DATABASE SOLUTIONS (2nd Edition)

THOMAS M CONNOLLY & CAROLYN E BEGG

SOLUTIONS TO REVIEW QUESTIONS

Chapter 1 Introduction- Review questions

1.1 List four examples of database systems other than those listed in Section 1.1.

Some examples could be:

- A system that maintains component part details for a car manufacturer;
- An advertising company keeping details of all clients and adverts placed with them;
- A training company keeping course information and participants' details;
- An organization maintaining all sales order information.

1.2 Discuss the meaning of each of the following terms:

(a) data

For end users, this constitutes all the different values connected with the various objects/entities that are of concern to them.

(b) database

A shared collection of logically related data (and a description of this data), designed to meet the information needs of an organization.

(c) database management system

A software system that: enables users to define, create, and maintain the database and provides controlled access to this database.

(d) application program

A computer program that interacts with the database by issuing an appropriate request (typically an SQL statement) to the DBMS.

(e) data independence

This is essentially the separation of underlying file structures from the programs that operate on them, also called program-data independence.

(f) views.

A virtual table that does not necessarily exist in the database but is generated by the DBMS from the underlying base tables whenever it's accessed. These present only a subset of the database that is of particular interest to a user. Views can be customized, for example, field names may change, and they also provide a level of security preventing users from seeing certain data.

1.3 Describe the main characteristics of the database approach.

Focus is now on the data first, and then the applications. The structure of the data is now kept separate from the programs that operate on the data. This is held in the system catalog or data dictionary. Programs can now share data, which is no longer fragmented. There is also a reduction in redundancy, and achievement of program-data independence.

1.4 Describe the five components of the DBMS environment and discuss how they relate to each other.

(1) *Hardware*: The computer system(s) that the DBMS and the application programs run on. This can range from a single PC, to a single mainframe, to a network of computers.

(2) *Software*: The DBMS software and the application programs, together with the operating system, including network software if the DBMS is being used over a network.

(3) Data: The data acts as a bridge between the hardware and software components and the human components. As we've already said, the database contains both the operational data and the meta-data (the 'data about data').

(4) Procedures: The instructions and rules that govern the design and use of the database. This may include instructions on how to log on to the DBMS, make backup copies of the database, and how to handle hardware or software failures.

(5) People: This includes the database designers, database administrators (DBAs), application programmers, and the end-users.

1.5 Describe the problems with the traditional two-tier client-server architecture and discuss how these problems were overcome with the three-tier client-server architecture.

In the mid-1990s, as applications became more complex and potentially could be deployed to hundreds or thousands of end-users, the client side of this architecture gave rise to two problems:

- A 'fat' client, requiring considerable resources on the client's computer to run effectively (resources include disk space, RAM, and CPU power).
- A significant client side administration overhead.

By 1995, a new variation of the traditional two-tier client-server model appeared to solve these problems called the **three-tier client-server architecture**. This new architecture proposed three layers, each potentially running on a different platform:

(1) The *user interface* layer, which runs on the end-user's computer (the *client*).
(2) The *business logic and data processing* layer. This middle tier runs on a server and is often called the **application server**. One application server is designed to serve multiple clients.
(3) A DBMS, which stores the data required by the middle tier. This tier may run on a separate server called the **database server**.

The three-tier design has many advantages over the traditional two-tier design, such as:

- A 'thin' client, which requires less expensive hardware.
- Simplified application maintenance, as a result of centralizing the business logic for many end-users into a single application server. This eliminates the concerns of software distribution that are problematic in the traditional two-tier client-server architecture.
- Added modularity, which makes it easier to modify or replace one tier without affecting the other tiers.
- Easier load balancing, again as a result of separating the core business logic from the database functions. For example, a **Transaction Processing Monitor (TPM)** can be used to reduce the number of connections to the database server. (A TPM is a program that controls data transfer between clients and servers in order to provide a consistent environment for Online Transaction Processing (OLTP).)

An additional advantage is that the three-tier architecture maps quite naturally to the Web environment, with a Web browser acting as the 'thin' client, and a Web server acting as the application server. The three-tier client server architecture is illustrated in Figure 1.4.

1.6 Describe the functions that should be provided by a modern full-scale multi-user DBMS.

Data Storage, Retrieval and Update	Authorization Services
A User-Accessible Catalog	Support for Data Communication
Transaction Support	Integrity Services
Concurrency Control Services	Services to Promote Data Independence
Recovery Services	Utility Services

1.7 Of the functions described in your answer to Question 1.6, which ones do you think would not be needed in a standalone PC DBMS? Provide justification for your answer.

Concurrency Control Services - only single user.
Authorization Services - only single user, but may be needed if different individuals are to use the DBMS at different times.
Utility Services - limited in scope.
Support for Data Communication - only standalone system.

1.8 Discuss the advantages and disadvantages of DBMSs.

Some advantages of the database approach include control of data redundancy, data consistency, sharing of data, and improved security and integrity. Some disadvantages include complexity, cost, reduced performance, and higher impact of a failure.

Chapter 2 The Relational Model - Review questions

2.1 Discuss each of the following concepts in the context of the relational data model:

(a) relation

A table with columns and rows.

(b) attribute

A named column of a relation.

(c) domain

The set of allowable values for one or more attributes.

(d) tuple

A record of a relation.

(e) relational database.

A collection of normalized tables.

2.2 Discuss the properties of a relational table.

A relational table has the following properties:

- The table has a name that is distinct from all other tables in the database.
- Each cell of the table contains exactly one value. (For example, it would be wrong to store several telephone numbers for a single branch in a single cell. In other words, tables don't contain repeating groups of data. A relational table that satisfies this property is said to be *normalized* or in *first normal form*.)
- Each column has a distinct name.
- The values of a column are all from the same domain.
- The order of columns has no significance. In other words, provided a column name is moved along with the column values, we can interchange columns.
- Each record is distinct; there are no duplicate records.

- The order of records has no significance, theoretically.

2.3 Discuss the differences between the candidate keys and the primary key of a table. Explain what is meant by a foreign key. How do foreign keys of tables relate to candidate keys? Give examples to illustrate your answer.

The primary key is the candidate key that is selected to identify tuples uniquely within a relation. A foreign key is an attribute or set of attributes within one relation that matches the candidate key of some (possibly the same) relation.

2.4 What does a null represent?

Represents a value for a column that is currently unknown or is not applicable for this record.

2.5 Define the two principal integrity rules for the relational model. Discuss why it is desirable to enforce these rules.

Entity integrity In a base table, no column of a primary key can be null.

Referential integrity If a foreign key exists in a table, either the foreign key value must match a candidate key value of some record in its home table or the foreign key value must be wholly null.

Chapter 3 SQL and QBE - Review questions

3.1 What are the two major components of SQL and what function do they serve?

A data definition language (DDL) for defining the database structure.

A data manipulation language (DML) for retrieving and updating data.

3.2 Explain the function of each of the clauses in the SELECT statement. What restrictions are imposed on these clauses?

FROM specifies the table or tables to be used;

WHERE filters the rows subject to some condition;

GROUP BY forms groups of rows with the same column value;

HAVING filters the groups subject to some condition;

SELECT specifies which columns are to appear in the output;

ORDER BY specifies the order of the output.

3.3 What restrictions apply to the use of the aggregate functions within the SELECT statement? How do nulls affect the aggregate functions?

An aggregate function can be used only in the SELECT list and in the HAVING clause.

Apart from COUNT(*), each function eliminates nulls first and operates only on the remaining non-null values. COUNT(*) counts all the rows of a table, regardless of whether nulls or duplicate values occur.

3.4 Explain how the GROUP BY clause works. What is the difference between the WHERE and HAVING clauses?

SQL first applies the WHERE clause. Then it conceptually arranges the table based on the grouping column(s). Next, applies the HAVING clause and finally orders the result according to the ORDER BY clause.

WHERE filters rows subject to some condition; HAVING filters groups subject to some condition.

3.5 What is the difference between a subquery and a join? Under what circumstances would you not be able to use a subquery?

With a subquery, the columns specified in the SELECT list are restricted to one table. Thus, cannot use a subquery if the SELECT list contains columns from more than one table.

3.6 What is QBE and what is the relationship between QBE and SQL?

QBE is an alternative, graphical-based, 'point-and-click' way of querying the database, which is particularly suited for queries that are not too complex, and can be expressed in terms of a few tables. QBE has acquired the reputation of being one of the easiest ways for non-technical users to obtain information from the database.

QBE queries are converted into their equivalent SQL statements before transmission to the DBMS server.

Chapter 4 Database Systems Development Lifecycle - Review questions

4.1 Describe what is meant by the term 'software crisis'.

The past few decades has witnessed the dramatic rise in the number of software applications. Many of these applications proved to be demanding, requiring constant maintenance. This maintenance involved correcting faults, implementing new user requirements, and modifying the software to run on new or upgraded platforms. With so much software around to support, the effort spent on maintenance began to absorb resources at an alarming rate. As a result, many major software projects were late, over budget, and the software produced was unreliable, difficult to maintain, and performed poorly. This led to what has become known as the 'software crisis'. Although this term was first used in the late 1960s, more than 30 years later, the crisis is still with us. As a result, some people now refer to the software crisis as the 'software depression'.

4.2 Discuss the relationship between the information systems lifecycle and the database system development lifecycle.

An information system is the resources that enable the collection, management, control, and dissemination of data/information throughout a company. The database is a fundamental component of an information system. The lifecycle of an information system is inherently linked to the lifecycle of the database that supports it.

Typically, the stages of the information systems lifecycle include: planning, requirements collection and analysis, design (including database design), prototyping, implementation, testing, conversion, and operational maintenance. As a database is a fundamental component of the larger company-wide information system, the database system development lifecycle is inherently linked with the information systems lifecycle.

4.3 Briefly describe the stages of the database system development lifecycle.

See **Figure 4.1** Stages of the database system development lifecycle.

Database planning is the management activities that allow the stages of the database system development lifecycle to be realized as efficiently and effectively as possible.

System definition involves identifying the scope and boundaries of the database system including its major user views. A user view can represent a job role or business application area.

Requirements collection and analysis is the process of collecting and analyzing information about the company that is to be supported by the database system, and using this information to identify the requirements for the new system.

There are three approaches to dealing with multiple user views, namely the centralized approach, the view integration approach, and a combination of both. The **centralized approach** involves collating the users' requirements for different user views into a single list of requirements. A data model representing all the user views is created during the database design stage. The **view integration approach** involves leaving the users' requirements for each user view as separate lists of requirements. Data models representing each user view are created and then merged at a later stage of database design.

Database design is the process of creating a design that will support the company's mission statement and mission objectives for the required database. This stage includes the logical and physical design of the database.

The aim of **DBMS selection** is to select a system that meets the current and future requirements of the company, balanced against costs that include the purchase of the DBMS product and any additional software/hardware, and the costs associated with changeover and training.

Application design involves designing the user interface and the application programs that use and process the database. This stage involves two main activities: transaction design and user interface design.

Prototyping involves building a working model of the database system, which allows the designers or users to visualize and evaluate the system.

Implementation is the physical realization of the database and application designs.

Data conversion and loading involves transferring any existing data into the new database and converting any existing applications to run on the new database.

Testing is the process of running the database system with the intent of finding errors.

Operational maintenance is the process of monitoring and maintaining the system following installation.

4.4 Describe the purpose of creating a mission statement and mission objectives for the required database during the database planning stage.

The mission statement defines the major aims of the database system, while each mission objective identifies a particular task that the database must support.

4.5 Discuss what a user view represents when designing a database system.

A user view defines what is required of a database system from the perspective of a particular job (such as Manager or Supervisor) or business application area (such as marketing, personnel, or stock control).

4.6 Compare and contrast the centralized approach and view integration approach to managing the design of a database system with multiple user views.

An important activity of the requirements collection and analysis stage is deciding how to deal with the situation where there is more than one user view. There are three approaches to dealing with multiple user views:

- the centralized approach,
- the view integration approach, and
- a combination of both approaches.

Centralized approach

Requirements for each user view are merged into a single list of requirements for the new database system. A logical data model representing all user views is created during the database design stage.

The **centralized approach** involves collating the requirements for different user views into a single list of requirements. A data model representing all user views is created in the database design stage. A diagram representing the management of user views 1 to 3 using the centralized approach is shown in Figure 4.4. Generally, this approach is preferred when there is a significant overlap in requirements for each user view and the database system is not overly complex.

See **Figure 4.4** The centralized approach to managing multiple user views 1 to 3.

View integration approach

Requirements for each user view remain as separate lists. Data models representing each user view are created and then merged later during the database design stage.

The **view integration approach** involves leaving the requirements for each user view as separate lists of requirements. We create data models representing each user view. A data model that represents a single user view is called a **local logical data model**. We then merge the local data models to create a **global logical data model** representing all user views of the company.

A diagram representing the management of user views 1 to 3 using the view integration approach is shown in Figure 4.5. Generally, this approach is preferred when there are significant differences between user views and the database system is sufficiently complex to justify dividing the work into more manageable parts.

See **Figure 4.5** The view integration approach to managing multiple user views 1 to 3.

For some complex database systems it may be appropriate to use a combination of both the centralized and view integration approaches to managing multiple user views. For example, the requirements for two or more users views may be first merged using the centralized approach and then used to create a **local logical data model**. (Therefore in this situation the local data model represents not just a single user view but the number of user views merged using the centralized approach). The local data models representing one or more user views are then

merged using the view integration approach to form the **global logical data model** representing all user views.

4.7 Explain why it is necessary to select the target DBMS before beginning the physical database design phase.

Database design is made up of two main phases called logical and physical design. During logical database design, we identify the important objects that need to be represented in the database and the relationships between these objects. During physical database design, we decide how the logical design is to be physically implemented (as tables) in the target DBMS. Therefore it is necessary to have selected the target DBMS before we are able to proceed to physical database design.

See Figure 4.1 Stages of the database system development lifecycle.

4.8 Discuss the two main activities associated with application design.

The database and application design stages are parallel activities of the database system development lifecycle. In most cases, we cannot complete the application design until the design of the database itself has taken place. On the other hand, the database exists to support the applications, and so there must be a flow of information between application design and database design.

