, and data is automatically indexed and optimised for performance.

Cloud computing provides availability and partition tolerance, but sacrifices consistency.

14.14 C.J. Date's 12 Commandments for Distributed Databases

Date's commandments describe a fully distributed database and, although no current DDBMS conforms to all of them, the rules do constitute a useful distributed database target. The 12 rules are:

1 Local Site Independence Each local site can act as an independent, autonomous, centralised DBMS. Each site is responsible for security, concurrency control, backup and recovery.

2 Central Site Independence No site in the network relies on a central site or any other site. All sites have the same capabilities.

3 Failure Independence The system is not affected by node failures. The system is in continuous operation, even in the case of a node failure or an expansion of the network.

4 Location Transparency The user does not need to know the location of the data in order to retrieve those data.

5 **Fragmentation Transparency** The user sees only one logical database. Data fragmentation is transparent to the user. The user does not need to know the name of the database fragments in order to retrieve them.

6 **Replication Transparency** The user sees only one logical database. The DDBMS transparently selects the database fragments to access. To the user, the DDBMS manages all fragments transparently.

7 **Distributed Query Processing** A distributed query may be executed at several different DP sites. Query optimisation is performed transparently by the DDBMS.

8 **Distributed Transaction Processing** A transaction may update data at several different sites. The transaction is transparently executed at several different DP sites.

9 **Hardware Independence** The system must run on any hardware platform.

10 **Operating System Independence** The system must run on any operating system software platform.

11 **Network Independence** The system must run on any network platform.

12 **Database Independence** The system must support any vendor's database product.

Chapter 15

Databases for Business Intelligence

15.1 Business Intelligence

Business Intelligence (BI) is a term that describes a comprehensive, cohesive, and integrated set of tools and processes used to capture, collect, integrate, store and analyse data with the purpose of generating and presenting information to support business decision making. This intelligence is based on learning and understanding the facts about the business environment. BI is a framework that allows a business to transform data into information, information into knowledge, and knowledge into wisdom.

Implementing BI in an organisation involves capturing not only internal and external business data, but also the **metadata**, or knowledge about the data.

BI is not a product by itself, but a framework of concepts, practices, tools and technologies that help a business better understand its core capabilities, provide snapshots of the company situation, and identify key opportunities to create competitive advantage.

In general, BI provides a framework for:

1. Collecting and storing operational data
2. Aggregating the operational data into decision support data
3. Analysing decision support data to generate information
4. Presenting such information to the end user to support business decisions
5. Making business decisions, which in turn generate more data that are collected, stored, and so on.
6. Monitoring results to evaluate outcomes of the business decisions, which again provides moire data to be collected and stored.
7. Predicting future behaviours and outcomes with a high degree of accuracy.

The seven points above represent a system-wide view of the flow of data, processes, and outcomes within the BI framework.

15.1.1 Business Intelligence Architecture

The general BI framework has six basic components that encompass the functionality required in most current-generation BI systems:

- **ETL Tools** Data **extraction, transformation and loading (ETL)** tools collect, filter, integrate, and aggregate internal and external data to be saved in a data store optimised for decision support. Internal data are generated by the company during its day-to-day operations, such as product sales history, invoicing, and payments. The external data sources provide data that cannot be found within the company, but are relevant to the business.
- **Data Store** Optimised for decision support and generally represented by a *data warehouse* or *data store*. The data are stored in structures that are optimised for data analysis and query speed.
- **Query and Reporting** Performs data selection and retrieval, and is used by the data analyst to create queries that access the database and create the required reports.
- **Data Visualisation** Presents data to the end user. This tool helps the end user select the most appropriate presentation format.
- **Data Monitoring and Alerting** Allows real-time monitoring of business activities. The BI system will present the concise information in a single integrated view for the data analyst. This integrated view could include specific metrics about the system performance or activities. Alerts can be placed on a given metric; once the value of a metric goes above or below a certain baseline, the system will perform a given action.
- **Data Analytics** Performs data analysis and data-mining tasks using the data in the data store. This tool advices the user about which data analysis tool to select, and how to build a reliable business data model. Business models are generated by special algorithms that identify and enhance the understanding of business situations and problems. Data analysis can be either explanatory or predictive:
  - **Explanatory Analysis** Uses the existing data in the data store to discover relationships and their types.
  - **Predictive Analysis** Creates statistical models of the data that allow predictions of future values and events.

The focus of traditional information systems was on operational automation and reporting; in contrast, BI tools focus on the strategic and tactical use of information. To achieve this goal, BI recognises that technology alone is not enough. Therefore, BI uses an arrangement of best management practices to manage data as a corporate asset.

Master data management (MDM) is a collection of concepts, techniques, and processes for the proper identification, definition, and management of data elements within an organisation. MDM's main goal is to provide a comprehensive and consistent definition of all data within an organisation. MDM ensures that all company resources that work with data have uniform and consistent views of the company's data.

An added benefit of this approach is that it provides a framework for business governance. **Governance** is a method or process of government. In this case, BI provides a method for controlling and monitoring business health and for consistent decision making. Governance also creates accountability for business decisions.

Sample of Business Intelligence Tools

- **Dashboards** Use Web-based technologies to present key business performance indicators or information in a single integrated view, generally using graphics that are clear, concise, and easy to understand.
- **Portals** Provide a unified, single point of entry for information distribution. Portals are a Web-based technology that uses a Web browser to integrate data from multiple sources into a single Web page.
- **Data Analysis and Reporting Tools** Used to query multiple and diverse data sources to create integrated reports.
- **Data-Mining Tools** Provide advanced statistical analysis to uncover problems and opportunities hidden within business data.
- **Data Warehouses** The foundation of a BI infrastructure. Data are captured from the production system and placed in the data warehouse on a near real-time basis. BI provides company-wide integration of data and the capability to respond to business issues in a timely manner.
- **Online Analytical Processing (OLAP) Tools** Provide multidimensional data analysis.
- **Data Visualisation** Provide advanced visual analysis and techniques to enhance understanding and create additional insight into business data and its true meaning.

Monitoring a business's health is crucial to understanding where the company is and where it is headed. To do this, BI makes use of a special type of metrics called key performance indicators. **Key performance indicators (KPIs)** are quantifiable numeric or scale-based measurements that assess the company's effectiveness or success in reaching its strategic and operational goals.

KPIs are determined after the main strategic, tactical and operational goals are defined for a business. To tie the KPI to the strategic master plan of an organisation, a KPI is compared to a desired goal within a specific time frame.

A modern BI system provides three distinctive reporting styles:

- **Advanced Reporting** A BI system provides insightful information about the organisation in a variety of presentational formats. The reports provide interactive features that allow the end user to study the data from multiple points of view – from highly summarised to very detailed. The reports present key actionable information used to support decision making.
- **Monitoring and Alerting** After a decision has been made, the BI system offers ways to monitor the decision's outcome. The BI system provides the end user with ways to define metrics and other key performance indicators to evaluate different aspects of an organisation. Exceptions and alerts can be set to warn managers promptly about deviations or problem areas.
- **Advanced Data Analytics** A BI system provides tools to help the end user discover relationships, patterns, and trends hidden within the organisation's data. These tools are used to create both explanatory and predictive analysis.

Business Intelligence Benefits

- Provides an integrating architecture.
- Provides a common user interface for data reporting and analysis.
- Having a common data repository fosters a single version of company data.
- Improved organisational performance.

15.1.2 Business Intelligence Evolution

The precursor of the modern BI environment was the first-generation decision support system. A **decision support system (DSS)** is an arrangement of computerised tools used to assist managerial decision making. A DSS typically has a much narrower focus and reach than a BI solution.

15.2 Decision Support Data

Although BI is used at strategic and tactical managerial levels within an organisation, its effectiveness depends on the quality of the data gathered at the operational level.

15.2.1 Operational Data vs Decision Support Data

Most operational data are stored in a relational database in which the structures (tables) tend to be highly normalised. Operational data storage is optimised to support transactions that represent daily operations. To provide effective update performance, operational systems store data in many tables, each with a minimum number of fields. Although such an arrangement is excellent in an operational database, it is not query friendly. Whereas operational data capture daily business transactions, DSS data give tactical and strategic business meaning to the operational data.

DSS data differ from operational data in three main areas:

- **Time Span** Operational data cover a short time frame. In contrast, decision support data tend to cover a longer time frame.
- **Granularity (level of aggregation)** DSS data must be presented at different levels of aggregation, from highly summarised to near atomic. Managers require summarised data, but the data also needs to be in a structure that enables them to **drill down**, or decompose, data into more atomic components. In contrast, when you **roll up** the data, you are aggregating the data to a higher level.
- **Dimensionality** Operational data focus on representing individual transactions rather than on the effects of the transactions over time. In contrast, data analysts tend to include many data dimensions and are interested in how the data relate over those dimensions.

Decision support data can be examined from multiple dimensions, using a variety of filters to produce each dimension. The ability to analyse, extract, and present information in meaningful ways is one of the differences between decision support data and transaction-at-a-time operational data.

From a designer's point of view, the differences between operational and DSS data are:

- Operational data represent transactions as they happen, in real time. DSS data are a snapshot of the operational data at a given point in time. Therefore, DSS data are historic, representing a time slice of the operational data.