The two main activities associated with the application design stage is the design of the user interface and the application programs that use and process the database.

We must ensure that all the functionality stated in the requirements specifications is present in the application design for the database system. This involves designing the interaction between the user and the data, which we call *transaction design*. In addition to designing how the required functionality is to be achieved, we have to design an appropriate *user interface* to the database system.

4.9 Describe the potential benefits of developing a prototype database system.

The purpose of developing a prototype database system is to allow users to use the prototype to identify the features of the system that work well, or are inadequate, and if possible to suggest

improvements or even new features for the database system. In this way, we can greatly clarify the requirements and evaluate the feasibility of a particular system design. Prototypes should have the major advantage of being relatively inexpensive and quick to build.

4.10 Discuss the main activities associated with the implementation stage.

The database implementation is achieved using the *Data Definition Language (DDL)* of the selected DBMS or a graphical user interface (GUI), which provides the same functionality while hiding the low-level DDL statements. The DDL statements are used to create the database structures and empty database files. Any specified user views are also implemented at this stage.

The application programs are implemented using the preferred **third or fourth generation language (3GL or 4GL)**. Parts of these application programs are the database transactions, which we implement using the *Data Manipulation Language (DML)* of the target DBMS, possibly embedded within a host programming language, such as Visual Basic (VB), VB.net, Python, Delphi, C, C++, C#, Java, COBOL, Fortran, Ada, or Pascal. We also implement the other components of the application design such as menu screens, data entry forms, and reports. Again, the target DBMS may have its own fourth generation tools that allow rapid development of applications through the provision of non-procedural query languages, reports generators, forms generators, and application generators.

Security and integrity controls for the application are also implemented. Some of these controls are implemented using the DDL, but others may need to be defined outside the DDL using, for example, the supplied DBMS utilities or operating system controls.

4.11 Describe the purpose of the data conversion and loading stage.

This stage is required only when a new database system is replacing an old system. Nowadays, it's common for a DBMS to have a utility that loads existing files into the new database. The utility usually requires the specification of the source file and the target database, and then automatically converts the data to the required format of the new database files. Where applicable, it may be possible for the developer to convert and use application programs from the old system for use by the new system.

4.12 Explain the purpose of testing the database system.

Before going live, the newly developed database system should be thoroughly tested. This is achieved using carefully planned test strategies and realistic data so that the entire testing process is methodically and rigorously carried out. Note that in our definition of testing we have not used the commonly held view that testing is the process of demonstrating that faults are not present. In fact, testing cannot show the absence of faults; it can show only that software faults are present. If testing is conducted successfully, it will uncover errors in the application programs and possibly the database structure. As a secondary benefit, testing demonstrates that the database and the application programs *appear* to be working according to their specification and that performance requirements *appear* to be satisfied. In addition, metrics collected from the testing stage provides a measure of software reliability and software quality.

As with database design, the users of the new system should be involved in the testing process. The ideal situation for system testing is to have a test database on a separate hardware system, but often this is not available. If real data is to be used, it is essential to have backups taken in case of error.

Testing should also cover usability of the database system. Ideally, an evaluation should be conducted against a usability specification. Examples of criteria that can be used to conduct the evaluation include (Sommerville, 2000):

- Learnability - How long does it take a new user to become productive with the system?
- Performance - How well does the system response match the user's work practice?
- Robustness - How tolerant is the system of user error?
- Recoverability - How good is the system at recovering from user errors?
- Adapatability - How closely is the system tied to a single model of work?

Some of these criteria may be evaluated in other stages of the lifecycle. After testing is complete, the database system is ready to be 'signed off' and handed over to the users.

4.13 What are the main activities associated with operational maintenance stage.

In this stage, the database system now moves into a maintenance stage, which involves the following activities:

- Monitoring the performance of the database system. If the performance falls below an acceptable level, the database may need to be tuned or reorganized.

- Maintaining and upgrading the database system (when required). New requirements are incorporated into the database system through the preceding stages of the lifecycle.

Chapter 5 Database Administration and Security - Review questions

5.1 Define the purpose and tasks associated with data administration and database administration.

Data administration is the management and control of the corporate data, including database planning, development and maintenance of standards, policies and procedures, and logical database design.

Table 5.1 Data administration tasks.

Selecting appropriate productivity tools
Assisting in the development of the corporate IT/IS and business strategies
Undertaking feasibility studies and planning for database development
Developing a corporate data model
Determining the organization's data requirements
Setting data collection standards and establishing data formats
Estimating volumes of data and likely growth
Determining patterns and frequencies of data usage
Determining data access requirements and safeguards for both legal and corporate requirements
Undertaking logical database design
Liaising with database administration staff and application developers to ensure applications meet all stated requirements
Educating users on data standards and legal responsibilities
Keeping up to date with IT/IS and business developments
Ensuring documentation is complete, including the corporate data model, standards, policies, procedures, and controls on end-users
Managing the data dictionary
Liaising with end-users and database administration staff to determine new requirements and to resolve data access or performance problems
Developing a security policy

Database administration is the management and control of the physical realization of the corporate database system, including physical database design and implementation, setting security and integrity controls, monitoring system performance, and reorganizing the database as necessary.

Table 5.2 Database administration tasks.

Evaluating and selecting DBMS products
Undertaking physical database design
Implementing a physical database design using a target DBMS
Defining security and integrity constraints
Liaising with database system developers
Developing test strategies
Training users
Responsible for 'signing off' the implemented database system
Monitoring system performance and tuning the database, as appropriate
Performing backups routinely
Ensuring recovery mechanisms and procedures are in place
Ensuring documentation is complete, including in-house produced material
Keeping up to date with software and hardware developments and costs, and installing updates as necessary

5.2 Compare and contrast the main tasks carried out by the DA and DBA.

The *Data Administrator (DA)* and *Database Administrator (DBA)* are responsible for managing and controlling the activities associated with the corporate data and the corporate database, respectively. The DA is more concerned with the early stages of the lifecycle, from planning through to logical database design. In contrast, the DBA is more concerned with the later stages, from application/physical database design to operational maintenance. Depending on the size and complexity of the organization and/or database system the DA and DBA can be the responsibility of one or more people.

Table 5.3 Data/Database administration – main task differences.

Data administration	Database administration
Involved in strategic IS planning	Evaluates new DBMSs
Determines long-term goals	Executes plans to achieve goals
Determines standards, policies, and procedures	Enforces standards, policies, and procedures
Determines data requirements	Implements data requirements
Develops logical database design	Develops physical database design
Develops and maintains corporate data model	Implements physical database design
Coordinates database development	Monitors and controls database use
Managerial orientation	Technical orientation
DBMS independent	DBMS dependent

5.3 Explain the purpose and scope of database security.

Security considerations do not only apply to the data held in a database. Breaches of security may affect other parts of the system, which may in turn affect the database. Consequently, database security encompasses hardware, software, people, and data. To effectively implement security requires appropriate controls, which are defined in specific mission objectives for the system. This need for security, while often having been neglected or overlooked in the past, is now increasingly recognized by organizations. The reason for this turn-around is due to the increasing amounts of crucial corporate data being stored on computer and the acceptance that any loss or unavailability of this data could be potentially disastrous.

5.4 List the main types of threat that could affect a database system, and for each, describe the possible outcomes for an organization.

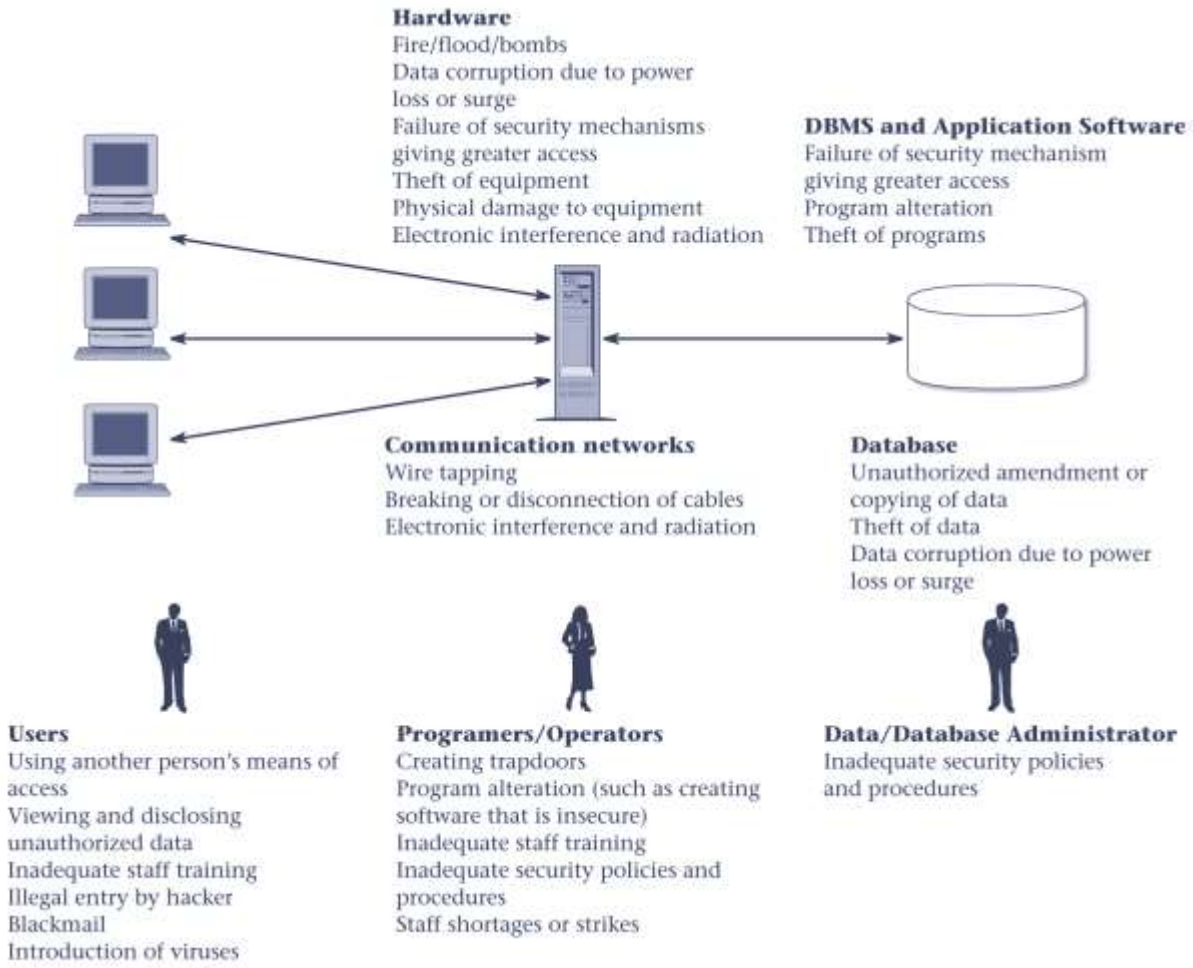


Figure 5.1 A summary of the potential threats to computer systems.

5.5 Explain the following in terms of providing security for a database:

authorization;

views;

backup and recovery;

integrity;

encryption;

RAID.

Authorization

Authorization is the granting of a right or privilege that enables a subject to have legitimate access to a system or a system's object. Authorization controls can be built into the software, and govern not only what database system or object a specified user can access, but also what the user may do with it. The process of authorization involves authentication of a subject

requesting access to an object, where 'subject' represents a user or program and 'object' represents a database table, view, procedure, trigger, or any other object that can be created within the database system.

Views

A view is a *virtual table* that does not necessarily exist in the database but can be produced upon request by a particular user, at the time of request. The view mechanism provides a powerful and flexible security mechanism by hiding parts of the database from certain users. The user is not aware of the existence of any columns or rows that are missing from the view. A view can be defined over several tables with a user being granted the appropriate privilege to use it, but not to use the base tables. In this way, using a view is more restrictive than simply having certain privileges granted to a user on the base table(s).

Backup and recovery

Backup is the process of periodically taking a copy of the database and log file (and possibly programs) onto offline storage media. A DBMS should provide backup facilities to assist with the recovery of a database following failure. To keep track of database transactions, the DBMS maintains a special file called a log file (or journal) that contains information about all updates to the database. It is always advisable to make backup copies of the database and log file at regular intervals and to ensure that the copies are in a secure location. In the event of a failure that renders the database unusable, the backup copy and the details captured in the log file are used to restore the database to the latest possible consistent state. Journaling is the process of keeping and maintaining a log file (or journal) of all changes made to the database to enable recovery to be undertaken effectively in the event of a failure.

Integrity constraints

Contribute to maintaining a secure database system by preventing data from becoming invalid, and hence giving misleading or incorrect results.

Encryption

Is the encoding of the data by a special algorithm that renders the data unreadable by any program without the decryption key. If a database system holds particularly sensitive data, it may be deemed necessary to encode it as a precaution against possible external threats or

attempts to access it. Some DBMSs provide an encryption facility for this purpose. The DBMS can access the data (after decoding it), although there is degradation in performance because of the time taken to decode it. Encryption also protects data transmitted over communication lines. There are a number of techniques for encoding data to conceal the information; some are termed irreversible and others reversible. *Irreversible techniques*, as the name implies, do not permit the original data to be known. However, the data can be used to obtain valid statistical information. *Reversible techniques* are more commonly used. To transmit data securely over insecure networks requires the use of a cryptosystem, which includes:

- an encryption key to encrypt the data (plaintext);
- an encryption algorithm that, with the encryption key, transforms the plain text into ciphertext;
- a decryption key to decrypt the ciphertext;
- a decryption algorithm that, with the decryption key, transforms the ciphertext back into plain text.

Redundant Array of Independent Disks (RAID)

RAID works by having a large disk array comprising an arrangement of several independent disks that are organized to improve reliability and at the same time increase performance. The hardware that the DBMS is running on must be *fault-tolerant*, meaning that the DBMS should continue to operate even if one of the hardware components fails. This suggests having redundant components that can be seamlessly integrated into the working system whenever there is one or more component failures. The main hardware components that should be fault-tolerant include disk drives, disk controllers, CPU, power supplies, and cooling fans. Disk drives are the most vulnerable components with the shortest times between failures of any of the hardware components.