- Operational and DSS data are different in terms of the transaction *type* and the transaction *volume*. Whereas operational data are characterised by update transactions, DSS data are mainly characterised by *query* (read-only) transactions. DSS data also require *periodic* updates to load new data that are summarised from the operational data. Finally, the transaction volume in operational data tends to be very high when compared with the low-to-medium levels found in decision support data.
- Operational data are commonly stored in many tables, and the stored data represent the information about a given transaction only. DSS data are generally stored in a few tables that store data derived from operational data. The DSS data do not include the details of each operational transaction. Instead, DSS data represent transaction *summaries*; therefore the decision support stores data that are integrated, aggregated and summarised for decision support purposes.
- The degree to which DSS data are summarised is very high compared to operational data. Therefore, there is a great deal of derived data in decision support databases.
- The data models that govern operational data and DSS data are different. The operational database's frequent and rapid data updates make data anomalies a potentially devastating problem. Therefore, the data requirements in an operational system generally require normalised structures that yield many tables, each of which contains the minimum number of attributes. In contrast, the decision support database is not subject to such transaction updates, and the focus is on querying capability. Therefore, decision support databases tend to be non-normalised and include few tables, each of which contains a large number of attributes.
- Query activity (frequency and complexity) in the operational database tend to be low to allow additional query processing cycles for the more crucial update transactions. Therefore, queries against operational data are typically narrow in scope, low in complexity, and speed-critical. In contrast, queries against decision support data are typically broad in scope, high in complexity, and less speed-critical.
- DSS data are characterised by large amounts of data. The large amount of data is the result of two factors:
  - Data are stored in non-normalised data structures that are likely to display many data redundancies and duplications.
  - The same data can be categorised in many different ways to represent different snapshots.

15.2.2 Decision Support Database Requirements

There are four main requirements for a decision support database: the database schema, data extraction and loading, the end-user analytical interface, and database size.

Database Schema

The decision support database schema must support complex (non-normalised) data representations. The database must contain data that are aggregated and summarised. The queries must also be able to extract multidimensional time slices. If you are using a relational DBMS, this suggests using non-normalised and even duplicated data.

The decision support database must also be optimised for query (read-only) retrievals. To optimise query speed, the DBMS must support features such as bitmap indexes and data partitioning to increase search speed. In addition, the DBMS query optimiser must be enhanced to support the non-normalised and complex structures found in decision support databases.

Data Extraction and Filtering

The decision support database is created largely by extracting data from the operational database and by importing additional data from external sources. Thus, the DBMS must support advanced data extraction and filtering tools. To minimise the impact on the operational database, the data extraction capabilities should allow batch and scheduled data extraction. The data extraction capabilities should also support different data sources. **Data filtering** capabilities must include the ability to check for inconsistent data or data validation rules. To filter and integrate the operational data into the DSS database, the DBMS must support advanced data integration, aggregation, and classification.

Using data from multiple external sources also usually means having to solve data-formatting conflicts.

Database Size

Decision support databases tend to be enormous: terabyte and petabyte ranges are not unusual. Therefore, the DBMS must be capable of supporting **very large databases (VLDBs)**. To support a VLDB, the DBMS might be required to use advanced hardware, and to support multiple-processor technologies.

15.3 The Data Warehouse

The data warehouse is usually a read-only database optimised for data analysis and query processing. Typically, data are extracted from various sources and then transformed and integrated (passed through a data filter) before being loaded into the data warehouse. This process is known as **Extraction, Transformation, and Loading (ETL)**.

Bill Inmon's Definition of a Data Warehouse – Top Down

A **data warehouse** is an integrated, subject-orientated, time-variant, nonvolatile collection of data that provides support for decision making.

- **Integrated** The data warehouse is a centralised, consolidated database that integrates data derived from the entire organisation and from multiple sources with diverse formats. Data integration implies that all business entities, data elements, data characteristics and business metrics are *described in the same way throughout the enterprise*.
- **Subject-Orientated** Data warehouse data are arranged and optimised to provide answers to questions coming from diverse functional areas within a company. Data warehouse data are organised and summarised by topic. For each topic, the data warehouse contains specific subjects of interest. Dara warehouse designers focus specifically on the data, rather than the processes that modify the data.
- **Time-Variant** In contrast to operational data, which focus on current transactions, warehouse data represent the flow of data through time. The data warehouse can even contain projected data, generated through statistical and other models. It is also time-variant in that once data are periodically updated to the data warehouse, all time-dependent aggregations are recomputed. As data in a data warehouse constitute a snapshot of the company history as measured by its variables, the time component is crucial. The data warehouse contains a time ID that is used to generate summaries and aggregations by week, month, quarter, year, and so on. Once the data enter the data warehouse, the time ID assigned to the data cannot be changed.
- **Non-Volatile** Once data enter the data warehouse, they are never removed. Because the data in the warehouse represent the company's history, the operational data (representing the near-term history) are always added to it. Data are never deleted, and new data are continually added, so the data warehouse is always growing.

This approach is called the **Top Down Approach**, and revolves around the created of a large centralised enterprise-wide data warehouse, which is liked to a number of departmental databases known as **data marts**.

This method structured data using the relational model in third normal form.

Ralph Kimball's Definition of a Data Warehouse – Bottom Up

A **data warehouse** is a copy of transaction data specifically structured for query and analysis.

This definition focuses on the functionality of the data warehouse, and not how it should be developed.

This approach is called the **Bottom Up Approach**, and first build several data marts within different departments in an organisation, and then virtually integrating these data marts to ensure one consistent view of an organisation.

This method creates multidimensional models of the data, i.e the **star schema**.

15.3.1 Twelve Rules that Define a Data Warehouse

1. The data warehouse and operational environments are separated.
2. The data warehouse data are integrated.
3. The data warehouse contains historical data over a long time horizon.
4. The data warehouse data are snapshot data captured at a given point in time.
5. The data warehouse data are subject-orientated.
6. No online updates are allowed.
7. The data warehouse development life cycle differs from classical systems development. The data warehouse development cycle is **data-driven**, the classical approach is **process-driven**.
8. The data warehouse contains data with several levels of detail.
9. The data warehouse environment is characterised by read-only transactions to very large data sets. The operational environment is characterised by numerous update transactions to a few date entities at a time.
10. The data warehouse environment has a system that traces data sources, transformations, and storage.
11. The data warehouse's metadata are a critical component of this environment. The metadata identify and define all data elements. The metadata provide the source, transformation, integration, storage, usage, relationships and history of each data element.
12. The data warehouse contains a chargeback mechanism for resource usage that enforces optimal use of the data by end users.

15.3.2 Data Marts

A **data mart** is a small, single-subject data warehouse subset that provides decision support to a small group of people. A data mart could also be created from the data extracted from a larger data warehouse for the specific purpose of supporting faster data access to a target group or function.

Data marts can serve as a test vehicle for companies exploring the potential benefits of data warehouses. Bu migrating gradually from data marts to data warehouses, a specific department's decision support needs can be addressed within a reasonable time frame (six months to one year) as compared to the longer time frame required to implement a data warehouse (one to three years).

The only difference between a data mart and a data warehouse is the size and scope of the problem being solved. Therefore, the problem definitions and data requirements are essentially the same for both.

15.3.3 Designing and Implementing a Data Warehouse

Different organisations have different constraints on information system development, which means there is no single formula for describing perfect data warehouse development.

The Data Warehouse as an Active Decision Support Framework

A data warehouse is not a static database. It is a dynamic framework for decision support that is always a work in progress.

A Company-Wide Effort That Requires User Involvement

Defining a data warehouse means being given an opportunity to help develop an integrated data model that captures the data that are considered to be essential to the organisation, from both end user and business perspectives.

The foremost technical concern in implementing a data warehouse is to provide end-user decision support with advanced data analysis capabilities – at the right moment, in the right format, with the right data, at the right cost.

Apply Database Design Procedures

One of the key differences from traditional the database design process is the level of detail required in defining the business model. Each business process that is to be modelled within the data warehouse must be described in detail in order to:

- Identify business measures.
- Identify the level of detail or granularity of the data.
- Check all data sources to ensure that the level of data required can actually be obtained from the existing source systems.

15.3.4 The Extraction, Transformation, Loading Process

The ETL process must ensure that the data that is loaded into the warehouse is high-quality, accurate, relevant, useful, and accessible. This is the most time-consuming phase in building a warehouse, as routines must be developed to select the required fields from often many sources of data.

The process of data extraction takes preselected data fields from a number of sources ready for transformation and loading into the data warehouse. Typically, data is not just extracted from the current operational systems, but also from archives, files from old systems and external data. Data is often in many different formats, and can be contradictory in nature.

Types of data that may be extracted include:

- **Operational Data** The main source of data into the warehouse. This data can come directly from any DBMS or application within the organisation. The key is to determine which operational data is relevant as not all will be included within the warehouse.
- **Historical Archived Data** Useful to perform predictive analytics. However, systems that store this data are often obsolete. Unique data extraction and transformation routines are therefore required to load the data in the warehouse during the first time load.

- **Internal Data** Data within the organisation such as budgets or sales forecasts, which may exist in spreadsheets.
- **Exterrnal Data** Important for comparing the business performance to enable an organisation to be competitive. The main problem with external data is that it can be available at any time and constant monitoring is required to determine when it is available. In addition, the format of the external data will be different from the internal data, and may require unique one-off transformations.

An organisation may choose to buy tools to extract data or may write individual routines in-house. The main issue is cost.

Once the data has been determined, each selected attribute must be mapped into the data warehouse. In each case, it may be necessary to apply a transformation rule. This is known as **mapping**. The process of data transformation aims to eliminate any data anomalies found in the extracted data, especially when it is from an operational source. Transformation also scrubs (or cleans) the data and ensures it in a standardised format ready for being presented as subject-orientated data.

Some common source data anomalies are:

- **Name and Address Inconsistencies** Storing names and addresses within a DBMS and a data warehouse provides many unique challenges for the designer. There is often no unique key, and values may be missing. A person may have more than one address, two or more people may be stored under the same address, and both names and addresses may be spelt incorrectly.

  In order to solve such problems, the data warehouse developer needs to ensure that each component part of the name and address appears in a standardised format. Rules should also be in place to ensure that, when data is entered, a person with the same name does not already exist. It is also necessary to check whether there is not already a person living at that address.
- **Multiple Encoding Problems** These occur when merging data from a number of operations systems. The solution is to agree on a format for each file in the database, and write routines to transform the data values into the correct format. It is very important that these rules also pick up erroneous data and flag records where fields cannot be standardised.
- **Different Country Standards** Multiple country standards are likely to exist in global organisations. Standards cover the type of currency, the format of the date, and measurements. Often, one standard is agreed on, and then routines exist to perform automatic conversions of the data as it is transported from the source file into the data warehouse.
- **Missing Values** Often, when you extract individual fields into the data warehouse, values may be missing. Sometimes, information may not have been collected; or values may be missing due to human error, no data available at the input stage in the source system, or data may simply have been mismatched from being selected across a number of sources. How to deal with the missing values depends on the significance of the field within the data warehouse.
- **Referential Integrity** This states that a foreign key must have a null entry or an entry that matches the primary key value in a table to which it is related. Referential integrity checks must be made on all data that is extracted from source systems prior to insertion into the data warehouse. As data is combined from different databases, violations of referential integrity constraints are more likely to occur. To ensure that this does not happen, a set of data warehouse referential integrity rules are applied to each relationship that will determine the status of foreign key columns when records are first extracted and then inserted into the data warehouse.

It is important that all transformation routines are documented within the data warehouse metadata.

The final stage of the ETL process is known as **loading**, in which the data is moved into the data warehouse. Loading data can be a very time-consuming process, and is usually done in two stages. The first stage, known as **first-time load**, takes place only once and is used initially to load historical data into the data warehouse. Due to unique extraction and transformation routines being used, and the large volume of data and processing required, the first time load is very time and resource intensive.

Once the data warehouse is live, it will need to be updated or refreshed at regular intervals. How often it is updated is dependent on the organisation's business cycle and the scheduling of its business intelligence activities. The update of a data warehouse is less complex that the first time load. The extraction and transformation loads are less intricate, and there is less data. However, the update will put pressure on the organisation's systems and networks, and therefore the **load window** (the time is takes to update the warehouse) should be scheduled during non-business hours.

15.4 Star Schemas

The **star schema** is a data modelling technique used to map multidimensional decision support data into a relational database. It creates the near equivalent of a multidimensional database schema from the existing relational database.

Star schemas yield an easily implemented model for multidimensional data analysis while still preserving the relational structures on which the operational database is built. The basic star schema has four components:

- **Facts** Numeric measurements (values) that represent a specific business aspect or activity. Facts commonly used in business data analysis are units, costs, prices, and revenues. They are normally stored in a fact table that is the centre of the star schema. The **fact table** contains facts that are linked through their dimensions. The fact table is updated periodically with data from operational databases.

  Facts can also be computed or derived at run time. These facts are sometimes called **metrics**, to differentiate them from stored facts.

- **Dimensions** Qualifying characteristics that provide additional perspectives to a given fact. Dimensions are of interest because DSS data are almost always viewed in relation to other data. Dimensions are normally stored in **dimension tables**.

- **Attributes** Each dimension table contains attributes. Attributes are often used to search, filter, or classify facts. Dimensions provide descriptive characteristics about the facts through their attributes. Therefore, the data warehouse designer must define common business attributes that will be used by the data analyst to narrow a search, group information, or describe dimensions.

  Conceptually, the multidimensional data model is best represented by a 3D cube. This is only a *conceptual* representation of the multidimensional data. The ability to focus on slices of the cube to perform a more detailed analysis is known as **slice and dice**. Each cut across the cube yields a slice. Intersecting slices yields small cubes: the 'dice' part. To slice and dict, it must be possible to identify each slice of the cube. This is done by using the values of each attribute in a given dimension.

- **Attribute Hierarchies** Attributes within dimensions can be ordered in a well-defined attribute hierarchy. The **attribute hierarchy** provides a top-down data organisation that is used for two main purposes: aggregation and drill-down/roll-up data analysis. The hierarchy allows the data warehouse and OLAP systems to have a defined path that will identify how data are to be

decomposed and aggregated for drill-down and roll-up operations. It is not necessary for all attributes to be part of an attribute hierarchy; some attributes merely exist to provide narrative descriptions of the dimensions. Attributes from different dimensions can be grouped to form a hierarchy.

Attribute hierarchies determine how the data in the data warehouse are extracted and presented. The attribute hierarchy information is stored in the DBMS's data dictionary, and is used by the OLAP tool to access the data warehouse properly. Once such access is ensured, query tools must be closely integrated with the data warehouse's metadata and they must support powerful analytical capabilities.

15.4.1 Star Schema Representation

Facts and dimensions are normally represented by physical tables in the data warehouse database. The fact table is related to each dimension table in a many-to-one relationship. In other words, many fact rows are related to each dimension row.

Fact and dimension tables are related by foreign keys. For primary key on the dimension table (the '1' side) is stored as part of the primary key on the fact table (the 'many' side). Because the fact table is related to many dimension tables, the primary key of the fact table is a composite primary key. By default, the fact table's primary key is always formed by combining the foreign keys pointing to the dimension tables to which they are related.

Since fact tables contain the actual values used in the decision support process, those values are repeated many times in the fact tables. Therefore, the fact tables are always the largest tables in the star schema. Since the dimension tables contain only non-repetitive information, the dimension tables are always smaller than the fact tables.

In a typical star schema, each dimension record is related to thousands of fact records. That characteristic of the star schema facilitates data retrieval functions because most of the time the data analyst will look at the facts through the dimension's attributes. Therefore, a DSS-optimised data warehouse DBMS first searches the smaller dimension tables before accessing the larger fact tables.

Data warehouses usually have many fact tables. Each fact table is designed to answer specific decision support questions. Multiple fact tables can also be created for performance and semantic reasons.

15.4.2 Star Schema Performance-Improving Techniques

The creation of a database that provides fast and accurate answers to data analysis queries is the data warehouse's prime objective. So, performance-enhancement actions might target query speed through the facilitation of SQL code as well as through better semantic representation of business dimensions. Four techniques are often used to optimise data warehouse design:

- **Normalising Dimensional Tables** Dimensional tables are normalised to achieve semantic simplicity and facilitate end-user navigation through the dimensions. For example, if the location dimension table contains transitive dependencies among region, province, and city, you can revise those relationships to the third normal form. A **snowflake schema** is a type of star schema in which the dimension tables can have their own dimension tables. The snowflake schema is usually the result of normalising dimension tables.

  By normalising the dimension tables, you simplify the data-filtering operations related to the dimensions. While this gains structural simplicity, it will increase the complexity of SQL statements, as it increases the number of joins needed.

- **Maintaining Multiple Fact Tables Representing Different Aggregation Levels** You can speed up query operations by creating and maintaining multiple fact tables each related to each level of aggregation. These aggregate tables are precomputed at the data-loading phase rather than at run time. The purpose of this technique is to save processor cycles at run time, thereby speeding up data analysis. An end-user query tool optimised for decision analysis then properly accesses the summarised fact tables instead of computing the values by accessing a lower level of detail fact table.

  The data warehouse designer must identify which levels of aggregation to pre-compute and store in the database. These multiple aggregate fact tables are updated during each load cycle in batch mode. Because the objective is to minimise access and processing time, according to the expected frequency of use and the processing time required to calculate a given aggregation level at run time, the data warehouse designer must select which aggregation fact tables to create.

- **Denormalising Fact Tables** Denormalising fact tables improves data access performance and saves data storage space. Denormalisation improves performance by using a single record to store data that normally take many records. Design criteria, such as frequency of use and performance requirements, are evaluated against the possible overload placed on the DBMS to manage the denormalised relations.

- **Partitioning and Replicating Tables** Table partitioning and replication are particularly important when a DSS is implemented in widely dispersed geographic areas. **Partitioning** splits a table into subsets of rows and columns and places the subsets close to the client computer to improve data access time. **Replication** makes a copy of a table and places it in a different location, also to improve access time.

  No matter which performance-enhancement scheme is used, time is the most common dimension used in business data analysis. Therefore, it is very common to have one fact table for each level of aggregation defined within the time dimension. Those fact tables must have an implicit or explicit periodicity defined. **Periodicity**, usually expressed as current year only, previous years, or all years, provides information about the timespan of the data stored in the table.

15.5 Data Analytics

Data analytics is a subset of BI functionality that encompasses a wide range of mathematical, statistical, and modelling techniques with the purpose of extracting knowledge from data. It is used at all levels within the BI framework, and represents what business managers really want from BI: the ability to extract actionable business insight from current events and foresee future problems or opportunities.

Data analytics discovers characteristics, relationships, dependencies or trends in the organisation's data, and then explains the discoveries and predicts future events based on the discoveries. The outcomes of data analytics then become part of the information framework on which decisions are built.