One solution is the use of Redundant Array of Independent Disks (RAID) technology. RAID works by having a large disk array comprising an arrangement of several independent disks that are organized to improve reliability and at the same time increase performance.

Chapter 6 Fact-Finding - Review questions

6.1 Briefly describe what the process of fact-finding attempts to achieve for a database developer.

Fact-finding is the formal process of using techniques such as interviews and questionnaires to collect facts about systems, requirements, and preferences.

The database developer uses fact-finding techniques at various stages throughout the database systems lifecycle to capture the necessary facts to build the required database system. The necessary facts cover the business and the users of the database system, including the terminology, problems, opportunities, constraints, requirements, and priorities. These facts are captured using fact-finding techniques.

6.2 Describe how fact-finding is used throughout the stages of the database system development lifecycle.

There are many occasions for fact-finding during the database system development lifecycle. However, fact-finding is particularly crucial to the early stages of the lifecycle, including the database planning, system definition, and requirements collection and analysis stages. It's during these early stages that the database developer learns about the terminology, problems, opportunities, constraints, requirements, and priorities of the business and the users of the system. Fact-finding is also used during database design and the later stages of the lifecycle, but to a lesser extent. For example, during physical database design, fact-finding becomes technical as the developer attempts to learn more about the DBMS selected for the database system. Also, during the final stage, operational maintenance, fact-finding is used to determine whether a system requires tuning to improve performance or further developed to include new requirements.

6.3 For each stage of the database system development lifecycle identify examples of the facts captured and the documentation produced.

Table 6.1 Examples of the data captured and the documentation produced for each stage of the database system development lifecycle.

Stage of database system development lifecycle	Examples of data captured	Examples of documentation produced
Database planning	Aims and objectives of database project	Mission statement and objectives of database system
System definition	Description of major user views (includes job roles and/or business application areas)	Definition of scope and boundary of database system; definition of user views to be supported
Requirements collection and analysis	Requirements for user views; systems specifications, including performance and security requirements	Users' requirements specifications and system specifications
Database design	Users' responses to checking the logical database design; functionality provided by target DBMS	Logical database design (includes ER diagram(s), data dictionary, and tables); physical database design
Application design	Users' responses to checking interface design	Application design (includes description of programs and user interface)
DBMS selection	Functionality provided by target DBMS	DBMS evaluation and recommendations
Prototyping	Users' responses to prototype	Modified users' requirements specifications and systems specification
Implementation	Functionality provided by target DBMS	
Data conversion and loading	Format of current data; data import capabilities of target DBMS	
Testing	Test results	Testing strategies used; analysis of test results
Operational maintenance	Performance testing results; new or changing user and system requirements	User manual; analysis of performance results; modified users' requirements and systems specification

6.4 A database developer normally uses several fact-finding techniques during a single database project. The five most commonly used techniques are examining documentation, interviewing, observing the business in operation, conducting research, and using questionnaires. Describe each fact-finding technique and identify the advantages and disadvantages of each.

Examining documentation can be useful when you're trying to gain some insight as to how the need for a database arose. You may also find that documentation can be helpful to provide information on the business (or part of the business) associated with the problem. If the problem relates to the current system there should be documentation associated with that system. Examining documents, forms, reports, and files associated with the current system, is a good way to quickly gain some understanding of the system.

Interviewing is the most commonly used, and normally most useful, fact-finding technique. You can interview to collect information from individuals face-to-face. There can be several objectives to using interviewing such as finding out facts, checking facts, generating user interest and feelings of involvement, identifying requirements, and gathering ideas and opinions.

Table 6.3 Advantages and disadvantages of using interviewing as a fact-finding technique.

Advantages	Disadvantages
Allows interviewer to follow up on interesting comments made by interviewee	Very time-consuming and costly, and therefore may be impractical
Allows interviewer to adapt or re-word questions during interview	Success is dependent on communication skills of interviewer
Allows interviewer to observe interviewee's body language	
Allows interviewee to respond freely and openly to questions	
Allows interviewee to feel part of project	

Observation is one of the most effective fact-finding techniques you can use to understand a system. With this technique, you can either participate in, or watch a person perform activities to learn about the system. This technique is particularly useful when the validity of data

collected through other methods is in question or when the complexity of certain aspects of the system prevents a clear explanation by the end-users.

Table 6.4 Advantages and disadvantages of using observation as a fact-finding technique.

Advantages	Disadvantages
Allows the validity of facts and data to be checked	People may knowingly or unknowingly perform differently when being observed
Observer can see exactly what is being done	May miss observing tasks involving different levels of difficulty or volume normally experienced during that time period
Observer can also obtain data describing the physical environment of the task	Some tasks may not always be performed in the manner in which they are observed
Relatively inexpensive	May be impractical
Observer can do work measurements	

A useful fact-finding technique is to **research** the application and problem. Computer trade journals, reference books, and the Internet are good sources of information. They can provide you with information on how others have solved similar problems, plus you can learn whether or not software packages exist to solve your problem.

Table 6.5 Advantages and disadvantages of using research as a fact-finding technique.

Advantages	Disadvantages
Can save time if solution already exists	Can be time-consuming
Researcher can see how others have solved similar problems or met similar requirements	Requires access to appropriate sources of information
Keeps researcher up to date with current developments	May ultimately not help in solving problem because problem is not documented elsewhere

Another fact-finding technique is to conduct surveys through **questionnaires**. Questionnaires are special-purpose documents that allow you to gather facts from a large number of people

while maintaining some control over their responses. When dealing with a large audience, no other fact-finding technique can tabulate the same facts as efficiently.

Table 6.6 Advantages and disadvantages of using questionnaires as a fact-finding technique.

Advantages	Disadvantages
People can complete and return questionnaires at their convenience	Number of respondents can be low, possibly only 5–10 percent (particularly if the postal service or e-mail is used to deliver the questionnaires)
Relatively inexpensive way to gather data from a large number of people	Questionnaires may be returned incomplete
People more likely to provide the real facts as responses can be kept confidential	No opportunity to adapt or re-word questions that may have been misinterpreted
Responses can be tabulated and analyzed quickly	Can't observe and analyze the respondent's body language
Can be delivered using various modes, including person-to-person, postal service, and e-mail	Can be time-consuming to prepare questionnaire

6.5 Describe the purpose of defining a mission statement and mission objectives for a database system.

The mission statement defines the major aims of the database system. Those driving the database project within the business (such as the Director and/or owner) normally define the mission statement. A mission statement helps to clarify the purpose of the database project and provides a clearer path towards the efficient and effective creation of the required database system.

Once the mission statement is defined, the next activity involves identifying the mission objectives. Each mission objective should identify a particular task that the database must support. The assumption is that if the database supports the mission objectives then the mission statement should be met. The mission statement and objectives may be accompanied with additional information that specifies, in general terms, the work to be done, the resources with which to do it, and the money to pay for it all.

6.6 What is the purpose of the systems definition stage?

The purpose of the system definition stage is to identify the scope and boundary of the database system and its major user views. Defining the scope and boundary of the database system helps to identify the main types of data mentioned in the interviews and a rough guide as to how this data is related. A user view represents the requirements that should be supported by a database system as defined by a particular job role (such as Manager or Assistant) or business application area (such as video rentals or stock control).

6.7 How do the contents of a users' requirements specification differ from a systems specification?

There are two main documents created during the requirements collection and analysis stage, namely the users' requirements specification and the systems specification.
The users' requirements specification describes in detail the data to be held in the database and how the data is to be used.
The systems specification describes any features to be included in the database system such as the required performance and the levels of security.

6.8 Describe one approach to deciding whether to use centralized, view integration, or a combination of both when developing a database system for multiple user views.

One way to help you make a decision whether to use the centralized, view integration, or a combination of both approaches to manage multiple user views is to examine the overlap in terms of the data used between the user views identified during the system definition stage.

It's difficult to give precise rules as to when it's appropriate to use the centralized or view integration approaches. As the database developer, you should base your decision on an assessment of the complexity of the database system and the degree of overlap between the various user views. However, whether you use the centralized or view integration approach or a mixture of both to build the underlying database, ultimately you need to create the original user views for the working database system.

Chapter 7 Entity-Relationship Modeling - Review questions

7.1 Describe what entities represent in an ER model and provide examples of entities with a physical or conceptual existence.

Entity is a set of objects with the same properties, which are identified by a user or company as having an independent existence. Each object, which should be uniquely identifiable within the set, is called an entity occurrence. An entity has an independent existence and can represent objects with a physical (or 'real') existence or objects with a conceptual (or 'abstract') existence.

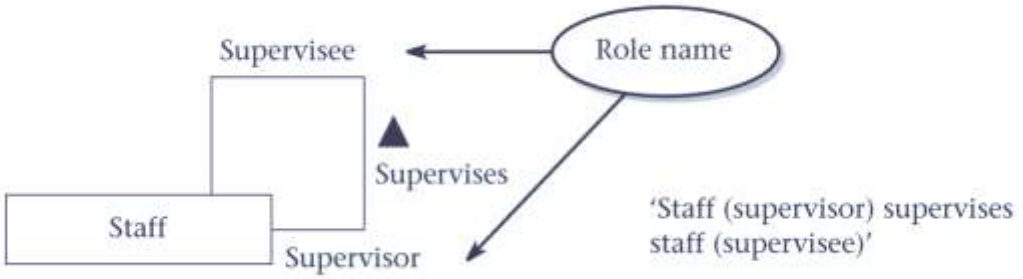
Physical existence	Conceptual existence
Member	Role
Video	Rental
Branch	Registration

7.2 Describe what relationships represent in an ER model and provide examples of unary, binary, and ternary relationships.

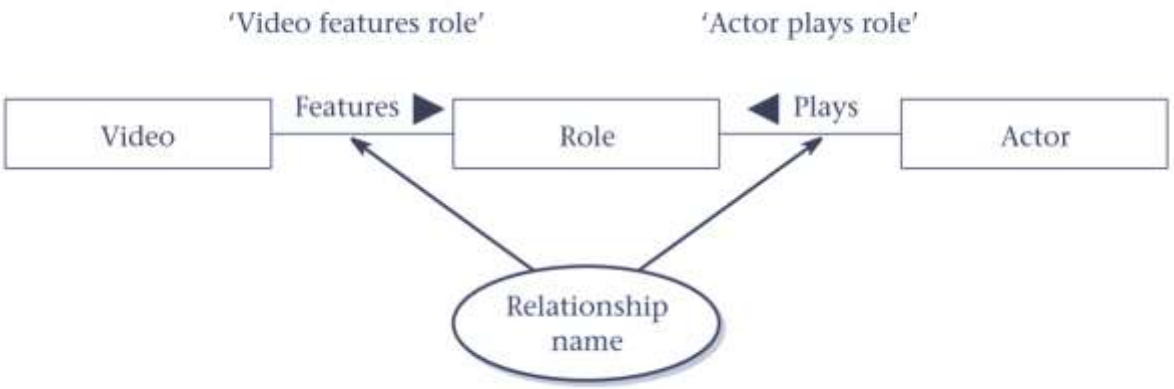
Relationship is a set of meaningful associations among entities. As with entities, each association should be uniquely identifiable within the set. A uniquely identifiable association is called a relationship occurrence. Each relationship is given a name that describes its function. For example, the Actor entity is associated with the Role entity through a relationship called *Plays*, and the Role entity is associated with the Video entity through a relationship called *Features*.

The entities involved in a particular relationship are referred to as participants. The number of participants in a relationship is called the degree and indicates the number of entities involved in a relationship. A relationship of degree one is called **unary**, which is commonly referred to as a *recursive* relationship. A unary relationship describes a relationship where the *same* entity participates more than once in *different* roles. An example of a unary relationship is *Supervises*,

which represents an association of staff with a supervisor where the supervisor is also a member of staff. In other words, the Staff entity participates twice in the *Supervises* relationship; the first participation as a supervisor, and the second participation as a member of staff who is supervised (supervisee). See Figure 7.5 for a diagrammatic representation of the *Supervises* relationship.



A relationship of degree two is called **binary**.



A relationship of a degree higher than binary is called a complex relationship. A relationship of degree three is called **ternary**. An example of a ternary relationship is *Registers* with three participating entities, namely Branch, Staff, and Member. The purpose of this relationship is to represent the situation where a member of staff registers a member at a particular branch, allowing for members to register at more than one branch, and members of staff to move between branches.

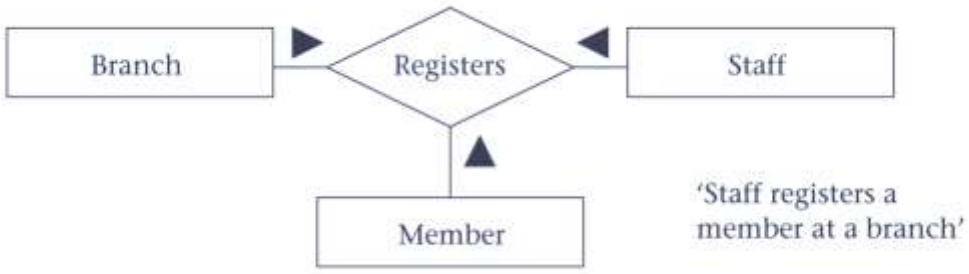


Figure 7.4 Example of a ternary relationship called *Registers*.

7.3 Describe what attributes represent in an ER model and provide examples of simple, composite, single-value, multi-value, and derived attributes.

An **attribute** is a property of an entity or a relationship.

Attributes represent what we want to know about entities. For example, a Video entity may be described by the catalogNo, title, category, dailyRental, and price attributes. These attributes hold values that describe each video occurrence, and represent the main source of data stored in the database.

Simple attribute is an attribute composed of a single component. Simple attributes cannot be further subdivided. Examples of simple attributes include the category and price attributes for a video.

Composite attribute is an attribute composed of multiple components. Composite attributes can be further divided to yield smaller components with an independent existence. For example, the name attribute of the Member entity with the value 'Don Nelson' can be subdivided into fName ('Don') and lName ('Nelson').