Data analytics tools can be grouped into two separate (but closely related and often overlapping) areas:

- **Explanatory Analytics** Focuses on discovering and explaining data characteristics and relationships based on existing data. Uses statistical tools to formulate hypotheses, test them, and answer the how and why of such relationships.
- **Predictive Analytics** Focuses on predicting future outcomes with a high degree of accuracy. Uses sophisticated statistical tools to help the end user create advanced models and answer questions about future data occurrences.

Predictive analytics uses explanatory analytics as a stepping stone to create predictive models.

15.5.1 Data Mining

Data mining refers to analysing massive amounts of data to uncover hidden trends, patterns and relationships; to form computer models to simulate and explain the findings; and then to use such models to support decision making. In other words, data mining focuses on the discovery and explanation stages of knowledge acquisition.

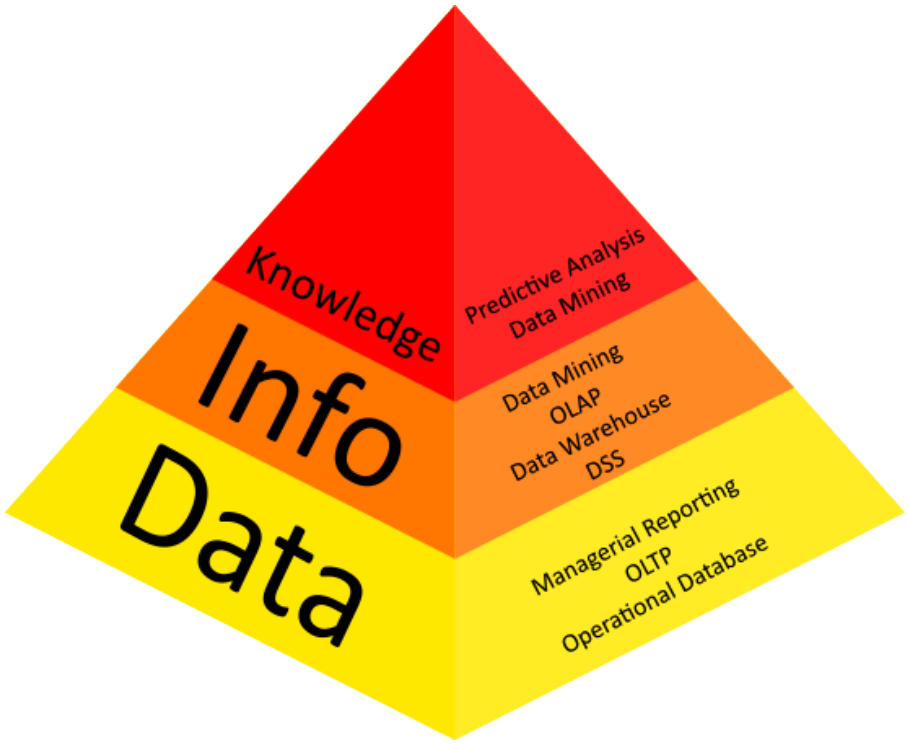


Figure 15.1: Extracting Knowledge from Data

In Figure 15.1, *data* form the pyramid base and represent what most organisations collect in their operational databases. The second level contains *information*, that represents the purified and processed data. Information forms the basis for decision making and business understanding. *Knowledge* is found at the pyramid's apex and represents highly specialised information.

In spite of the lack of precise standards, data mining is subject to four general phases:

1 **Data Preparation** The main data sets to be used by the data mining operation are identified and cleansed of any data impurities. Because the data in the data warehouse are already integrated and filtered, the data warehouse is usually the target set for data mining operations.

2 **Data Analysis and Classification** Studies the data to identify common data characteristics or patterns. During this phase, the data mining tool applies specific algorithms to find:

   - data groupings, classifications, clusters, or sequences
   - data dependencies, links, or relationships
   - data patterns, trends, and deviations.

3 **Knowledge Acquisition** Uses the results of the data analysis and classification phase. The data mining tool (with possible intervention by the end user) selects the appropriate modelling or knowledge acquisition algorithms. The most common algorithms used in data mining are based on neural networks, decision trees, rules induction, classification and regression trees, memory-based reasoning, and nearest neighbour and data visualisation. Hybrid algorithms also exist. A data mining tool may use many of these algorithms in any combination to generate a computer model that reflects the behaviour of the target set.

4 **Prognosis** Although many data mining tools stop at the knowledge acquisition phase, others continue to this phase. In this phase, the data mining findings are used to predict future behaviour and forecast business outcomes.

Data mining can be run in two modes:

- **Guided** The end user guides the data mining tool step by step to explore and explain known patterns and relationships. In this mode, the end user decides which techniques to apply to the data.
- **Automated** The end user sets up the data mining tool to run automatically, and uncover hidden patterns, trends, and relationships. The data mining tool applies multiple techniques to find significant relationships.

15.5.2 Predictive Analytics

Predictive analytics refers to the use of advanced mathematical, statistical and modelling tools to predict future business outcomes with high degrees of accuracy. Data mining and predictive analysis use similar and overlapping tools, but with a slightly different focus. Data mining focuses on answering the 'how' and 'what' of past data, while predictive analysis focuses on creating actionable models to predict future behaviours and events.

15.6 Online Analytical Processing

A new generation of tools, called **online analytical processing (OLAP)**, create an advanced data analysis environment that supports decision making, business modelling and operations research. OLAP systems share three main characteristics. They:

- Use multidimensional data analysis techniques.
- Provide advanced database support.
- Provide easy-to-use interfaces.

15.6.1 Multidimensional Data Analysis Techniques

In multidimensional analysis, data are processed and viewed as part of a multidimensional structure. The end user's view of data from a business perspective is more closely represented by the multidimensional view than by the tabular (operational) view. The multidimensional view also allows end users to consolidate or aggregate data at different levels, as well as to easily switch business perspectives or dimensions.

Multidimensional data analysis techniques are augmented by the following functions:

- Advanced data presentation functions.
- Advanced data aggregation, consolidation and classification functions that allow the data analyst to create multiple data aggregation levels, slice-and-dice data, and drill-down and roll-up data across different dimensions and aggregation levels.
- Advanced computational functions.
- Advanced data modelling functions.

Predictive modelling allows the system to build advanced statistical models to predict future values (business outcomes) with a high percentage of accuracy.

15.6.2 Advanced Database Support

To deliver efficient decision support, OLAP tools must have advanced data access features, which include:

- Access to many different kinds of DBMSs, flat files, and internal and external data sources
- Access to aggregated warehouse data as well as to the detail data in operational databases
- Advanced data navigation features, such as drill-down and roll-up
- Rapid and consistent query response times
- The ability to map end-user requests, expressed in either business or model terms, to the appropriate data source and then to the proper data access language. The query code must be optimised to match the data source, regardless of whether the source is operational or data warehouse data.
- Support for very large databases.

To provide a seamless interface, OLAP tools map the data elements from the data warehouse and from the operational database to their own data dictionaries. These metadata are used to translate end-user data analysis requests into their proper (optimised) query codes, which are then directed to the appropriate data source(s).

15.6.3 Easy-to-Use End-User Interface

When properly implemented, an analytical interface permits the user to navigate the data in a way that simplifies and accelerates decision making or data analysis.

15.6.4 OLAP Architecture

OLAP operational characteristics can be divided into three main modules:

- Graphical user interface (GUI)
- Analytical process logic
- Data processing logic

OLAP systems are designed to use both operational and data warehouse data. A common and practical architecture is one in which the OLAP GUI runs on client workstations, while the OLAP engine, or server, composed of the OLAP analytical processing logic and OLAP data processing logic, runs on a shared computer. In this case, the OLAP server will be a front end to the data warehouse's decision support data. This front or middle layer accepts and processes the data-processing requests generated by the many end-user OLAP workstations.

The OLAP system could merge the data warehouse and data mart approaches by storing extracts of the data warehouse at the end-user workstations. The objective is to increase the speed of data access and data visualisation. The logic behind this approach is the assumption that most end users usually work with fairly small, stable data warehouse subsets.

15.6.5 Relational OLAP

Relational online analytical processing (ROLAP) provides OLAP functionality by using relational databases and familiar relational query tools to store and analyse multidimensional data. ROLAP adds the following extensions to traditional RDBMS technology:

- **Multidimensional Data Schema Support within the RDBMS** Relational technology uses normalised tables to store data. The reliance on normalisation as the design methodology for relational databases is seen as a stumbling block to its use in OLAP systems. For decision support data, it is easier to understand data when they are seen with respect to other data.

  ROLAP uses star schemas to support multidimensional data representations, which are designed to optimise data query operations rather than data update operations.

- **Data Access Language and Query Performance Optimised for Multidimensional Data** ROLAP extends SQL so that it can differentiate between access requirements for data warehouse data (based on the star schema) and operational data (normalised tables). In that way, a ROLAP system is able to generate the SQL code required to access the star schema data.

  Query performance is also improved because the query optimiser is modified to identify the SQL code's intended query targets. Another source of improved query performance is the use of advanced indexing techniques such as bitmapped indexes within relational databases.

- **Support for Very Large Databases** When the relational database is used in a DSS role, it must also be able to store very large amounts of data.

With an open client/server architecture, ROLAP provides advanced decision support capabilities that are scalable to the entire enterprise. ROLAP is a logical choice for companies that already use relational databases for their operational data.