Single-valued attribute is an attribute that holds a single value for an entity occurrence. The majority of attributes are single-valued for a particular entity. For example, each occurrence of the Video entity has a single-value for the catalogNo attribute (for example, 207132), and therefore the catalogNo attribute is referred to as being single-valued.

Multi-valued attribute is an attribute that holds multiple values for an entity occurrence. Some attributes have multiple values for a particular entity. For example, each occurrence of the Video entity may have multiple values for the category attribute (for example, 'Children' and 'Comedy'), and therefore the category attribute in this case would be multi-valued. A multi-valued attribute may have a set of values with specified lower and upper limits. For example, the category attribute may have between one and three values.

Derived attribute is an attribute that represents a value that is derivable from the value of a related attribute, or set of attributes, not necessarily in the same entity. Some attributes may be related for a particular entity. For example, the age of a member of staff (age) is derivable from the date of birth (DOB) attribute, and therefore the age and DOB attributes are related. We refer to the age attribute as a derived attribute, the value of which is derived from the DOB attribute.

7.4 Describe what multiplicity represents for a relationship.

Multiplicity is the number of occurrences of one entity that may relate to a single occurrence of an associated entity.

7.5 What are business rules and how does multiplicity model these constraints?

Multiplicity constrains the number of entity occurrences that relate to other entity occurrences through a particular relationship. Multiplicity is a representation of the policies established by the user or company, and is referred to as a **business rule**. Ensuring that all appropriate business rules are identified and represented is an important part of modeling a company.

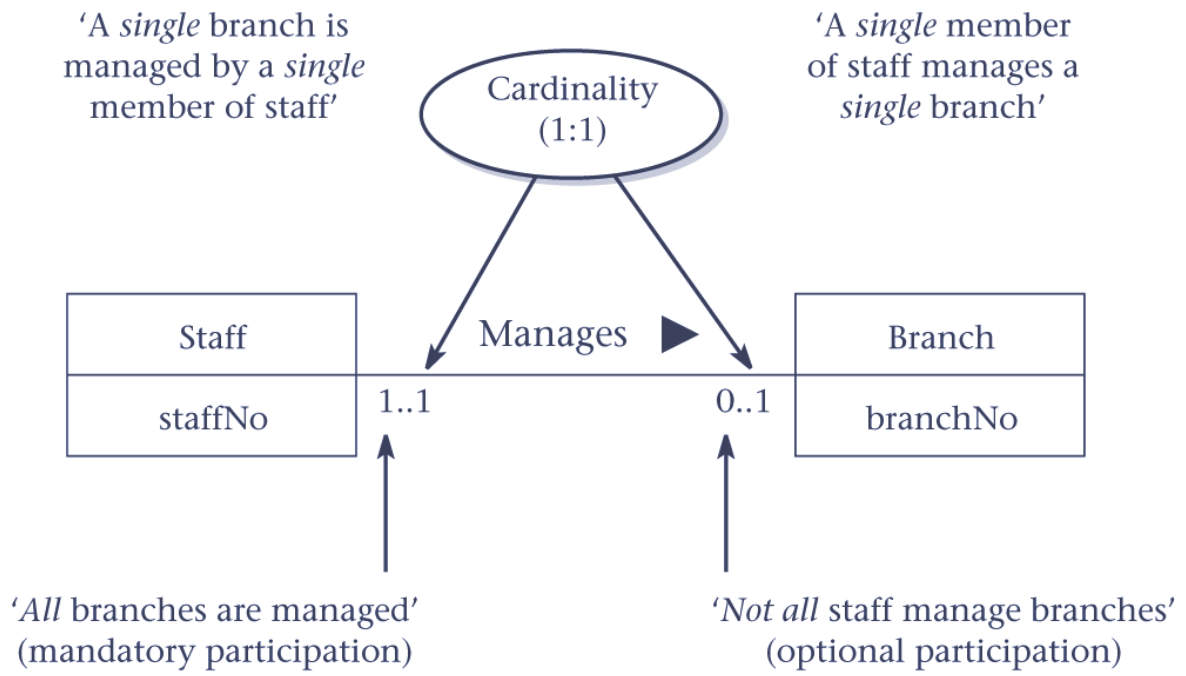
The multiplicity for a binary relationship is generally referred to as one-to-one (1:1), one-to-many (1:*), or many-to-many (*:*). Examples of three types of relationships include:

- A member of staff manages a branch.
- A branch has members of staff.
- Actors play in videos.

7.6 How does multiplicity represent both the cardinality and the participation constraints on a relationship?

Multiplicity actually consists of two separate constraints known as cardinality and participation. **Cardinality** describes the number of possible relationships for each participating entity. **Participation** determines whether all or only some entity occurrences participate in a relationship. The cardinality of a binary relationship is what we have been referring to as one-to-one, one-to-many, and many-to-many. A participation constraint represents whether all entity occurrences are involved in a particular relationship (*mandatory participation*) or only some

(*optional participation*). The cardinality and participation constraints for the Staff *Manages* Branch relationship are shown in Figure 7.11.



7.7 Provide an example of a relationship with attributes.

An example of a relationship with an attribute is the relationship called *PlaysIn*, which associates the Actor and Video entities. We may wish to record the character played by an actor in a given video. This information is associated with the *PlaysIn* relationship rather than the Actor or Video entities. We create an attribute called character to store this information and assign it to the *PlaysIn* relationship, as illustrated in Figure 7.12. Note, in this figure the character attribute is shown using the symbol for an entity; however, to distinguish between a relationship with an attribute and an entity, the rectangle representing the attribute is associated with the relationship using a dashed line.

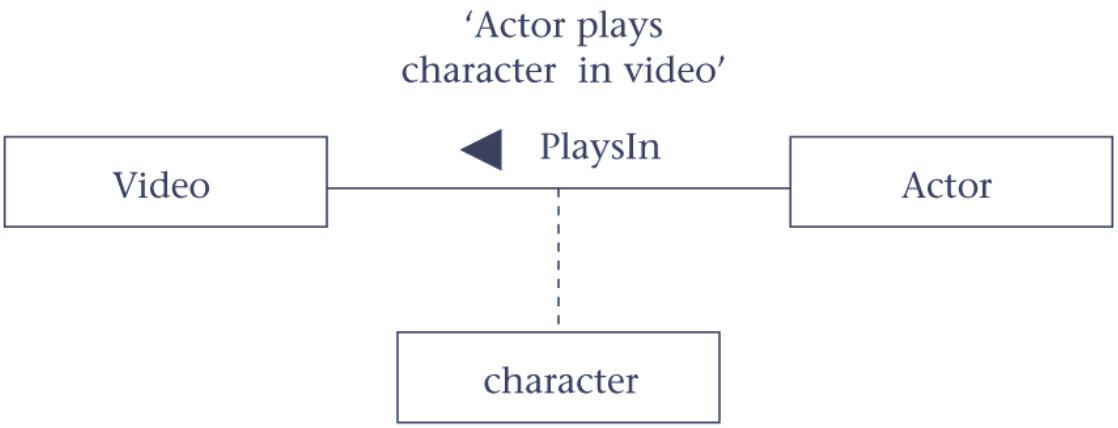


Figure 7.12 A relationship called *PlaysIn* with an attribute called character.

7.8 Describe how strong and weak entities differ and provide an example of each.

We can classify entities as being either strong or weak. A s**trong entity** is *not* dependent on the existence of another entity for its primary key. A **weak entity** is partially or wholly dependent on the existence of another entity, or entities, for its primary key. For example, as we can distinguish one actor from all other actors and one video from all other videos without the existence of any other entity, Actor and Video are referred to as being strong entities. In other words, the Actor and Video entities are strong because they have their own primary keys. An example of a weak entity called Role, which represents characters played by actors in videos. If we are unable to uniquely identify one Role entity occurrence from another without the existence of the Actor and Video entities, then Role is referred to as being a weak entity. In other words, the Role entity is weak because it has no primary key of its own.

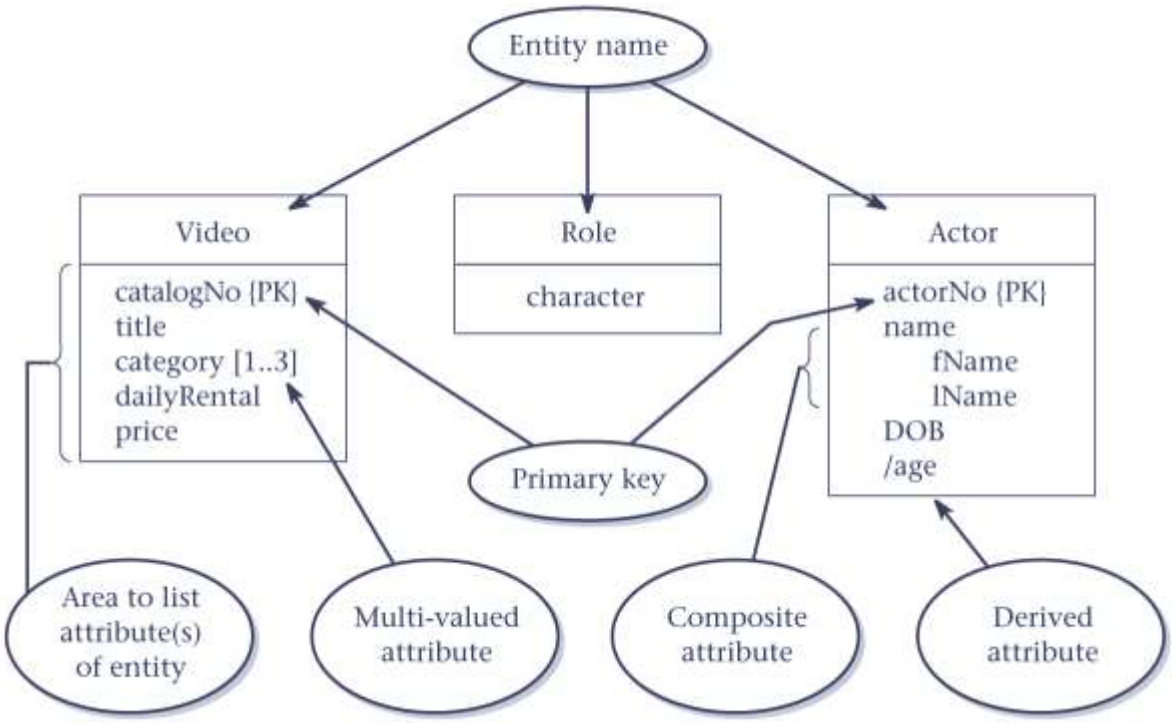


Figure 7.6 Diagrammatic representation of attributes for the Video, Role, and Actor entities.

Strong entities are sometimes referred to as *parent, owner*, or *dominant entities* and weak entities as *child, dependent*, or *subordinate entities.*

7.9 Describe how fan and chasm traps can occur in an ER model and how they can be resolved.

Fan and chasm traps are two types of connection traps that can occur in ER models. The traps normally occur due to a misinterpretation of the meaning of certain relationships. In general, to identify connection traps we must ensure that the meaning of a relationship (and the business rule that it represents) is fully understood and clearly defined. If we don't understand the relationships we may create a model that is not a true representation of the 'real world'.

A **fan trap** may occur when two entities have a 1:* relationship that fan out from a third entity, but the two entities should have a direct relationship between them to provide the necessary information. A fan trap may be resolved through the addition of a direct relationship between the two entities that were originally separated by the third entity.

A **chasm trap** may occur when an ER model suggests the existence of a relationship between entities, but the pathway does not exist between certain entity occurrences. More specifically, a chasm trap may occur where there is a relationship with optional participation that forms part of the pathway between the entities that are related. Again, a chasm trap may be resolved by the addition of a direct relationship between the two entities that were originally related through a pathway that included optional participation.

Chapter 8 Normalization – Review questions

8.1 Discuss how normalization may be used in database design.

Normalization can be used in database design in two ways: the first is to use normalization as a bottom-up approach to database design; the second is to use normalization in conjunction with ER modeling.

Using normalization as a **bottom-up approach** involves analyzing the associations between attributes and, based on this analysis, grouping the attributes together to form tables that represent entities and relationships. However, this approach becomes difficult with a large number of attributes, where it's difficult to establish all the important associations between the attributes. Alternatively, you can use a **top-down approach** to database design. In this approach, we use ER modeling to create a data model that represents the main entities and relationships. We then translate the ER model into a set of tables that represents this data. It's at this point that we use normalization to check whether the tables are well designed.

8.2 Describe the types of update anomalies that may occur on a table that has redundant data.

Tables that have redundant data may have problems called **update anomalies**, which are classified as insertion, deletion, or modification anomalies. See Figure 8.2 for an example of a table with redundant data called StaffBranch. There are two main types of insertion anomalies, which we illustrate using this table.

Insertion anomalies

(1) To insert the details of a new member of staff located at a given branch into the StaffBranch table, we must also enter the correct details for that branch. For example, to insert the details of a new member of staff at branch B002, we must enter the correct details of branch B002 so that the branch details are consistent with values for branch B002 in other records of the StaffBranch table. The data shown in the StaffBranch table is also shown in the Staff and Branch tables shown in Figure 8.1. These tables do have redundant data and do not suffer from this potential inconsistency, because for each staff member we only enter the appropriate branch number into the Staff table. In addition, the

details of branch B002 are recorded only once in the database as a single record in the Branch table.

(2) To insert details of a new branch that currently has no members of staff into the StaffBranch table, it's necessary to enter nulls into the staff-related columns, such as staffNo. However, as staffNo is the primary key for the StaffBranch table, attempting to enter nulls for staffNo violates entity integrity, and is not allowed. The design of the tables shown in Figure 8.1 avoids this problem because new branch details are entered into the Branch table separately from the staff details. The details of staff ultimately located at a new branch can be entered into the Staff table at a later date.

Deletion anomalies

If we delete a record from the StaffBranch table that represents the last member of staff located at a branch, the details about that branch are also lost from the database. For example, if we delete the record for staff Art Peters (S0415) from the StaffBranch table, the details relating to branch B003 are lost from the database. The design of the tables in Figure 8.1 avoids this problem because branch records are stored separately from staff records and only the column branchNo relates the two tables. If we delete the record for staff Art Peters (S0415) from the Staff table, the details on branch B003 in the Branch table remain unaffected.