15.6.6 Multidimensional OLAP

Multidimensional online analytical processing (MOLAP) extends OLAP functionality to **multidimensional database management systems (MDBMSs)**. An MDBMS uses special proprietary techniques to store data in matrix like n-dimensional arrays. MOLAP's premise is that multidimensional databases are best suited to manage, store, and analyse multidimensional data. Most of the proprietary techniques used are derived from engineering fields such as computer-aided design/manufacturing (CAD/CAM) and geographic information systems (GIS). MOLAP tools store data using multidimensional arrays, row stores or column stores.

Conceptually, MDBMS end users visualise the stored data as a 3D cube known as a **data cube**. The location of each data value in the data cube is a function of the x, y, and z axes in a 3D space. These axes represent the dimensions of the data value. The data cubes can grow to n number of dimensions, thus becoming **hypercubes**. Data cubes are created by extracting data from the operational databases or from the data warehouse. One important characteristic of data cubes is that they are *static*: they are not subject to change and must be created before they can be used. Data cubes cannot be created by ad hoc queries: instead, you query pre-created cubes with defined axes.

MOLAP databases are known to be much faster than their ROLAP counterparts, especially when dealing with large to medium data sets. To speed up data access, data cubes are normally held in memory in what is called the **cube cache**. Since MOLAP also benefits from a client/server infrastructure, the cube cache can be located at the MOLAP server, at the MOLAP client, or in both locations.

As the data cube is pre-defined with a set number of dimensions, the addition of a new dimension requires that the entire data cube be recreated. This recreation process is a time-consuming operation. Therefore, when data cubes are created too often, the MDBMS loses some of its speed advantage.

The MDBMS is best suited to small and medium data sets. Scalability is somewhat limited, because the size of the data cube is restricted to avoid lengthy data access times caused by having less work space (memory) available for the operating system and the application programs.

The MDBMS makes use of proprietary data storage techniques, which require proprietary data access methods using a multidimensional query language.

Multidimensional data analysis is also affected by how the database system handles sparsity. **Sparsity** is a measurement of the density of the data held in the data cube. Sparsity is computed by dividing the total number of actual values in the cube by the total number of cells in the cube. Since the data cube's dimensions are predefined, not all cells are populated. Multidimensional databases must handle sparsity effectively to reduce processing overhead and resource requirements.

15.7 SQL Analytic Functions

A database is at the core of all data warehouses. Therefore, all SQL commands (such as `CREATE`, `INSERT`, `UPDATE`, `DELETE`, and `SELECT`) will work in the data warehouse as expected. However, most queries you run in a data warehouse tend to include data groupings and aggregations over multiple columns.

15.7.1 The ROLLUP Extension

The ROLLUP Extension is used with the GROUP BY clause to generate aggregates by different dimensions. The GROUP BY clause generates only one aggregate for each new value combination of attributes listed in the clause. The ROLLUP extension goes one step further; it enables you to get a subtotal for each column listed except for the last one, which gets a grand total instead.

The order of the column list within the GROUP BY ROLLUP is important: the last column in the list generates a grand total, all other columns generate subtotals.

The ROLLUP extension is particularly useful when you want to obtain multiple nested subtotals for a dimension hierarchy.

15.7.2 The CUBE Extension

The CUBE extension is also used with the GROUP BY clause to generate aggregates by the listed columns, including the last one. The CUBE extension enables you to get a subtotal for each column listed in an expression, in addition to a grand total for the last column listed.

The CUBE extension is particularly useful when you want to compute all possible subtotals within groupings based on multiple dimensions. Cross tabulations are especially good candidates for application of the CUBE extension.

15.7.3 Materialised Views

The data warehouse normally contains fact tables that store specific measurements of interest to an organisation. Such measurements are organised by different dimensions. The vast majority of OLAP business analytics is based on comparisons of data that are aggregated at different levels.

Since businesses normally use a predefined set of summaries for benchmarking, it is reasonable to predefine such summaries for future use by creating summary fact tables. However, creating multiple summary fact tables that use GROUP BY queries with multiple table joins could become a resource-intensive operation. Also, data warehouses must be able to maintain up-to-data summarised data at all times. So, when new data is added to the base fact tables, the summary tables would need to be re-created.

To save query processing time, most database vendors have implemented additional functionality to manage aggregate summaries more efficiently. This new functionality resembles the standard SQL views for which the SQL code is predefined in the database. However, the added functionality difference is that the views also store the pre-aggregated rows, something like a summary table. For example, Microsoft SQL server provides indexed views, while Oracle provides materialised views.

A **materialised view** is a dynamic table that contains not only the SQL query command to generate rows, but also stores the actual rows. The materialised view is created the first time the query is run and the summary rows are stored in the table. The materialised view rows are automatically updated when the base tables are updated. That way, the data warehouse administrator creates the view but will not have to update the view. The use of materialised views is totally transparent to the end user.

Materialised View Syntax

```
CREATE MATERIALISED VIEW view_name
BUILD {IMMEDIATE | DEFERRED}
REFRESH {[FAST | COMPLETE | FORCE]} ON COMMIT
[ENABLE QUERY REWRITE]
AS select_query;
```

The `BUILD` clause indicates when the materialised view rows are actually populated. `IMMEDIATE` indicates that the materialised view rows are populated right after the command is entered. `DEFERRED` indicates that the materialised view rows are populated at a later time. Until then, the materialised view is in an 'unusable' state. The DBMS provides a special routine that an administrator runs to populate materialised views.

The `REFRESH` clause lets you indicate when and how to update the materialised view when new rows are added to the base tables. `FAST` indicates that whenever a change is made in the base tables, the materialised view only updates the affected rows. `COMPLETE` indicates that a complete update is made for all rows in the materialised view when the select query on which the view is based is re-run. `FORCE` indicates that the DBMS will first try to do a `FAST` update; otherwise it will do a `COMPLETE` update. The `ON COMMIT` clause indicates that the updates to the materialised view will take place as part of the commit process of the underlying DML statement, that is, as part of the commit of the DML transaction that updated the base tables. The `ENABLE QUERY REWRITE` option allows the DBMS to use the materialised views in query optimisation.

15.8 Data Visualisation

Data visualisation is the process of abstracting data to provide a visual data representation that enhances the user's ability to comprehend the meaning of the data. The goal of data visualisation is to allow the user to quickly and efficiently see the data's big picture by identifying trends, patterns, and relationships. Providing summarised tabular data to managers does not give them enough insight into the meaning of the data to make informed decisions. Data visualisation encodes the data into visually rich formats.

An advantage of data visualisation is that is is an effective communication tool that makes is easier to understand data – in particular, large amounts of data. As a communication tool, data visualisation helps discover the message hidden in the data. But such data has to be properly vetted – processed, validated (distilled of bad data points) and organised within a context. Data visualisation is just a tool, not an end in itself. Data visualisation allows end users to explore data quickly and gain insights about it, but is does not replace rigorous data analysis using other tools such as statistics, data modelling, and predictive modelling.

15.8.1 The Science of Data Visualisation

Data visualisation has its roots in the cognitive sciences, which study how the human brain receives, interprets, organises, and processes information. Specifically, data visualisation relates to how our brains process visual data.

Data visualisation is concerned with both form and function. Form means using the proper visual construct, and function means applying the correct data transformations.

As a discipline, data visualisation can be studied as a group of visual communication techniques used to explore and discover data insights by applying:

- **Pattern recognition** Visually identifying trends, distribution and relationships.
- **Spatial awareness** Use of size and orientation to compare and relate data.
- **Aesthetics** Use of shapes and colours to highlight and contrast data composition and relationships.

In general, data visualisation uses five characteristics: shape, colour, size, position, and grouping/order, to convey and highlight the meaning of the data.

15.8.2 Understanding the Data

In general, there are two types of data:

- **Qualitative** Describes qualities of the data. This type of data can be subdivided into two subtypes:
    - **Nominal** Data that can be counted but not ordered or aggregated.
    - **Ordinal** Data that can be counted and ordered, but not aggregated.
- **Quantitative** Describes numeric facts or measured of the data. This type of data can be counted, ordered, and aggregated.

You can think of qualitative data as being the dimensions on a star schema, and the quantitative data as being the facts of a star schema.

Chapter 17

Database Connectivity and Web Technologies

17.1 Database Connectivity

Database connectivity refers to the mechanisms through which application programs connect and communicate with data repositories. Databases store data in persistent storage structures so that they can be retrieved at a later time for processing. The DBMS acts as an intermediary between the data and the end user's applications.

Database connectivity software works in a client/server architecture, in which processing tasks are split among multiple software layers. In this model, the multiple layers exchange control messages and data. In the case of database connectivity, you could break down its functionality into three broad layers:

1. A **data layer** where the data resides. You can think of this layer as the actual data repository interface. This layer resides closest to the database itself and is normally provided by the DBMS vendor.
2. A **middle layer** that manages multiple connectivity and data transformation issues. This layer is in charge of dealing with data logic issues, data transformations, ways to 'talk' to the database below it, and so on. This would also include translating multiple data manipulation languages to the native language supported by the specific data repository.
3. A **top layer** that interfaces with the actual external application. This mostly comes in the form of an application programming interface that publishes specific protocols for the external programs to interact with the data.

Database connectivity software is also known as **database middleware**, because it provides an interface between the application program and the database or data repository. The data repository, also known as the **data source**, represents the data management application that will be used to store the data generated by the application program. Ideally, a data source can be located anywhere, and hold any type of data. Furthermore, the same database connectivity middleware can support multiple data sources at the same time. This multidata-source-type capability is based on the support of well-established data access standards. The need for standard database connectivity interfaces cannot be overstated.