Modification anomalies

If we want to change the value of one of the columns of a particular branch in the StaffBranch table, for example the telephone number for branch B001, we must update the records of all staff located at that branch. If this modification is not carried out on all the appropriate records of the StaffBranch table, the database will become inconsistent. In this example, branch B001 would have different telephone numbers in different staff records.

The above examples illustrate that the Staff and Branch tables of Figure 8.1 have more desirable properties than the StaffBranch table of Figure 8.2. In the following sections, we examine how normal forms can be used to formalize the identification of tables that have desirable properties from those that may potentially suffer from update anomalies.

8.3 Describe the characteristics of a table that violates first normal form (1NF) and then describe how such a table is converted to 1NF.

The rule for **first normal form (1NF)** is a table in which the intersection of every column and record contains only *one* value. In other words a table that contains more than one atomic value in the intersection of one or more column for one or more records is not in 1NF. The non 1NF table can be converted to 1NF by restructuring original table by removing the column with the multi-values along with a copy of the primary key to create a new table. See Figure 8.4 for an example of this approach. The advantage of this approach is that the resultant tables may be in normal forms later that 1NF.

8.4 What is the minimal normal form that a relation must satisfy? Provide a definition for this normal form.

Only first normal form (1NF) is critical in creating appropriate tables for relational databases. All the subsequent normal forms are optional. However, to avoid the update anomalies discussed in Section 8.2, it's normally recommended that you proceed to third normal form (3NF).

First normal form (1NF) is a table in which the intersection of every column and record contains only *one* value.

8.5 Describe an approach to converting a first normal form (1NF) table to second normal form (2NF) table(s).

Second normal form applies only to tables with composite primary keys, that is, tables with a primary key composed of two or more columns. A 1NF table with a single column primary key is automatically in at least 2NF.

A **second normal form (2NF)** is a table that is already in 1NF and in which the values in each non-primary-key column can be worked out from the values in *all* the columns that makes up the primary key.

A table in 1NF can be converted into 2NF by removing the columns that can be worked out from only part of the primary key. These columns are placed in a new table along with a copy of the part of the primary key that they can be worked out from.

8.6 Describe the characteristics of a table in second normal form (2NF).

Second normal form (2NF) is a table that is already in 1NF and in which the values in each non-primary-key column can only be worked out from the values in *all* the columns that make up the primary key.

8.7 Describe what is meant by full functional dependency and describe how this type of dependency relates to 2NF. Provide an example to illustrate your answer.

The formal definition of **second normal form (2NF)** is a table that is in first normal form and every non-primary-key column is **fully functionally dependent** on the primary key. Full functional dependency indicates that if A and B are columns of a table, B is fully functionally dependent on A, if B is not dependent on any subset of A. If B is dependent on a subset of A, this is referred to as a **partial dependency**. If a partial dependency exists on the primary key, the table is not in 2NF. The partial dependency must be removed for a table to achieve 2NF.

See **Section 8.4** for an example.

8.8 Describe the characteristics of a table in third normal form (3NF).

Third normal form (3NF) is a table that is already in 1NF and 2NF, and in which the values in all non-primary-key columns can be worked out from *only* the primary key (or candidate key) column(s) and no other columns.

8.9 Describe what is meant by transitive dependency and describe how this type of dependency relates to 3NF. Provide an example to illustrate your answer.

The formal definition for **third normal form (3NF)** is a table that is in first and second normal forms and in which no non-primary-key column is **transitively dependent** on the primary key. Transitive dependency is a type of functional dependency that occurs when a particular type of relationship holds between columns of a table. For example, consider a table with columns A, B, and C. If B is functionally dependent on A (A → B) and C is functionally dependent on B (B → C), then C is transitively dependent on A via B (provided that A is not functionally dependent on B or C). If a transitive dependency exists on the primary key, the table is not in 3NF. The transitive dependency must be removed for a table to achieve 3NF.

See **Section 8.5** for an example.

Chapter 9 Logical Database Design – Step 1- Review questions

9.1 Describe the purpose of a design methodology.

A design methodology is a structured approach that uses procedures, techniques, tools, and documentation aids to support and facilitate the process of design.

9.2 Describe the main phases involved in database design.

Database design is made up of two main phases: logical and physical database design.

Logical database design is the process of constructing a model of the data used in a company based on a specific data model, but independent of a particular DBMS and other physical considerations.

In the logical database design phase we build the logical representation of the database, which includes identification of the important entities and relationships, and then translate this representation to a set of tables. The logical data model is a source of information for the physical design phase, providing the physical database designer with a vehicle for making tradeoffs that are very important to the design of an efficient database.

Physical database design is the process of producing a description of the implementation of the database on secondary storage; it describes the base tables, file organizations, and indexes used to achieve efficient access to the data, and any associated integrity constraints and security restrictions. In the physical database design phase we decide how the logical design is to be physically implemented in the target relational DBMS. This phase allows the designer to make decisions on how the database is to be implemented. Therefore, physical design is tailored to a specific DBMS.

9.3 Identify important factors in the success of database design.

The following are important factors to the success of database design:

- Work interactively with the users as much as possible.
- Follow a structured methodology throughout the data modeling process.
- Employ a data-driven approach.
- Incorporate structural and integrity considerations into the data models.
- Use normalization and transaction validation techniques in the methodology.
- Use diagrams to represent as much of the data models as possible.

- Use a database design language (DBDL).
- Build a data dictionary to supplement the data model diagrams.
- Be willing to repeat steps.

9.4 Discuss the important role played by users in the process of database design.

Users play an essential role in confirming that the logical database design is meeting their requirements. Logical database design is made up of two steps and at the end of each step (Steps 1.9 and 2.5) users are required to review the design and provide feedback to the designer. Once the logical database design has been 'signed off' by the users the designer can continue to the physical database design stage.

9.5 Discuss the main activities associated with each step of the logical database design methodology.

The logical database design phase of the methodology is divided into two main steps.

- In *Step 1* we create a data model and check that the data model has minimal redundancy and is capable of supporting user transactions. The output of this step is the creation of a logical data model, which is a complete and accurate representation of the company (or part of the company) that is to be supported by the database.
- In *Step 2* we map the ER model to a set of tables. The structure of each table is checked using normalization. Normalization is an effective means of ensuring that the tables are structurally consistent, logical, with minimal redundancy. The tables are also checked to ensure that they are capable of supporting the required transactions. The required integrity constraints on the database are also defined.

9.6 Discuss the main activities associated with each step of the physical database design methodology.

Physical database design is divided into six main steps:

- *Step 3* involves the design of the base tables and integrity constraints using the available functionality of the target DBMS.
- *Step 4* involves choosing the file organizations and indexes for the base tables. Typically, DBMSs provide a number of alternative file organizations for data, with the exception of PC DBMSs, which tend to have a fixed storage structure.

- *Step 5* involves the design of the user views originally identified in the requirements analysis and collection stage of the database system development lifecycle.
- Step 6 involves designing the security measures to protect the data from unauthorized access.
- *Step 7* considers relaxing the normalization constraints imposed on the tables to improve the overall performance of the system. This is a step that you should undertake only if necessary, because of the inherent problems involved in introducing redundancy while still maintaining consistency.
- *Step 8* is an ongoing process of monitoring and tuning the operational system to identify and resolve any performance problems resulting from the design and to implement new or changing requirements.

9.7 Discuss the purpose of Step 1 of logical database design.

Purpose of Step 1 is to build a logical data model of the data requirements of a company (or part of a company) to be supported by the database.
Each logical data model comprises:

- entities,
- relationships,
- attributes and attribute domains,
- primary keys and alternate keys,
- integrity constraints.

The logical data model is supported by documentation, including a data dictionary and ER diagrams, which you'll produce throughout the development of the model.

9.8 Identify the main tasks associated with Step 1 of logical database design.

Step 1 Create and check ER model

Step 1.1 Identify entities

Step 1.2 Identify relationships

Step 1.3 Identify and associate attributes with entities or relationships

Step 1.4 Determine attribute domains

Step 1.5 Determine candidate, primary, and alternate key attributes

Step 1.6 Specialize/Generalize entities (optional step)

Step 1.7 Check model for redundancy

Step 1.8 Check model supports user transactions

Step 1.9 Review model with users

9.9 Discuss an approach to identifying entities and relationships from a users' requirements specification.

Identifying entities

One method of identifying entities is to examine the users' requirements specification. From this specification, you can identify nouns or noun phrases that are mentioned (for example, staff number, staff name, catalog number, title, daily rental rate, purchase price). You should also look for major objects such as people, places, or concepts of interest, excluding those nouns that are merely qualities of other objects.

For example, you could group staff number and staff name with an entity called Staff and group catalog number, title, daily rental rate, and purchase price with an entity called Video.

An alternative way of identifying entities is to look for objects that have an existence in their own right. For example, Staff is an entity because staff exists whether or not you know their names, addresses, and salaries. If possible, you should get the user to assist with this activity.

Identifying relationships

Having identified the entities, the next step is to identify all the relationships that exist between these entities. When you identify entities, one method is to look for nouns in the users' requirements specification. Again, you can use the grammar of the requirements specification to identify relationships. Typically, relationships are indicated by verbs or verbal expressions. For example:

- Branch *Has* Staff
- Branch *IsAllocated* VideoForRent
- VideoForRent *IsPartOf* RentalAgreement

The fact that the users' requirements specification records these relationships suggests that they are important to the users, and should be included in the model.

Take great care to ensure that all the relationships that are either explicit or implicit in the users' requirements specification are noted. In principle, it should be possible to check each pair

of entities for a potential relationship between them, but this would be a daunting task for a large system comprising hundreds of entities. On the other hand, it's unwise not to perform some such check. However, missing relationships should become apparent when you check the model supports the transactions that the users require. On the other hand, it is possible that an entity can have no relationship with other entities in the database but still play an important part in meeting the user's requirements.

9.10 Discuss an approach to identifying attributes from a users' requirements specification and the association of attributes with entities or relationships.

In a similar way to identifying entities, look for nouns or noun phrases in the users' requirements specification. The attributes can be identified where the noun or noun phrase is a property, quality, identifier, or characteristic of one of the entities or relationships that you've previously found.

By far the easiest thing to do when you've identified an entity or a relationship in the users' requirements specification is to consider "*What information are we required to hold on . . .?*". The answer to this question should be described in the specification. However, in some cases, you may need to ask the users to clarify the requirements. Unfortunately, they may give you answers that also contain other concepts, so users' responses must be carefully considered.

9.11 Discuss an approach to checking a data model for redundancy. Give an example to illustrate your answer.

There are three approaches to identifying whether a data model suffers from redundancy:

(1) re-examining one-to-one (1:1) relationships;

(2) removing redundant relationships;

(3) considering the time dimension when assessing redundancy.

However, to answer this question you need only describe one approach. We describe approach (1) here.

An example of approach (1)

In the identification of entities, you may have identified two entities that represent the same object in the company. For example, you may have identified two entities named Branch and

Outlet that are actually the same; in other words, Branch is a synonym for Outlet. In this case, the two entities should be merged together. If the primary keys are different, choose one of them to be the primary key and leave the other as an alternate key.

9.12 Describe two approaches to checking that a logical data model supports the transactions required by the user.

The two possible approaches to ensuring that the logical data model supports the required transactions, includes:

(1) Describing the transaction

Using the first approach, you check that all the information (entities, relationships, and their attributes) required by each transaction is provided by the model, by documenting a description of each transaction's requirements.

(2) Using transaction pathways

The second approach to validating the data model against the required transactions involves representing the pathway taken by each transaction directly on the ER diagram. Clearly, the more transactions that exist, the more complex this diagram would become, so for readability you may need several such diagrams to cover all the transactions.

9.13 Identify and describe the purpose of the documentation generated during Step 1 of logical database design.

Document entities

The data dictionary describes the entities including the entity name, description, aliases, and occurrences.

Entity name	Description	Aliases	Occurrence
Branch	Place of work	Outlet and Branch Outlet	One or more *StayHome* branches are located in main cities throughout the US.
Staff	General term describing all staff employed by *StayHome*	Employee	Each member of staff works at a particular branch.

Figure 9.2 Extract from the data dictionary for the Branch user views of *StayHome* showing a description of entities.

ER diagrams

Throughout the database design phase, ER diagrams are used whenever necessary, to help build up a picture of what you're attempting to model. Different people use different notations for ER diagrams. In this book, we've used the latest object-oriented notation called **UML (Unified Modeling Language)**, but other notations perform a similar function.

Document relationships

As you identify relationships, assign them names that are meaningful and obvious to the user, and also record relationship descriptions, and the multiplicity constraints in the data dictionary.

Entity	Multiplicity	Relationship	Multiplicity	Entity
Branch	1..*	Has	1..1	Staff
Branch	1..*	IsAllocated	1..1	VideoForRent
Staff	0..1	Manages	1..1	Branch
Staff	0..*	Supervises	0..1	Staff

Figure 9.7 Extract from the data dictionary for the Branch user views of *StayHome* showing descriptions of relationships.

Document attributes

As you identify attributes, assign them names that are meaningful and obvious to the user. Where appropriate, record the following information for each attribute:

- attribute name and description;
- data type and length;
- any aliases that the attribute is known by;
- whether the attribute must always be specified (in other words, whether the attribute allows or disallows nulls);
- whether the attribute is multi-valued;
- whether the attribute is composite, and if so, which simple attributes make up the composite attribute;
- whether the attribute is derived and, if so, how it should be computed;
- default values for the attribute (if specified).

Entity	Attributes	Description	Data type and length	Nulls	Multi-valued	...
Branch	branchNo	Uniquely identifies a branch	4 fixed characters	No	No	
	address:					
	street	Street of branch address	30 variable characters	No	No	
	city	City of branch address	20 variable characters	No	No	
	state	State of branch address	2 fixed characters	No	No	
	zipCode	Zip code of branch address	5 variable characters	No	No	
	telNo	Telephone numbers of branch	10 variable characters	No	Yes	
Staff	staffNo	Uniquely identifies a member of staff	5 fixed characters	No	No	
	name	Name of staff member	30 variable characters	No	No	

Figure 9.8 Extract from the data dictionary for the Branch user views of *StayHome* showing descriptions of attributes.

Document attribute domains

As you identify attribute domains, record their names and characteristics in the data dictionary. Update the data dictionary entries for attributes to record their domain in place of the data type and length information.

Document candidate, primary, and alternate keys

Record the identification of candidate, primary, and alternate keys (when available) in the data dictionary.

Entity	Attributes	Description	Key	Nulls	...
Branch	branchNo	Uniquely identifies a branch	Primary key	No	
	address:				
	street	Street of branch address		No	
	city	City of branch address		No	
	state	State of branch address		No	
	zipCode	Zip code of branch address	Alternate key	No	
	telNo	Telephone numbers of branch		No	
Staff	staffNo	Uniquely identifies a member of staff	Primary key	No	
	name	Name of staff member		No	

Figure 9.10 Extract from the data dictionary for the Branch user views of *StayHome* showing attributes with primary and alternate keys identified.

Document entities

You now have a logical data model that represents the database requirements of the company (or part of the company). The logical data model is checked to ensure that the model supports the required transactions. This process creates documentation that ensures that all the information (entities, relationships, and their attributes) required by each transaction is provided by the model, by documenting a description of each transaction's requirements. Alternative approach to validating the data model against the required transactions involves representing the pathway taken by each transaction directly on the ER diagram. Clearly, the more transactions that exist, the more complex this diagram would become, so for readability you may need several such diagrams to cover all the transactions.

Chapter 10 Logical Database Design – Step 2 – Review questions

10.1 Describe the main purpose and tasks of Step 2 of the logical database design methodology.

To create tables for the logical data model and to check the structure of the tables.

The tasks involved in Step 2 are:

- Step 2.1 Create tables
- Step 2.2 Check table structures using normalization
- Step 2.3 Check tables support user transactions
- Step 2.4 Check business rules
- Step 2.5 Review logical database design with users

10.2 Describe the rules for creating tables that represent:

(a) strong and weak entities;

(b) one-to-many (1:*) binary relationships;

(c) one-to-many (1:*) recursive relationships;

(d) one-to-one (1:1) binary relationships;

(e) one-to-one (1:1) recursive relationships;

(f) many-to-many (*:*) binary relationships;

(g) complex relationships;

(h) multi-valued attributes.

Give examples to illustrate your answers.

Table 10.1 Summary of how to represent entities, relationships, and multi-valued attributes as tables.

Entity/Relationship/Attribute	Representation as table(s)
Strong or weak entity	Create table that includes all simple attributes.
1:* binary relationship	Post copy of primary key of entity on 'one' side to table representing entity on 'many' side. Any attributes of relationship are also posted to 'many' side.
1:* recursive relationship	As entity on 'one' and 'many' side is the same, the table representing the entity receives a second copy of the primary key, which is renamed, and also any attributes of the relationship.
1:1 binary relationship:	
Mandatory participation on *both* sides	Combine entities into one table.
Mandatory participation on *one* side	Post copy of primary key of entity with optional participation to table representing entity with mandatory participation. Any attributes of relationship are also posted to table representing entity with mandatory participation.
Optional participation on *both* sides	Without further information, post copy of primary key of one entity to the other. However, if information is available, treat entity that is closer to having mandatory participation as being the child entity.
: binary relationship/ complex relationship	Create a table to represent the relationship and include any attributes associated with the relationship. Post a copy of the primary key from each parent entity into the new table to act as foreign keys.
Multi-valued attribute	Create a table to represent the multi-valued attribute and post a copy of the primary key of the parent entity into the new table to act as a foreign key.

Examples are provided throughout the description of Step 2.1 in Chapter 10.

10.3 Discuss how the technique of normalization can be used to check the structure of the tables created from the ER model and supporting documentation.

The purpose of the technique of normalization to examine the groupings of columns in each table created in Step 2.1. You check the composition of each table using the rules of normalization, to avoid unnecessary duplication of data.

You should ensure that each table created is in at least third normal form (3NF). If you identify tables that are not in 3NF, this may indicate that part of the ER model is incorrect, or that you have introduced an error while creating the tables from the model. If necessary, you may need to restructure the data model and/or tables.

10.4 Discuss one approach that can be used to check that the tables support the transactions required by the users.

One approach to checking that the tables support a transaction is to examine the transaction's data requirements to ensure that the data is present in one or more tables. Also, if a

transaction requires data in more than one table you should check that these tables are linked through the primary key/foreign key mechanism.

10.5 Discuss what business rules represent. Give examples to illustrate your answers.

Business rules are the constraints that you wish to impose in order to protect the database from becoming incomplete, inaccurate, or inconsistent. Although you may not be able to implement some business rules within the DBMS, this is not the question here. At this stage, you are concerned only with high-level design that is, specifying *what* business rules are required irrespective of *how* this might be achieved. Having identified the business rules, you will have a logical data model that is a complete and accurate representation of the organization (or part of the organization) to be supported by the database. If necessary, you could produce a physical database design from the logical data model, for example, to prototype the system for the user.

We consider the following types of business rules:

- required data,
- column domain constraints,
- entity integrity,
- multiplicity,
- referential integrity,
- other business rules.

10.5 Describe the alternative strategies that can be applied if there is a child record referencing a parent record that we wish to delete.

If a record of the parent table is deleted, referential integrity is lost if there is a child record referencing the deleted parent record. In other words, referential integrity is lost if the deleted branch currently has one or more members of staff working at it. There are several strategies you can consider in this case:

- NO ACTION Prevent a deletion from the parent table if there are any referencing child records. In our example, 'You cannot delete a branch if there are currently members of staff working there'.
- CASCADE When the parent record is deleted, automatically delete any referencing child records. If any deleted child record also acts as a parent record in another relationship then the delete operation should be applied to the records in this child table, and so on in a cascading manner. In other words, deletions from the parent table cascade to the child table. In our

example, 'Deleting a branch automatically deletes all members of staff working there'. Clearly, in this situation, this strategy would not be wise.

- SET NULL When a parent record is deleted, the foreign key values in all related child records are automatically set to null. In our example, 'If a branch is deleted, indicate that the current branch for those members of staff previously working there is unknown'. You can only consider this strategy if the columns comprising the foreign key can accept nulls, as defined in Step 1.3.
- SET DEFAULT When a parent record is deleted, the foreign key values in all related child records are automatically set to their default values. In our example, 'If a branch is deleted, indicate that the current assignment of members of staff previously working there is being assigned to another (default) branch'. You can only consider this strategy if the columns comprising the foreign key have default values, as defined in Step 1.3.
- NO CHECK When a parent record is deleted, do nothing to ensure that referential integrity is maintained. This strategy should only be considered in extreme circumstances.

10.6 Discuss what business rules represent. Give examples to illustrate your answers.

Finally, you consider constraints known as business rules. Business rules should be represented as constraints on the database to ensure that only permitted updates to tables governed by 'real world' transactions are allowed. For example, *StayHome* has a business rule that prevents a member from renting more than 10 videos at any one time.

Chapter 11 Enhanced Entity-Relationship Modeling – Review questions

11.1 Describe what a superclass and a subclass represent.

Superclass is an entity that includes one or more distinct groupings of its occurrences, which require to be represented in a data model. **Subclass** is a distinct grouping of occurrences of an entity, which require to be represented in a data model.

11.2 Describe the relationship between a superclass and its subclass.

The relationship between a superclass and any one of its subclasses is one-to-one (1:1) and is called a superclass/subclass relationship. For example, Staff/Manager forms a superclass/subclass relationship. Each member of a subclass is also a member of the superclass but has a distinct role.

11.3 Describe and illustrate using an example the process of attribute inheritance.

An entity occurrence in a subclass represents the same 'real world' object as in the superclass. Hence, a member of a subclass inherits those attributes associated with the superclass, but may also have subclass-specific attributes. For example, a member of the SalesPersonnel subclass has subclass-specific attributes, salesArea, vehLicenseNo, and carAllowance, and all the attributes of the Staff superclass, namely staffNo, name, position, salary, and branchNo.

11.4 What are the main reasons for introducing the concepts of superclasses and subclasses into an EER model?

There are two important reasons for introducing the concepts of superclasses and subclasses into an ER model. The first reason is that it avoids describing similar concepts more than once, thereby saving you time and making the ER model more readable. The second reason is that it adds more semantic information to the design in a form that is familiar to many people. For example, the assertions that 'Manager IS-A member of staff' and 'van IS-A type of vehicle' communicate significant semantic content in an easy-to-follow form.

11.5 Describe what a shared subclass represents.

A subclass is an entity in its own right and so it may also have one or more subclasses. A subclass with more than one superclass is called a shared subclass. In other words, a member of a shared subclass must be a member of the associated superclasses. As a consequence, the attributes of the superclasses are inherited by the shared subclass, which may also have its own additional attributes. This process is referred to as multiple inheritance.

11.6 Describe and contrast the process of specialization with the process of generalization.

Specialization is the process of maximizing the differences between members of an entity by identifying their distinguishing characteristics. Specialization is a top-down approach to defining a set of superclasses and their related subclasses. The set of subclasses is defined on the basis of some distinguishing characteristics of the entities in the superclass. When we identify a subclass of an entity, we then associate attributes specific to the subclass (where necessary), and also identify any relationships between the subclass and other entities or subclasses (where necessary).

Generalization is the process of minimizing the differences between entities by identifying their common features. The process of generalization is a bottom-up approach, which results in the identification of a generalized superclass from the original subclasses. The process of generalization can be viewed as the reverse of the specialization process.

11.7 Describe the two main constraints that apply to a specialization/generalization relationship.

There are two constraints that may apply to a superclass/subclass relationship called participation constraints and disjoint constraints.

Participation constraint determines whether every occurrence in the superclass must participate as a member of a subclass. A participation constraint may be mandatory or optional. A superclass/subclass relationship with a *mandatory participation* specifies that every entity occurrence in the superclass must also be a member of a subclass. A superclass/subclass

relationship with *optional participation* specifies that a member of a superclass need not belong to any of its subclasses.

Disjoint constraint describes the relationship between members of the subclasses and indicates whether it's possible for a member of a superclass to be a member of one, or more than one, subclass. The disjoint constraint only applies when a superclass has more than one subclass. If the subclasses are *disjoint*, then an entity occurrence can be a member of only one of the subclasses. To represent a disjoint superclass/subclass relationship, an 'Or' is placed next to the participation constraint within the curly brackets. If subclasses of a specialization/generalization are not disjoint (called nondisjoint), then an entity occurrence may be a member of more than one subclass. The participation and disjoint constraints of specialization/generalization are distinct giving the following four categories: mandatory and nondisjoint, optional and nondisjoint, mandatory and disjoint, and optional and disjoint.

Chapter 12 Physical Database Design – Step 3 – Review questions

12.1 Explain the difference between logical and physical database design. Why might these tasks be carried out by different people?

Logical database design is independent of implementation details, such as the specific functionality of the target DBMS, application programs, programming languages, or any other physical considerations. The output of this process is a logical data model that includes a set of relational tables together with supporting documentation, such as a data dictionary. These represent the sources of information for the physical design process, and they provide you with a vehicle for making trade-offs that are so important to an efficient database design.

Whereas logical database design is concerned with the *what*, physical database design is concerned with the *how*. In particular, the physical database designer must know how the computer system hosting the DBMS operates, and must also be fully aware of the functionality of the target DBMS. As the functionality provided by current systems varies widely, physical design must be tailored to a specific DBMS system. However, physical database design is not an isolated activity – there is often feedback between physical, logical, and application design. For example, decisions taken during physical design to improve performance, such as merging tables together, might affect the logical data model.

12.2 Describe the inputs and outputs of physical database design.

The inputs are the logical data model and the data dictionary. The outputs are the base tables, integrity rules, file organization specified, secondary indexes determined, user views and security mechanisms.

12.3 Describe the purpose of the main steps in the physical design methodology presented in this chapter.

Step 3 produces a relational database schema from the logical data model, which defines the base tables, integrity rules, and how to represent derived data.

12.4 Describe the types of information required to design the base tables.

You will need to know:

- how to create base tables;
- whether the system supports the definition of primary keys, foreign keys, and alternate keys;
- whether the system supports the definition of required data (that is, whether the system allows columns to be defined as NOT NULL);
- whether the system supports the definition of domains;
- whether the system supports relational integrity rules;
- whether the system supports the definition of business rules.

12.5 Describe how you would you handle the representation of derived data in the database. Give an example to illustrate your answer.

From a physical database design perspective, whether a derived column is stored in the database or calculated every time it's needed is a trade-off. To decide, you should calculate:

- the additional cost to store the derived data and keep it consistent with the data from which it is derived, and
- the cost to calculate it each time it's required,

and choose the less expensive option subject to performance constraints.

Chapter 13 Physical Database Design – Step 4 – Review questions

13.1 Describe the purpose of Step 4 in the database design methodology.

Step 4 determines the file organizations for the base tables. This takes account of the nature of the transactions to be carried out, which also determine where secondary indexes will be of use.

13.2 Discuss the purpose of analyzing the transactions that have to be supported and describe the type of information you would collect and analyze.

You can't make meaningful physical design decisions until you understand in detail the transactions that have to be supported. In analyzing the transactions, you're attempting to identify performance criteria, such as:

- the transactions that run frequently and will have a significant impact on performance;
- the transactions that are critical to the operation of the business;
- the times of the day/week when there will be a high demand made on the database (called the *peak load*).

You'll use this information to identify the parts of the database that may cause performance problems. At the same time, you need to identify the high-level functionality of the transactions, such as the columns that are updated in an update transaction or the columns that are retrieved in a query. You'll use this information to select appropriate file organizations and indexes.

13.3 When would you not add any indexes to a table?

(1) Do not index small tables. It may be more efficient to search the table in memory than to store an additional index structure.
(2) Avoid indexing a column or table that is frequently updated.
(3) Avoid indexing a column if the query will retrieve a significant proportion (for example, 25%) of the records in the table, even if the table is large. In this case, it may be more efficient to search the entire table than to search using an index.
(4) Avoid indexing columns that consist of long character strings.