Native SQL Connectivity

This refers to the connection interface that is provided by the database vendor, and is unique to that vendor. The best example is the Oracle RDBMS. To configure a client application to an Oracle database, you must install and configure Oracle's SQL*Net interface in the client computer.

Native database connectivity interfaces are optimised for their DBMS, and these interfaces support access to most, if not all, of the database features. However, maintaining multiple native interfaces for different databases can become a burden for the programmer.

Usually, the native database connectivity interface provided by the vendor is not the only way to connect to a database; most current DBMS produces support other database connectivity standards, the most common being ODBC.

17.1.1 ODBC, DAO, RDO, and UDA

Developed in the early 1990s, **Open Database Connectivity (ODBC)** is Microsoft's implementation of a superset of the SQL Access Group **Call Level Interface (CLI)** standard for database access. It is probably the most widely supported database connectivity interface. ODBC allows any Windows application to access relational data sources, using SQL via a standard application programming interface (API). Although APIs are designed for programmers, they are ultimately good for users because they guarantee that all programs using a common API will have similar interfaces. That makes it easy for users to learn new programs.

ODBC was the first widely adopted database middleware standard. As programming languages evolved, ODBC did not provide significant functionality beyond the ability to execute SQL to manipulate relational style data. Therefore, programmers needed a better way to access data. To answer that need, Microsoft developed other data access interfaces:

- **Data Access Objects (DAO)** Microsoft's API that allows access to the Microsoft Access Database.
- **Remote Data Objects (RDO)** A higher level object-oriented application interface primarily used in Microsoft Visual Basic. It allowed developers to interface directly with ODBC data sources.
- **Universal Data Access (UDA)** A new proposed framework that is designed to bring about a single uniform API that will allow access to relational and non-relational databases.

Client applications can use ODBC to access relational data sources. However, the DAO and RDO provide more functionality. DAO and RDO make use of the underlying ODBC data services. ODBC, DAO, and RDO are implemented a shared code that is dynamically linked to the Windows operating environment through **dynamic link libraries (DLLs)**. Running as a DLL, the code speeds up load and run times.

The basic ODBC architecture has three main components:

- A high-level **ODBC API** through which application programs access ODBC functionality.
- A **driver manager** that is in charge of managing all database connections.
- An **ODBC driver** that communicates directly with the DBMS.

Defining a data source is the first step in using ODBC. To define a data source, you must create a **data source name (DSN)** for the data source. To create a DSN, you need to provide:

- **ODBC Driver** You need to identify the driver used to connect to the data source. Normally provided by the database vendor, although Microsoft provides several drivers that connect to the most common databases.
- **Name** A unique name by which the data source will be known to ODBC, and therefore, to applications. ODBC provides two types of data sources:
  - **User data sources** Available only to the user.
  - **System data sources** Available to all users, including operating system services.
- **ODBC Driver Parameters** Most ODBC drivers require specific parameters to establish a connection to the database.

Once the ODBC data source is defined, application programmers can write to the ODBC API by issuing specific commands and providing the required parameters. The ODBC driver manager will properly route the calls to the appropriate data source. The ODBC API standard defined three levels of compliance: Core, Level-1, and Level-2. The compliance levels provide increasing features. The database vendors can choose which level to support. However, to interact with ODBC, the database vendor must implement all of the features indicated in the ODBC API support level.

As much of the functionality provided by these interfaces is oriented to accessing relational data sources, the use of the interfaces is limited when they are used with other data source types.

17.1.2 OLE-DB

The connectivity standards in the previous section did not provide support for non-relational data. To answer that need, and to simplify data connectivity, Microsoft developed **Object-Linking and Embedding for Databases (OLE-DB)**. Based on Microsoft's Component Object Model (COM), OLE-DB is database middleware that adds object-oriented functionality for access to relational and non-relational data.

OLE-DB is composed of a series of COM objects that provide low-level database connectivity for applications. Since OLE-DB is based on COM, the objects contain data and methods. The OLE-DB is better understood when you divide its functionality into two types of objects:

- **Consumers** Objects (applications or processes) that request and use data. The data consumers request data by invoking the methods exposed by the data provider objects and passing the required parameters.
- **Providers** Objects that manager the connection with a data source and provide data to the consumers. Divided into two categories:
  - **Data Providers** Provide data to other processes. Database vendors create data provider objects that expose the functionality of the underlying data source.
  - **Service Providers** Provide additional functionality to consumers. The service provider is located between the data provider and the consumer. The service provider requests data from the data provider, transforms the data, and then provides the transformed data to the data consumer. In other words, the service provider acts like a data consumer to the data provider, and as a data provider to the data consumer.

As a common practice, many vendors provide OLE-DB objects to augment their ODBC support, effectively creating a shared object layer on top of their existing database connectivity through which applications can interact.

By using OLE-DB objects, the database vendor can choose which functionality to implement in a modular way, instead of being forced to include all of the functionality all of the time.

OLE-DB provides additional capabilities for the applications accessing the data. However, it does not provide support for scripting languages, especially the ones used for Web development, such as Active Server Pages (ASP) and ActiveX. To provide that support, Microsoft developed a new object framework called **ActiveX Data Objects (ADO)**. (A **script** is written in a programming language that is not compiled but is interpreted and executed at run time). ADO provides a high-level API to interact with OLE-DB, DAO and RDO. It provides a unified interface to access data from any programming language that uses the underlying OLE-DB objects.

17.1.3 ADO.NET

Based on ADO, **ADO.NET** is the data access component of Microsoft's .NET application development framework. The **Microsoft .NET framework** is a component-based platform for developing distributed, heterogeneous, interoperable applications aimed at manipulating any type of data over any network under any operating system and programming language.

The .NET framework extends and enhances the functionality provided by the ADO/OLE-DB duo. It introduced two new features critical for the development of distributed applications: DataSets and XML support.

A **DataSet** is a disconnected memory-resident representation of the database. The DataSet contains tables, columns, rows, relationships and constraints. Once the data are read from a data provider, the data are placed on a memory-resident DataSet. The DataSet is then disconnected from the data provider. The data consumer application interacts with the data in the DataSet object to make changes (inserts, updates, and deletes) in the DataSet. Once the processing is done, the DataSet data are synchronised with the data source and the changes are made permanent.

The DataSet is internally stored in XML format, and the data in the DataSet can be made persistent as XML documents. You can think of the DataSet as an XML, in-memory database that represents the persistent data stored in the data source.

The ADO.NET framework consolidated all data access functionality under one integrated object model. In this object model, several objects interact with one another to perform specific data manipulation functions. These objects can be grouped as data providers and consumers.

Data provider objects are provided by the database vendors, but ADO.NET comes with two standard data providers: a data provider for OLE-DB sources and a data provider for SQL Server.

Whatever the data provider is, it must support a set of specific objects in order to manipulate the data in the data source:

- **Connection** Defines the data source used, the name of the server, the database, and so on. This object enables the client application to open and close a connection to a database.
- **Command** Represents a database command to be executed within a specified database connection. This object contains the actual SQL code or a stored procedure call to be run by the database. When a `SELECT` statement is executed, the Command object returns a set of rows and columns.

- **DataReader** A specialised object that creates a read-only session with the database to retrieve data sequentially (forward only) in a rapid manner.
- **DataAdapter** In charge of managing a DataSet object. The most specialised object in the ADO.NET framework, and contains objects used to populate and synchronise the data in the DataSet with the permanent data source data.
- **DataSet** The in-memory representation of the data in the database. Contains two main objects:
  - **DataTableCollection** Contains a collection of DataTable objects that make up the 'in-memory' database.
  - **DataRelationCollection** Contains a collection of objects describing the data relationships and ways to associate one row in a table to the related row in another table.
- **DataTable** Represents the data in tabular format. This object has one important property: PrimaryKey, which allows the enforcement of entity integrity. In turn, the DataTable object is composed of three main objects:
  - **DataColumnCollection** Contains one or more column descriptions. Each column description has properties such as column name, data type, nulls allowed, minimum value and maximum value.
  - **DataRowCollection** Contains zero or more rows with data as described in the DataColumnCollection.
  - **ConstraintCollection** Contains the definition of the constraints for the table. Two types of constraints are supported: ForeignKeyConstraint, and UniqueConstraint.

A DataSet is a simple database with tables, rows, and constraints. It doesn't require a permanent connection to the data source: once it is populated, it is completely independent of the data source.

DataTable objects in a DataSet can come from different data sources.

The ADO.NET framework is optimised to work in disconnected environments. In a disconnected environment, applications exchange messages in request/reply format. The most common example of a disconnected system is the Internet.

17.1.4 Java Database Connectivity (JDBC)

Java is an object-oriented programming language that runs on top of Web browser software. A programmer can write a Java application once and then run it in multiple environments without any modification. The cross-platform capabilities are based on its portable architecture. Java code is stored in pre-processed chunks called **applets** that run in a virtual machine environment in the host operating system. This environment has well-defined boundaries, and all interactivity with the host OS is closely monitored.

When Java applications need to access data outside the Java run-time environment, they use predefined APIs. **Java Database Connectivity (JDBC)** is an API that allows a Java program to interact with a wide range of data sources. JDBC allows a Java program to establish a connection with a data source, prepare and send the SQL code to the database server, and process the result set.

One main advantage of JDBC is that it allows a company to leverage its existing investment in technology and personnel training. JDBC allows programmers to use their SQL skills to manipulate the data in the company's databases. JDBC allows direct access to a database server, or access via database middleware.