13.4 Discuss some of the main reasons for selecting a column as a potential candidate for indexing. Give examples to illustrate your answer.

(1) In general, index the primary key of a table if it's not a key of the file organization. Although the SQL standard provides a clause for the specification of primary keys as discussed in Step 3.1 covered in the last chapter, note that this does not guarantee that the primary key will be indexed in some RDBMSs.

(2) Add a secondary index to any column that is heavily used for data retrieval. For example, add a secondary index to the Member table based on the column lName, as discussed above.

(3) Add a secondary index to a foreign key if there is frequent access based on it. For example, you may frequently join the VideoForRent and Branch tables on the column branchNo (the branch number). Therefore, it may be more efficient to add a secondary index to the VideoForRent table based on branchNo.

(4) Add a secondary index on columns that are frequently involved in:

(a) selection or join criteria;

(b) ORDER BY;

(c) GROUP BY;

(d) other operations involving sorting (such as UNION or DISTINCT).

(5) Add a secondary index on columns involved in built-in functions, along with any columns used to aggregate the built-in functions. For example, to find the average staff salary at each branch, you could use the following SQL query:

```
SELECT branchNo, AVG(salary)
FROM Staff
GROUP BY branchNo;
```

From the previous guideline, you could consider adding an index to the branchNo column by virtue of the GROUP BY clause. However, it may be more efficient to consider an index on both the branchNo column and the salary column. This may allow the DBMS to perform the entire query from data in the index alone, without having to access the data file. This is sometimes called an *index-only plan*, as the required response can be produced using only data in the index.

(6) As a more general case of the previous guideline, add a secondary index on columns that could result in an index-only plan.

13.5 Having identified a column as a potential candidate, under what circumstances would you decide against indexing it?

Having drawn up your 'wish-list' of potential indexes, consider the impact of each of these on update transactions. If the maintenance of the index is likely to slow down important update transactions, then consider dropping the index from the list.

Chapter 14 Physical Database Design – Steps 5 and 6 – Review questions

14.1 Describe the purpose of the main steps in the physical design methodology presented in this chapter.

Step 5 designs the user views for the database implementation. Step 6 designs the security mechanisms for the database implementation. This includes designing the access rules on the base relations.

14.2 Discuss the difference between system security and data security.

System security covers access and use of the database at the system level, such as a username and password. **Data security** covers access and use of database objects (such as tables and views) and the actions that users can have on the objects.

14.3 Describe the access control facilities of SQL.

Each database user is assigned an **authorization identifier** by the Database Administrator (DBA); usually, the identifier has an associated password, for obvious security reasons. Every SQL statement that is executed by the DBMS is performed on behalf of a specific user. The authorization identifier is used to determine which database objects that user may reference, and what operations may be performed on those objects. Each object that is created in SQL has an owner, who is identified by the authorization identifier. By default, the owner is the only person who may know of the existence of the object and perform any operations on the object.

Privileges are the actions that a user is permitted to carry out on a given base table or view. For example, SELECT is the privilege to retrieve data from a table and UPDATE is the privilege to modify records of a table. When a user creates a table using the SQL CREATE TABLE statement, he or she automatically becomes the owner of the table and receives full privileges for the table. Other users initially have no privileges on the newly created table. To give them access to the table, the owner must explicitly grant them the necessary privileges using the SQL GRANT statement. A WITH GRANT OPTION clause can be specified with the GRANT statement to allow the receiving user(s) to pass the privilege(s) on to other users. Privileges can be revoked using the SQL REVOKE statement.

When a user creates a view with the CREATE VIEW statement, he or she automatically becomes the owner of the view, but does not necessarily receive full privileges on the view. To create the view, a user must have SELECT privilege to all the tables that make up the view. However, the owner will only get other privileges if he or she holds those privileges for every table in the view.

14.3 Describe the security features of Microsoft Access 2002.

Access provides a number of security features including the following two methods:

(a) setting a password for opening a database (system security);
(b) user-level security, which can be used to limit the parts of the database that a user can read or update (data security).

In addition to the above two methods of securing a Microsoft Access database, other security features include:

- *Encryption/decryption:* encrypting a database compacts a database file and makes it indecipherable by a utility program or word processor. This is useful if you wish to transmit a database electronically or when you store it on a floppy disk or compact disc. Decrypting a database reverses the encryption.
- *Preventing users from replicating a database, setting passwords, or setting startup options;*
- *Securing VBA code:* this can be achieved by setting a password that you enter once per session or by saving the database as an MDE file, which compiles the VBA source code before removing it from the database. Saving the database as an MDE file also prevents users from modifying forms and reports without requiring them to specify a log on password or without you having to set up user-level security.

Chapter 15 Physical Database Design – Step 7 – Review questions

15.1 Describe the purpose of Step 7 in the database design methodology.

Step 8 considers relaxing the normalization constraints imposed on the logical data model to improve the overall performance of the system.

15.2 Explain the meaning of denormalization.

Formally, the term **denormalization** refers to a change to the structure of a base table, such that the new table is in a lower normal form than the original table. However, we also use the term more loosely to refer to situations where we combine two tables into one new table, where the new table is in the same normal form but contains more nulls than the original tables.

15.3 Discuss when it may be appropriate to denormalize a table. Give examples to illustrate your answer.

There are no fixed rules for determining when to denormalize tables. Some of the more common situations for considering denormalization to speed up frequent or critical transactions are:

- Step 7.2.1 Combining one-to-one (1:1) relationships
- Step 7.2.2 Duplicating nonkey columns in one-to-many (1:*) relationships to reduce joins
- Step 7.2.3 Duplicating foreign key columns in one-to-many (1:*) relationships to reduce joins
- Step 7.2.4 Duplicating columns in many-to-many (*:*) relationships to reduce joins
- Step 7.2.5 Introducing repeating groups
- Step 7.2.6 Creating extract tables
- Step 7.2.7 Partitioning tables

15.4 Describe the two main approaches to partitioning and discuss when each may be an appropriate way to improve performance. Give examples to illustrate your answer.

Horizontal partitioning Distributing the **records** of a table across a number of (smaller) tables.
Vertical partitioning Distributing the **columns** of a table across a number of (smaller) tables (the primary key is duplicated to allow the original table to be reconstructed).

Partitions are particularly useful in applications that store and analyze large amounts of data. For example, let's suppose there are hundreds of thousands of records in the VideoForRent

table that are held indefinitely for analysis purposes. Searching for a particular record at a branch could be quite time consuming, however, we could reduce this time by horizontally partitioning the table, with one partition for each branch.

There may also be circumstances where we frequently examine particular columns of a very large table and it may be appropriate to vertically partition the table into those columns that are frequently accessed together and another vertical partition for the remaining columns (with the primary key replicated in each partition to allow the original table to be reconstructed).

Chapter 16 Physical Database Design – Step 8 – Review questions

16.1 Describe the purpose of the main steps in the physical design methodology presented in this chapter.

Step 9 monitors the database application systems and improves performance by making amendments to the design as appropriate.

16.2 What factors can be used to measure efficiency?

There are a number of factors that we may use to measure efficiency:

- *Transaction throughput:* this is the number of transactions processed in a given time interval. In some systems, such as airline reservations, high transaction throughput is critical to the overall success of the system.
- *Response time:* this is the elapsed time for the completion of a single transaction. From a user's point of view, you want to minimize response time as much as possible. However, there are some factors that influence response time that you may have no control over, such as system loading or communication times. You can shorten response time by:
  - reducing contention and wait times, particularly disk I/O wait times;
  - reducing the amount of time resources are required;
  - using faster components.
- *Disk storage:* this is the amount of disk space required to store the database files. You may wish to minimize the amount of disk storage used.

16.3 Discuss how the four basic hardware components interact and affect system performance.

- main memory
- CPU
- disk I/O
- network.

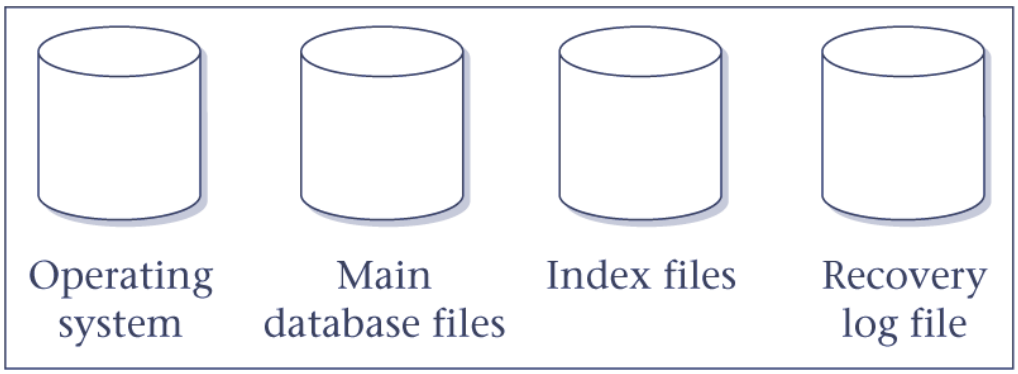
Each of these resources may affect other system resources. Equally well, an improvement in one resource may effect an improvement in other system resources. For example:

- Adding more main memory should result in less paging. This should help avoid CPU bottlenecks.
- More effective use of main memory may result in less disk I/O.

16.4 How should you distribute data across disks?

Figure 16.1 illustrates the basic principles of distributing the data across disks:

- The operating system files should be separated from the database files.
- The main database files should be separated from the index files.
- The recovery log file, if available and if used, should be separated from the rest of the database.



16.5 What is RAID technology and how does it improve performance and reliability?

RAID originally stood for *Redundant Array of Inexpensive Disks*, but more recently the 'I' in RAID has come to stand for *Independent*. RAID works on having a large disk array comprising an arrangement of several independent disks that are organized to increase performance and at the same time improve reliability.

Performance is increased through *data striping*: the data is segmented into equal-size partitions (the *striping unit*), which are transparently distributed across multiple disks. This gives the appearance of a single large, very fast disk where in actual fact the data is distributed across several smaller disks. Striping improves overall I/O performance by allowing multiple I/Os to be serviced in parallel. At the same time, data striping also balances the load among disks. Reliability is improved through storing redundant information across the disks

using a *parity* scheme or an *error-correcting* scheme. In the event of a disk failure, the redundant information can be used to reconstruct the contents of the failed disk.

Chapter 19 Current and Emerging Trends – Review questions

19.1 Discuss the general characteristics of advanced database applications.

- Design data is characterized by a large number of types, each with a small number of instances. Conventional databases are typically the opposite.
- Designs may be very large, perhaps consisting of millions of parts, often with many interdependent subsystem designs.
- The design is not static but evolves through time. When a design change occurs, its implications must be propagated through all design representations. The dynamic nature of design may mean that some actions cannot be foreseen at the beginning.
- Updates are far-reaching because of topological or functional relationships, tolerances, and so on. One change is likely to affect a large number of design objects.
- Often, many design alternatives are being considered for each component, and the correct version for each part must be maintained. This involves some form of version control and configuration management.
- There may be hundreds of staff involved with the design, and they may work in parallel on multiple versions of a large design. Even so, the end product must be consistent and coordinated. This is sometimes referred to as *cooperative engineering*.

19.2 Discuss why the weaknesses of the relational data model and relational DBMSs may make them unsuitable for advanced database applications.

Poor representation of 'real world' entities

Normalization generally leads to the creation of tables that do not correspond to entities in the 'real world'. The fragmentation of a 'real world' entity into many tables, with a physical representation that reflects this structure, is inefficient leading to many joins during query processing.

Semantic overloading

The relational model has only one construct for representing data and relationships between data, namely the *table*. For example, to represent a many-to-many (*:*) relationship between two entities A and B, we create three tables, one to represent each of the entities A and B, and one to represent the relationship. There is no mechanism to distinguish between entities and relationships, or to distinguish between different kinds of relationship that exist between entities. For example, a 1:* relationship might be *Has*, *Supervises*, *Manages*, and so on. If such distinctions could be made, then it might be possible to build the semantics into the operations. It is said that the relational model is **semantically overloaded**.

Poor support for business rules

In Section 2.3, we introduced the concepts of entity and referential integrity, and in Section 2.2.1 we introduced domains, which are also types of business rules. Unfortunately, many commercial systems do not fully support these rules, and it's necessary to build them into the applications. This, of course, is dangerous and can lead to duplication of effort and, worse still, inconsistencies. Furthermore, there is no support for other types of business rules in the relational model, which again means they have to be built into the DBMS or the application.

Limited operations

The relational model has only a fixed set of operations, such as set and record-oriented operations, operations that are provided in the SQL specification. However, SQL currently does not allow new operations to be specified. Again, this is too restrictive to model the behavior of many 'real world' objects. For example, a GIS application typically uses points, lines, line groups, and polygons, and needs operations for distance, intersection, and containment.

Difficulty handling recursive queries

Atomicity of data means that repeating groups are not allowed in the relational model. As a result, it's extremely difficult to handle recursive queries: that is, queries about relationships that a table has with itself (directly or indirectly). To overcome this problem, SQL can be embedded in a high-level programming language, which provides constructs to facilitate iteration. Additionally, many RDBMSs provide a report writer with similar constructs. In either case, it is the application rather than the inherent capabilities of the system that provides the required functionality.

Impedance mismatch

In Section 3.1.1, we noted that until the most recent version of the standard SQL lacked *computational completeness*. To overcome this problem and to provide additional flexibility, the SQL standard provides embedded SQL to help develop more complex database applications. However, this approach produces an **impedance mismatch** because we are mixing different programming paradigms:

(1) SQL is a declarative language that handles rows of data, whereas a high-level language such as 'C' is a procedural language that can handle only one row of data at a time.

(2) SQL and 3GLs use different models to represent data. For example, SQL provides the built-in data types Date and Interval, which are not available in traditional programming languages. Thus, it's necessary for the application program to convert between the two representations, which is inefficient, both in programming effort and in the use of runtime resources. Furthermore, since we are using two different type systems, it's not possible to automatically type check the application as a whole.