Another advantage of JDBC over other middleware is that it requires no configuration on the client side. The JDBC driver is automatically downloaded and installed as part of the Java applet download. Because Java is a web-based technology, applications can connect to a database directly using a simple URL. Once the URL is invoked, the Java architecture domes into play, the necessary applets are downloaded to the client (including the JDBC database driver and all configuration information) and then the applets are executed securely in the client's run-time environment.

17.1.5 PHP

PHP is a widely-used general-purpose scripting language that is especially suited for Web development and is seen as an alternative to Microsoft's ASP. It is a server-side scripting language, which means it is interpreted on the web server before the web page is sent to a web browser to be displayed. PHP supports connectivity to a number of different databases through specific database extensions in the case of MySQL, to connecting through ODBC extensions for Oracle.

PHP offers an extension module that can be used to connect to Oracle databases. The Oracle Call Interface (OCI 11) is an CPI that allows the creation of applications that use a series of function calls to access an Oracle database server. OCI 11 gives the developer control over all stages of SQL query execution. A typical application developed using OCI 11 must connect to one or more databases, open the cursors needed by the program to hold the data extracted by the SQL query, process any SQL statements within the application, close the cursor, and then disconnect the database. Oracle believes that OCI 11 is the most efficient way of connecting to any Oracle database, as it gives greater control over how an application is designed and also a higher degree of control over program execution.

17.2 Database Internet Connectivity

17.2.1 Web-to-Database Middleware: Server-Side Extensions

In general, the Web server is the main hub through which all Internet services are accessed. When the web server receives a page request, it looks for a page on the hard disk; when it finds the page, the server sends it back to the client.

Dynamic web pages are at the heart of current generation websites. In this database query scenario, the Web server generates the web page contents before it sends the page to the client web browser. The only problem with this scenario is that the Web server must include the database query result *before* it sends that page back to the client. Unfortunately, neither the Web browser nor the Web server knows how to connect to and read data from the database. Therefore, to support this type of request, the Web server's capability must be extended so it can understand and support database requests. This job is done through a server-side extension.

A **server-side extension** is a program that interacts directly with the web server to handle specific types of requests. In the previous database query example, the server-side extension program retrieves the data from databases and passes the retrieved data to the web server, which, in turn, sends the data to the client's browser for display purposes. The server-side extension makes it possible to retrieve and present the query results, but more importantly, *it provides its services to the web server in a way that is totally transparent to the client browser*.

A database server-side extension is also known as **Web-to-database middleware**.

Server-Side Extension Steps

1. The client browser sends a page request to the Web server.
2. The web server receives and validates the request. In this case, the server passes the request to the Web-to-database middleware for processing.
3. The Web-to-database middleware reads, validates, and executes the script. In this case, it connects to the database and passes the query, using the database connectivity layer.
4. The database server executes the query and passes the result back to the Web-to-database middleware.
5. The Web-to-database middleware compiles the result set, dynamically generates an HTML-formatted page that includes the data retrieved from the database, and sends it to the Web server.
6. The Web server returns the just-created HTML page, which now includes the query result set, to the client browser.
7. The client browser displays the page on the local computer.

17.2.2 Web Server Interfaces

Extending Web server functionality implies that the Web server and the Web-to-database middleware will properly interpolate with each other. **Interpolate** means that each party to the communication can respond to the communications of the other. If a web server is to communicate successfully with an external program, both must define a standard way to exchange messages and respond to requests. There are two well-defined web server interfaces:

- Common Gateway Interface (CGI)
- Application Programming Interface (API).

The **Common Gateway Interface (CGI)** uses script files to perform specific functions based on the client's parameters that are passed to the Web server. The script file is a small program written in a programming language. The script file's contents can be used to connect to the database and to retrieve data from it, using the parameters passed by the Web server. Next, the script converts the retrieved data to HTML format and passes the data to the Web server, which sends the HTML-formatted page to the client.

The main disadvantage of using CGI scripts is that the script file is an external program that is individually executed for each user request. That scenario decreases system performance. The language and method used to create the script can also affect performance.

An **application programming interface (API)** is a newer Web server interface standard that is more efficient and faster than a CGI script. APIs are more efficient because they are implemented as shared code or dynamic link libraries. That means the API is treated as part of the Web server program that is dynamically invoked when needed. APIs are faster than CGI scripts because the code resides in memory and there is no need to run an external program for each request. Instead, the same API serves all requests. Another advantage is that an API can use a shared connection to the database instead of creating a new one every time, as is the case with CGI scripts.

Although APIs are more efficient in handling requests, they have some disadvantages. Because the APIs share the same memory space as the Web server, an API error can bring down the server. The other disadvantage is that APIs are specific to the Web server and to the OS.

17.2.3 The Web Browser

The Web browser is the application software that lets end users navigate (browse) the Web. Each time the end user clicks a hyperlink, the browser generates an HTTP `GET` page request that is sent to the designated Web server, using the TCP/IP internet protocol.

The Web browser's job is to *interpret* the HTML code that it receives from the Web server and to present the different page components in a standard formatted way. Unfortunately, the browser's interpretation and presentation capabilities are not sufficient to develop Web-based applications.

The Web is a **stateless system**: at any given time, a Web server does not know the status of any of the clients communicating with it. There is no open communication line between the server and each client accessing it. Instead, clients and servers interact in short 'conversations' that follow the request-reply model. The only time the client and server computers communicate is when the client requests a page – when the user clicks a link – and the server sends the requested page to the client.

The Web browser, though its use of HTML, does not have computational abilities beyond formatting output text and accepting form field inputs. Even when the browser accepts form field data, there is no way to perform immediate data entry validation. To perform processing in the client, the Web defers to other Web programming languages. To improve the capabilities of the Web browser, you must use plugins and other client-side extensions.

17.2.4 Client Side Extensions

Client-side extensions add functionality to the Web browser. The most commonly encountered extensions are:

- **Plugins** External applications that are automatically invoked by the browser when needed. Because it is an *external* application, the plugin is OS-specific. The plugin is associated with a data object – generally using the file extension – to allow the Web server to handle data that are not originally supported.
- **JavaScript** A scripting language that allows Web authors to design interactive sites. Simpler to generate than Java, and easier to learn. JavaScript code is embedded in the Web pages. It is downloaded with the Web page and is executed when a specific event takes place.
- **ActiveX and VBScript** Microsoft's alternative to Java. A specification for writing programs that ran inside Microsoft's browser (Internet Explorer). Not truly cross-platform, and support was dropped in 2015.

From the developer's point of view, using routines that permit data validation on the client side is an absolute necessity. Most data validation routines are done in Java, JavaScript, or VBScript.

17.2.5 Web Application Servers

A **Web application server** is a middleware application that expands the functionality of Web servers by linking them to a wide range of services, such as databases, directory systems, and search engines. The Web application server also provides a consistent run-time environment for Web applications.

17.2.6 Web Database Development

Web database development deals with the process of interfacing databases with the Web browser – in short, how to create Web pages that access data in a database. Multiple web environments can be used to develop Web database applications.

Once of the most common web application development environments is known as **LAMP**. LAMP is made of the Linux OS, the Apache web server, MySQL database, and the PHP programming language. It is often used within organisations that need an effective way of managing organisational data, but do not have the time or money to invest in a large-scale, costly Web-development project.

The main benefits of LAMP are that it is easy to program, and applications can be developed offline and then deployed onto the Web. Deployment is relatively straightforward, as PHP is easily integrated with the Apache server and MySQL.

Current generation systems involve more than just the development of Web-enabled database applications. They also require applications that can communicate with one another, and with other systems not based on the Web. Systems must be able to exchange data in a standard-based format, which is the role of XML.

17.3 Extensible Markup Language (XML)

Extensible Markup Language (XML) is a metalanguage used to represent and manipulate data elements. It is designed to facilitate the exchange of structured documents over the Internet.

The XML metalanguage allows the definition of new tags, to describe the data elements used in an XML document. Given that feature, XML is said to be an **extensible** language. XML is derived from the Standard Generalised Markup Language (SGML), which is an international standard for the publication and distribution of highly complex technical documents that are too complex and unwieldy for the Web. Just like HTML, an XML document is a text file, but is has a few additional characteristics:

- XML allows the definition of new tags to describe data elements.
- XML is case sensitive.
- XML tags must be well-formed: each opening tag must have a corresponding closing tag.
- XML tags must be properly nested.
- You can use the `<--` and `-->` symbols to enter comments in the XML document.
- The `XML` and `xml` prefixes are reserved for XML only.

XML is not a new version or replacement for HTML. It is concerned with the description and representation of the data, rather than the way the data are displayed. XML provides the semantics that facilitate the sharing, exchange, and manipulation of structured documents over organisational boundaries.

- The first line of an XML document is the XML document declaration, and it is mandatory.

  ```
  <?xml version='1.0'?>
  ```

- Every XML document has a **root element**.
- The root element contains **child elements**, or sub-elements.
- Each element can contain **sub-elements**.

17.3.1 Document Type Definitions (DTDs) and XML Schemas

Companies that use B2B transactions must have a way to understand and validate one another's tags. One way to accomplish that is through the use of Document Type Definitions. A **Document Type Definition (DTD)** is a file with a `.dtd` extension that describes XML elements – in effect, a DTD file provides the composition of the database's logical model and defines the syntax rules or valid tags for each type of XML document.

- The first line declares the root element, and lists the possible child elements of the root element.
- A plus (+) symbol on a child element means that child element occurs at least once.
- An asterisk (*) means the element occurs zero or more times.
- A question mark (?) means the child element is optional.
- The `#PCDATA` keyword represents the actual text data.

To be able to use a DTD file to define elements within an XML document, the DTD must be referenced within that XML document. The DTD can be referenced by many XML documents of the same type.

Although the use of DTDs is a great improvement for data sharing over the web, a DTD provides only descriptive information for understanding how the elements relate to one another. A DTD provides limited additional semantic value, such as data type support or data validation rules. To solve this problem, the W3C published an XML schema standard to provide a better way to describe XML data.

The **XML schema** is an advanced data definition language that is used to describe the structure (elements, data types, relationship types, ranges, and default values) of XML data documents. One of the main advantages of an XML schema is that it more closely maps to database terminology and features. Many vendors are adopting this new standard, and are supplying tools to translate DTD documents into XML Schema Definition (XSD) documents. Unlike a DTD document, which uses a unique syntax, an **XML schema definition (XSD)** file uses a syntax that resembles an XML document.

17.3.2 XML Presentation

One of the main benefits of XML is that it separates data structure from its presentation and processing. By separating data and presentation, you are able to present the same data in different ways – which is similar to having views in SQL. The **Extensible Style Language (XSL)** specification provides the mechanism to display XML data. XSL is used to define the rules by which XML data are formatted and displayed. The XSL specification is divided into two parts:

- **Extensible Style Language Transformations** Describe the general mechanism that is used to extract and process data from one XML document and enable its transformation within another document. Using XSLT, you can extract data from an XML document and convert it into a text file, an HTML web page, or a web page that is formatted for a mobile device. What the user sees in those cases is actually a view (or HTML representation) of the actual XML data. XSLT can also be used to extract certain elements from an XML document, or to transform an XML document into another XML document.
- **XSL Style Sheets** Define the presentation rules applied to XML elements. The XSL style sheet describes the formatting options to apply to XML elements when they are displayed on a browser, smartphone, and so on.

17.3.3 SQL/XML and XQuery

XML is used to transfer data from a Web-based application to the database and back again. SQL/XML and XQuery are two standard querying languages that are used to retrieve data from a relational database in the XML format. **XQuery 1.0** is the W3C language designed for querying XML data and it is relatively similar to SQL, except it was designed to query semi-structured XML data. **SQL/XML** is an extension of SQL that is part of the ANSI/ISO SQL 2011 standard.

Subqueries in SQL/XML are only designed to return one row, so if multiple rows are to be returned, they must be aggregated into one single value using the function `xmlagg()`.

An alternative approach to SQL/XML is XQuery, which is a language that can query, store, process, and exchange structured or semi-structured XML data. XQuery is used together with XPath, which is used to navigate through elements and attributes in an XML document.

17.4 Cloud Computing Services

Cloud computing is a computing model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computer resources that can be rapidly provisioned and released with minimal management effort or service provider interaction. **Cloud services** refer to the services provided by cloud computing, and allow any organisation to add information technology services such as applications, storage, processing power, databases, and infrastructure to its IT portfolio quickly and economically.

The technologies that make cloud computing work have been around for a few years now; these include the Web, messaging, virtualisation, remote desktop protocols, VPN, and XML.

- **Private Cloud** Internal cloud built by an organisation for the sole purpose of servicing its own needs. Often used by large, geographically dispersed organisations to add agility and flexibility to internal IT services. The cloud infrastructure could be managed by internal IT staff, or an external third party.
- **Community Cloud** Built by and for a specific group or organisations that share a common trade. The cloud infrastructure could be managed by internal IT staff, or an external third party.

17.4.1 Characteristics of Cloud Services

- **Ubiquitous access via Internet technologies** All cloud services use Internet and Web technologies to provision, deliver, and manage the services they provide. The basic requirement is that the device has access to the Internet.
- **Shared infrastructure** The cloud service infrastructure is shared by multiple users. Sharing is made possible by Web and virtualisation technologies.
- **Lower costs and variable procing**
- **Flexible and scalable services** Cloud services are built on infrastructure that is highly scalable, fault tolerant, and very reliable. The services can scale up and down on demand according to resource demands.
- **Dynamic provisioning** The consumer can quickly provision any needed resources, including servers, processing power, storage and email, by accessing the Web management dashboard and then adding and removing services on demand. This process can be automated via other services.

- **Service orientation** Cloud computing focuses on providing consumers with specific, well-defined services that use well-known interfaces. These interfaces hide the complexity from the end user, and can be delivered any time and anywhere.
- **Managed operations** Cloud computing minimises the need for extensive and expensive in-house IT staff. The system is managed by the cloud provider.

17.4.2 Types of Cloud Services

Cloud services often follow an 'à la carte' model; consumers can choose multiple service options according to their individual needs. These services can build on top of one another to provide sophisticated solutions.

Based on the types of services provided, cloud services can be classified by the following categories:

- **Software as a Service (SaaS)** The cloud service provider offers turnkey applications that run in the cloud. Consumers can run the provider's applications internally in their organisations via the Web or any mobile device. The consumer can customise certain aspects of the application, but cannot make changes to the application itself. The application is actually shared among users from multiple organisations.
- **Platform as a Service (PaaS)** The cloud service provider offers the capability to build and deploy consumer created applications using the provider's cloud infrastructure. The consumer can build, deploy, and manage applications using the provider's cloud tools, languages, and interfaces. However, the consumer does not manage the underlying cloud infrastructure.
- **Infrastructure as a Service (IaaS)** The cloud service provider offers consumers the ability to provision their own resources on demand; these resources include storage, servers, databases, processing units, and even a complete virtualised desktop. The consumer can then add or remove the resources as needed.

17.4.3 Cloud Services: Advantages and Disadvantages

Cloud Services Advantages

- **Low initial cost of entry** Lower costs of entry compared to building in-house.
- **Scalability/Elasticity** Easy to add and remove resources on demand.
- **Support for mobile computing**
- **Ubiquitous access** Consumers can access cloud resources from anywhere at any time, as long as they have Internet access.
- **High reliability and performance** Cloud providers build solid infrastructures that are otherwise difficult for the average organisation to leverage.
- **Fast provisioning** Resources can be provisioned on demand in a matter of minutes with minimal effort.
- **Managed Infrastructure** Most cloud implementations are managed by dedicated internal or external staff. This allows an organisation's IT staff to focus on other areas.

Cloud Computing Disadvantages

- **Issues of security, privacy, and compliance** Trusting sensitive company data to external entities is difficult for most data-cautious organisations.
- **Hidden costs of implementation and operation** It is hard to estimate bandwidth and data migration costs.
- **Data migration is a difficult and lengthy process** Migrating large amounts of data to and from the cloud infrastructure can be difficult and time-consuming.
- **Complex licensing schemes** Organisations that implement cloud services are faced with complex licensing schemes and complicated service-level agreements.
- **Loss of ownership and control** Companies that use cloud services are no longer in complete control of their data.
- **Organisation culture** End users tend to be resistant to change. Do the savings justify being dependent on a single provider? Will the cloud provider be around in ten years?
- **Difficult integration with internal IT system** Configuring the cloud services to integrate transparently with internal authentication and other internal services could be a daunting task.

17.4.4 SQL Data Services

SQL data services (SDS) refers to a cloud computing-based data management service that provides relational data storage, access, and management to companies of all sizes without the typically high costs of in-house hardware, software, infrastructure, and personnel. This type of service provides some unique benefits:

- **Hosted data management** SDS typically uses a cluster of database servers that provide a large subset of database functionality over the Internet to database administrators and users. Typically, features such as SQL queries, indexing, stored procedures, triggers, reporting, and analytical functions are available to end users. Other features, such as data synchronisation, data backup and restore, and data importing and exporting are available for administrative purposes.
- **Standard protocols** SDS uses standard data communication and relational data access protocols. Typically, these services encapsulate SQL networking protocols inside the TCP/IP networking protocol.
- **A common programming interface** SDS is transparent to application developers. Programmers continue to use familiar programming interfaces to manipulate the data. Programmers write embedded SQL code in their applications and connect to the database as if the data were stored locally instead of in a remote location on the Internet. One potential disadvantage, however, is that some specialised data types may not be supported by SDS.

SQL data services offer the following advantages when compared with in-house systems:

- Highly reliable and scalable relational database for a fraction of the cost.
- High level of failure tolerance because data are normally distributed and replicated among multiple servers.
- Dynamic and automatic load balancing.
- Automated data backup and disaster recovery included with the service.
- Dynamic creation and allocation of database processes and storage.

17.5 The Semantic Web

The Semantic Web was conceived by Tim Berners-Lee as a mechanism for describing concepts in a way that computers can actually understand. It is an extension of the current web in which information is given well-defined meaning, better enabling computers and people to work in cooperation.

The WWW represents information using a variety of formats, including images, multimedia, and natural language, which is easy for the majority of human beings to understand, as they comprehend the semantics of language. A traditional computer, however, can only understand the syntax of the language and cannot relate the meanings between two concepts between two concepts written using different natural languages that actually have the same meaning.

The Semantic Web is often referred to as a *Web of data*.

The aim is to produce a framework that allows all data to be shared and reused across applications, without any boundaries. The framework will establish common formats for integration and combination of data from different sources and develop a language for modelling how data relates to real-world objects. The framework is based on the **Resource Description Framework (RDF)**, which is a model that has a number of features for interchanging data over the WWW.