The latest release of the SQL standard, SQL3, addresses some of the above deficiencies with the introduction of many new features, such as the ability to define new data types and operations as part of the data definition language, and the addition of new constructs to make the language computationally complete.

19.3 Explain what is meant by a DDBMS, and discuss the motivation in providing such a system.

A **Distributed Database Management System** (**DDBMS**) consists of a single logical database that is split into a number of **fragments**. Each fragment is stored on one or more computers (**replicas**) under the control of a separate DBMS, with the computers connected by a communications network. Each site is capable of independently processing user requests that require access to local data (that is, each site has some degree of local autonomy) and is also capable of processing data stored on other computers in the network.

19.4 Compare and contrast a DDBMS with distributed processing. Under what circumstances would you choose a DDBMS over distributed processing?

Distributed processing: a centralized database that can be accessed over a computer network.

The key point with the definition of a distributed DBMS is that the system consists of data that is physically distributed across a number of sites in the network. If the data is centralized, even though other users may be accessing the data over the network, we do not consider this to be a distributed DBMS, simply distributed processing.

19.5 Discuss the advantages and disadvantages of a DDBMS.

Advantages

Reflects organizational structure Many organizations are naturally distributed over several locations. It's natural for databases used in such an application to be distributed over these locations.

Improved shareability and local autonomy The geographical distribution of an organization can be reflected in the distribution of the data; users at one site can access data stored at other sites. Data can be placed at the site close to the users who normally use that data. In this way, users have local control of the data, and they can consequently establish and enforce local policies regarding the use of this data.

Improved availability In a centralized DBMS, a computer failure terminates the operations of the DBMS. However, a failure at one site of a DDBMS, or a failure of a communication link making some sites inaccessible, does not make the entire system inoperable.

Improved reliability As data may be replicated so that it exists at more than one site, the failure of a node or a communication link does not necessarily make the data inaccessible.

Improved performance As the data is located near the site of 'greatest demand', and given the inherent parallelism of DDBMSs, it may be possible to improve the speed of database accesses than if we had a remote centralized database. Furthermore, since each site handles only a part of the entire database, there may not be the same contention for CPU and I/O services as characterized by a centralized DBMS.

Economics It's generally accepted that it costs much less to create a system of smaller computers with the equivalent power of a single large computer. This makes it more cost-effective for corporate divisions and departments to obtain separate computers. It's also much more cost-effective to add workstations to a network than to update a mainframe system.

Modular growth In a distributed environment, it's much easier to handle expansion. New sites can be added to the network without affecting the operations of other sites. This flexibility allows an organization to expand relatively easily.

Disadvantages

Complexity A DDBMS that hides the distributed nature from the user and provides an acceptable level of performance, reliability, and availability is inherently more complex than a centralized DBMS. Replication also adds an extra level of complexity, which if not handled adequately, will lead to degradation in availability, reliability, and performance compared with the centralized system, and the advantages we cited above will become disadvantages.

Cost Increased complexity means that we can expect the procurement and maintenance costs for a DDBMS to be higher than those for a centralized DBMS. Furthermore, a DDBMS requires additional hardware to establish a network between sites. There are ongoing communication costs incurred with the use of this network. There are also additional manpower costs to manage and maintain the local DBMSs and the underlying network.

Security In a centralized system, access to the data can be easily controlled. However, in a DDBMS not only does access to replicated data have to be controlled in multiple locations, but the network itself has to be made secure. In the past, networks were regarded as an insecure communication medium. Although this is still partially true, significant developments have been made recently to make networks more secure.

Integrity control more difficult Enforcing integrity constraints generally requires access to a large amount of data that defines the constraint, but is not involved in the actual update operation itself. In a DDBMS, the communication and processing costs that are required to enforce integrity constraints may be prohibitive.

Lack of standards Although DDBMSs depend on effective communication, we are only now starting to see the appearance of standard communication and data access protocols. This lack of standards has significantly limited the potential of DDBMSs. There are also no tools or methodologies to help users convert a centralized DBMS into a distributed DBMS.

Lack of experience General-purpose DDBMSs have not been widely accepted, although many of the protocols and problems are well understood. Consequently, we do not yet have the same level of experience in industry as we have with centralized DBMSs. For a prospective adopter of this technology, this may be a significant deterrent.

Database design more complex Besides the normal difficulties of designing a centralized database, the design of a distributed database has to take account of fragmentation of data, allocation of fragments to specific sites, and data replication.

19.6 Describe the expected functionality of a replication server.

At its basic level, we expect a distributed data replication service to be capable of copying data from one database to another, synchronously or asynchronously. However, there are many other functions that need to be provided, such as:

- *Specification of replication schema* The system should provide a mechanism to allow a privileged user to specify the data and objects to be replicated.
- *Subscription mechanism* The system should provide a mechanism to allow a privileged user to subscribe to the data and objects available for replication.
- *Initialization mechanism* The system should provide a mechanism to allow for the initialization of a target replica.
- *Scalability* The service should be able to handle the replication of both small and large volumes of data.
- *Mapping and transformation* The service should be able to handle replication across different DBMSs and platforms. This may involve mapping and transforming the data from one data model into a different data model, or the data in one data type to a corresponding data type in another DBMS.
- *Object replication* It should be possible to replicate objects other than data. For example, some systems allow indexes and stored procedures (or triggers) to be replicated.
- *Easy administration* It should be easy for the DBA to administer the system and to check the status and monitor the performance of the replication system components.

19.7 Compare and contrast the different ownership models for replication. Give examples to illustrate your answer.

Ownership relates to which site has the privilege to update the data. The main types of ownership are **master/slave**, **workflow**, and **update-anywhere** (sometimes referred to as *peer-to-peer* or *symmetric replication*).

Master/slave ownership

With master/slave ownership, asynchronously replicated data is owned by one site, the *master* or *primary* site, and can be updated by only that site. Using a '*publish-and-subscribe*' metaphor, the master site (the publisher) makes data available. Other sites 'subscribe' to the data owned by the master site, which means that they receive read-only copies on their local systems. Potentially, each site can be the master site for non-overlapping data sets. However, there can only ever be one site that can update the master copy of a particular data set, and so update conflicts cannot occur between sites.

A master site may own the data in an entire table, in which case other sites subscribe to read-only copies of that table. Alternatively, multiple sites may own distinct fragments of the table, and other sites then subscribe to read-only copies of the fragments. This type of replication is also known as **asymmetric replication**.

Workflow ownership

Like master/slave ownership, this model avoids update conflicts while at the same time providing a more dynamic ownership model. Workflow ownership allows the right to update replicated data to move from site to site. However, at any one moment, there is only ever one site that may update that particular data set. A typical example of workflow ownership is an order processing system, where the processing of orders follows a series of steps, such as order entry, credit approval, invoicing, shipping, and so on. In a centralized DBMS, applications of this nature access and update the data in one integrated database: each application updates the order data in sequence when, and only when, the state of the order indicates that the previous step has been completed.

Update-anywhere (symmetric replication) ownership

The two previous models share a common property: at any given moment, only one site may update the data; all other sites have read-only access to the replicas. In some environments, this is too restrictive. The update-anywhere model creates a peer-to-peer environment where

multiple sites have equal rights to update replicated data. This allows local sites to function autonomously, even when other sites are not available.

Shared ownership can lead to conflict scenarios and the replication architecture has to be able to employ a methodology for conflict detection and resolution. A simple mechanism to detect conflict within a single table is for the source site to send both the old and new values (*before-* and *after-images*) for any records that have been updated since the last refresh. At the target site, the replication server can check each record in the target database that has also been updated against these values. However, consideration has to be given to detecting other types of conflict such as violation of referential integrity between two tables. There have been many mechanisms proposed for conflict resolution, but some of the most common are: earliest/latest timestamps, site priority, and holding for manual resolution.

19.8 Give a definition of an OODBMS. What are the advantages and disadvantages of an OODBMS.

OODM A (logical) data model that captures the semantics of objects supported in object-oriented programming.

OODB A persistent and sharable collection of objects defined by an OODM.

OODBMS The manager of an OODB.

19.9 Give a definition of an ORDBMS. What are the advantages and disadvantages of an ORDBMS.

Thus, there is no single extended relational model; rather, there are a variety of these models, whose characteristics depend upon the way and the degree to which extensions were made. However, all the models do share the same basic relational tables and query language, all incorporate some concept of 'object', and some have the ability to store methods (or procedures or triggers) as well as data in the database.

19.10 Give a definition of a data warehouse. Discuss the benefits of implementing a data warehouse.

Data warehouse: a consolidated/integrated view of corporate data drawn from disparate operational data sources and a range of end-user access tools capable of supporting simple to highly complex queries to support decision-making.

Benefits:

Potential high return on investment

Competitive advantage

Increased productivity of corporate decision-makers

19.11 Describe the characteristics of the data held in a data warehouse.

The data held in a data warehouse is described as being subject-oriented, integrated, time-variant, and non-volatile (Inmon, 1993).

- *Subject-oriented* as the warehouse is organized around the major subjects of the organization (such as customers, products, and sales) rather than the major application areas (such as customer invoicing, stock control, and product sales). This is reflected in the need to store decision-support data rather than application-oriented data.
- *Integrated* because of the coming together of source data from different organization-wide applications systems. The source data is often inconsistent using for example, different data types and/or formats. The integrated data source must be made consistent to present a unified view of the data to the users.
- *Time-variant* because data in the warehouse is only accurate and valid at some point in time or over some time interval. The time-variance of the data warehouse is also shown in the extended time that the data is held, the implicit or explicit association of time with all data, and the fact that the data represents a series of snapshots.
- *Non-volatile* as the data is not updated in real-time but is refreshed from operational systems on a regular basis. New data is always added as a supplement to the database, rather than a replacement. The database continually absorbs this new data, incrementally integrating it with the previous data.

19.12 Discuss how data marts differ from data warehouses and identify the main reasons for implementing a data mart.

A data mart holds a subset of the data in a data warehouse normally in the form of summary data relating to a particular department or business area such as Marketing or Customer Services. The data mart can be stand-alone or linked centrally to the corporate data warehouse. As a data warehouse grows larger, the ability to serve the various needs of the organization may

be compromised. The popularity of data marts stems from the fact that corporate data warehouses proved difficult to build and use.

19.13 Discuss what online analytical processing (OLAP) is and how OLAP differs from data warehousing.

Online analytical processing (OLAP): The dynamic synthesis, analysis, and consolidation of large volumes of multi-dimensional data. The key characteristics of OLAP applications include multi-dimensional views of data, support for complex calculations, and time intelligence.

19.14 Describe OLAP applications and identify the characteristics of such applications.

Table 19.1 Examples of OLAP applications in various business areas.

Business area	Examples of OLAP applications
Finance	Budgeting, activity-based costing, financial performance analysis, and financial modeling.
Sales	Sales analysis and sales forecasting.
Marketing	Market research analysis, sales forecasting, promotions analysis, customer analysis, and market/customer segmentation.
Manufacturing	Production planning and defect analysis.

An essential requirement of all OLAP applications is the ability to provide users with just-in-time (JIT) information, which is necessary to make effective decisions about an organization's strategic directions.

19.15 Discuss how data mining can realize the value of a data warehouse.

Simply storing information in a data warehouse does not provide the benefits an organization is seeking. To realize the value of a data warehouse, it's necessary to extract the knowledge hidden within the warehouse. However, as the amount and complexity of the data in a data warehouse grows, it becomes increasingly difficult, if not impossible, for business analysts to identify trends and relationships in the data using simple query and reporting tools. Data mining is one of the best ways to extract meaningful trends and patterns from huge amounts of data. Data mining discovers information within data warehouses that queries and reports cannot effectively reveal.

19.16 Why would we want to dynamically generate web pages from data held in the operational database? List some general requirements for web-database integration.

An HTML/XML document stored in a file is an example of a static Web page: the content of the document does not change unless the file itself is changed. On the other hand, the content of a dynamic Web page is generated each time it's accessed. As a result, a dynamic Web page can have features that are not found in static pages, such as:

- It can respond to user input from the browser. For example, returning data requested by the completion of a form or the results of a database query.
- It can be customized by and for each user. For example, once a user has specified some preferences when accessing a particular site or page (such as area of interest or level of expertise), this information can be retained and information returned appropriate to these preferences.

Not in any ranked order, the requirements are as follows:

- The ability to access valuable corporate data in a secure manner.
- Data and vendor independent connectivity to allow freedom of choice in the selection of the DBMS now and in the future.
- The ability to interface to the database independent of any proprietary Web browser or Web server.
- A connectivity solution that takes advantage of all the features of an organization's DBMS.
- An open-architecture approach to allow interoperability with a variety of systems and technologies.
- A cost-effective solution that allows for scalability, growth, and changes in strategic directions, and helps reduce the costs of developing and maintaining applications.
- Support for transactions that span multiple HTTP requests.
- Support for session- and application-based authentication.
- Acceptable performance.
- Minimal administration overhead.

- A set of high-level productivity tools to allow applications to be developed, maintained, and deployed with relative ease and speed.

19.17 What is XML and discuss the approaches for managing XML-based data.

XML: a meta-language (a language for describing other languages) that enables designers to create their own customized tags to provide functionality not available with HTML.

It's anticipated that there will be two main models that will exist: data-centric and document-centric. In a **data-centric model**, XML is used as the storage and interchange format for data that is structured, appears in a regular order, and is most likely to be machine processed instead of read by a human. In a data-centric model, the fact that the data is stored and transferred as XML is incidental and other formats could also have been used. In this case, the data could be stored in a relational, object-relational, or object-oriented DBMS. For example, Oracle has completely integrated XML into its Oracle 9i system. XML can be stored as entire documents using the data types XMLType or CLOB/BLOB (Character/Binary Large Object) or can be decomposed into its constituent elements and stored that way. The Oracle query language has also been extended to permit searching of XML-based content.

In a **document-centric model**, the documents are designed for human consumption (for example, books, newspapers, and email). Due to the nature of this information, much of the data will be irregular or incomplete, and its structure may change rapidly or unpredictably. Unfortunately, relational, object-relational, and object-oriented DBMSs do not handle data of this nature particularly well. Content management systems are an important tool for handling these types of documents. Underlying such a system, you may now find a *native XML database*: